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

Tarea 3

Selección de características

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: 16 de Mayo de 2021
Santiago, Chile

Índice de Contenidos

Índice de Figuras

Índice de Códigos

1. Introducción

En este informe se detallará la tarea número 3 del curso EL4106 Inteligencia Computacional, tarea que consiste en realizar selección de características utilizando un clasificador SVM y un Random Forest, esto se lograría aplicando 3 formas de "elegir" las características, una de ellas sería aplicando la estrategia de *Wrapper*, otra forma sería mediante el uso de una estrategia de filtro, donde se verá el caso de elegir 4 y 2 características. Luego de obtener las características reducidas se reentrenarán los clasificadores sobre el conjunto de características reducidas para finalmente comparar matrices de confusión y Accuracy en los diferentes casos. Para la realización de esta tarea se utilizará la herramienta *Google Colaboratory*, herramienta que permite la simulación de notebooks de Python, además, se recurrirá a las librerías *Pandas*, *Numpy*, *Seaborn*, *Sklearn* y *Scipy.io*.

Las secciones que tendrá el presente informe se enuncian a continuación:

- **Pruebas:** En esta sección se detallarán las diferentes actividades a ser programadas.
  - **Actividad A:** En esta actividad se deberá realizar un explicación pequeña de cuáles son las características que tiene el dataset *diabetes.arff*.
  - **Actividad B:** En la segunda actividad se deberá dar lectura al archivo *diabetes.arff*, una vez realizado esto se deberán cambiar las etiquetas de la columna *"class"* por un 0 o un 1 según corresponda para finalmente proceder a dividir el dataset en 3 conjuntos, el de entrenamiento, que tendrá el 60 % de las muestras del dataset anterior, el de validación, que tendrá un 20 % de las muestras, y el de prueba, que tendrá el 20 % de las muestras restantes, para realizar esto se hará uso de la función *train_test_split()*.
  - **Actividad C:** En esta actividad se entrenará un *StandardScaler* con el conjunto de entrenamiento, una vez se complete el entrenamiento se deben procesar las características de los distintos conjuntos. Lo anterior se realiza con la finalidad de normalizar las características de los conjuntos.
  - **Actividad D:** En la cuarta actividad se deberá realizar una clasificación inicial usando un SVM lineal, para ello se usará una grilla sobre el hiperparámetro "C", en esta grilla se deberá usar el conjunto de Entrenamiento y Validación concatenados, indicándole a la grilla que elementos corresponden al test de entrenamiento y cuales pertenecen al de validación. Finalmente se deberá mostrar el tiempo de entrenamiento del clasificador ante esta situación.
  - **Actividad E:** En la quinta actividad se deberá obtener la matriz de confusión normalizada del clasificador SVM obtenido en la actividad anterior, para luego mostrar dicha matriz utilizando la función *heatmap()* de *Seaborn*, además se deberá calcular el accuracy del clasificador. Para realizar las tareas anteriores se hará uso del conjunto de validación una vez ya fueron procesadas sus características con el *StandardScaler*.
  - **Actividad F:** En esta actividad se realizará selección de características mediante la estrategia de *Wrapper*, para ello se utilizará la función *SelectFromModel()* de la librería *sklearn*, en dicha función se utilizará el mejor clasificador obtenido en los puntos anteriores como parámetro. Finalmente se deberán indicar cuales fueron las características seleccionadas.
  - **Actividad G:** Con las características seleccionadas en la actividad anterior se deberán obtener los conjuntos de características reducidos y con ello entrenar un nuevo clasificador, indicando el tiempo de entrenamiento.

- **Actividad H:** En esta actividad se deberá obtener la matriz de confusión y el accuracy del clasificador con las características reducidas, para ello se hará uso del conjunto de validación.
- **Actividad I:** En la novena actividad se deberán repetir las actividades G y H utilizando clasificador SVM lineal pero ahora utilizando una estrategia de tipo filtro eligiendo 4 características, esto se logrará mediante la función *SelectKBest()* de la librería *sklearn*.
- **Actividad J:** Esta actividad es idéntica a la anterior pero seleccionando 2 características en el filtro.
- **Actividad K:** En la onceava actividad se deberá repetir la actividad D y E pero ahora utilizando un clasificador Random Forest de profundidad 3 y la grilla será sobre el hiperparámetro "n_estimators".
- **Actividad L:** En esta actividad se tendrán que repetir las actividades G y H pero usando un clasificador Random Forest y la estrategia de *Wrapper*.
- **Actividad M:** En esta actividad se deberán repetir las actividades G y H con un clasificador Random Forest y utilizando una estrategia de tipo filtro, eligiendo 4 características.
- **Actividad N:** En la penúltima actividad de la sección de pruebas se deberá realizar lo mismo que en la actividad anterior pero realizando un filtrado para elegir 2 características.
- **Actividad O:** Finalmente se deberán evaluar los distintos clasificadores obtenidos utilizando las características reducidas sobre el conjunto de prueba, indicando las matrices de confusión y los accuracy de cada clasificador.

- **Análisis:** En esta sección se hará el análisis de los resultados obtenidos en la sección anterior.
  - **Actividad A:** En la primera actividad de la sección de análisis se tendrá que indicar que clasificador entrega, en general, los mejores resultados.
  - **Actividad B:** En esta actividad se deberá indicar el efecto de reducir características sobre el accuracy obtenido. Para ello se considerará el número de características seleccionadas versus el accuracy obtenido.
  - **Actividad C:** Finalmente, en base a las dos actividades anteriores, se analizará la utilidad de reducir las características en el conjunto de datos utilizado.

2. Desarrollo

En las siguientes subsecciones se exhibirá el código implementado para las diferentes actividades solicitadas por el equipo docente, indicando que es lo que está haciendo el código así como también indicando los resultados obtenidos. Cabe destacar que en esta sección solo se explicará el código implementado, mas no se dirá que es lo que se debe hacer en cada actividad, esto ya fue realizado en la introducción.

2.1. Pruebas

2.1.1. Actividad A

Dentro del dataset se tenían las características *"preg"*, *"plas"*, *"pres"*, *"skin"*, *"insu"*, *"mass"*, *"pedi"*, *"age"* y *"class"*. A continuación se enlista que indica cada característica:

- **Característica "preg":** Esta característica indica cuantas veces ha estado embarazada la persona.
- **Característica "plas":** Indica la concentración de la glucosa en plasma.
- **Característica "pres":** Esta característica indica la presión sanguínea, está medida en milímetros de mercurio.
- **Característica "skin":** Indica el espesor del trícep y está medida en milímetros.
- **Característica "insu":** En esta característica se indica la concentración de insulina, está medida en micro unidades por mililitro.
- **Característica "mass":** Esta característica indica el índice de masa corporal.
- **Característica "pedi":** En esta característica se indica la función de diabetes de una persona basado en su árbol genealógico.
- **Característica "age":** Esta característica indica la edad de la persona.
- **Característica "class":** Finalmente se indica si la persona es o no diabetica.

2.1.2. Actividad B

Para subir al archivo *diabetes.arff* al entorno de trabajo se empleó el código 1.

Código 1: Código utilizado para subir los archivos.

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

Una vez subido el archivo se le debía dar lectura, para ello se utilizó la función *loadarff()*, una vez realizado esto existía el problema de que pese a que el archivo ya era leído por el entorno de ejecución, este no estaba en un formato adecuado como para trabajarlo utilizando *Pandas*, para solucionar esto se utilizó la función *DataFrame()*, función con la que se transformó el archivo a un dataframe. Finalmente se obtuvo el vector que indica los nombres de las columnas del dataframe y

por tanto indica las características. El código encargado de realizar lo anterior se observa en el código 2.

