Universidad de Chile
Facultad de Ciencias Físicas y Matemáticas
Departamento de Ingeniería Eléctrica

Informe tarea 2

Support Vector Machine (SVM)

Integrantes: Bastián Garcés
Profesor: Javier Ruiz del Solar
Auxiliar: Patricio Loncomilla Z.
Ayudantes: Juan Pablo Cáceres B.
Rudy García
Rodrigo Salas O.
Sebastian Solanich
Pablo Troncoso P.

Fecha de entrega: 28 de Abril de 2021
Santiago, Chile

Índice de Contenidos

Índice de Figuras

Índice de Códigos

1. Introducción

En el presente informe se detallará la tarea número 2 del curso EL4106 Inteligencia Computacional, dicha tarea consiste en la implementación de 4 tipos de versiones de *Support Vector Machine*, las versiones a implementar son SVM con kernel lineal, otro con kernel polinomial pero para dos grados de polinomio diferentes y otro con kernel rbf. Para realizar dicha tarea se dispondra del conjunto de datos *Magic Gamma Telescope Data Set*, conjunto de datos que contiene muestras con datos de clase hadrón y de clase no-hadrón (rayos Gamma). La herramienta utilizada para la resolución de la tarea será *Google Colaboratory*, un entorno web capaz de simular notebooks de *Python*, además se usarán las bibliotecas *Pandas*, *Numpy*, *MatplotLib* y *Sklearn*.

Las secciones de las que dispondrá el informe se enuncian a continuación:

- **Actividad 1:** En esta actividad se deberá implementar un código que sea capaz de leer el dataset "Magic04" entregado por el cuerpo docente.
- **Actividad 2:** En la segunda actividad se deberá re-muestrear el dataset entregado, tomando 3000 muestras de clase positiva (hadrón) y otras 3000 muestras de clase negativa (no-hadrón).
- **Actividad 3:** En la tercera actividad se deberá separar el conjunto de 6000 muestras obtenidas en la actividad anterior, para hacer esto se tomará el 60 % del conjunto obtenido para obtener al conjunto de entrenamiento, luego otro 20 % se usará para definir al conjunto de validación y con el 20 % restante se definirá al conjunto de prueba, cabe destacar que estos conjuntos deberán ser representativos, es decir, deberán contener la misma cantidad de muestras tipo hadrón y tipo no-hadrón.
- **Actividad 4:** En esta actividad se deberá entrenar un *StandardScaler*, para realizar este entrenamiento se usará el conjunto de entrenamiento, la finalidad de entrenar al *StandardScaler* es la de normalizar las características.
- **Actividad 5:** En la quinta actividad se deberá entrenar un SVM lineal utilizando el conjunto de entrenamiento, sin embargo, para lograr obtener un buen clasificador se utilizará una grilla, grilla encargada de encontrar los mejores hiper parámetros del clasificador, para crear la grilla se deberá recurrir utilizar la función *model_selection.GridSearchCV()*.
- **Actividad 6:** Lo que se debe realizar en esta actividad es evaluar el clasificador obtenido sobre el conjunto de validación y generar las matrices de confusión, se obtendrán dos matrices, una será la versión normalizada y la otra sin normalizar.
- **Actividad 7:** En esta actividad se deberá obtener la curva ROC del clasificador sobre el conjunto de validación y obtener el área bajo la curva.
- **Actividad 8:** En la octava actividad se deberá generar la curva de precision-recall y obtener el average precision score, que corresponde al área bajo la curva de precision-recall.
- **Actividad 9:** En la novena actividad se tendrán que repetir las actividades 5, 6, 7 y 8 pero utilizando SVM con kernel polinomial, se deberán utilizar dos grados de polinomio diferente, es decir, se tendrán que generar dos SVM.
- **Actividad 10:** En la penúltima actividad se tendrán que volver a repetir las actividades 5, 6, 7 y 8 pero esta vez utilizando un SVM con kernel rbf.

- **Actividad 11:** Finalmente se deberán obtener las métricas de las actividades 6, 7 y 8 para los SVM implementados pero utilizando el conjunto de prueba.

2. Desarrollo

En las siguientes subsecciones se exhibirá el código implementado, así como también se explicará que es lo que hace el código y demás. Se debe mencionar que en esta sección no se volverá a decir que es lo que se debe hacer en cada actividad, eso ya fue hecho en la introducción, por lo cual solo se explicará el código en cuestión.

2.1. Actividad 1

El código implementado para leer el dataset entregado por el cuerpo docente es el que se puede apreciar en el código 1.

Código 1: Código utilizado para subir los archivos.

```
from google.colab import files
uploaded = files.upload()
```

Luego al dataframe fue leído mediante la función *read_csv()* de *Pandas*, utilizando como primer parámetro el nombre del archivo subido, nombre que corresponde al de *'magic04'* y como segundo parámetro una lista que contiene los nombres de las columnas, dichos nombres corresponden a: *fLength*, *fWidth*, *fSize*, *fConc*, *fConc1*, *fAsym*, *fM3Long*, *fM3Trans*, *fAlpha*, *fDist* y *class*. Finalmente el dataframe leído fue llamado como df.

2.2. Actividad 2

El código utilizado para obtener las muestras de hadrón y no-hadrón corresponde al código 2.

Código 2: Código utilizado para realizar el muestreo.

```
#Primero separamos el dataframe en hadron y no-hadron
H = df[df['class'] == 'h'].reset_index(drop=True)
NH = df[df['class'] == 'g'].reset_index(drop=True)

#Volvemos a desordenar los conjuntos obtenidos
H_des = H.sample(frac=1).reset_index(drop=True)
NH_des = NH.sample(frac=1).reset_index(drop=True)

Cantidad = int(3000) #Cantidad de muestras de cada tipo deseadas
#Seleccionamos 3000 muestras de cada uno
MuestrasH = H_des.sample(n=Cantidad).reset_index(drop=True)
MuestrasNH = NH_des.sample(n=Cantidad).reset_index(drop=True)

#Unimos ambos dataframes
Muestreo = pd.concat([MuestrasH,MuestrasNH]).reset_index(drop=True)
```

El procedimiento realizado consiste primero en separar el dataframe df en hadrón y no-hadrón y una vez separados se desordenan las muestras usando la función *sample(frac=1)*, esto se hace para evitar sesgos a la hora de elegir muestras. Una vez desordenados los conjuntos se define una cantidad, dicha cantidad representa la cantidad de muestras de cada clase que se desean, en este caso 3000, luego utilizando la función *sample(n=Cantidad)* se pueden obtener las muestras deseadas de

los conjuntos *H_des* y *NH_des*, finalmente solo se utiliza la función *concat()* de la librería *Pandas* para unir ambos dataframes obtenidos. Cabe destacar que se utilizó en repetidas ocasiones la función *reset_index(drop=True)* esto se hace para evitar que los datos conserven los índices que tenían previamente.

2.3. Actividad 3

Para obtener los conjuntos de entrenamiento, validación y prueba se utilizó el código 3.

