Recapitulación

Alcance

- Repasar los conceptos importantes que hemos visto sobre el área de machine learning.
- Recordar las distintas formas en las que podemos evaluar el desempeño de un modelo.
- Aprender sobre algunas de las técnicas y aplicaciones actuales que están marcando pautas de efectividad en problemas complejos.

Hemos recorrido un largo camino dentro del cual han aprendido una gran variedad de modelos. Iniciamos nuestro aprendizaje con modelos simples como una regresión lineal y terminamos con uno de los modelos más complejos actualmente: redes neuronales.

Como se mencionó durante el curso, no basta aprender a utilizar solo un conjunto pequeño de modelos con los que nos sintamos cómodos, es necesario tener una batería lo suficientemente grande como para ser capaz de tomar decisiones informadas al momento de elegir un modelo o evaluar la efectividad de uno.

Aunque no lo hemos mencionado formalmente, muchos de los contenidos que estudiamos en este curso corresponden a los que formalmente se ven al estudiar la disciplina de Inteligencia Artificial. De hecho, la ahora llamada disciplina de 'Machine Learning' en su gran mayoría no es más que un gran compendio de modelos estadísticos, numéricos y computacionales que ya existían desde hace décadas. Este es un aspecto interesante que está generando ciertos problemas en la actualidad debido a que términos como 'Ingeniería de Datos', 'Data Science', 'Inteligencia Artificial' y 'Machine Learning' aún no están bien delimitados y comparten gran parte de su cuerpo de aplicaciones, como consejo, cuando busquen trabajo en esta área no se limiten a buscar solo por el nombre de la oferta, una buena práctica es basarse simplemente en los requerimientos y habilidades que se piden más que en el solo nombre de la oferta laboral.

Problemas canónicos del Aprendizaje de Máquinas

En Machine Learning existen dos problemas canónicos que hemos tratado de abordar a lo largo de todo el curso. La gran mayoría de los métodos en Machine Learning resuelven alguno de estos (o una variación de los mismos), estos problemas son:

1. **Regresión**: A partir de un conjunto de atributos, ser capaz de predecir valores numéricos.

2. **Clasificación**: A partir de un conjunto de atributos, ser capaz de clasificar la instancia dentro de ciertas clases predefinidas.

Recordar la arquitectura del entorno en el que trabajamos

- **Python**: Lenguaje de programación de alto nivel.

- **Jupyter Notebook**: Aplicación web que permite la creación de documentos de markdown con código incurstado para su ejecución.

- **Matplotlib**: Librería dedicada a graficos.

- **Tensorflow**: Librería dedicada a proveer métodos de trabajo con tensores (estructuras de datos multidimensionales).

- **Keras**: Librería que implementa métodos de alto nivel para poder utilizar el potencial de trabajo con tensores de Tensorflow, específicamente para redes neuronales.

- **Scikit-learn**: Librería que implementa gran variedad de métodos de machine learning, así como también métricas de evaluación de los mismos.

- **Numpy**: Librería dedicada a proveer métodos numéricos y operaciones matriciales.

Cabe destacar que si bien nuestro framework es uno de los dominantes en la industria, existen otras alternativas que vale la pena aprender, tales como:

- `R`
- `Julia`
- `Scala`

Conceptos fundamentales en ML

Conjunto de Entrenamiento

Es el conjunto de datos que contiene los atributos y la variable objetivo instanciada para cada registro, con este conjunto entrenamos nuestros modelos candidatos.

Conjunto de Validación

Puede ser extraído del conjunto de entrenamiento mediante técnicas de muestreo aleatorio y dinámicas como **validación cruzada** o puede ser un conjunto de datos apartado (**Hold-out validation**) al cual el modelo nunca tiene acceso al momento de ingestar datos, sin embargo, lo utiliza para evaluar su desempeño en cada iteración del entrenamiento.

Conjunto de Pruebas

Es un conjunto de datos sobre el cual probaremos nuestro rendimiento final del modelo una vez entrenado, es importante recordar que este conjunto nunca debe ser utilizado para arreglar cualquier imperfección del modelo puesto que perdemos la objetividad.

Función Objetivo

Es la función que se busca optimizar, puede ser minimizada o maximizada dependiendo del problema, su valor depende de los parámetros que el modelo tenga.

Entrenamiento

Es el proceso mediante el cual el optimizador minimiza/maximiza la función objetivo y al hacerlo, encuentra los parámetros del modelo que mejor se ajustan a la tarea que queremos.

Espacio de parámetros

Es el espacio donde se encuentran todos los posibles valores que pueden tomar los parámetros del modelo. Este es el espacio que la máquina debe explorar en busca de la combinación de valores para los parámetros que le entreguen el mejor valor de la función objetivo.

Atributos y variable objetivo

Un atributo es una dimensión de medición, una columna en nuestro dataset. Por ejemplo, la estatura de una persona es un atributo, la edad es otro atributo dentro de una base de datos. La variable objetivo (o target) es la variable que queremos predecir, ésta puede ser numérica (regresión) o categórica (clasificación), por ejemplo, el sexo de una persona (categórica) o la edad de la misma (numérica).

Datos dispersos

Se le dice a un cierto conjunto de datos 'disperso' si presenta gran cantidad de elementos nulos (o casi nulos) y poca cantidad de datos con magnitud mayor a cero, en nuestro caso, esto es más visible cuando observamos vectores de atributos con una gran dimensionalidad pero en los cuales solo algunos elementos tenían valores distintos de cero.

Overfitting (sobre-ajuste)

Fenómeno en el cual nuestro modelo aprendió demasiado bien el conjunto de datos de entrenamiento. Como consecuencia de esto, el modelo generó reglas internas que se apegan demasiado a estos datos y al evaluarlo en datos que nunca antes ha visto (datos nuevos), se comporta mal.