Código 2: Lectura del archivo.

```
data = arff.loadarff('diabetes.arff')
df = pd.DataFrame(data[0])
nombres = df.columns
```

Luego para realizar el cambio de la columna *"class"* se hizo uso de la función *loc()* de Pandas, con ello se hizo que a las muestras con clase *"b'tested_negative'"* se les identificara con un 0 y a las muestras con clase *"b'tested_positive'"* se les identificará con un 1. Una vez realizado lo anterior se utilizó la función *train_test_split()* para realizar la separación de conjuntos, esta función recibe como parámetros un dataset y el porcentaje de muestras que se desea que el conjunto de test tenga, en este caso no se necesita dividir al dataframe en conjunto de entrenamiento y test, sino que también se debe obtener el conjunto de validación, por esto es que se utiliza 2 veces la función anterior, la primera vez que se utiliza se obtiene el conjunto de entrenamiento, que tiene un 60 % de las muestras, y otro conjunto que contiene el 40 % restante, luego sobre el conjunto que contiene el 40 % de las muestras se vuelve a utilizar *train_test_split()* para obtener al conjunto de validación y de prueba. Finalmente solo se reiniciaron los índices de los conjuntos, para que así las muestras seleccionadas no conserven los índices que tenían antes de producirse la separación de conjuntos. El código encargado de hacer lo anterior corresponde al código 3.

Código 3: Cambios en la clase y definición de conjuntos.

```
#En el dataframe df cambiamos la clase como:
#Un 1 si la persona dio positivo o un 0 sino
df.loc[df['class'] == b'tested_negative', 'class'] = 0
df.loc[df['class'] == b'tested_positive', 'class'] = 1

#Definimos los conjuntos
Entrena, Otro = sklearn.model_selection.train_test_split(df, test_size=0.4)
Vali, Test = sklearn.model_selection.train_test_split(Otro, test_size=0.5)

Entrena.reset_index(drop=True)
Vali.reset_index(drop=True)
Test.reset_index(drop=True)
print('')
```

2.1.3. Actividad C

El código 4 fue utilizado para implementar el *StandardScaler* y normalizar las características de los distintos conjuntos.

Código 4: Implementación del StandardScaler y normalización de características.

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

#Entrenamos el scaler en el conjunto de entrenamient
scaler.fit(Entrena.iloc[:,:8])
```

```
6
7  #Aplicamos el scaler a los conjuntos
8  S_Entrena = scaler.transform(Entrena.iloc[:,:8])
9  S_Vali = scaler.transform(Vali.iloc[:,:8])
10 S_Test = scaler.transform(Test.iloc[:,:8])
11
12 #Obtenemos la columna class de cada conjunto
13 Entrena_class = np.array(Entrena['class'], dtype='int64')
14 Vali_class = np.array(Vali['class'], dtype='int64')
15 Test_class = np.array(Test['class'], dtype='int64')
```

La metodología tras este código es que primero se crea el *StandardScaler*, para ello se utiliza la función que tiene el mismo nombre y que es de la librería *sklearn.preprocessing*. Una vez definido el *StandardScaler* se entrena con el conjunto de entrenamiento, para ello se utiliza la función *fit()* e *iloc()*, la primera de ellas se encarga de hacer el entrenamiento como tal y la segunda sirve para seleccionar que parte del conjunto se quiere utilizar, en este caso solo quieren utilizarse las características, por este motivo es que la columna de la clase fue omitida. Con el entrenamiento realizado se normalizaron las características de los distintos conjuntos utilizando la función *transform()*, donde nuevamente solo se consideraron las características sin la clase. Finalmente se obtuvo la columna *"class"* de cada conjunto y se transformó a un array mediante la función *array()* de *Numpy*, también se configuró que los valores de la clase fuesen enteros.

2.1.4. Actividad D

El código utilizado para entrenar un SVM lineal se observa en el código 5.

Código 5: Entrenamiento de un SVM lineal.

```
1  #Antes de crear el GridSearch concatenamos conjunto de entrenamiento y de validacion
2  Entrena_Vali = np.concatenate([S_Entrena, S_Vali])
3  Entrena_Vali_class = np.concatenate([Entrena_class, Vali_class])
4
5  #Vemos que indices del conjunto Entrena_Vali pertenecen al conjunto de entrenamiento
6  index = np.zeros(Entrena.shape[0]+Vali.shape[0])
7  for i in range(Entrena.shape[0]):
8    index[i] = -1
9
10 #Definimos el predifinedSplit
11 ps = sklearn.model_selection.PredefinedSplit(index)
12
13 #Creamos el SVM lineal
14 SVM_lin = sklearn.svm.SVC(kernel='linear', probability=False)
15 #Creamos el gridSearch
16 grid_lin = sklearn.model_selection.GridSearchCV(estimator=SVM_lin, param_grid={'C' : [0.0001,
       ↪ 0.001, 0.1]}, cv=ps)
17
18 time0 = time.time() #<-- Inicio del entrenamiento
19 #Entrenamos el grid con el conjunto de entrenamiento y validacion concatenados
20 grid_lin.fit(Entrena_Vali, Entrena_Vali_class)
21 time1 = time.time() #<- Final del entrenamiento
22
```

```
#Vemos los hiperparametros del grid
print('Los mejores hiperparametros son: '+str(grid_lin.best_params_))
#Vemos el score
print('El score es: '+str(grid_lin.best_score_))
#Vemos el tiempo de entrenamiento
tiempo = time1-time0
print('El tiempo de entrenamiento es: '+str(tiempo))

#El mejor estimador es:
estim_lin = grid_lin.best_estimator_
```

El código anterior opera como sigue: Primero definimos un conjunto que es la concatenación de los conjuntos de entrenamiento y validación obtenidos anteriormente, luego se hizo lo mismo con las clases de estos. Una vez definidos los conjuntos se procedió a generar un vector que servirá para indicar que elementos del conjunto de entrenamiento y validación concatenados pertenecen al conjunto de entrenamiento, esto se hace porque en el *GridSearch*, que se definirá más adelante, se le hará entrega del conjunto concatenado y necesita saber que elementos pertenecen a cada conjunto, los elementos que pertenezcan al conjunto de entrenamiento deben ser marcados por un -1 y los que no con un 0. Una vez definido este vector se le entrega a la función *PredefinedSplit()* de la librería *sklearn.model_selection*, esta función evaluada en el vector definido anteriormente será el parámetro que utilizará el *GridSearch* para identificar a los elementos que pertenecen al conjunto de entrenamiento. Con lo anterior definido se tiene casi todo lo necesario para crear el *GridSearch*, sin embargo hace falta el estimador que se utilizará, dicho estimador es un SVM con kernel lineal, definido el estimador se crea el *GridSearch* utilizando la función *GridSearchCV()* de la librería *sklearn.model_selection*, función que recibe como parámetros el estimador sobre el cual se hará la búsqueda de hiperparámetros, un diccionario que tiene a los hiperparámetros, en este caso solo tendrá al parámetro "C", y finalmente el resultado obtenido de utilizar la función *PredefinedSplit()*. Finalmente solo resta entrenar la grilla obtenida, para ello se utiliza la función *fit()* y el conjunto de entrenamiento y validación concatenados. El resto del código entrega parámetros de relevancia como los hiperparámetros que seleccionó la grilla, el score del clasificador y el tiempo de entrenamiento, luego se obtiene el mejor estimador que se desprende de la grilla. Los hiperparámetros, el score y el tiempo de entrenamiento se observa en el código 6.

Código 6: Resultados de la actividad D.

```
Los mejores hiperparametros son: {'C': 0.1}
El score es: 0.7272727272727273
El tiempo de entrenamiento es: 0.02456045150756836
```

2.1.5. Actividad E

El código 7 muestra la forma utilizada para obtener las diferentes métricas solicitadas.

Código 7: Métricas obtenidas del mejor SVM lineal.