Código 3: Código utilizado para obtener los distintos conjuntos.

```
#Puntos de corte
CorEntre = int(Cantidad*0.6)
CorVali = int(Cantidad*0.8)

#Conjuntos
Entrena = pd.concat([MuestrasH.iloc[:CorEntre,:],MuestrasNH.iloc[:CorEntre,:]]).reset_index(drop=
    ↪ True)
Vali = pd.concat([MuestrasH.iloc[CorEntre:CorVali,:],MuestrasNH.iloc[CorEntre:CorVali,:]]).
    ↪ reset_index(drop=True)
Prue = pd.concat([MuestrasH.iloc[CorVali:,:],MuestrasNH.iloc[CorVali:,:]]).reset_index(drop=True)
```

Donde se puede ver que primero se definieron los puntos de corte a utilizar, estos definirán la forma en que se obtendrán los diferentes conjuntos, el primer punto de corte permitirá elegir el 60 % de los datos, el segundo punto de corte permitirá elegir el 20 % de los datos, quedando un 20 % sobrante. Luego mediante la función *concat()* se obtendrán los conjuntos, la forma en que se obtienen es: primero del conjunto que contiene a las muestras tipo hadrón se tomarán los elementos desde el inicio del dataframe hasta el primer punto de corte, en ese punto de corte se tendría el 60 % de los datos tipo hadrón y se uniría con el 60 % de los datos tipo no-hadrón, luego el conjunto de validación se obtendría tomando los datos del conjunto de muestras tipo hadrón desde el primer punto de corte hasta el segundo y se uniría con su símil pero en el conjunto de muestras tipo no-hadrón y finalmente el conjunto de prueba se obtendría tomando las muestras del conjunto hadrón desde el segundo punto de corte hasta el final, uniéndolo con su equivalente pero en el conjunto de muestras tipo no-hadrón.

2.4. Actividad 4

Para realizar el entrenamiento del *StandardScaler* se utilizó el código 4.

Código 4: Código utilizado para obtener entrenar el *StandardScaler*.

```
#Creamos el StandardScaler
scaler = sklearn.preprocessing.StandardScaler()

#Entrenamos el scaler
scaler.fit(Entrena.iloc[:,:10])

#Aplicamos el scaler a los distintos conjuntos
S_Entrena = scaler.transform(Entrena.iloc[:,:10])
S_Vali = scaler.transform(Vali.iloc[:,:10])
S_Prue = scaler.transform(Prue.iloc[:,:10])
```

```
11
12 #Obtenemos la columna clase de cada conjunto
13 #cambiamos los 'g' por 0 y los 'h' por 1
14 #y luego se pasan a array
15 Entrena_class = Entrena.iloc[:,10].replace({'g' : 0,'h' : 1}).to_numpy()
16 Vali_class = Vali.iloc[:,10].replace({'g' : 0,'h' : 1}).to_numpy()
17 Prue_class = Prue.iloc[:,10].replace({'g' : 0,'h' : 1}).to_numpy()
```

En dicho código primero se define el *StandardScaler* y se entrena sobre las características del conjunto de entrenamiento utilizando la función *fit()*, acá es relevante que solo deben considerarse las características, la columna que contiene a la clase no es considerada, es por esto que se utilizó a la función *iloc()*. Una vez entrenado el *StandardScaler* se aplica sobre las características de los conjuntos de entrenamiento, validación y prueba mediante la función *transform()*, al hacer esto se están normalizando las características de los distintos conjuntos. Finalmente se obtuvo la columna que contiene a la clase de cada conjunto y se reemplazaron las 'g' (no-hadrón) por un 0 y las 'h' (hadrón) por un 1, esto se realizó mediante la función *replace()*, finalmente se pasó a las columnas con las clases en cada conjunto a un array mediante la función *to_numpy()*.

2.5. Actividad 5

Para definir el SVM lineal y entrenar la grilla para encontrar los mejores hiper parámetros se utilizó el código 5.

Código 5: Entrenamiento de la grilla para el SVM lineal.

```
1 #Creamos el SVM lineal
2 SVM_lin = sklearn.svm.SVC(kernel='linear',probability=False)
3
4 #Creamos lista de parametros C que probara el Gridsearch
5 param_lin = {'C': [0.01,0.1,1,10,100]}
6 grid_lin = sklearn.model_selection.GridSearchCV(SVM_lin,param_lin,cv=5)
7
8 #Entrenamos con el test de entrenamiento
9 grid_lin.fit(S_Entrena,Entrena_class)
10 print(' ')
```

Primero se creó el SVM lineal, esto se hizo mediante la función *svm.SVC()* de *sklearn*, dandole como primer parámetro el kernel deseado, en este caso lineal e indicando como segundo parámetro probability=False. Luego se pasó a crear la grilla que buscaría los mejores hiper parámetros y definirá el mejor clasificador en base a estos, para ello se utilizó la función *model_selection.GridSearchCV()*, función a la que se le daba como parámetros el SVM al que se le quieren buscar los hiper parámetros, un diccionario que contiene los candidatos a hiper parámetros, en este caso el hiper parámetro corresponde a C y la grilla debe ver entre los valores 0.01, 0.1, 1, 10 y 100 cual es el más apropiado y finalmente el último parámetro corresponde a un número que indica la estrategia de división. Posteriormente se entrenó la grilla obtenida mediante el uso de la función *fit()*, función que recibe como parámetros el conjunto que desea clasificarse, en este caso el conjunto de entrenamiento normalizado, y la correcta clasificación del conjunto que se le entregó.

2.6. Actividad 6

Para obtener las matrices de confusión del SVM lineal sobre el conjunto de validación se utilizó el código 6.

Código 6: Matrices de confusión para el SVM lineal.

```
#Evaluamos sobre el conjunto de validacion normalizado
predict_lin = grid_lin.predict(S_Vali) #<-- De aca obtenemos las predicciones del clasificador

#Generamos las matrices de confusion
#La version normalizada:
Matrix_lin_N = sklearn.metrics.confusion_matrix(Vali_class,predict_lin,normalize='true')
#La version sin normalizar:
Matrix_lin_NoN = sklearn.metrics.confusion_matrix(Vali_class,predict_lin)
print('La matriz de confusion normalizada es:')
print(Matrix_lin_N)
print('La matriz de confusion sin normalizar es:')
print(Matrix_lin_NoN)
```

El código anterior opera como sigue: primero se ve la predicción que hace el clasificador sobre el conjunto de validación normalizado, para esto se utilizó la función *predict()*, función que retorna un array con las clasificaciones que se hicieron para cada elemento del conjunto que se le entregó. Una vez realizada esta predicción se utilizó la función *metrics.confusion_matrix()* función a la que se le entrega la correcta clasificación del conjunto (en este caso correspondería a la columna clase de los datos) , como segundo parámetro recibe la predicción realizada por el clasificador y finalmente recibe un string que indica si se desea normalizada o no la matriz a obtener, al utilizar el string *'true'* se está indicando que se desea normalizar por la cantidad de datos hadrón o no-hadrón que tenga el conjunto según corresponda, con esto se obtuvieron dos matrices, una sería la versión normalizada y la otra sería la versión sin normalizar. Las matrices de confusión obtenidas para el SVM lineal sobre el conjunto de validación se pueden apreciar en la figura 1.

```
La matriz de confusion normalizada es:
[[0.84833333 0.15166667]
 [0.26       0.74      ]]
La matriz de confusion sin normalizar es:
[[509  91]
 [156 444]]
```

Figura 1: Matrices de confusión del SVM lineal.