Un ejemplo simple para recordar esta definición es pensar en un mal estudiante que aprende de memoria como hacer los ejercicios de la guía, sin entender en realidad el fenómeno que ocurre por detrás, como es de esperarse, cuando llega el momento de la prueba/certamen, no es capaz de generalizar lo poco que sabe y le va mal. Usualmente evitamos el overfitting implementando técnicas de regularización.

En algunos casos es posible identificar el overfitting fácilmente, por ejemplo, un caso típico de overfitting es una disminución del error de entrenamiento al mismo tiempo que un aumento del error de validación.

Podemos amortiguar el efecto del overfitting en una máquina implementando métodos de regularización.

Underfitting (sub-ajuste)

Fenómeno en el cual nuestro modelo aprendió muy poco sobre el fenómeno subyacente en los datos y no es capaz de generalizar lo que aprendió durante el entrenamiento a datos nuevos. Usualmente evitamos el underfitting implementando técnicas de data augmentation o simplemente recolectando más data.

Optimizador

Es un algoritmo cuyo objetivo es encontrar el mínimo o máximo de una función en particular, en nuestro caso, la función que nos interesa optimizar es la función objetivo, para lograr esto, el optimizador recorre el espacio de parámetros bajo alguna regla matemática o heurística. Los optimizadores que hemos visto en este curso son:

- Stochastic Gradient Descent (SGD).
- Adadelta (learning rate adaptativo).
- Adagrad (learning rate adaptativo).
- Adam (learning rate adaptativo).

Learning rate

Es una medida de qué tan fácilmente el modelo abandonará las hipótesis actuales en pro de nueva información. En nuestro caso, esto se presenta más visiblemente al recordar que el learning rate multiplica al gradiente de la función de pérdida en la actualización de los parámetros, mientras mayor sea el learning rate, mayor será el movimiento a través del espacio.

Overshooting

Es el fenómeno en el cual la tasa de aprendizaje es demasiado grande, provocando que la máquina sea incapaz de llegar al mínimo pues se "pasa de largo". Una forma de identificar este fenómeno es observando las curvas de error de entrenamiento, cuando se produce overshooting esta curva suele ser accidentada, "montañosa" si lo prefieren, debido a que la máquina en ocasiones está en sectores cercanos a un óptimo local pero se pasa y llega a otra zona.

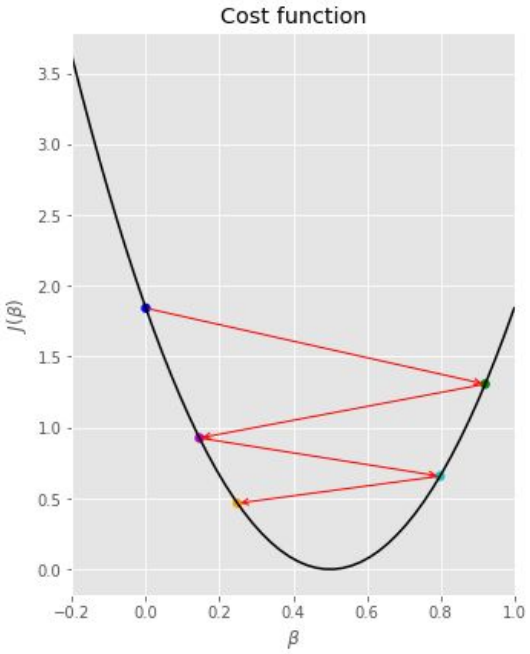

Imagen 1. Cost Function.

Undershooting

Es el fenómeno en el cual la máquina es incapaz de aprender lo suficiente rápido como para abarcar las distancias necesarias en el espacio de parámetros y llegar a óptimos. Es posible identificar el undershooting observando la curva de error de entrenamiento y validación, cuando hay undershooting los errores de entrenamiento y validación presentan muy poca varianza resultando en casi nada de optimización.

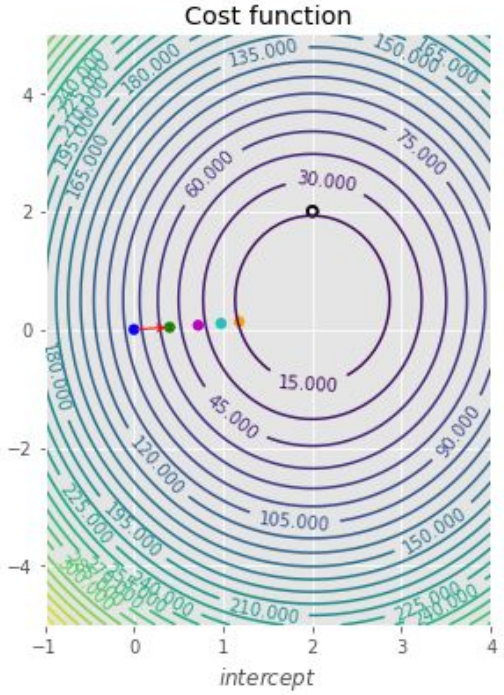

Imagen 2. Cost Function 2.

Hiper Parámetros o Parámetros

Nos referimos a Hiper Parámetros como aquellos valores que dictan características del entrenamiento o entorno del mismo y que **no son capaces de afectar la geografía** del espacio de parámetros. Estos sí pueden, sin embargo, afectar la rapidez con la que lo recorremos o la calidad del recorrido.

Algunos ejemplos de hiper parámetros en los modelos que hemos visto:

- Learning Rate.
- Coeficientes de regularización.
- Coeficiente de Dropout en redes neuronales.

Parámetros son todas aquellas elecciones, numéricas o de arquitectura del modelo, que **si determinan la geografía del espacio de parámetros**, un determinado modelo es potente porque, para un determinado conjunto de datos expresados mediante cierto descriptor es capaz de generar un espacio de parámetros donde las combinaciones numéricas de los parámetros se encuentren en regiones de relativo fácil acceso para un optimizador, o en caso contrario, es capaz de expresar dichas combinaciones en su espacio.