```
#Obtenemos la prediccion del clasificador sobre el conjunto de validacion
predi_vali_lin = estim_lin.predict(S_Vali)

#Obtenemos la matriz de confusión
Confu = sklearn.metrics.confusion_matrix(Vali_class, predi_vali_lin, normalize='true')
```

```
6
7  #Mostramos la matriz de confusion
8  matrix = sns.heatmap(Confu, annot=True)
9
10 #Obtenemos el accuracy del SVM lineal
11 accuracy_lin = sklearn.metrics.accuracy_score(Vali_class, predi_vali_lin)
12
13 print('El accuracy es: '+str(accuracy_lin))
```

En el código anterior se obtuvo la matriz de confusión mediante la utilización de la función *confusion_matrix()* de la librería *sklearn.metrics*, función que recibe como parámetros la correcta clasificación del conjunto de validación, la predicción que realiza el clasificador sobre dicho conjunto y un parámetro que indique si se desea obtener la matriz normalizada o no. Cabe destacar que para obtener la predicción del estimador se hizo uso de la función *predict()*. Luego utilizando *heatmap()* de *Seaborn* se logra mostrar en pantalla la matriz de confusión resultante. Finalmente el accuracy del clasificador sobre el conjunto de validación se obtiene mediante la función *accuracy_score()*, función que recibe los mismos parámetros que *confusion_matrix()*, pero sin el parámetro que indique la normalización. La matriz de confusión se puede observar en la figura 1 y el accuracy obtenido se observa en el código 8.

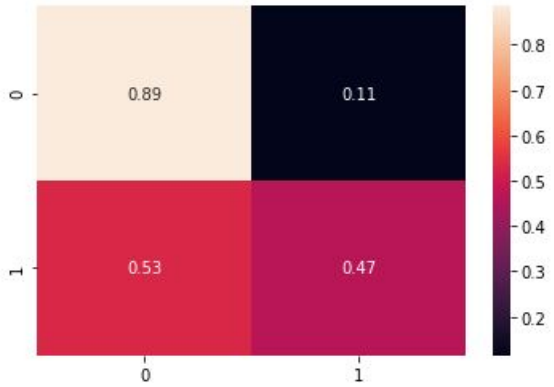

Figura 1: Matriz de confusión del SVM lineal.

Código 8: Accuracy del SVM lineal.

```
El accuracy es: 0.7272727272727273
```

2.1.6. Actividad F

Para realizar la selección de características mediante el método de *Wrapper* se implementó el código 9.

Código 9: Obtención de características mediante el método de Wrapper.

```
#Hacemos el Model
model_lin = sklearn.feature_selection.SelectFromModel(estim_lin)
#Entrenamos el modelo con el conjunto de entrenamiento
model_lin.fit(S_Entrena, Entrena_class)
```

```
5
6  #Obtenemos las caracteristicas seleccionadas
7  caracte_lin = model_lin.get_support(indices=False) #<- Array que indica las caracteristicas
       ↪ consideradas
8
9  #Recorremos la lista de caracteristicas seleccionadas y las añadimos a una lista donde se consideran
10 #solo a las caracteristicas seleccionadas
11 select_carac_lin = []
12 for i in range(len(caracte_lin)):
13   if caracte_lin[i] == True:
14     select_carac_lin.append(nombres[i])
15 print('Las caracteristicas seleccionadas son: '+str(select_carac_lin))
```

En el código anterior se utilizó el método de *Wrapper* para seleccionar características, la función que permite realizar esto es la función *SelectFromModel()* de la librería *sklearn.feature_selection*, esta función recibe como parámetro a un estimador, en este caso se utilizó el mejor estimador obtenido del *GridSearch* definido anteriormente, luego simplemente se entrena utilizando el conjunto de entrenamiento y la correcta clasificación de dicho conjunto. Con el método entrenado obtenemos las características seleccionadas mediante el uso de la función *get_support()*, esta función entrega un vector con valores Booleanos, este vector tendrá el mismo largo que la cantidad de características que tenga el conjunto, en este caso tendrá un largo de 8, luego si se tuviese la celda i con un True significa que la celda i del vector "nombres" (característica en la posición i de dicho vector), vector definido en la actividad B, ha sido seleccionada y por tanto es una característica relevante. La última parte del código es simplemente una iteración sobre los elementos del array "nombres" y el array obtenido al utilizar *get_support()*, realizando esta iteración se obtienen las características seleccionadas, dichas características se pueden apreciar en el código 10.

Código 10: Características seleccionadas.

```
1 Las caracteristicas seleccionadas son: ['plas', 'mass']
```

2.1.7. Actividad G

En la actividad anterior se obtuvieron las características seleccionadas al utilizar el método de *Wrapper* y un SVM lineal, en la actividad actual se reentrena un SVM lineal utilizando las características seleccionadas anteriormente, esto se puede observar en el código 11.

Código 11: Reentrenamiento del SVM lineal con las características reducidas.

```
1 #Obtenemos los distintos conjuntos con las caracteristicas reducidas
2 Entrena_Vali_redu = model_lin.transform(Entrena_Vali) #<--- Entrenamiento y validacion
      ↪ concatenados
3 Vali_redu = model_lin.transform(S_Vali) #<----- Conjunto de validacion
4 Test_redu = model_lin.transform(S_Test) #<----- Conjunto de prueba
5
6 #Reentrenamos con el conjunto reducido
7 SVM_lin2 = sklearn.svm.SVC(kernel='linear', probability=False)
8 grid_lin2 = sklearn.model_selection.GridSearchCV(SVM_lin2, param_grid={'C' : [0.0001, 0.001,
      ↪ 0.1]}, cv=ps)
9
```

```
time2 = time.time() #<------- Inicio del entrenamiento
grid_lin2.fit(Entrena_Vali_redu, Entrena_Vali_class)
time3 = time.time() #<-------- Final del entrenamiento

#Vemos los hiperparametros del grid
print('Los mejores hiperparametros son: '+str(grid_lin2.best_params_))
#Vemos el score
print('El score es: '+str(grid_lin2.best_score_))
#Vemos el tiempo de entrenamiento
tiempo1 = time3-time2
print('El tiempo de entrenamiento es de: '+str(tiempo1))

#Obtenemos el mejor estimador
estim_lin2 = grid_lin2.best_estimator_
```

Lo más destacable del código anterior es lo que se observa en el inicio de este, donde se realizó la reducción de los diferentes conjuntos, conjuntos que solo considerarán a las características que se seleccionaron anteriormente, para realizar esto se utilizó la función *transform()*. Luego simplemente se repitió lo mismo que en la actividad D pero con los conjuntos de características reducidos, primero se configuró un *GridSearch* utilizando como parámetros un SVM lineal, los mismos hiperparámetros utilizados con anterioridad y el resultado obtenido de utilizar *PredifinedSplit()* en la actividad D. Posteriormente simplemente se obtuvieron los mejores hiperparámetros, el score del clasificador y el tiempo de entrenamiento del mismo. Lo anterior se puede observar en el código 12.

Código 12: Resultados de la actividad G.

```
Los mejores hiperparametros son: {'C': 0.1}
El score es: 0.7142857142857143
El tiempo de entrenamiento es de: 0.015648841857910156
```

2.1.8. Actividad H

El código utilizado para obtener la matriz de confusión y el accuracy del SVM lineal al considerar el conjunto de características reducido se aprecia en el código 13.

Código 13: Métricas del SVM lineal con características reducidas.

```
#Obtenemos la prediccion del clasificador sobre el conjunto de validacion
predi_vali_lin2 = estim_lin2.predict(Vali_redu)

#Obtenemos la matriz de confusión
Confu2 = sklearn.metrics.confusion_matrix(Vali_class, predi_vali_lin2, normalize='true')

#Mostramos la matriz de confusion
matrix2 = sns.heatmap(Confu2, annot=True)

#Obtenemos el accuracy del SVM lineal
accuracy_lin2 = sklearn.metrics.accuracy_score(Vali_class, predi_vali_lin2)

print('El accuracy al usar un conjunto con las caracteristicas reducidas es: '+str(accuracy_lin2))
```

La matriz de confusión y el accuracy se obtuvieron de manera análoga a los observado en la actividad E, simplemente se cambiaron las variables antiguas por las correspondientes a la situación. Ambas métricas obtenidas se pueden observar en la figura 2 y el código 14.

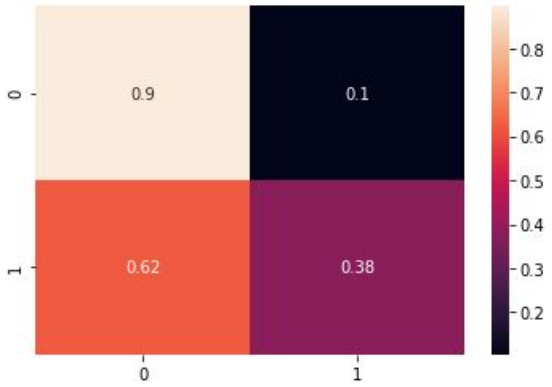

Figura 2: Matriz de confusión del SVM lineal con características reducidas.

Código 14: Accuracy del SVM lineal con características reducidas.