2.7. Actividad 7

Con la finalidad de obtener los parámetros de la curva ROC se hizo uso del código 7.

Código 7: Obtención de parámetros de la curva ROC para el SVM lineal.

```
#Obtenemos la funcion de decision
deci_lin = grid_lin.decision_function(S_Vali)

#Obtenemos la curva ROC
```

```
TFP_lin,TVP_lin,threshold_lin = sklearn.metrics.roc_curve(Vali_class,deci_lin)

#Obtenemos el area bajo la curva
Area_lin = sklearn.metrics.auc(TFP_lin,TVP_lin)
print('El area bajo la curva ROC es de: ',str(Area_lin))

#Graficamos la curva ROC
plt.plot(TFP_lin,TVP_lin)
plt.title('Curva ROC del SVM lineal')
plt.xlabel('Tasa falsos positivos')
plt.ylabel('Tasa verdaderos positivos')
plt.grid()
```

La lógica tras el código 7 es primero obtener la función de decisión que sigue el clasificador a la hora de clasificar al conjunto de validación, para hacer esto se hace uso de la función *decision_function()*, función a la que se le entrega como parámetro el conjunto que se le desea obtener su función de decisión. Una vez realizado lo anterior se utilizó la función *metrics.roc_curve()* de *sklearn*, este función recibe 2 parámetros y retorna 3 arrays, el primer parámetro que recibe corresponde a la clasificación correcta del conjunto que fue clasificado por el SVM y como segundo parámetro recibe la función de decisión calculada anteriormente, en cuanto a los outputs que entrega la función estos son la tasa de falsos positivos, la tasa de verdaderos positivos y los umbrales utilizados para calcular las tasas. Con los parámetros de la curva ROC obtenidos se procedió a calcular el área bajo la curva ROC, para esto se hizo uso de la función *metrics.auc()*, función que recibe como primer parámetro el "eje x" de la curva ROC (La tasa de falsos positivos) y como segundo parámetro el "eje y" de la curva ROC (La tasa de verdaderos positivos). Finalmente solo se implementó un código que permitía ver la curva ROC. La curva ROC obtenida, así como el área bajo la curva de esta se puede apreciar en la figura 2.

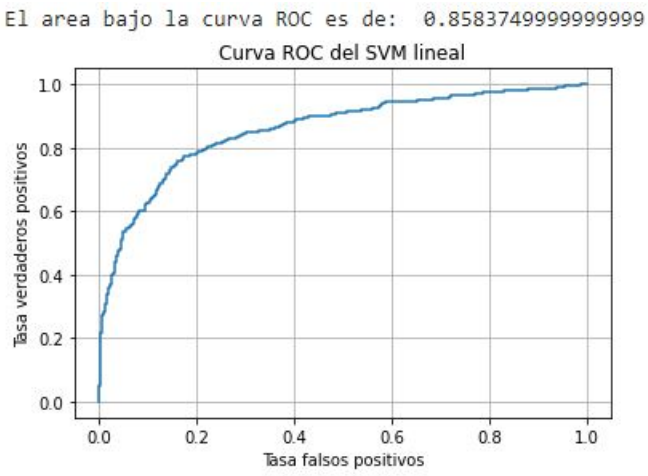

Figura 2: Curva ROC y área bajo la curva.

2.8. Actividad 8

Para obtener la curva de precision-recall, así como también el average precision score (área bajo la curva de precision-recall) se implementó el código 8.

Código 8: Obtención de parámetros de la curva precision-recall.

```
#Obtenemos la curva de precision-recall
P_lin, R_lin, threshold_lin_PR =  sklearn.metrics.precision_recall_curve(Vali_class,deci_lin)

#Obtenemos el average precision
prom_lin = sklearn.metrics.average_precision_score(Vali_class,deci_lin)
print('El average precision score es: ',str(prom_lin))

#Graficamos la curva precision-recall
plt.plot(R_lin,P_lin)
plt.title('Curva precision-recall del SVM lineal')
plt.xlabel('Recall')
plt.ylabel('precision')
plt.grid()
```

El procedimiento realizado para obtener la curva de precision-recall es análogo al realizado para obtener la curva ROC pero utilizando otras funciones. Para obtener los parámetros de la curva de precision-recall se utilizó la función *metrics.precision_recall_curve()* de la librería *sklearn*, función que recibe los mismos parámetros que la función que permite obtener los parámetros de la curva ROC y retorna como primer parámetro el precision, como segundo parámetro el recall y como último parámetro los umbrales utilizados para la obtención del precision y el recall. Luego se utilizó la función *metrics.average_precision_score()* que recibe los mismos parámetros que la función previa y entrega el área bajo la curva precision-recall, finalmente solo se implementó un código que fuese capaz de graficar la curva precision-recall obtenida. La curva obtenida, así como el average precision score se pueden apreciar en la figura 3.

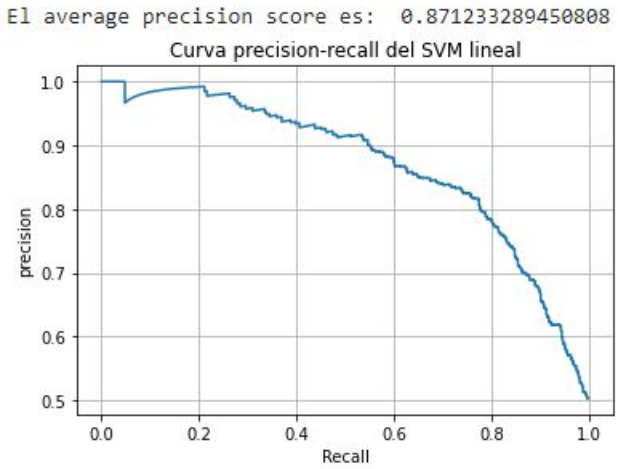

Figura 3: Curva precision-recall y average precision score.

2.9. Actividad 9

En esta actividad se tenían que implementar 2 SVMs, ambos polinomiales pero de distintos grados, el código implementado así como las explicaciones pertinentes vienen en las subsecciones que siguen.

2.9.1. Primer SVM polinomial

Para la implementación del primer SVM polinomial se hizo uso del código 9, donde se implementó un SVM polinomial de grado 4.