El método que hasta ahora ha demostrado ser el más efectivo es el de búsqueda de hiper parámetros por grilla. En sklearn podemos encontrar este método bajo el nombre de `sklearn.model_selection.GridSearchCV`.

Regularización

Los métodos de regularización consisten en penalizar la función de costo para evitar comportamientos anómalos de la máquina, evitando que esta tienda a soluciones infactibles.

- **L1 (Lasso)**:

$$\beta_{\text{Lasso}} = \underset{\beta}{\text{argmin}} \sum_{i}^{n} (y_i - \hat{y}_i)^2 + \lambda \sum_{j=1}^{p} |\beta_j|$$

- **L2 (Ridge)**:

$$\beta_{\text{Ridge}} = \underset{\beta}{\text{argmin}} \sum_{i}^{n} (y_i - \hat{y}_i)^2 + \lambda \sum_{j=1}^{p} \beta_j^2$$

- **ElasticNet**: Es una combinación de Ridge y Lasso.

$$\beta_{\text{ElasticNet}} = \underset{\beta}{\text{argmin}} \sum_{i}^{n} (y_i - \hat{y}_i)^2 + \lambda_1 \sum_{j=1}^{p} |\beta_j| + \lambda_2 \sum_{j=1}^{p} \beta_j^2$$

Dropout: Consiste en seleccionar al azar, con una cierta probabilidad asignada previa al entrenamiento, ciertos pesos de la red en cero para un determinado instante, no permitiendo que estos aporten al output de la red, esto obliga a la red a buscar variables latentes en otras sub-secciones del espacio de parámetros. Evita que la red se apoye demasiado en ciertos pesos o sub-redes y sobre ajuste demasiado.

- Podemos implementar este método en Keras importando una capa especializada en este método y agregandola al modelo como si fuese una capa más:

```
from keras.layers import Dropout, Dense
# Creamos la capa que queremos regularizar
model.add(Dense(<nº neuronas en la capa>, <otros parámetros>))

# Aplicaremos Dropout con p=0.2 a esta capa específica
model.add(Dropout(0.2, <otros parámetros>))
```

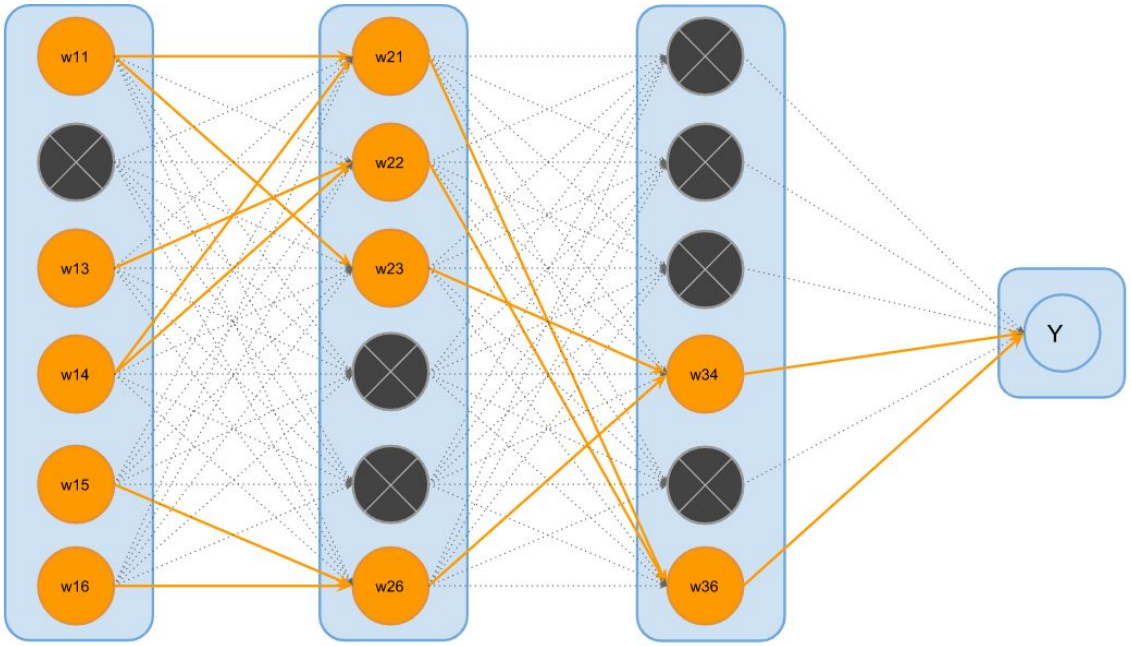

Imagen 3. Dropout.

Algunos modelos vistos durante el módulo

Repasemos algunos de los métodos más importantes vistos:

Regresión Lineal

En este modelo buscamos modelar la variable objetivo como una combinación lineal de los atributos, escalados por un cierto coeficiente, a estos coeficientes β_i se les conoce como coeficientes de regresión y son los parámetros del modelo:

$$y_i = \sum_{j=1}^{p} x_{i,j}\beta_j + \beta_0$$

Modelos Aditivos Generalizados

Estos modelos parten desde el supuesto que la función de regresión es desconocida, tienen la siguiente forma:

$$g(\mathbb{E}(Y)) = \beta_0 + f_1(x_1) + f_2(x_2) + \ldots + f_m(x_m)$$

Donde:

- β_o es el intercepto $g(\cdot)$ es llamada función de vínculo que permite relacionar el valor esperado de la variable objetivo con las variables predictivas x_i $f_i(x_i)$ es el output de la i-ésima función (spline) sobre el i-ésimo atributo.

La potencia de los GAM está en que permiten incorporar distintas familias de funciones para procesar distintos atributos, a diferencia del modelo lineal en el que todos los atributos se trataban bajo una función identidad.

