Basile Álvarez Andrés José

No. Cuenta: 316617187

Email: andresbasile123@gmail.com

Fecha: 2/12/21

Inteligencia Artificial

Grupo III

Semestre 2022-1

Reporte Práctica 15: Clasificación (SVM)

Objetivo: Clasificar registros clínicos de tumores malignos y benignos de cáncer de mama a partir de imágenes digitalizadas

Fuente de datos: Estudios clínicos a partir de imágenes digitalizadas de pacientes con cáncer de mama de Wisconsin (WDBC, Wisconsin Diagnostic Breast Cancer), donde:

- ID number: Identifica al paciente (valor discreto).
- Diagnosis: Diagnóstico (M=maligno, B=benigno).
- Radius: Media de las distancias del centro y puntos del perímetro.
- Texture: Desvaición estándar de la escala de grises.
- Perimeter: Valor del perímetro del cáncer de mama.
- Area: Valor del área del cáncer de mama.
- Smoothness: Variación de la longitud del radio.
- Compactness: Perímetro ^2 / Area -1
- Concavity: Caída o gravedad de las curvas de nivel.
- Concave Points: Número de sectores de contorno cóncavo.
- Symmetry: Simetría de la imagen
- Fractal dimensión: Aproximación de frontera – 1.

https://archive.ics.uci.edu/ml/datasets/Breast+Cancer+Wisconsin+(Diagnostic)

Características Generales:

En esta práctica, utilizaremos la clasificación por Máquinas de Soporte Vectorial, la cual consiste en una separación utilizando hiperplanos generados con vectores de soporte para dividir elementos dentro de un conjunto de datos. Se busca que la separación entre las dos clases sea la más amplia posible.

Desarrollo

Primeramente, tenemos que definir aquellas bibliotecas de Python que nos serán útiles para importar, limpiar y analizar los datos contenidos en el archivo separado por comas *WDBCOriginal.csv*. Estas serán: *pandas* (manipulación y análisis de datos), *matplotlib* (para la creación de gráficas y visualización de los datos), *numpy* para utilizar vectores y matrices de *n* dimensiones.

Más tarde, importamos el archivo *WDBCOriginal.csv* y lo primero que hacemos es guardarlo en un DataFrame. La importación del archivo se realizó a partir del explorador de archivos que abrimos utilizando el comando *files.upload()*. Una vez importados los datos, mostramos el DataFrame que los contiene:

```
BCancer = pd.read_csv('WDBCOriginal(1).csv')
BCancer
```

	IDNumber	Diagnosis	Radius	Texture	Perimeter	Area	Smoothness	Compactness	Concavity	ConcavePoints	Symmetry	FractalDimension
0	P-842302	M	17.99	10.38	122.80	1001.0	0.11840	0.27760	0.30010	0.14710	0.2419	0.07871
1	P-842517	M	20.57	17.77	132.90	1326.0	0.08474	0.07864	0.08690	0.07017	0.1812	0.05667
2	P-84300903	M	19.69	21.25	130.00	1203.0	0.10960	0.15990	0.19740	0.12790	0.2069	0.05999
3	P-84348301	M	11.42	20.38	77.58	386.1	0.14250	0.28390	0.24140	0.10520	0.2597	0.09744
4	P-84358402	M	20.29	14.34	135.10	1297.0	0.10030	0.13280	0.19800	0.10430	0.1809	0.05883
...	...	...	...	...	...	...	...	...	...	...	...	...
564	P-926424	M	21.56	22.39	142.00	1479.0	0.11100	0.11590	0.24390	0.13890	0.1726	0.05623
565	P-926682	M	20.13	28.25	131.20	1261.0	0.09780	0.10340	0.14400	0.09791	0.1752	0.05533
566	P-926954	M	16.60	28.08	108.30	858.1	0.08455	0.10230	0.09251	0.05302	0.1590	0.05648
567	P-927241	M	20.60	29.33	140.10	1265.0	0.11780	0.27700	0.35140	0.15200	0.2397	0.07016
568	P-92751	B	7.76	24.54	47.92	181.0	0.05263	0.04362	0.00000	0.00000	0.1587	0.05884

569 rows × 12 columns

Figura 1: Data Frame con algunos de los datos de tumores de mama.

Una vez hecho lo anterior, se hace una agrupación de los datos:

```
Diagnosis
B    357
M    212
dtype: int64
```

Figura 2: Agrupación de los datos según el diagnóstico.

