UNIVERSIDAD NACIONAL DE LA PLATA

FACULTAD DE INFORMÁTICA

TESINA DE LICENCIATURA

TÍTULO: Aplicación de tecnologías de aprendizaje automático para predecir negocios y tomar decisiones empresariales.

AUTOR: Ignacio Joakin

DIRECTOR ACADÉMICO: Dr. Claudia Pons

DIRECTOR PROFESIONAL:

CARRERA: Licenciatura en Sistemas

Resumen

En la actualidad las empresas poseen una gran cantidad de información, esta información se utiliza para la toma de decisiones según diferentes perspectivas, expectativas o criterios.
El manejo eficiente de información es clave para la toma de decisiones. En este trabajo se muestra como arquitecturas de software específicas, microservicios y herramientas de aprendizaje automático pueden integrarse a un sistema de gestión y ayudar en el manejo eficiente de la información agregando valor al momento de tomar decisiones.

Palabras Clave

Arquitectura de software, microservicios, Inteligencia Artificial, Machine Learning, data mining, redes neuronales, algoritmo, decisión, Intención de compra, Arquitectura de software

Conclusiones

Teniendo en cuenta los resultados obtenidos podemos afirmar que la utilización de técnicas de predicción utilizando algoritmos de Machine Learning para la toma de decisiones es efectiva y es una gran herramienta a la hora de tomar ciertas decisiones críticas para una empresa.

Trabajos Realizados

Se ha realizado un sistema integral de manejo de productos, se utilizó una arquitectura basada en servicios, en la cual se ha implementado dos microservicios utilizando las mejores prácticas actuales como IoC. Por otro lado, la solución integra un módulo de Machine learning basado en redes neuronales y series temporales la cual realiza una predicción de las intenciones de compra para el siguiente mes. El sistema recomienda los movimientos de stock

Trabajos Futuros

El modelo de Series Temporales con Múltiples Variables y Embeddings puede mejorarse.

Fecha de la presentación:

FACULTAD DE INFORMATICA
UNIVERSIDAD NACIONAL DE LA PLATA

Aplicación de tecnologías de aprendizaje automático para predecir negocios y tomar decisiones empresariales.

Tesina de Grado

Licenciatura en Sistemas – Plan 2007

Ignacio Joakin.

Director: Dra. Claudia Pons

Facultad de Informática

Fecha: 5 de septiembre de 2021

“Sometimes it is the people no one can imagine anything of who do the things no one can imagine.”

— Alan Turing

Índice

Índice de figuras

Introducción

En la actualidad las empresas poseen una gran cantidad de información, esta información se utiliza para la toma de decisiones según diferentes perspectivas, expectativas o criterios.
El manejo eficiente de información es clave para la toma de decisiones. En este trabajo se muestra como herramientas de aprendizaje automático (Machine Learning) pueden integrarse a un sistema de gestión y ayudar en el manejo eficiente de la información agregando valor al momento de tomar decisiones.

Para la aplicación se realizará una introducción a las arquitecturas de software y se plantea una arquitectura basada en microservicios, la cual se utilizará en este trabajo dando la oportunidad de poder balancear el trabajo del backend en servicios, por otro lado, se muestra la separación en capas no solo del backend sino del frontend con la utilización de diferentes técnicas como IoC.

Al presente, el área más exitosa de IA está relacionada con el aprendizaje automático [Bishop, 2008] [Flach, 2012] [Alpaydin, 2014], más conocido como Machine Learning (ML), basado en ANNs (redes neuronales). El ML como veremos en el capítulo 3 es el estudio científico de algoritmos y modelos estadísticos que los sistemas informáticos utilizan para realizar de manera efectiva una tarea específica sin utilizar instrucciones explícitas, sino que se basan en patrones e inferencias. Los algoritmos de ML construyen un modelo matemático a partir de datos de muestra (conocido como "datos de entrenamiento"), para hacer predicciones o tomar decisiones sin estar explícitamente programado para realizar la tarea.
Existen numerosos algoritmos disponibles para ML, que generan automáticamente modelos no-lineales sobre las relaciones entre los datos. Se pueden emplear para resolver un problema sin necesitad de programar manualmente ninguna lógica en el sistema inteligente.

El propósito de esta investigación es analizar e integrar herramientas de aprendizaje automático en negocios convencionales para la toma de decisiones, y tener un ejemplo de cómo estas herramientas se integran a una solución. Para concretar dicha integración diseñaremos e implementaremos una arquitectura de software basada en microservicios. **Específicamente se implementará un sistema de gestión de productos, y se utilizará un algoritmo de Deep learning para predecir la cantidad de ventas que una empresa realizará a futuro, estimando así cuál será el stock necesario para poder afrontar esas ventas futuras, el sistema**

recomendará compra o movimiento de stock entre los distintos puntos de ventas, dando la posibilidad al usuario final (en este caso al gestor de productos) de tomar la decisión de realizar el movimiento de stock.

Objetivos de esta tesis.

- Presentar todas las cuestiones principales asociadas al uso de Machine Learning que tienen que ver con:
    - Obtener y curar los datos necesarios para ser suministrados al algoritmo, lo cual no es una tarea trivial. La cantidad y calidad de los datos es de suma importancia,
    - Seleccionar los algoritmos de ML más apropiados al problema De acuerdo al Teorema de 'No Free Lunch' [Wolpert, 1997], no existe ningún algoritmo que puede ser aplicado para cualquier problema.
    - Determinar la configuración (o parametrización). Por ejemplo, especificar la cantidad de nodos de una red. Aunque existen diversos mecanismos que permiten definir automáticamente estas configuraciones, en ciertas implementaciones se considera importante incorporar modificaciones basadas en el conocimiento de expertos del dominio.
    - Mencionar potenciales líneas de investigación.
- Diseñar e implementar una arquitectura de software que facilite la integración de un modelo de negocios con los servicios de aprendizaje automático.
- Implementar un modelo de negocio general y aplicar distintas herramientas de aprendizaje automático, realizando distintas predicciones y análisis para la toma de decisiones.

Organización del trabajo de tesis.

El presente trabajo se organiza de la siguiente forma: el capítulo 1 se expone una introducción a la toma de decisiones, luego en el capítulo 2 se explicará las arquitecturas de software asociadas a la solución y se mencionan las tecnologías más significativas utilizadas en el desarrollo del sistema.

Luego, en los capítulos 3 y 4 como marco teórico se realiza una introducción a machine learning y Deep learning en los cuales se enmarca el módulo utilizado para la predicción de intenciones de compra.
En el capítulo 5, se realiza una explicación detallada de todos los módulos del sistema y se explica las partes relevantes de la solución, dando una explicación de su ejecución y utilización.

En el capítulo 6 se presentan las conclusiones y en el capítulo 7 las líneas de trabajo futuro.

Finalmente, esta tesina posee tres anexos, en el primero haremos un recorrido por la historia de la IA y su evolución hasta el machine learning y Deep learning, en el segundo describimos las redes neuronales y su importancia para la inteligencia artificial. En el tercero describimos el termino data mining, su diferencia con machine learning y su importancia para poder trabajar con datos grandes.

Capítulo 1 - Toma de decisiones

En este capítulo se desarrollará un breve resumen de lo que implica tomar una decisión y específicamente lo que se refiere con tomar una decisión dentro de una empresa; teniendo en cuenta los atributos que pueden incidir en estas. No es idea de este trabajo entrar en el terreno psicológico de la toma de decisiones, sino más bien especificar ciertos puntos que se deben tomarse en cuenta.

Empezaremos dando la definición de lo que es tomar una decisión, cuando una persona debe elegir entre distintas opciones pasa por un proceso que se lo denomina toma de decisiones. En una empresa o en la vida diaria una persona se puede encontrar con estas situaciones donde debe optar por algo, por ejemplo**,** si debe o no utilizar ML dentro de su negocio, como en el anterior ejemplo no siempre resulta simple. El proceso de la toma de decisiones hace hincapié en conflictos que se presentan y a los cuales hay que encontrarles solución.

Cómo ejemplo de lo que puede implicar una buena toma de decisiones se puede mencionar el caso de Amazon, cuando todas las empresas empezaron a realizar la **transformación digital**, algunas empresas de indumentaria decidieron no agregar una plataforma web para sus ventas ya que pensaban que las personas preferirían probarse su ropa antes de comprar. Hoy en día Amazon es una de las plataformas de venta de indumentaria más importante del mundo.

Otro buen ejemplo de malas decisiones empresariales es la de blockbuster, una empresa de alquiler de películas a domicilio. En el momento de mayor prosperidad de la empresa, Netflix le ofreció a Blockbuster agregar un componente online a su operación de alquiler de cintas y DVDs, a cambio de que la compañía de videoclubes dedicara un espacio en sus tiendas a Netflix (que por ese entonces también ofrecía DVDs de alquiler por correo), Blockbuster declinó éste y otra serie de potenciales acuerdos con Netflix, que se convirtió en su principal amenaza.

Gradualmente fue perdiendo mercado hasta que se declaró en quiebra en 2010 y fue comprada en remate por Dish Network (que terminó de cerrar los locales que quedaban): para entonces, era claro que el futuro del cine hogareño era vía internet y con el crecimiento del streaming de series y películas, hoy los videocasetes han quedado en el olvido, casi lo mismo que los DVDs.

En estos claros ejemplos podemos visualizar que jamás hay que tomar decisiones a la ligera dentro de una compañía, pues de estas decisiones puede depender la estabilidad de la compañía, su continuidad o su crecimiento.

Por ello, al iniciar el proceso de decisión debe tener en cuenta estas 10 claves para alejarse de determinaciones empresariales erróneas[1], de acuerdo con Mauricio Rodríguez, profesor de liderazgo en la Universidad Externado y de Los Andes.

1. Basarse en información incompleta, desactualizada o no confiable

Muchas veces el no poseer los números o los datos reales pueden alejar a persona de la situación precisa por las que la compañía este pasando en un momento determinado. Esto significa que tendrá en cuenta una realidad diferente al tomar la decisión. Este es uno de los puntos más importantes de este trabajo, el poder manejar de manera precisa y confiable la información siendo esta muchas veces de diferentes fuentes para la toma de decisiones es realmente muy importante.

2. Exceso de análisis que demora y confunde

El exceso de análisis muchas veces puede posponer el inicio de un proyecto llevando a confusiones constantes, realizar un correcto análisis y poder definir los límites del mismo puede favorecer al inicio correcto de un proyecto. Este problema también es llamado "parálisis" del análisis, en este caso utilizar las herramientas adecuadas en el momento adecuado puede que ayude a que no exista este exceso de análisis.

3. Falta de análisis

Como se mencionó anteriormente el exceso de análisis es nocivo, por lo tanto, también es su escasez. Esta situación hace tomar decisiones 'a la ligera', que pueden causar grandes daños a la compañía.

4. Falta de método

Lo métodos y procesos son importantes para el análisis de la información como así también las herramientas, estas herramientas y métodos deben tener en cuenta todos los posibles escenarios.

[1] https://www.larepublica.co/alta-gerencia/las-razones-de-malas-decisiones-empresariales-3021705

5. Decisiones emocionales

Las decisiones tomadas desde las emociones y sin tener en cuenta la razón o la información obtenida muchas veces llevan a decisiones erróneas.

6. Decisiones "políticas"

Otras veces las decisiones son tomadas desde un punto político, por ejemplo, para negociar acuerdos entre personas o teniendo en cuenta otros puntos que no tienen que ver con aspectos del proyecto, también puede llevar a decisiones erróneas.

7. Comunicación poco asertiva de las decisiones

En muchas ocasiones se toman decisiones buenas, pero que no se explican bien al resto de la organización. Esto hace que su efecto positivo se demore y/o enreda.

8. Exceso de decisiones

En una empresa se puede llegar a tomar muchas decisiones, hay veces que la cantidad es excesiva y puede que no se pueda hacer un análisis correcto de cada una.

9. Confusión entre los tipos de decisiones

Según algunos expertos en liderazgo, pueden llegar a confundirse las decisiones estratégicas (de mediano/largo plazo) con las tácticas (de un año) y las operativas (del día a día)".

10. Se toman las decisiones en los niveles equivocados

Por último, puede que se tome la decisión en el nivel equivocado de la estructura organizacional. Esto puede que las decisiones no puedan implementarse o que se hagan correctamente.

Por otro lado, sabemos que **no** todas las personas responden de la misma forma a una misma situación o problemática, esto es debido a distintos elementos como la estructura de personalidad, el desarrollo, madurez, a la etapa de la vida en la que se encuentre esa persona, estos son algunos de los ejemplos de los elementos que pueden inferir en la toma de una decisión.

Se han creado diversos modelos a partir de distintos enfoques teóricos, éstos sirven tanto para encontrar la explicación a la conducta en situaciones problemáticas, como para tener las bases

en la elaboración de técnicas terapéuticas para ayudar, a quienes lo precisan, a desarrollar y potenciar la toma de decisiones.

Componentes de la toma de decisiones

Resolver un problema necesita de los siguientes conceptos, ya que todos ellos son importantes no sólo para encontrar un resultado inicial, sino para el aprendizaje y mejoría de la resolución de problemas, favoreciendo ampliamente la detección de las propias herramientas (competencias). [2]

- **Decisión.** Todas las posibles combinaciones que incluyen tanto las acciones a llevar a cabo como las situaciones.
- **Resultado.** Hipotéticas situaciones que tendrían lugar si se toma una u otra opción de las decisiones antes señaladas.
- **Consecuencia.** Evaluación basada en la subjetividad, por ejemplo ganancias o pérdida.
- **Incertidumbre.** Aquí juegan un papel fundamental tanto la probabilidad, como la confianza y posibilidad, frente a lo desconocido, sobre todo cuando no se tiene experiencia en algún problema en particular.
- **Preferencias.** Tendencia a tomar una alternativa y no otra, se ve condicionada por la experiencia.
- **Toma de decisión.** Acción de decidir.
- **Juicio.** Evaluación.

En base a ello podemos exponer el siguiente modelo de resolución de problemas:

- **Definir el** problema**.** Requiere del análisis de la situación que se enfrenta.
- **Alternativas posibles.** Son todas las combinaciones de acciones que se pueden tomar.
- **Prever resultados.** Como hasta ahora son sólo hipótesis, se requiere asociar las posibles consecuencias de cada una de las alternativas.
- **Elegir.** Optar por alguna de ellas.
- **Control.** Es necesario siempre tener todo bajo control sin dejar nada al azar, siendo monitores, responsables y con actitud participativa en el proceso.
- **Evaluación.** Ver los pros y los contras de lo que se ha decidido, algo primordial para el aprendizaje.

[2] https://concepto.de/toma-de-decisiones/

¿Qué dificulta el proceso para tomar una decisión?

- **Disonancia cognitiva.** Cuando lo que se quiere hacer y lo que se acaba haciendo no son coincidentes.
- **Efecto Halo.** Ocurre cuando la sombra de otras experiencias hace que se deduzca erróneamente, presuponiendo y anticipando precipitadamente una decisión.
- **Pensamiento de** grupo**.** Ocurre cuando un grupo de personas decide por otras, a pesar de éstas estar en desacuerdo. Es decir, no hay consenso, sino que miedo, autoridad, temor a equivocarse, rechazo o cuestionamiento grupal.
- **Adaptación Hedonista.** Estado de bienestar y placer que no permite relacionarse adecuadamente con el conflicto.
- **Sesgo de confirmación.** Para poder realizar una correcta evaluación de los resultados, es necesario tener la suficiente flexibilidad cognitiva como para poder modificar las creencias si llega a ser necesario, ya que el siguiente objetivo será no volver a cometer el mismo error, algo que no ocurre cuando seguimos manteniendo la misma posición al respecto, rechazando todo el nuevo contenido.
- **Sesgo de autoridad.** Seguir lo que plantean expertos, sin tener en cuenta los propios deseos.

Ahora que sabemos que es una decisión y cuáles son sus componentes veamos que hay que tener en cuenta antes de tomar una decisión, es importante detectar cuál es su origen o detonante. A partir de ahí, se recomienza analizar toda la información de forma lógica y racional, además de identificar posibles problemas y/o oportunidades. Aplicar la escala de prioridades puede facilitar este trabajo. Sin embargo, hay que evitar dejarse llevar únicamente por las emociones, la intuición o experiencias pasadas.

La toma de decisiones se ha convertido en una dinámica más del entorno cotidiano de las personas. Tanto es así que, a veces, se llevan a cabo de forma inconsciente o impulsiva. Es en estos casos en los que hay que poner más atención, ya que son los más peligrosos.

Desde el punto de vista empresarial, repetir los fracasos del pasado; justificar constantemente los errores; delegar las tareas por exceso y por defecto; ser dependientes; y la falta de compromiso, son algunos de los factores que pueden llevar a tomar decisiones inapropiadas para las compañías.

Tipos de decisiones

Las decisiones pueden organizarse atendiendo a dos criterios diferentes: por método y por niveles: [3]

1. **Por método:**
    - *Decisiones racionales o estructuradas:* tomadas en base a un proceso o criterio específico (por ejemplo, unas reglas o leyes) y su surgimiento se da de manera repetitiva y rutinaria. A su vez, por su naturaleza, son predecibles.
    - *Decisiones intuitivas o no estructuradas:* son aquellas decisiones que deben ser tomadas para encarar o afrontar una situación nueva. A diferencia de las primeras, no parten de un proceso o criterio previo porque no ha surgido antes la necesidad de afrontarlas o por contar con una naturaleza compleja y no son predecibles.

2. **Por niveles:**
    - *Decisiones planificadas:* las decisiones planificadas por lo general, son tomadas por los líderes más altos en la escala jerárquica de una empresa o corporación e incumben a las relaciones entre la entidad y su entorno, son decisiones de gran trascendencia y no repetitivas. Requieren de un gran análisis de la información y reflexión.
    - *Decisiones tácticas*: pueden ser repetitivas -o no- y, con frecuencia, las toman directivos de rango intermedio.
    - *Decisiones operativas:* relacionadas con actividades cotidianas de la empresa y de ellas se encargan los líderes de nivel inferior.

Etapas del proceso

“El proceso de la toma de decisiones debe ser de elaboración simple; aplicable tanto para grupos como para individuos; natural a la intuición y al pensamiento general; alentar la transacción y la formación de consensos; no requerir demasiada especialización para dominarla y comunicarla; y ser fácil de examinar”. (Saaty, 2014, como se menciona en https://cepymenews.es/proceso-toma-decisiones/)

Se pueden identificar seis etapas básicas en la elaboración de un proceso de toma de decisiones:

[3] https://cepymenews.es/proceso-toma-decisiones/

- **Identificación de la dificultad:** identificar la dificultad o problema para poder afrontarla y el origen de la urgencia que conlleva. A partir de allí, se definen los criterios a seguir para su mejor resolución.
- **Propuesta de soluciones o alternativas:** compilar todas las alternativas que sean aplicables a dicho problema y que sean de mayor utilidad para la resolución del mismo.
- **Evaluación:** con todas las propuestas sobre la mesa, analizar cada una de ellas, evaluar sus pros y contras, además de las posibles consecuencias que puedan desprenderse de ellas.
- **Elección de la mejor solución o alternativa:** tras superar la etapa anterior, y tras hacer un ejercicio de análisis y razonamiento, llega el momento de elegir la decisión a adoptar. Para ello se discriminarán en primer lugar las que más se alejen de las necesidades u objetivos, hasta encontrar la que se considere más acertada.
- **Implantación de la solución o alternativa:** una vez tomada la decisión, hay que informar sobre la misma a las personas o empresas afectadas. Esto requiere de un proceso de planificación y organización previo.
- **Análisis de los resultados:** La última etapa debe ser el análisis de los resultados para comenzar el ciclo en caso de ser necesario. Encontrar solución al problema inicial sin que suponga la aparición de otros, sería el ejemplo de éxito más significativo. De ser así, la alternativa elegida podría aplicarse a otros conflictos similares. En caso contrario, otorgaría información para evitar errores en el futuro.

La toma de decisiones es un proceso adherido, especialmente, a los líderes, pero no deja de ser una habilidad que cualquier persona de la empresa puede aprender y desarrollar. Aunque en un primer momento pueda ser difícil de comprender o implantar, la mayoría de las personas tienen interiorizada esta dinámica por su propia experiencia. Es elemental tomar las decisiones correctas, ya que de ellas depende el éxito.

Siempre se deben tomar decisiones sobre cómo hacer para que un negocio funcione y alcance su potencial. Los datos siempre deben ser centrales para la toma de decisiones.

Capítulo 2 - Arquitectura de software en aplicaciones empresariales

Las aplicaciones empresariales son soluciones de software diseñadas para resolver problemas grandes y cumplir con ciertos requerimientos para clientes empresariales en los sectores de IT, gobierno, educación y público. Estas aplicaciones permiten transformar digitalmente sus negocios con capacidades como compra de productos, procesamiento de pagos, facturación automatizada y administración de clientes.

Para garantizar que los sistemas empresariales sigan siendo altamente confiables, disponibles y de alto rendimiento, es muy importante obtener el diseño y la arquitectura correctos. El diseño y la arquitectura forman la base de cualquier buen software. Fundamentan el resto del ciclo de vida del desarrollo de software y, por lo tanto, son muy importantes para disminuir cualquier reproceso posterior, lo que podría resultar muy costoso dependiendo de los cambios que sean necesarios.

En este capítulo se expondrá la teoría de las arquitecturas de software y tecnologías que se han utilizado como base para este trabajo y su utilidad en aplicaciones empresariales modernas.
Hay algunos principios y arquitecturas que se practican comúnmente al diseñar aplicaciones empresariales. El propósito general o el objetivo de tener una arquitectura de software correspondiente al negocio es que respalde las necesidades comerciales al menor costo posible (los costos son tiempo y recursos), por ejemplo, el tener una buena arquitectura podría ayudar a realizar Unit Tests y por lo tanto se podría evitar realizar una regresión total al realizar cambios, siendo este un ejemplo de ahorra en tiempos y recursos.

Una empresa quiere software que pueda ser utilizado de forma segura, estable y con cierta performance. Por lo tanto la disponibilidad, la confiabilidad y el rendimiento son los tres KPI de cualquier sistema, podemos definir a estos KPIs como los indicadores clave de rendimiento (KPI) y son valores que miden el rendimiento de su negocio en general, por lo tanto una selección incorrecta de software podría generar un cuello de botella y muchas veces que nuestro software no sea apropiado llevando al fracaso del sistema, por estas razones la selección de una buena arquitectura es principalmente importante.
Antiguamente se utilizaban arquitecturas de software monolíticas, este tipo de aplicaciones en la que la capa de interfaz de usuario y la capa de acceso a datos están combinadas en un mismo

programa y sobre una misma plataforma, poseen varios problemas, por ejemplo, en una aplicación monolítica, la única forma de escalar horizontalmente es agregando más computación al sistema. Esto conduce a mayores costos operativos y una utilización de recursos no optimizada. A veces, el escalado se vuelve imposible debido a necesidades conflictivas en términos de recursos.

Como todas las funciones utilizan principalmente almacenamiento único, existe la posibilidad de que se produzcan bloqueos que provoquen una latencia alta y también habrá límites físicos en cuanto a hasta dónde puede escalar una única instancia de almacenamiento.

A continuación, se muestran algunos problemas asociados con la disponibilidad, la confiabilidad y el rendimiento.

- Cualquier cambio en el sistema requerirá la redistribución de todos los componentes, lo que generará tiempo de inactividad y baja disponibilidad.
- Cualquier estado no persistente, como las sesiones almacenadas en una aplicación web, se perderá después de cada implementación. Esto conducirá al abandono de todos los flujos de trabajo que fueron activados por los usuarios.
- Cualquier error en un módulo, como pérdidas de memoria o errores de seguridad, hace que todos los módulos sean vulnerables y tienen el potencial de afectar a todo el sistema.
- Debido a la naturaleza altamente acoplada y al intercambio de recursos dentro de los módulos, siempre habrá un uso no optimizado de los recursos, lo que generará una alta latencia en el sistema.

Estos son algunas de las razones por las cuales se ha decidido no utilizar un sistema monolítico en el presente trabajo, y si se ha utilizado un sistema separado en capas y con distintos servicios.

Separation of concerns - Separación de preocupaciones / arquitectura de responsabilidad única

El software debe dividirse en componentes o módulos según el tipo de trabajo que realiza. Cada módulo o componente debe tener una única responsabilidad. La interacción entre los componentes debe realizarse a través de interfaces o sistemas de mensajería. Veamos la arquitectura de n-tier y microservicios y cómo se soluciona la separación de concerns o preocupaciones.

N-tier architecture

Una arquitectura en capas o de “n capas” divide la aplicación de un sistema en tres o más niveles:

- Presentación (conocida como capa UX, capa UI o superficie de trabajo)
- Negocios (conocida como capa de reglas comerciales o capa de servicios)
- Datos (conocida como capa de acceso y almacenamiento de datos)

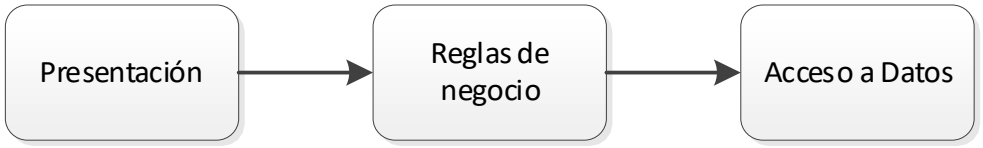


Figure 1 - N Tier Architecture

Estos niveles se pueden poseer / administrar / implementar por separado. Por ejemplo, varias capas de presentación, como las capas web, móviles y de bot, pueden aprovechar el mismo nivel empresarial y de datos.

Arquitectura basada en microservicios (Microservices architecture)

La arquitectura de microservicios consta de servicios pequeños, poco acoplados, independientes y autónomos. Los servicios se pueden implementar y escalar de forma independiente. Un problema en un servicio tendrá un impacto local y se puede solucionar simplemente implementando el servicio afectado. No es necesario compartir una tecnología o un marco.

Podemos definir un microservicio como un servicio, modular, que se puede implementar y desarrollar de forma independiente, que aborda un proceso o problema empresarial específico y único, y se comunica a través de una arquitectura ligera basada en eventos, asíncrona y basada en mensajes. Básicamente, es una serie de servicios que provee la funcionalidad a una determinada parte del negocio.
A pesar de que la palabra microservicio se refiere a un servicio pequeño no tiene por qué ser así, la utilización de microservicios favorece en los siguientes aspectos.

- Cada microservicio se puede implementar, desarrollar, mantener y luego volver a implementar de forma independiente.
- Cada microservicio se enfoca en un propósito y objetivo comercial específico y no es monolítico.
- Cada microservicio recibe solicitudes, las procesa y luego puede o no enviar una respuesta.
- Los microservicios practican la gobernanza descentralizada y, en algunos casos, cuando está permitido, la gestión de datos descentralizada.

Los microservicios están diseñados para fallar. Al seguir este paradigma, siempre se podrá manejar los fallos con elegancia y no permitir que un microservicio fallido afecte negativamente a todo el ecosistema.

Por ejemplo, en la solución propuesta en este trabajo si el microservicio de manejo de productos tiene alguna falla **el portal de ventas puede seguir operando de forma continua, realizando ventas y asentando las intenciones de compra.**

Estos son algunos de los puntos positivos de una arquitectura de microservicio:

- Brindan a los desarrolladores la libertad de diseñar, desarrollar e implementar servicios de forma independiente.
- Los microservicios se pueden desarrollar en diferentes idiomas si está permitido
- Integración e implementación más fáciles que las aplicaciones y servicios monolíticos tradicionales
- Los microservicios están organizados en torno a capacidades comerciales específicas
- Cuando se requiere un cambio, solo se debe cambiar y volver a implementar el microservicio específico
- Aislamiento de fallas mejorado

- Son más fáciles de escalar
- Se facilita la integración a servicios externos

Stateless services architecture (Arquitectura de servicios sin estado)

Los servicios no deben tener ningún estado. El estado y los datos deben administrarse independientemente de los servicios, es decir, externamente. Al delegar el estado externamente, los servicios tendrán los recursos para atender más solicitudes con alta confiabilidad.

La afinidad de sesión no debe habilitarse, ya que genera problemas de sesiones complicadas y evitará que obtenga los beneficios del equilibrio de carga, la escalabilidad y la distribución del tráfico.

Se ha tenido en cuenta la separación de preocupaciones / SRP en cada nivel. El nivel de presentación que contiene la interfaz de usuario está separado del nivel de servicios que contiene la lógica empresarial, que nuevamente está separado del nivel de acceso a datos que contiene el almacén de datos.

Los componentes de alto nivel desconocen que los componentes de bajo nivel los consumen. El nivel de acceso a datos desconoce los servicios que lo consumen y los servicios desconocen el nivel de UX que los consume.
Cada servicio se separa según la lógica empresarial y la funcionalidad que se supone que debe realizar.

La encapsulación se ha cuidado a nivel de arquitectura y también debe cuidarse durante el desarrollo. Cada componente de la arquitectura interactuará con otros componentes a través de interfaces y contratos bien definidos. Deberíamos poder reemplazar cualquier componente en el diagrama sin preocuparnos por su implementación interna si se adhiere a los contratos.

La arquitectura poco acoplada aquí también ayuda a un desarrollo más rápido y una implementación más rápida. Varios equipos pueden trabajar en paralelo en cada uno de sus componentes de forma independiente. Comparten los contratos y los plazos para las pruebas de integración al principio, y una vez que se realizan la implementación interna y las pruebas unitarias, pueden comenzar con las pruebas de integración.

Inversión de control e Inyección de dependencias

Un gran problema al que puede enfrentarse una aplicación empresarial es la complejidad de conectar diferentes elementos y administrar su vida útil. Para abordar esto, utilizamos el principio de inversión de control (IoC), que recomienda eliminar la dependencia entre objetos. Al delegar el flujo de control, IoC hace que el programa sea extensible y aumenta el modularidad. Los eventos, los delegados de devolución de llamada, el patrón de observador y la inyección de dependencia (DI) son algunas de las formas de lograr IoC.

La DI es una técnica en la que un objeto recibe los objetos de los que depende. El patrón DI cumple con el principio DI cubierto como parte de los principios de diseño SOLID. Con el uso de DI, el código será más fácil de mantener, legible, comprobable y extensible. DI es uno de los métodos más conocidos para ayudar a lograr un código mejor mantenible. DI tiene tres entidades involucradas, como se muestra en la figura 2:

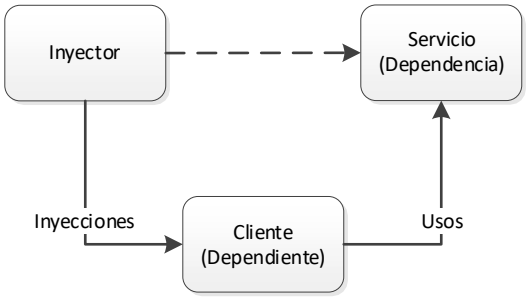


Figure 2 - Inyección de dependencias

El Injector crea una instancia de Service y la inyecta en el objeto Client. El cliente depende del servicio inyectado para realizar sus operaciones. IoC Container, también conocido como DI Container, es un marco para implementar DI automático. En la Figura 2, se le conoce como Inyector. Es responsable de crear o hacer referencia a la dependencia e inyectarla en el Cliente.

Hay varias formas de inyectar un servicio en una dependencia. Según la forma en que se inyecta el servicio en el objeto del cliente, DI se clasifica en tres tipos:

- **Inyección de constructor:** las dependencias se inyectan a través de un constructor mientras se instancia el dependiente.
- **Inyección de setter:** en el caso de la inyección de setter, el dependiente expone un método o propiedad de setter que el inyector usa para inyectar la dependencia.