Un aspecto interesante de los GAM es que al ser modelos aditivos es posible estudiar la forma en la que un cierto atributo influye en la determinación de la variable objetivo de forma independiente de los demás atributos:

Imagen 4. Gam.

Naive bayes

Este modelo se basa en el teorema de bayes para estimar la función de probabilidad condicional $P(y|x)$ a partir de la probabilidad conjunta $P(x,y)$.

Este método se basa en encontrar la clase con mayor probabilidad de ser la correcta basándose en la información observada:

$$\hat{y}_{\text{map}} = \underset{y \in \mathbb{Y}}{\operatorname{argmax}}\, \hat{\Pr}(y|X) = \underset{y \in \mathbb{Y}}{\operatorname{argmax}}\, \hat{\Pr}(y) \prod_{1 \le k \le n_d} \Pr(X_k|y)$$

Naive Bayes tiene la ventaja de entregarnos la probabilidad de pertenencia a cada clase dada una observación. Gracias a esto, NB pertenece a la clase de métodos generativos, los cuales pueden ser utilizados en simulaciones debido a su capacidad de generar datos sintéticos a partir de la distribución de $P(y|x)$.

La debilidad de NB está en asumir que todos los atributos X_i son independientes entre sí, lo cual no es necesariamente correcto para todos los fenómenos.

Finalmente, NB al basarse en el teorema de bayes, permite introducir la probabilidad a priori de ocurrencia de ciertas clases.

SVM

Este modelo se encuentra dentro de los más efectivos hasta la fecha, combina varios conceptos interesantes que lo hacen bastante robusto, eficiente y efectivo. Las SVM implementan un concepto conocido como **máximo margen** el cual se basa en encontrar una frontera de clasificación que maximice el margen entre las clases.

$$\underset{\beta,\beta_0,||\beta||=1}{\text{argmax}} \; M$$

$$\text{Subjeto a: } y_i(\mathbf{X}^{\mathrm{T}}\beta + \beta_0) \geq M \qquad \forall i \in N$$

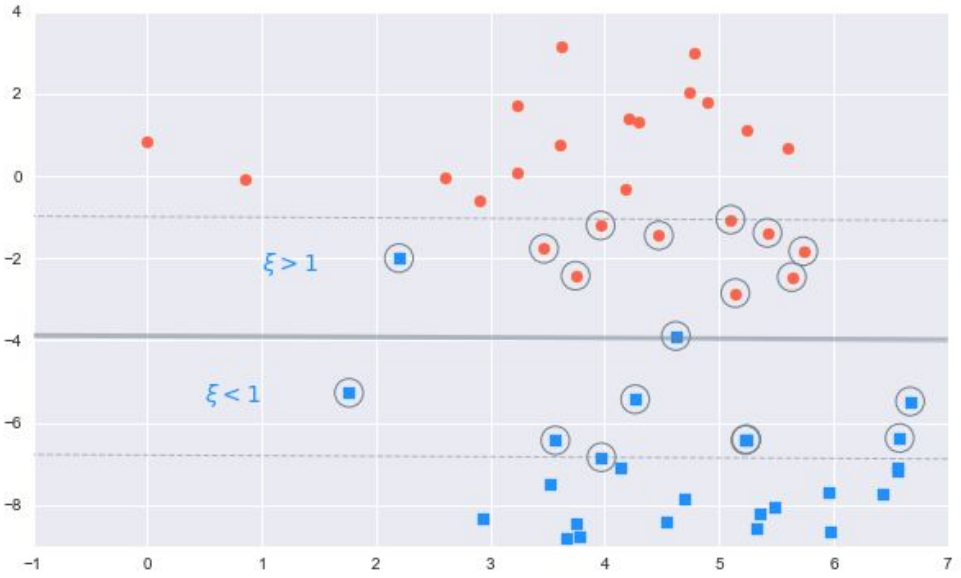

Imagen 5. SVM.

Métodos basados en árboles

Estos métodos se basan en encontrar particiones del espacio muestral que generen las mejores clasificaciones, por lo general basándose en la ganancia de información de forma local con cada partición. Se caracterizan por ser rápidos de computar y tener la enorme ventaja de que son muy interpretables. El árbol de decisión generado es una jerarquía de los atributos que mejor ayudan a predecir la clase de la instancia.

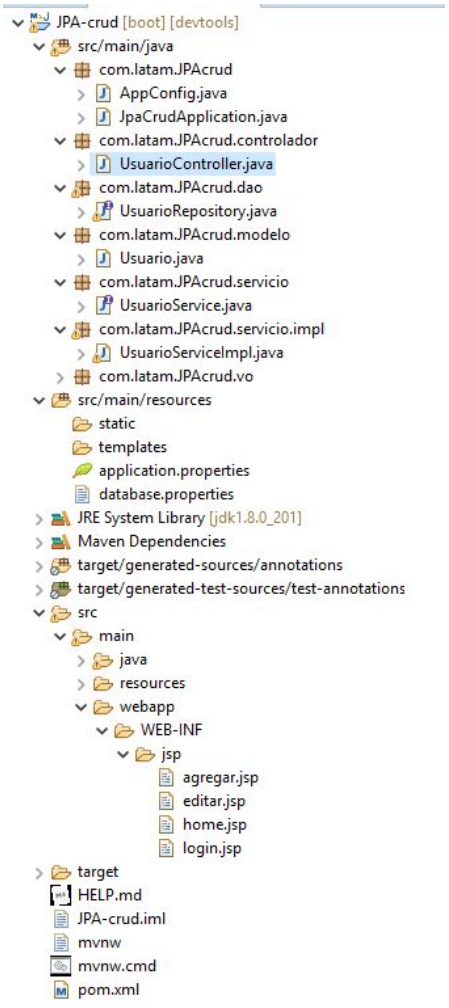

Imagen 6. Tree.