```
El accuracy al usar un conjunto con las caracteristicas reducidas es: 0.7012987012987013
```

2.1.9. Actividad I

El código implementado para realizar la selección de 4 características mediante el método de filtro corresponde al código 15.

Código 15: Selección de 4 características mediante el método de filtro en un SVM lineal.

```
#Obtenemos el conjunto de caracteristicas (4)
SelectKBest4 = sklearn.feature_selection.SelectKBest(k=4).fit(S_Entrena, Entrena_class)
caract_KBest4 = SelectKBest4.get_support(indices=False) #<--- Array que indica las caracteristicas consideradas
#Recorremos la lista de caracteristicas seleccionadas y las añadimos a una lista donde se consideran
#solo a las caracteristicas seleccionadas
select_carac_KBest4 = []
for i in range(len(caract_KBest4)):
  if caract_KBest4[i] == True:
    select_carac_KBest4.append(nombres[i])
print('Las caracteristicas seleccionadas son: '+str(select_carac_KBest4))

#Obtenemos los distintos conjuntos con las caracteristicas reducidas
Entrena_Vali_redu_KBest4 = SelectKBest4.transform(Entrena_Vali) #<--- Entrenamiento y validacion concatenados
Vali_redu_KBest4 = SelectKBest4.transform(S_Vali) #<----- Conjunto de validacion
Test_redu_KBest4 = SelectKBest4.transform(S_Test) #<----- Conjunto de prueba

#Reentrenamos con el conjunto reducido
SVM_KBest4 = sklearn.svm.SVC(kernel='linear', probability=False)
```

```
grid_KBest4 = sklearn.model_selection.GridSearchCV(SVM_KBest4, param_grid={'C' : [0.0001, 0.001, 0.1]}, cv=ps)

time4 = time.time() #<------- Inicio del entrenamiento
grid_KBest4.fit(Entrena_Vali_redu_KBest4, Entrena_Vali_class)
time5 = time.time() #<-------- Final del entrenamiento

#Vemos los hiperparametros del grid
print('Los mejores hiperparametros son: '+str(grid_KBest4.best_params_))
#Vemos el score
print('El score es: '+str(grid_KBest4.best_score_))
#Vemos el tiempo de entrenamiento
tiempo2 = time5-time4
print('El tiempo de entrenamiento es de: '+str(tiempo2))

#Obtenemos el mejor estimador
estim_KBest4 = grid_KBest4.best_estimator_

#Obtenemos la prediccion del clasificador sobre el conjunto de validacion
predi_vali_KBest4 = estim_KBest4.predict(Vali_redu_KBest4)

#Obtenemos la matriz de confusión
Confu3 = sklearn.metrics.confusion_matrix(Vali_class, predi_vali_KBest4, normalize='true')

#Mostramos la matriz de confusion
matrix3 = sns.heatmap(Confu3, annot=True)

#Obtenemos el accuracy del SVM lineal
accuracy_KBest4 = sklearn.metrics.accuracy_score(Vali_class, predi_vali_KBest4)

print('El accuracy al usar un conjunto de caracteristicas reducidas (4 caracteristicas) es: '+str(accuracy_KBest4))
```

El código anterior es idéntico a los códigos implementados con anterioridad pero en este caso se implementó una selección de características utilizando un método de tipo filtro, para realizar esto se recurrió a la función *SelectKBest()* de la librería *sklearn.feature_selection*, esta función recibe como parámetro la cantidad de características que "sobrevivirán" después del filtro, en este caso se indicó que fuesen 4 características. Una vez implementado lo anterior se entrenó el filtro con el conjunto de entrenamiento y se utilizó la función *get_support()* para obtener un array que indique cuales características son las sobrevivientes después de pasar por el filtro, para luego iterar sobre dicho array con la finalidad de obtener los nombres de las características obtenidas. Luego, mediante la función *transform()* se obtuvieron los diferentes conjuntos al considerar solo a las características seleccionadas. Con lo anterior realizado simplemente se configuró un *GridSearch* y se entrenó con los conjuntos de características reducidos. Finalmente solo se obtuvieron las diferentes métricas del SVM lineal al considerar 4 características. Las características seleccionadas así como las métricas obtenidas se observan en el código 16, mientras que la matriz de confusión correspondiente se observa en la figura 3.

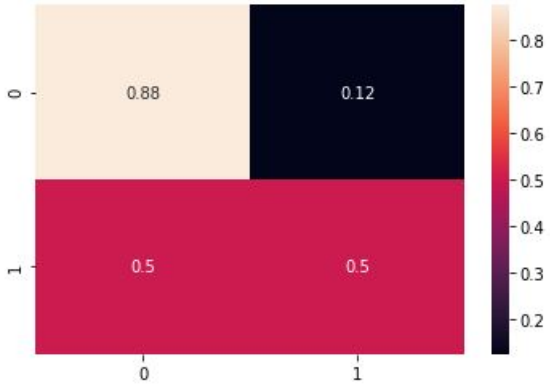

Figura 3: Matriz de confusión del SVM lineal con características reducidas a 4 (método de filtro).

Código 16: Características y métricas del SVM lineal al aplicar un método de tipo filtro.

```
Las caracteristicas seleccionadas son: ['preg', 'plas', 'mass', 'age']
Los mejores hiperparametros son: {'C': 0.1}
El score es: 0.7077922077922078
El tiempo de entrenamiento es de: 0.020557880401611328
El accuracy al usar un conjunto de caracteristicas reducidas (4 caracteristicas) es:
    ↪ 0.7337662337662337
```

2.1.10. Actividad J

El código implementado para obtener 2 características mediante el método de filtro se observa en el código 17.

Código 17: Accuracy del SVM lineal con características reducidas.

```
#Obtenemos el conjunto de caracteristicas (2)
SelectKBest2 = sklearn.feature_selection.SelectKBest(k=2).fit(S_Entrena, Entrena_class)
caract_KBest2 = SelectKBest2.get_support(indices=False) #<--- Array que indica las caracteristicas
    ↪ consideradas
#Recorremos la lista de caracteristicas seleccionadas y las añadimos a una lista donde se consideran
#solo a las caracteristicas seleccionadas
select_carac_KBest2 = []
for i in range(len(caract_KBest2)):
  if caract_KBest2[i] == True:
    select_carac_KBest2.append(nombres[i])
print('Las caracteristicas seleccionadas son: '+str(select_carac_KBest2))

#Obtenemos los distintos conjuntos con las caracteristicas reducidas
Entrena_Vali_redu_KBest2 = SelectKBest2.transform(Entrena_Vali) #<--- Entrenamiento y
    ↪ validacion concatenados
Vali_redu_KBest2 = SelectKBest2.transform(S_Vali) #<----- Conjunto de validacion
Test_redu_KBest2 = SelectKBest2.transform(S_Test) #<----- Conjunto de prueba

```

```
#Reentrenamos con el conjunto reducido
SVM_KBest2 = sklearn.svm.SVC(kernel='linear', probability=False)
grid_KBest2 = sklearn.model_selection.GridSearchCV(SVM_KBest2, param_grid={'C' : [0.0001,
    ↪ 0.001, 0.1]}, cv=ps)

time6 = time.time() #<------- Inicio del entrenamiento
grid_KBest2.fit(Entrena_Vali_redu_KBest2, Entrena_Vali_class)
time7 = time.time() #<-------- Final del entrenamiento

#Vemos los hiperparametros del grid
print('Los mejores hiperparametros son: '+str(grid_KBest2.best_params_))
#Vemos el score
print('El score es: '+str(grid_KBest2.best_score_))
#Vemos el tiempo de entrenamiento
tiempo3 = time7-time6
print('El tiempo de entrenamiento es de: '+str(tiempo3))

#Obtenemos el mejor estimador
estim_KBest2 = grid_KBest2.best_estimator_

#Obtenemos la prediccion del clasificador sobre el conjunto de validacion
predi_vali_KBest2 = estim_KBest2.predict(Vali_redu_KBest2)

#Obtenemos la matriz de confusión
Confu4 = sklearn.metrics.confusion_matrix(Vali_class, predi_vali_KBest2, normalize='true')