Código 9: Implementación del primer SVM polinomial y sus respectivas métricas.

```
#Creamos el SVM polinomial de grado 4
SVM_pol4 = sklearn.svm.SVC(kernel='poly',degree=4,probability=False)

#Creamos diccionario de parametros que usara la grilla para ambos clasificadores
param_pol = {'C' : [0.01,0.1,1,10],
             'gamma' : [0.0001,0.001,0.01,0.1]}
grid_pol4 = sklearn.model_selection.GridSearchCV(SVM_pol4,param_pol,cv=5)

#Entrenamos el grid con el test de entrenamiento
grid_pol4.fit(S_Entrena,Entrena_class)

#Vemos las predicciones que hacen los clasificadores
predict_pol4 = grid_pol4.predict(S_Vali)

#Obtenemos las matrices de confusion:
#Version normalizada:
Matrix_pol4_N = sklearn.metrics.confusion_matrix(Vali_class,predict_pol4,normalize='true')
#Version no normalizada:
Matrix_pol4_NoN = sklearn.metrics.confusion_matrix(Vali_class,predict_pol4)
print('La matriz de confusion normalizada es:')
print(Matrix_pol4_N)
print('La matriz de confusion sin normalizar es:')
print(Matrix_pol4_NoN)

#Obtenemos la funcion de decision
deci_pol4 = grid_pol4.decision_function(S_Vali)

#Obtenemos los parametros de la curva ROC
TFP_pol4,TVP_pol4,threshold_pol4 = sklearn.metrics.roc_curve(Vali_class,deci_pol4)

#Obtenemos el area bajo la curva ROC
Area_pol4 = sklearn.metrics.auc(TFP_pol4,TVP_pol4)
print('El area bajo la curva ROC es de: ',str(Area_pol4))

#Graficamos las curvas ROC
plt.figure(1)
plt.title('Curva ROC del SVM polinomial grado 4')
plt.xlabel('Tasa falsos positivos')
```

```
plt.ylabel('Tasa verdaderos positivos')
plt.grid()
plt.plot(TFP_pol4,TVP_pol4)

#Obtenemos los parametros de la curva de precision-recall
P_pol4, R_pol4, threshold_pol4_PR = sklearn.metrics.precision_recall_curve(Vali_class,deci_pol4)

#Obtenemos el average precision
prom_pol4 = sklearn.metrics.average_precision_score(Vali_class,deci_pol4)
print('El average precision score es: ',str(prom_pol4))

#Graficamos la curva precision-recall
plt.figure(2)
plt.title('Curva precision-recall para el SVM polinomial grado 4')
plt.xlabel('Recall')
plt.ylabel('Precision')
plt.grid()
plt.plot(R_pol4,P_pol4)
```

Al observar el código se puede apreciar que es análogo al implementado durante las actividades 5, 6, 7 y 8 pero cambiando parámetros, se puede apreciar que ahora el SVM es de kernel polinomial y que el polinomio es de grado 4, luego al implementar la grilla se utilizaron 4 valores para C y otros 4 para gamma como posibles valores de hiper parámetros, lo que sigue del código es análogo a lo visto para el SVM lineal pero renombrando variables. Las métricas obtenidas en este caso se pueden apreciar en la figura 4, 5 y 6.

```
La matriz de confusion normalizada es:
[[0.93666667 0.06333333]
 [0.35166667 0.64833333]]
La matriz de confusion sin normalizar es:
[[562  38]
 [211 389]]
El area bajo la curva ROC es de:  0.8813361111111112
El average precision score es:  0.8984564038286467
```

Figura 4: Matrices de confusión del SVM polinomial de grado 4.

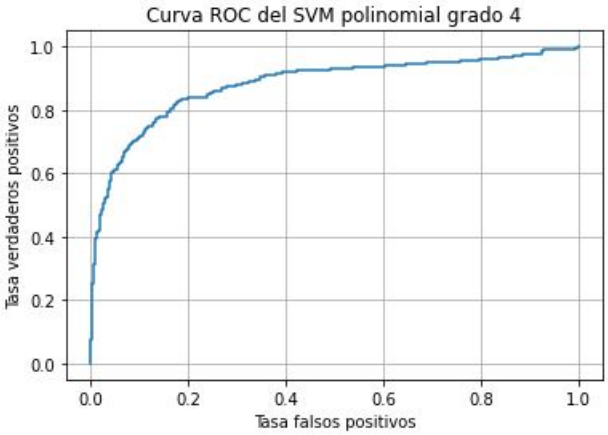

Figura 5: Curva ROC del SVM polinomial de grado 4.

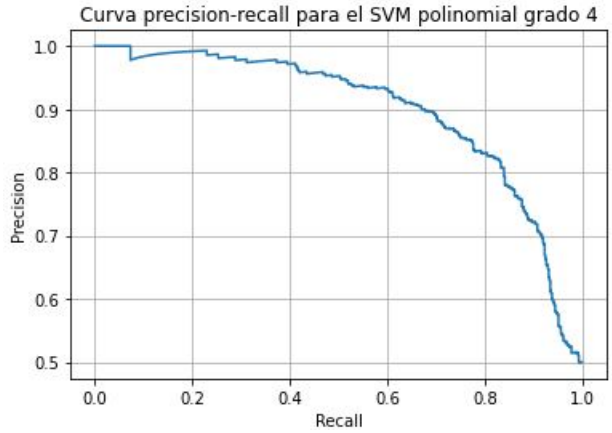

Figura 6: Curva de precision-recall del SVM polinomial de grado 4.

2.9.2. Segundo SVM polinomial

El segundo SVM implementado corresponde a un SVM polinomial de grado 7. La implementación realizada corresponde a la del código 10.

Código 10: Implementación del segundo SVM polinomial y sus respectivas métricas.

```
#Creamos el SVM polinomial de grado 7
SVM_pol7 = sklearn.svm.SVC(kernel='poly',degree=7,probability=False)

#Grilla del clasificador
grid_pol7 = sklearn.model_selection.GridSearchCV(SVM_pol7,param_pol,cv=5)

#Entrenamos el grid con el conjunto de entrenamiento
grid_pol7.fit(S_Entrena,Entrena_class)

#Vemos las predicciones que hace el clasificador
```

```
predict_pol7 = grid_pol7.predict(S_Vali)

#Obtenemos las matrices de confusion para el clasificador
#Version normalizada:
Matrix_pol7_N = sklearn.metrics.confusion_matrix(Vali_class,predict_pol7,normalize='true')
#Version no normalizada:
Matrix_pol7_NoN = sklearn.metrics.confusion_matrix(Vali_class,predict_pol7)
print('La matriz de confusion normalizada es:')
print(Matrix_pol7_N)
print('La matriz de confusion sin normalizar es:')
print(Matrix_pol7_NoN)

#Obtenemos la funcion de decision del clasificador
deci_pol7 = grid_pol7.decision_function(S_Vali)

#Obtenemos los parametros de la curva ROC
TFP_pol7,TVP_pol7,threshold_pol7 = sklearn.metrics.roc_curve(Vali_class,deci_pol7)

#Obtenemos el area bajo la curva ROC
Area_pol7 = sklearn.metrics.auc(TFP_pol7,TVP_pol7)
print('El area bajo la curva ROC es de: ',str(Area_pol7))

#Graficamos la curva ROC
plt.figure(1)
plt.plot(TFP_pol7,TVP_pol7)
plt.title('Curva ROC del SVM polinomial grado 7')
plt.xlabel('Tasa falsos positivos')
plt.ylabel('Tasa verdaderos positivos')
plt.grid()

#Obtenemos los parametros de la curva de precision-recall
P_pol7, R_pol7, threshold_pol7_PR = sklearn.metrics.precision_recall_curve(Vali_class,deci_pol7)

#Obtenemos el average precision
prom_pol7 = sklearn.metrics.average_precision_score(Vali_class,deci_pol7)
print('El average precision score es: ',str(prom_pol7))

#Graficamos la curva precision-recall
plt.figure(2)
plt.plot(R_pol7,P_pol7)
plt.title('Curva precision-recall para el SVM polinomial grado 7')
plt.xlabel('Recall')
plt.ylabel('Precision')
plt.grid()
```

