Guía de Laboratorio – Aplicar métodos de minería de datos a bases de datos

Sistema de conocimiento:

- Métodos de Regresión
- Métodos de clasificación

Objetivos:

- Aplicar técnicas de minería de datos (regresión y clasificación) sobre conjuntos de datos ya preprocesados.

Bibliografía:

- Johnston, B., & Mathur, I. (2019). *Applied supervised learning with Python: Use scikit-learn to build predictive models from real-world datasets and prepare yourself for the future of machine learning*. Packt Publishing Ltd.

Introducción

En conferencias pasadas estudiamos varias técnicas de minería de datos para el descubrimiento de conocimiento en las bases de datos. Estas técnicas se corresponden con una de las etapas de KDD (Figura 1) que recibe como entrada el conjunto de datos ya preprocesado y obtiene como salida un modelo que puede ser usado para predecir la clase (clasificación) u obtener el valor calculado (regresión).

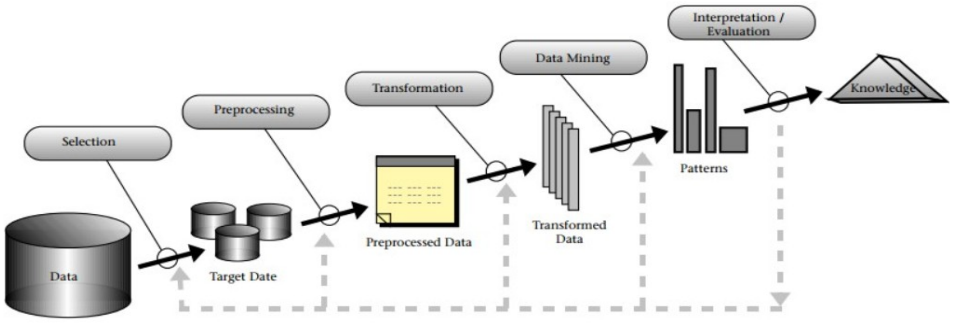


Figura 1: Fases de KDD.

Entre estas técnicas, vimos en clase las siguientes:

- KNN
- Árboles de decisión (Random Forest)
- Redes Neuronales de Simple Capa

Aunque existen muchas otras técnicas que pueden ser utilizadas con la biblioteca Scikit-learn. En el laboratorio de hoy exploraremos varios de estos métodos de minería de datos.

Los problemas de aprendizaje supervisado tienen como objetivo asignar la información de entrada a un valor de salida conocido o etiqueta, pero hay otras dos subcategorías a considerar. Tanto los problemas de aprendizaje supervisado como los no supervisado pueden dividirse en problemas de regresión o clasificación.

Los problemas de regresión pretenden *predecir* o *modelizar* valores continuos, por ejemplo, predecir la temperatura mañana en grados centígrados o determinar la localización de una cara dentro de una imagen. En cambio, los problemas de clasificación, en lugar de devolver un valor continuo, *predicen la pertenencia* a una de un número determinado de clases o categorías.

Datos, modelos, entrenamiento y evaluación

Antes de comenzar nuestra inmersión en los problemas de regresión y clasificación veremos primero las cuatro etapas principales que intervienen en la creación de cualquier modelo de aprendizaje automático. Estas etapas son las siguientes:

1. Preparación de los datos
2. Especificación de la arquitectura del modelo
3. El diseño y la ejecución del proceso de entrenamiento
4. Evaluación del modelo entrenado

Es aconsejable que se asegure de que entiende perfectamente este proceso ya que cada una de estas etapas es fundamental para lograr un rendimiento del sistema alto o incluso razonable.

Métodos de regresión

Comenzaremos nuestra investigación de los problemas de regresión con la selección de un modelo lineal. Los modelos lineales, además de ser una excelente primera opción debido a su naturaleza intuitiva, son también muy potentes en su poder predictivo, suponiendo que los conjuntos de datos contengan cierto grado de incertidumbre. También muy potentes en su poder predictivo, suponiendo que los conjuntos de datos contengan algún grado de relación lineal o polinómica entre las características y los valores de entrada.

La naturaleza intuitiva de los modelos lineales a menudo surge de la capacidad de ver los datos trazados en un gráfico y observar un patrón de tendencia en los datos con, por ejemplo, la salida (el

valor del eje y para los datos) con tendencia positiva o negativa con la entrada (valor del eje x). Aunque a menudo no presentados como tales, los componentes fundamentales de los modelos de regresión lineal también se suelen aprenderse durante las clases de matemáticas de la escuela secundaria. Quizá recuerde que la ecuación de una línea recta, o modelo lineal, se define de la siguiente manera:

$$y = mx + n$$

Ecuación 1: Ecuación de una línea recta.

