UNIVERSIDAD AUTÓNOMA DE NUEVO LEÓN

Facultad de Ciencias Físico Matemáticas

Licenciatura en Ciencias Computacionales

Introducción al Aprendizaje Automático

Maestro: Angel Adrian Dominguez Lozano

“Tarea 3 - Entregable 1”

Nombre	Matricula
Sebastian Terrazas Santillana	1847317
Susana Enriquez Godina	1841762
Mario Eduardo Lara Loredo	1844214
Humberto Gerardo Peña Páez	1862464

San Nicolás de los Garza, Nuevo León. 6 de octubre del 2021.

1. Métodos de ensamble

Un método de ensamblaje es un conjunto de modelos de Machine Learning, donde cada modelo trabaja de una forma diferente, para que en conjunto puedan resolver un solo problema. Si bien, cada modelo realiza una predicción diferente, decir que son totalmente independientes es un error, ya que los modelos suelen estar relacionados y debido a esto, suelen compensar sus errores, proporcionando estabilidad y una mejor predicción general.

1. *Random Forest* es un algoritmo que utiliza un conjunto de árboles de decisión para tomar decisiones, este conjunto de árboles forman un bosque. Cada árbol del bosque funciona de manera individual, ya que tienen una selección de valores de características diferentes, al igual que una cantidad de muestras aleatorias, de aquí viene el nombre de Random Forest. Los distintos árboles otorgan diferentes predicciones. Finalmente la predicción con mayor cantidad de votos es elegida como la correcta.

    1. *Bagging* es una técnica que se usa para reducir la varianza en un conjunto de datos. Para lograr esto cada modelo selecciona un subconjunto de datos y es entrenado con este, logrando con esto una predicción con una estimación más precisa.

    2. El hiper parámetro más importante en el algoritmo de Bagging es el número de árboles de decisión a usar. Por cada árbol que aumentamos en el algoritmo, esto hace que se estabilice cada vez más el modelo. Si bien el aumento de árboles no provoca un sobreajuste, después de cierta cantidad de árboles, el rendimiento se estanca y no se consigue ninguna mejora.

Otro hiper parámetro con el que se puede jugar es el tamaño de las muestras. Si bien, ya se selecciona un subconjunto de datos para mejorar la varianza, se puede reducir aún más el tamaño de la muestra para aumentar la varianza de los resultados y obtener un mejor rendimiento.

2. *Gradient Boosting* es un método que mejora los modelos con cada predicción que realiza, ya que construye nuevos modelos, en base a predicciones, hechas por los errores cometidos por estimaciones anteriores. El fin del método sería el de minimizar el error de predicción.
    1. Para que el algoritmo de Gradient Boosting funcione, se requiere de un *Boosting*. Comienza creando un modelo inicial a los datos, luego se crea otro segundo modelo que intenta predecir en lo que se equivocó el primero modelo, y continúa creando modelos, con el fin de compensar las deficiencias del anterior modelo.

2. Xgboost

Extreme Gradient Boosting es uno de los algoritmos de machine learning supervisado más utilizado en la actualidad. Se caracteriza por obtener resultados de predicción buenos con muy poco esfuerzo computacional.

La idea detrás del boosting es generar múltiples modelos de predicción “débiles” uno detrás de otro y que cada uno de estos tomé los resultados del anterior, para obtener un modelo más “fuerte” con mejores y estables resultados. Este proceso se repite hasta llegar a un número en el que la diferencia entre resultados de modelos es tan pequeña que insignificante o en consecuencia cuando se llega al número de iteraciones máximas definidas por el usuario. El algoritmo utiliza como sus modelos débiles árboles de decisión de diferentes tipos, que pueden ser usados para tareas de clasificación y de regresión.

1. Regularización L1 y L2

La regularización en este algoritmo aplicado en árboles de regresión potenciados por gradientes se aplica a los valores de las hojas y no a los coeficientes de características. Las regularizaciones L1 y L2 son penalizaciones por hojas residuales.

Cuando usamos regularización minimizamos la complejidad del modelo a la vez que minimizamos costes.

a. Regularización L1.

También conocida como la regularización Lasso. Se puede aplicar a regresiones lineales, polinómicas, regresión logística, redes neuronales, máquinas de vectores de soporte y más. La complejidad C se mide como la media del valor absoluto de los coeficientes del modelo, quedando como:

$$C = \frac{1}{N} \sum_{j=1}^{N} |w_i|$$

Se utiliza cuando se sospecha que varios atributos de entrada son irrelevantes, ya que al usar esta penalización estamos haciendo que la solución sea poco densa. Funciona mejor cuando los atributos no están muy correlados entre ellos.

b. Regularización L2.