Este código es exactamente el mismo implementado para el SVM polinomial de grado 4 pero reemplazando el grado del polinomio por 7, el resto del código es idéntico pero con otros nombres en las variables. Las métricas obtenidas son las que se observan en la figura 7, 8 y 9.

```
La matriz de confusion normalizada es:
[[0.96  0.04 ]
 [0.515 0.485]]
La matriz de confusion sin normalizar es:
[[576  24]
 [309 291]]
El area bajo la curva ROC es de:  0.8449722222222222
El average precision score es:  0.8660174983043489
```

Figura 7: Matrices de confusión del SVM polinomial de grado 7.

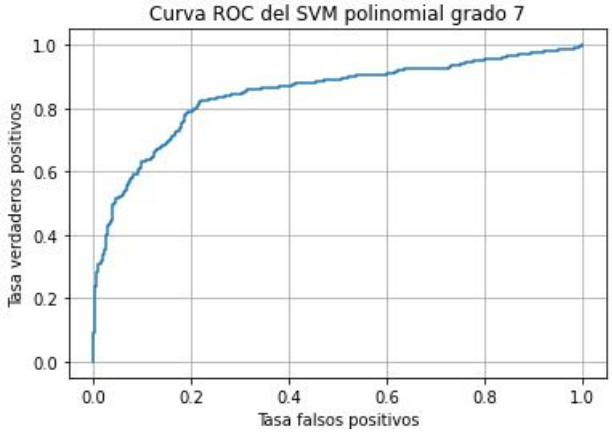

Figura 8: Curva ROC del SVM polinomial de grado 7.

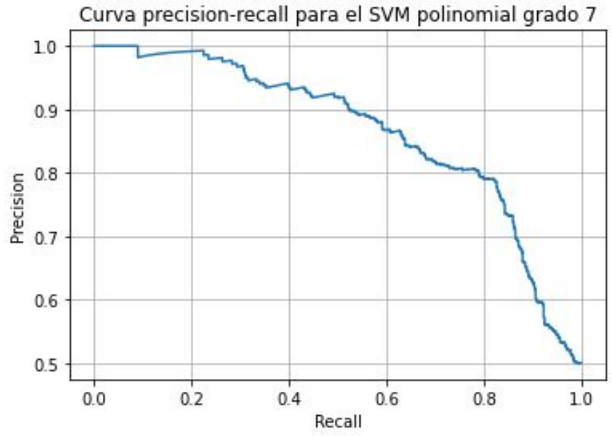

Figura 9: Curva de precision-recall del SVM polinomial de grado 7.

2.10. Actividad 10

La implementación del SVM con kernel rbf se puede observar en el código 11.

Código 11: Implementación SVM con kernel rbf y sus respectivas métricas.

```
1 #Creamos el SVM rbf
```

```
SVM_rbf = sklearn.svm.SVC(kernel='rbf',probability=False)

#Grilla del clasificador
param_rbf = {'C' : [0.01,0.1,1,10],
             'gamma' : [0.0001,0.001,0.01,0.1]}
grid_rbf = sklearn.model_selection.GridSearchCV(SVM_rbf,param_rbf,cv=5)

#Entrenamos el grid con el conjunto de entrenamiento
grid_rbf.fit(S_Entrena,Entrena_class)

#Obtenemos las predicciones que hace el clasificador
predict_rbf = grid_rbf.predict(S_Vali)

#Obtenemos las matrices de confusion
#La version normalizada:
Matrix_rbf_N = sklearn.metrics.confusion_matrix(Vali_class,predict_rbf,normalize='true')
#Version no normalizada:
Matrix_rbf_NoN = sklearn.metrics.confusion_matrix(Vali_class,predict_rbf)
print('La matriz de confusion normalizada es:')
print(Matrix_rbf_N)
print('La matriz de confusion sin normalizar es:')
print(Matrix_rbf_NoN)

#Obtenemos la funcion de decision
deci_rbf = grid_rbf.decision_function(S_Vali)

#Obtenemos los parametros de la curva ROC
TFP_rbf, TVP_rbf, threshold_rbf = sklearn.metrics.roc_curve(Vali_class,deci_rbf)

#Area bajo la curva ROC
Area_rbf = sklearn.metrics.auc(TFP_rbf,TVP_rbf)
print('El area bajo la curva ROC es de: ',str(Area_rbf))

#Obtenemos la curva ROC
plt.figure(1)
plt.title('Curva ROC del SVM rbf')
plt.xlabel('Tasa falsos positivos')
plt.ylabel('Tasa verdaderos positivos')
plt.grid()
plt.plot(TFP_rbf,TVP_rbf)

#Obtenemos los parametros para la curva precision-recall
R_rbf,P_rbf,threshold_rbf_PR = sklearn.metrics.precision_recall_curve(Vali_class,deci_rbf)

#Obtenemos el average precision score
prom_rbf = sklearn.metrics.average_precision_score(Vali_class,deci_rbf)
print('El average precision score es: ',str(prom_rbf))

#Graficamos la curva precision-recall
plt.figure(2)
plt.title('Curva precision-recall del SVM rbf')
```

```
plt.xlabel('Recall')
plt.ylabel('Precision')
plt.grid()
plt.plot(R_rbf,P_rbf)
```

Donde al igual que para el caso de los SVM polinomiales el código es análogo al del SVM lineal, simplemente se hizo un cambio a la hora de definir el SVM, eligiéndose ahora un kernel rbf, el resto del código es igual al de los SVM anteriores. Finalmente las métricas obtenidas se observan en la figura 10, 11 y 12.

```
La matriz de confusion normalizada es:
[[0.91       0.09      ]
 [0.18666667 0.81333333]]
La matriz de confusion sin normalizar es:
[[546  54]
 [112 488]]
El area bajo la curva ROC es de:  0.9280666666666666
El average precision score es:  0.9407432944318636
```

Figura 10: Matrices de confusión del SVM rbf.