Aquí, x es el valor de entrada e y es el valor de salida o predicho correspondiente. Los parámetros del modelo son el gradiente o pendiente de la línea (cambio en los valores y dividido por el cambio en x) definido por m, así como el valor de la intersección y b, que indica donde la línea cruza el eje y. Con un modelo de este tipo, podemos proporcionar valores para los parámetros m y b para construir un modelo lineal. Por ejemplo, y = 2x + 1, tiene una pendiente de 2, lo que indica que los cambios en los valores de y son el doble que los de x; la línea cruza el eje y en 1:

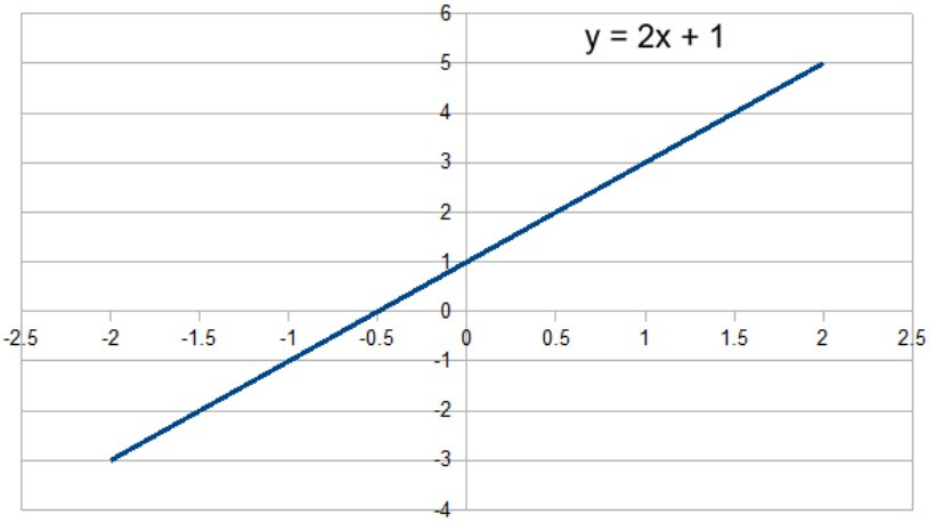


Figura 2: Parámetros de una línea recta.

Así, tenemos una comprensión de los parámetros que se requieren para definir una línea recta, pero esto realmente no está haciendo nada particularmente interesante. Acabamos de dictar los parámetros del modelo para construir una línea. Lo que queremos hacer es tomar un conjunto de datos y construir un modelo que describa lo mejor posible un conjunto de datos. Como se mencionó antes, este conjunto de datos necesita tener algo que se aproxime a una relación lineal entre las características de entrada y los valores de salida.

A continuación carguemos el conjunto de datos, graficamos, e interroguemos el dataset:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
```

	AverageTemperature	Year
0	12.980258	1841
1	13.689697	1842
2	12.485703	1843
3	14.202069	1844
4	12.831530	1845

```
df = pd.read_csv("./datasets/synth_temp.csv")
df.head()
```

Cómo solamente nos interesa los datos desde 1901 a 2010, vamos a eliminar las filas anteriores a 1901:

```
df = df.loc[df.Year > 1901]
df.head()
```

La salida del dataset ahora sería:

	AverageTemperature	Year
366	16.973653	1902
367	17.181773	1903
368	17.436933	1904
369	17.688948	1905
370	17.811166	1906

El conjunto de datos original contiene múltiples mediciones de temperatura por año, con más mediciones para los últimos años (12 para 2010) y menos para los primeros años (6 para 1841); sin

embargo, estamos interesados en una lista de temperaturas medias anuales. Agrupe los datos por año y utilice el método agg del DataFrame para crear las medias anuales:

```
df_group_year = df.groupby('Year').agg(np.mean)
df_group_year.head()
```

	AverageTemperature
Year	
1902	17.438122
1903	17.375456
1904	17.558674
1905	17.740646
1906	17.501770

Dado que los datos son bastante ruidosos, un filtro de media móvil proporcionaría un indicador útil de la tendencia general. Un filtro de media móvil calcula simplemente la media de los últimos N valores y asigna esta media a la (N+1)-esima muestra. Calcular los valores de una señal de media móvil para las mediciones de temperatura utilizando una ventana de 10 años:

```
window = 10
rolling = df_group_year.AverageTemperature.rolling(window).mean()
rolling.head(n=20)
```

Se obtendrá la siguiente salida:

```
Year
1902          NaN
1903          NaN
1904          NaN
1905          NaN
1906          NaN
1907          NaN
1908          NaN
1909          NaN
1910          NaN
1911    17.501145
1912    17.502700
1913    17.500737
1914    17.487112
1915    17.466333
1916    17.460069
1917    17.475434
1918    17.463959
1919    17.472423
1920    17.474037
1921    17.480317
Name: AverageTemperature, dtype: float64
```

Observe que las 9 primeras muestras son NaN, lo que se debe al tamaño de la ventana del filtro de media móvil. El tamaño de la ventana es 10, por lo que se necesitan 9 (10-1) muestras para generar la primera media, por lo que las 9 primeras muestras son NaN.

Finalmente, grafiquemos las medidas por año junto con la señal de ventana móvil:

```
fig = plt.figure(figsize=(10, 7))
ax = fig.add_axes([1, 1, 1, 1]);
# Temp measurements
ax.scatter(df_group_year.index, df_group_year.AverageTemperature,
            label='Raw Data', c='k');
ax.plot(df_group_year.index, rolling, c='k', linestyle='--',
label=f'{window} year moving average');
ax.set_title('Mean Air Temperature Measurements')
ax.set_xlabel('Year')
ax.set_ylabel('Temperature (degC)')
ax.set_xticks(
            range(df_group_year.index.min(), df_group_year.index.max(),10))
ax.legend();
```

La salida sería la siguiente:

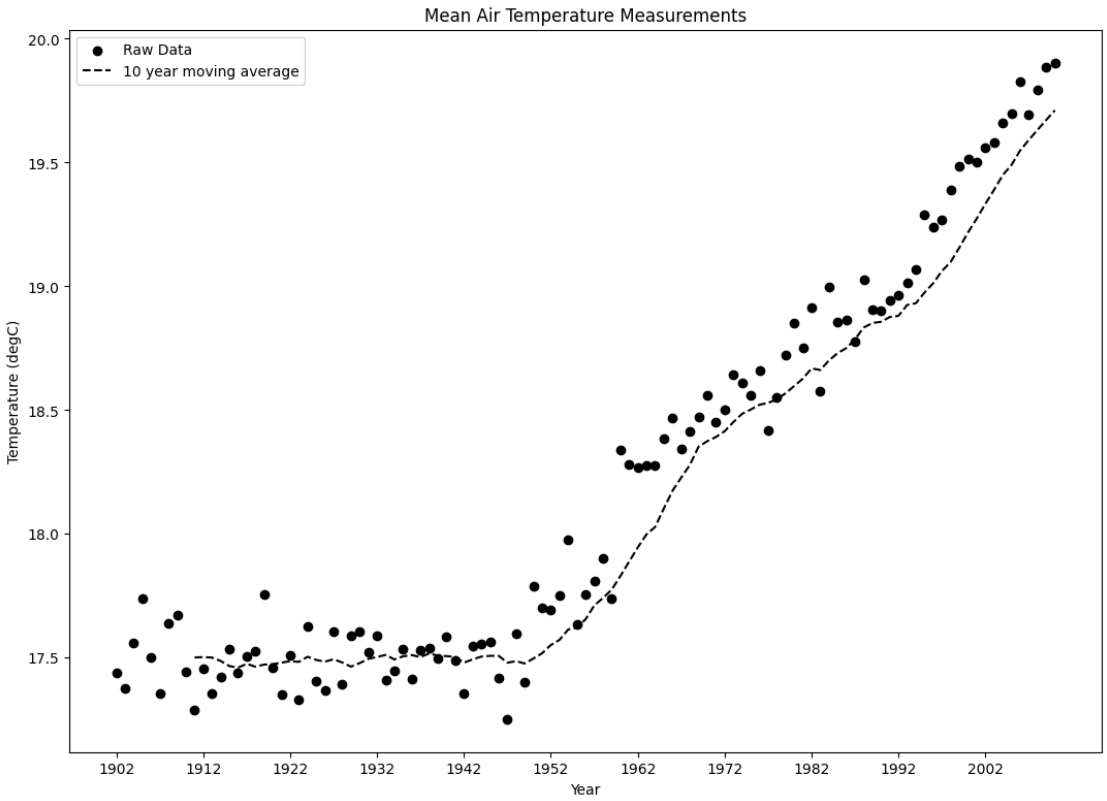


Figura 3: Temperatura media anual del aire.

La API de scikit-learn utiliza un patrón de código razonablemente sencillo, independientemente del tipo de modelo que se esté construyendo. En pocas palabras, primero se debe definir el modelo con todos los hiperparámetros apropiados que sean relevantes para el proceso de entrenamiento o ajuste. Al definir el modelo, se devuelve un objeto modelo, que se emplea durante la segunda etapa de la construcción del modelo, que es el entrenamiento o ajuste. Al llamar al método de ajuste en el objeto modelo con los datos de entrenamiento apropiados, se entrenará el modelo con los hiperparámetros definidos. Ahora usaremos este patrón para construir nuestro primer modelo de regresión lineal.

Utilizaremos el modelo LinearRegression de scikit-learn para este ejercicio, por lo que importaremos la clase del módulo linear_regression de scikit-learn. Luego se construye un modelo de regresión lineal utilizando los valores por defecto; es decir, calculando un valor para él intercepto y no normalizando los datos de entrada. Ahora estamos listos para ajustar o entrenar el modelo a los datos. Proporcionaremos los valores como entrada y la temperatura media anual como salida. Tenga en cuenta que el método de ajuste de modelos scikit-learn espera matrices 2D que se proporcionará como la X e Y. Como tal, los valores del año o del índice necesitan ser reformados para adaptarse al

método. Obtenga los valores del índice como una matriz NumPy usando el método *.values* y reforme los valores a ((-1, 1)) que es un array N x 1. El valor -1 en una definición de forma NumPy representa que su valor se infiere de la forma actual del array y la forma objetiva.

