**Introducción a Machine Learning II**

**Modelos de Clasificación en Algoritmos de Aprendizaje Supervisado:** Los métodos de clasificación pueden usarse en aplicaciones para predecir bajas de clientes a un servicio (bajas de *churn*); distinción entre comentarios positivos y negativos en las redes sociales; filtrar *spam* en servicios de e-mail; diagnosticar enfermedades; detectar fraudes de tarjetas de crédito.

**Modelo de Clasificación Naive Bayes:** Es un algoritmo simple de clasificación que asume que todas las *features* son independientes entre sí; se basa en probabilidades y es un modelo que se usa de base. Su performance es un piso para el resto de los modelos.

Ejemplo: Tenemos un Dataset de semillas de trigo con algunas de sus características. Nuestra variable target es el tipo de variedad (1 – Kama; 2 – Rosa; 3 - Canadian).

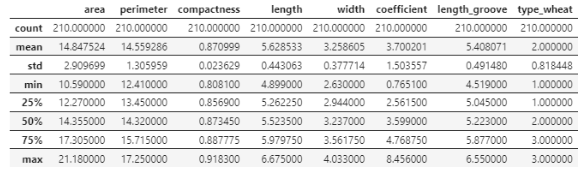
**En Python:**

df\_trigo = pd.read\_csv(‘../Data/seeds\_dataset.csv’)

df\_trigo.type\_wheat.unique()



df\_trigo.describe()



Vamos a aplicar los 7 pasos.

Paso 1: Selección del modelo

**from** sklearn.naive\_bayes **import** GaussianNB

Paso 2: Elegir los hiperparámetros

En este caso decidimos dejar los hiperparámetros por Default:

model = GaussianNB()

Paso 3: Preparar la Matriz Features y el Vector Target

X = df\_trigo.drop([‘type\_wheat’], axis = 1)

y = df\_trigo[‘type\_wheat’]

Paso 4: Separar los sets de entrenamiento y testing:

**from** sklearn.model\_selection **import** train\_test\_split

Xtrain, Xtest, ytrain, ytest = train\_test\_split(X, y, random\_state = 2)

Paso 5: Ajustar el Modelo a los Datos de Entrenamiento

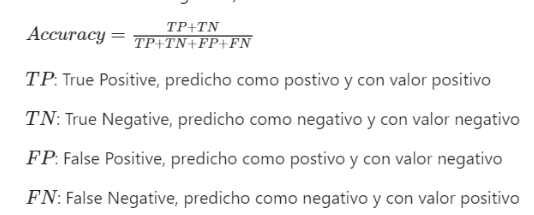
model.fit(Xtrain, ytrain)

Paso 6: Predecir Etiquetas para Datos Desconocidos

ypred = model.predict(Xtest)

Paso 7: Evaluar la Performance del modelo

Se calcula el accuracy, que es la proporción de predicciones correctas (tanto verdaderas positivas como verdaderas negativas) entre el número total de datos examinados.



**from** sklearn.metrics **import** accuracy\_score

round(accuracy\_score(ytest, ypred), 2)



En el ejemplo, da 87%. 87 de cada 100 casos fueron correctamente predichos.

**Aprendizaje No Supervisado:**

El objetivo básico de los **algoritmos de Clustering** es **descubrir grupos o patrones en los datos**. Esto lo logra agrupando observaciones similares y generando grupos distintos entre sí; y en general disjuntos. Puede usarse para segmentación de productos; Procesamiento de imágenes; Astronomía; Búsqueda de Outliers.

**K-means**: Una técnica clásica de Clustering es **k-means**.

Se trata de un algoritmo que genera **grupos de observaciones similares** entre sí ***sin tener*** *una* ***variable target***.

Debemos especificar de antemano la cantidad de Clusters a obtener (K). Se divide el dataset en K grupos de observaciones **disjuntos**.

Se usa **la distancia** para **determinar similaridad** entre las observaciones.

Cada **grupo** se define por su **mean**, representado por el **centroide**, que es un punto que no necesariamente es una observación. Las observaciones pertenecen al grupo que tenga el centroide más cercano a ellas.

Se realizan varias iteraciones hasta converger al modelo entrenado.

**En Python:**

Ejemplo: Vamos a usar un dataset que indica el salario medio según los años de experiencia.

df\_salary = pd.read\_csv(‘../Data/Salary.csv’)

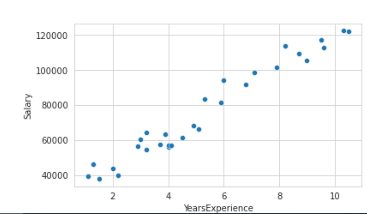
# Lo graficamos en un scatterplot:

sns.set\_style(‘whitegrid’)

fig = plt.figure(figsize=(6,3.5))

sns.scatterplot(data=df\_salary, x=’YearsExperience’, y=’Salary’)

plt.show()

**

Se aplican los pasos típicos del workflow de scikit-learn para generar el modelo:

Paso 1: Seleccionamos una clase de Modelo

**from** sklearn.cluster **import** KMeans

Paso 2: Elegimos los Hiperparámetros

Kmeans = KMeans(n\_clusters = 4)

Necesitamos estandarizar los valores para que todas las features queden en la misma escala, para que el modelo pueda determinar la distancia entre las observaciones calculando distancias.

Con la clase **StandardScaler()** pueden transformarse los datos de forma tal que las variables tengan media 0 y desvío estándar 1.

**En Python: from** sklearn.preprocessing **import** StandardScaler

scaler = StandardScaler()

X = df\_salary

X\_scaled = scaler.fit\_transform(X)

# De esta forma ajustamos el modelo a los datos estandarizados; en este caso (por el modelo con el que estamos trabajando) ya no necesitamos de un dataset de train y test.

Paso 5: Ajustar el Modelo a los Datos

kmeans.fit(X\_scaled)



Obtenemos los **parámetros del Modelo**:

* labels\_: Vector con la asignación del cluster a cada observación.
* centers\_: Coordenadas de los centroides.

labels = kmeans.labels\_

centroids = kmeans.cluster\_centers\_

# Ahora podemos graficar los Clusters y sus centroides. La escala de variables está estandarizada.

plt.figure(figsize=(8,5))

sns.scatterplot(x=X\_scaled[:,0], y=X\_scaled[:,1], hue=labels, legend=’full’)

plt.xlabel(‘Años de experiencia’, fontsize=15)

plt.ylabel(‘Salario’, fontsize=15)

plt.scatter(centroids[:,0], centroids[:,1], marker = ‘x’, s=50, color=’r’)