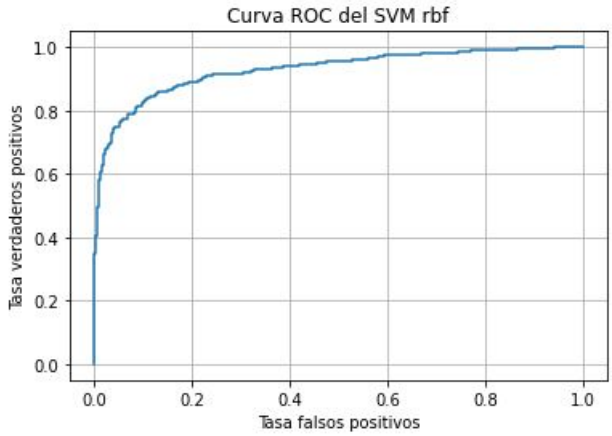

Figura 11: Curva ROC del SVM rbf.

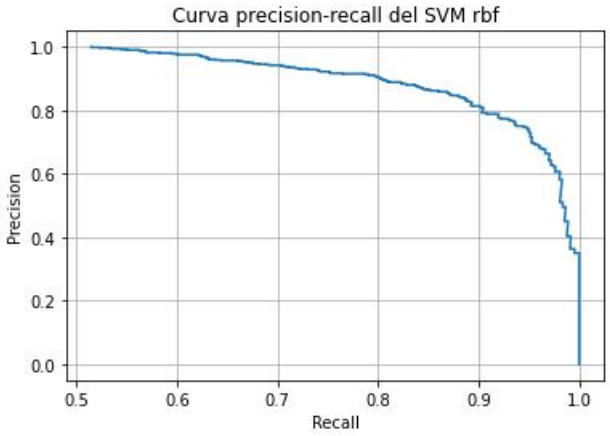

Figura 12: Curva de precision-recall del SVM rbf.

2.11. Actividad 11

Para la última actividad simplemente se debía volver a repetir el código mostrado para los 4 SVM pero sobre el conjunto de prueba, la implementación de esto se puede observar en el código 12, donde se puede observar que este código solo sirve para determinar las métricas sobre el conjunto de prueba, más no las muestras.

Código 12: Evaluación de los SVM sobre el conjunto de prueba.

```
#Obtenemos las predicciones que hacen los clasificadores
predict_lin_Prue = grid_lin.predict(S_Prue)
predict_pol4_Prue = grid_pol4.predict(S_Prue)
predict_pol7_Prue = grid_pol7.predict(S_Prue)
predict_rbf_Prue = grid_rbf.predict(S_Prue)

#Obtenemos las matrices de confusion que hacen los clasificadores
#La version normalizada:
Matrix_lin_Prue_N = sklearn.metrics.confusion_matrix(Prue_class,predict_lin_Prue,normalize='
    ↪ true')
Matrix_pol4_Prue_N = sklearn.metrics.confusion_matrix(Prue_class,predict_pol4_Prue,normalize
    ↪ ='true')
Matrix_pol7_Prue_N = sklearn.metrics.confusion_matrix(Prue_class,predict_pol7_Prue,normalize
    ↪ ='true')
Matrix_rbf_Prue_N = sklearn.metrics.confusion_matrix(Prue_class,predict_rbf_Prue,normalize='
    ↪ true')
#Version no normalizada:
Matrix_lin_Prue_NoN = sklearn.metrics.confusion_matrix(Prue_class,predict_lin_Prue)
Matrix_pol4_Prue_NoN = sklearn.metrics.confusion_matrix(Prue_class,predict_pol4_Prue)
Matrix_pol7_Prue_NoN = sklearn.metrics.confusion_matrix(Prue_class,predict_pol7_Prue)
Matrix_rbf_Prue_NoN = sklearn.metrics.confusion_matrix(Prue_class,predict_rbf_Prue)

#Obtenemos la funcion de decision de los clasificadores
deci_lin_Prue = grid_lin.decision_function(S_Prue)
deci_pol4_Prue = grid_pol4.decision_function(S_Prue)
deci_pol7_Prue = grid_pol7.decision_function(S_Prue)
```

```
deci_rbf_Prue = grid_rbf.decision_function(S_Prue)

#Obtenemos los parametros de la curva ROC de los clasificadores
TFP_lin_Prue, TVP_lin_Prue, threshold_lin_Prue = sklearn.metrics.roc_curve(Prue_class,
    ↪ deci_lin_Prue)
TFP_pol4_Prue, TVP_pol4_Prue, threshold_pol4_Prue = sklearn.metrics.roc_curve(Prue_class,
    ↪ deci_pol4_Prue)
TFP_pol7_Prue, TVP_pol7_Prue, threshold_pol7_Prue = sklearn.metrics.roc_curve(Prue_class,
    ↪ deci_pol7_Prue)
TFP_rbf_Prue, TVP_rbf_Prue, threshold_rbf_Prue = sklearn.metrics.roc_curve(Prue_class,
    ↪ deci_rbf_Prue)

#Area bajo la curva ROC de cada clasificador
Area_lin_Prue = sklearn.metrics.auc(TFP_lin_Prue,TVP_lin_Prue)
Area_pol4_Prue = sklearn.metrics.auc(TFP_pol4_Prue,TVP_pol4_Prue)
Area_pol7_Prue = sklearn.metrics.auc(TFP_pol7_Prue,TVP_pol7_Prue)
Area_rbf_Prue = sklearn.metrics.auc(TFP_rbf_Prue,TVP_rbf_Prue)

#Obtenemos los parametros para la curva precision-recall de los clasificadores
R_lin_Prue,P_lin_Prue,threshold_lin_Prue_PR = sklearn.metrics.precision_recall_curve(
    ↪ Prue_class,deci_lin_Prue)
R_pol4_Prue,P_pol4_Prue,threshold_pol4_Prue_PR = sklearn.metrics.precision_recall_curve(
    ↪ Prue_class,deci_pol4_Prue)
R_pol7_Prue,P_pol7_Prue,threshold_pol7_Prue_PR = sklearn.metrics.precision_recall_curve(
    ↪ Prue_class,deci_pol7_Prue)
R_rbf_Prue,P_rbf_Prue,threshold_rbf_Prue_PR = sklearn.metrics.precision_recall_curve(
    ↪ Prue_class,deci_rbf_Prue)

#Obtenemos el average precision score de los clasificadores
prom_lin_Prue = sklearn.metrics.average_precision_score(Prue_class,deci_lin_Prue)
prom_pol4_Prue = sklearn.metrics.average_precision_score(Prue_class,deci_pol4_Prue)
prom_pol7_Prue = sklearn.metrics.average_precision_score(Prue_class,deci_pol7_Prue)
prom_rbf_Prue = sklearn.metrics.average_precision_score(Prue_class,deci_rbf_Prue)
```

El código que muestra las métricas se ejemplifica en el código 13, donde se ejemplifican los comandos utilizados para obtener las métricas deseadas para el SVM lineal, los casos restantes son análogos.

