Comparación de Algoritmos de Machine Learning problema del Titanic

Machine Learning Algorithms Comparison

Autor 1: Alejandro Osorio Trujillo Autor 2: Brayan Gómez Betacnur, Autor 3: Daniel Eduardo Saldarriaga Velásquez, Autor 4: Hernán Mauricio Zuluaga Bucheli
Universidad Tecnológica de Pereira
Correos: a.osorio@utp.edu.co, bryan2621@utp.edu.co, daniedusal@utp.edu.co, herzulu@utp.edu.co

***Resumen*— El Machine Learning es un área de las ciencias de la computación que permite, por medio del aprendizaje basado en el ser humano, que una máquina tome decisiones con respecto a sus "conocimientos" en un momento dado.**

Esta toma de decisiones es útil para resolver problemas del mundo real, como lo es el que nos compete: descubrir quiénes sobrevivieron a la tragedia del hundimiento del Titanic; los datos para predecir este suceso son la información de los pasajeros de la embarcación en cuestión, el propósito de los modelos de Machine Learning será descubrir qué variables son más importantes y hacer predicciones de la mortalidad de los pasajeros. Todo esto con el propósito de disminuir la tasa de mortalidad que se presenta en este tipo de eventos.

Se seleccionaron cinco (5) modelos o métodos que se postularon eficientes para entrenar y predecir, éstos son: Regresión Logística, Random Forest, Máquinas de Vectores de Soporte, K Vecinos, Random Forest y Descenso de Gradiente Estocástico, cada método tiene sus fortalezas y debilidades por lo que será interesante compararlos entre sí.

De cada algoritmo se computó el tiempo de ejecución, así como exactitud, precisión, sensibilidad, especificidad, f1 score, etc. Con estos valores se realizará un proceso comparativo para así determinar cuál algoritmo es mejor sobre este problema en específico.

***Palabras clave*— Machine Learning, Inteligencia Artificial, Algoritmo, Comparativa, Regresión Logística, Random Forest, Máquinas de Vectores de Soporte, K Vecinos más Cercanos, Random Forest, Descenso de Gradiente Estocástico, Sobrecarga, Parámetro de Regularización, Métrica.**

***Abstract*— Machine Learning is an area of computer science that allows, through human-based learning, that a machine makes decisions regarding its "knowledge" at a given time.**

This decision-making is useful for solving real-world problems, such as the one that competes us: discovering who survived the tragedy of the sinking of the Titanic; The data to predict this event is the information of the passengers of the vessel in question, the purpose of the Machine Learning models will be to discover which variables are most important and make predictions of the mortality of the passengers.

Five (5) models or methods were selected to train and predict, options are: Logistic Regression, Random Forest, Support Vector Machines, K Neighbors, Random Forest and Stochastic Gradient Descent, each method has its strengths and weaknesses so that it will be interesting to compare them with each other.

The execution time, as well as accuracy, precision, sensitivity, specificity, f1 score, etc., were computed for each algorithm. With these values, a comparative process will be executed to determine which is the best algorithm on this specific problem.

***Key Words* — Machine Learning, Artificial Intelligence, Algorithm, Comparative, Logistic Regression, Random Forest, Support Vector Machines, K Nearest Neighbors, Random Forest, Stochastic Gradient Descent, Overfitting, Regularization, Metrics.**

I. INTRODUCCIÓN

Con el fin de practicar lo aprendido durante el curso de Machine Learning, el propósito de este artículo es mostrar los resultados obtenidos de la predicción y clasificación de un problema en particular, el problema en cuestión es un conjunto de datos de tripulantes de la famosa embarcación Titanic.

El dataset contiene los detalles de un subconjunto de los pasajeros a bordo (891, para ser exactos), lo que es más importante, revela si sobrevivieron o no a dicha tragedia. Algunos datos están perdidos, por lo que se realizó un proceso de limpieza y estandarización de los datos.

La importancia de clasificar si un pasajero sobrevivió o no es poder evaluar sus características para así determinar quiénes tienen más riesgo en caso de una emergencia y poder actuar de manera acorde, y en el proceso, salvar vidas.

Fueron utilizadas las librerías pandas y scikit-learn como base para resolver el problema (El problema es predecir si un tripulante, con sus características únicas dentro del barco, sobrevivió o no al hundimiento) y por último, la librería matplotlib para evidenciar de una mejor manera los datos de forma gráfica. Todo esto tomando como base el lenguaje de programación Python.