#Mostramos la matriz de confusion
matrix4 = sns.heatmap(Confu4, annot=True)

#Obtenemos el accuracy del SVM lineal
accuracy_KBest2 = sklearn.metrics.accuracy_score(Vali_class, predi_vali_KBest2)

print('El accuracy al usar un conjunto de caracteristicas reducidas (2 caracteristicas) es: '+str(
    ↪ accuracy_KBest2))
```

El código anterior es idéntico al obtenido en la actividad I, simplemente se hizo que la función *SelectKBest()* realizara un filtrado tal que sobreviviesen 2 características, el resto del código es idéntico, simplemente se renombraron variables. Las características seleccionadas así como también algunas métricas se observan en el código 18, la matriz de confusión se observa en la figura 4.

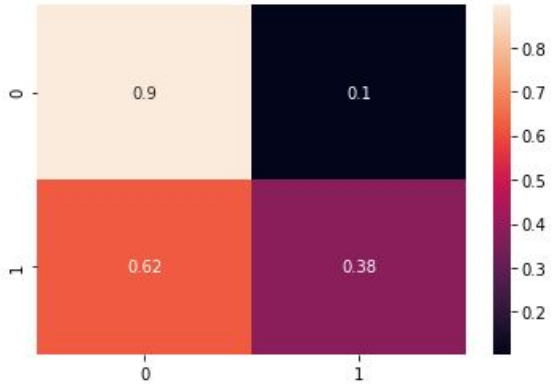

Figura 4: Matriz de confusión del SVM lineal con características reducidas a 2 (método de filtro).

Código 18: Características y métricas del SVM lineal al aplicar un método de tipo filtro.

```
Las caracteristicas seleccionadas son: ['plas', 'mass']
Los mejores hiperparametros son: {'C': 0.1}
El score es: 0.7142857142857143
El tiempo de entrenamiento es de: 0.015395879745483398
El accuracy al usar un conjunto de caracteristicas reducidas (2 caracteristicas) es:
    ↪ 0.7012987012987013
```

2.1.11. Actividad K

El código utilizado para implementar y entrenar un clasificador Random Forest corresponde al código 19.

Código 19: Entrenamiento de un clasificador Random Forest.

```
#Definimos el clasificador Random Forest
Forest1 = sklearn.ensemble.RandomForestClassifier(max_depth=3)
#Definimos el grid y lo entrenamos
grid_Forest1 = sklearn.model_selection.GridSearchCV(Forest1, {'n_estimators' : [50, 100, 150, 200,
    ↪ 250]}, cv=ps)

time8 = time.time() #<-- Inicio del entrenamiento
grid_Forest1.fit(Entrena_Vali, Entrena_Vali_class)
time9 = time.time() #Final del entrenamiento

#Vemos los hiperparametros del grid
print('Los mejores hiperparametros son: '+str(grid_Forest1.best_params_))
#Vemos el score
print('El score es: '+str(grid_Forest1.best_score_))
#Vemos el tiempo de entrenamiento
tiempo4 = time9-time8
print('El tiempo de entrenamiento es: '+str(tiempo4))

```

```
#Obtenemos el mejor estimador
estim_Forest1 = grid_Forest1.best_estimator_

#Obtenemos la prediccion del clasificador sobre el conjunto de validacion
predi_vali_Forest1 = estim_Forest1.predict(S_Vali)

#Obtenemos la matriz de confusion
Confu5 = sklearn.metrics.confusion_matrix(Vali_class, predi_vali_Forest1, normalize='true')

#Mostramos la matriz de confusion
matrix5 = sns.heatmap(Confu5, annot=True)

#Obtenemos el accuracy
accuracy_Forest1 = sklearn.metrics.accuracy_score(Vali_class, predi_vali_Forest1)
print('El accuracy es: '+str(accuracy_Forest1))
```

El código anterior presenta muchas similitudes a los observados en actividades anteriores, como la actividad D y E, con la diferencia de que acá se utilizó un clasificador Random Forest en vez de un SVM lineal. Para definir el Random Forest se utilizó la función *RandomForestClassifier()* de la librería *sklearn.ensemble*, dicha función recibe como parámetro la máxima profundidad del árbol, en este caso, por recomendación del equipo docente, se utilizó una profundidad de 3. El resto del código es similar a lo visto en la actividad D y E, primero se define un *GridSearch* utilizando el clasificador Random Forest implementado anteriormente, un diccionario con los hiperparámetros que la grilla deberá considerar, en este caso el hiperparámetro corresponde a "n_estimators", y finalmente se volvió a hacer uso del *PredifinedSplit* obtenido en la actividad D. Con la grilla configurada se entrenó utilizando el conjunto de entrenamiento y validación concatenados. Finalmente se obtuvieron el accuracy, la matriz de confusión, el score y los hiperparámetros del grid que se se puede observar en el código 20 y la figura 5.

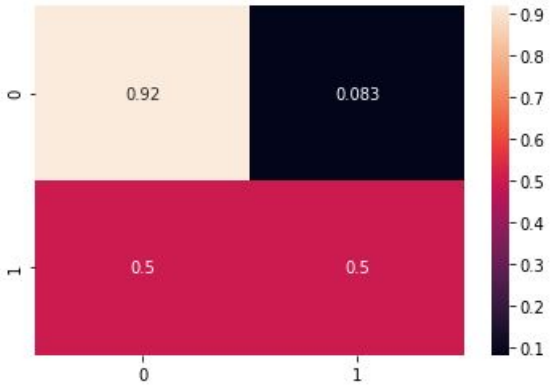

Figura 5: Matriz de confusión del Random Forest.

Código 20: Características y Métricas del Random Forest.

```
Los mejores hiperparametros son: {'n_estimators': 50}
El score es: 0.7142857142857143
El tiempo de entrenamiento es: 1.1715118885040283
El accuracy es: 0.7597402597402597
```

2.1.12. Actividad L

La selección de características al utilizar un clasificador Random Forest y el método de *Wrapper* se realizó utilizando el código 21.

Código 21: Selección de características utilizando un Random Forest y el método de Wrapper.

```
#Hacemos el model
model_Forest1 = sklearn.feature_selection.SelectFromModel(estim_Forest1)

#Entrenamos el model con el conjunto de entrenamiento
model_Forest1.fit(S_Entrena, Entrena_class)

#Obtenemos las caracteristicas seleccionadas
caracte_Forest1 = model_Forest1.get_support(indices=False)

#Recorremos la lista de caracteristicas seleccionadas y las añadimos a una lista donde se consideran
#solo a las caracteristicas seleccionadas
select_carac_Forest1 = []
for i in range(len(caracte_Forest1)):
  if caracte_Forest1[i] == True:
    select_carac_Forest1.append(nombres[i])
print('Las caracteristicas seleccionadas son: '+str(select_carac_Forest1))

#Obtenemos los distintos conjuntos con las caracteristicas reducidas
Entrena_Vali_redu_Forest1 = model_Forest1.transform(Entrena_Vali) #<--- Entrenamiento y
    ↪ validacion concatenados
Vali_redu_Forest1 = model_Forest1.transform(S_Vali) #<----- Conjunto de validacion
Test_redu_Forest1 = model_Forest1.transform(S_Test) #<----- Conjunto de prueba

#Reentrenamos con el conjunto reducido
Forest2 = sklearn.ensemble.RandomForestClassifier(max_depth=3)
grid_Forest2 = sklearn.model_selection.GridSearchCV(Forest2, param_grid={'n_estimators' : [50,
    ↪ 100, 150, 200, 250]}, cv=ps)

time10 = time.time() #<------- Inicio del entrenamiento
grid_Forest2.fit(Entrena_Vali_redu_Forest1, Entrena_Vali_class)
time11 = time.time() #<-------- Final del entrenamiento

#Vemos los hiperparametros del grid
print('Los mejores hiperparametros son: '+str(grid_Forest2.best_params_))
#Vemos el score
print('El score es: '+str(grid_Forest2.best_score_))
#Obtenemos el tiempo de entrenamiento
tiempo5 = time11-time10
print('El tiempo de entrenamiento es de: '+str(tiempo5))

