Universidad Nacional Autónoma de México
Facultad de Estudios Superiores Acatlán
Licenciatura: Ciencia de Datos
Materia: Aprendizaje de Maquina y Minieria de Datos Avanzados
Profesor: Dr. Eduardo Eloy Loza Pacheco
Parcial 1

Fecha:13/04/2023	Nombre: Gustavo Adolfo Alvarez Hernández	No Cuenta: 316017972

Preprocesamiento de los datos

Primero mostraremos la obtención de los datos y las transformaciones que se realizaron para implementar los algoritmos.

Se cargan los conjuntos de datos proporcionados y se separa en variables explicativas y la variable objetivo.

Cargar datos

```
data1= pd.read_csv("ConjuntoDeDatos1.csv")
data2= pd.read_csv("ConjuntoDeDatos2.csv")
```

```
X= data1.iloc[:,0:13].values
y= data1.iloc[:,13].values
X2= data2.iloc[:,1:3].values
y2= data2.iloc[:,4].values
```

Al tener algunos algoritmos que funcionan con datos escalados, se procede a separar en train y test los datos y escalarlos posteriormente con un StandarScaler y sería el preprocesamiento realizado.

Dividir datos

```
x_train,x_test,y_train,y_test= train_test_split(X,y, test_size=.25, random_state=0)
x2_train,x2_test,y2_train,y2_test= train_test_split(X2,y2, test_size=.25, random_state=0)
```

Escalamiento de los datos

```
sx= StandardScaler()
x_train= sx.fit_transform(x_train)
x_test= sx.fit_transform(x_test)
sx2= StandardScaler()
x2_train= sx2.fit_transform(x2_train)
x2_test= sx2.fit_transform(x2_test)
```

Conjunto de datos 1

Todo lo realizado se usan los datos de calidad del vino dados por el conjunto de datos 1.

Alcohol	Malic_Acid	Ash	Ash_Alcanity	Magnesium	Total_Phenols	Flavanoids	Nonflavanoid_Phenols	Proanthocyanins	Color_Intensity	Hue	OD280	Proline	Customer_Segment
14.23	1.71	2.43	15.6	127	2.80	3.06	0.28	2.29	5.64	1.04	3.92	1065	1
13.20	1.78	2.14	11.2	100	2.65	2.76	0.26	1.28	4.38	1.05	3.40	1050	1
13.16	2.36	2.67	18.6	101	2.80	3.24	0.30	2.81	5.68	1.03	3.17	1185	1
14.37	1.95	2.50	16.8	113	3.85	3.49	0.24	2.18	7.80	0.86	3.45	1480	1
13.24	2.59	2.87	21.0	118	2.80	2.69	0.39	1.82	4.32	1.04	2.93	735	1

1.PCA

Realizamos PCA sobre el train para reducirlo a 2 dimensiones y poder graficar los datos.

PCA

```
pca= PCA(n_components=2)
x_train= pca.fit_transform(x_train)
x_test= pca.transform(x_test)
```

```
pca.explained_variance_ratio_
```

```
array([0.37281068, 0.18739996])
```

Explicando el 56% de la varianza total de los datos totales.

A) Árbol de decisión

Se entrena un árbol de decisión con al menos 3 datos en cada hoja, se entrena con los datos de train.

```
tree = DecisionTreeClassifier(random_state=0, min_samples_leaf=3)
tree.fit(x_train,y_train)
```

Se procede a calcular la matriz de confusión sobre los datos de test que no ha visto el algoritmo.

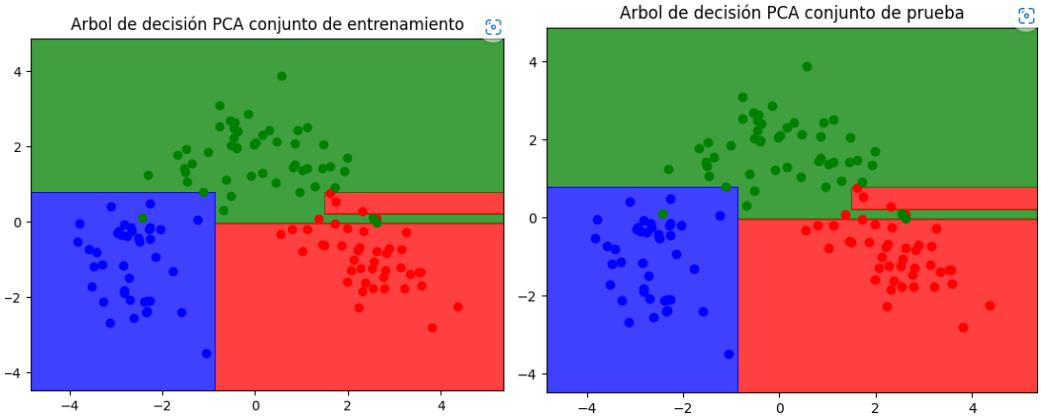
```
y_tree=tree.predict(x_test)
cm= confusion_matrix(y_test,y_tree)
```

```
dc_acurracy["arbol PCA"]= accuracy_score(y_test, y_tree)
```

```
cm
```

```
array([[16,  0,  0],
       [ 1, 18,  2],
       [ 0,  0,  8]])
```