También conocida como regularización Lasso. Se puede aplicar con varios métodos de aprendizaje automático. Aquí la complejidad C se mide como la media del cuadrado de los coeficientes del modelo, quedando como:

$$C = \frac{1}{2N} \sum_{j=1}^{N} w_i^2$$

Se utiliza cuando se sospecha que varios atributos de entradas estan correlados entre ellos. Hace que los coeficientes acaben siendo más pequeños por lo que la disminución de los coeficientes minimiza el efecto de la correlación entre los atributos de entrada y hace que el modelo generalice mejor. Funciona mejor cuando la mayoría de atributos son relevantes.

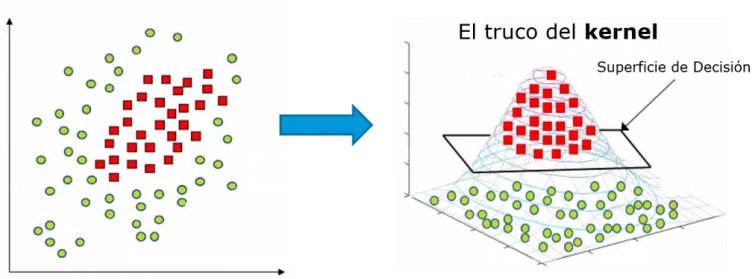
3. SVM

También conocido como (máquina de vectores de soporte) es un método de clasificación y regresión que aprovecha la precisión de las predicciones de un modelo sin ajustar excesivamente los datos de entrenamiento. Es ideal para analizar datos con un gran número de campos.

El método funciona correlacionando datos a un espacio de características de grandes dimensiones de forma que los puntos se categoricen. Se detecta un separador entre las categorías y los datos se transforman de forma que el separador se puede obtener como un hiperplano y los nuevos datos se pueden utilizar para predecir al grupo al que se pertenece.

1. Truco del kernel

Hay veces en las que no se encuentra la forma de encontrar un hiperplano que nos permite separar dos clases, en estos casos se dice que las clases no son linealmente separables, pero podemos utilizar el truco del kernel para resolver este problema. Consiste en inventar una nueva dimensión en la que podamos encontrar un hiperplano para separar clases.



2. Tipos de kernel

Entre los kernels más populares para usar con SVM se encuentran:

a. Kernel Lineal. Este cuantifica la similitud de un par de observaciones usando la correlación de Pearson. El clasificador obtenido es equivalente SVC.

$$K(x_i, x_{i'}) = \sum_{j=1}^{p} x_{ij} x_{i'j}$$

b. Kernel polinomico. Un kernel polinómico de grado d (siendo d>1)permite un límite de decisión mucho más flexible. Cuando un SVC se combina con un kernel no lineal, se obtiene un SVM.

$$K(x_i, x_{i'}) = (1 + \sum_{j=1}^{p} x_{ij} x_{i'j})^d$$

c. Kernel radial. Donde γ es una constante positiva que cuanto mayor sea, mayor la flexibilidad del SVM. Suponiendo que una observación de test x∗=x1∗...xp∗ se encuentra alejada de una observación de entrenamiento xi en términos de distancia Euclídea, entonces K=x∗,xi será muy pequeño, lo que significa que xi no influirá en f(x∗). Tiene un comportamiento muy local, en el sentido de que solo las observaciones de entrenamiento cercanas a una observación de test tendrán efecto sobre su clasificación.

$$K(x_i, x_{i'}) = \exp(-\gamma \sum_{j=1}^{p} (x_{ij} - x_{i'j})^2)$$

4. Redes neuronales

Las redes neuronales son concatenación de funciones, reciben su nombre ya que estas tratan de simular el modo en el que el cerebro humano funciona (o bien las conexiones y el proceso de aprendizaje de las redes neuronales humanas).

Normalmente se conforman de 3 capas; la primera es la capa de entrada o input, la 2 (pueden ser más) las capas ocultas y por último una capa de salida o output.

Y básicamente funciona de la siguiente manera: la red genera predicciones basándose en los datos de cada una de las “neuronas” y cuando una de sus predicciones es correcta entonces actualiza su información, se puede decir que aprende y así sigue mejorando conforme pase el tiempo y tenga más datos.

1. Un *perceptrón* es la red neuronal más simple que se utiliza cuando los elementos de distintas categorías se pueden separar en una sola línea sin que se mezclen.