Así pues, inicialmente se muestra una breve explicación de cada uno de los campos del dataset:

- Passenger Id: Id único de cada pasajero.

- Survived: Determina si un pasajero sobrevivió (1) o no (0). Este será nuestro conjunto y
- Pclass: Clase del tiquete.
- Name: Nombre del pasajero.
- Sex: Sexo del pasajero.
- Age: Edad del pasajero.
- SibSp: # de hermanas / cónyuges a bordo del Titanic.
- Parch: # de padres / hijos a bordo del Titanic.
- Ticket: Número de tiquete.
- Fare: Valor pagado por el tiquete.
- Cabin: Indica la cubierta donde estaba ubicado el pasajero.
- Embarked: Puerto de embarcación al Titanic.

Los métodos o modelos utilizados para resolver el problema son los siguientes:

- Regresión logística.
- Máquinas de Soporte Vectorial(SVM).
- K Vecinos más Cercanos(KNN).
- Random Forest.
- Descenso de Gradiente Estoćastico (SGD)

II. MÉTODOS

Regresión Logística
El modelo de regresión logística es una técnica de estadística multivariable, que permite calcular el valor de la relación entre una variable dependiente en este caso binaria, y un conjunto de variables dependientes.
Se utilizó para predecir las salidas en un momento de tiempo específico.
En la aplicación del algoritmo de regresión logística en la base de datos, con las variables independientes más importantes que son el precio del tiquete y la edad para así poder predecir qué tanto están relacionadas las variables con respecto a saber si la persona muere o no muere.
Otro concepto importante es el de kernel, el kernel aparece cuando no es posible encontrar un hiperplano que permita separar dos clases. Cuando se presente esta situación el kernel inventa una nueva dimensión en la que sea posible encontrar un hiperplano para separar las clases.

Máquina de Vectores de Soporte
El modelo Support Vector Machine es generalmente utilizado para problemas de clasificación, la máquina (Que viene de **Machine** Learning) de Vectores de Soporte consiste en la utilización de vectores de soporte, valga la redundancia, para hacer la separación de las clases deseadas. 5
Como generalmente los datos presentan ruido, o que no están etiquetados perfectamente, como lo es el problema que se trata aquí, o que incluso es muy complicado clasificarlos correctamente. Para estos casos, se le dice al algoritmo que se prefiere que se generalice bien para la mayoría de los casos, aunque algunos pocos casos del conjunto de entrenamiento no vayan a estar perfectamente clasificados.

K Vecinos más Cercanos
El modelo K-Nearest-Neighbor es un algoritmo basado en instancia de tipo supervisado de Machine Learning. Puede usarse para clasificar nuevas muestras (valores discretos) o para predecir (regresión, valores continuos) en esta ocasión se toma un pequeño grupo de los datos con todos sus campos para posteriormente tratar de clasificar (0 no sobrevive, 1 sobrevive) el nuevo conjunto de datos que se agrupó. Se debe tener en cuenta a la hora de aplicar este modelo las características de cada CPU donde se pretende implementar ya que puede ser un limitante a la hora de realizar pruebas de entrenamiento con este ya que si se posee un dataset con un conjunto de datos muy extenso, el algoritmo tiende a consumir mucha CPU, ya que este calcula punto por punto las salidas por lo cual se recomienda hacer un agrupación teniendo en cuenta esto.

Random Forest
Es una combinación de árboles predictores en donde cada árbol depende de los valores de un vector aleatorio probado de manera independiente y con la misma distribución para cada uno de ellos. Es una modificación sustancial de bagging que construye una larga colección de árboles no correlacionados y luego los promedia.

La idea esencial del bagging es promediar muchos modelos ruidosos, pero aproximadamente imparciales, y por tanto reducir la variación. Los árboles son los candidatos ideales para el bagging, dado que ellos pueden registrar estructuras de interacción compleja en los datos, y si crecen suficientemente profundo, tienen relativamente baja parcialidad. Producto de que los árboles son notoriamente ruidosos, ellos se benefician enormemente al promediar.