Un contratiempo de estos métodos es que tienden con rapidez hacia el overfitting, haciendo casi obligatorio la implementación de regularizadores y la correcta elección de hiper parámetros (como por ejemplo el tamaño máximo del árbol).

Ensembles

Estos modelos se basan en la idea de juntar una batería de modelos y hacer que resuelvan el mismo problema, con la diferencia que nuestra respuesta final estará basada en el resultado de estos modelos, por ejemplo, implementando un sistema de votación para tareas de clasificación. Los métodos de ensamblado que vimos en el curso son:

- Boosting: Agregar máquinas 'débiles' para que se potencien entre sí y formar máquinas con mejores características basandose en corregir los errores de los modelos anteriores de manera **secuencial**. Un ejemplo de este tipo de métodos es AdaBoost cuyo base learner son árboles de clasificación o GradientBoosting.

- Bagging: Agregar máquinas débiles para que logren, en su conjunto, llegar a una solución buena, para esto genera muestras aleatorias del dataset para cada learner, evitando entrenar a todos los modelos con el mismo dataset. Un ejemplo de este tipo de métodos es el de Random Forest.

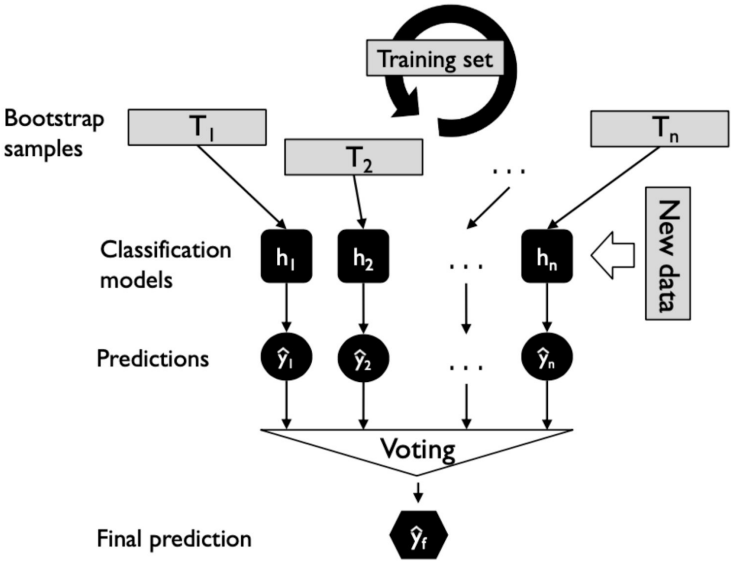

Imagen 7. Prediction.

La diferencia entre bagging y boosting radica en que boosting entrena a todos los clasificadores bajo el mismo conjunto de datos de entrenamiento y hace que el modelo generado en una cierta iteración se centre en corregir los errores del modelo anterior, en otras palabras, boosting entrena un modelo que cuya aproximación al problema es 'tapar agujeros' no cubiertos por el modelo de la iteración anterior. Bagging por otro lado, busca inyectar diversidad mediante el muestreo aleatorio de columnas y filas mediante booststrap para generar muestras aleatorias distintas con las que entrenar cada modelo.

Redes neuronales

Estos modelos se basan en la generación de una arquitectura (en nuestro caso nos remitimos únicamente a feed forward) compuesta de células individuales de procesamiento apiladas en capas. La potencia de estos modelos radica en su gran capacidad de descubrir variables latentes, su desventaja, sin embargo, es que son muy proclives al overfit.

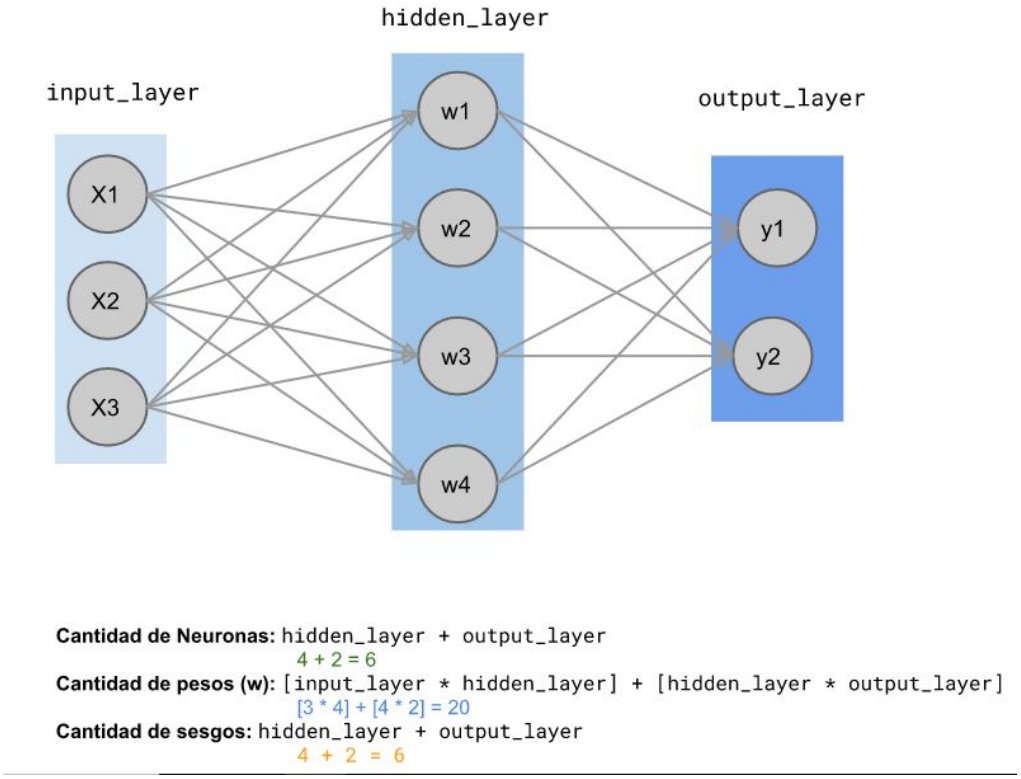

Imagen 8. Neural Minima

Algunos ejemplos de modelos que caen en esta familia de arquitecturas son:

- **Redes Feed-Forward**: Son redes compuestas por una pila de capas completamente conectadas entre sí de manera secuencial, el entrenamiento se hace mediante el método de **backpropagation** y las predicciones mediante **forward propagation**.