- **Inyección de método:** otra forma de inyectar dependencia es pasándola como parámetro de método.

.NET Foundation

El backend del sistema propuesto se desarrolló en el lenguaje C# que es parte del ecosistema de .NET, .NET es de código abierto y pertenece a .NET Foundation. Es una organización independiente que fomenta el desarrollo abierto y la colaboración en torno al ecosistema .NET.
C# es un lenguaje de programación de propósito general, tanto C# como Java son lenguajes de la familia de C, pero C# tiene muchos aspectos también de Visual Basic. Por otro lado, C# admite una serie de características, como expresiones lambda y tipos anónimos, que tradicionalmente se encuentran en varios lenguajes funcionales (por ejemplo, LISP o Haskell). Además, posee Language Integrated Query (LINQ), esto hace que C# admita una serie de construcciones que lo hacen bastante único en el panorama de la programación. Sin embargo, la mayor parte de C# está influenciada por los lenguajes basados en C.

C# posee gestión automática de memoria mediante Garbage Collector. Dado esto, C# no admite una palabra clave de eliminación. C# agrega construcciones sintácticas formales para clases, interfaces, estructuras, enumeraciones y delegados.

C# tiene la capacidad similar a C ++ de sobrecargar operadores para un tipo personalizado, sin la complejidad. Además, tiene soporte para programación basada en atributos. Esta marca de desarrollo le permite anotar tipos y sus miembros para calificar aún más su comportamiento. Por ejemplo, si marca un método con el atributo [Obsoleto], los programadores verán su mensaje de advertencia personalizado impreso si intentan hacer uso del miembro decorado.

Entity Framework

Dentro de la solución propuesta se ha especificado una capa de abstracción de Datos, la misma se encuentra dentro de la aplicación como un proyecto DataAccess, el mismo posee una clase que especifica el contexto de la base de datos, esta capa es utilizada por las clases de negocio y posee la definición de la clase repository, además posee la definición del contexto siendo este última la que posee una referencia a Entity framework.
Entity framework es una librería que permite realizar el mapeo de objetos con una base de datos

relacional, dando la posibilidad de realizar consultas con un lenguaje moderno y poderoso, Linq.

Para la generación de base de datos a partir del modelo de objetos se utilizó Entity Framework con code-first, como ya se mencionó la implementación del mismo se encuentra en la capa de acceso a datos dentro de la solución de backend.

Entity Framework introdujo el enfoque Code-First con Entity Framework 4.1. Code-First es principalmente útil en el diseño controlado por dominios. En el enfoque Code-First, usted se enfoca en el dominio de su aplicación y comienza a crear clases para su entidad de dominio en lugar de diseñar su base de datos primero y luego crear las clases que coincidan con el diseño de su base de datos. La figura 5 ilustra el enfoque de Code-First (código primero).

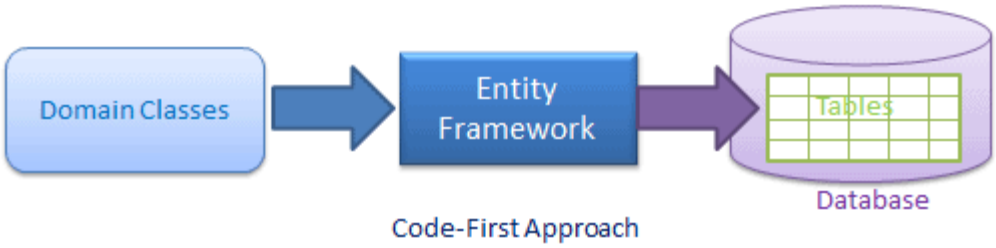


Figure 3 - Entity framework – code first approach

Como puede ver en la figura 5, EF API creará la base de datos en función de las clases y la configuración de su dominio. Esto significa que primero se debe comenzar a codificar en C# y luego EF creará la base de datos a partir de su código.

Code-First Workflow

La figura 6 ilustra el flujo de trabajo de desarrollo de código primero.

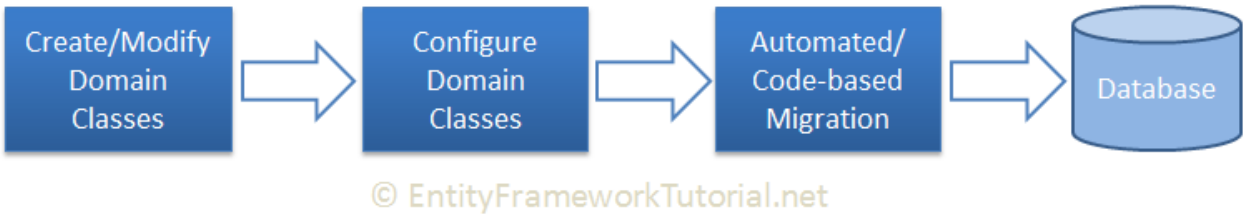


Figure 4 - Entity Framework - Domain

El flujo de trabajo de desarrollo en el enfoque de código primero sería: Crear o modificar clases de dominio -> configurar estas clases de dominio usando Fluent-API o atributos de anotación de datos -> Crear o actualizar el esquema de base de datos usando migración automatizada o migración basada en código.

Angular:

Se ha utilizado el framework Angular como plataforma de desarrollo para la interfaz gráfica del sistema ya que aporta muchos beneficios. Angular es un framework para crear aplicaciones cliente en HTML y JavaScript o typescript (un lenguaje que se compila en JavaScript).

El marco de angular consta de varias bibliotecas, algunas de ellas centrales y otras opcionales. Angular representa una reescritura completa del marco AngularJS, presentando una arquitectura de aplicación completamente nueva construida completamente desde cero en TypeScript, un superconjunto estricto de JavaScript que agrega escritura estática opcional y soporte para interfaces y decoradores.

En pocas palabras, las aplicaciones Angular se basan en un diseño de arquitectura que comprende árboles de componentes web interconectados por su interfaz de E/S particular. Bajo el capó, cada componente aprovecha un mecanismo de inyección de dependencia completamente renovado. Entre otras cosas Angular tiene las siguientes características.

- Basado en componentes
- Inyección de dependencias
- Enrutamiento
- Formularios web
- Comunicación HTTP

Se escriben aplicaciones angulares componiendo plantillas HTML con marcado Angularized, escribiendo estas clases de componentes para administrar esas plantillas, agregando lógica de aplicación en servicios y componentes y servicios de boxing en módulos

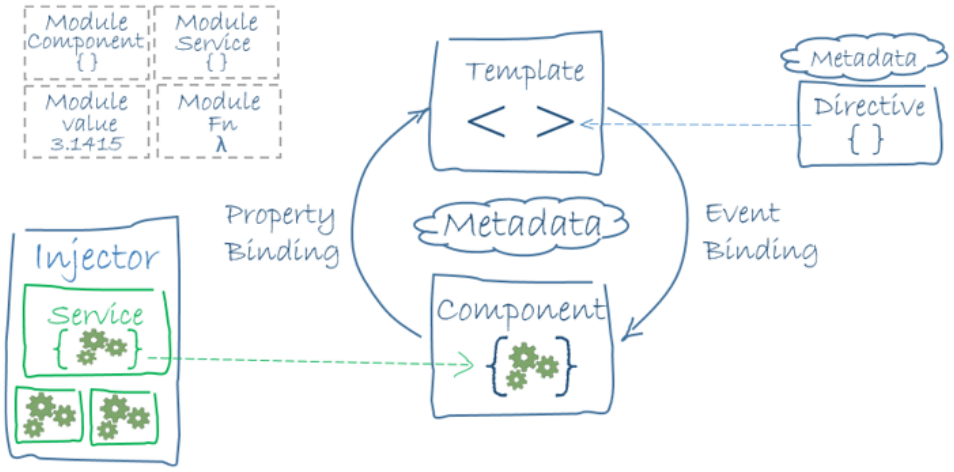


Figure 5 - Arquitectura general de Angular

Capítulo 3 - Machine learning (aprendizaje automático).

Surgimiento del Machine Learning

En los últimos años, el aprendizaje automático se ha convertido en una de las ramas de inteligencia artificial y TI más importantes y productiva.
Las aplicaciones de machine learning se han extendido en todos los sectores empresariales, con herramientas y resultados nuevos y más potentes. Los marcos de código abierto listos para la producción, junto con cientos de artículos publicados cada mes, contribuyen a uno de los procesos de democratización más generalizados en la historia de la IA. Para poder abordar temas de aprendizaje automático empezaremos realizando un pequeño repaso por las definiciones de IA y sus orígenes.

Barr y Feigenbaum dan una definición sobre la inteligencia artificial:
La inteligencia artificial es la parte de las ciencias computacionales que se encarga de diseñar sistemas de cómputo inteligentes, esto es, sistemas que exhiban características que asociamos con la inteligencia en el comportamiento humano.

El ganador del Pulitzer **Douglas Hofstadter**, creo una lista de las habilidades fundamentales de la inteligencia. Es decir, que una entidad inteligente, sea humana o computacional, debe cubrir la siguiente lista:

- Responder a las situaciones de manera muy flexible
- Darle sentido a mensajes ambiguos o contradictorios
- Reconocer la importancia relativa de los diferentes elementos en una situación
- Encontrar similitudes entre situaciones a pesar de las diferencias que las separan
- Encontrar diferencias entre situaciones a pesar de las similitudes que las ligan.

Alan Turing fue el primero en definir una visión de la IA en su artículo. ***Computing Machinery and Intelligence*****,** en 1950. Ahí, introdujo la prueba de Turing, el aprendizaje automático, los algoritmos genéricos y el aprendizaje por refuerzo.
Si el lector está interesado, en el anexo 1 de este trabajo, se realizó un repaso simple por la historia de la inteligencia artificial y su surgimiento.

La idea de máquinas conscientes e inteligentes no es algo nuevo, sin embargo, gracias a los avances de estos últimos años, es posible procesar grandes volúmenes de datos a una escala sin igual. Es así que se podría decir que vivimos en una era de Información, siendo uno de los principales desafíos de las empresas y organizaciones el poder obtener información a partir de todos estos datos, que les permita tomar mejores decisiones (data-driven decisiones).

Para poder tomar decisiones inteligentes a partir de una gran cantidad de datos **no alcanza** con los paradigmas de programación tradicionales. El programador debería contar con un conocimiento muy amplio del dominio sobre el cual está trabajando para poder determinar todas las correlaciones existentes entre las distintas variables de sus datos. Luego, debería invertir mucho tiempo en la codificación de conjuntos de reglas extremadamente complejos para intentar llevar a cabo cualquier tipo de análisis (un sistema basado en reglas) sobre los datos disponibles. Esta aproximación presenta varias deficiencias. Por un lado, ciertas relaciones podrían ser menos evidentes y podrían ser pasadas por alto. Por otro lado, este tipo de sistemas son muy costosos de mantener. Cualquier cambio que se introduzca, ya sea para crear una nueva regla, modificar o eliminar una existente, es costoso y puede introducir errores.

Así surgió el concepto de machine learning o aprendizaje automático, como una necesidad de tomar decisiones más rápidamente y de mejor calidad. A diferencia de los paradigmas tradicionales, basados en reglas, machine learning utiliza un conjunto de datos, denominados observaciones, para la construcción de un modelo. Este modelo utilizará dichos datos para deducir todas los posibles patrones y correlaciones existentes entre ellos. A partir de este conocimiento adquirido, el modelo será capaz de predecir valores de salida para nuevas observaciones no vistas anteriormente.

Hoy en día existe una gran cantidad de información proveniente de distintos orígenes, se calcula que tenemos aproximadamente 40 Zettabytes (43 trillones de gigabytes) de información en todo el mundo.

Como se observa en la figura 8, estos datos pueden provenir de distinto orígenes.

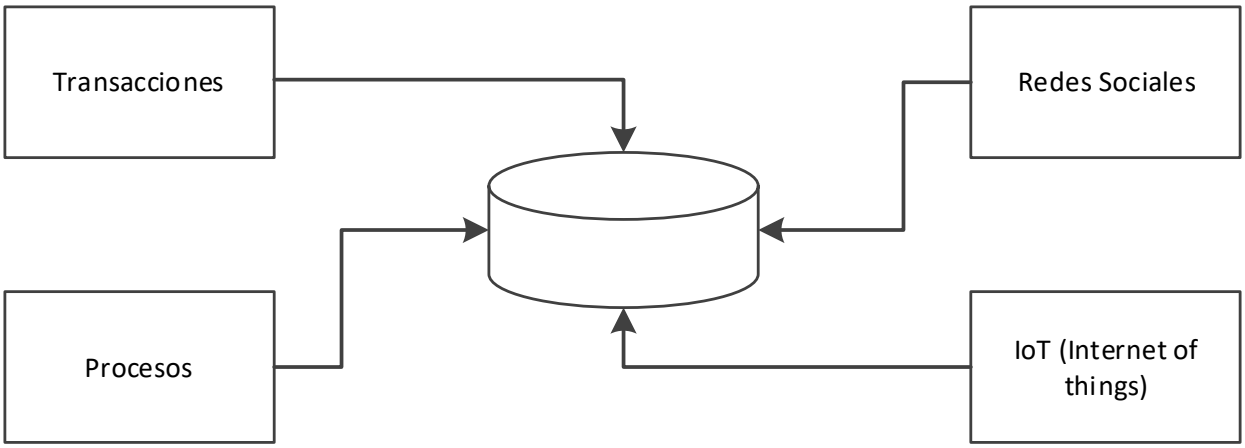


Figure 6 - orígenes de datos

Machine Learning o aprendizaje automático es un subgrupo de la Inteligencia Artificial. Se basa en crear sistemas que puedan aprender automáticamente, es decir, pueden descubrir patrones complejos enterrados en grandes conjuntos de datos sin la necesidad de interferencia humana.

En otras palabras, el aprendizaje automático (machine learning) es el campo dedicado al desarrollo de métodos computacionales para los procesos de aprendizaje, y a la aplicación de los sistemas informáticos de aprendizaje a problemas prácticos, por otro lado, la **minería de datos** como se ejemplifico es la búsqueda de patrones e importantes regularidades en las bases de datos de gran volumen, estos dos campos han crecido a lo largo de los años y han cobrado una importancia considerable. El campo de minería de datos se detalla en el anexo 3 de este trabajo, a pesar de que no se utiliza para la solución de proyecto es importante para el lector tener una idea de lo que es minería de datos y lo que no, y poder diferenciarlo de machine learning.

Puede ser difícil ver cómo el aprendizaje automático (ML) afecta la vida cotidiana de la gente común. De hecho, ¡ML está en todas partes! En el proceso de búsqueda de un restaurante para cenar, en la búsqueda de un vestido para una cena, en el camino una cita para cenar, si alguien ha utilizado una alguna de las aplicaciones para compartir viajes ha utilizado ML.
ML se ha utilizado tanto, que se ha convertido en una parte esencial de nuestras vidas, aunque generalmente no se nota.
Con datos cada vez mayores y su accesibilidad, las aplicaciones y necesidades de ML están aumentando rápidamente en varias industrias. Sin embargo, el ritmo de crecimiento de los científicos de datos capacitados aún no ha alcanzado el ritmo de crecimiento de las necesidades

de ML en las empresas, a pesar de los abundantes recursos y bibliotecas de software que facilitan la creación de modelos de ML, debido al hecho de que se necesita tiempo y experiencia para que científicos e ingenieros de ML **puedan dominar tales conjuntos de habilidades**.

El aprendizaje automático es un término que a menudo se confunde con la inteligencia artificial. Se origina en la década de 1950, y fue definido por primera vez por **Arthur Lee Samuel en 1959** como se mencionó en el capítulo 1 de este trabajo.

Definición de aprendizaje automático.

Se ofrecen dos definiciones de aprendizaje automático. Arthur Samuel lo describió como: "el campo de estudio que brinda a las computadoras la capacidad de aprender sin ser programado explícitamente". Esta es una definición más antigua e informal.

Tom Mitchell proporciona una definición más moderna: **"Se dice que un programa de computadora aprende de la experiencia E con respecto a alguna clase de tareas T y medida de desempeño P, si su desempeño en tareas en T, medido por P, mejora con la experiencia E. ".**

Ejemplo: jugar a las damas.

E = la experiencia de jugar muchos juegos de damas

T = la tarea de jugar damas.

P = la probabilidad de que el programa gane el próximo juego.

En general, cualquier problema de aprendizaje automático puede asignarse a una de las siguientes clasificaciones amplias:

1. **Aprendizaje supervisado**
2. **Aprendizaje no supervisado**

A partir de estas definiciones, podemos concluir que el aprendizaje automático es una forma de lograr la inteligencia artificial. Sin embargo, se puede tener inteligencia artificial sin aprendizaje automático.

Existen muchas áreas donde Machine Learning está siendo usado en nuestras vidas sin ser notado. Compañías de medios utilizan machine learning para recomendar el contenido más relevante, como artículos de diarios, películas o música. Las compañías de e-commerce usan ML para sugerir los ítems de tu interés, compañías de video juegos usan ML para detectar tu movimiento y emularlo.

Las empresas de comercio electrónico utilizan ML para sugerir los artículos que son de su interés y que es más probable que compre. Las compañías de juegos usan ML para detectar su movimiento y movimientos articulares para sus juegos de sensores de movimiento. Algunos otros usos comunes de ML en la industria incluyen la detección de rostros en cámaras para un mejor enfoque, respuesta automática de preguntas donde los bots de chat o asistentes virtuales interactúan con los clientes para responder preguntas y solicitudes, y detectar y prevenir transacciones fraudulentas. En esta sección, veremos algunas de las aplicaciones que utilizamos en nuestra vida diaria que utilizan ML en gran medida:

Algunos ejemplos que hoy utilizan algunas empresas.

- **Google News feed**: El feed de noticias de Google utiliza ML para generar una secuencia personalizada de artículos en función de los intereses del usuario y otros datos de perfil. Los algoritmos de filtrado colaborativo se utilizan con frecuencia para tales sistemas de recomendación y se crean a partir de los datos del historial de vistas de su base de usuarios. Las compañías de medios utilizan dichos sistemas de recomendación personalizados para atraer más tráfico a sus sitios web y aumentar el número de suscriptores.

- **Amazon product recommendations**: Amazon utiliza los datos del historial de búsqueda y pedidos de los usuarios para capacitar a un modelo de ML para recomendar productos que es más probable que compre un usuario. Este es un buen caso de uso para el aprendizaje supervisado en la industria del comercio electrónico. Estos algoritmos de recomendación ayudan a las empresas de comercio electrónico a maximizar sus ganancias al mostrar los elementos que son más relevantes para los intereses de cada usuario.

- **Netflix movie recommendation**: Netflix utiliza clasificaciones de películas, historial de vistas y perfiles de preferencias para recomendar otras películas que le gusten a un usuario. Entrenan algoritmos de filtrado colaborativo con datos para hacer recomendaciones personalizadas. Teniendo en cuenta que más del 80 por ciento de los programas de televisión que las personas miran en Netflix se descubren a través del sistema de recomendaciones de la plataforma según un artículo en Wired (http://www.wired.co.uk/article/how-do-netflixs-algorithms -working-machine-learning-ayuda-a-predecir-qué-les gustará a los espectadores), este es un ejemplo muy útil y rentable de ML en una empresa de medios.

- **Face detection on cameras**: Las cámaras detectan rostros para un mejor enfoque y medición de luz. Este es el ejemplo más utilizado de visión por computadora y clasificación. Además, algunos softwares de administración de fotografías utilizan algoritmos de agrupamiento para agrupar caras similares en sus imágenes para que pueda buscar fotos de ciertas personas en ellas más adelante.

- **Alexa – Virtual assistant**: Los sistemas de asistente virtual, como Alexa, pueden responder preguntas como ¿Qué tiempo hace en Nueva York? o completar ciertas tareas, como encender las luces de la sala de estar. Este tipo de sistema de asistente virtual generalmente se construye utilizando reconocimiento de voz, comprensión del lenguaje natural (NLU), aprendizaje profundo y varias otras tecnologías de aprendizaje automático.

- **Microsoft Xbox Kinect**: Kinect puede detectar qué tan lejos está cada objeto del sensor y detectar posiciones conjuntas. Kinect está entrenado con un algoritmo de decisión aleatoria de tipo bosque, que construye muchos árboles de decisión individuales a partir de imágenes de profundidad.

Pasos para crear modelos de ML

Ahora que hemos visto algunos ejemplos de aplicaciones de ML existentes en el mundo real. Empezaremos definiendo cuáles son los pasos principales a seguir al momento de crear un modelo de ML.

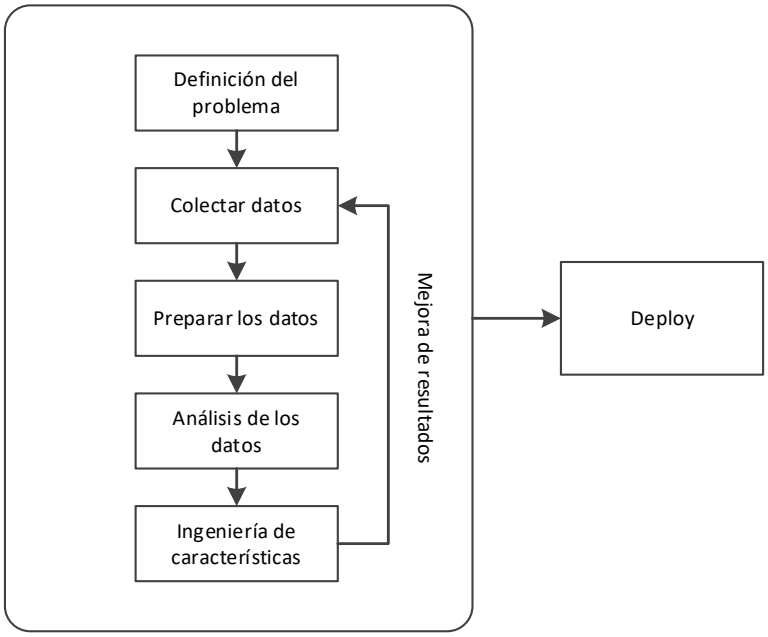


Figure 7 - Pasos para crear modelos de ML

Cómo se puede ver en el diagrama anterior, los pasos a seguir son los siguientes:

- **Definición del problema**: El primer paso para comenzar cualquier Proyecto no es solo entender el problema sino también definir el problema que uno está tratando de resolver utilizando ML, una definición pobre dará como resultado un sistema de ML sin sentido ya que los modelos habrán sido entrenados y optimizados para un problema que en realidad no está tratando de resolver. Este primer paso es indiscutiblemente el paso más importante en la construcción de modelos y aplicaciones de ML útiles. Al menos debe responder las siguientes cuatro preguntas antes de comenzar a construir modelos de ML.

    - **¿Cuál es el problema?** Aquí es donde describe y declara el problema que está tratando de resolver. Por ejemplo, una descripción del problema puede necesitar un sistema para evaluar la capacidad del propietario de una pequeña

empresa para pagar un préstamo para un proyecto de préstamo para una pequeña empresa.

- **¿Por qué es un problema?** Es importante definir por qué tal problema es realmente un problema y por qué el nuevo modelo ML será útil. Tal vez ya tenga un modelo que funcione y haya notado que está funcionando peor que antes; es posible que haya obtenido nuevas fuentes de datos que puede usar para construir un nuevo modelo de predicción; o tal vez desee que su modelo existente produzca resultados de predicción más rápidamente. Puede haber múltiples razones por las que cree que esto es un problema y por qué necesita un nuevo modelo. Definir por qué es un problema lo ayudará a mantenerse en el camino correcto mientras construye un nuevo modelo de ML.

- **¿Cuáles son algunos de los enfoques para resolver este problema?** Aquí es donde intercambias ideas sobre tus enfoques para resolver el problema dado. Debería pensar cómo se usará este modelo (¿necesita que sea un sistema en tiempo real o se ejecutará como un proceso por lotes?), Qué tipo de problema es (es un problema de clasificación, regresión, agrupamiento u otra cosa?), y qué tipos de datos necesitaría para su modelo. Esto proporcionará una buena base para futuros pasos en la construcción de su modelo de aprendizaje automático.

- **¿Cuáles son los criterios de éxito?** Aquí es donde define sus puntos de control. Debe pensar qué métricas analizará y cómo debería ser el rendimiento de su modelo de destino. Si está creando un modelo que se utilizará en un sistema en tiempo real, también puede establecer la velocidad de ejecución objetivo y la disponibilidad de datos en tiempo de ejecución como parte de sus criterios de éxito. Establecer estos criterios de éxito lo ayudará a seguir avanzando sin atascarse en un determinado paso.

- **Recolección de datos**: Obtener los datos es la parte más esencial y crítica de la construcción de un modelo de ML, preferiblemente muchos datos. Sin datos, no hay modelo. Dependiendo de su proyecto, sus enfoques para recopilar datos pueden variar. Puede comprar fuentes de datos existentes de otros proveedores, puede raspar sitios

web y extraer datos de allí, puede usar datos disponibles públicamente o también puede recopilar sus propios datos. Hay varias formas en que puede recopilar los datos que necesita para su modelo de ML, pero debe tener en cuenta estos dos elementos de sus datos cuando está en el proceso de recopilación de datos: la variable objetivo y las variables de características. La variable objetivo es la respuesta para sus predicciones y las variables de características son los factores que usarán sus modelos para aprender a predecir la variable objetivo. A menudo, las variables objetivo no están presentes en forma etiquetada. Por ejemplo, cuando se trata de datos de Twitter para predecir el sentimiento de cada tweet, es posible que no haya etiquetado los datos de sentimiento para cada tweet. En este caso, tendrá que dar un paso adicional para etiquetar las variables de destino. Una vez que haya recopilado sus datos, puede pasar al paso de preparación de datos.

- **Data preparation**: Una vez que haya reunido todos sus datos de entrada, debe prepararlos para que estén en un formato utilizable. Este paso es más importante de lo que piensas. Si tiene datos desordenados y no los limpió para sus algoritmos de aprendizaje, sus algoritmos no aprenderán bien de su conjunto de datos y no funcionarán como se esperaba. Además, incluso si tiene datos de alta calidad, si sus datos no están en un formato con el que puedan entrenarse sus algoritmos, entonces no tiene sentido tener datos de alta calidad. Malos datos, mal modelo. Al menos debe manejar algunos de los problemas comunes enumerados a continuación para tener sus datos listos para los siguientes pasos:

- **File format**: Si obtiene sus datos de múltiples fuentes de datos, lo más probable es que se ejecute en diferentes formatos para cada fuente de datos. Algunos datos pueden estar en formato CSV, mientras que otros datos están en formato JSON o XML. **Algunos datos podrían incluso almacenarse en una base de datos relacional como los que se han recopilado para el sistema que se implementara al final de este trabajo**. Para entrenar su modelo ML, primero deberá fusionar todas estas fuentes de datos en diferentes formatos en un formato estándar.

- **Data format**: También puede darse el caso de que los formatos de datos varíen entre las diferentes fuentes de datos. Por ejemplo, algunos datos pueden tener el

campo de dirección desglosado en dirección, ciudad, estado y código postal, mientras que otros no. Algunos datos pueden tener el campo de fecha en el formato de fecha estadounidense (mm / dd / aaaa), mientras que otros pueden estar en formato británico (dd / mm / aaaa). Estas discrepancias en el formato de los datos entre las fuentes de datos pueden causar problemas al analizar los valores. Para entrenar su modelo ML, necesitará tener un formato de datos uniforme y único para cada campo.

- **Duplicate records**: A menudo verá los mismos registros exactos que se repiten en su conjunto de datos. Este problema puede ocurrir en el proceso de recopilación de datos en el que registró un punto de datos más de una vez o cuando fusionó diferentes conjuntos de datos en su proceso de preparación de datos. Tener registros duplicados puede afectar negativamente a su modelo y es bueno verificar si hay duplicados en su conjunto de datos antes de continuar con los siguientes pasos.

- **Missing values**: También es común ver algunos registros con valores vacíos o faltantes en los datos. Esto también puede tener un efecto adverso cuando está entrenando sus modelos de ML. Hay varias formas de manejar los valores perdidos en sus datos, pero deberá tener cuidado y comprender muy bien sus datos, ya que esto puede cambiar el rendimiento de su modelo de manera espectacular. Algunas de las formas en que puede manejar los valores perdidos incluyen soltar registros con valores perdidos, reemplazar los valores perdidos con la media o la mediana, reemplazar los valores perdidos con una constante o reemplazar los valores perdidos con una variable ficticia y una variable indicadora para los que faltan. Será beneficioso estudiar sus datos antes de tratar con los valores faltantes.

- **Análisis de los datos**: Ahora que sus datos están listos, es hora de mirar los datos y ver si puede reconocer algún patrón y extraer algunas ideas de los datos. Las estadísticas resumidas y los gráficos son dos de las mejores formas de describir y comprender sus datos. Para variables continuas, mirar el mínimo, el máximo, la media, la mediana y los cuartiles es un buen lugar para comenzar. Para variables categóricas, puede ver los recuentos y porcentajes de categorías. Mientras observa estas estadísticas resumidas,

también puede comenzar a trazar gráficos para visualizar las estructuras de sus datos. Los histogramas se usan con frecuencia para mostrar e inspeccionar las distribuciones subyacentes de variables, valores atípicos y asimetría. Los diagramas de caja se usan con frecuencia para visualizar un resumen de cinco números, valores atípicos y asimetría. Los diagramas de dispersión por pares se usan con frecuencia para detectar correlaciones obvias por pares entre las variables:

Tipos de aprendizajes

Cómo dijimos anteriormente existen dos tipos de aprendizaje automático, el aprendizaje supervisado y el no supervisado.

Aprendizaje supervisado

Es uno de los tipos de problemas de machine learning más comunes, la máquina aprende por medio de ejemplos. El algoritmo se entrena por medio de preguntas, conocidas como características (features); y respuestas, denominadas etiquetas (labels). Cuando el algoritmo tiene la respuesta a cierta pregunta, guarda esa información para hacer previsiones futuras.
En la figura 10 podemos ver lo que hemos explicado hasta el momento.

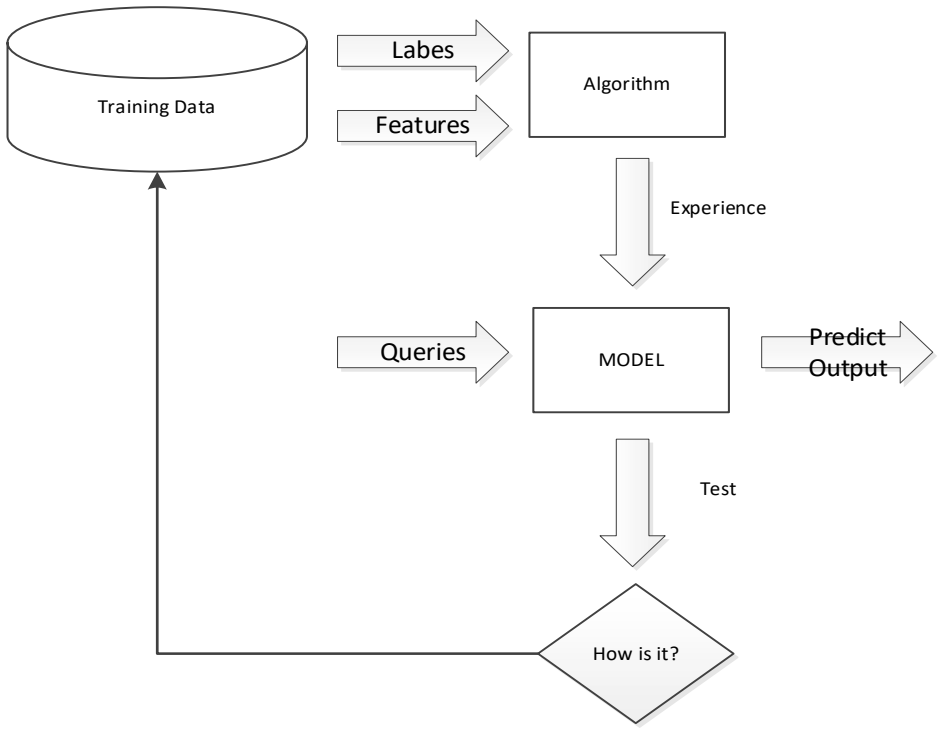


Figure 8 - Características y labels en ML

En el grafico anterior podemos visualizar como las características y las etiquetas se utilizan para entrenar los datos, aplicando determinado algoritmo como puede ser regresión lineal, obtenemos un modelo, al cual podemos consultar, testear y obtener predicciones desde él.