Por último, mostramos como clasifica el árbol en los dos conjuntos de datos



2. LDA

Se realiza el análisis de discriminante sobre los mismos datos y se utiliza para entrenar dos nuevos modelos.

```
lda = LDA(n_components=2)
x_train=lda.fit_transform(x_train, y_train)
x_test = lda.transform(x_test)
```

A) Árbol de regresión

Se entrena un árbol de regresión con los datos de entrenamiento.

```
ldaTree = DecisionTreeRegressor()
ldaTree.fit(x_train,y_train)
```

Se genera la matriz de confusión

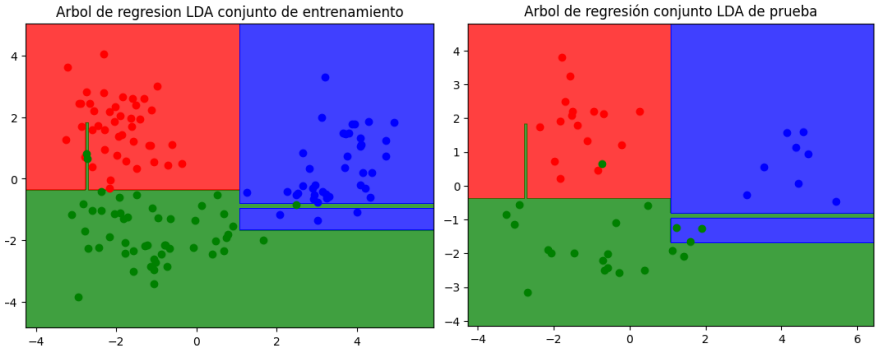
```
y_ldaTree = ldaTree.predict(x_test)
cm = confusion_matrix(y_test,y_ldaTree)
```

```
dc_acurracy["arbol LDA"]= accuracy_score(y_test, y_ldaTree)
```

```
cm
```

```
array([[16,  0,  0],
       [ 1, 17,  3],
       [ 0,  0,  8]])
```

Por último, se muestra las gráficas del árbol sobre train y test



B) KNN

Se entrena el algoritmo de KNN con k=5

```
knn = KNeighborsClassifier(n_neighbors=5, metric='minkowski',p=2)
knn.fit(x_train,y_train)
```

Se genera la matriz de confusión con los datos de prueba

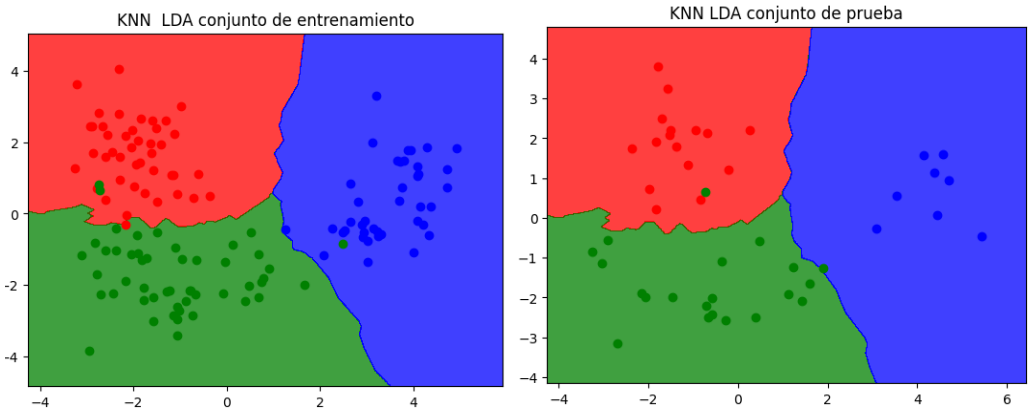
```
y_knn=knn.predict(x_test)
cm= confusion_matrix(y_test,y_knn)
```

```
dc_acurracy["knn LDA"]= accuracy_score(y_test, y_knn)
```

```
cm
```

```
array([[16,  0,  0],
       [ 1, 19,  1],
       [ 0,  0,  8]])
```

Y se genera la gráfica en cada conjunto de datos.