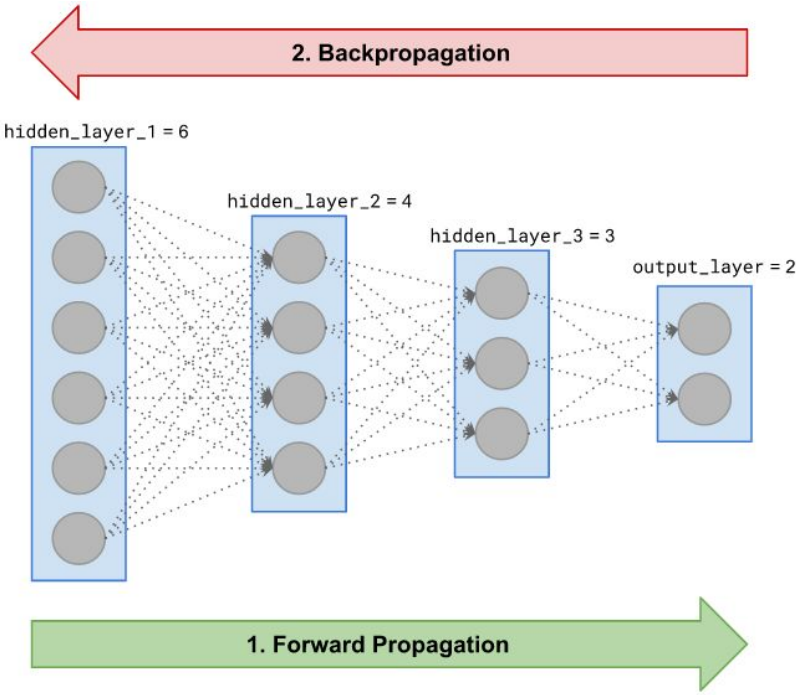

Imagen 9. Forward.

Podemos crear una red neuronal dense feed forward de la siguiente forma en `keras`.

```
from keras.models import Sequential
from keras.layers import Dense, Dropout
# generamos una capa oculta con 20 neuronas
model.add(Dense(20,
  # donde ingresan todos los atributos
  input_shape = (input_dim, ),
  # implementamos un regularizador con norma l2 y lambda=0.01
  kernel_regularizer = ridge_regularizer,
  # imponemos restricción
  kernel_constraint = constraint,
  # implementamos una función de activación ReLU
  activation = 'relu',
  # consideramos una neurona extra que representa el sesgo
  use_bias = True,
  # Le asignamos un nombre
  name = '1ra_capa'))
model.add(Dropout(0.2, name = '1st_dropout'))
```

- **Redes Recurrentes**: Son modelos neuronales creados para modelar secuencias temporales, para lograr esto implementan la posibilidad de conectar capas entre sí de forma temporal entre ciclos de entrenamiento, estas redes tienen ciertas variantes, como LSTM y GRU.

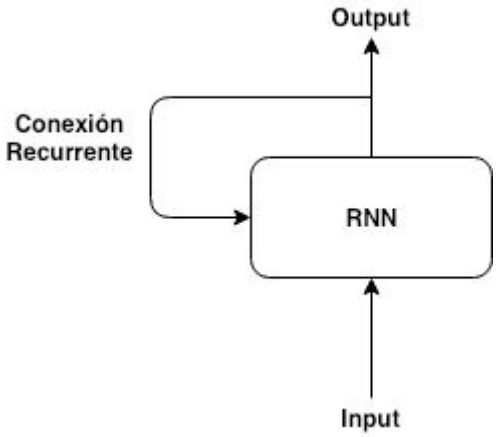

Imagen 10. Conexión Recurrente.

Podemos crear una red recurrente utilizando cualquiera de los tres tipos de capa disponibles en keras para esto:

```
from keras.layers import SimpleRNN, GRU, LSTM
from keras.models import Sequential

model_LSTM = Sequential()
model_LSTM.add(LSTM(4, input_shape = (1,lag),
              activation='tanh',
              recurrent_activation='sigmoid',
             return_sequences = True, name = 'Capa_LSTM'))

model_GRU = Sequential()
model_GRU.add(GRU(4, input_shape = (1,lag),
              activation='tanh',
              recurrent_activation='sigmoid',
             return_sequences = True, name = 'Capa_LSTM'))

model_simpleRNN = Sequential()
model_simplernn.add(SimpleRNN(units = 1,input_shape = (1,lag),
return_sequences = True))
```

Aplicaciones actuales y extensiones útiles

Word2Vec

Para poder trabajar con texto o secuencias de palabras por lo general tenemos que crear un vocabulario, es decir, un repositorio con todas las palabras observadas en el documento, este vocabulario es la representación del texto con el que podremos trabajar. Antes de poder utilizar un vocabulario en un método de machine learning tenemos que generar una codificación del mismo de forma que el modelo sea capaz de entender y leer las palabras en sus ecuaciones. Veamos una aproximación ingenua a este problema:

Ya hemos visto que cuando tenemos variables categóricas en una columna con la que deseamos trabajar una aproximación es utilizar **One-hot vectors** y codificar la ocurencia de cada clase como un determinado patrón de activación en el vector, por ejemplo, para codificar el sexo de las personas podemos utilizar un vactor de dimensión dos: $Masculino = [1, 0]$ y $Femenino = [0, 1]$.