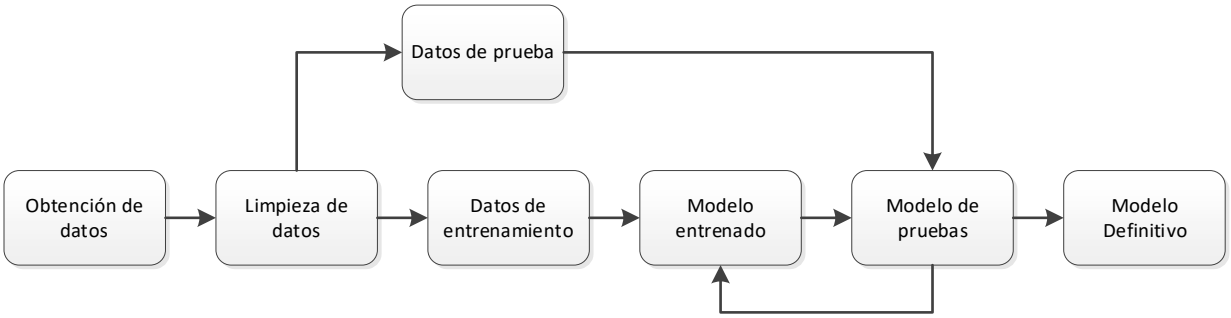


Figure 9 - Flujo de trabajo en ML

Un escenario supervisado se caracteriza por el concepto de un supervisor, cuya tarea principal es proporcionar al agente una medida precisa de su error (directamente comparable con los valores de salida). Con algoritmos reales, esta función es proporcionada por **un conjunto de datos de entrenamiento compuesto por entradas y para estas entradas se proporciona las salidas esperadas)**. A partir de esta información, el agente puede corregir sus parámetros para reducir la magnitud de una función de pérdida global.
Después de cada iteración, si el algoritmo es lo suficientemente flexible y los elementos de datos son coherentes, la precisión general aumenta y la diferencia entre los valores pronosticados y esperados se acerca a cero.

Por el otro lado, en un escenario supervisado, el objetivo es entrenar un sistema que también pueda funcionar con muestras que nunca antes se habían visto. Por lo tanto, es necesario permitir que el modelo desarrolle una capacidad de generalización y evitar un problema común llamado **sobreajuste**, que causa un sobre aprendizaje debido a una capacidad excesiva, que uno de los principales efectos de tal problema es la capacidad de predecir correctamente solo las muestras utilizadas para el entrenamiento, mientras que la tasa de error para las restantes siempre es muy alta).

Demos un ejemplo, queremos realizar un sistema de reconocimiento de imágenes, que a partir de distintas imágenes de gatos y perros el sistema pueda decir si es un gato o un perro. Lo

primero que hacemos es darle a nuestro sistema un grupo de imágenes y decirle si efectivamente es un gato o un perro, a esta información es a la que llamamos de entrenamiento, lo interesante de nuestro sistema será que luego podríamos poder tomar una imagen nueva y que a partir de distintas características el sistema podría identificar si es un gato o un perro.

Podemos dividir el aprendizaje supervisado en dos tipos:

Regresión: Tiene el objetivo de predecir valores continuos. Con el valor numérico de las etiquetas, se utilizan diferentes variables para obtener los datos que nos interesen. Este tipo de aprendizaje sirve para una serie de finalidades concretas como predecir el precio de un producto, una propiedad o el valor de una acción en la bolsa.

Un ejemplo, que puede verse en una gran cantidad de libros de texto o en cursos, para este tipo de algoritmo es predecir el valor de una casa o propiedad, a partir de los valores de las casas de una zona. De esta manera podemos definir ciertas características (**features**) de una casa como la cantidad de metros cuadrados, cantidad de habitaciones etc., y el valor de la casa en si (labels).

Clasificación: La idea de este algoritmo es encontrar diferentes patrones y clasificar los elementos en diferentes grupos.

Este modelo busca sacar conclusiones de los valores observados, ya que una o más entradas intentan predecir el valor de uno o más resultados. Un claro ejemplo es el filtro de correos electrónicos para ver si son spam o no: solo hay dos resultados posibles, ya que cuando analizamos los datos de la transacción podemos dividirlos en dos categorías; ya sea “fraudulento”, o “autorizado”.

Ejemplos comunes de aplicaciones de aprendizaje supervisado:

- Análisis predictivo basado en regresión o clasificación categórica
- Detección de spam
- Detección de patrones
- PNL
- Análisis de los sentimientos
- Clasificación automática de imágenes
- Procesamiento automático de secuencia (por ejemplo, música o voz)

En el siguiente gráfico, hay un ejemplo de clasificación de elementos con dos características. La mayoría de los algoritmos intentan encontrar el mejor hiperplano de separación (en este caso, es un problema lineal) imponiendo diferentes condiciones.

Sin embargo, el objetivo es siempre el mismo: reducir el número de clasificaciones erróneas y aumentar la robustez del ruido. Por ejemplo, observe el punto triangular que está más cerca del plano (en este caso de la línea) (sus coordenadas son aproximadamente [5.1 - 3.0]). Si la magnitud de la segunda característica se viera afectada por el ruido y el valor fuera bastante menor que 3.0, un hiperplano un poco más alto podría clasificarlo erróneamente.

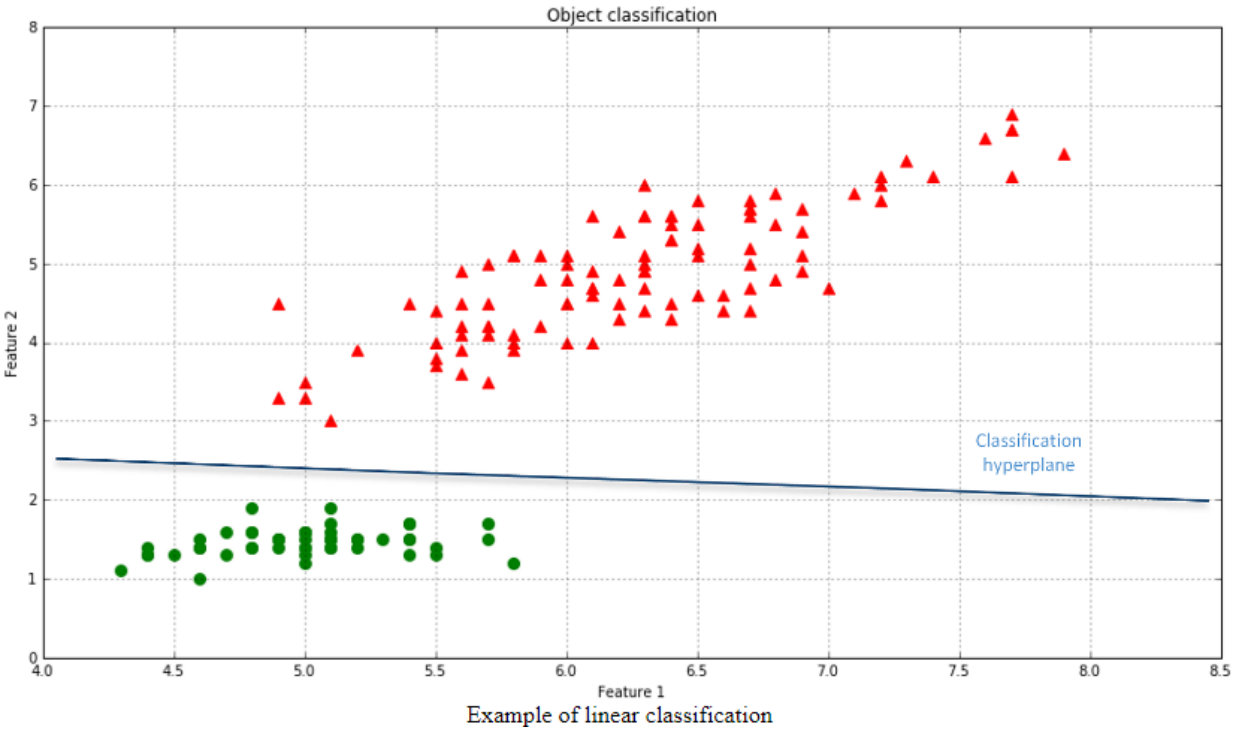


Figure 10 - Ejemplo de clasificación lineal

Aprendizaje no supervisado

Este enfoque se basa en la ausencia de cualquier **supervisor** y, por lo tanto, de medidas de error absoluto. Es útil cuando es necesario aprender cómo se puede agrupar (agrupar) un conjunto de elementos de acuerdo con su similitud (o medida de distancia).

Por ejemplo, mirando el gráfico anterior del algoritmo de clasificación, un ser humano puede identificar inmediatamente dos conjuntos sin considerar los colores o las formas. De hecho, los

puntos circulares (así como los triangulares) determinan un conjunto coherente; a simple vista se pueden visualizar dos conjuntos bien separados, por ejemplo, se puede ver que unos puntos están más separados del otro conjunto que de lo que están separados internamente.

Claramente, el aprendizaje no supervisado proporciona un análisis descriptivo implícito porque todas las piezas de información descubiertas por el algoritmo de agrupamiento se pueden utilizar para obtener una visión completa del conjunto de datos. De hecho, todos los objetos comparten **un subconjunto de características**, estas características son las que hacen posible que este conjunto se pueda agrupar, mientras que por otro lado son diferentes bajo otros puntos de vista o características.

El proceso de agregación también tiene como objetivo extender las características de algunos puntos a sus vecinos, asumiendo que la similitud no se limita a algunas características específicas. Por ejemplo, en un motor de recomendaciones, un grupo de usuarios puede agruparse según la preferencia expresada para algunos libros. Si el criterio elegido detectó algunas analogías entre los usuarios A y B, podemos compartir los elementos que no se superponen entre los usuarios. Por lo tanto, si A ha leído un libro que puede ser adecuado para B, estamos autorizados implícitamente para recomendarlo. En este caso, la decisión se toma considerando un objetivo (compartir las características) y un análisis descriptivo. Sin embargo, como el modelo también puede (y debe) administrar usuarios desconocidos, su propósito también es predictivo.

En el siguiente gráfico, cada elipse representa un grupo y todos los puntos dentro de su área se pueden etiquetar de la misma manera. También hay puntos límite (como los triángulos que se superponen al área del círculo) que necesitan un criterio específico (normalmente una medida de distancia de compensación) para determinar el grupo correspondiente. Al igual que para la clasificación con ambigüedades (P y R malformada), un buen enfoque de agrupamiento debe considerar la presencia de valores atípicos y tratarlos para aumentar tanto la coherencia interna (visualmente, esto significa elegir una subdivisión que maximice la densidad local) como la separación entre grupos.

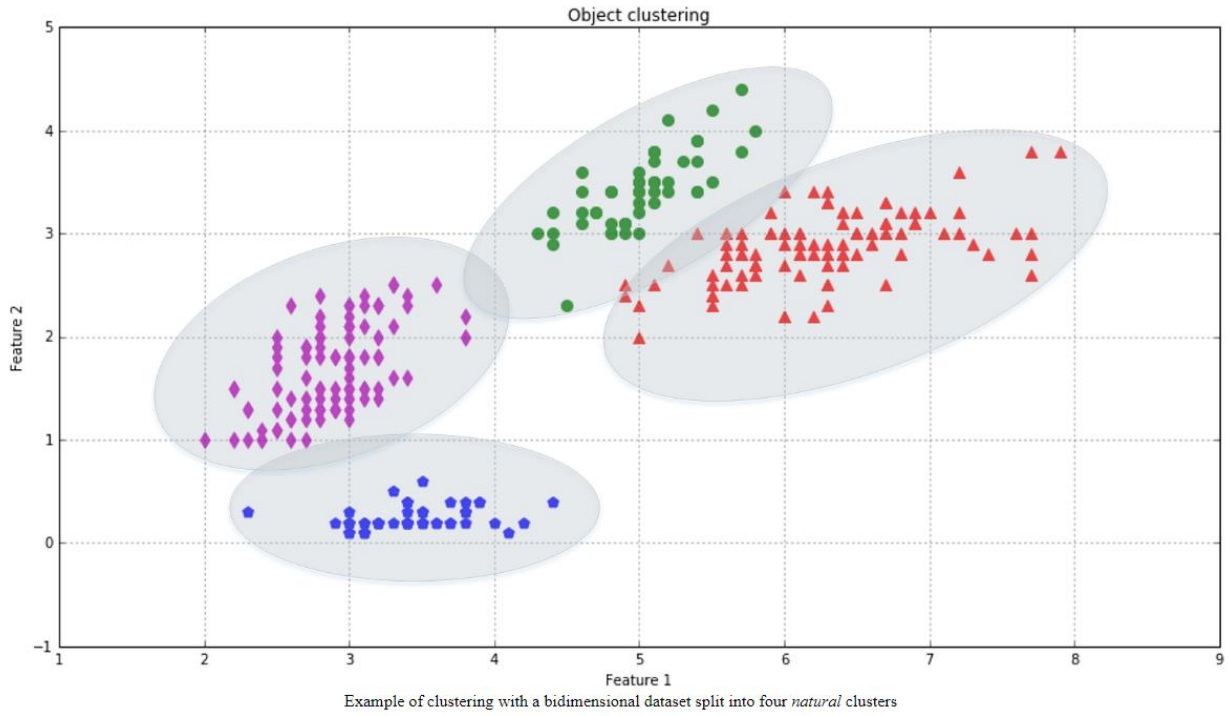


Figure 11 - Ejemplo de clasificación bidimensional

Por ejemplo, es posible dar prioridad a la distancia entre un solo punto y un centroide, o la distancia promedio entre puntos que pertenecen al mismo grupo y a otros diferentes. En este gráfico, todos los triángulos límite están cerca uno del otro, por lo que el vecino más cercano es otro triángulo. Sin embargo, en problemas de la vida real, a menudo hay áreas límite donde hay una superposición parcial, lo que significa que algunos puntos tienen un alto grado de incertidumbre debido a sus valores de características

En este caso, también existen dos modelos:

- **Análisis clúster:** Extrae referencias de conjuntos de datos como parte de los datos de entrada sin respuestas etiquetadas sin conocer nada de cómo se clasifican. Se trata de clasificar un conjunto de datos en grupos lo más homogéneos entre sí y lo más heterogéneos entre ellos. Un ejemplo claro es la segmentación que llevan a cabo las empresas para sus campañas, dividiéndolas por tipo de cliente.

- **Reducción de la dimensionalidad:** Se trata de reducir el número de variables aleatorias mediante la obtención de un conjunto de variables principales, eliminando así variables irrelevantes y mejorando el rendimiento computacional. Se utiliza como método intermedio para reducir la complejidad del sistema. Donde más se utiliza es

en el ámbito de la biología y la medicina, donde existen muchísimas variables y se requiere de la eliminación del máximo posible de ellas.

Aprendizaje semi-supervisado

Existen muchos problemas en los que la cantidad de muestras etiquetadas es muy pequeña en comparación con el número potencial de elementos. Un enfoque supervisado directo es inviable porque los datos utilizados para entrenar el modelo no pueden ser representativos de toda la distribución, por lo tanto, es necesario encontrar un equilibrio entre una estrategia supervisada y una no supervisada. El aprendizaje semi-supervisado se ha estudiado principalmente para resolver este tipo de problemas. El tema es un poco más avanzado y no se tratará en este trabajo de tesis. Sin embargo, los objetivos principales que persigue un enfoque de aprendizaje semi-supervisado son los siguientes:

La propagación de etiquetas a muestras no etiquetadas considerando el gráfico de todo el conjunto de datos. Las muestras con etiquetas se convierten en atractores que extienden su influencia a los vecinos hasta que se alcanza un punto de equilibrio.

Realización de un modelo de entrenamiento de clasificación (en general, Máquinas de vectores de soporte (SVM); utilizando las muestras etiquetadas para hacer cumplir las condiciones necesarias para una buena separación al intentar explotar las muestras no etiquetadas como equilibradores, cuya influencia debe ser mediada por los etiquetados. Los SVM semi-supervisados pueden funcionar extremadamente bien cuando el conjunto de datos contiene solo unas pocas muestras etiquetadas y reducir drásticamente la carga de construir y administrar conjuntos de datos muy grandes.

Reducción de la dimensionalidad no lineal considerando la estructura gráfica del conjunto de datos. Este es uno de los problemas más desafiantes debido a las restricciones existentes en los conjuntos de datos de alta dimensión (es decir, las imágenes). Encontrar una distribución de baja dimensión que represente la distribución original minimizando la discrepancia es una tarea fundamental necesaria para visualizar estructuras con más de tres dimensiones. Además, la capacidad de reducir la dimensionalidad sin una pérdida significativa de información es un elemento clave siempre que sea necesario trabajar con modelos más simples.

Debería quedar claro que el aprendizaje semi-supervisado explota la capacidad de descubrir hiperplanos de separación (clasificación) junto con el autodescubrimiento de relaciones estructurales (agrupamiento). Sin una pérdida de generalidad, podríamos decir que el supervisor real, en este caso, es el gráfico de datos (que representa las relaciones) que corrige las decisiones de acuerdo con la capa de información subyacente. Para comprender mejor la lógica, podemos imaginar que tenemos un conjunto de usuarios, pero solo el 1% de ellos han sido etiquetados (por simplicidad, supongamos que están distribuidos uniformemente). Nuestro objetivo es encontrar las etiquetas más precisas para la parte restante. Un algoritmo de agrupamiento puede reorganizar la estructura de acuerdo con las similitudes (como las muestras etiquetadas son uniformes, podemos esperar encontrar vecinos no etiquetados cuyo centro sea etiquetado). Bajo algunos supuestos, podemos propagar la etiqueta del centro a los vecinos, repitiendo este proceso hasta que cada muestra se estabilice. En este punto, todo el conjunto de datos está etiquetado y es posible emplear otros algoritmos para realizar operaciones específicas. Claramente, esto es solo un ejemplo, pero en la vida real, es extremadamente común encontrar escenarios en los que el costo de etiquetar millones de muestras no esté justificado teniendo en cuenta la precisión lograda por los métodos semi-supervisados.

Análisis descriptivo

Antes de probar cualquier solución de ML, es necesario crear una descripción abstracta del contexto. La mejor manera de lograr este objetivo es definir un modelo matemático, que tenga la ventaja de ser inmediatamente comprensible por cualquiera (asumiendo el conocimiento básico). Sin embargo, el objetivo del análisis descriptivo es encontrar una descripción precisa de los fenómenos que se observan y validar todas las hipótesis.

Tomando como ejemplo el mismo que se implementara más adelante se trata de optimizar la cadena de productos de una gran tienda. Comenzamos a recopilar datos sobre compras y ventas y, después de una discusión con un gerente, definimos las hipótesis genéricas de que el volumen de ventas aumenta durante el día anterior al fin de semana. Esto significa que nuestro modelo debe basarse en una **periodicidad**. Un análisis descriptivo tiene la tarea de validarlo, pero también descubrir todas esas otras **características particulares** que inicialmente se descuidaron.

Al final de esta etapa, deberíamos saber, por ejemplo, si la **serie temporal** (supongamos que consideramos solo una variable) es periódica, si tiene una tendencia, si es posible encontrar un conjunto de reglas estándar, y así adelante. Un paso es definir un modelo de diagnóstico que

debe ser capaz de conectar todos los efectos con causas precisas. Este proceso parece ir en la dirección opuesta, pero su objetivo está muy cerca del análisis descriptivo. De hecho, cada vez que describimos un fenómeno, nos vemos naturalmente obligados a encontrar una razón racional que justifique cada paso específico. Supongamos que, después de haber observado la periodicidad en nuestra serie de tiempo, encontramos una secuencia **que no obedece esta regla**.

El objetivo del análisis de diagnóstico es dar una respuesta adecuada (por ej., la tienda está abierta los domingos). Esta nueva información enriquece nuestro conocimiento y lo especializa: ahora, podemos afirmar que la serie es periódica solo cuando hay un día libre, y por lo tanto no esperamos un aumento en las ventas antes de una jornada laboral. Como muchos modelos de aprendizaje automático tienen requisitos previos específicos, un análisis descriptivo nos permite comprender de inmediato si un modelo funcionará mal o si es la mejor opción teniendo en cuenta todos los factores conocidos.

Análisis predictivo

El objetivo del aprendizaje automático está casi relacionado con esta etapa precisa. De hecho, una vez que hemos definido un modelo de nuestro sistema, necesitamos inferir sus estados futuros, dadas algunas condiciones iniciales. Este proceso se basa en el descubrimiento de las reglas que subyacen al fenómeno para impulsarlas en el tiempo (en el caso de una serie temporal) y observar los resultados. **El objetivo de un modelo predictivo es minimizar el error entre el valor real y el predictivo, considerando todos los posibles factores de interferencia.**

En el ejemplo de la tienda grande, un buen modelo debería ser capaz de pronosticar un pico antes de un día libre y un comportamiento normal en todos los demás casos. Además, una vez que un modelo predictivo ha sido definido y entrenado, puede usarse como parte fundamental de un proceso basado en decisiones. En este caso, la predicción debe convertirse en una receta sugerida. Esta es una tarea común en el aprendizaje por refuerzo, pero también es extremadamente útil cuando un gerente tiene que tomar una decisión en un contexto donde hay muchos factores. **El modelo resultante es, por lo tanto, una tubería que se alimenta con entradas en bruto y utiliza los resultados individuales como entradas para modelos posteriores.**

Por lo tanto, el primer paso es el análisis predictivo, mientras que el segundo es prescriptivo, lo que puede tener en cuenta muchos factores que el modelo anterior descarta (es decir, diferentes proveedores pueden tener tiempos de entrega más cortos o más largos o pueden aplicar descuentos según al volumen). El gerente probablemente definirá un objetivo en términos de una función para maximizar (o minimizar), y el modelo debe encontrar la mejor cantidad de productos para ordenar para cumplir con el requisito principal (que, por supuesto, es la disponibilidad, y depende de la predicción de ventas). para seguir adelante, necesitamos definir por qué el aprendizaje y por qué es tan importante en contextos comerciales cada vez más diferentes.
Como definimos en el primer y segundo capítulo, podemos decir que aprender es la capacidad de cambiar de acuerdo con estímulos externos **(utilizando agentes inteligentes)** y recordar la mayoría de nuestras experiencias anteriores. Por lo tanto, el aprendizaje automático es un enfoque de ingeniería que otorga la máxima importancia a cada técnica que aumenta o mejora la propensión a cambiar de forma adaptativa.

Consiguientemente, el objetivo principal del aprendizaje automático es estudiar, diseñar y mejorar modelos matemáticos que puedan ser entrenados (una vez o continuamente) con datos relacionados con el contexto (proporcionados por un entorno genérico) para inferir el futuro y tomar decisiones sin un conocimiento completo. de todos los elementos influyentes (factores externos**). En otras palabras, un agente (que es una entidad de software que recibe información de un entorno, elige la mejor acción para alcanzar un objetivo específico y observa los resultados)** adopta un enfoque de aprendizaje estadístico, tratando de determinar las distribuciones de probabilidad correctas, y úselos para calcular la acción (valor o decisión) que es más probable que tenga éxito (con la menor cantidad de errores).

Formato de los datos.
Tanto en los problemas de aprendizaje supervisados como no supervisados, siempre habrá un conjunto de datos, definido como un conjunto finito de vectores reales con m características cada uno:

$$X = \{\bar{x}_1, \bar{x}_2, \ldots, \bar{x}_n\} \quad where \;\; \bar{x}_i \in \mathbb{R}^m$$

Teniendo en cuenta que nuestro enfoque siempre es probabilístico, debemos suponer que cada X se extrae de una distribución estadística multivariada, D, que comúnmente se conoce como un proceso generador de datos (la función de densidad de probabilidad a menudo se denota como pdata (x)). Para nuestros propósitos, también es útil agregar una condición muy importante en

todo el conjunto de datos X: esperamos que todas las muestras sean independientes e idénticamente distribuidas (i.i.d). Esto significa que todas las variables pertenecen a la misma distribución, D, y considerando un subconjunto arbitrario de k valores, sucede que lo siguiente es cierto:

$$p(\bar{x}_1, \bar{x}_2, \ldots, \bar{x}_k) = \prod_{i=1}^{k} p(\bar{x}_i)$$

Es fundamental comprender que todas las tareas de aprendizaje automático se basan en la suposición de trabajar con **distribuciones bien definidas** (incluso si pueden ser parcialmente desconocidas), y los conjuntos de datos reales se componen de muestras extraídas de él. Anteriormente definimos el concepto de aprendizaje considerando la interacción entre un agente y una situación desconocida. ¡Esto es posible debido a la capacidad de aprender una representación de la distribución y no del conjunto de datos en sí! Por lo tanto, a partir de ahora, siempre que se emplee un conjunto de datos finito, siempre debemos considerar la posibilidad de hacer frente a nuevas muestras que comparten la misma distribución.

Los valores de salida correspondientes pueden ser numéricos continuos o categóricos. Ejemplos de salidas numéricas son los siguientes:

Cuando la etiqueta puede asumir un número finito de valores (por ejemplo, binario o bipolar), el problema es discreto (también conocido como categórico, teniendo en cuenta que cada etiqueta normalmente está asociada con una clase o categoría bien definida)

Capítulo 4 - Deep Learning

¿Por qué es importante el Deep Learning hoy en día?

Antes de comenzar a hablar de Deep learning, si el lector no está familiarizado con redes neuronales y su importancia en la inteligencia artificial diríjase al anexo 2 de redes neuronales.

Hoy disfrutamos de los beneficios de algoritmos y estrategias que no teníamos hace 20 o 30 años, que nos permiten tener aplicaciones asombrosas que están cambiando vidas. A continuación, mostraremos algunos ejemplos de Deep Learning en la actualidad.

El Deep Learning (o Aprendizaje Profundo) es un método del Machine Learning que nos permite entrenar una Inteligencia Artificial para obtener una predicción dado un conjunto de entradas. Esta inteligencia logrará un nivel de cognición por jerarquías. Se puede utilizar Aprendizaje Supervisado o No Supervisado.

Entrenamiento en mini lotes (**Training in mini-batches**)**:** Esta estrategia nos permite hoy tener conjuntos de datos muy grandes y entrenar un modelo de aprendizaje profundo poco a poco. En el pasado, teníamos que cargar todo el conjunto de datos en la memoria, lo que lo hacía imposible computacionalmente para algunos conjuntos de datos grandes. Hoy, sí, puede llevar un poco más de tiempo, pero al menos podemos realizar el entrenamiento en un tiempo finito.
Funciones de activación novedosas: las unidades lineales rectificadas (ReLU), por ejemplo, son un tipo de activación relativamente nuevo que resolvió muchos de los problemas del entrenamiento a gran escala con estrategias de retropropagación. Estas nuevas activaciones permiten que los algoritmos de entrenamiento converjan en arquitecturas profundas cuando, en el pasado, nos quedábamos atascados en sesiones de entrenamiento no convergentes que terminarían teniendo gradientes explosivos o que desaparecían.
Nuevas arquitecturas de redes neuronales (Novel activation functions: **Rectified linear units** (**ReLUs**))**:** las redes convolucionales o recurrentes, por ejemplo, han estado transformando el mundo al abrir las posibilidades de cosas que podemos hacer con las redes neuronales. Las redes convolucionales se aplican ampliamente en aplicaciones de visión por computadora u otras áreas en las que la operación de convolución es algo natural, por ejemplo, análisis multidimensional de señales o audio. Las redes neuronales recurrentes con memoria se utilizan

ampliamente para analizar secuencias de texto, lo que nos permite tener redes que entienden palabras, oraciones y párrafos, y podemos usarlas para traducir entre idiomas y muchas más cosas.

Funciones de pérdida interesantes (Interesting loss functions): estas pérdidas juegan un papel interesante en el aprendizaje profundo porque, en el pasado, solo usábamos las mismas pérdidas estándar una y otra vez; pérdidas como el MSE. Hoy, podemos minimizar el MSE y, al mismo tiempo, minimizar la norma de los pesos o la salida de algunas neuronas, lo que conduce a pesos y soluciones más dispersos que, a su vez, hacen que el modelo producido sea mucho más eficiente cuando se implementa. en producción.

Estrategias novedosas que se asemejan a la biología (Novel strategies resembling biology)**:** cosas como la falta o la eliminación de conexiones entre neuronas, en lugar de tenerlas completamente conectadas todo el tiempo, es más realista o comparable al diseño de redes neuronales biológicas. Además, eliminar o eliminar neuronas por completo es una nueva estrategia que puede impulsar a algunas neuronas a sobresalir cuando se eliminan otras, aprendiendo representaciones más ricas y, al mismo tiempo, reduce los cálculos durante el entrenamiento y cuando se despliegan. El intercambio de parámetros entre redes neuronales diferentes y especializadas también ha demostrado ser interesante y efectivo en la actualidad.

Entrenamiento adversario: hacer que una red neuronal compita contra otra red cuyo único propósito es generar puntos de datos fraudulentos, ruidosos y confusos que intentan hacer que la red falle, ha demostrado ser una excelente estrategia para que las redes aprendan mejor de los datos y sean robustas contra entornos ruidosos cuando se implementa en producción.

Hay muchos otros hechos y puntos interesantes que hacen del aprendizaje profundo un área interesante y justifican la redacción de este libro. Espero que esté tan emocionado como todos nosotros y comience a leer este libro sabiendo que vamos a codificar algunas de las redes neuronales más emocionantes e increíbles de nuestro tiempo. Nuestro objetivo final será crear redes neuronales profundas que se puedan generalizar.

Como podemos visualizar en la figura 14 el Deep learning o aprendizaje profundo es un grupo de sub-algoritmos dentro del campo del machine learning.

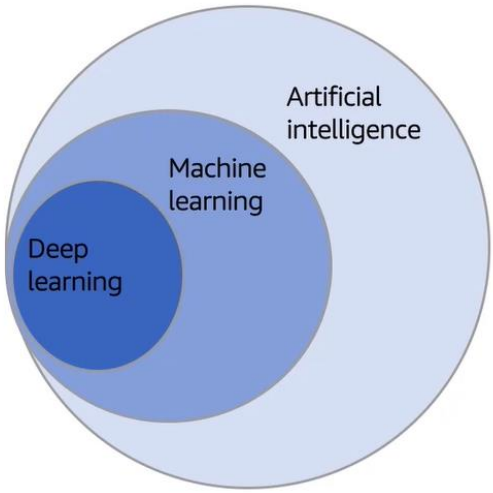


Figure 12 - Clasificaciones de algoritmos para deeplearning

Definición de redes feedforward

Las redes de retroalimentación profunda, también llamadas redes neuronales de retroalimentación, a veces también se denominan perceptrones multicapa (MLP).
El objetivo de una red feedforward es aproximar la función de $f*$. Por ejemplo, para un clasificador, $y = f * (x)$ asigna una entrada x a una etiqueta y. Una red feedforward define un mapeo desde la entrada hasta la etiqueta $y = f (x; \theta)$. Aprende el valor del parámetro θ que resulta en la mejor aproximación de función.