Algunas características de Random Forest:

- Random Forest se considera como la “panacea” en todos los problemas de ciencia de datos.
- Útil para regresión y clasificación.
- Sirve como una técnica para la reducción de la dimensionalidad.
- Se generan múltiples árboles.
- Cada árbol da una clasificación (Cota por una clase). Y el resultado es la clase con mayor número de votos en todo el bosque (Forest).
- Para regresión, se toma el promedio de las salidas de todos los árboles.

Descenso de Gradiente Estocástico
El modelo Stochastic Gradient Descent, o como lo dicen sus siglas SGD, es un método iterativo para optimizar una función objetivo con propiedades de suavidad adecuadas. Puede ser considerado como una aproximación estocástica de la optimización del descenso del gradiente, ya que reemplaza el

gradiente real (Se calcula a partir de todo el conjunto de datos) por una estimación del mismo (Calculado a partir de un subconjunto de datos seleccionado al azar). 1

El gradiente descendente es una técnica de optimización que puede ser usada con casi cualquier algoritmo de Machine Learning, siendo el gradiente la pendiente de una función. Lo que hace esta técnica es medir el grado del cambio de una variable en respuesta a los cambios en otra variable; como se había mencionado anteriormente, el descenso gradiente estocástico toma aleatoriamente unos datos y no todo el conjunto de los datos y calcula, en cada iteración, el gradiente.

De esta manera, el descenso de gradiente estocástico reduce el costo computacional al no utilizar la totalidad de los datos sino una muestra aleatoria. Aunque el camino tomado por este algoritmo para alcanzar un mínimo tiene usualmente más ruido que el algoritmo de descenso de gradiente típico, esto generalmente no importa mucho puesto que de igual manera se llega a un mínimo con un tiempo de entrenamiento significativamente menor.

III. EXPERIMENTOS Y RESULTADOS

El experimento que se llevó a cabo tuvo en cuenta el conjunto de algoritmos y el conjunto de datos o dataset. La secuencia de pasos fue hacer inicialmente la división de los datos en datos de entrenamiento (X_train, y_train) y datos de pruebas (X_test, y_test), este proceso se hizo con la función train_test_split propia de la librería scikit-learn, esta función toma los datos y los divide, en nuestro caso: 70% para datos de entrenamiento y 30% para datos de prueba. Y de igual manera se definió que los datos no fueran mezclados de forma aleatoria, siguiendo el proceso que se hizo en clase.

Luego, para cada modelo, sigue el entrenamiento con los conjuntos de datos X_train y una salida y_train, se hace validación de los resultados y se hacen las pruebas pertinentes con datos a predecir y_pred y su valor verdadero y_test.

Posteriormente, se comparan distintas métricas entre los diferentes algoritmos como lo son el tiempo de ejecución, así como también valores referentes a problemas de clasificación binaria, que provienen de la matriz de confusión.

Otro paso importante fue ‘tunear’ los hiper-parámetros para optimizar cada uno de los modelos. A continuación se presentan los hiper-parámetros modificados en cada modelo y la razón de haberlos seleccionado y modificado.

Hiper-Parámetros de la Regresión Logística
La regresión logística al ser un método lineal necesita del hiper-parámetro regularized logistic regression 2, que viene por defecto, este método de regularización permite manejar las densidades y el esparcimiento de la entrada, cuando se probó la regularización L1 y L2 con una formulación doble para penalización de L2 se obtuvieron los mejores resultados que se profundizan posteriormente.
cabe recalcar que con este método se disminuyen los sobreajustes.
Para prevenir los sobreajustes se realizó una limpieza de la base de datos principal que implica un costo de trabajo que el algoritmo no tuvo que asumir, por lo cual su rendimiento fue superior y acertado.

Hiper-parámetros de la Máquina de Soporte Vectorial
La implementación de la librería scikit-learn se basa en libsvm. El tiempo de ajuste se escala al menos de forma cuadrática con el número de muestras y puede no ser práctico más allá de decenas de miles de muestras. El hiper-parámetro que se tocó fue:

- kernel: Se tomó el kernel lineal al ser un problema de clasificación en dos clases separables entre sí de forma lineal.

Hiper-parámetros de K Vecinos
En este método se fija primero el n_neighbors (número de puntos cercanos) que es 3, ya que si se pone un valor más grande, el algoritmo corre el riesgo de volverse cada vez más lento.
algorithm, que es quien controla el algoritmo que computa las distancias, para este caso se usará kd_tree, en el cual los datos almacenados en cada nodo son un punto K-dimensional en el espacio y weights, que ayuda a la ponderación de los votos de cada punto y, en este caso, se utilizó uniform, donde los votos de los puntos tienen igual peso.