```
from sklearn.linear_model import LinearRegression

model = LinearRegression()
model.fit(df_group_year.index.values.reshape((-1, 1)), df_group_year.AverageTemperature)
```

Una vez entrenado el modelo podemos obtener los valores de los parámetros del modelo imprimiendo el valor ***model.coef***_ (que es el valor de m) y ***model.intercept***_ (que es el valor para la intercepción de y).

```
print(f'm = {model.coef_[0]}')
print(f'c = {model.intercept_}')
print('\nModel Definition')
print(f'y = {model.coef_[0]:0.4}x + {model.intercept_:0.4f}')
```

En este ejemplo la salida sería:

```
m = 0.023146460838006862
c = -27.080386660799967

Model Definition
y = 0.02315x + -27.0804
```

Ahora que tenemos nuestro modelo generado, podemos predecir algunos valores para construir nuestra línea de tendencia. Utilicemos el primer valor, el último y el valor medio del año como entrada para predecir la temperatura local. Construimos una matriz NumPy con estos valores y la llamamos la matriz trend_x. Una vez que hayas terminado, pasa los valores de trend_x al método predict del modelo para obtener los valores predichos:

```
trend_x = np.array([
df_group_year.index.values.min(),
df_group_year.index.values.mean(),
df_group_year.index.values.max()
])
trend_y = model.predict(trend_x.reshape((-1, 1)))
trend_y
```

Trace ahora la línea de tendencia producida por el modelo, con los parámetros del modelo sobre el gráfico anterior con los datos brutos:

```
fig = plt.figure(figsize=(10, 7))
ax = fig.add_axes([1, 1, 1, 1]);
# Temp measurements
ax.scatter(df_group_year.index, df_group_year.AverageTemperature,
label='Raw Data', c='k');
ax.plot(df_group_year.index, rolling, c='k', linestyle='--',
label=f'{window} year moving average');
ax.plot(trend_x, trend_y, c='k', label='Model: Predicted trendline')
ax.set_title('Mean Air Temperature Measurements')
ax.set_xlabel('Year')
ax.set_ylabel('Temperature (degC)')
ax.set_xticks(range(df_group_year.index.min(), df_group_year.index.max(),
10))
ax.legend();
```

La salida es la siguiente gráfica:

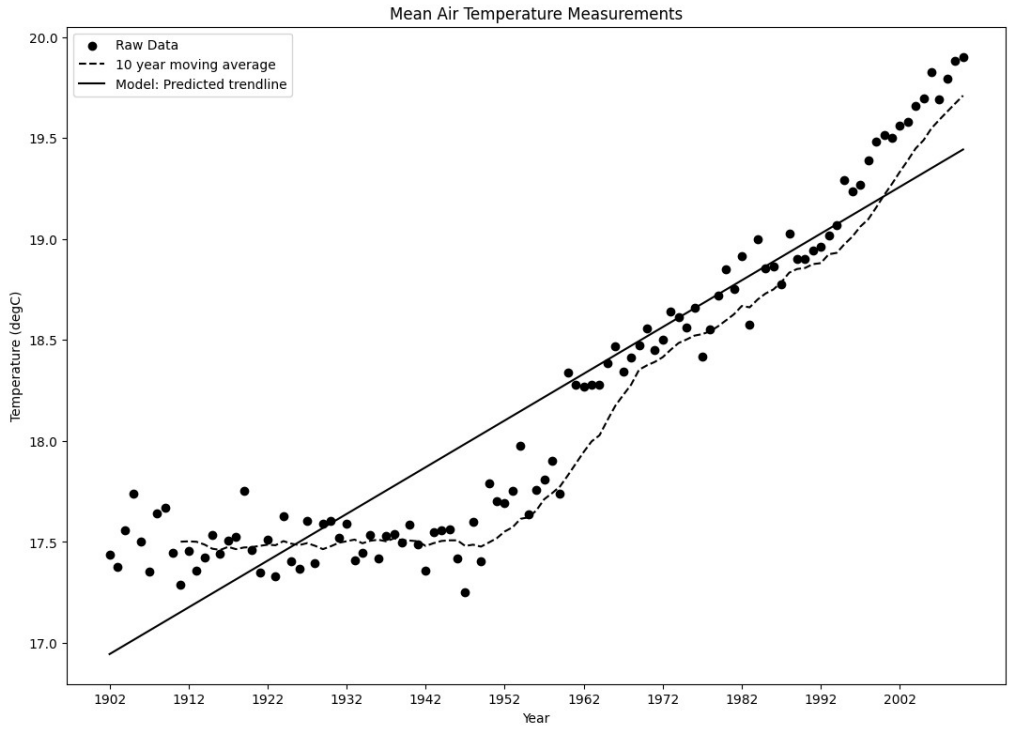


Figura 4: Regresión lineal: un primer modelo lineal simple.

Para profundizar en el estudio de otros métodos de regresión puede leer el capítulo 3 Regression Analysis del libro (Johnston & Mathur, 2019).

Métodos de clasificación

Recordemos que las tareas de clasificación tienen como objetivo predecir, dado un conjunto de datos de entrada, a cuál de un número determinado de grupos de clases pertenecen los datos.

Clasificación usando K-Nearest Neighbors

Inicialmente mostraremos como entrenar un modelo usando el algoritmo KNN (Figura 5). Se trata de un método práctico, ya que puede utilizarse tanto en problemas de clasificación supervisada como en problemas de clasificación no supervisada.

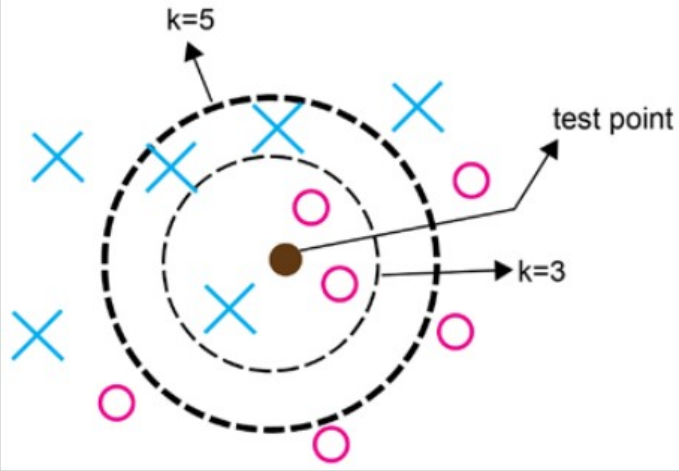


Figura 5: Representación visual del método KNN. Fuente: (Johnston & Mathur, 2019)

Para permitir la visualización del proceso K-NN, en este ejercicio nos centraremos en el conjunto de datos Iris.

Para este caso de estudio necesitaremos las bibliotecas pandas, matplotlib, y la clase KneighborsClassifier de scikit-learn.