Redes neuronales recurrentes. Las redes de retroalimentación son un trampolín conceptual en el camino hacia las redes recurrentes, **que impulsan muchas aplicaciones del lenguaje natural**. Las redes neuronales feedforward se denominan redes porque componen juntas muchas funciones diferentes que las representan. Estas funciones están compuestas en un grafo acíclico dirigido.

El modelo está asociado con un grafo acíclico dirigido que describe cómo se componen las funciones juntas. Por ejemplo, hay tres funciones f(1), f(2) y f(3) conectadas para formar f(x) = f(3) (f(2) (f(1) (x))). Estas estructuras de cadena son las estructuras de redes neuronales más utilizadas. En este caso, f(1) se denomina primera capa de la red, f(2) se denomina segunda capa, y así sucesivamente. La longitud total de la cadena da la profundidad del modelo. **De esta terminología surge el nombre de aprendizaje profundo o Deep learning en inglés**. La capa

final de una red feedforward se llama capa de salida. Estas redes se denominan neuronales porque están inspiradas en la neurociencia. Cada capa oculta es un vector. La dimensionalidad de estas capas ocultas determina el ancho del modelo.

Se les llama así a las redes en que las salidas de una capa son utilizadas como entradas en la próxima capa. Esto quiere decir que no hay loops "hacia atrás". Siempre se "alimenta" de valores hacia adelante. Hay redes en las que sí que existen esos loops (Recurrente Neural Networks) estas redes están fuera del alcance de este trabajo de tesis.
Además, existe el concepto de "fully connected Feedforward Networks" y se refiere a que todas las neuronas de entrada, están conectadas con todas las neuronas de la siguiente capa.

Redes neuronales y la back propagación:

Cuando se utiliza una red neuronal de retroalimentación para aceptar una entrada x y producir una salida y, la información fluye hacia adelante a través de los elementos de la red. La entrada x proporciona la información que luego se propaga hasta las unidades ocultas en cada capa y produce $\hat{y}$. A esto se le llama propagación hacia adelante, como vimos anteriormente en las redes feedforward.
Durante el entrenamiento, la propagación hacia adelante continúa hasta que produce un costo escalar $J(\theta)$. El algoritmo de retropropagación, a menudo llamado backprop, permite que la información del costo fluya **hacia atrás a través de la red para calcular el gradiente**.

En el capítulo de machine learning vimos como a partir de los datos podemos entrenar a nuestro modelo, la estrategia es simple, es buscar a partir del método de mínimos cuadrados ordinarios obtener una fórmula que nos diera exactamente el punto mínimo de la función de coste. Pero este método está limitado a ese modelo y a esa función de coste, para modificar esto vamos a utilizar el método del descenso del gradiente.

A partir de los datos, podemos visualizar cual es el error que tiene nuestro modelo, y este está dado por la función de coste, que nos dará el error para cada una de las combinaciones de datos que utilicemos como parámetros. Como dijimos esta función de coste está asociada a nuestros datos y nos dará el error. Para obtener el error mínimo debemos encontrar los mínimos de esa función de coste, matemáticamente debemos obtener la derivada de la función y donde la misma es cero obtendremos los mínimos locales o un mínimo global de la función.

En regresión lineal las funciones son todas convexas y el mínimo de esta función siempre será único. En las funciones no convexas debemos encontrar una manera de obtener el mínimo global a estas funciones y es aquí donde el algoritmo del descenso de la gradiente ayuda.
Debemos realizar las derivadas parciales para cada uno de los parámetros en nuestro modelo, y obtenemos el vector gradiente, utilizando este vector podemos movernos hacía el punto mínimo de la función, la velocidad en la que nos movemos es la ratio de aprendizaje y es muy importante para que la optimización con la que encontremos el mínimo converja en el punto.
Este algoritmo se encuentra en el corazón de casi todos los algoritmos de machine learning, y es un componente fundamental en el campo de redes neuronales.

Calcular una expresión analítica para el gradiente es sencillo, pero evaluar numéricamente tal expresión puede resultar computacionalmente costoso. El algoritmo de retropropagación lo hace mediante un procedimiento simple y económico.

El algoritmo de retropropagación se introdujo originalmente en la década de 1970, pero su importancia no se apreció por completo hasta un famoso artículo de 1986 de David Rumelhart, Geoffrey Hinton y Ronald Williams. existen varias redes neuronales en las que la propagación hacia atrás funciona mucho más rápida que los enfoques anteriores de aprendizaje, lo que hace posible el uso de redes neuronales para resolver problemas que antes no tenían solución. Hoy en día, el algoritmo de retro propagación es el caballo de batalla del aprendizaje en redes neuronales y Deep learning.

En el corazón de la propagación inversa se encuentra una expresión para la derivada parcial $\partial C / \partial w$ de la función de costo C con respecto a cualquier peso w (o sesgo b) en la red. La expresión nos dice qué tan rápido cambia el costo cuando cambiamos los pesos y los sesgos. Y aunque la expresión es algo compleja, también tiene una belleza, y cada elemento tiene una interpretación natural e intuitiva. Entonces, la propagación hacia atrás no es solo un algoritmo rápido para el aprendizaje. De hecho, nos brinda información detallada sobre cómo cambiar los pesos y los sesgos cambia el comportamiento general de la red. Vale la pena estudiarlo en detalle.

Gracias al algoritmo de backpropagation se hizo posible entrenar redes neuronales de múltiples capas de manera supervisada. Al calcular el error obtenido en la salida e ir propagando hacia las capas anteriores se van haciendo ajustes pequeños (minimizando costo) en cada iteración para

lograr que la red aprenda consiguiendo que la red pueda -por ejemplo- clasificar las entradas correctamente.

Capítulo 5 – Implementación del sistema de logística de productos

El sistema implementado dará soporte al área de logística y distribución de la empresa ficticia **Your Choice SA**, la misma cuenta con un sistema de ventas en cada punto venta (sucursal) distribuidos por el mundo. En nuestro sistema utilizaremos un punto en cada país.

El proceso de ventas se realiza en cada punto de venta, y un cliente puede solicitar un producto de todo el catálogo de productos disponibles por la empresa. Se ha obtenido la nómina de productos de *the Observatory of econmic complexity* [4], por el momento solo se tomarán los 100 primeros productos para hacer las pruebas.

Un concepto fundamental que debemos tener presente dentro del sistema que se ha desarrollado es la **Intención de compra,** éste hace referencia al momento en que un cliente o usuario entra a un punto de venta y solicita un producto. La intención de compra será satisfactoria si el producto se encuentra en el stock de ese punto de venta (y por lo tanto se realizará la venta) o es **fallida si ocurre lo contrario**. La **intención de compra** es un punto importante a tener cuenta para la compra de nuevos productos o el movimiento de productos desde otra sucursal para ese punto de venta, este concepto es el que vamos a tener en cuenta al hacer las predicciones.

La distribución inicial de productos se realiza de forma aleatoria, a medida que se realiza el proceso de ventas y compras este ira cambiando.

Mientras que se van realizando las distintas las ventas y se van computando todas las intenciones de compra en cada punto de venta, se realizan distintos procesamientos de información y el sistema con un algoritmo de machine learning realizará la predicción de la cantidad de un producto que se necesitará a futuro para poder afrontar las ventas, este proceso asiste a los usuarios de logística de manera eficiente para la compra o movimiento de productos.

[4] El Observatorio de Complejidad Económica es una herramienta que permite a los usuarios componer rápidamente una narrativa visual sobre los países y los productos que intercambian. Fue la Tesis de Maestría de Alexander Simoes en Artes y Ciencias de los Medios en el Laboratorio de Medios del MIT.
El proyecto se realizó en el grupo de MIT Media Lab Macro Connections (ahora aprendizaje colectivo). El asesor de Alex fue César A. Hidalgo, investigador principal de Macro Connections. Desde su creación en 2010, el desarrollo del Observatorio de Complejidad Económica ha sido apoyado por los consorcios de The MIT Media Lab para la investigación no dirigida.

Arquitectura de la solución

Como se mencionó en el capítulo 2, es sumamente importante tener una arquitectura que se adapte a las necesidades del sistema, dando la posibilidad de hacer mejores integraciones con los diferentes módulos, posibilitando que el sistema pueda escalar y dar un mantenimiento simple, ahorrando en recursos y en tiempo.

La solución que se propone en este trabajo tiene la siguiente arquitectura de software, como podemos ver en la figura 13, lo primero que podemos observar que el sistema está separado en dos microservicios principales, uno es el encargado de proveer toda la funcionalidad relacionada con el manejo de productos y el otro microservicio está encargado de suministrar la funcionalidad para realizar las ventas e intenciones de compras, cada uno de los microservicios posee los endpoints y servicios relacionados con una parte de la solución total del sistema. De esta manera, si uno de los servicios falla, él otro microservicio puede continuar operando.

- **Product Management Service**: Este microservicio posee todos los servicios para el manejo de productos.
- **Portal de Ventas Service**: posee aquellos servicios que se utilizaran para el portal de ventas.

Como se puede visualizar en la figura 13, al tener un sistema basados en microservicios, las distintas interfaces (UI) de usuario, pueden consumir datos de ella, por ejemplo, podemos ver como la UI de manejo de productos y la UI de proceso de emulación de ventas e intenciones de compras consumen datos del mismo servicio (producto management)

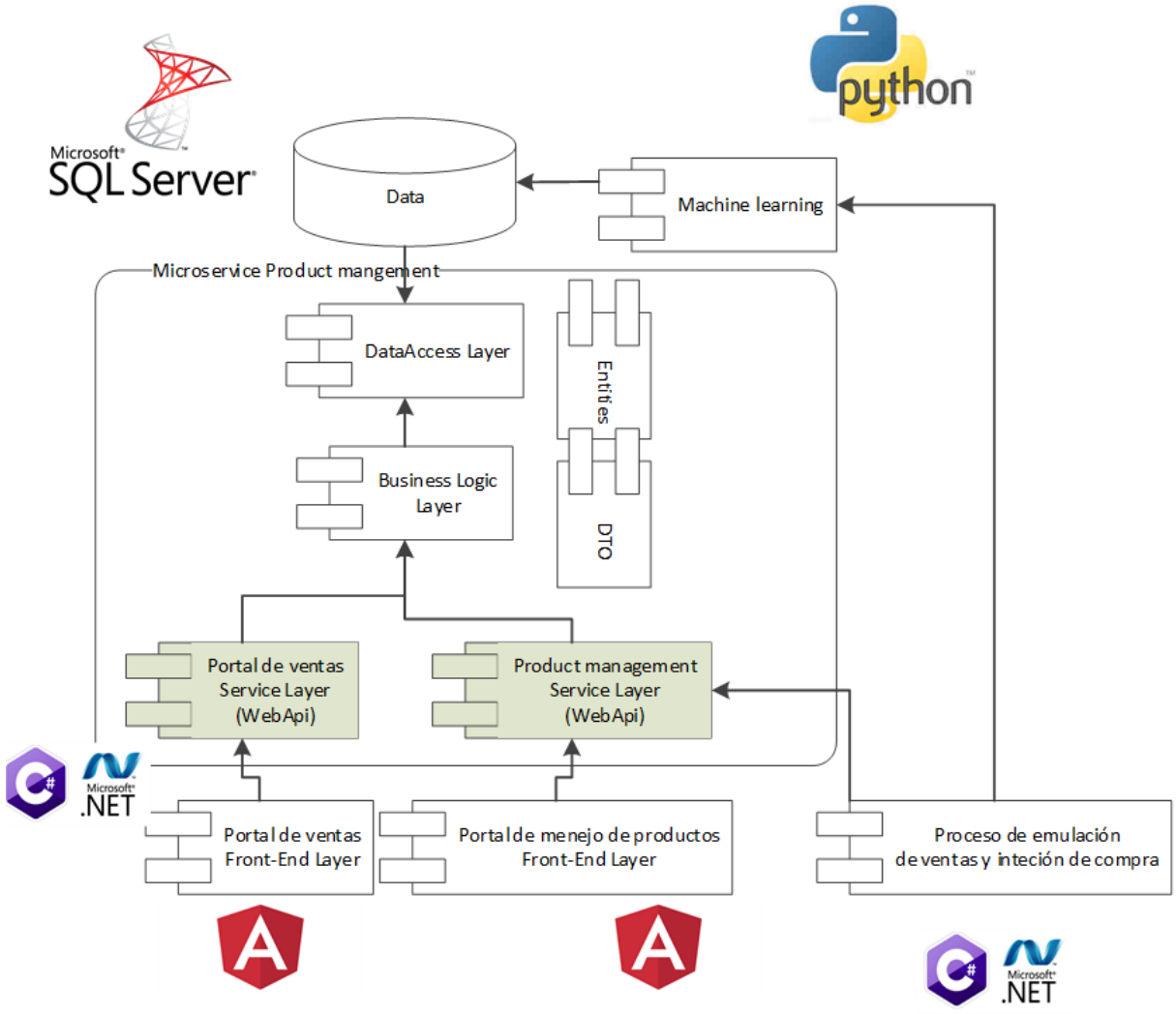


Figure 13 - Arquitectura general del sistema

Por otro lado, como se puede visualizar en la figura 13 el backend del sistema fue dividido en N capas, esto permite que cada capa pueda ser desarrollada por separado y pueda ser modificada o remplazada de manera que el resto de la aplicación no tenga consecuencias o modificaciones, mientras se respeten sus interfaces, y no exista acoplamiento entre capas. Una de las herramientas que se implementó en el sistema para evitar el acoplamiento es la utilización de los patrones de IoC (inversión de control) y de DI (inyección de dependencias) que se explicaron en el capítulo 2.

Por ejemplo, en el backend del sistema utilizamos la inyección de dependencia para inyectar los repositorios a los objetos de lógica negocios, lo hacemos por medio del constructor de la clase.

En la figura 14 podemos visualizar como la clase **ProductsBL** la cual posee las reglas de negocio para los productos, tiene varias instancias de repositorios las cuales son inyectadas en el constructor al momento de crear una instancia de esta clase. De esta manera no existe acoplamiento entre los repositorios y la clase de negocio.

```
    public class ProductsBL: IProductsBL
     {
         private readonly IRepository<Product> repositoryProduct;
         private readonly IRepository<ProductType> repositoryProductType;
         private readonly IRepository<Stock> repositoryStock;
         private readonly IRepository<Country> repositoryCountry;
         private readonly IRepository<SalePoint> repositorySalePoint;

         public ProductsBL(IRepository<Product> _repository,
             IRepository<ProductType> _repositoryProductType,
             IRepository<Stock> _repositoryStock,
             IRepository<Country> _repositoryCountry,
             IRepository<SalePoint> _repositorySalePoint)
         {
             repositoryProduct = _repository;
             repositoryProductType = _repositoryProductType;
             repositoryStock = _repositoryStock;
             repositoryCountry = _repositoryCountry;
             repositorySalePoint = _repositorySalePoint;
         }

 }
```

Figure 14 - Inyección de dependencias en productos

Dentro de la solución del sistema existe un proyecto especial en donde se especifican todas dependencias y como deben ser resueltas utilizando DI, especificando al contenedor de clases cuales son las instancias específicas que se deben instanciar para cada una de las interfaces.
En este proyecto podemos encontrar dos clases, la primera `DataAccessDIResolution` es la que está encargada de especificar los repositorios mientras que la clase `BusinessLogicDIResolution` especifica las clases de negocio las cuales se inyectaran en los controladores para cada uno de los servicios.
AspNet Core posee las implementaciones correspondientes como parte del mismo framework para poder utilizar IoC y DI como parte del pipeline de ejecución del mismo.

```
public static class DataAccessDIResolution
{
    public static void AddDataAccessDependencies(this IServiceCollection services, string connectionString)
    {
        services.AddScoped<ProductsDBContext>(_ => new ProductsDBContext(connectionString));
        services.AddTransient<IRepository<Product>, Repository<Product>>();
        services.AddTransient<IRepository<ProductType>, Repository<ProductType>>();
        services.AddTransient<IRepository<Sale>, Repository<Sale>>();
        services.AddTransient<IRepository<User>, Repository<User>>();
        services.AddTransient<IRepository<Stock>, Repository<Stock>>();
        services.AddTransient<IRepository<Country>, Repository<Country>>();
        services.AddTransient<IRepository<SalePoint>, Repository<SalePoint>>();
        services.AddTransient<IRepository<Search>, Repository<Search>>();
```

Figure 15 - Clase de resolución de dependencias

Como vemos en la figura 15 se especifica al contenedor de IoC que tipo de clase se realizará para un tipo de interfaz con su clase genérica incluida. Por ejemplo, para la interfaz IRepository<Product> se obtendrá en tiempo de ejecución una instancia de Repository<Product>.

En la implementación del **backend** se utilizó la plataforma AspNet Core 2.1 y 3.0 y se desarrolló una aplicación basada en capas como se detalla a continuación.

- **Capa de acceso a datos**: En esta capa se utilizó el ORM Entity Framework para aspnet core, el cual permite al sistema interactuar con una base de datos relacional utilizando objetos y un lenguaje de consultas como linq.

- **Capa de lógica de negocio:** se implementó esta capa para poder independizar las tareas de negocio del acceso a base datos y de la implementación de los servicios.

- **Capa de servicios:** Esta capa expone la funcionalidad con servicios rest.

Frontend:

Para la implementación del frontend se utilizó Angular versión 11.0, el cual se especifico algunas de sus features en el capítulo 2, por otro lado se utilizó plotly para la visualización de gráficos dentro del sistema.

Machine Learning:

Este módulo de sistema realiza la ejecución del notebook de Python basado en Deep learning con keras y guarda en base de datos las predicciones.

Base de Datos:

El modelo se implementará sobre una base de datos SQL Server 2014+.

Control de código fuente:

La aplicación y toda su implementación, así como datos de prueba y todo lo necesario para su ejecución se encuentra disponible a través de un repository de github público en la siguiente ubicación:

git clone https://github.com/ijoakin/tesis_ignacio_joakin.git

A continuación, se detallarán cada uno de los componentes de la solución y su utilización.

1. Portal de manejo de productos
2. Portal de ventas
3. Microservicios
4. Módulo de simulación de ventas y Machine Learning
5. Módulo de procesamiento Machine Learning
6. Modelo de Base de datos

Portal de manejo de productos:

El portal de manejo de productos es utilizado por el usuario final para el manejo y visualización de toda la información relacionada con productos, cantidades en stock, sucursales, etc.
En la misma el usuario no solo podrá manejar la administración de productos, sino que posee una herramienta para el manejo de la base de datos

Las aplicaciones web están implementadas en **Angular 11** podemos realizar la compilación de las mismas en cualquier servidor web como apache, IIS u otro similar. El sitio web de manejo de productos, por ejemplo, se encuentra compilado en la siguiente ubicación del repositorio.

products\angular-client\dist

En el caso de que no se quiera ejecutar la versión de distribución por la falta de un servidor web, se puede iniciar las pruebas desde el sistema compilando (transpiling) el sitio desde el código fuente y servirlo con el comando propio de Angular-CLI.
Pre requisitos para compilar el código fuente.

NodeJs versión 14 o superior.

Antes de poder ejecutar la compilación verifique que las herramientas de angular-cli estén instaladas, en el caso de que no lo estén se puede utilizar el siguiente comando.

npm install -g @angular/cli

Utilizando la línea de comando de **angular-cli** en la carpeta del cliente angular: **products\angular-client**

Una vez instaladas las herramientas anteriormente detalladas, utilizar los siguientes comandos para su compilación.

npm install
ng serve -o

Este comando compilará la aplicación y abrirá por nosotros el navegador por defecto con la dirección **http://localhost:4200/,** en la cual ya podremos interactuar con la interfaz de usuario web del sistema.

Hoy en día las aplicaciones web UI, realizadas con frameworks actuales como Angular o ReactJs poseen una complejidad más alta que en la antigüedad, las mismas también suelen separarse en capas e implementar ciertos patrones como Model View View Model (MVVM) o Model view controller (MVC), por ejemplo, en las aplicaciones web incluidas en este trabajo se han dividido en capas como el framework de Angular lo sugiere y es acorde a su arquitectura (capitulo 2).

Como podemos ver en la figura 16, existe una capa para los componentes, otra para el modelo y otra para interactuar con los servicios. La implementación también requiere de inyección de dependencias, por lo tanto, cada componente tiene inyectado al momento de su creación los servicios que se van a utilizar dentro de ese componente facilitando así el testing de la aplicación al momento de realizar un mock de los servicios.

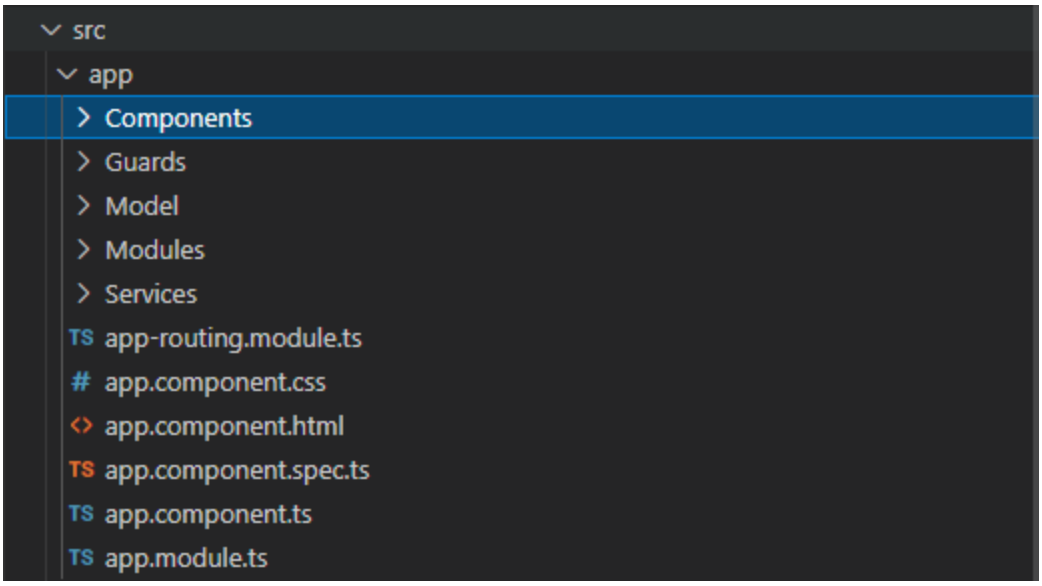


Figure 16 - Capas de las aplicaciones web (UI)

Una vez que la aplicación web se encuentra funcionando correctamente el usuario puede acceder a los distintos módulos por medio de un menú principal como se visualiza en la imagen 17.

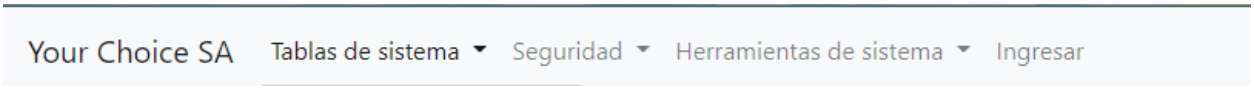


Figure 17 - Menú del sistema

Tablas de Sistema:

En este ítem del menú se puede acceder a cada una de las tablas del sistema visualizando cada uno de los elementos, como ventas, productos, stock, intenciones de compra etc., pudiendo realizar las acciones elementas de CRUD para cada elemento de la base de datos, en la siguiente imagen podemos visualizar los distintos puntos de este menú.

Figure 18 - Menú Productos

Tomaremos *Productos* de las tablas del sistema o tablas paramétricas a modo de ejemplo y se mostrará a continuación como el usuario podría manipular de forma manual la base de datos, para lo mismo debe entrar al tipo de entidad que desea modificar del menú anterior (en nuestro caso accederemos a Productos) y se visualizará una grilla con los datos como se muestra a continuación (figura 19).

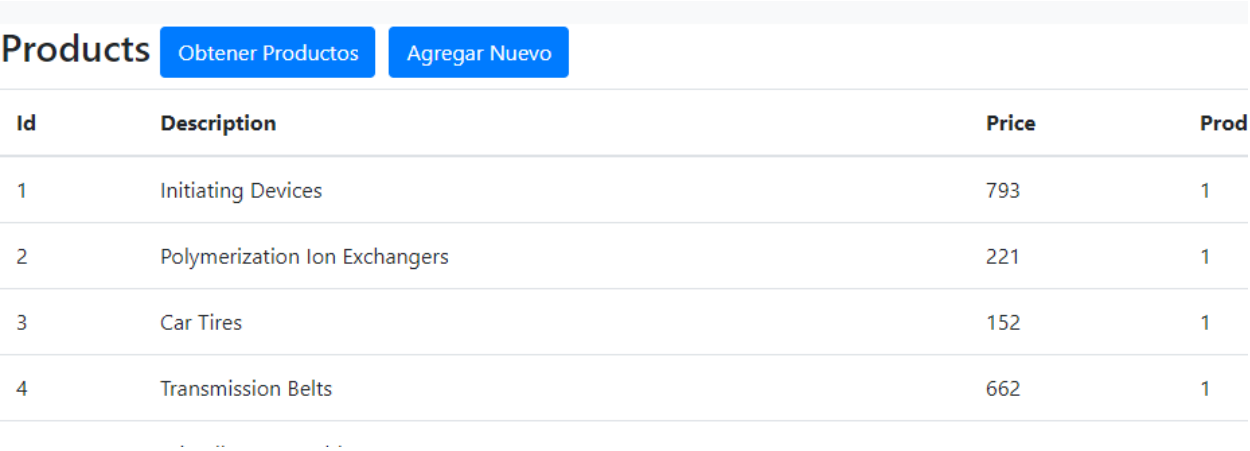


Figure 19 - Grilla de información

En la parte superior de la grilla encontraremos los botones para obtener los productos (refrescará los datos al presionarlo) y el botón para agregar un nuevo producto a la base de datos.

Debajo de la grilla se cuenta con un conjunto de herramientas para el paginado el cual permite navegar por los ítems de 10 en 10, pudiendo acceder a todos los datos

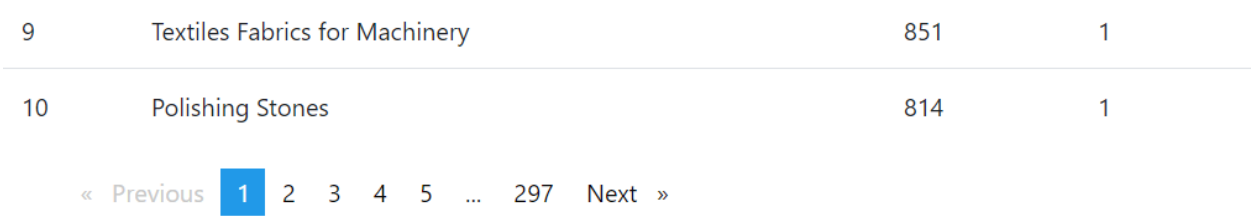


Figure 20 - Paginación del sistema

En el caso que se desee agregar un nuevo producto, editar o eliminar uno ya existente el usuario cuenta con las herramientas para hacerlo, por ejemplo, al presionar el botón con forma de “ojo” en la columna de comandos aparecerá una ventana desplegable con los datos de del producto que desea editar, al presionar el botón con forma de cruz podrá eliminar o así también podrá agregar un nuevo producto con el botón debajo de la grilla “Agregar nuevo”

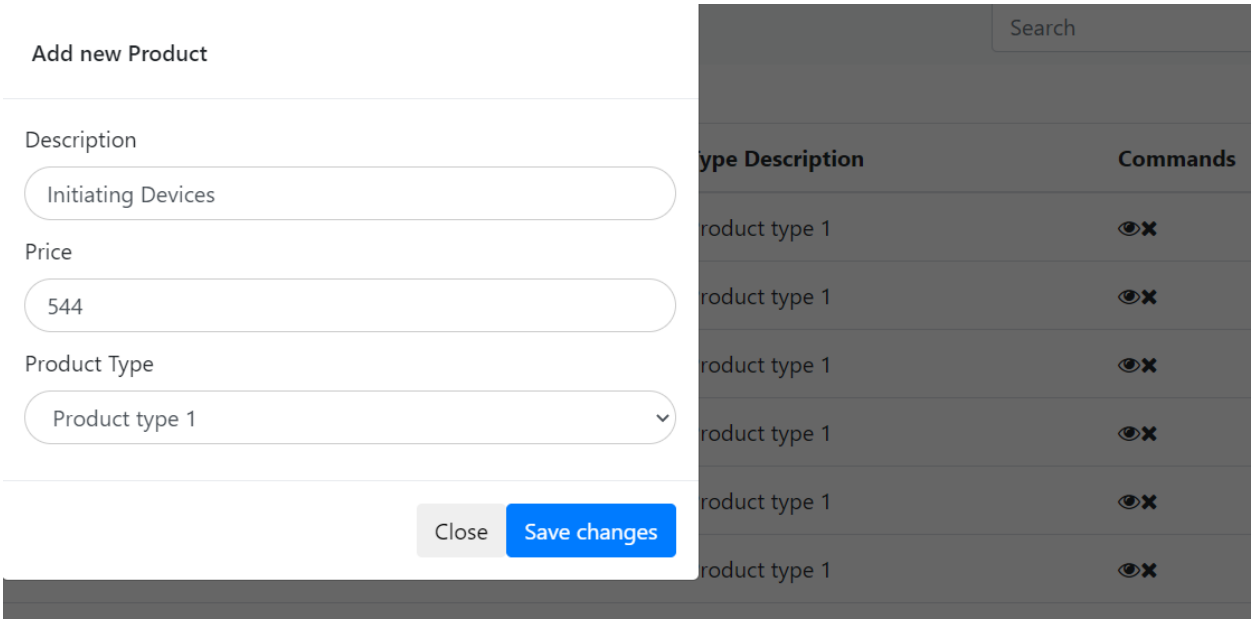
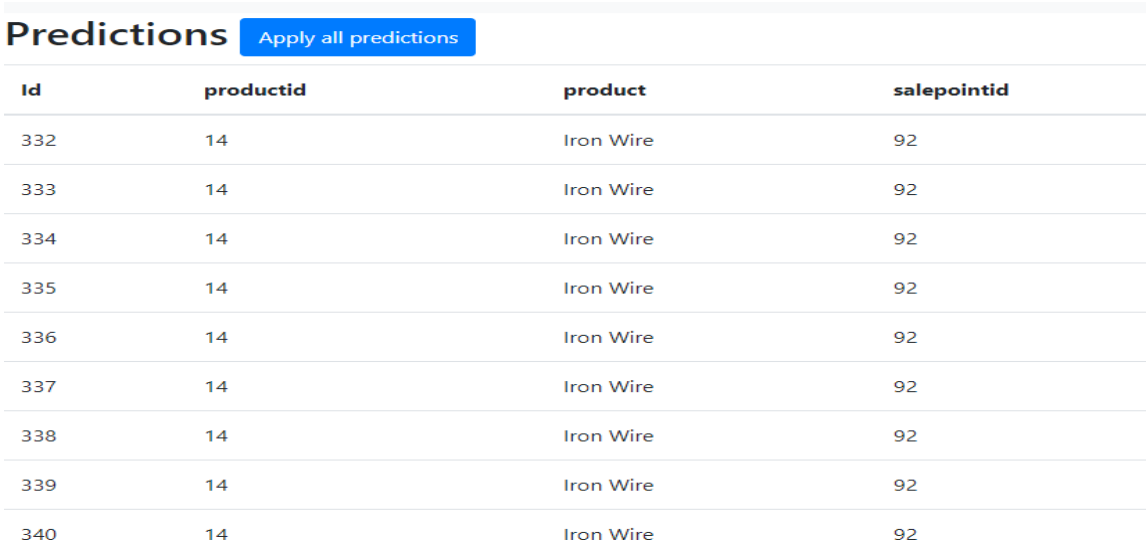


Figure 21 - Agregar nuevo registro

Prácticamente el resto de las tablas paramétricas como países, stock y ventas se manejan de la misma manera, visualizando la grilla y luego realizando las acciones sobre cada uno de los ítems. La página predicciones, que se encuentra en este mismo menú, es un poco diferente al resto. Las predicciones son generadas por el notebook python del módulo de machine learning, una

vez generadas el usuario puede decidir si aplicarlas o no desde la UI, esta UI permitirá al sistema comprar nuevos productos. Para lo cual debe acceder al ítem del menú predicciones, una vez en la grilla el usuario podrá aplicar las predicciones presionando el botón en la parte superior de la pantalla como se visualiza en la figura 20



Id	productid	product	salepointid
332	14	Iron Wire	92
333	14	Iron Wire	92
334	14	Iron Wire	92
335	14	Iron Wire	92
336	14	Iron Wire	92
337	14	Iron Wire	92
338	14	Iron Wire	92
339	14	Iron Wire	92
340	14	Iron Wire	92

Figure 22 - Tabla de predicciones

Herramientas de base de datos.