#Obtenemos el mejor estimador
estim_Forest2 = grid_Forest2.best_estimator_

#Obtenemos la predicion del randomForest sobre el conjunto de validacion
```

```
predi_vali_Forest2 = estim_Forest2.predict(Vali_redu_Forest1)

#Obtenemos la matriz de confusion
Confu6 = sklearn.metrics.confusion_matrix(Vali_class, predi_vali_Forest2, normalize='true')

#Mostramos la matriz de confusion
matrix6 = sns.heatmap(Confu6, annot=True)

#Obtenemos el accuracy
accuracy_Forest2 = sklearn.metrics.accuracy_score(Vali_class, predi_vali_Forest2)
print('El accuracy es de: '+str(accuracy_Forest2))
```

De dicho código se puede apreciar que presenta una cantidad de similitudes considerable a lo realizado en las actividades F, G y H. Para realizar el método de *Wrapper* se utilizó la función *SelectFromModel()* dándole como parámetro el mejor clasificador obtenido de la grilla de la actividad anterior, el resultado de esto fue entrenado utilizando el conjunto de entrenamiento y luego mediante la función *get_support()* se obtuvo el arreglo que indica las características seleccionadas. Una vez se obtuvieron las características seleccionadas se procedió a reducir el conjunto de entrenamiento y validación concatenados, el de validación y el de prueba a conjuntos que solo consideren a las características seleccionadas, con esto se volvió a configurar otro *GridSearch* utilizando un clasificador Random Forest de profundidad 3, se entrenó y se obtuvieron las métricas solicitadas. Dicha métricas se pueden observar en el código 22 y en la figura 6.

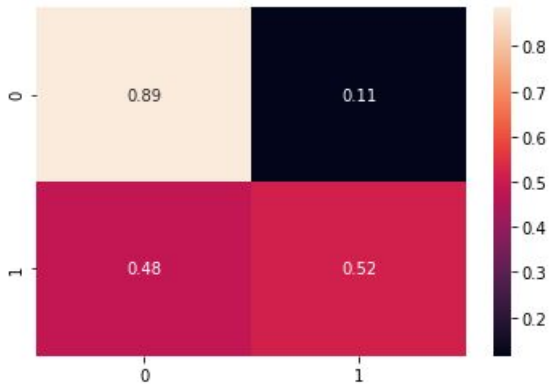

Figura 6: Matriz de confusión del Random Forest al reducir características con el método Wrapper.

Código 22: Características y Métricas del Random Forest al reducir características con el método de Wrapper.

```
Las caracteristicas seleccionadas son: ['plas', 'mass', 'age']
Los mejores hiperparametros son: {'n_estimators': 50}
El score es: 0.7272727272727273
El tiempo de entrenamiento es de: 1.1783244609832764
El accuracy es de: 0.7467532467532467
```

2.1.13. Actividad M

Para reducir características mediante el método tipo filtro, obteniendo 4 características luego del filtrado, se implementó el código 23.

Código 23: Selección de 4 características mediante el método de filtro en un Random Forest.

```
#Obtenemos el conjunto de caracteristicas (4)
Forest_SelectKBest4 = sklearn.feature_selection.SelectKBest(k=4).fit(S_Entrena, Entrena_class)
caract_Forest_KBest4 = Forest_SelectKBest4.get_support(indices=False) #<- Array que indica las caracteristicas consideradas
#Recorremos la lista de caracteristicas seleccionadas y las añadimos a una lista donde se consideran
#solo a las caracteristicas seleccionadas
select_carac_Forest_KBest4 = []
for i in range(len(caract_Forest_KBest4)):
  if caract_Forest_KBest4[i] == True:
    select_carac_Forest_KBest4.append(nombres[i])
print('Las caracteristicas seleccionadas son: '+str(select_carac_Forest_KBest4))

#Obtenemos los distintos conjuntos con las caracteristicas reducidas
Entrena_Vali_redu_Forest_KBest4 = Forest_SelectKBest4.transform(Entrena_Vali) #<--- Entrenamiento y validacion concatenados
Vali_redu_Forest_KBest4 = Forest_SelectKBest4.transform(S_Vali) #<----- Conjunto de validacion
Test_redu_Forest_KBest4 = Forest_SelectKBest4.transform(S_Test) #<----- Conjunto de prueba

#Reentrenamos con el conjunto reducido
Forest_KBest4 = sklearn.ensemble.RandomForestClassifier(max_depth=3)
grid_Forest_KBest4 = sklearn.model_selection.GridSearchCV(Forest_KBest4, param_grid={'n_estimators' : [50, 100, 150, 200, 250]}, cv=ps)

time12 = time.time() #<------- Inicio del entrenamiento
grid_Forest_KBest4.fit(Entrena_Vali_redu_Forest_KBest4, Entrena_Vali_class)
time13 = time.time() #<-------- Final del entrenamiento

#Vemos los hiperparametros del grid
print('Los mejores hiperparametros son: '+str(grid_Forest_KBest4.best_params_))
#Vemos el score
print('El score es: '+str(grid_Forest_KBest4.best_score_))
#Obtenemos el tiempo de entrenamiento
tiempo6 = time13-time12
print('El tiempo de entrenamiento es de: '+str(tiempo6))

#Obtenemos el mejor estimador
estim_Forest_KBest4 = grid_Forest_KBest4.best_estimator_

#Obtenemos la prediccion del clasificador sobre el conjunto de validacion
predi_vali_Forest_KBest4 = estim_Forest_KBest4.predict(Vali_redu_Forest_KBest4)

#Obtenemos la matriz de confusión
Confu7 = sklearn.metrics.confusion_matrix(Vali_class, predi_vali_Forest_KBest4, normalize='true')

```

```
#Mostramos la matriz de confusion
matrix7 = sns.heatmap(Confu7, annot=True)

#Obtenemos el accuracy del SVM lineal
accuracy_Forest_KBest4 = sklearn.metrics.accuracy_score(Vali_class, predi_vali_Forest_KBest4)

print('El accuracy al usar un conjunto de caracteristicas reducidas (4 caracteristicas) es: '+str(
    ↪ accuracy_Forest_KBest4))
```

Dicho código utiliza la función *SelectKBest()* para aplicar un método de tipo filtro, a dicha función se le indicó que filtrara las características con tal de que "sobrevivieran" 4. Luego simplemente se obtuvieron las características consideradas, los distintos conjuntos con características reducidas y se entrenó un *GridSearch* utilizando un Random Forest de profundidad 3 y el conjunto de entrenamiento y validación concatenados. Finalmente se obtuvieron métricas como el score, el accuracy y el tiempo de entrenamiento del Random Forest al usar un método de tipo filtro para seleccionar características, además se obtuvo la matriz de confusión normalizada. Lo anterior se puede observar en el código 24 y la figura 7.

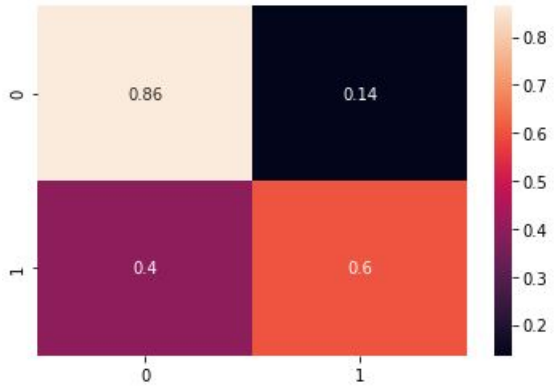

Figura 7: Matriz de confusión del Random Forest con características reducidas a 4 (método de filtro).

Código 24: Características y métricas del Random Forest al aplicar un método de tipo filtro.

```
Las caracteristicas seleccionadas son: ['preg', 'plas', 'mass', 'age']
Los mejores hiperparametros son: {'n_estimators': 50}
El score es: 0.7207792207792207
El tiempo de entrenamiento es de: 1.180464506149292
El accuracy al usar un conjunto de caracteristicas reducidas (4 caracteristicas) es:
    ↪ 0.7662337662337663
```

2.1.14. Actividad N

El código implementado en la presente actividad corresponde al observado en el código 25

Código 25: Selección de 2 características mediante el método de filtro en un Random Forest.