Sin embargo, esta aproximación presenta varios problemas:

- Necesitamos un vector con tantas dimensiones como clases tengamos. Si lo que nos interesa es codificar un vocabulario de palabras del orden de miles de palabras distintas, eso significa utilizar un vector de tantas miles de dimensiones como palabras haya en el vocabulario.

- Esta codificación no permite incluir información sobre el contexto o semántica de las palabras, por ejemplo, las palabras "grande" y "enorme" tienen significados relacionados y que esperaríamos que de cierta forma la codificación les entregase vectores que estén cercanos. One-Hot vector hace todo lo contrario a lo anterior!Todos los vectores creados por one-hot vector son perpendiculares entre sí y por lo tanto **todos están completamente no correlacionados**, la palabra "diminuto" está igual de cerca de "grande" que "enorme".

- Puesto que one-hot vector crea un vector completamente perpendicular a los demás para cada palabra, esto implica que los vectores son dispersos, es decir, hay muy poca información en ellos en comparación con la cantidad de dimensiones que presentan.

Por las razones anteriores, no suele ser buena idea utilizar one-hot vectors para codificar clases que se sabe que tienen cierta correlación o semejanza o para casos en los que la cantidad de clases distintas es muy grande como en el caso del vocabulario de un texto. Es aquí donde recurriremos a otra técnica.

Una de las formas más populares de codificar un vocabulario es mediante **Word embeddings**, estos son representaciones vectoriales de las palabras de un vocabulario.

Un popular y eficiente método de aprender word embeddings para las palabras de un vocabulario es mediante la utilización de una red neuronal poco profunda (recordemos que las capas completamente conectadas son especialmente útiles para encontrar variables latentes). **Word2Vec** es un popular método para generar word embeddings a partir de redes neuronales, fue propuesto por Tomás Mikilov en 2013[1].

CNN

En el tema 15 vimos cómo podemos utilizar redes neuronales recurrentes para poder analizar datos secuenciales mediante el paso de información a través de pasos temporales en el entrenamiento. Otra forma de hacer esto es mediante redes neuronales que se basan en una operación matemática llamada convolución. No entraremos en detalle en cuanto a la definición del operador convolución, sin embargo, podemos quedarnos con la idea de que permite analizar varios datos conjuntamente.

Quizás las redes convolucionales sean más conocidas por su capacidad de analizar imágenes, al ser capaces de observar varios valores a la vez las redes convolucionales 'recorren' una imagen en cuestión a través de una especie de ventana, esto alivia el problema de tener que determinar como particionar la imagen para pasarla al modelo. Actualmente las convolucionales se encuentran en la cabecera de los logros y aplicaciones en el área de computer vision, lamentablemente estas redes generan una cantidad de parámetros considerablemente grande, lo que hace su entrenamiento lento y tedioso, además de eso la construcción de arquitecturas que involucran redes convolucionales es compleja.

Autoencoders

Los autoencoders son un modelo neuronal interesante, se basan en la siguiente idea: Hacer que una red neuronal tenga como objetivo la reconstrucción de su propio input, es decir, los atributos utilizados para entrenar la red son los mismos que la red debe poder predecir. La clave de la aproximación anterior está en obligar a la red a reconstruir dicho input pero a partir de una dimensionalidad menor a la de entrada, consideremos la siguiente red por ejemplo:

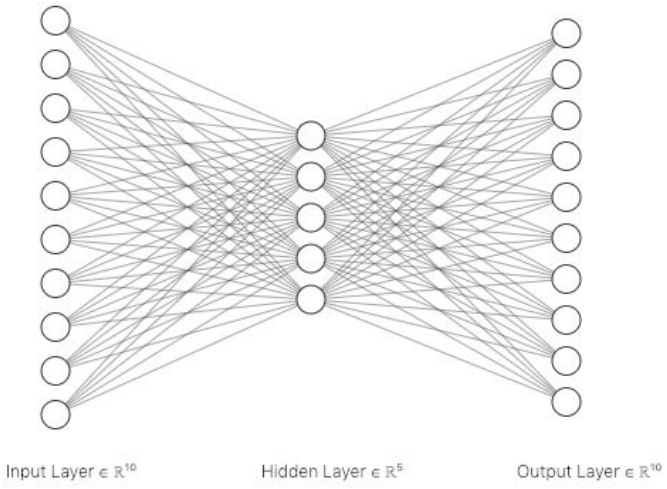

Imagen 11. Input-Output.

Si le indicamos a esta red que su objetivo es el mismo input (de dimensionalidad 10), la red se verá en el problema de reconstruir dicho input a partir de su capa oculta de dimensionalidad 5. Esta *'compresión*' de los atributos obliga a la red a buscar características distintivas de los atributos de forma tal que sea capaz de expresarlos en una dimensionalidad menor.

Los autoencoders (auto-codificadores) pueden ser utilizados para compresión de datos, sin embargo, hasta el momento no parecen superar la eficiencia de otros métodos como PCA. También vale la pena mencionar que son modelos generativos, lo que implica que pueden generar nuevos datos, esta característica hace posible que sean utilizados para reconstrucción de imágenes por ejemplo. La siguiente imagen muestra una reconstrucción de imágenes utilizando autoencoders básicos que disminuyen la dimensionalidad del input original a tan solo 32 dimensiones:

Imagen 12. Numeric.

Cosas que es importante recordar

Datos

≠

Información

Un problema común en las organizaciones de hoy en día es la mala gestión de sus datos para la extracción de información, uno de los motivos principales de esto es la incapacidad de diferenciar entre lo que es un **dato** de lo que es **información**. Los datos carecen de contexto y significado por sí solos, no permiten tomar decisiones informadas, la información por otro lado es el refinamiento de los datos con un objetivo en específico. No importa qué tantas tablas en Excel tenga una empresa, si no implementa los correctos flujos de refinamiento de esos datos le es imposible extraer información real de estos.