Como se mencionó, se dispone de una pantalla con la cual se podrá interactuar con la base de datos desde la aplicación web, a la misma se puede acceder desde el menú **>Herramientas de sistema>Herramientas de base de datos.**

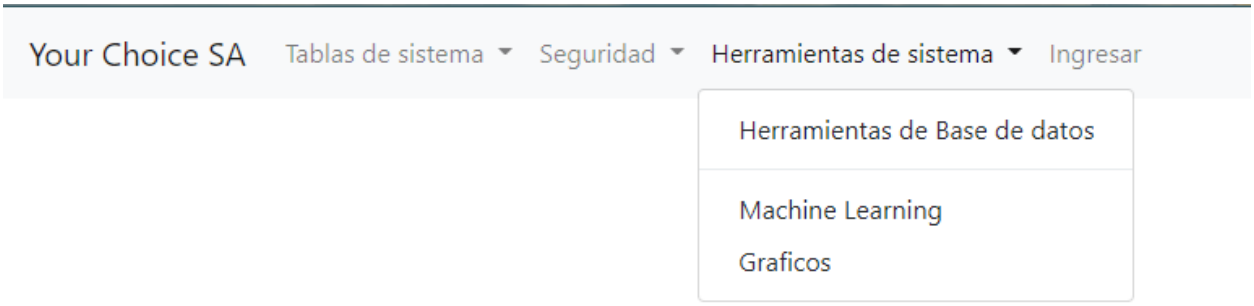


Figure 23 - menú de Herramientas del sistema

En esta pantalla se podrán realizar las siguientes acciones

- *Inicializar la base de datos desde cero*, pudiendo crear las tablas, agregar el catálogo de productos, los países con sus atributos.
    - Este punto permite iniciar la aplicación y las simulaciones desde cero.
- *Inicializar en los puntos de venta y el stock de forma aleatoria*.
- *Simular el proceso de ventas para una determinada cantidad*
- *Simular el proceso de búsquedas sobre productos*.
- *Simular el proceso entero realizando ventas y compras*, es decir, cuando un cliente entra a un punto de venta y solicita un producto (el cual puede o no estar en el stock de ese punto de venta) generando una intención de compra y realizando la venta en el caso de que si se encuentre ese producto en ese punto de venta.

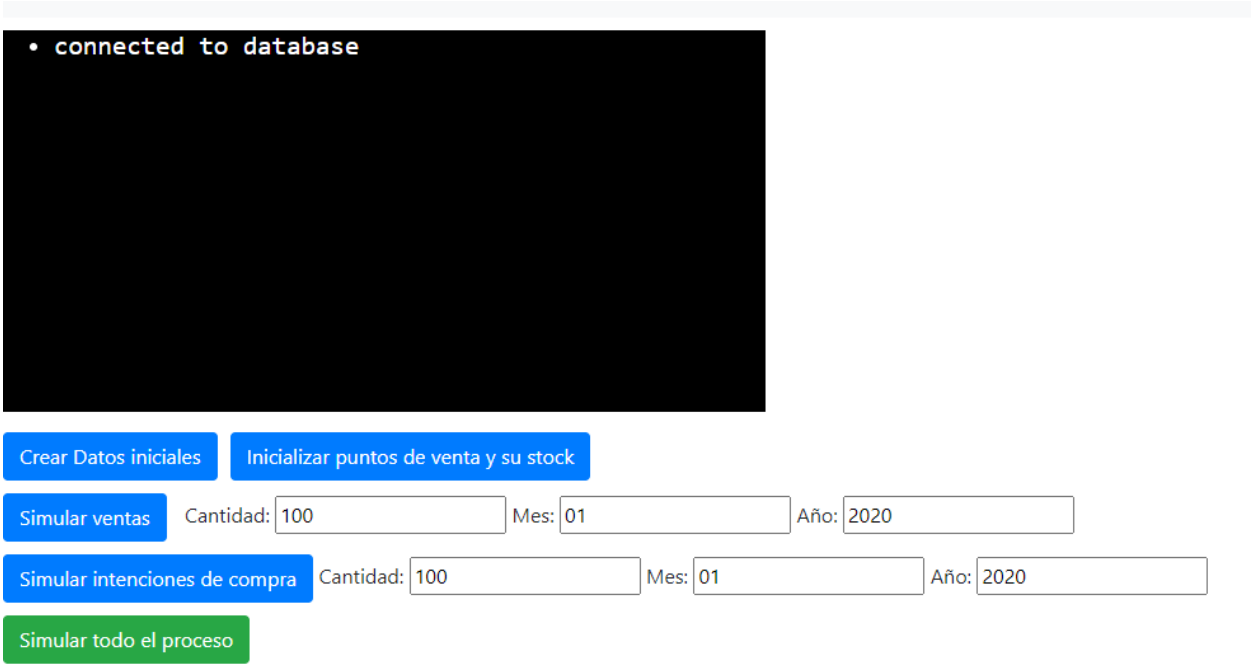


Figure 24 - Herramientas de base de datos

Al ejecutar la creación de datos iniciales, el sistema ubicará los archivos de migración (script SQL para realizar la carga inicial de datos), estos son el catálogo de productos, categorías para los países, tipos de productos etc.

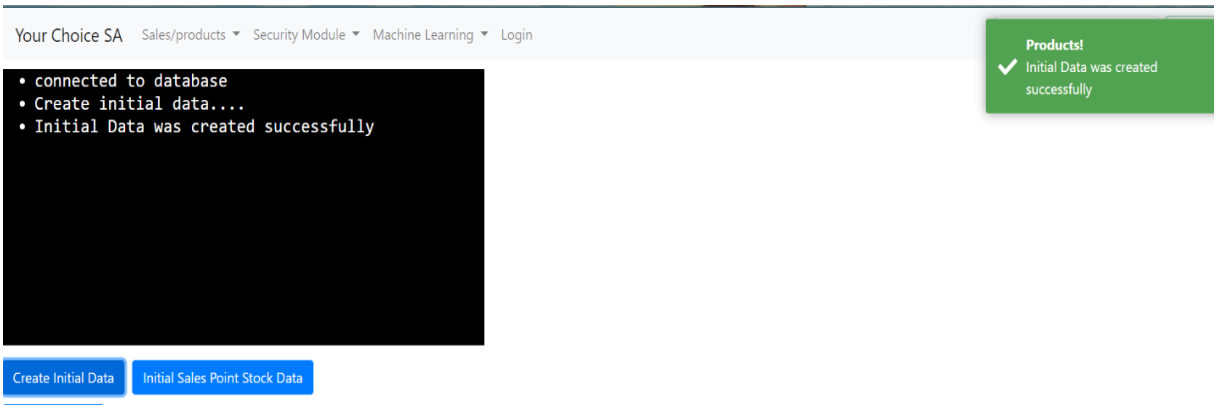


Figure 25 - Ejecución de inicialización de la base de datos

Una vez realizada la creación inicial de la base de datos, se necesitará inicializar el stock y puntos de ventas, en el cual se le realizará una distribución aleatoria de los productos.
Poseemos en esta página tres botones de simulación, con los cuales podemos realizar una simulación de ventas, intenciones de compra o simular todo el proceso.

Portal de ventas

El portal de ventas se realizó con la misma tecnología que el portal de manejo de productos, se implementó por separado con la intención de que sea una aplicación autónoma, la misma consume su información de un microservicio separado del manejo de productos, esto nos permite realizar un despliegue de la aplicación por separado, permitiendo escalabilidad y balance de carga.

En la figura 25 se visualiza el portal de ventas, al ingresar, el usuario debe cargar en el campo búsqueda de productos el nombre del producto, luego debe seleccionar la sucursal donde desea comprarlo. Al presionar el botón buscar el sistema buscará el producto a vender. En el caso de que se encuentre stock aparecerá la cantidad disponible en pantalla y el usuario puede realizar la compra.

Por otro lado, en el caso de que no haya stock disponible aparecerá la leyenda en rojo (En este momento no se posee stock en esta sucursal). Este será el momento en el cual el sistema guardará esta acción como una intención de compra.

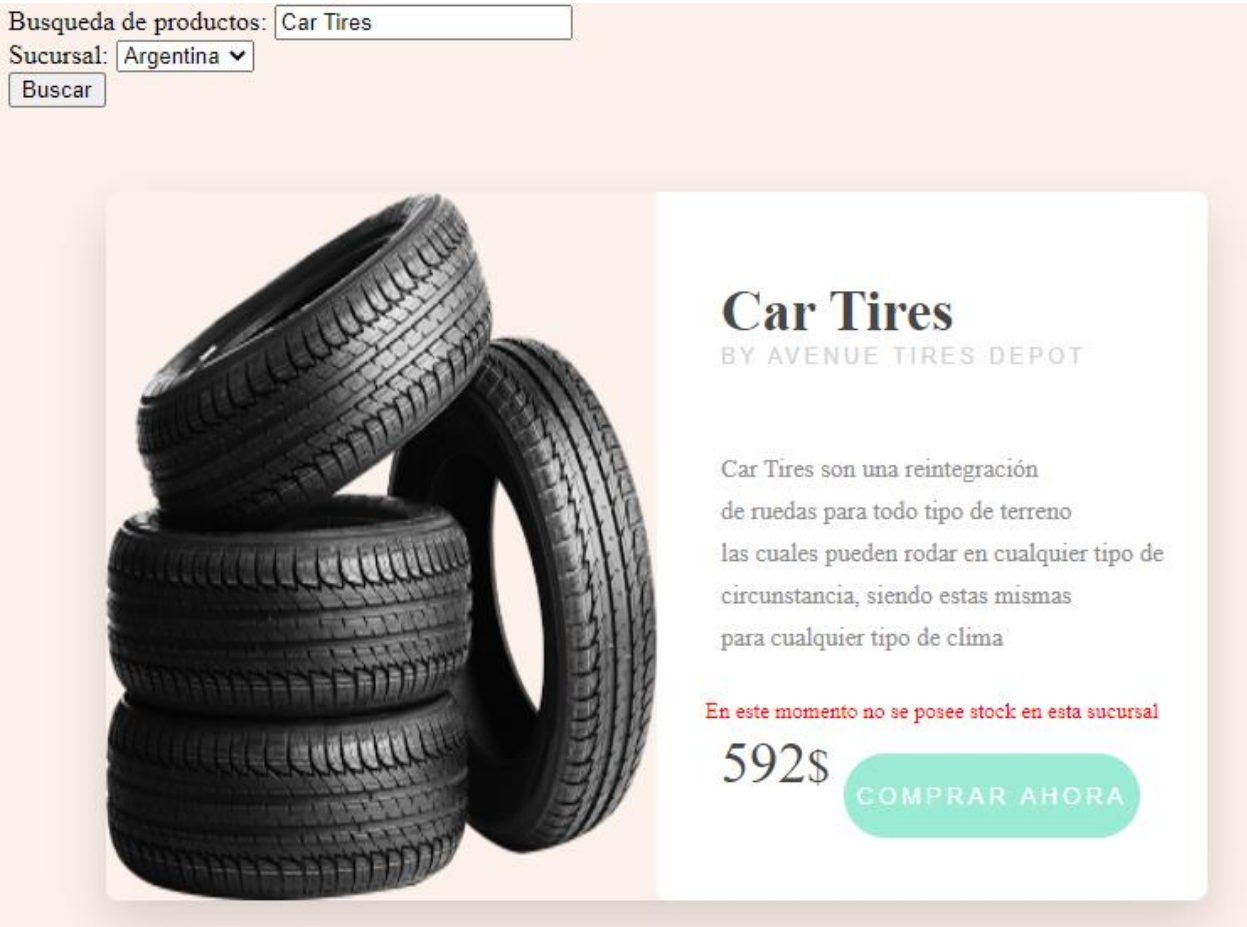


Figure 26 - Portal de ventas

Módulo de servicios

La finalidad de este módulo es exponer servicios con el objetivo de proveer de datos a las aplicaciones web (UI) y a la aplicación de simulación de ventas dándole la posibilidad de interactuar con la base de datos, obteniendo, creando y modificando información.

El stack de tecnologías en este módulo es el siguiente:

1) Aspnet Core 2.1
2) Acceso a datos mediante Entity Framework
3) Auto mapper
4) Dependency Injection
5) Testing using Nunit

Los servicios de la aplicación se dividen de la siguiente manera:

Auth: en este servicio se encuentra el servicio de login, el cual devolverá JWT (json web token) este mismo se utilizará para enviar la autorización a cada uno de los servicios subsiguientes.
Países: En estos servicios se encuentra la manipulación de los países.
Productos: En estos servicios se encuentra la manipulación de los Productos.
Predicciones: En estos servicios se encuentra la manipulación de las predicciones.
Ventas: En estos servicios se encuentra la manipulación de las ventas.
DBTools: En este (controlador) controller se encuentran los servicios generales para la manipulación de la base de datos y su información.

Para una mejor documentación de este módulo de servicios se ha implementado swagger, este es una framework para documentar y poder consumir servicios REST, el mismo posee una aplicación web para poder realizar consultas a los servicios sin utilizar otra herramienta como postman o soapUI.

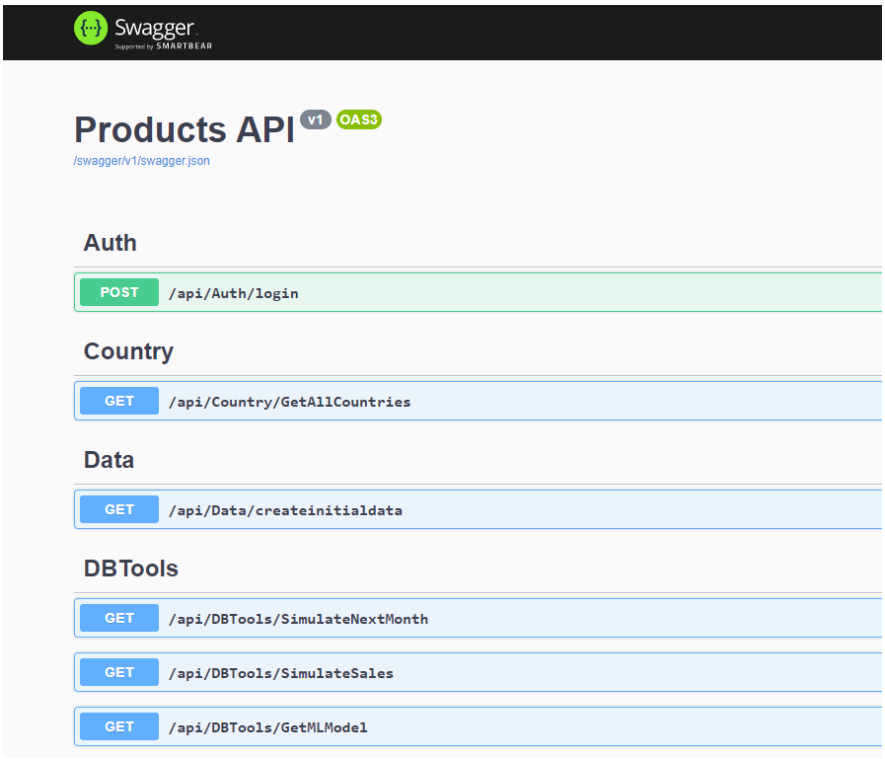


Figure 27 - Herramienta de Swagger para webApi

Módulo de simulación de ventas y machine learning.

La finalidad de este módulo es poseer una herramienta que realice automáticamente las simulaciones correspondientes al momento en que un cliente entra en un punto de venta y tiene la intención de realizar una compra (esta puede ser por el sistema o físicamente a un negocio donde se use el sistema), el cliente seleccionará un producto a comprar de todo el catálogo de la empresa ***(el cual puede o no estar disponible en el punto de venta donde se encuentre el cliente)*** en este momento se generará una **intención de compra**.

Una vez que poseemos la intención de compra el sistema procederá a asentarla en la base de datos, la misma será una intención de compra satisfactoria si el producto se encuentra en ese punto de venta y se realizará la venta, sino será fallida.

Para este módulo se implementó una aplicación basada en consola que puede ser ejecutada por el usuario, la misma se encuentra en la carpeta ProductsSolution\process del código de la aplicación.

Al ejecutar el archivo process.exe se presentará la siguiente aplicación de consola.

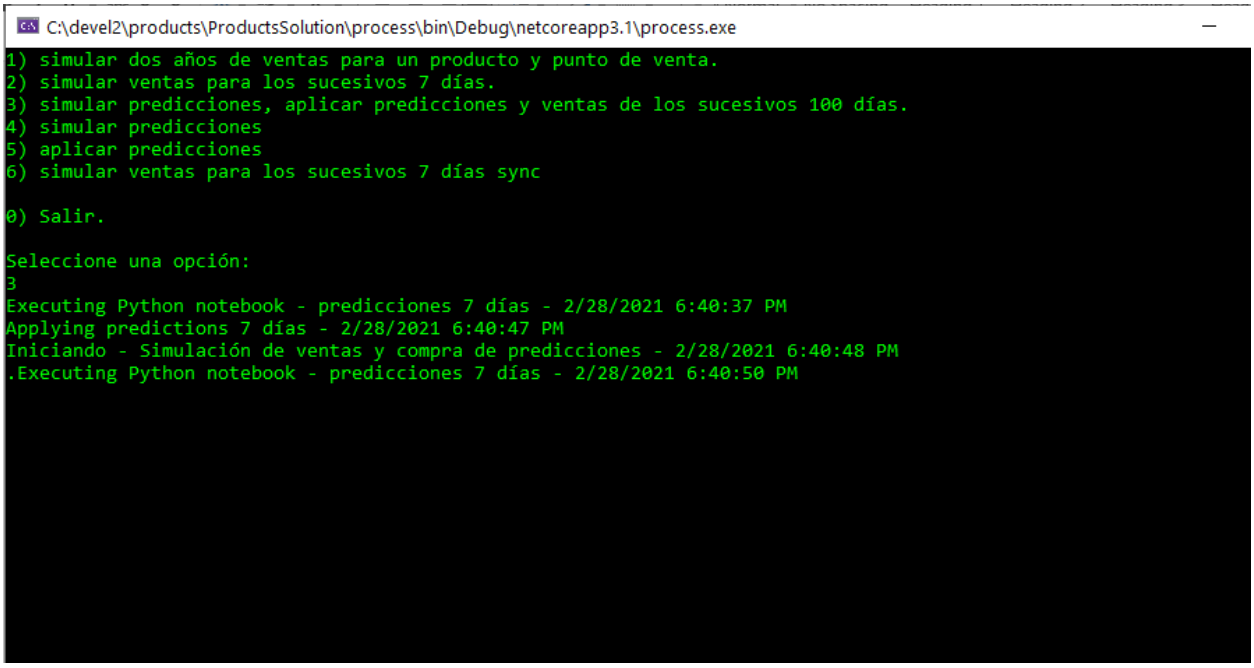


Figure 28 - Sistema de simulación de ventas y predicciones

En la misma el usuario puede seleccionar y ejecutar distintas simulaciones seleccionando la opción que desea:

1. Esta opción el sistema procederá a simular dos años completos de ventas y compras de forma aleatoria para un determinado producto en un determinado punto de venta.
2. En esta otra opción el sistema obtendrá la fecha de las últimas ventas realizadas y comenzará desde ese punto realizando ventas para los 7 días posteriores, sin realizar ninguna compra. Por lo tanto, si el stock del punto de venta no es suficiente se obtendrán intenciones de compras fallidas.
3. En este punto, y siendo el más importante, el sistema simulará los próximos 7 días de ventas, luego las ejecutará las predicciones utilizando el notebook de Python del módulo de machine learning, y una vez terminadas las predicciones las aplicará este proceso se realizará por un año completo.
4. Esta opción ejecutará el notebook de Python por separado para predecir los próximos 7 días de ventas.
5. Este punto recolectará aquellas predicciones que no hayan sido aplicadas al sistema y las aplicará, comprando los productos necesarios para afrontar las siguientes ventas.
6. Esta opción simulará ventas para los próximos 7 días.

La aplicación de consola para la simulación de ventas y ejecución de predicciones posee un archivo de configuración **appsettings.json,** en el mismo podremos en contrar los siguientes atributos, el punto de venta, producto y conexión a la base de datos.

```
{
  "salePointId": 92,
  "productId": 14,
  "cnx": "Data Source=.;User ID=sa; Password=******;Initial Catalog=Products;"
}
```

Módulo de procesamiento Machine Learning. (Pronóstico de Ventas Diarias con Redes Neuronales)

Para la implementación del módulo de machine learning se ha realizado una investigación de distintas herramientas, frameworks y lenguajes utilizados en el mercado. En un principio se utilizó ML.Net, luego de utilizar el mismo se ha logrado tomar algunos resultados y ver cuál era su integración con el resto de los módulos del sistema, uno de los principales beneficios en este

contexto es que al ser un framework en el mismo lenguaje de programación y poder utilizar el mismo IDE de desarrollo, su integración fue bastante buena pero el desarrollo sobre el mismo no es muy intuitivo, aunque para realizar ejemplos como **regresión lineal** o **clasificación** es realmente simple.

Cómo otro punto interesante y a favor, una vez agregada la librería, no se necesita de ningún otro tipo de librería o modulo, es decir que no se necesita mucho más para continuar con el procesamiento de los datos.

Al momento de tratar de trabajar con Deep learning y series temporales, se descartó el framework basado en .NET ya que su utilización no era realmente conveniente para este tipo de tarea. En cambio, se utilizó una de las herramientas más utilizadas en el mercado, un notebook de Python utilizando tensorflow con Keras.

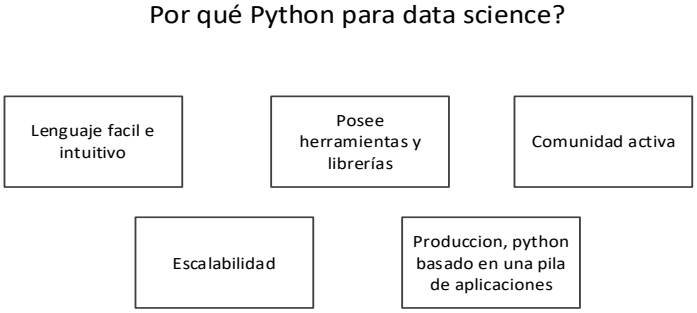


Figure 29 - Características de Python

Se investigó las herramientas para Python con sus distribuciones y packages para data science, a continuación, damos algunas razones por lo cual en este trabajo se seleccionó el mismo para la ejecución del módulo de machine learning.

Lenguaje intuitivo: Python es un popular lenguaje de programación de propósito general que se puede utilizar para una amplia variedad de aplicaciones. Incluye estructuras de datos de alto nivel, tipeo dinámico, enlace dinámico y muchas más características que lo hacen tan útil para el desarrollo de aplicaciones complejas como lo es para secuencias de comandos. También se puede extender para hacer llamadas al sistema operativo y es compatible con casi todos los sistemas operativos o ejecutar código escrito en C o C ++.
Debido a su ubicuidad y capacidad para ejecutarse en casi todas las arquitecturas de sistemas, Python es un lenguaje universal que se encuentra en una variedad de aplicaciones diferentes.

Python es Open source: lo cual quiere decir que es gratis, Python cuenta con una gran y activa comunidad científica con acceso al código fuente del software contribuyendo al continuo desarrollo y mejoras del mismo, dependiendo de las necesidades de los usuarios. Esta es la razón de por qué Python es multiplataforma, es decir se encuentra disponible para la mayoría de los sistemas operativos disponibles, Windows, Mac y Linux.
Esta característica juega un rol fundamental si lo comparamos con otras plataformas como Matlab, que también es utilizada para resolver problemas de Machine Learning en el área financiera, se prefiere, en ocasiones, utilizar Python en vez de Matlab, ya que esta última es paga.

Propósito General: Python no es solamente un lenguaje de programación para ser utilizado en Machine Learning o Ciencia de los Datos, también puede ser utilizado para programar páginas web utilizando el frameworks como Django, aunque ya está área esta fuera del alcance de este trabajo, Python es realmente muy versátil y puede ser utilizado en un sin fin de aplicaciones más allá de las que veremos en este trabajo, así como también en la interoperabilidad entre otros lenguajes de programación como C, Java, R, entre otros.

Lenguaje de Alto Nivel: Las ventajas de utilizar un lenguaje de alto nivel son gigantescas ya que es muy difícil programar y entender los lenguajes de programación de bajo nivel ya que son muy técnicos, por esta razón los lenguajes de alto nivel implementan sintaxis mucho más cercana a la lógica de los humanos lo que hace que sea mucho más fácil de aprender e implementar.
Resumiendo, las ventajas técnicas que ofrece Python en comparación con otros lenguajes de programación son las siguientes:

- Es gratis y actualizado constantemente.
- Puede ser utilizado en múltiples dominios.
- No requiere de mucho tiempo para procesar cálculos y posee una sintaxis intuitiva que permite programar complejas líneas de código.

Con todas estas características hace que Python pueda ser implementando en un sin fin de aplicaciones.

Por lo tanto, Python basa su popularidad en dos pilares, el primero es que es fácil de aprender ya que contiene una sintaxis clara e intuitiva y la segunda razón es que es muy poderoso ya que puede ejecutar una variedad de complejas líneas de código.

Para la utilización de Python en data sciense y machine learning existen una gran cantidad de frameworks y una distribución que ya posee varias herramientas como Jupyter Notebook. Entre las librerías que utilizaremos se encuentra **Pandas**, es una librería de código abierto escrita como extensión de **NumPy (la cual también utilizaremos en este trabajo)** para la manipulación y análisis de datos en Python. Por otro lado, permite de forma rápida para, el análisis, la limpieza y la preparación de los datos, además Ofrece un gran rendimiento y una gran productividad.

Tiene características de visualización preconstruidas, permite trabajar con datos de una gran variedad de fuentes y si uno tiene instalado la distribución anaconda, para instalar pandas, solo tienes que ejecutar el comando.

conda install pandas.

En el caso de que uno no tenga anaconda, se utiliza el comando **pip install pandas.**

Tensor Flow

Otra librería que utilizaremos dentro de nuestro trabajo es tensorflow. Tensor Flow de Google, es una biblioteca de código abierto que se centra principalmente en el aprendizaje profundo (Deep learning). Utiliza gráficos de flujo de datos computacionales para representar una arquitectura de red neuronal complicada. Los nodos en el gráfico denotan cálculos matemáticos, también llamados ops (operaciones), mientras que los bordes denotan los tensores de datos transferidos entre ellos. Además, los gradientes relevantes se almacenan en cada nodo del gráfico computacional, y durante la retropropagación, estos se combinan para obtener los gradientes con respecto a cada peso. Los tensores son matrices de datos multidimensionales que usa TensorFlow.

Serie temporal.

En este trabajo de tesis, visualizaremos las intenciones de compra como una serie temporal la cual es un **conjunto de muestras tomadas a intervalos de tiempo regulares (en el caso de este sistema podemos decir diariamente)**. Es interesante analizar su comportamiento al mediano y largo plazo, intentando detectar patrones y poder hacer pronósticos de cómo será su comportamiento futuro. Lo que hace *diferente* a una **una serie temporal** de un "problema" de Regresión son dos cosas:

1. **Es dependiente del Tiempo**. Esto rompe con el requerimiento que tiene la regresión lineal de que sus observaciones sean **independientes**, este es el principal punto por el cual se utilizó una serie temporal a diferencia de una simple regresión lineal.

2. **Suelen tener algún tipo de estacionalidad**, de tendencias a crecer o decrecer. Pensemos en cuánto más producto vendemos en el sistema en sólo 4 meses al año que en el resto de estaciones.

Una serie temporal es una sucesión de observaciones de una variable realizadas a intervalos regulares de tiempo. Según realicemos la medida de la variable considerada podemos distinguir distintos tipos de series temporales:

2. **Discretas o Continuas:** en base al intervalo de tiempo considerado para su medición.
3. **Flujo o Stock:** En economía, se dice que una serie de datos es de tipo flujo si está referida a un período determinado de tiempo (un día, un mes, un año, etc.). Por su parte, se dice que una serie de datos es de tipo stock si está referida a una fecha determinada (por ejemplo, el 31 de diciembre de cada año). Un ejemplo de datos de tipo flujo serían las ventas de una empresa ya que éstas tendrán un valor distinto si se obtiene el dato al cabo de una semana, un mes o un año; por su parte, la cotización de cierre de las acciones de esa misma empresa sería una variable de tipo stock, ya que sólo puede ser registrado a una fecha y hora determinadas. Obsérvese que existe relación entre ambos tipos de variables, pues la cotización al cierre de las acciones no es más que el precio de cierre del día anterior más, o menos, el flujo de precios de la sesión considerada.
4. **Dependiendo de la unidad de medida:** podemos encontrar series temporales en unidades de dinero como pesos, euros o dólares, o también en diversas magnitudes físicas (kilogramos, litros, millas, etc.)

5. **En base a la periodicidad de los datos:** podemos distinguir series temporales de datos diarios, semanales, mensuales, trimestrales, anuales, etc.

Antes de profundizar en el análisis de las series temporales es necesario señalar que, para llevarlo a cabo, hay que tener en cuenta los siguientes supuestos:

1. Se considera que existe una cierta estabilidad en la estructura del fenómeno estudiado. Para que se cumpla este supuesto será necesario estudiar períodos lo más homogéneos posibles.

2. Los datos deben ser homogéneos en el tiempo, o lo que es lo mismo, se debe mantener la definición y la medición de la magnitud objeto de estudio. Este supuesto no se da en muchas de las series económicas, ya que es frecuente que las estadísticas se perfeccionen con el paso del tiempo, produciéndose saltos en la serie debidos a un cambio en la medición de la magnitud estudiada. Un caso particularmente frecuente es el cambio de base en los índices de precios, de producción, etc. Tales cambios de base implican cambios en los productos y las ponderaciones que entran en la elaboración del índice que repercuten considerablemente en la comparabilidad de la serie en el tiempo.