Código 13: Implementación para obtener las métricas del SVM lineal sobre el conjunto de prueba..

```
print('')
print('Metricas del SVM lineal sobre el conjunto de Prueba')
print('')
#Matrices de confusion
print('La matriz de confusion normalizada es:')
print(Matrix_lin_Prue_N)
print('La matriz de confusion sin normalizar es:')
print(Matrix_lin_Prue_NoN)

#Area bajo la curva ROC
print('El area bajo la curva ROC es de: ',str(Area_lin_Prue))

```

```
#Curva ROC
plt.figure(1)
plt.title('Curva ROC del SVM lineal sobre el conjunto de prueba')
plt.xlabel('Tasa falsos positivos')
plt.ylabel('Tasa verdaderos positivos')
plt.grid()
plt.plot(TFP_lin_Prue,TVP_lin_Prue)

#Obtenemos el average precision score
print('El average precision score es: ',str(prom_lin_Prue))

#Graficamos la curva precision-recall
plt.figure(2)
plt.title('Curva precision-recall del SVM lineal sobre el conjunto de prueba')
plt.xlabel('Recall')
plt.ylabel('Precision')
plt.grid()
plt.plot(R_lin_Prue,P_lin_Prue)
```

2.12. Resultados

Las métricas obtenidas para los distintos clasificadores aplicados sobre el conjunto de prueba se muestran a continuación:

- **SVM lineal:** Las métricas obtenidas para el clasificador lineal corresponden a las de las figuras 13, 14 y 15.

```
Metricas del SVM lineal sobre el conjunto de Prueba

La matriz de confusion normalizada es:
[[0.83       0.17      ]
 [0.31166667 0.68833333]]
La matriz de confusion sin normalizar es:
[[498 102]
 [187 413]]
El area bajo la curva ROC es de:  0.8313222222222223
El average precision score es:  0.836780302581633
```

Figura 13: Matrices de confusión del SVM lineal sobre el conjunto de prueba.

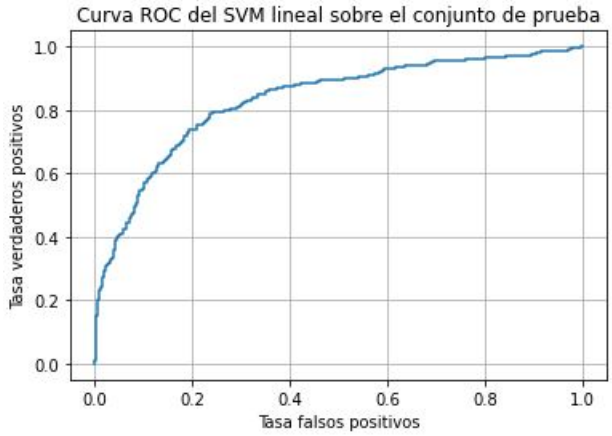

Figura 14: Curva ROC del SVM lineal sobre el conjunto del prueba.

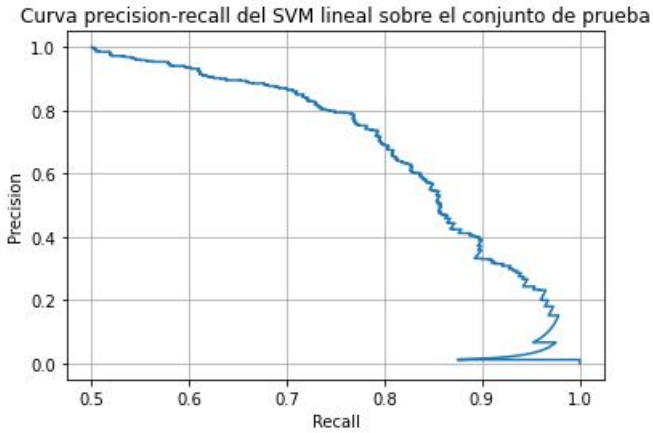

Figura 15: Curva precision-recall del SVM lineal sobre el conjunto del prueba.

- **SVM polinomial grado 4:** Las métricas obtenidas para el clasificador polinomial de grado 4 corresponden a las de las figuras 16, 17 y 18.

```
Metricas del SVM polinomial de grado 4 sobre el conjunto de Prueba

La matriz de confusion normalizada es:
[[0.93666667 0.06333333]
 [0.37666667 0.62333333]]
La matriz de confusion sin normalizar es:
[[562  38]
 [226 374]]
El area bajo la curva ROC es de:  0.8922249999999999
El average precision score es:  0.8952073757489647
```

Figura 16: Matrices de confusión del SVM polinomial grado 4 sobre el conjunto del prueba.

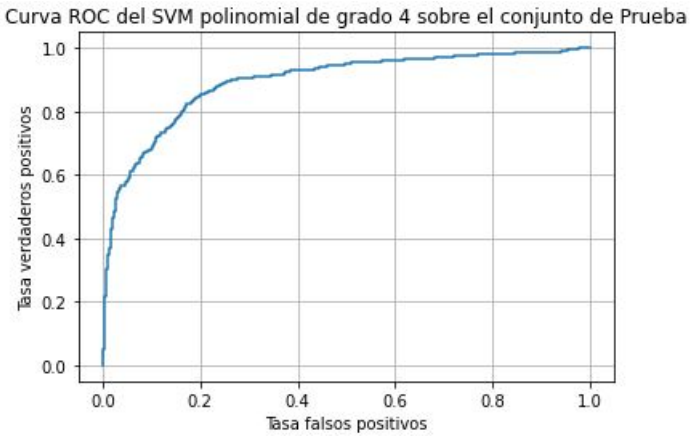

Figura 17: Curva ROC del SVM polinomial grado 4 sobre el conjunto de prueba.

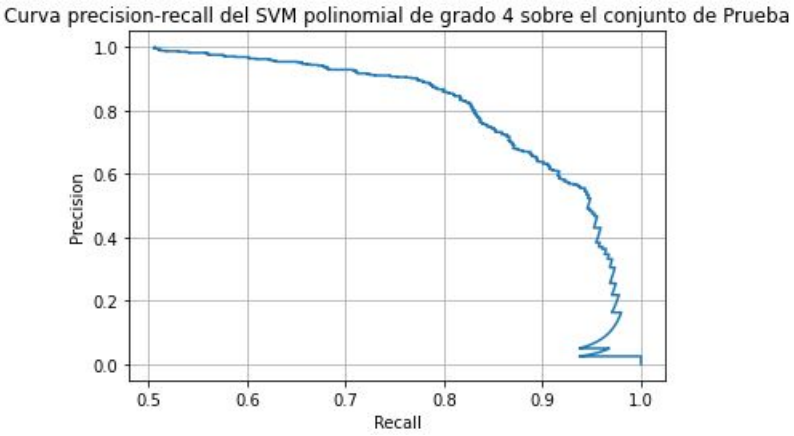

Figura 18: Curva precision-recall del SVM polinomial grado 4 sobre el conjunto de prueba.

- **SVM polinomial grado 7:** Las métricas obtenidas para el clasificador polinomial de grado 7 corresponden a las de las figuras 19, 20 y 21.

```
Metricas del SVM polinomial de grado 7 sobre el conjunto de Prueba

La matriz de confusion normalizada es:
[[0.95333333 0.04666667]
 [0.565      0.435     ]]
La matriz de confusion sin normalizar es:
[[572  28]
 [339 261]]
El area bajo la curva ROC es de:  0.8400944444444445
El average precision score es:  0.8487988434354651
```

Figura 19: Matrices de confusión del SVM polinomial grado 7 sobre el conjunto del prueba.