No-free-lunch theorem

Ya lo hemos mencionado antes: **Ningún modelo es lo suficientemente potente como para resolver todos o la mayoría de los problemas**, a esto se le conoce como **No-free-lunch theorem** y fue acuñado por David Wollpert y William Macready, este postulado es la razón por la que debemos siempre implementar una batería de modelos para solucionar el mismo problema y decidir en base a los resultados de cada uno con cual quedarnos.

Es un campo de constante avance y cambio

El campo de machine learning, data science, data engineering y deep learning esta en constante avance y cada día se publican nuevas propuestas de modelos y métodos que tratan de solucionar problemas cada vez más complejos, por esta razón, cualquier practicante de esta área debe tener claro que el aprendizaje es algo carente de término. Suscribirse a canales de difusión o leer los papers enviados a conferencias puede ser un buen método para mantenerse al tanto de lo que se está haciendo en el mundo.

Aspectos no cubiertos en el módulo (y que son importantes)

Data engineering

Por muy buen piloto que uno sea, es imposible ganar una carrera sin un buen auto, de la misma forma, aunque es muy tentadora el área de la ciencia de datos es importante entender que la maquinaria que hay detrás y que es necesaria para preprocesar los datos desde su extracción hasta las manos del científico de datos es la parte clave del proceso, en otras palabras, un científico de datos es tan bueno como los datos que tiene en sus manos. Si en una organización no hay políticas de recolección y almacenamiento ad-hoc a las necesidades del negocio es casi imposible extraer información valiosa de los datos. Actualmente a quienes se dedican a confeccionar y orquestar la distribución de datos a lo largo de la organización se les está llamando 'Data Engineers' y todo indica que en un futuro van a ser tanto o más cotizados que los data scientist.

High Performance Computing

Un patrón común a lo largo de este curso ha sido la utilización de datasets con "grandes" cantidades de datos, o más específicamente, con una cantidad de registros del orden de los miles, esto no es casualidad, aunque los algoritmos vistos son potentes todos tienen la gran debilidad de depender de la calidad y cantidad de datos. Sobre esto, en las organizaciones se acostumbra a almacenar toda la información referente a las operaciones realizadas y procesos internos, resultando en una enorme cantidad de datos los cuales resulta casi imposible tratar en entornos que no sean especializados, en palabras simples, hay un límite para lo que podemos abrir en nuestro computador de escritorio, más allá de eso entramos en el terreno de la computación distribuida y High Performance Computing.

¿Cómo recuperar, almacenar y servir los datos?

En las organizaciones usualmente se acostumbra a almacenar grandes cantidades de información proveniente de distintas fuentes, esta información no puede ser accedida por los analistas en su forma bruta pues los requerimientos técnicos se escapan muchas veces de sus curriculums, es entonces cuando se deben implementar políticas de ingesta de datos y procesos de extracción (extract), carga (load) y transformación (transform) de los datos para asergurar que tengan la granularidad necesaria, esten libres de incosistencias, respondan a las necesidades del análisis y sean de fácil y rápido acceso, a este proceso se le conoce como ETL.

¿Qué pasa cuando hay demasiados datos?

Imagine que tiene 1000 imágenes de 500x500 px de una misma sección del espacio observada durante un tiempo. Una de las dificultades de encontrar quasares es que a veces están apagados y a veces están iluminados, no conocemos su ciclo ni la distancia si aún no hemos visto el quasar, por lo que encontrar quasares nuevos es una tarea no sencilla. Una forma de encontrar quasares es calculando la mediana de cada pixel, al final del proceso, la imagen resultante mostrará claramente aquellos puntos brillantes donde se encuentren quasares.

El problema es que para saber si hay algún quasar en esa región del espacio tenemos que calcular la mediana de **500x500 = 250000** muestras (cada pixel), cada una de tamaño **1000**, puesto que la mediana requiere que la muestra esté ordenada el problema es ordenar esas **250000** muestras para recién ver si en esa región del espacio existe algo interesante.

Este problema muestra que si bien, tener muchos datos es algo por lo general deseable, viene a un alto costo. Si no se tiene métodos numéricos eficientes y arquitecturas que puedan soportar ese tipo de cálculos nos veremos en problemas. Si alguna vez se encuentra con un conjunto de datos demasiado grande para su entorno, estudie la posibilidad de muestrearlo aleatoriamente y trabaje sobre una muestra reducida, siempre y cuando el tamaño muestral sea lo suficientemente grande como para ver convergencia de las distribuciones sus análisis probablemente serán correctos.

¿Qué pasa cuando hay muy pocos datos?

No siempre se puede recolectar la cantidad de datos necesaria para poder aplicar los métodos que hemos mencionado, un ejemplo de esto es por ejemplo el estudio que realiza la NASA sobre sus lanzamientos espaciales. Es tan caro el proceso de generar una instancia de la muestra (un lanzamiento) que pensar en calcular cosas simples estadísticamente comienza a ser complicado, ni hablar de utilizar modelos como los vistos en ese tipo de casos con datasets reducidos. Si alguna vez se encuentra en esta situación, primero estudie la factibilidad de recuperar más datos desde la fuente, también estudié imbalances entre clases y métodos estadísticos para trabajar bajo este tipo de situaciones.

Referencias

- [1] Distributed Representations of Words and Phrases and their Compositionality .- T. Mikilov et al, Google inc. - 2013.

Bibliografía relevante utilizada en el curso

- Deep Learning with python.- Francois Chollet.
- Deep Learning: A practitioner's approach.- J. Patterson y A. Gibbson.
- Data Science from scratch.- J. Grus.
- Elements of statistical learning.- T. Hastie, R. Tibshirani y J. Friedman.
- Deep Learning.- I. Goodfellow, Y. Bengio y A. Courville.