El módulo de machine learning se ejecutará una vez al día para poder predecir la cantidad de productos que se necesitarán para el próximo día y poder realizar los movimientos adecuados, una vez terminada su ejecución se carga una tabla de la base de datos con las predicciones.
El archivo del código fuente del módulo de machine learning es el modelo_final.py dentro de la carpeta PythonML/Tesis/

Daremos una pequeña explicación del código del módulo de Deep learning

- Primero importaremos las librerías correspondientes de pandas, numpy, matplot, keras.

```
import warnings
warnings.filterwarnings("ignore", category=DeprecationWarning)
import urllib3
urllib3.disable_warnings(urllib3.exceptions.InsecureRequestWarning)
import pymssql as pms
import pandas as pd
import numpy as np

import matplotlib.pylab as plt

from keras.models import Sequential
from keras.layers import Dense, Activation, Flatten
from sklearn.preprocessing import MinMaxScaler
```

Figure 30 - Módulo de Predicción – Import de librerías

En este caso además de las librerías de panda, numpy y keras también importaremos la librería pymsql que nos permite conectarnos a la base de datos de sqlserver para obtener la información para crear los dataframes.

- Como primera tarea creamos un objeto con los datos de la conexión a la base de datos, y utilizamos el método connect de la librería pymsql para obtener un cursor a los datos, una vez que tenemos el cursor comenzaremos a recorrer los datos para su procesamiento.

```
cnx = {
    'host': 'WIN10',
    'username': 'sa',
    'password': 'Ignacio05',
    'db': 'Products'
}

conn = pms.connect(cnx['host'], cnx['username'], cnx['password'], cnx['db'])

cursor = conn.cursor(as_dict=True)
```

Figure 31 - Módulo de Predicción - Conexión a base de datos

- La primera consulta que vamos a ejecutar es para obtener una lista de las intenciones de compra para un producto y un punto de venta, estos datos son los que vamos a utilizar para entrenar a nuestra red neuronal.

En el siguiente código crearemos un dataframe a partir de los datos obtenidos de la consulta a la base de datos, y marcaremos el campo/columna fecha como un índice.

```
query = '''
    select date as fecha, sum(amount) as unidades from Searches
    where  salepointid = ''' + str(salepointid) + ''' and productid = ''' + str(productid) + '''
    group by date
    '''

df = pd.read_sql(query, conn)
df = df.set_index("fecha")
print(df.head())
print(df.index.min())
print(df.index.max())
```

Figure 32 - Módulo de Predicción - dataframe de productos y punto de venta

Se utiliza el comando print de Python para imprimir los valores de cabecera de la consulta y el visualizar la fecha máxima y mínima.

Una vez que tenemos esta información cargada en un dataframe podemos utilizar todas las funciones que posee pandas para la visualización de datos como por ejemplo podemos describir para obtener los datos estadísticos o por otro lado podemos utilizar el siguiente código para hacer un promedio por mes.

```
meses = df.resample('M').mean()
print(meses)
```

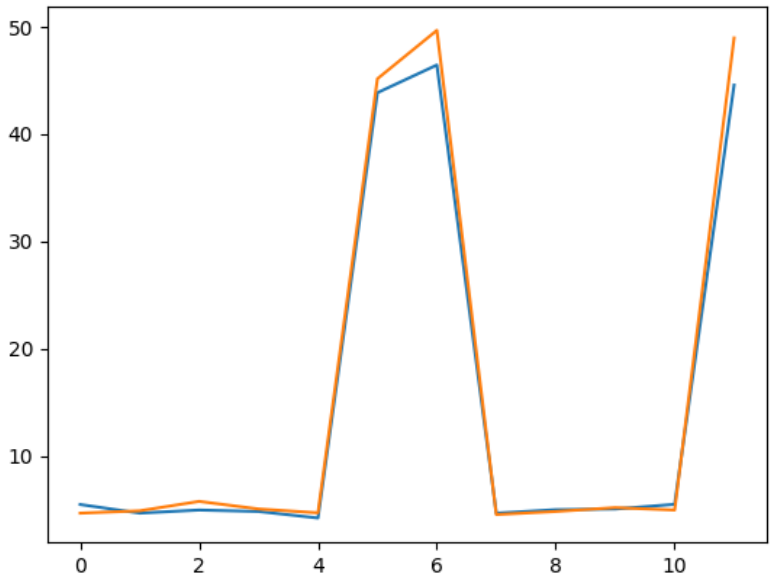


Figure 33 - Módulo de Predicción - gráficos usando matplotlib

Podemos visualizar en la figura 34 como en los meses de verano (hemisferio norte) y diciembre por las navidades las ventas son mayores.

Una vez que tenemos el dataframe compondremos los valores que utilizaremos para entrenar a nuestra red neuronal y la información que utilizaremos como tests.
Con esta información podemos confirmar que la serie es estacionaria, obteniendo valores similares en los periodos de tiempo acordados con lo cual podemos hacer los pronósticos

Se puede encontrar variadas técnicas para efectuar pronóstico, En este caso, las ventas se comportan de manera similar año a año, con esta idea podríamos utilizar el siguiente método “Si en diciembre del 2018 vendimos en promedio de 200 unidades, se podría pronosticar que en diciembre de este año será similar”. Otro método muy utilizado en estadística es el llamado ARIMA.

En este módulo para poder predecir las ventas y proveer el stock a futuro para un punto de venta utilizaremos una arquitectura sencilla de red neuronal FeedForward la cual se explicó en el capítulo anterior, con pocas neuronas y como método de activación tangente hiperbólica pues entregaremos valores transformados entre -1 y 1.

Para poder preparar los datos para nuestro entrenamiento y tests, vamos a utilizar los datos obtenidos de la base datos **(fecha, cantidad de intenciones de compra)** recordar que no lo haremos sobre las ventas sino sobre las intenciones de compras, sean satisfactorias o no.
Convertiremos estos datos en varias columnas. En realidad lo que haremos es tomar nuestra serie temporal y la convertiremos en un **"problema de tipo supervisado"** para poder alimentar nuestra red neuronal y poder entrenarla con backpropagation ("como es habitual").

Entradas: para la entrada utilizaremos "7 columnas" que representan las intenciones de compra de los 7 días anteriores.
Salida: El valor del "8vo día". Es decir, las intenciones de compra (en unidades) de ese día.

Esta transformación se efectuará en una función llamada ***series_to_supervised()*,** Antes de que se pueda utilizar el aprendizaje automático, los problemas de predicción de series de tiempo deben enmarcarse como problemas de aprendizaje supervisado.
Una serie de tiempo es una secuencia de números ordenados por un índice de tiempo, en nuestro caso tenemos la fecha como índice y la cantidad de intenciones de compras, debemos convertir los datos para que se puedan considerar como una lista de columnas de valores ordenados.

Una vez realizada esta transformación, los datos se ven de la siguiente manera, donde el índice es cada columna.

	var1(t-7)	var1(t-6)	var1(t-5)	var1(t-4)	var1(t-3)	var1(t-2)	var1(t-1)	var1(t)
7	-0.314815	-0.311111	-0.114815	-0.370370	-0.714815	-0.103704	-0.225926	-0.433333
8	-0.311111	-0.114815	-0.370370	-0.714815	-0.103704	-0.225926	-0.433333	-0.607407
9	-0.114815	-0.370370	-0.714815	-0.103704	-0.225926	-0.433333	-0.607407	-0.522222
10	-0.370370	-0.714815	-0.103704	-0.225926	-0.433333	-0.607407	-0.522222	-0.644444
11	-0.714815	-0.103704	-0.225926	-0.433333	-0.607407	-0.522222	-0.644444	-0.344444

Figure 34 - Información Transformada

Usaremos como entradas las columnas encabezadas como var1(t-7) a (t-1) y nuestra salida (lo que sería el valor "Y" de la función) será el var1(t) -la última columna-.

A continuación, el código de la función:

```python
# convert series to supervised learning
def series_to_supervised(data, n_in=1, n_out=1, dropnan=True):
    n_vars = 1 if type(data) is list else data.shape[1]
    df = pd.DataFrame(data)
    cols, names = list(), list()
    # input sequence (t-n, ... t-1)
    for i in range(n_in, 0, -1):
        cols.append(df.shift(i))
        names += [('var%d(t-%d)' % (j + 1, i)) for j in range(n_vars)]
    # forecast sequence (t, t+1, ... t+n)
    for i in range(0, n_out):
        cols.append(df.shift(-i))
        if i == 0:
            names += [('var%d(t)' % (j + 1)) for j in range(n_vars)]
        else:
            names += [('var%d(t+%d)' % (j + 1, i)) for j in range(n_vars)]
    # put it all together
    agg = pd.concat(cols, axis=1)
    agg.columns = names
    # drop rows with NaN values
    if dropnan:
        agg.dropna(inplace=True)
    return agg
```

Figure 35 - Función para pasar de una serie a un problema supervisado

Función pandas shift ()

Una función clave para ayudar a transformar los datos de series de tiempo en un problema de aprendizaje supervisado es la función shift () de Pandas.
Dado un DataFrame, la función shift() se puede usar para crear copias de columnas que se empujan hacia adelante (filas de valores de NaN agregadas al frente) o retiradas (filas de valores de NaN agregadas al final).

Para favorecer los cálculos que realiza la red neurona, es importante transformar el rango de valores entre -1 y 1 para lo cual utilizaremos la función MinMaxScaler.
Luego en el siguiente código vemos como realizamos la llamada a la función series_to_supervised.

```
# plt.show()
# Cargamos el dataset
values = df.values
# nos aseguramos que todos los velores sean float 32, recordemos que CUDA y la grafica utiliza float
values = values.astype('float32')
# normalizamos las features
scaler = MinMaxScaler(feature_range=(-1, 1))
values = values.reshape(-1, 1)  # esto lo hacemos porque tenemos 1 sola dimension
scaled = scaler.fit_transform(values)
# encuadramos el problema como un problema de aprendizaje supervisado
reframed = series_to_supervised(scaled, PASOS, 1)
```

Figure 36 - Llamando a la función series_to_supervised

Creamos la Red Neuronal Artificial [5]

Antes de crear la red neuronal, dividiremos el conjunto de datos en entrenamiento y en datos de testeo. Para nuestro problema tenemos que mantener el orden de los datos de entrada, con los que alimentaremos la red. Por lo tanto, obtenemos los primeros 567 días consecutivos para entrenamiento de la red y los 30 siguientes para su validación.

```
# dividimos la información en datos de entrenamiento y datos de prueba
values = reframed.values
n_train_days = 315 + 289 - (30 + PASOS)
train = values[:n_train_days, :]
test = values[n_train_days:, :]
# dividimos en entrada y salida
x_train, y_train = train[:, :-1], train[:, -1]
```

Figure 37 - Entrenamiento de la red neuronal

La entrada la transformamos en un arreglo con forma (567,1,7)
La arquitectura de la red neuronal será:

- Entrada 7 inputs, como dijimos antes
- 1 capa oculta con 7 neuronas.
- La salida será 1 sola neurona
- Como función de activación utilizamos tangente hiperbólica puesto que utilizaremos valores entre -1 y 1.

[5] https://www.aprendemachinelearning.com/pronostico-de-series-temporales-con-redes-neuronales-en-python/

- Utilizaremos como optimizador Adam y métrica de pérdida (loss) Mean Absolute Error
- Como la predicción será un valor continuo y no discreto, para calcular el Acuracy utilizaremos Mean Squared Error y para saber si mejora con el entrenamiento se debería ir reduciendo con las EPOCHS.

En la siguiente función vemos como creamos el modelo para nuestra red neuronal y especificamos cada uno de los valores al modelo.

```
def crear_modeloFF():
    model = Sequential()
    model.add(Dense(PASOS, input_shape=(1, PASOS), activation='tanh'))
    model.add(Flatten())
    model.add(Dense(1, activation='tanh'))
    model.compile(loss='mean_absolute_error', optimizer='Adam', metrics=["mse"])
    model.summary()
    return model
```

Figure 38 - Creación del modelo de red neuronal

Ejecución del módulo de Deep learning.

Primero configuraremos la cantidad de épocas para nuestro aprendizaje, el número de épocas es un parámetro que define el número de veces que el algoritmo de aprendizaje funcionará en todo el conjunto de datos de entrenamiento.

Una época significa que cada muestra del conjunto de datos de entrenamiento ha tenido la oportunidad de actualizar los parámetros internos del modelo. Una época se compone de uno o más lotes. Por ejemplo, como se indicó anteriormente, una época que tiene un lote se denomina algoritmo de aprendizaje de descenso de gradiente por lotes.

Puede pensar en un bucle for sobre el número de épocas en las que cada bucle avanza sobre el conjunto de datos de entrenamiento. Dentro de este bucle for hay otro bucle for anidado que itera sobre cada lote de muestras, donde un lote tiene el número de muestras de "tamaño de lote" especificado.

El número de épocas es tradicionalmente grande, en nuestro caso hemos utilizado 80, pero debería ser más grande para poder tener un error mucho mejor y optimizar el stock al máximo.
Al comienzo del archivo Python veremos la configuración del algoritmo, como se muestra en la figura 37. Especificando la cantidad de épocas, los PASOS y los datos del producto y punto de venta a predecir en los años.

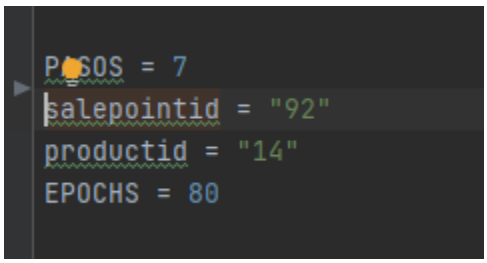


Figure 39 - Configuración del algoritmo

En la figura 38 vemos una de las líneas más importante del algoritmo, el entrenamiento al modelo, al cual le pasamos los datos de entrenamiento, la cantidad de épocas y los pasos.

```
history = model.fit(x_train, y_train, epochs=EPOCHS, validation_data=(x_val, y_val), batch_size=PASOS)
print(x_val)
# Visualizamos al conjunto de validación (recordemos que eran 30 días)

result_scatter = model.predict(x_val)
plt.scatter(range(len(y_val)), y_val, c='g')
plt.scatter(range(len(result_scatter)), result_scatter, c='r')
plt.title('validate')
#plt.show()
```

Figure 40 - ejecución del entrenamiento

En su ejecución vemos una reducción del valor de pérdida tanto del set de entrenamiento como de la validación.

```
50/50 [==============================] - 0s 821us/step - loss: 0.2523 - mse: 0.1035
Epoch 77/80
50/50 [==============================] - 0s 833us/step - loss: 0.2530 - mse: 0.1047
Epoch 78/80
50/50 [==============================] - 0s 831us/step - loss: 0.2526 - mse: 0.1034
Epoch 79/80
50/50 [==============================] - 0s 877us/step - loss: 0.2519 - mse: 0.1036
Epoch 80/80
50/50 [==============================] - 0s 858us/step - loss: 0.2527 - mse: 0.1030
```

Figure 41 - Ejecución de las épocas

En el código anterior vemos como se realiza un gráfico en el cual podemos ver la diferencia entre los valores de prueba y el error en la figura 40 los puntos rojos deben acercarse a los verdes para tener menos error, al tener más épocas (EPOCHS) el error será menor.

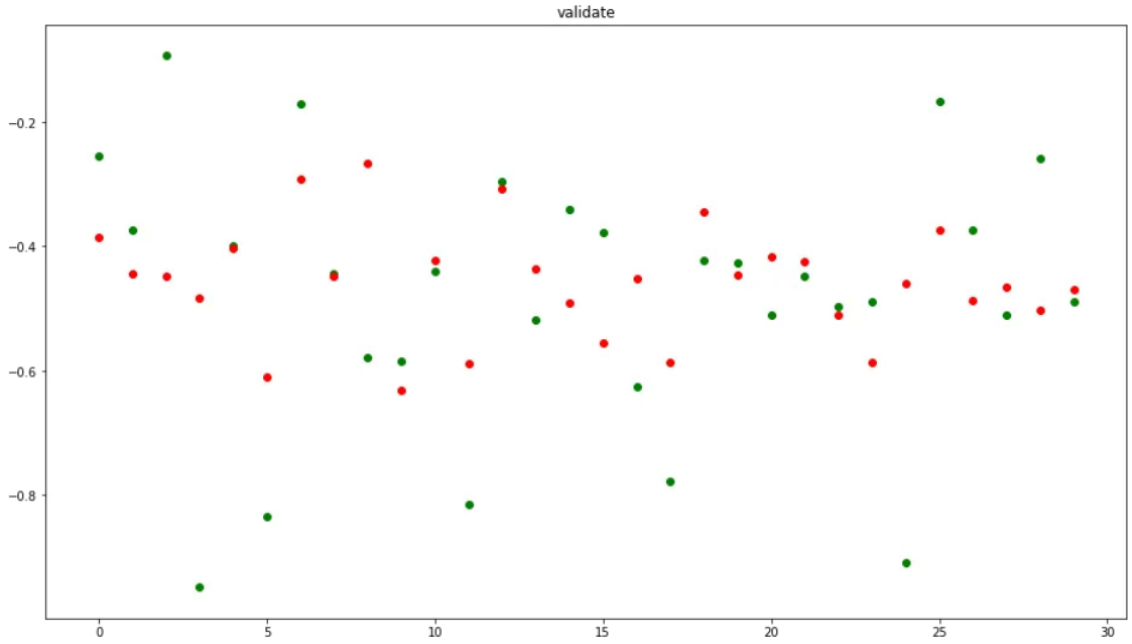


Figure 42 - Diferencia entre los datos de entrenamiento y el error

Con el código anterior tenemos nuestra red configurada y entrenada, ahora podemos utilizarla para predecir intenciones de compra futuras, para esto debemos establecer en qué periodo estamos para poder realizar estimaciones, vamos a obtener desde la base de datos los últimos 7 días.

```
queryUltimos = '''select top 15 CONVERT(date, [date]) as fecha from Searches order by [date] desc'''

dfultimos = pd.read_sql(queryUltimos, conn)
dfultimos = dfultimos.set_index("fecha")
print(dfultimos.head())
print("primer día:" + str(dfultimos.index.min()))
print("ultimo día:" + str(dfultimos.index.max()))

ultimosDias = df[dfultimos.index.min():dfultimos.index.max()]
```

Figure 43 - tomamos los valores para estimar

Ahora hay que realizar el mismo preprocesamiento de datos que se hizo para el entrenamiento: escalando los valores, llamando a la función *series_to_supervised* pero esta vez sin incluir la

columna de salida “Y” pues es la que queremos hallar. Para remover la última columna hacemos drop().

```
values = ultimosDias.values
values = values.astype('float32')
# Normalizamos los features
values = values.reshape(-1, 1)  # esto lo hacemos porque tenemos 1 sola dimension
scaled = scaler.fit_transform(values)
reframed = series_to_supervised(scaled, PASOS, 1)
reframed.drop(reframed.columns[[7]], axis=1, inplace=True)
print(reframed.head(7))

values = reframed.values
x_test = values
#x_test = values[6:, :]
x_test = x_test.reshape((x_test.shape[0], 1, x_test.shape[1]))
print(x_test)

results = []
for i in range(7):
    parcial = model.predict(x_test)
    results.append(parcial[0])
    print(x_test)
    x_test = agregarNuevoValor(x_test, parcial[0])
```

Figure 44 - Creación de los datos de predicción

Una vez que eliminamos la última columna que será nuestra salida, creamos un arreglo con los resultados de cada predicción obteniendo por ejemplo en la figura siguiente la predicción en la última columna.

De esta manera hemos predicho los siguientes 7 días de intenciones de compra, los cuales guardaremos en la base de datos de la siguiente manera.

```
adimen = [x for x in results]
inverted = scaler.inverse_transform(adimen)
inverted

prediccion = pd.DataFrame(inverted)
prediccion.columns = ['pronostico']
prediccion.plot()
#plt.show()
print(prediccion.info())

for index, row in prediccion.iterrows():
    value = row['pronostico']
    print(type(value))
    print(value)
    print(to_str(value))
    with conn.cursor() as cursor:
        query = '''
        INSERT INTO [dbo].[Predictions]
           ([date]
           ,[day]
           ,[month]
           ,[year]
           ,[salepointid]
           ,[productid]
           ,[amount], applied)
         VALUES
           ('2020-12-01'
           ,''' + str(index) + '''
           ,12
           ,2020
           ,''' + str(salepointid) + '''
           ,''' + str(productid) + '''
           ,''' + to_str(value) + '''
           ,0)'''
        # Create a new record
        cursor.execute(query)
        conn.commit()
    #print(str(row['pronostico']))
```

Figure 45 - Insertamos los valores en la base de datos

En el código anterior podemos utilizar la función plot del dataframe generado desde las predicciones para poder graficarlas.

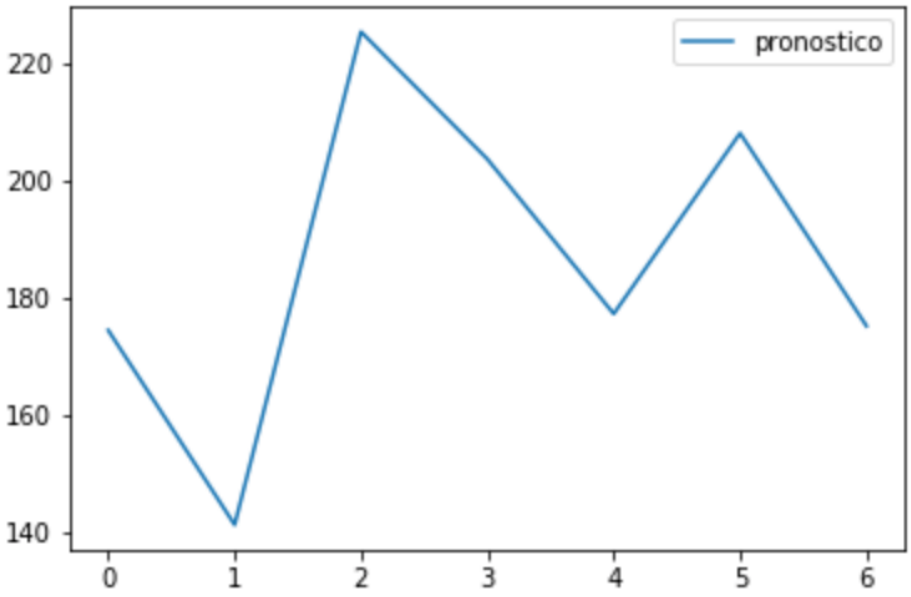


Figure 46 - Grafica de las predicciones

Una vez que se ha realizado la predicción y se ha guardado correctamente en la base de datos se utilizara la UI o el sistema de emulación de ventas para poder aplicar las predicciones y modificar el stock del punto de venta que se está prediciendo.

Predicciones y movimientos de productos entre países.

Una vez que se han realizado las predicciones para los próximos 7 días el usuario tiene la posibilidad, no solo de comprar productos aplicando las predicciones, sino también puede revisar las sugerencias para el movimiento de productos entre países utilizando la siguiente pantalla.

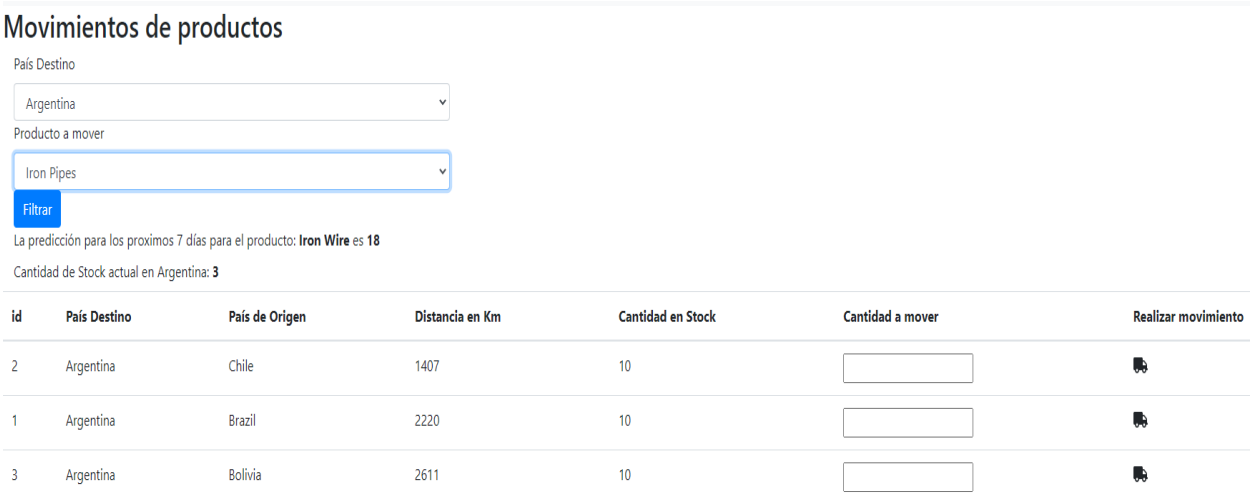


Figure 47 - Pantalla de sugerencias de movimientos

En la figura 45, el usuario debe primero seleccionar el producto y el país origen, es decir el país en donde se realizó la predicción y al cual se quiere mover los productos, una vez seleccionado país y producto el usuario presionará el botón filtrar, el sistema mostrará la predicción para los próximos siete días y cuál es el stock actual en ese país.

Una vez que se ha filtrado la pantalla, el sistema sugerirá los países que poseen stock y cuál es la distancia que se debe recorrer para realizar esos movimientos, el usuario tiene la posibilidad de aplicar movimientos no solo de un país sino combinarlos, presionando el icono del camión el sistema realizara los movimientos necesarios de stock.

Modelo de Base de datos SQL Server.

Los archivos para la generación de la base de datos se encuentran en el proyecto de servicios dentro de la carpeta SQL, en la figura 25 se muestra la carpeta con los archivos sql que el sistema ejecutará al momento de generar los datos iniciales.

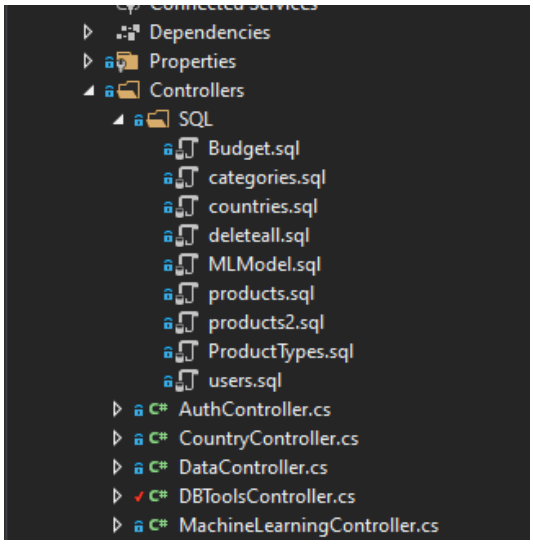


Figure 48 - Scripts de Sql para la generación de basa de datos

El ejecutar la creación inicial de la base de datos se visualizará en pantalla los comandos que se van realizando como se muestra en la figura.

En la figura 46 podemos visualizar el diagrama entidad relación (DER) de la base que da soporte al sistema, la misma fue implementada en una base de datos SQL Server en su versión developer. SQL Server Developer es una edición gratuita con todas las funciones, con licencia para su uso como base de datos de desarrollo y prueba en un entorno que no es de producción. Se utilizo esta base de datos ya que se posee toda su funcionalidad y su fácil integración con el resto del stack de tecnologías.

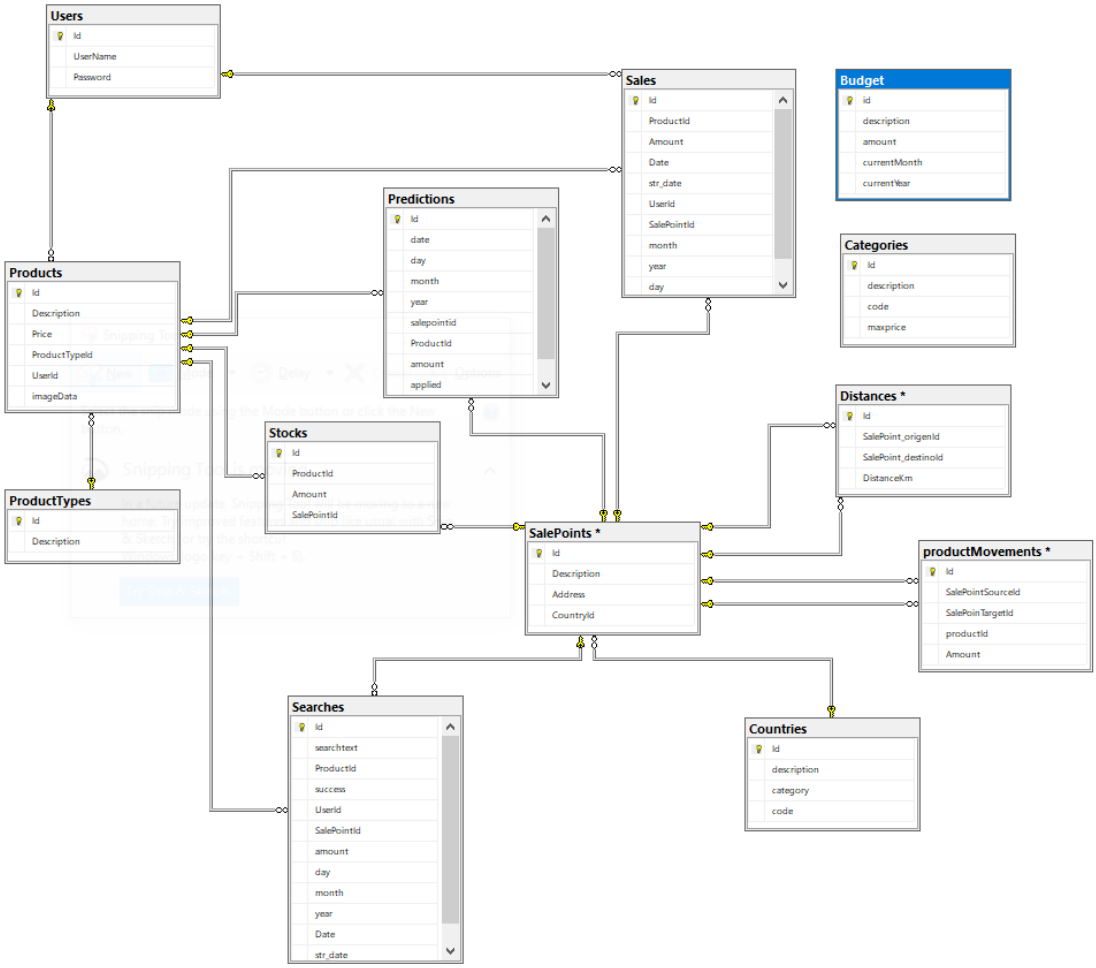


Figure 49 - Modelo de base de datos en SQL Server

Capítulo 6.- Conclusión

A partir de lo expuesto se puede afirmar que la aplicación desarrollada cumple con el objetivo esta tesis, poder ser una herramienta valiosa para el sector de logística de una empresa, incorporando un módulo de machine learning a una aplicación comercial.

La aplicación se ha ejecutado y emulado para visualizar sus resultados unas 10 veces.

Computador de pruebas:

- **Procesador**: Intel Core I7 8750h (6 núcleos, 12 Hilos)
- **Memoria**: 32gb ram
- **GPU**: Nvidia 1050 mobile 4gb
- **Sistema Operativo**: Windows 10

Para las pruebas realizadas se ha simulado las intenciones de compra y ventas de dos años, desde enero del 2017 hasta diciembre del 2018, luego se utiliza esta información para poder predecir los siguientes 7 días, se aplican las predicciones, se simulan las ventas de los siguientes 7 días, se vuelve a utilizar la red para poder predecir nuevamente los próximos 7 días realizando este proceso por todo un año. De esta manera podemos comparar las pérdidas de ventas que se han realizado durante los dos años anteriores donde se ha provisto al punto de venta con productos con valores necesarios para cubrir el día anterior y a partir del año 2019 se puede visualizar que se comenzó a utilizar este algoritmo para predecir el stock necesario.