```
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.neighbors import KNeighborsClassifier as KNN
```

Por tanto, la primera tarea que debemos realizar es cargar el conjunto de datos:

```
df = pd.read_csv("./datasets/iris-data.csv")
df
```

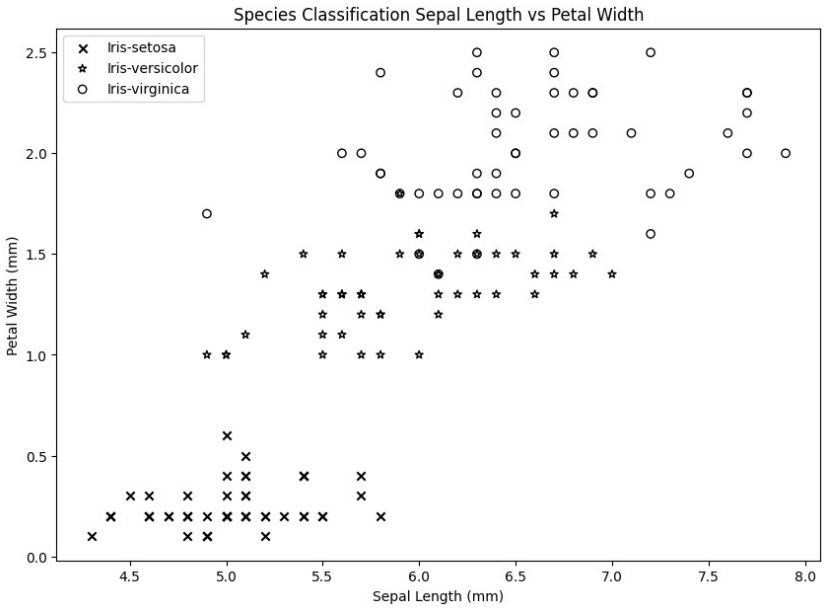
En esta fase, tenemos que elegir las características más apropiadas del conjunto de datos para el clasificador. Podríamos simplemente seleccionar las cuatro (sépalo y pétalo, longitud y anchura), sin embargo, como este caso de estudio está diseñado para permitir la visualización de la K-NN, sólo

seleccionaremos la longitud del sépalo y la anchura del pétalo. Construiremos un gráfico de dispersión para la longitud del sépalo frente a la anchura del pétalo para cada una de las clases del conjunto de datos con la especie correspondiente:

```
markers = {
    'Iris-setosa': {'marker': 'x', 'facecolor': 'k', 'edgecolor': 'k'},
    'Iris-versicolor': {'marker': '*', 'facecolor': 'none', 'edgecolor':'k'},
    'Iris-virginica': {'marker': 'o', 'facecolor': 'none', 'edgecolor':'k'},
}
plt.figure(figsize=(10, 7))
for name, group in df.groupby('Species'):
    plt.scatter(group['Sepal Length'],
                group['Petal Width'],
                label=name,
                marker=markers[name]['marker'],
                facecolors=markers[name]['facecolor'],
                edgecolor=markers[name]['edgecolor'])

plt.title('Species Classification Sepal Length vs Petal Width');
plt.xlabel('Sepal Length (mm)');
plt.ylabel('Petal Width (mm)');
plt.legend();
```

La salida es la siguiente:



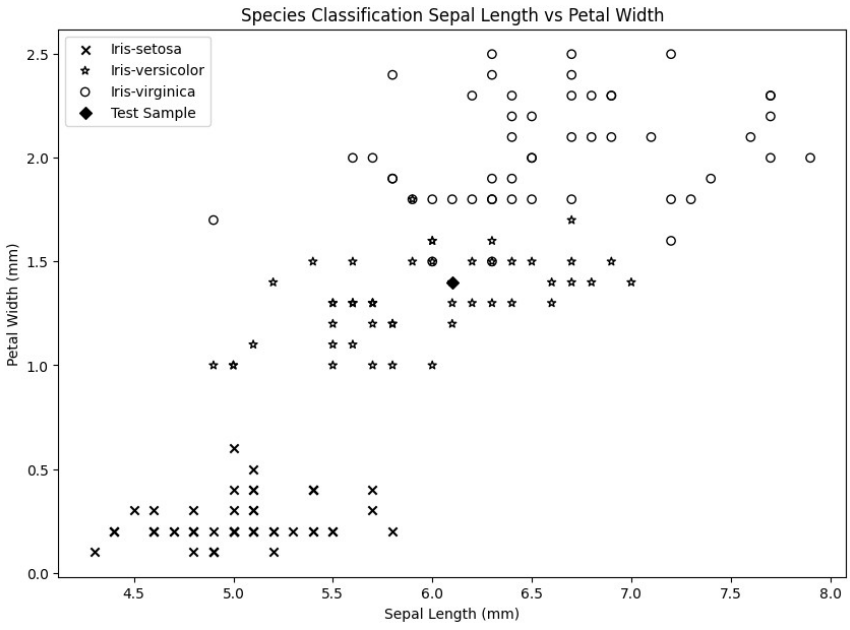
Observando este gráfico, podemos ver que las especies están razonablemente bien separadas por la anchura de los pétalos, con la mayor similitud entre las especies Iris versicolor e Iris virginica. Hay un

par de puntos de especies de Iris virginica que se encuentran dentro de Iris versicolor. Como punto de prueba para su uso posterior, seleccione uno de estos puntos en el límite-muestra 134:

```
df_test = df.iloc[134]
df = df.drop([134]) # Remove the sample
df_test
```

Ahora mostremos nuevamente el gráfico anterior resaltando este punto de prueba.