Hiper-parámetros de Random Forest
En este método fijamos un número de árboles en 40 ya que el aumento de ellos no evidenciaba ningún aumento porcentual en la predicción del modelo así que lo dejamos en 40, también se establece en 1 el hiper-parámetro min_samples_leaf ya que gracias a esto obtenemos menos subdivisiones en las hojas y por ende evitar un ajuste excesivo en el modelo; obteniendo así alrededor de un 95% de precisión, lo cual es muy bueno.

En la predicción visualizamos un resultado similar a los métodos regresión logística y red neuronal, a continuación, una tabla comparativa de los resultados de estos métodos para visualizar la similitud de predicción entre los métodos:

Método	Precisión	Predicción
Regresión logística	0.8115330520393812	7
SVM	0.6863572433192686	2
KNN	0.8776371308016878	11

Random Forest	0.9521800281293952	7
Red neuronal	0.8424753867791842	6

Cómo podemos ver en la tabla anterior el método Random Forest obtuvo el mismo resultado predicción que el de regresión logística, pero difieren en cuanto a precisión, en cambio con el método KNN se acerca más en precisión que los demás, pero su resultado de predicción es muy diferente.

Hiper-parámetros de Descenso de Gradiente Estocástico

Se seleccionó el modelo SGD de la librería scikit-learn y se implementó; inicialmente no se modificaron los hiper-parámetros pero la exactitud del algoritmo no fue la mejor, es por esto que se decidió 'tunearlos' un poco. Así pues, los hiper-parámetros que se utilizaron fueron:

- loss: La función de pérdida que se utilizará. El valor predeterminado es "hinge", que proporciona una SVM lineal. Se seleccionó la función de pérdida por defecto puesto que al tener en esencia un problema lineal (Clasificación en dos clases), fue la más adecuada.
- shuffle: Esta variable determina si los datos de entrenamiento deben mezclarse o no después de cada época. Su valor es verdadero, por lo que los datos sí serán mezclados aleatoriamente después de cada época.
- random_state: Se utiliza para mezclar los datos cuando shuffle está establecida en True. Pasa un entero para una salida reproducible a través de múltiples llamadas a las funciones. Se seleccionó el valor de 101 después de distintas pruebas, en pocas palabras, se fue cambiando su valor hasta que el algoritmo fue optimizado.

Finalmente, se hicieron dos pruebas:

1. La primera se hizo con los modelos sin optimizar, es decir, sin modificar los hiper-parámetros.
2. Y la segunda, se realizó después de haber variado dichos hiper-parámetros.

Como se mencionó anteriormente, se muestran los resultados para la primera parte de la prueba:

Algoritmo	L_Reg	SVM	KNN	R_For	SGD
Tiempo (s)	0.0300	0.013	0.01021	0.1167	0.0062

Tabla 1. Comparativa de tiempos de ejecución entre algoritmos pre-optimización.

De igual manera, a continuación se muestra una representación más gráfica de los tiempos de ejecución.

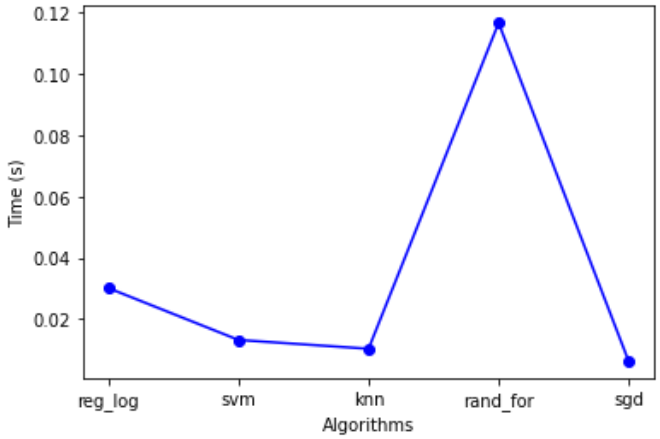

Figura 1. Gráfico comparativo tiempos de ejecución de los algoritmos pre-optimización.

Siendo SGD el algoritmo más rápido y Random Forest el que más tardó en completar su ejecución.