3. Elección del mejor modelo

Como podemos observar tanto las transformaciones como el algoritmo realizan cambios significativos en la forma en que se clasifica, en general en ambos árboles se ve un overfitting por lo cual no se generaliza de una buena forma.

Por su parte KNN lo hace de una mejor manera, aunque si su desempeño es lento, también se analiza el acurracy de los modelos en la siguiente imagen:

```
]: dc_acurracy
```

```
]: {'arbol PCA': 0.9333333333333333,
    'arbol LDA': 0.9111111111111111,
    'knn LDA': 0.9555555555555556}
```

Entonces al ser KNN el que tiene mejor acurracy y generaliza de mejor manera, esta será la elección apropiada para el modelo.

2. Conjunto de datos 2

Este conjunto de datos contiene datos sobre anuncios recibidos por usuarios y si compro o no el usuario.

	User ID	Gender	Age	EstimatedSalary	Purchased
0	15624510	1	19.0	19000.0	0
1	15810944	1	35.0	20000.0	0
2	15668575	0	26.0	43000.0	0
3	15603246	0	27.0	57000.0	0
4	15804002	1	19.0	76000.0	0

Kernel PCA

Se utiliza kernel PCA con kernel lineal para reducir nuestros datos en dos dimensiones.

```
transformer = KernelPCA(n_components=2, kernel='linear')
x2_train= transformer.fit_transform(x2_train)
x2_test= transformer.transform(x2_test)
```

A) Regresión logística

Se utiliza una regresión logística como primer algoritmo para clasificar si compro o no.

```
regLog= LogisticRegression(random_state=0)
regLog.fit(x2_train, y2_train)
```

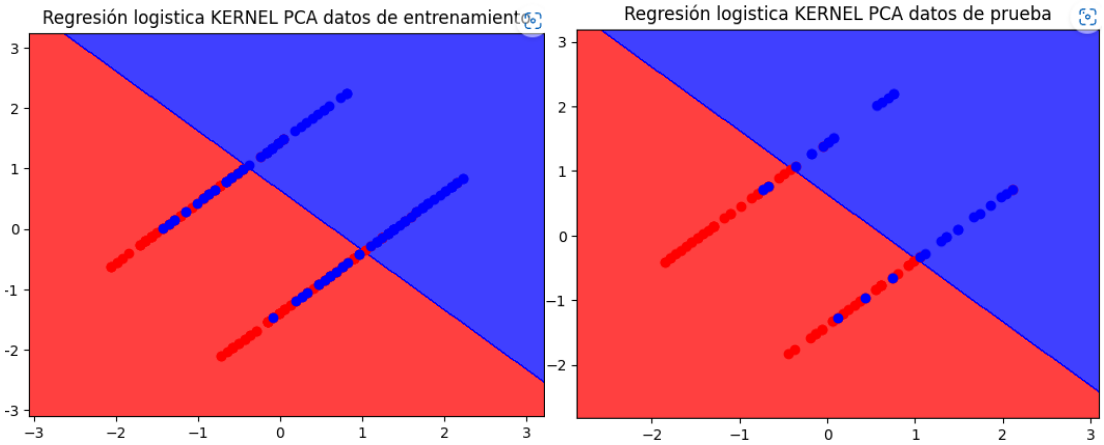
Se genera la matriz de confusión

```
y_log= regLog.predict(x2_test)
cm=confusion_matrix(y2_test,y_log)
```

```
cm
```

```
array([[63,  5],
       [ 6, 26]])
```

Y se visualiza la clasificación en ambos conjuntos de datos



B) Árbol de clasificación

Se genera otro árbol de clasificación, pero ahora con los datos de anuncios modificados por el Kernel PCA

```
tree2 = DecisionTreeClassifier(random_state=0, min_samples_leaf=3)
tree2.fit(x2_train,y2_train)
```

Obtenemos su matriz de confusión sobre los datos de prueba.

```
y_tree2= tree2.predict(x2_test)
cm=confusion_matrix(y2_test,y_tree2)
```

```
cm
```

```
array([[61,  7],
       [ 7, 25]])
```

Y visualizamos el comportamiento en cada conjunto.