```
#Obtenemos el conjunto de caracteristicas (2)
Forest_SelectKBest2 = sklearn.feature_selection.SelectKBest(k=2).fit(S_Entrena, Entrena_class)
caract_Forest_KBest2 = Forest_SelectKBest2.get_support(indices=False) #<- Array que indica las caracteristicas consideradas
#Recorremos la lista de caracteristicas seleccionadas y las añadimos a una lista donde se consideran
#solo a las caracteristicas seleccionadas
select_carac_Forest_KBest2 = []
for i in range(len(caract_Forest_KBest2)):
  if caract_Forest_KBest2[i] == True:
    select_carac_Forest_KBest2.append(nombres[i])
print('Las caracteristicas seleccionadas son: '+str(select_carac_Forest_KBest2))

#Obtenemos los distintos conjuntos con las caracteristicas reducidas
Entrena_Vali_redu_Forest_KBest2 = Forest_SelectKBest2.transform(Entrena_Vali) #<--- Entrenamiento y validacion concatenados
Vali_redu_Forest_KBest2 = Forest_SelectKBest2.transform(S_Vali) #<----- Conjunto de validacion
Test_redu_Forest_KBest2 = Forest_SelectKBest2.transform(S_Test) #<----- Conjunto de prueba

#Reentrenamos con el conjunto reducido
Forest_KBest2 = sklearn.ensemble.RandomForestClassifier(max_depth=3)
grid_Forest_KBest2 = sklearn.model_selection.GridSearchCV(Forest_KBest2, param_grid={'n_estimators' : [50, 100, 150, 200, 250]}, cv=ps)

time14 = time.time() #<------- Inicio del entrenamiento
grid_Forest_KBest2.fit(Entrena_Vali_redu_Forest_KBest2, Entrena_Vali_class)
time15 = time.time() #<-------- Final del entrenamiento

#Vemos los hiperparametros del grid
print('Los mejores hiperparametros son: '+str(grid_Forest_KBest2.best_params_))
#Vemos el score
print('El score es: '+str(grid_Forest_KBest2.best_score_))
#Vemos el tiempo de entrenamiento
tiempo7 = time15-time14
print('El tiempo de entrenamiento es de: '+str(tiempo7))

#Obtenemos el mejor estimador
estim_Forest_KBest2 = grid_Forest_KBest2.best_estimator_

#Obtenemos la prediccion del clasificador sobre el conjunto de validacion
predi_vali_Forest_KBest2 = estim_Forest_KBest2.predict(Vali_redu_Forest_KBest2)

#Obtenemos la matriz de confusión
Confu8 = sklearn.metrics.confusion_matrix(Vali_class, predi_vali_Forest_KBest2, normalize='true')

#Mostramos la matriz de confusion
matrix8 = sns.heatmap(Confu8, annot=True)

#Obtenemos el accuracy del SVM lineal
accuracy_Forest_KBest2 = sklearn.metrics.accuracy_score(Vali_class, predi_vali_Forest_KBest2)

```

```
print('El accuracy al usar un conjunto de caracteristicas reducidas (2 caracteristicas) es: '+str(
    ↪ accuracy_Forest_KBest2))
```

El código anterior es análogo al de la actividad M pero considerando un método de tipo filtro que rescate 2 características en vez de 4. Las métricas resultantes corresponden a las apreciadas en el código 26 y la figura 8.

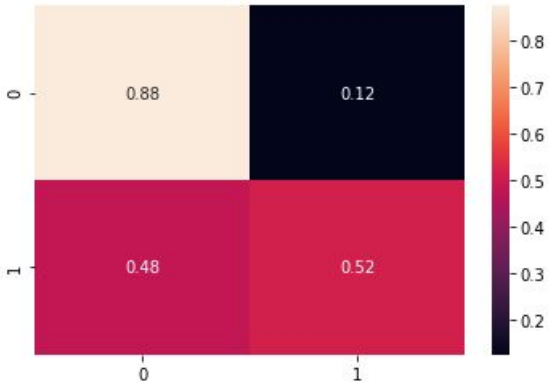

Figura 8: Matriz de confusión del Random Forest con características reducidas a 2 (método de filtro).

Código 26: Características y métricas de un Random Forest al aplicar un método de tipo filtro.

```
Las caracteristicas seleccionadas son: ['plas', 'mass']
Los mejores hiperparametros son: {'n_estimators': 200}
El score es: 0.7077922077922078
El tiempo de entrenamiento es de: 1.3751015663146973
El accuracy al usar un conjunto de caracteristicas reducidas (2 caracteristicas) es:
    ↪ 0.7402597402597403
```

2.1.15. Actividad O

Finalmente, en la última actividad de la sección de pruebas se debían probar los distintos clasificadores obtenidos de las grillas implementadas a lo largo de las demás actividades sobre los conjuntos de pruebas con características reducidas correspondientes. Lo anterior se observa en el código 27.

Código 27: Prueba de los distintos clasificadores ante el conjunto de prueba con características reducido.

```
#El estimador SVM lineal sin reducir caracteristicas es estim_lin
#El estimador del SVM lineal con caracteristicas reducidas al usar SelectFromModel es estim_lin2
#El estimador del SVM lineal al seleccionar a las 4 mejores caracteristicas es estim_KBest4
#El estimador del SVM lineal al seleccionar a las 2 mejores caracteristicas es estim_KBest2
#El randomForest sin reducir caracteristicas es estim_Forest1
#El randomForest con caracteristicas reducidas al usar SelectFromModel es estim_Forest2
#El randomFores al seleccionar a las 4 mejores caracteristicas es estim_Forest_KBest4
#El randomForest al seleccionar a las 2 mejores caracteristicas es estim_Forest_KBest2

```

```
#Obtenemos las prediciones de los distintos clasificadores sobre el conjunto Test
predi_test_lin = estim_lin.predict(S_Test) #SVM lineal sin reducir caracteristicas
predi_test_lin2 = estim_lin2.predict(Test_redu) #SVM lineal con SelectFromModel
predi_test_KBest4 = estim_KBest4.predict(Test_redu_KBest4) #SVM lineal con las 4 mejores caracteristicas
predi_test_KBest2 = estim_KBest2.predict(Test_redu_KBest2) #SVM lineal con las 2 mejores caracteristicas
predi_test_Forest1 = estim_Forest1.predict(S_Test) #Random Forest sin reducir caracteristicas
predi_test_Forest2 = estim_Forest2.predict(Test_redu_Forest1) #Random Forest con SelectFromModel
predi_test_Forest_KBest4 = estim_Forest_KBest4.predict(Test_redu_Forest_KBest4) #Random Forest con las 4 mejores caracteristicas
predi_test_Forest_KBest2 = estim_Forest_KBest2.predict(Test_redu_Forest_KBest2) #Random Forest con las 2 mejores caracteristicas

#Obtenemos las matrices de confusion
Confu_lin = sklearn.metrics.confusion_matrix(Test_class, predi_test_lin, normalize='true')
Confu_lin2 = sklearn.metrics.confusion_matrix(Test_class, predi_test_lin2, normalize='true')
Confu_KBest4 = sklearn.metrics.confusion_matrix(Test_class, predi_test_KBest4, normalize='true')
Confu_KBest2 = sklearn.metrics.confusion_matrix(Test_class, predi_test_KBest2, normalize='true')
Confu_Forest1 = sklearn.metrics.confusion_matrix(Test_class, predi_test_Forest1, normalize='true')
Confu_Forest2 = sklearn.metrics.confusion_matrix(Test_class, predi_test_Forest2, normalize='true')
Confu_Forest_KBest4 = sklearn.metrics.confusion_matrix(Test_class, predi_test_Forest_KBest4, normalize='true')
Confu_Forest_KBest2 = sklearn.metrics.confusion_matrix(Test_class, predi_test_Forest_KBest2, normalize='true')

#Obtenemos los accuracys
accuracy_f_lin = sklearn.metrics.accuracy_score(Test_class, predi_test_lin)
accuracy_f_lin2 = sklearn.metrics.accuracy_score(Test_class, predi_test_lin2)
accuracy_f_KBest4 = sklearn.metrics.accuracy_score(Test_class, predi_test_KBest4)
accuracy_f_KBest2 = sklearn.metrics.accuracy_score(Test_class, predi_test_KBest2)
accuracy_f_Forest1 = sklearn.metrics.accuracy_score(Test_class, predi_test_Forest1)
accuracy_f_Forest2 = sklearn.metrics.accuracy_score(Test_class, predi_test_Forest2)
accuracy_f_Forest_KBest4 = sklearn.metrics.accuracy_score(Test_class, predi_test_Forest_KBest4)
accuracy_f_Forest_KBest2 = sklearn.metrics.accuracy_score(Test_class, predi_test_Forest_KBest2)
```

En dicho código se observa que se obtuvieron las matrices de confusión y los accuracy de los distintos clasificadores. Debido a que se debían probar 8 clasificadores los resultados obtenidos se separarán por clasificador.

- **SVM lineal sin realizar selección de características:** El accuracy para este caso se observa en el código 28, mientras que la matriz de confusión se aprecia en la figura 9.

Código 28: Métricas de un SVM lineal sin reducir características en el conjunto de prueba.