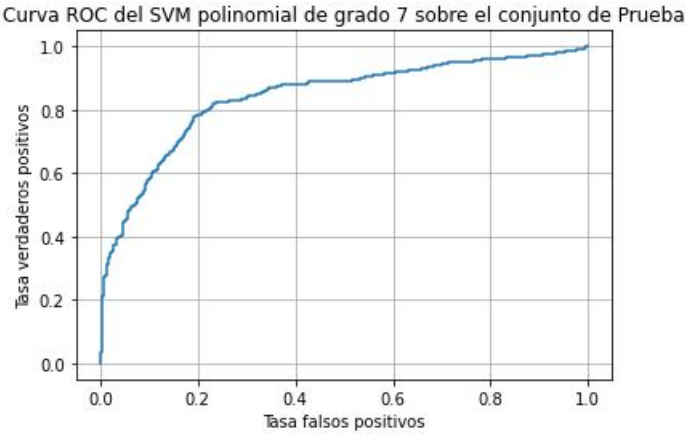

Figura 20: Curva ROC del SVM polinomial grado 7 sobre el conjunto de prueba.

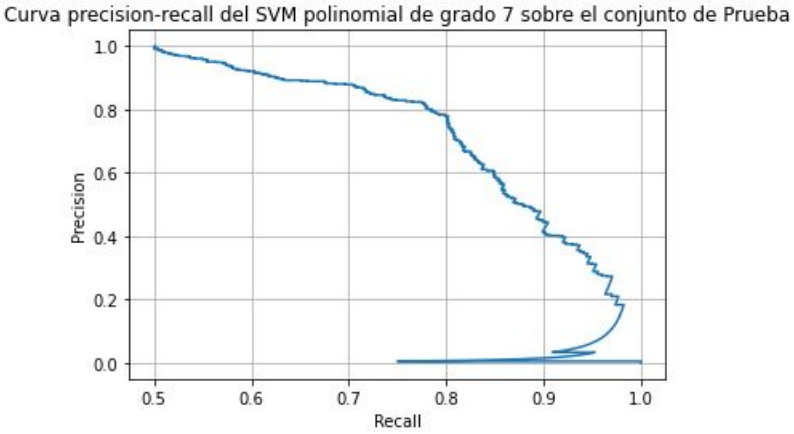

Figura 21: Curva precision-recall del SVM polinomial grado 7 sobre el conjunto de prueba.

- **SVM rbf:** Las métricas obtenidas para el clasificador rbf corresponden a las de las figuras 22, 23 y 24.

```
Metricas del SVM rbf sobre el conjunto de Prueba

La matriz de confusion normalizada es:
[[0.90666667 0.09333333]
 [0.20833333 0.79166667]]
La matriz de confusion sin normalizar es:
[[544  56]
 [125 475]]
El area bajo la curva ROC es de:  0.9239583333333332
El average precision score es:  0.9329882178565811
```

Figura 22: Matrices de confusión del SVM rbf sobre el conjunto del prueba.

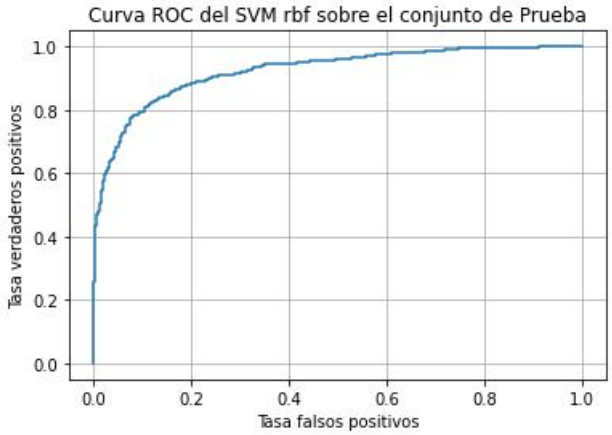

Figura 23: Curva ROC del SVM rbf sobre el conjunto de prueba.

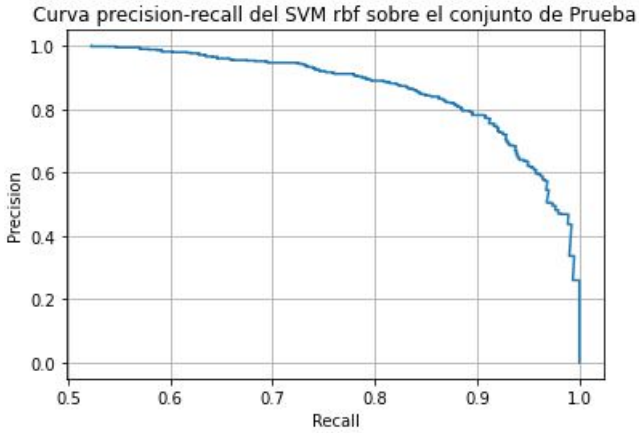

Figura 24: Curva precision-recall del SVM rbf sobre el conjunto de prueba.

.

3. Conclusiones

A modo de conclusión se puede afirmar el cumplimiento del objetivo impuesto en la tarea número 2 del curso EL4106 Inteligencia Computacional, donde se logró implementar varios *Support Vector Machine* de kernels diferentes.

Dentro de los aprendizajes obtenidos se destaca el uso de la biblioteca *sklearn* que provee todas las funciones necesarias para implementar SVMs con diferentes tipos de kernel, así como también provee lo necesario para obtener la grilla de hiper parámetros más adecuada para cada SVM así como también permite obtener las diferentes métricas usadas a lo largo de esta tarea. También se destaca el hecho de que a la hora de entregar posibles hiper parámetros a la grilla los gamma no podían ser muy grandes, pues para valores superiores a 0.1 el programa se demora demasiado en ejecutar, por lo cual si se desea una respuesta rápida no se sugiere el uso de gammas muy grandes.

Dentro de los resultados obtenidos a la hora de evaluar el conjunto de prueba se logra observar que el SVM con kernel rbf es el que realiza una mejor clasificación, esto se respalda con los valores de área bajo la curva ROC y average precision score obtenidos, valores que eran mayores a sus equivalentes pero en los otros clasificadores también se puede dar fe de esto al observar la forma que tienen las curvas ROC y de precision-recall, que se acercan bastante a lo que sería una curva ROC/precision-recall de un clasificador perfecto, aunque claramente esta perfección no se alcanza.

Una forma de optimizar el código implementado sería hacer una búsqueda iterativa de hiper-parámetros, de forma tal que la grilla solo deba discernir entre los mejores hiper parámetros posibles, esto podría hacer que los SVM mejoren incluso más. Por otro lado si se quisiese optimizar el tiempo de ejecución podrían darse valores pequeños de posibles hiper parámetros a la grilla y en los SVM polinomiales elegir polinomios de grados pequeños.