En la figura 47 podemos visualizar el “big picture” del proceso de simulación del sistema.

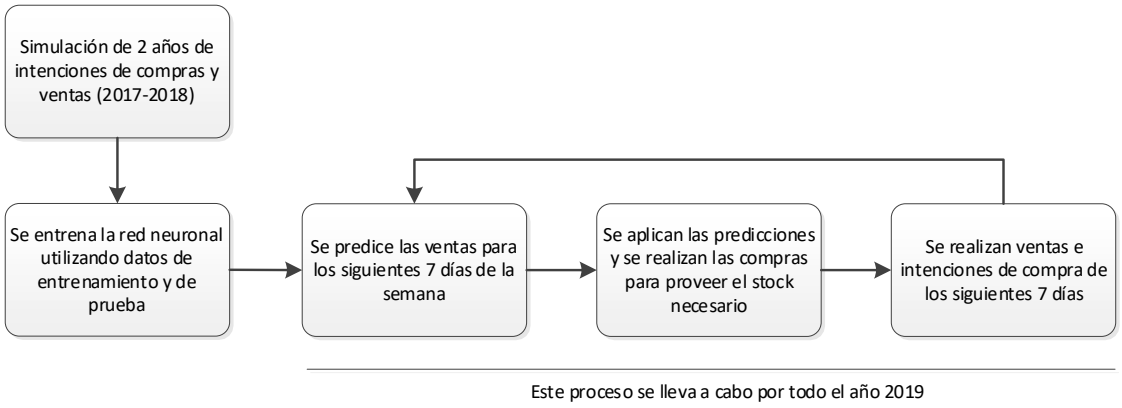


Figure 50 - Big picture de simulación del sistema

Se observa en las siguientes figuras 48 y 49, en verde las intenciones de compra que se han convertido en ventas efectivas y en rojo aquellas intenciones de compra fallidas por falta de stock.

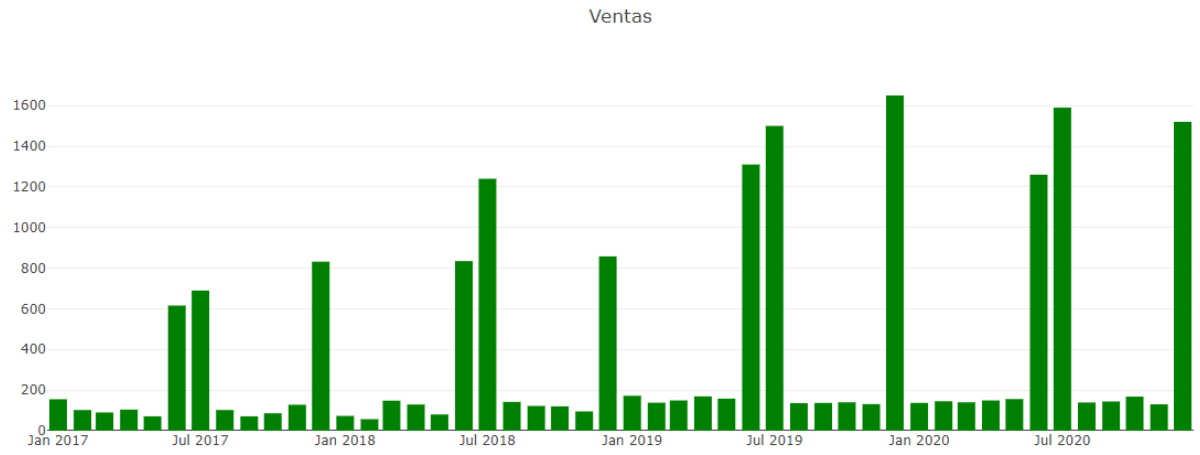


Figure 51 - ventas realizadas entre 2017 y 2020

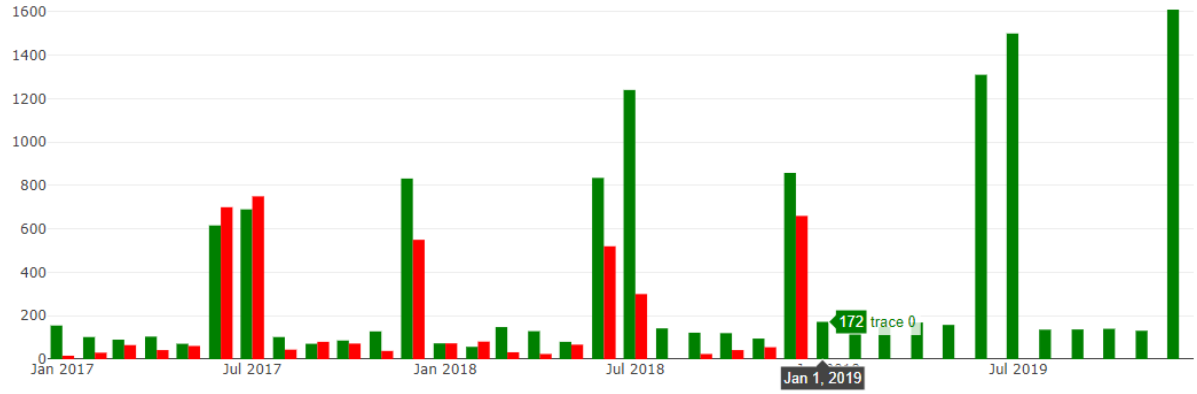


Figure 52- Intenciones de compras y la perdida por falta de stock

En el desarrollo del sistema se ha podido sacar varias conclusiones a partir de distintos tipos de pruebas y se han obtenido distintas observaciones:

1) Al momento de integrar el módulo de machine learning primero se quiso consumir y generar los datos para este módulo utilizando los servicios provistos por el backend en vez de utilizar la base de datos directamente, esto hubiera favorecido en abstracción y en escalabilidad del módulo de ML, en cambio por un problema de performance se decidió que este módulo pueda interactuar directamente con la base de datos. **La cantidad de datos para el entrenamiento de la red**

neuronal en cada ejecución es muy grande para realizar la serialización y pasarlo por un servicio web.

2) Se ha intentado realizar distintos tipos de ejecuciones, como predecir todos los puntos de venta de una sola vez, nuevamente por un problema de performance se decidió que cada ejecución sea para un punto de venta determinado y para un producto. Quizás al tener servidores dedicados, esto podría ser factible.

Conclusiones sobre la arquitectura, al utilizar esta arquitectura y la separación en capas se puede visualizar y garantizar que el sistema es escalable, por ejemplo, en el caso de que la cantidad de usuario crezca y que una sola instancia del microservicio del portal de ventas no pueda dar servicio, se podrá deployar una nueva instancia del mismo utilizando un balanceador de carga.
De la misma manera podría pasar con las aplicaciones de interfaz gráficas, como el portal de manejo de productos y o portal de ventas. Los mismos podría deployarse totalmente de manera autónoma y por separada, lo cual daría la posibilidad de escalar tanto como se necesite.

Cómo se ha mencionado se ha tenido en cuenta la separación de Concerns -preocupaciones-en cada nivel. El nivel de presentación que contiene la interfaz de usuario está separado del nivel de servicios, de la capa de lógica de negocio, que además está separado de la capa de acceso a datos que contiene todo lo necesario para conectarse a la base de datos.

Todos los componentes entre base de datos, lógica de negocio y servicios se conocen por medio de DI (inyección de dependencia) facilitando la escritura de unit tests e integration tests.
Los componentes de alto nivel desconocen que los componentes de bajo nivel los consumen. El nivel de acceso a datos desconoce los servicios que lo consumen y los servicios desconocen el nivel de UX que los consume.

Cada servicio se separa según la lógica empresarial y la funcionalidad que se supone que debe realizar.

La encapsulación se ha cuidado a nivel de arquitectura y también se ha tenido en cuenta durante el desarrollo. Cada componente de la arquitectura interactuará con otros componentes a través de interfaces y contratos bien definidos. Deberíamos poder reemplazar cualquier componente en el diagrama (figura 13) sin preocuparnos por su implementación interna si se adhiere a los contratos.

La arquitectura poco acoplada aquí también ayuda a un desarrollo más rápido y una implementación más rápida en el mercado para los clientes. Varios equipos pueden trabajar en paralelo en cada uno de sus componentes de forma independiente mientras sincronicen los contratos y los plazos para las pruebas de integración.

Capítulo 7 - Trabajos futuros y limitaciones

Testing unitario y testing de integración

La arquitectura utilizada para implementar el sistema permite realizar pruebas unitarias y pruebas de integración con distintos tipos de herramientas, una de las herramientas posibles a implementar para este caso es Selenium, la cual se puede utilizar para realizar pruebas automatizadas de las aplicaciones web. El boilerplate utilizado para implementar angular desde la CLI, agregara Karma para las pruebas unitarias en la web.

Por otro lado, en el backend podemos escribir pruebas unitarias utilizando NUnit para comprobar todas las reglas de negocio de la capa de negocio del sistema. En el código encontraremos varios ejemplos de la manera de escribir estas pruebas, pero como trabajo futuro se podría realizar un coverage más amplio del mismo.

Más allá del aprendizaje automático: aprendizaje profundo y sistemas adaptativos bio-inspirados

En la última década, muchos investigadores comenzaron a entrenar modelos cada vez más grandes, construidos con varias capas diferentes (es por eso que este enfoque se llama aprendizaje profundo), con el fin de resolver nuevos problemas desafiantes. La disponibilidad de computadoras baratas y rápidas les permitió obtener resultados en plazos aceptables y utilizar conjuntos de datos muy grandes (compuestos de imágenes, textos y animaciones). Este esfuerzo condujo a resultados impresionantes, en particular para la clasificación basada en elementos fotográficos y la interacción inteligente en tiempo real mediante el aprendizaje por refuerzo.

La idea detrás de estas técnicas es crear algoritmos que funcionen como un cerebro, y se han logrado muchos avances importantes en este campo gracias a la contribución de las neurociencias y la psicología cognitiva. En particular, existe un creciente interés en el reconocimiento de patrones y los recuerdos asociativos cuya estructura y funcionamiento son similares a lo que sucede en la neocorteza. Tal enfoque también permite algoritmos más simples llamados libres de modelo; estos no se basan en ninguna formulación matemática-física de un problema en particular, sino en técnicas de aprendizaje genéricas y experiencias repetitivas.

Por supuesto, probar diferentes arquitecturas y algoritmos de optimización es bastante más simple (y se puede hacer con procesamiento paralelo) que definir un modelo complejo (que también es más difícil de adaptar a diferentes contextos). Además, el aprendizaje profundo mostró un mejor rendimiento que otros enfoques, incluso sin un modelo basado en el contexto. Esto sugiere que, en muchos casos, es mejor tener una decisión menos precisa tomada con incertidumbre que una precisa determinada por el resultado de un modelo muy complejo (a menudo no tan rápido). Para los animales, esto es a menudo una cuestión de vida o muerte, y si tienen éxito, es gracias a una renuncia implícita de cierta precisión.

Limitaciones del Deep learning

Las técnicas de aprendizaje profundo se adaptan mejor a problemas que se pueden definir con reglas matemáticas formales (como representaciones de datos). Si un problema es difícil de definir de esta manera, es probable que el aprendizaje profundo no proporcione una solución útil. Además, si los datos disponibles para un problema dado están sesgados o solo contienen representaciones parciales de las funciones subyacentes que generan ese problema, las técnicas de aprendizaje profundo solo podrán reproducir el problema y no aprender a resolverlo.

Recuerde que los algoritmos de aprendizaje profundo aprenden diferentes representaciones de datos para aproximarse a una función determinada. Si los datos no representan una función de manera adecuada, es probable que la función sea representada incorrectamente por la red neuronal. Considere la siguiente analogía: está tratando de predecir los precios nacionales de la gasolina (es decir, el combustible) y crear un modelo de aprendizaje profundo. Utiliza el extracto de su tarjeta de crédito con sus gastos diarios en gasolina como datos de entrada para ese modelo. El modelo puede eventualmente aprender los patrones de su consumo de gasolina, pero probablemente tergiversará las fluctuaciones del precio de la gasolina causadas por otros factores que solo se representan semanalmente en sus datos, como las políticas gubernamentales, la competencia del mercado, la política internacional, etc. El modelo finalmente producirá resultados incorrectos cuando se use en producción.

Para evitar este problema, hay que asegurarse de que los datos utilizados para entrenar un modelo representen el problema que el modelo está tratando de abordar con la mayor precisión posible.

Los investigadores han sugerido que el uso de modelos de aprendizaje profundo sin considerar el sesgo inherente en los datos de entrenamiento puede conducir no solo a soluciones de bajo rendimiento sino también a complicaciones éticas.

Por ejemplo, a finales de 2016, investigadores de la Universidad Jiao Tong de Shanghai en China crearon una red neuronal que clasificaba correctamente a los delincuentes utilizando solo imágenes de sus rostros. Los investigadores utilizaron 1.856 imágenes de hombres chinos, de los cuales la mitad habían sido condenados. Su modelo identificó a los presos con un 89,5% de precisión.

Mejora del modelo de Series Temporales con Múltiples Variables y Embeddings

Lo que se hizo en este trabajo hasta el momento es crear una Red Neuronal MLP (Multilayered Perceptron) feedforward de pocas capas, y el mayor trabajo que se hizo fue trabajar los datos de entrada para realizar un buen entrenamiento.
De esta manera utilizamos esos datos para alimentar la red y ésta fue capaz de realizar pronósticos aceptables. **Sólo utilizamos la columna de unidades**, para lo que sigue podemos utilizar las fechas para mejorar el modelo.
Para esta mejora se podría utilizar la fecha como embeddings**.** Los embeddings son una manera de dar valor útil- a datos categóricos. Para ello asignaremos una profundidad a cada "identificador", es decir **un vector con valores continuos inicialmente aleatorios**. Esos valores se ajustarán con backpropagation al igual que nuestra red neuronal. Y finalmente nuestros datos categóricos quedan enriquecidos y dejan de ser "lunes" para ser unos vectores con valores que "significan algo".*¿Qué significan?* para simplificar e intentar entenderlo podemos decir que esos vectores "acercan identificadores similares entre sí y distancia a los opuestos". Un ejemplo: cuando se utiliza en Natural Language Processing (NLP) con un gran número de palabras, los Embeddings logran hacer que palabras sobre sentimientos positivos -"alegría", "felicidad"- queden cercanas pero distanciadas de las que significan sentimientos negativos "odio", "tristeza".
En el caso de nuestro sistema podríamos utilizar la columna de categoría de país para darle un significa mejor a una intención de compra, si el valor del producto supera cierto dinero y la intención de compra es en un país más adinerado, esa intención podría tener más valor que otra, se deja este tipo de análisis para trabajos futuros.

Anexo 1 - Inteligencia Artificial y sus orígenes

Como se menciona anteriormente Alan Turing fue el primero en definir una visión de la IA en su artículo. ***Computing Machinery and Intelligence*****,** en 1950. Ahí, introdujo la prueba de Turing, el aprendizaje automático, los algoritmos genéricos y el aprendizaje por refuerzo.

La prueba de Turing se diseñó para proporcionar una definición operacional y satisfactoria de inteligencia. Alan Turing, en vez de listar una gran cantidad de características para definir si un sistema posee inteligencia, propuso una prueba basada en que un observador no pueda diferenciar si una entidad es un humano o no, manteniendo una conversación.

Estas son algunas de las características de las que un computador de hoy en día debería poseer para que supere la prueba de Turing:

- Procesamiento de lenguaje natural
- Representación del conocimiento
- Razonamiento automático
- Aprendizaje automático.

Además, para la prueba global de Turing se requiere:

- Visión computacional
- Robótica

Orígenes de la inteligencia artificial y aprendizaje automático.

Escapa de los límites del presente trabajo examinar exhaustivamente la evolución histórica de la inteligencia artificial y el aprendizaje automático, sin embargo, importa, sí, dejar sentado lo que a continuación se expresa.

Durante la historia de la inteligencia artificial muchas disciplinas han contribuido con ideas, puntos de vista y técnicas para su desarrollo como la filosofía, matemática e Ingeniería computacional que lo ha hecho desde el año 1940 hasta el presente.

Por ejemplo, los filósofos delimitaron las ideas más importantes de la IA, pero para pasar de ahí a una ciencia formal es necesario contar con una formulación matemática en tres áreas fundamentales: lógica, computación y probabilidad.

El concepto de lógica formal se remonta a los filósofos de la antigua Grecia, pero su desarrollo matemático comenzó realmente con el trabajo de **George Boole** entre los años **(1815-1925)** que definió la lógica proposicional o lógica Booleana.
Varios años después en 1943 se empezaron a construir los pilares de lo que hoy conocemos como IA. Tomaremos este año como punto de entrada para su evolución:

(1943)
En 1943 Warren McCulloch y Walter Pitts elaboraron un modelo constituido por neuronas artificiales basándose en el conocimiento sobre la fisiología básica y el funcionamiento de las neuronas en el cerebro, este modelo se ha convertido en uno de los pilares más importantes de la IA en la historia, En el modelo propuesto por McCulloch y Pitts cada una de las neuronas se caracterizaban por tener un estado, el cual podía estar activada o desactivada, la activación se daba como respuesta a la estimulación producida por una cantidad suficiente de neuronas vecinas.
El estado de una neurona era equivalente a una proposición con unos estímulos adecuados, **Mostraron, por ejemplo, que cualquier función de cómputo podría calcularse mediante alguna red de neuronas interconectadas,** y que todos los conectores lógicos (*and*, *or*, *not*, etc.) se podrían implementar utilizando estructuras de red sencillas, McCulloch y Pitts también sugirieron que redes adecuadamente definidas podrían aprender, se estudiara en profundidad en el capítulo 2 de este trabajo.

(1949)
Donald Hebb (1949) propuso y demostró una sencilla regla de actualización para modificar las intensidades de las conexiones entre neuronas. Su regla, ahora llamada **de aprendizaje Hebbiano o de Hebb**, sigue vigente en la actualidad.

(1950)
Existieron distintos trabajos iniciales que se pueden caracterizar como de IA, pero fue Alan Turing (1912 – 1954) matemático y científico en computación, quien articuló primero una visión de la IA en

su artículo ***Computing Machinery and Intelligence***, en 1950. Ahí, introdujo la prueba de Turing, el aprendizaje automático, los algoritmos genéricos y el aprendizaje por refuerzo.
El análisis formal de la lógica proposicional de Russell y Whitehead y la teoría de la computación de Turing ha contribuido de forma sustancial a la IA y han dado una base fundamental para el estudio de la misma.

(1951)
Dos estudiantes graduados en el Departamento de Matemáticas de Princeton, Marvin Minsky y Dean Edmonds, construyeron el primer computador a partir de una red neuronal en 1951.
La cual se llamó el SNARC (*Stochastic Neural Analog Reinforcement Calculator,* utilizaba 3.000 válvulas de vacío y un mecanismo de piloto automático obtenido de los desechos de un avión bombardero B-24 para simular una red con 40 neuronas.
El comité encargado de evaluar el doctorado de Minsky veía con escepticismo el que este tipo de trabajo pudiera considerarse como matemático, pero Von Newmann (1903- 1957) dio apoyo al proyecto y cito, «Si no lo es actualmente, algún día lo será». posteriormente probó teoremas influyentes que mostraron las limitaciones de la investigación con redes neuronales.

(1956)
McCarthy, Minsky, Claude Shannony y Nathaniel Rochester Organizaron un taller con una duración de dos meses en Darmouth en el verano de1956. Hubo diez asistentes en total, entre los que se incluían Trenchard More de Princeton, Arthur Samuel de IBM, y Ray Solomonoff y Oliver Selfridge del MIT.
Dos investigadores del Carnegie Tech, Allen Newell y Herbert Simon, acapararon la atención. Si bien los demás también tenían algunas ideas y, en algunos casos, programas para aplicaciones determinadas como el juego de damas, Newell y Simon contaban ya con un programa de razonamiento, el Teórico Lógico (TL), del que Simona firmaba: «Hemos inventado un programa de computación capaz de pensar de manera no numérica, con lo que ha quedado resuelto el venerable problema de la dualidad mente-cuerpo». Se dice que Russell se manifestó complacido cuando Simon le mostró que la demostración de un teorema que del programa que había generado era más corta que la que aparecía en *Principia*.

El taller de Dartmouth no produjo ningún avance notable, pero puso en contacto a las figuras importantes de este campo. Durante los siguientes 20 años, el campo estuvo dominado por estos personajes, así como por sus estudiantes y colegas del MIT, CMU, Stanford e IBM.

De este taller surgió el consenso en adoptar el nuevo nombre propuesto por McCarthy, IA (Inteligencia artificial). Quizá «racionalidad computacional» hubiese sido más adecuado, pero «IA» se ha mantenido en el tiempo y revisando la propuesta del taller de Dartmouth (McCarthy *et al.*, 1955), se puede apreciar por qué fue necesario para la IA convertirse en un campo separado.
Estos primeros años de la IA estuvieron llenos de éxitos (aunque con ciertas limitaciones). Teniendo en cuenta lo primitivo de las computadoras del momento y las herramientas de programación de aquella época.

(1957 - 1958)
En 1957 Frank Rosenblatt diseña el perceptrón, una red neuronal en hardware para el reconocimiento de caracteres. El propósito era de explicar y modelar las habilidades de reconocimiento de patrones de los sistemas visuales biológicos.

Tanto John McCarthy como Newell y Simon seguirían aportando e investigando durante los siguientes años, como el sistema de resolución general de problemas o SRGP de Newell y Simon El éxito del SRGP y de los programas que le siguieron, como los modelos de cognición, llevaron a Newell y Simon (1976) a formular la famosa hipótesis del **sistema de símbolos físicos,** o las contribuciones de McCarthy en 1958 que dieron facilidades para que la IA siga creciendo, la creación del lenguaje LISP, el tiempo compartido y el programa el Generador de Consejos que fue parte de un artículo titulado *Programs with Common Sense publicado por* McCarthy, , un programa hipotético que podría considerarse como el primer sistema de IA completo

Al igual que el Teórico Lógico y el Demostrador de Teoremas de Geometría, McCarthy diseñó su programa para buscar la solución a problemas utilizando el conocimiento. Pero, a diferencia de los otros, manejaba el conocimiento general del mundo. Por ejemplo, mostró cómo algunos axiomas sencillos permitían a un programa generar un plan para conducirnos hasta el aeropuerto y tomar un avión. El Generador de Consejos incorporaba así los principios centrales de la representación del conocimiento y el razonamiento: es útil contar con una representación formal y explícita del mundo y de la forma en que la acción de un agente afecta al mundo, así como, ser capaces de manipular estas representaciones con procesos deductivos.

Luego más tarde Minsky se unió a las investigaciones de IA aunque McCarthy se centró en la representación y el razonamiento con lógica formal, mientras que Minsky estaba más interesado en lograr que los programas funcionaran y eventualmente desarrolló un punto de vista anti-lógico.

Stanford fue cuna de muchos investigadores de IA y acuno a McCarthy. El trabajo realizado en Stanford hacía énfasis en los métodos de propósito general para el razonamiento lógico. Un proyecto importante para la IA fue el proyecto de robótica de Shakey fue el primero que demostró la total integración del razonamiento lógico y la actividad física.

En el año 1959 Arthur Lee Samuel popularizo el termino machine learning, Samuel fue un pionero en el campo de programación de juegos e Inteligencia Artificial, muchos papers escritos por Samuel siguen valiendo la pena su estudio. Él ha inventado una gran cantidad de técnicas alrededor del aprendizaje como por ejemplo funciones de evaluación mutable, escalada y tablas de firmas.

Los métodos de aprendizaje de Hebbse reforzaron con las aportaciones de Bernie Widrow (Widrow y Hoff, 1960; Widrow,1962), quien llamó **adalines** a sus redes, y por Frank Rosenblatt (1962) con sus **perceptrones**.

(1971-1974)

Para este punto los desarrollos e investigaciones ya habían crecido, durante el año 1963 un grupo de estudiantes realizaron un conjunto de trabajos denominados micro mundos, con los cuales se podían resolver problemas de integración de cálculo en forma cerrada y requerían inteligencia. El micro mundo más famoso fue **el mundo de los bloques**, que consiste en un conjunto de bloques sólidos colocados sobre una mesa y una tarea típica de este mundo es la reordenación de los bloques de cierta manera, con la ayuda de la mano de un robot que es capaz de tomar un bloque cada vez.

El mundo de los bloques fue el punto de partida para el proyecto de visión de David Huffman (1971), la visión y el trabajo de propagación con restricciones de David Waltz (1975), la teoría del aprendizaje de Patrick Winston(1970), del programa **para la comprensión de lenguaje natural** de Terry Winograd(1972) y del planificador de Scott Fahlman (1974).

Rosenblatt demostró el famoso teorema del perceptrón, con lo que mostró que su algoritmo de aprendizaje podría ajustar las intensidades de las conexiones de un perceptrón para que se adaptaran a los datos de entrada, siempre y cuando existiera una correspondencia

(1979)

Luego en 1979, los estudiantes de la universidad de Stanford diseñan un auto autónomo que posee la capacidad de moverse autónomamente por una habitación evitando obstáculos.

(1969-1981)

Hasta el momento se utilizaba el mecanismo de búsqueda de propósito general, en los cuales se entrelazaban elementos de razonamiento básico de propósito general para encontrar así soluciones, este procedimiento se lo denomina método débil. La alternativa a los métodos débiles es el uso de conocimiento específico del dominio que facilita el desarrollo de etapas de razonamiento más largas, pudiéndose así resolver casos recurrentes en dominios de conocimiento restringido. El programa DENDRAL (Buchanan et al., 1969) constituye uno de los primeros ejemplos de este enfoque.

Luego en 1981 Gerald DeJong crea el concepto de aprendizaje basado en la experiencia, haciendo que un computador analice información de entrenamiento y cree una regla general que le permita descartar información no importante.

Es importante este punto de la historia ya que muchos modelos de aprendizaje automático se basan en aprendizaje basado en experiencia. (Russel y Norvig, 1994, pp 19 - 31).

(1990's)

En esta década se han realizado distintos tipos de trabajos con IA como la creación de Deep blue, un computador capaz de jugar al ajedrez la cual ha jugado varias partidas contra Gary Kasparov.

Por otro lado, en los años 90's se empieza a ver diferentes tipos de aplicaciones para el manejo de grandes cantidades de información cómo la utilización de Big Data, este término ha estado en uso desde la década de 1990, y algunos le dan crédito a John Mashey por popularizar el término. Se realizará el estudio de bug data en el Capítulo de 6.

(2006-2014)

Luego de un tiempo podemos destacar que en el año 2006 en el cual Geoffrey Hinton presenta el concepto de Deep learning (aprendizaje profundo) para explicar los nuevos algoritmos que permiten que los computadores puedan distinguir objetos y textos en imágenes y videos. Este tipo de algoritmo se utiliza en muchos ámbitos, como por ejemplo en 2010, el Kinect de Microsoft puede reconocer varias características del cuerpo humano gracias a la IA.

Otro ejemplo de aprendizaje profundo es el denominado proyecto "Google Brain" comenzó en 2011 como una colaboración de investigación a tiempo parcial entre Jeff Fellow Jeff Dean, el investigador de Google Greg Corrado y el profesor de la Universidad de Stanford Andrew Ng. Ng en 2011 comenzó a colaborar con Dean y Corrado para construir un sistema de software de aprendizaje profundo a gran escala sobre la infraestructura de la nube de Google.

En junio de 2012, el New York Times informó que un grupo de 16,000 computadoras dedicadas a imitar algunos aspectos de la actividad del cerebro humano se había entrenado con éxito para reconocer a un gato basado en 10 millones de imágenes digitales tomadas de videos de YouTube.

Volviendo a la prueba de Turing de 1950 recién en 2014 se realizó un sistema basado en IA (chatbot) que solo pudo convencer a un 33% de los jurados que era genuinamente un humano.
En el año 2014, Deepface es otro ejemplo de un algoritmo basado en aprendizaje profundo el cual puede reconocer una persona a partir de una foto.

(2015 al presente)
Estos son los años realmente importantes para el machine learning, Amazon lanza su propia plataforma de aprendizaje automático y Microsoft lanza DMTK https://www.microsoft.com/en-us/research/project/dmtk/ DMTK es un proyecto de código abierto que se ejecuta en el grupo de Microsoft para compartir los resultados de investigación recientes sobre aprendizaje automático distribuido con la comunidad de código abierto.
Para este año Google crea otro chatbot, el cual no solo realiza soporte técnico, sino que puede tener varias conversaciones profundas y realizar observaciones.
En este año OpenAI es creada, es una compañía sin fines de lucro con la finalidad de desarrollar y promover la Inteligencia Artificial. Uno de los creadores de esta fundación es Elon Musk.

Anexo 2 - Redes Neuronales.

En 1943 McCulloch y Pitts elaboraron un modelo constituido por neuronas artificiales, en este capítulo se elaborará una definición más detallada sobre esta estructura, con la intención de tener un marco teórico sobre este modelo computacional, y **poder abordar tecnologías como Deep learning con Redes Neuronales utilizando Tensorflow.**

Las redes neuronales se han convertido en el grupo de algoritmos más populares en esta última oleada de la inteligencia artificial que estamos viviendo.
Gracias a la mejora de técnicas y a la tecnología que este grupo de algoritmos han comenzado a utilizar en muchas áreas de la investigación, como reconocimiento de caracteres, reconocimiento de imágenes, reconocimiento de voz, generación de texto, traducción de idiomas, prevención de fraude, conducción autónoma y análisis genético, y estos son solo algunos de los ejemplos en donde las redes neuronales han sido utilizadas.

Daremos principal importancia a este concepto para este trabajo de tesis, ya que la implementación final del proyecto utilizará Deep learning y TensorFlow con keras para realizar la predicción de las ventas.

McCulloch y Pitts se basaron en el modelo biológico de las redes neuronales y de cómo realmente funcionan estas en el cerebro humano, demos una explicación sobre este modelo.

El modelo Biológico.

Existen muchos modelos computacionales que han tomado la naturaleza como inspiración, al igual que la **lógica difusa**, las redes neuronales es una tecnología de inteligencia artificial que toma a esta como inspiración, y tratan de emular el funcionamiento de una neurona real.
En el cerebro humano existen una gran cantidad de células (**neuronas)**, una neurona es una célula viva y, como tal, contienen los mismos elementos que forman parte de todas las células biológicas.

Una neurona consta de un **cuerpo**, una **rama principal** y varias ramas cortas llamadas **dendritas**.

Una de las características que diferencian a las neuronas del resto de las células vivas, es la capacidad que tienen de comunicarse con el resto de neuronas, las dendritas y el cuerpo celular reciben señales de entrada, las combina e integra y emite señales de salida. Por lo general, una neurona recibe información de miles neuronas y envía información otros miles más. Se calcula que en el cerebro humano existen del orden de 10^15 conexiones.