```
plt.figure(figsize=(10, 7))
for name, group in df.groupby('Species'):
    plt.scatter(group['Sepal Length'], group['Petal Width'],
    label=name,
    marker=markers[name]['marker'],
    facecolors=markers[name]['facecolor'],
    edgecolor=markers[name]['edgecolor'])
plt.scatter(df_test['Sepal Length'], df_test['Petal Width'], label='Test Sample', c='k',
marker='D')
plt.title('Species Classification Sepal Length vs Petal Width');
plt.xlabel('Sepal Length (mm)');
plt.ylabel('Petal Width (mm)');
plt.legend();
```



Construyamos un modelo de clasificación KNN con K=3 y entrenémoslo con los datos de entrenamiento:

```
model = KNN(n_neighbors=3)
model.fit(X=df[['Petal Width', 'Sepal Length']], y=df.Species)
model.get_params()
```

La salida es la siguiente:

{'algorithm': 'auto', 'leaf_size': 30, 'metric': 'minkowski', 'metric_params': None, 'n_jobs': None, 'n_neighbors': 3, 'p': 2, 'weights': 'uniform'}

Para comprobar el rendimiento del modelo contra el conjunto de entrenamiento podemos hacerlo de la siguiente manera:

```
model.score(X=df[['Petal Width', 'Sepal Length']], y=df.Species)
```

De la salida de esta línea podemos comprobar que el modelo está por encima del 97%. El siguiente paso es comprobar el caso de prueba que guardamos anteriormente:

```
model.predict(df_test[['Petal Width', 'Sepal Length']].values.reshape((-1,2)))[0]
```

Para comprobar el valor real de ese punto podemos comprobarlo de la siguiente manera:

```
df.iloc[134].Species
```

Esta predicción es claramente incorrecta, pero no es sorprendente dada la ubicación del punto de prueba en la frontera.

Clasificación usando Árboles de decisión

Existen varios algoritmos de aprendizaje automático que se engloban dentro de la árboles de decisión, como ID3, CART y los potentes clasificadores de bosque aleatorios. Utilizaremos la implementación CART de scikit-learn CART para clasificar el conjunto de datos Iris.

Lo primero es cargar las bibliotecas que emplearemos y cargamos el conjunto de datos.

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.tree import DecisionTreeClassifier

df = pd.read_csv("./datasets/iris-data.csv")
```

Tome una muestra aleatoria de 10 filas para utilizarla en las pruebas. Los árboles de decisión pueden sobreajustar los datos de entrenamiento, por lo que esto proporcionará una medida independiente de la precisión del árbol:

```
np.random.seed(10)
samples = np.random.randint(0, len(df), 10)
df_test = df.iloc[samples]
```

```
df = df.drop(samples)
```

Ajuste el modelo a los datos de entrenamiento y compruebe la precisión correspondiente:

```
model = DecisionTreeClassifier()
model = model.fit(df[['Sepal Length', 'Sepal Width', 'Petal Length', 'Petal Width']],
df.Species)
model.score(df[['Sepal Length', 'Sepal Width', 'Petal Length', 'Petal Width']], df.Species)
```

Al ejecutar este código se puede observar que se alcanza el 100% de eficiencia sobre los datos de entrenamiento. Ahora comprobemos el rendimiento sobre los datos de prueba:

```
model.score(df_test[['Sepal Length', 'Sepal Width', 'Petal Length', 'Petal Width']],
df_test.Species)
```

De nuevo se obtiene una eficiencia del 100%.