Funciona de una manera muy sencilla, primero lee todos los inputs, después los suma y por último introduce el resultado en una función de activación para así poder obtener el resultado final.

2. Una *función de activación* se encarga de filtrar o limitar la salida que obtenga una neurona, es decir está la modifica para que el resultado esté en los límites que tenemos. Cada capa de las redes neuronales utiliza una función de activación que nos ayuda con las predicciones.

3. Existen diferentes *tipos de funciones de activación* por ejemplo; la función sigmoide tiene una escala de 0 y 1, entonces cuando tiene un valor muy bajo lo convierte en cero y cuando tiene un valor alto lo convierte en uno. Otro ejemplo podría ser la función de unidad lineal rectificada la cual anula los valores negativos que recibe pero los positivos no los modifica.

4. Otro concepto importante en las redes neuronales es el *Deep Learning,* el cual es el estudio de las redes neuronales que contienen muchas capas ocultas ya que estas son más complejas pero por ende obtienen mejores resultados porque conforme van avanzando entre capas se obtienen mejores resultados, más precisos lo que ocasiona una mejor salida.

Fuentes de consulta para ampliar el contexto:

Métodos de ensamble

→ Brownlee, J. (27 de Abril de 2020). How to Develop a Bagging Ensemble with Python. Recuperado el 3 de Octubre de 2021, de Machine Learning Mastery: https://machinelearningmastery.com/bagging-ensemble-with-python/

→ Gonzales, L. (29 de Marzo de 2019). Métodos de Ensamble de Modelos. Recuperado el 03 de Octubre de 2021, de aprendeIA: https://aprendeia.com/metodos-de-ensamble-de-modelos-machine-learning-ensemble-methods-en-espanol/

→ Hoare, J. (5 de Junio de 2017). Gradient Boosting Explained – The Coolest Kid on The Machine Learning Block. Recuperado el 3 de Octubre de 2021, de Displayr: https://www.displayr.com/gradient-boosting-the-coolest-kid-on-the-machine-learning-block/

→ IBM Cloud Education. (26 de Mayo de 2021). Boosting. Recuperado el 3 de Octubre de 2021, de IBM: https://www.ibm.com/cloud/learn/boosting

→ Priyankur, S. (2019 de Octubre de 14). Bagging and Random Forest in Machine Learning. Recuperado el 3 de Octubre de 2021, de KnowledgeHut: https://www.knowledgehut.com/blog/data-science/bagging-and-random-forest-in-machine-learning

→ Yiu, T. (12 de Junio de 2019). Understanding Random Forest. Recuperado el 3 de Octubre de 2021, de towards data science: https://towardsdatascience.com/understanding-random-forest-58381e0602d2

Redes neuronales

→ IBM Docs. (2021, August 17). Ibm.com. https://www.ibm.com/docs/es/spss-modeler/SaaS?topic=networks-neural-model

→ AI, B. (2019, November 14). Redes neuronales - Bootcamp AI - Medium. Medium; Medium. https://bootcampai.medium.com/redes-neuronales-13349dd1a5bb

→ Ximena Islas. (2021). ¿Qué es un perceptrón? La red neuronal artificial más antigua. Https://Www.crehana.com; Crehana. https://www.crehana.com/mx/blog/desarrollo-web/que-es-perceptron-algoritmo/

→ Calvo, D. (2018, December 7). Función de activación - Redes neuronales - Diego Calvo. Diego Calvo. https://www.diegocalvo.es/funcion-de-activacion-redes-neuronales/

→ Enzyme Advising Group. (2012). Redes Neuronales Artificiales y Deep Learning, explicado para dummies. Enzymeadvisinggroup.com. https://blog.enzymeadvisinggroup.com/redes-neuronales-artificiales-y-deep-learning

XGBoost

➔ “Regularización Lasso L1, Ridg L2 y ElasticNet”, Jose Martinez Heras, consultado en: www.iartificial.net.
➔ “XGBoost en Python”, Juan Bosco Mendoza, consultado en: medium.com
➔ “L1, L2 Regularization in XGBoost Regression”, Albert Um, consultado en:medium.com.
➔ ”Algortitmo XGBoost”, consultado en: programmerclick.com.

SVM

➔ “Máquinas de Vector Soporte”, Cristina Gil Martinez, consultado en: rpubs.com.
➔ “Máquinas de Vectores de Soporte (SVM)”, Jose Martinez Heras, consultado en : www.iartificial.net.
➔ “Modelos de máquina de vectores de soporte”, consultado en www.ibm.com.