La matriz de confusión tiene cuatros salidas: Verdaderos Positivos, Falsos Positivos, Falsos Negativos y Verdaderos Negativos. Siendo, por ejemplo, los Verdaderos Positivos los elementos de la clase 1 que son clasificados como clase 1 en la predicción. De estos valores se desprenden los valores de accuracy, precision, sensitivity y specificity (Se calculan a partir de los primeros).

Para los modelos antes de su optimización se tiene que:

acc test	prec test	sens test	spec test	f1 score	mse	time	method
0.794007	0.783505	0.690909	0.866242	0.734300	0.205993	0.030045	reg_log
0.629213	0.617021	0.263636	0.885350	0.369427	0.370787	0.013098	svm
0.737828	0.681818	0.681818	0.777070	0.681818	0.262172	0.010212	knn
0.764045	0.790123	0.581818	0.891720	0.670157	0.235955	0.116731	rand_for
0.689139	0.728814	0.390909	0.898089	0.508876	0.310861	0.006156	sgd

Tabla 2. Resultados Pre-Optimización para cada algoritmo.

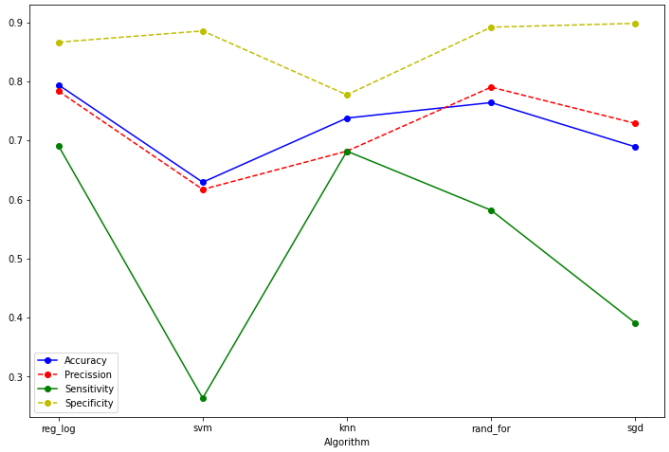

Figura 2. Comparación de valores calculados a partir de la matriz de confusión de los algoritmos pre-optimización.

Después de realizar el proceso de modificar los hiper-parámetros se tienen los siguientes resultados:

Algoritmo	L_Reg	SVM	KNN	R_For	SGD
Tiempo (s)	0.0517	0.777	0.01200	0.1336	0.0039

Tabla 3. Comparativa de tiempos de ejecución entre algoritmos post-optimización.

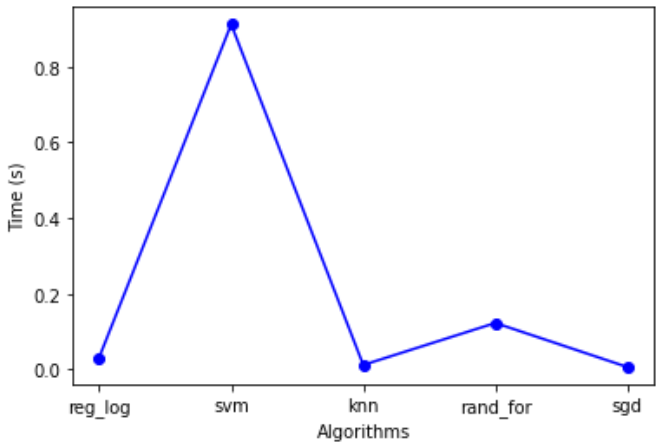

Figura 3. Gráfico comparativo tiempos de ejecución de los algoritmos post-optimización.

Se puede ver que ahora el algoritmo que más se demora es SVm, mientras que SGD se mantiene como el más rápido.

Y para los valores relevantes al problema de clasificación:

acc test	prec test	sens test	spec test	f1 score	mse	time	method
0.794007	0.783505	0.690909	0.866242	0.734299	0.205992	0.031649	reg_log
0.771536	0.747474	0.672727	0.840764	0.708134	0.228464	0.912225	svm
0.737828	0.681818	0.681818	0.777070	0.681818	0.262172	0.011999	knn
0.794007	0.761904	0.727272	0.840764	0.744186	0.205992	0.123022	rand_for
0.756554	0.706422	0.7	0.796178	0.703196	0.243445	0.006484	sgd

Tabla 4. Resultados Post-Optimización para cada algoritmo.