Una vez hecho lo anterior, utilizamos una gráfica de dispersión para visualizar la ubicación de los elementos acorde a su área y su textura.

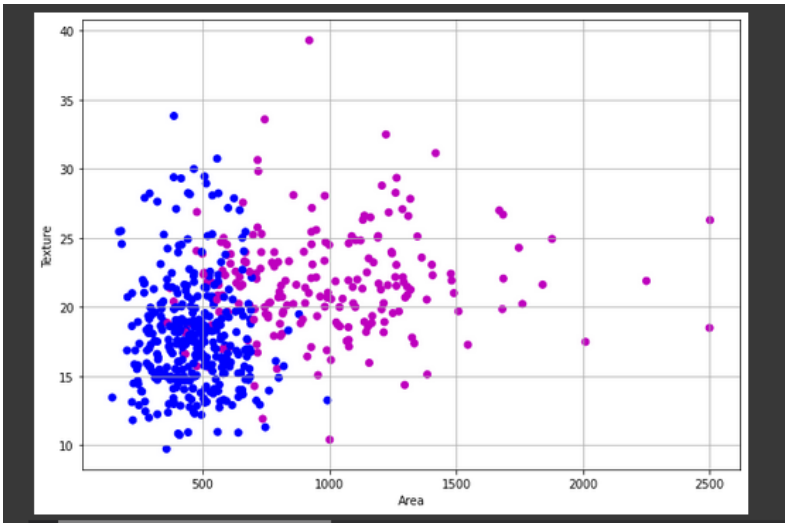

Figura 3: Dispersión de los datos.

H

Luego, creamos una matriz de correlaciones con el propósito de seleccionar variables significativas en el conjunto de datos y reducir la dimensionalidad de este. En la figura 4 se muestra la matriz de correlación de Pearson, utilizando colores para distinguir las variables que tienen una mayor dependencia y así poder realizar la selección de variables.

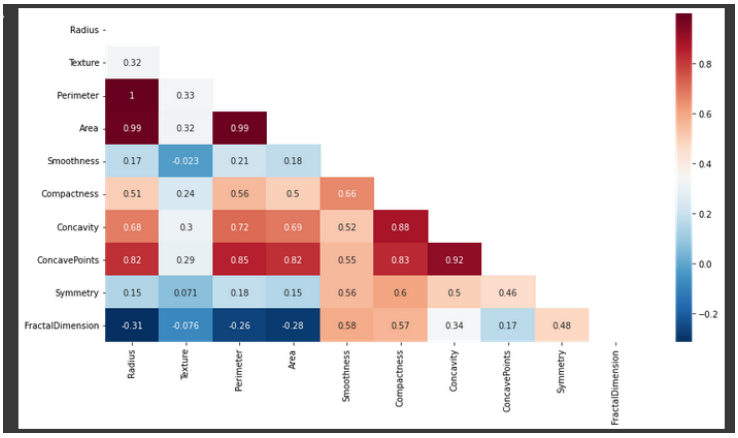

Figura 4: Matriz con las correlaciones entre variables.

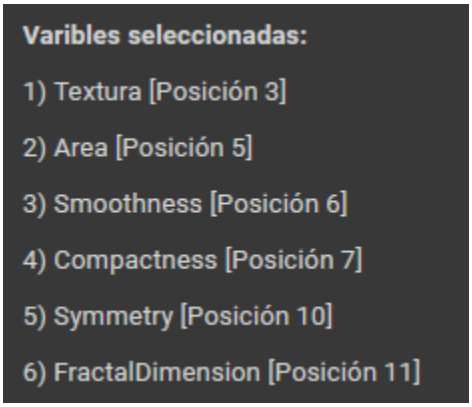

Figura 5: Variables seleccionadas

Definimos entonces un *array* con las variables predictoras que nos permitirán pronosticar el valor del diagnóstico en la clasificación, como se muestra en la figura 6.

```
#Variables predictoras
X = np.array(BCancer[['Texture',
                      'Area',
                      'Smoothness',
                      'Compactness',
                      'Symmetry',
                      'FractalDimension']])
pd.DataFrame(X)
```

	0	1	2	3	4	5
0	10.38	1001.0	0.11840	0.27760	0.2419	0.07871
1	17.77	1326.0	0.08474	0.07864	0.1812	0.05667
2	21.25	1203.0	0.10960	0.15990	0.2069	0.05999
3	20.38	386.1	0.14250	0.28390	0.2597	0.09744
4	14.34	1297.0	0.10030	0.13280	0.1809	0.05883
...	...	...	...	...	...	...
564	22.39	1479.0	0.11100	0.11590	0.1726	0.05623
565	28.25	1261.0	0.09780	0.10340	0.1752	0.05533
566	28.08	858.1	0.08455	0.10230	0.1590	0.05648
567	29.33	1265.0	0.11780	0.27700	0.2397	0.07016
568	24.54	181.0	0.05263	0.04362	0.1587	0.05884

569 rows × 6 columns

Figura 6: Parte de la matriz con las variables predictoras.