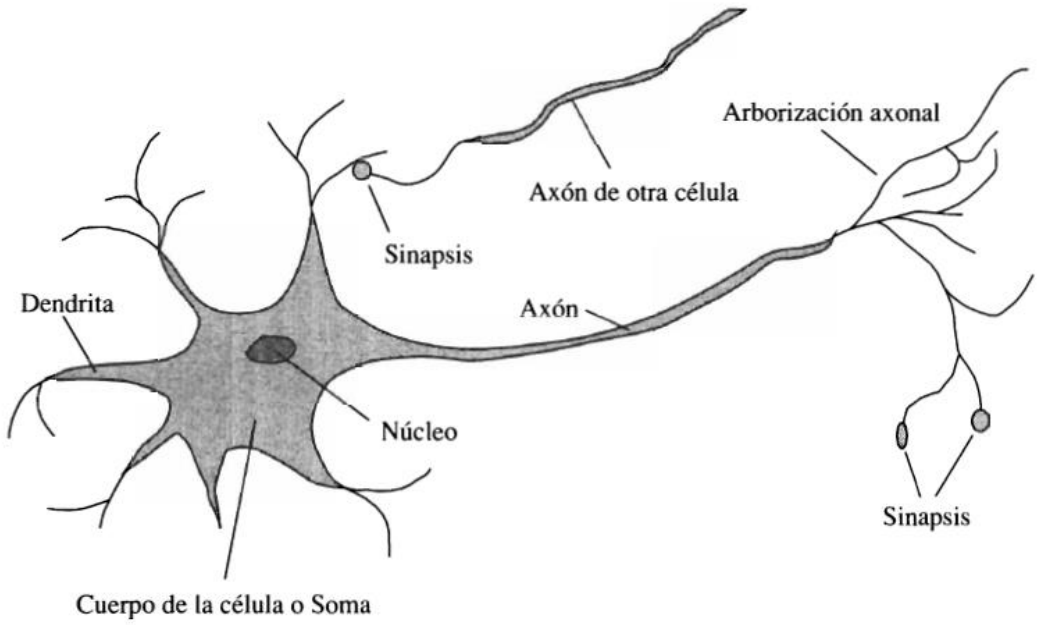


Figure 53 - Neurona biológica de un humano

Las neuronas artificiales pretenden mimetizar las características más importantes de las neuronas biológicas. Cada neurona i-ésima está caracterizada en cualquier instante por un valor numérico denominado **valor o estado de activación, es decir, en cualquier momento que observemos el estado de activación de una neurona obtendremos un valor numérico.**

Luego para que la red neuronal pueda realizar alguna tarea existe lo que definimos como una función de activación o de transferencia que transforma el estado actual de activación, en una señal de salida y_j, Dicha señal de salida es enviada a través de canales de comunicación unidireccionales a otras unidades de la red, en estos canales la señal se modifica de acuerdo con la sinapsis (el peso w_j) asociada a cada uno de ellos según una determinada regla.
Las señales moduladas que han llegado a la unidad j-ésima se combinan entre ellas, generando así la entrada total de Net_j.

$$Net_j = \sum_i^n y_j \ w_j$$

La actualización de las neuronas (evolución de la red neuronal) puede ser de dos tipos, modo asincrónico o sincrónico. En el primer caso, las neuronas evalúan su estado continuamente según les va llegando la información y lo hacen de forma independiente. En el caso sincrónico la información también llega de forma continua pero los cambios se realizan simultáneamente como si existiera un reloj interno que decidiera cuando deben cambiar su estado. Los sistemas biológicos quedan probablemente entre ambas posibilidades.

La Neurona artificial

Si se tienen N unidades de neuronas podemos ordenarlas arbitrariamente y designar a la j-ésima unidad como U subJ. **El trabajo de cada una de las neuronas es simple y consiste en recibir las entradas de las células vecinas y calcular un valor de salida y enviarlo a todas las células conectadas a ella.**

En cualquier sistema que se esté modelando es útil caracterizar tres tipos de unidades: entradas, salidas y ocultas. Las unidades de **entrada** reciben señales desde el entorno; estas entradas (que son a la vez entradas a la red) pueden ser señales provenientes de sensores o de otros sectores del sistema (salidas de la red). Las señales de salida pueden controlar directamente otros sistemas. Las unidades **ocultas** son aquellas que tienen sus entradas y salidas dentro del sistema es decir no tienen contacto con el exterior.
Se conoce como capa o nivel a un conjunto de neuronas cuyas entradas provienen de la misma fuente (que puede ser otra capa de neuronas).

Estado de activación

Como dijimos anteriormente adicionalmente al conjunto de unidades, la representación necesita los estados del sistema en un tiempo t. Esto se especifica por un vector de números reales A(t) , que representa el estado de activación de unidades de procesamiento. Cada elemento del vector representa la activación de una unida U_i, en el tiempo t. La activación de una unidad en el tiempo t se designa por $a_i(t)$, es decir.

El procesamiento que realiza la red se ve como la evolución de un patrón de activación en el conjunto de unidades que lo componen a través del tiempo.

Todas las neuronas que componen la red se hallan en cierto estado. Los valores que pueden tomar un estado pueden ser continuos o discretos. Además, pueden ser limitados. Si son discretos, suelen tomar un conjunto pequeño de calores o bien valores binarios.

Finalmente es necesario saber qué criterios o reglas siguen las neuronas para alcanzar tales estados de activación. En principio esto va a depender de dos factores. Por un lado, es necesario tener idea del mecanismo de interacción entre las neuronas. El estado de activación estará fuertemente influenciado por tales interacciones, ya que el efecto que producirá una neurona sobre otra será proporcional a la fuerza, peso o magnitud de la conexión entre ambas. Por otro lado, la señal que envía cada neurona a sus vecinas dependerá de su propio estado de activación.

Conexiones entre neuronas.

Las conexiones que existen en un a RNA tienen asociado un peso, qué es el que hace que la red neuronal adquiera conocimiento. Es la salida de una neurona i en un instante dado. Una neurona recibe un conjunto de señales que le dan información del estado de activación de todas las neuronas con las que se encuentra conectada. Cada conexión (sinapsis) entre la neurona i y la j está ponderada por un peso W_{ij},
Se considera que el efecto de cada señal es aditivo, de tal forma que la entrada neta que recibe una neurona (potencial postsináptico) Net_j es la suma del producto de cada señal individual por el valor de la sinapsis que conecta ambas neuronas:

$$Net_j = \sum_{i}^{N} w_{ij}\ y_i$$

El aprendizaje profundo es un subcampo especial o rama del aprendizaje automático o machine learning. **La metodología de aprendizaje profundo o Deep learning se basa en redes neuronales.**
Un excelente ejemplo de aprendizaje profundo más utilizados en la actualidad son las aplicaciones chatbots, la empresa **Hanson Robotics** ha llevado al extremo muchas de las técnicas de IA y posee uno de los ejemplos de chatbots más avanzados, el robot Sophia.

Otro ejemplo podría ser la atención al cliente en línea por parte de los bancos que es realizada a través de un chatbot de aplicaciones móviles o web.

Dichas aplicaciones (es decir, chatbots) son poderosas cuando se trata de comprender el contexto de las solicitudes, preferencias e intereses de los clientes. El chatbot está conectado a aplicaciones de back-end que interactúan con los almacenes de datos. Según los aportes del cliente o la selección de servicios, el chatbot presenta al cliente varios subservicios alternativos para elegir, es decir el mismo chatbot realiza la sugerencia al cliente de distintos servicios.

Las aplicaciones de aprendizaje profundo funcionan en capas. Cada capa se construye en base al aprendizaje o conocimiento reunido de la capa de complejidad anterior. Así es como funciona el aprendizaje profundo. El programa continúa aprendiendo, formando más conocimiento con nuevas capas de complejidad basadas en el conocimiento recibido de la capa anterior. La complejidad en capas es de donde se tomó la palabra profunda. El aprendizaje profundo es un tipo de aprendizaje **no supervisado,** por lo que es mucho más rápido que el aprendizaje supervisado.
El impacto principal del aprendizaje profundo es que el rendimiento del modelo es mejor, ya que puede acomodar un razonamiento más complejo. Queremos que las decisiones financieras se tomen con precisión. Esto significa que será más rentable dar a los accionistas de los bancos un rendimiento razonable mientras se equilibran los intereses de los clientes del banco.

Las aplicaciones comunes de aprendizaje profundo incluyen lo siguiente:

- Clasificación de la imagen
- Seguimiento visual en tiempo real
- Conducción autónoma
- Control de robot
- Optimización logística
- Bioinformática
- Reconocimiento de voz y comprensión del lenguaje natural (NLU)
- Generación del lenguaje natural (NLG) y síntesis del habla

Muchos de estos problemas también pueden resolverse mediante el uso de enfoques clásicos que a veces son mucho más complejos, pero el aprendizaje profundo los superó a todos. Además, permitió extender su aplicación a contextos inicialmente considerados extremadamente

complejos, como los automóviles autónomos o la identificación de objetos visuales en tiempo real.

Algoritmo de aprendizaje de perceptrones.

Aunque existen limitaciones a lo que pueden hacer los algoritmos de aprendizaje de Perceptrón, son el precursor de las técnicas avanzadas en Deep learning que vemos hoy. Por lo tanto, vale la pena un estudio del Perceptrones y el algoritmo de aprendizaje de Perceptrón. El nombre de perceptrón que fue acuñado por Frank Rosenblatt con 1962.

Los perceptrones son clasificadores binarios lineales que utilizan un hiperplano para separar las dos clases. Se garantiza que el algoritmo de aprendizaje perceptrón obtendrá un conjunto de pesos y sesgos que clasifique todas las entradas correctamente, siempre que exista un conjunto de pesos y sesgos tan factible.
El perceptrón es un clasificador lineal y, como vimos en el Capítulo anterior, los clasificadores lineales generalmente realizan una clasificación binaria al construir un hiperplano que separa la clase positiva de la clase negativa.
El hiperplano está representado por un vector de peso w que es perpendicular al hiperplano y un término de sesgo b que determina la distancia del hiperplano desde el origen.

La regla de aprendizaje del perceptrón solo puede separar clases si son linealmente separables en el espacio de entrada. Si por ejemplo graficaríamos una compuerta and donde cada eje fuera cada uno de nuestros valores y cada punto verde es donde la salida es verdadera y rojo sería falso, nuestro aprendizaje de perceptrón podría separar correctamente las categorías.

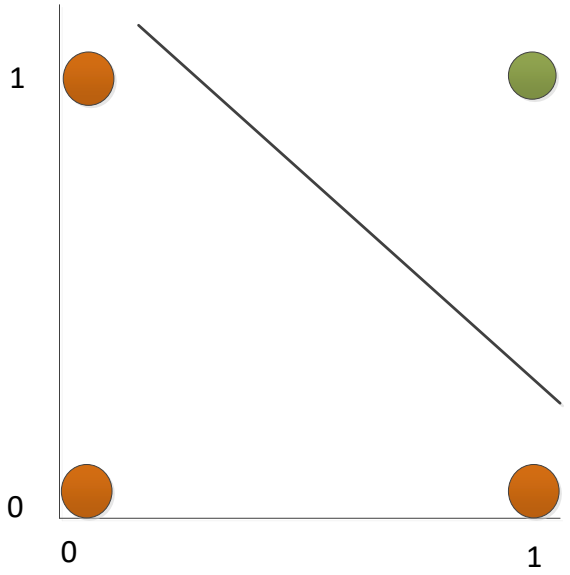


Figure 54 - Grafica de clasificación lineal

Pero luego si quisiéramos hacer lo mismo con una puerta lógica XOR ya no se podría implementar mediante la regla de aprendizaje del perceptrón ya que necesitaríamos dos líneas para poder separar los valores, como se muestra en la figura 52.

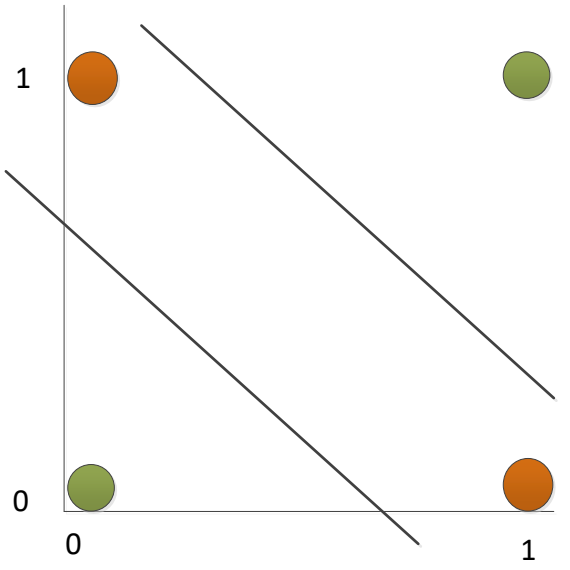


Figure 55 - Grafica de clasificación lineal - ejemplo XOR

Necesidad de la no linealidad

El algoritmo Perceptrón solo puede aprender un límite de decisión lineal para la clasificación y, por lo tanto, no puede resolver problemas donde la no linealidad en el límite de decisión es una necesidad.
Necesitamos tener dos hiperplanos para separar las dos clases, como se puede visualizar en la figura 52.

De la misma manera sucede con las redes neuronales, como vivos anteriormente en este capítulo, una neurona realiza un cálculo (una suma ponderada) sobre los valores de sus entradas y el peso de cada una de las conexiones, si la función de activación de la neurona es lineal cada una de las neuronas actuara de la misma manera que una regresión lineal y solo podrá hacer tareas que posean linealidad.

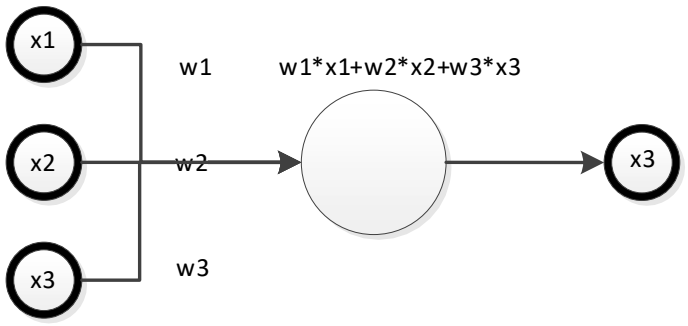


Figure 56 - Ejemplo de red neuronal

La solución al problema de la compuerta XOR con una sola neurona no tiene solución, este problema se conoce desde el año 1969 y demuestra que se debe combinar varias neuronas para resolver este tipo de problemas, en este caso deberemos agregar otra neurona para poder separar los datos.

Diferentes funciones de activación para una neurona

Hay varias funciones de activación para las unidades neuronales y su uso varía con respecto al problema en cuestión y la topología de la red neuronal. En esta sección, vamos a discutir algunas de las funciones de activación relevantes que se utilizan en las redes neuronales artificiales de hoy.

Función de activación lineal.

En una neurona lineal, la salida depende linealmente de sus entradas. Si la neurona recibe tres entradas x1, x2 y x3, entonces la salida y de la neurona lineal viene dada por $$ y = {w}1 {x}1 + {w}2 {x}2 + {w}3 {x}3 + b, donde w1, w2 y w3 son los pesos para la entrada x1, x2 y x3 respectivamente, y b es el sesgo en la unidad neuronal.

Neuronas Sigmoides

Para poder lograr que las redes de neuronas aprendieran solas fue necesario introducir un nuevo tipo de neuronas. Las llamadas Neuronas Sigmoides son similares al perceptrón, pero permiten que las entradas, en vez de ser ceros o unos, puedan tener valores reales como 0,5 o 0,377 o lo que sea. También aparecen las neuronas “bias” que siempre suman 1 en las diversas capas para resolver ciertas situaciones. Ahora las salidas en vez de ser 0 o 1, será d(w.x + b) donde d será la función sigmoide definida como $d(z) = 1/(1 +e^{-z})$. *Esta es la primer función de activación! No lineal*

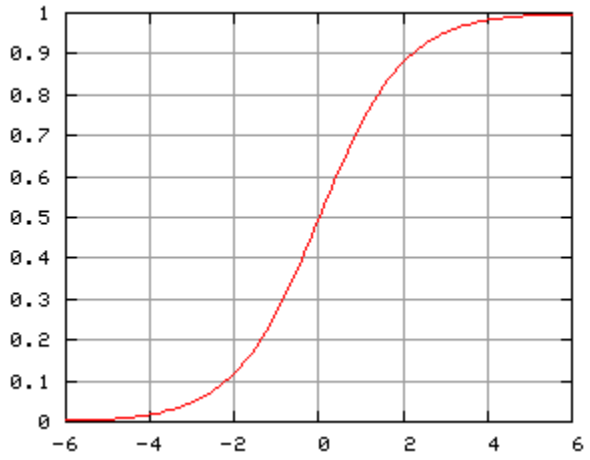


Figure 57 - Imagen de la Curva Logística Normalizada

Con esta nueva fórmula, se puede lograr que **pequeñas alteraciones en valores de los pesos (deltas) produzcan pequeñas alteraciones en la salida**. Por lo tanto, podemos ir ajustando muy de a poco los pesos de las conexiones e ir obteniendo las salidas deseadas.
Existen otras funciones de activación como la Relu que se pueden utilizar para diversos problemas, no entraremos en detalle en este trabajo.

Anexo 3 - Big Data y Data mining.

Los últimos años no solo han mostrado un importante incremento en el número y la complejidad de los datos, sino que se ha modificado la forma de almacenarlos y procesarlos. En la figura 55 se muestra como en solo en 40 años se puede acceder a un dispositivo de almacenamiento por el 7% del valor de un dispositivo en 1980 que provee 1.200.000 veces más de almacenamiento.

Figure 58 - Evolución del almacenamiento

La primera área asociada al manejo de grandes volúmenes de datos es el área de Big Data, luego del primer reléase de apache Hadoop, el cual implementa eficientemente el algoritmo de MapReduce, la cantidad de información manejada en distintas áreas de negocio creció exponencialmente.

Big data lo podemos definir como todo aquello referente al hecho de que los datos se han vuelto tan grandes que no se pueden procesar, almacenar y analizar mediante métodos convencionales.

Como mencionamos en el capítulo anterior en los negocios no siempre las situaciones son tan simples y estos deben adaptarse a cómo, hoy en día se pueden usar los datos para responder a ciertas preguntas. En estos casos puede ser posible que se tengan los datos de clientes en el software CRM que es gestión de relaciones con los clientes y el software CRM puede ser muy básico, puede ser muy sofisticado, depende de lo que tengamos. Además, se tienen en cuentan las redes sociales y sitios web, se puede ver los análisis integrados para cada uno de ellos. Todos estos datos pueden dar varias imágenes individuales de lo que está sucediendo en el negocio.

Pero hay un gran problema con todos esos enfoques: todos los están haciendo. Hay que recordar que, si se está en el mundo de los negocios, se encuentra en un entorno competitivo y necesita una ventaja competitiva. Por lo tanto, es bastante obvio que no se puede simplemente hacer las mismas cosas que todos sus competidores están haciendo, se necesita una ventaja competitiva y aquí es donde intervienen los sistemas de Big Data y minería de datos.

Recordemos que los sistemas de Big Data poseen datos que se caracterizan por su volumen inusual, su alta velocidad y su asombrosa variedad de formatos y fuentes.
Lo que en conjunto significan es que el big data posee el potencial de brindar una ventaja competitiva en las operaciones de un negocio y por eso lo analizamos en este trabajo de tesis. Se pueden destacar algunas cosas diferentes, por ejemplo, se puede identificar nuevos patrones en sus datos, puede encontrar relaciones, puede encontrar tendencias que otras personas no pueden porque puede verlo de una manera más detallada y matizada. Puede usar nuevas fuentes de datos, esta es una célula que conozco algunas compañías locales que hacen un trabajo increíble en experimentos biológicos que conectan cientos de miles por semana, luego usan microscopios para tomar fotos de las células y luego usan **inteligencia artificial** para procesar esas fotos, es una fuente de datos novedosa, son datos de muy alta calidad y en realidad les permite obtener una gran ventaja en el mercado farmacéutico.

Además, el uso inteligente de big data a través de la ciencia de datos, el aprendizaje automático y la inteligencia artificial le permite adaptarse dinámicamente en tiempo real, no solo trimestralmente, mensualmente o anualmente. Le permite identificar nuevas tendencias y nuevos mercados y comenzar a interactuar con ellos de inmediato. Puede convertirse en una organización ágil mediante el uso adecuado de big data. Ahora déjame decir un poco más sobre

cada uno de estos. Con los nuevos patrones, Big Data trae consigo un conjunto completamente nuevo de nuevos algoritmos, nuevas formas de procesamiento.
Pero a veces sus preguntas van más allá y necesita los algoritmos que vienen con la revolución del Big data. Puede enfocarse en casos excepcionales y especialmente porque va a tener muchos datos, recuerde, son grandes datos, va a tener muchos datos y puede enfocarse en esos casos extremos e identificar posiblemente un nuevo nicho que no ha sido aprovechado.

Diferencia entre sistemas convencionales y sistemas basados en big data.

Big data no es solo una versión de gran tamaño de problemas de los sistemas convencionales (aquellos que poseen datos pequeños). Es realmente diferente en varias formas que realmente importan para lo que se está haciendo.
En el libro, Principios de Big Data: preparación, intercambio y análisis de información compleja, Jules Berman describe 10 formas, que Big Data es diferente de los sistemas convencionales: vamos a verlos.

1. Primero, se diferencian en sus objetivos, para sistemas convencionales, el objetivo suele ser un objetivo específico y singular. Estamos tratando de lograr esta tarea analizando los datos. Por otro lado, con Big Data, los objetivos evolucionan y pueden redirigirse con el tiempo. Es posible que tenga uno cuando comience, pero las cosas pueden tomar direcciones inesperadas.

2. El segundo es la ubicación. Los sistemas convencionales generalmente están en un lugar y, a menudo, en un archivo de computadora o en un disquete. Pero en big data son mucho más grandes porque son grandes, y los datos pueden estar distribuidos en múltiples servidores en múltiples ubicaciones en cualquier lugar de Internet.

3. En tercer lugar, la estructura de datos y el contenido. Los sistemas convencionales generalmente se estructuran en una sola tabla, filas y columnas en una tabla, como una hoja de cálculo. Los sistemas de big data, por otro lado, pueden ser semiestructurados o desestructurados en muchas fuentes diferentes, y reunir esas cosas constituye uno de los principales desafíos.

4. La cuarta diferencia es la preparación de datos. Por lo general, el usuario final prepara pequeños datos para sus propios objetivos, de modo que la persona que está ingresando sabe por qué está allí y sabe lo que está tratando de lograr. Por otro lado, el big data es generalmente un deporte de equipo y muchas personas que pueden no ser los usuarios finales o los analistas pueden preparar los datos, por lo que el grado de coordinación que se requiere para los big data es extraordinariamente más avanzado.

5. La quinta diferencia tiene que ver con la longevidad. Los sistemas convencionales se pueden conservar solo durante un período de tiempo limitado una vez que finaliza el proyecto. Quizás unos meses, quizás unos años, pero después de eso, puede desaparecer. Realmente no importa. Los sistemas considerados de big data, por otro lado, pueden almacenar los datos a perpetuidad y en realidad pueden formar parte de proyectos posteriores, por lo que podrían agregarse datos futuros a datos existentes, datos históricos y datos de otras fuentes. Evoluciona con el tiempo.

6. La diferencia de seis tiene que ver con las mediciones. Los sistemas convencionales generalmente se miden en unidades estandarizadas usando un protocolo, generalmente porque una persona lo está haciendo y todo sucede en un punto en el tiempo. Por otro lado, los grandes datos pueden venir en muchos formatos diferentes que se miden en muchas unidades diferentes y se recopilan con muchos protocolos diferentes por diferentes personas y diferentes tiempos y lugares. Y, por lo tanto, no existe un supuesto de estandarización o uniformidad, al menos en la versión sin procesar de los datos.

7. La séptima diferencia tiene que ver con la reproducibilidad. En los sistemas convencionales, los proyectos generalmente se pueden reproducir, si es necesario, si sus datos fallan o si faltan, puede volver a hacerlo. En los círculos científicos, este tipo de replicabilidad en realidad se considera algo muy bueno. Sin embargo, con Big Data, la replicación, lo que significa que reunir todos los datos nuevamente puede no ser posible o factible. Los datos incorrectos pueden tener que identificarse a través de algún proceso forense, y puede haber intentos de reparar directamente las cosas o puede que tenga que prescindir de ellos.
8. La octava diferencia tiene que ver con las apuestas involucradas. En los sistemas convencionales, dado que los proyectos no son enormes, los riesgos generalmente son limitados. Si el proyecto no funciona o si se comete un error, generalmente no es

catastrófico. Pero con los proyectos de Big Data, porque hay tanto tiempo y esfuerzo y el trabajo de las personas invertido en él, los riesgos son enormes. Pueden costar cientos de millones de dólares y la pérdida de datos o los datos incorrectos pueden condenar un proyecto, tal vez incluso la organización que lo ejecuta.

9. La novena diferencia tiene un título peculiar, para aquellos de ustedes que no están en informática, y esa es la introspección. Este es un término que proviene de lenguajes de programación orientados a objetos como C ++, Java, Perl, etc. Tiene que ver con dónde están los datos y cómo identificarlos. Cuando hablamos de sistemas convencionales, generalmente tiene puntos de datos bien organizados e individuales que son fáciles de localizar y, a menudo, tienen metadatos claros para que sepa de dónde provienen. Sabes lo que significan los valores. Sin embargo, con Big Data, tiene muchos archivos diferentes en potencialmente muchos formatos diferentes, y puede ser difícil ubicar los puntos de datos que está buscando y si no están bien documentados y es fácil que las cosas pasen por alto grietas, puede ser difícil interpretar los datos y saber exactamente qué significa cada valor.

10. La décima diferencia tiene que ver con el análisis. para los sistemas convencionales, generalmente se puede analizar todos los datos en un procedimiento en una máquina, pero los sistemas basados en big data, de nuevo, porque son grandes, es posible que los datos deban separarse. Es posible que deba analizarse en varios pasos diferentes utilizando diferentes métodos y luego tratando de combinar los resultados al final. Todas estas diferencias se combinan para hacer que los datos tan grandes no solo sean más grandes. A veces no se trata solo de más datos o muchos datos, la gente lo llama. Es un enfoque cualitativamente diferente para resolver problemas que requiere métodos cualitativamente diferentes y ese es tanto el desafío como la promesa de big data.

Big Data y Data Science

Tengamos en cuenta que, aunque la ciencia de datos se ha asociado fuertemente con big data, aún es posible realizar proyectos de ciencia de datos sin involucrar a todos los elementos de big data. En el trabajo final de esta tesina **vemos como se aplica algoritmos de Machine learning sin incluir elementos de big data.**

Entonces, por ejemplo, puede usar la ciencia de datos en proyectos que tienen volumen sin velocidad o variedad, por ejemplo, un conjunto de datos muy grande y estático con un formato consistente. Los datos genéticos entrarán en esa categoría, o tal vez algo en la minería de datos o el análisis predictivo. Puede estar tratando de predecir un resultado único, como si una persona hará clic en un anuncio en una página web. En este caso, el conjunto de datos puede tener miles de variables y miles de millones de casos, pero todo está en un formato coherente. El tamaño es un desafío, pero de lo contrario, es un análisis relativamente, al menos en teoría, directo. También puede tener velocidad sin volumen o variedad. Entonces, piense en la transmisión de datos que tiene una estructura consistente. Eso podría ser la cantidad de visitas en páginas web o búsquedas de un término en particular, como las búsquedas de big data o data science en la última década en Google Trends. También puede obtener datos de un sensor como lecturas de temperatura que pueden venir cientos de veces por segundo.

Entonces, esto es minería de flujo de datos o, a veces, llamada clasificación en tiempo real de datos del sensor de transmisión. Por lo tanto, debe hacer ciencia de datos para poder procesar la información que llega tan rápido y agregarla, incluso si en última instancia no constituye un gran volumen o si tiene una estructura muy consistente. Y finalmente, hay variedad en los datos sin necesariamente tener velocidad o volumen. Entonces, piense, por ejemplo, en el reconocimiento facial en una colección de fotografías personales. Podrías decir, tal vez tienes unos miles de fotos. Eso no constituye exactamente grandes datos, pero debido a que son fotografías, tiene mucha variedad en los datos. Pero un ejemplo aún mejor es una visualización de datos de un conjunto de datos complejo pero estático. Eso implica el conjunto de herramientas de ciencia de datos, a pesar de que es distinto de Big Data porque se caracteriza solo por una de las tres V. Por lo tanto, la ciencia de datos y el big data tienen mucha historia en común y muchas prácticas en común, pero puedes ver que puedes hacer ciencia de datos, incluso cuando no tienes todos los elementos de Big Data.

No todos los problemas de aprendizaje automático son adecuados para Big Data, y no todos los grandes conjuntos de datos son realmente útiles al entrenar modelos. Sin embargo, su conjunción en situaciones particulares puede conducir a resultados extraordinarios al eliminar muchas limitaciones que a menudo afectan escenarios más pequeños. Desafortunadamente, tanto el aprendizaje automático como los grandes datos son temas sujetos a una exageración continua, por lo tanto, una de las tareas que un ingeniero / científico debe realizar es comprender cuándo una tecnología en particular es realmente útil y cuándo su carga puede ser más pesada que los beneficios. Las computadoras modernas a menudo tienen suficientes recursos para procesar conjuntos de datos que, hace unos años, se consideraban fácilmente un problema de big data.

Las cuatro 'V' de Big Data

- **Volumen**: el universo digital sigue expandiendo sus fronteras.
- **Velocidad**: la velocidad a la que generamos datos es muy elevada, y la proliferación de sensores es un buen ejemplo de ello. Además, los datos en tráfico –datos de vida efímera, pero con un alto valor para el negocio crecen más deprisa que el resto del universo digital.
- **Variedad**: los datos no solo crecen, sino que también cambian su patrón de crecimiento, a la vez que aumenta el contenido desestructurado
- **Valor**: Extraer valor de toda esta información marcará el futuro del manejo de información.

El valor lo podremos encontrar en diferentes formas:

- Mejoras en el rendimiento del negocio
- Segmentación de clientes
- Tomas de decisiones
- Automatización de decisiones tácticas

Tipos de Datos

- Datos estructurados
    - Bases de datos relacionales
- Datos semiestructurados
    - Archivos de texto plano, planillas de cálculo
- Datos no estructurados

- Texto escrito en lenguaje natural
- Contenido multimedia, imágenes, fotos, audio y video

Bibliografía

[1]. Thomas L. Saaty libro “Toma de decisiones para líderes” año??

[2]. Goodfellow,Ian, Bengio, Yoshua, Courville, Aaron (2017). Deep Learning. Editorial MIT Press Ltd.

[3]. Enterprise Application Development with C# 9 and .NET 5 - Ravindra Akella

[4]. Russel y Norvig: Inteligencia Artificial (1994) Un enfoque moderno. Prentice Hall

[5]. Alpaydin, E. (2014). Introduction to Machine Learning. MIT press.

[6]. Hands On - Machine Learning with ML.NET – jarred Capellman

[7]. Flach, P. (2012) “Machine Learning: The Art and Science of Algorithms that Make Sense of Data” Cambridge University Press.

[8]. Russell, S. and Norvig, P. (2010). “Artificial Intelligence: A modern approach”. Third Edition, Prentice Hall.

[9]. Pro Deep Learning with TensorFlow, A Mathematical Approach to Advanced Artificial Intelligence in Python, Santanu Pattanayak, Bangalore, Karnataka, India

[10]. Diapositivas de Conceptos y Aplicaciones de Big Data - Prof. Waldo Hasperué. Mas datos de estasdiapositivas.

Páginas de referencia

- https://oec.world/en/rankings/product/sitc/?year_range=2012-2017
- https://www.bravent.net/las-claves-del-machine-learning-y-las-redes-neuronales
- https://concepto.de/toma-de-decisiones/#ixzz6RvbjaIi6
- https://www.aprendemachinelearning.com/pronostico-de-series-temporales-con-redes-neuronales-en-python/
- https://www.aprendemachinelearning.com/pronostico-de-ventas-redes-neuronales-python-embeddings/

Cursos on-line

- Doing Data Science with Python (Pluralsight)
- Machine Learning. Curso básico de Machine Learning (Udemy) https://www.udemy.com/course/curso-machine-learning/
- coursera.org/learn/machine-learning/home/ (Coursera)
- https://www.lynda.com/IT-tutorials/Big-data-business-strategy/2810941/2247497-4.html