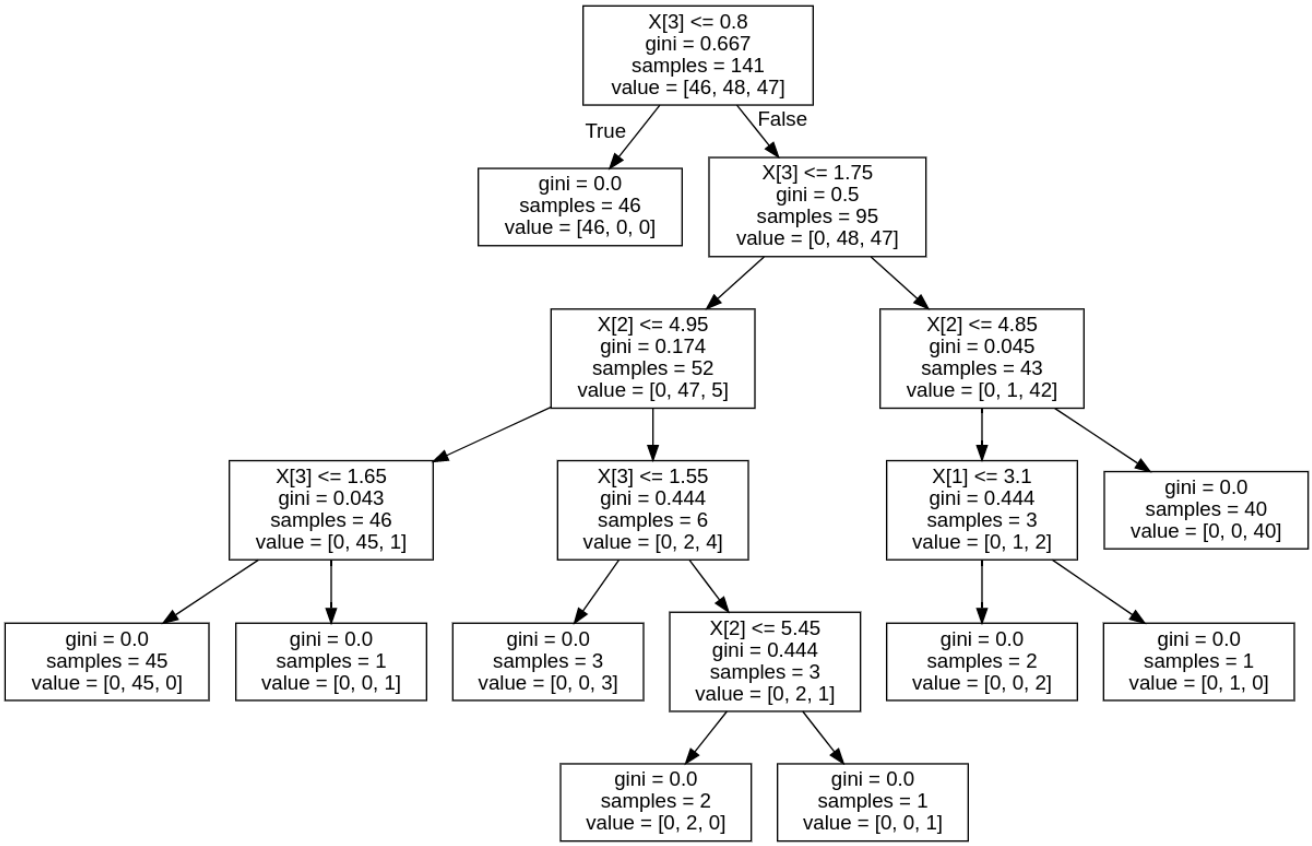
Una de las cosas grandiosas de los árboles de decisión es que podemos visualizar el modelo y ver exactamente que sucede. Instalemos la siguiente paquete:

```
pip install graphviz
```

Una vez instalado podemos dibujar el árbol de decisión construido de la siguiente manera:

```
import graphviz
from sklearn.tree import export_graphviz

dot_data = export_graphviz(model, out_file=None)
graph = graphviz.Source(dot_data)
graphs
```



Ejercicios

Cree un Jupyter Notebook y en el mismo resuelva los siguientes ejercicios.

1. Use el conjunto de datos ‘wine-white.csv’, realice el preprocesamiento que pueda requerir este y entrene un modelo de regresión lineal para determinar la calidad del vino.
2. Se desea entrenar un modelo de clasificación utilizando el árboles de decisión en el conjunto de datos ‘tic-tac-toe.csv’, para decidir si el resultado puede ser positivo o negativo.
   a) Grafique el árbol generado.
3. Carge el modelo de datos ‘breast-cancer.csv’ y entrene un modelo de clasificación usando KNN, para obtener un diagnostico.

Elaborado por: M.Sc. Angel Alberto Vazquez Sánchez
Profesor Auxiliar del Departamento de Inteligencia Computacional.