Además, definimos un vector con la variable clase o variable a clasificar *Diagnosis*, como se muestra en la figura 7.

```
#Variable clase
Y = np.array(BCancer[['Diagnosis']])
pd.DataFrame(Y)
```

	0
0	-1
1	-1
2	-1
3	-1
4	-1
...	...
564	-1
565	-1
566	-1
567	-1
568	1

569 rows × 1 columns

Figura 7: Parte de la matriz con la variable clase.

Para poder aplicar el algoritmo, importamos *SVC* (Support Vector Classifier), las funciones *classification_report*, *confusion_matrix*, *accuracy_score* y *model_selection*.

Realizamos la división de los datos para determinar el tamaño del conjunto de pruebas y del conjunto de entrenamiento, de manera similar a lo que se realizó en prácticas pasadas:

```
X_train, X_validation, Y_train, Y_validation = model_selection.train_test_split(X, Y,
                                                                                test_size = 0.2,
                                                                                random_state = 0,
                                                                                shuffle = True)
```

+ Code + Text

```
[16] print(len(X_train))
     print(len(X_validation))

     455
     114
```

Figura 7: División de los datos.

Para esta práctica, utilizaremos varios modelos de clasificación diferentes: Lineal, Polinomial, RBF y Sigmoide. Estos son distintos tipos de kernel que nos permitirán encontrar distintas transformaciones para el conjunto de datos, con el objetivo de encontrar el hiperplano que mejor clasifique los datos (minimizando el error de clasificación).

Comenzamos entonces con el modelo SVM lineal, donde, luego de aplicar el algoritmo y entrenar al modelo utilizando las siguientes líneas de código:

```
#Se declara el tipo de kernel y se entrena el modelo
ModeloSVM_1 = SVC(kernel='linear')
ModeloSVM_1.fit(X_train, Y_train) #entrenamiento del modelo, con 455 elementos de entrada definidos anteriormente para id vectorees de sc

/usr/local/lib/python3.7/dist-packages/sklearn/utils/validation.py:985: DataConversionWarning: A column-vector y was passed when a 1d arra
  y = column_or_1d(y, warn=True)
SVC(kernel='linear')
```

Figura 8: Entrenamiento del modelo

obtuvimos una exactitud promedio de

```
#Se calcula la exactitud promedio de la validación
ModeloSVM_1.score(X_validation, Y_validation)

0.9122807017543859
```

Figura 9: Exactitud del modelo.

Validando el modelo, vemos en la matriz de confusión:

Clasificación	-1	1
Real		
-1	41	6
1	4	63

Figura 10: Matriz de confusión.

Y los siguientes valores:

```
#Reporte de la clasificación
print("Exactitud", ModeloSVM_1.score(X_validation, Y_validation))
print(classification_report(Y_validation, Clasificaciones_1))
```

```
Exactitud 0.9122807017543859
              precision    recall  f1-score   support

          -1       0.91      0.87      0.89        47
           1       0.91      0.94      0.93        67

    accuracy                           0.91       114
   macro avg       0.91      0.91      0.91       114
weighted avg       0.91      0.91      0.91       114
```

Figura 11: Validación del modelo lineal. .

Además, obtenemos los valores de soporte que se utilizaron para el cálculo del hiperplano que clasifica los datos.

Hacemos el mismo procedimiento para los distintos tipos de kernel utilizados:

Para el polinomial, tenemos:

```
#Se declara el tipo de kernel y se entrena el modelo    # a ver si podemos mejorear ocn otra funcion
ModeloSVM_2 = SVC(kernel='poly')
ModeloSVM_2.fit(X_train, Y_train)

#Predeterminado: degree=3
```

```
/usr/local/lib/python3.7/dist-packages/sklearn/utils/validation.py:985: DataConversionWarning: A column-vec
  y = column_or_1d(y, warn=True)
SVC(kernel='poly')
```

```
#Se calcula la exactitud promedio de la validación
ModeloSVM_2.score(X_validation, Y_validation)
```

```
0.8859649122807017
```

Clasificación	-1	1
Real		
-1	35	12
1	1	66

```
Exactitud 0.8859649122807017
              precision    recall  f1-score   support

          -1       0.97      0.74      0.84        47
           1       0.85      0.99      0.91        67

    accuracy                           0.89       114
   macro avg       0.91      0.86      0.88       114
weighted avg       0.90      0.89      0.88       114
```

Figura 12: Modelo polinomial.