Aquí se evidencian mejoras para los valores de accuracy, precision, sensitivity y specificity para los diferentes algoritmos.

Así como su respectiva gráfica:

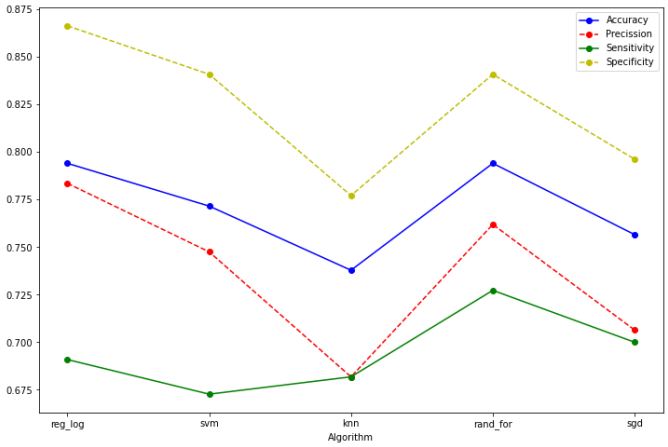

Figura 4. Comparación de valores calculados a partir de la matriz de confusión de los algoritmos post-optimización.

Se hace énfasis que en el Notebook del repositorio se pueden ver las gráficas de manera más clara.

IV. ENLACES RELEVANTES

Link del repositorio en Github: https://github.com/alejoso76/titanic-ml

V. CONCLUSIONES

- La regresión logística es un método eficaz para problemas donde la salida es binaria, da buenos tiempos de ejecución y de igual manera, sus resultados son acertados.
- Los árboles aleatorios nos brindan un excelente manejo en grandes cantidades de datos, teniendo un método efectivo para estimar datos faltantes y mantiene la precisión cuando falta una gran proporción de los datos.
- El algoritmo KNN es un método computacionalmente muy costoso por lo que se tienen que calcular muchas distancias, además de eso, es muy sensible al ruido, por lo cual se tiene que realizar un muy buen proceso de limpieza de los datos, en especial en este tipo de problemas donde la salida es binaria, ya que puede dar una mala predicción con facilidad.
- El proceso de 'tunear' los hiper-parámetros es clave para el éxito de un modelo de Machine Learning puesto que puede aumentar su eficacia así como reducir su tiempo de ejecución y su costo en memoria.
- El algoritmo SGD es particularmente útil para problemas con muchos datos, puesto que toma una muestra aleatoria, lo que reduce de forma considerable el costo en memoria y el tiempo de ejecución.
- SVM es un modelo ideal cuando no se conocen muy bien los datos del problema. Incluso puede trabajar con datos semi-estructurados o sin estructura.
- En este caso particular, elegir un kernel para SVM fue fácil; pero en general elegir una función de kernel que optimice el algoritmo no es fácil.
- Los algoritmos que mejor se desempeñaron para este problema de clasificación binaria fueron la Regresión Logística y Random Forest.
- Los algoritmos que peor se desempeñaron para este problema de clasificación binaria fueron KNN y SGD.

REFERENCIAS

[1] Bottou, Léon; Bousquet, Olivier (2012). "The Tradeoffs of Large Scale Learning". In Sra, Suvrit; Nowozin, Sebastian; Wright, Stephen J. (eds.). *Optimization for Machine Learning*. Cambridge: MIT Press. pp. 351–368. ISBN 978-0-262-01646-9.

[2] Orellana Alvear, J. O. A. (2018, 16 noviembre). Árboles de decisión y random forest. bookdown. https://bookdown.org/content/2031/ensambladores-random-forest-parte-i.html

[3] Decision Tree Hyperparameters: max_depth, min_samples_split, min_samples_leaf, max_features. (2019, 28 septiembre). [Vídeo]. YouTube.

[4] APRENDIZAJE SUPERVISADO: RANDOM FOREST CLASSIFICATION | #14 Curso de Introducción a Machine Learning. (2018, 8 abril). [Vídeo]. YouTube. https://www.youtube.com/watch?v=VH7eLWsLCks

[5] Heras, Máquinas de Vectores de Soporte (SVM). (2019). Recuperado en: https://www.iartificial.net/maquinas-de-vectores-de-soporte-svm/