Para el kernel RBF:

```
#Se declara el tipo de kernel y se entrena el modelo
ModeloSVM_3 = SVC(kernel='rbf')
ModeloSVM_3.fit(X_train, Y_train)
```

```
/usr/local/lib/python3.7/dist-packages/sklearn/utils/validat
  y = column_or_1d(y, warn=True)
SVC()
```

```
[71] #Se calcula la exactitud promedio de la validación
     ModeloSVM_3.score(X_validation, Y_validation)

     0.8771929824561403
```

Clasificación	-1	1
Real		
-1	34	13
1	1	66

```
Exactitud 0.8771929824561403
              precision    recall  f1-score   support

          -1       0.97      0.72      0.83        47
           1       0.84      0.99      0.90        67

    accuracy                           0.88       114
   macro avg       0.90      0.85      0.87       114
weighted avg       0.89      0.88      0.87       114
```

Figura 13: Validación del modelo RBF.

Para el kernel sigmoide:

```
#Se declara el tipo de kernel y se entrena el modelo
ModeloSVM_4 = SVC(kernel='sigmoid')
ModeloSVM_4.fit(X_train, Y_train)
```

```
/usr/local/lib/python3.7/dist-packages/sklearn/utils/val
  y = column_or_1d(y, warn=True)
SVC(kernel='sigmoid')
```

```
[78] #Se calcula la exactitud promedio de la validación
     ModeloSVM_4.score(X_validation, Y_validation)

     0.45614035087719296
```

Figura 14: Validación del modelo sigmoide.

Observamos entonces que los diferentes modelos tienen una respuesta diferente ante este conjunto de datos. El modelo lineal, por ejemplo, obtuvo una exactitud promedio del 91.2%, mientras que el polinomial tuvo una exactitud promedio menor (88.59%), al igual que el RBF (87.7%) y que el sigmoide (45.6%).

No obstante, la precisión con la que clasificó los elementos en las categorías “1” o “-1”, no correspondió exactamente con aquellos que tuvieron mayor exactitud promedio, sino que el RBF, por ejemplo, tuvo una precisión del 97% para los “-1” y del 84% para los “1”. Comparándolo con el modelo lineal, podemos ver que el modelo RBF tuvo mayor precisión.

Al finalizar con los modelos, podemos realizar nuevas clasificaciones a partir de valores del tumor, por ejemplo, para una nueva clasificación:

Nuevas clasificaciones

```
[79] #Paciente P-842302 -Tumor Maligno-
     PacienteID111 = pd.DataFrame({'Texture': [10.38],
                                   'Area': [1001.0],
                                   'Smoothness': [0.11840],
                                   'Compactness': [0.27760],
                                   'Symmetry': [0.2419],
                                   'FractalDimension': [0.07871]})
     ModeloSVM_1.predict(PacienteID111)
```

```
/usr/local/lib/python3.7/dist-packages/sklearn/base.py:439: UserWarning: X has feature names, but SVC was fitted without feature names
  f"X has feature names, but {self.__class__.__name__} was fitted without"
array([-1])
```

```
[ ]
```

Figura 15: Nuevas clasificaciones con el modelo lineal.

Lo anterior se podría realizar para cualquiera de los modelos presentados anteriormente.

Conclusiones

A lo largo de esta práctica, tuvimos como principal objetivo el obtener clasificaciones del diagnóstico de tumores utilizando máquinas de soporte vectorial. Al obtener la validación de los distintos modelos creados, notamos que tuvimos una mayor exactitud en el modelo lineal (91.2%), lo cual representaría un hiperplano en forma de línea que clasifica los datos entre los que podrían ser tumores malignos y los que son tumores benignos.

En general, creo que se cumplieron los objetivos de la práctica y que ésta fue útil para comprender de mejor manera cada uno de los kernels utilizados para las máquinas de soporte vectorial, así como sus diferencias en cuanto a exactitud y precisión en la clasificación.