```
Para el caso del SVM lineal sin reducir caracteristicas se escoge
el mejor clasificador y se evalua el conjunto de Test obteniendo lo siguiente:

El accuracy sobre el conjunto de test es: 0.7727272727272727
La matriz de confusión es:

```

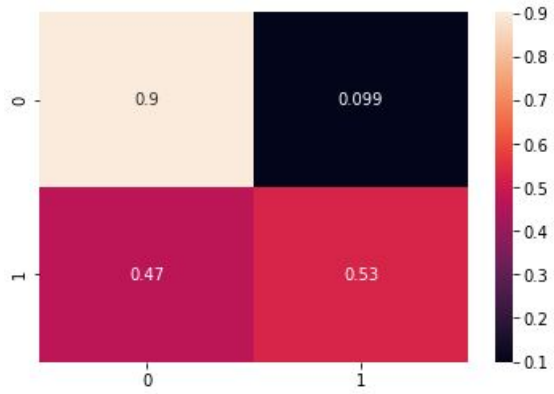

Figura 9: Matriz de confusión del SVM lineal sin reducir características en el conjunto de prueba.

- **SVM lineal al aplicar en método de *Wrapper*:** Para este caso el accuracy se aprecia en el código 29 y la matriz de confusión se observa en la figura 10.

Código 29: Métricas de un SVM lineal al aplicar una estrategia de Wrapper en el conjunto de prueba.

```
Para el caso del SVM lineal, al utilizar la funcion SelectFromModel, escoger
el mejor clasificador y evaluarlo sobre el conjunto de Test se obtuvo lo siguiente:

El accuracy sobre el conjunto de test es: 0.7662337662337663
La matriz de confusión es:

```

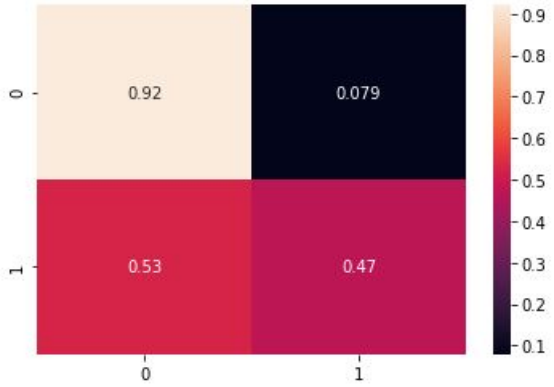

Figura 10: Matriz de confusión del SVM lineal al usar una estrategia de *Wrapper* en el conjunto de prueba.

- **SVM lineal al aplicar en método tipo filtro para conseguir 4 características:** Ante este método de reducción de características se obtuvo el accuracy del código 30 y la matriz de confusión de la figura 11.

Código 30: Métricas de un SVM lineal al aplicar un método de tipo filtro en el conjunto de prueba.

```
Para el caso del SVM lineal, usando un filtro para las caracteristicas escogiendo
las 4 mejores, tomando el mejor clasificador y evaluarlo sobre el conjunto de Test
se obtuvo lo siguiente:

El accuracy sobre el conjunto de test es: 0.7532467532467533
La matriz de confusión es:

```

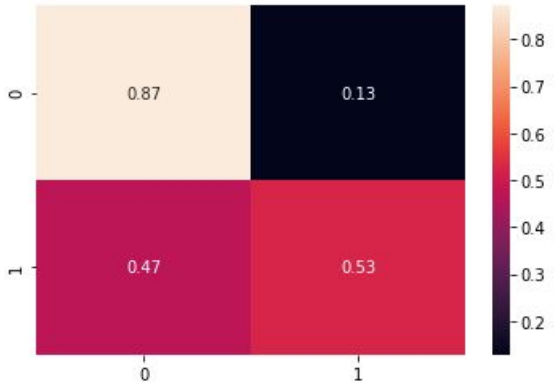

Figura 11: Matriz de confusión del SVM lineal con características reducidas a 4 (método de filtro) en el conjunto de prueba.

- **SVM lineal al aplicar en método tipo filtro para conseguir 2 características:** Al reducir a 2 las características en un SVM lineal se obtuvo el accuracy del código 31 y la matriz de confusión de la figura 12.

Código 31: Métricas de un SVM lineal al aplicar un método de tipo filtro en el conjunto de prueba.

```
Para el caso del SVM lineal, usando un filtro para las caracteristicas escogiendo
las 2 mejores, tomando el mejor clasificador y evaluarlo sobre el conjunto de Test
se obtuvo lo siguiente:

El accuracy sobre el conjunto de test es: 0.7662337662337663
La matriz de confusión es:

```

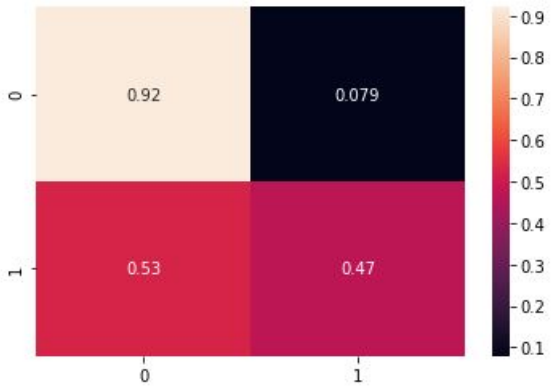

Figura 12: Matriz de confusión del SVM lineal con características reducidas a 2 (método de filtro) en el conjunto de prueba.

- **Random Forest sin realizar selección de características:** Si no se realiza selección de características se obtiene el accuracy del código 32 y la matriz de confusión de la figura 13.

Código 32: Métricas de un Random Forest sin reducir características en el conjunto de prueba.

```
Para el caso del Random Forest sin reducir caracteristicas se escoge
el mejor clasificador y se evalua el conjunto de Test obteniendo lo siguiente:

El accuracy sobre el conjunto de test es: 0.7532467532467533
La matriz de confusión es:

```

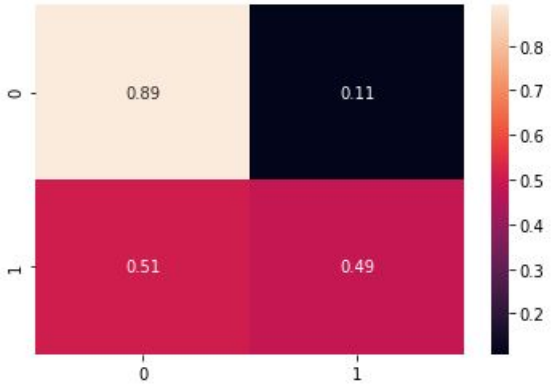

Figura 13: Matriz de confusión del Random Forest sin reducir características en el conjunto de prueba.

- **Random Forest al aplicar en método de *Wrapper*:** Al reducir las características en un Random Forest mediante el método de *Wrapper* se obtuvo el accuracy del código 33 y la matriz de confusión de la figura 14.

Código 33: Métricas de un Random Forest al aplicar una estrategia de Wrapper en el conjunto de prueba.

```
Para el caso del Random Forest, al utilizar la funcion SelectFromModel, escoger
el mejor clasificador y evaluarlo sobre el conjunto de Test se obtuvo lo siguiente:

El accuracy sobre el conjunto de test es: 0.7467532467532467
La matriz de confusión es:

```

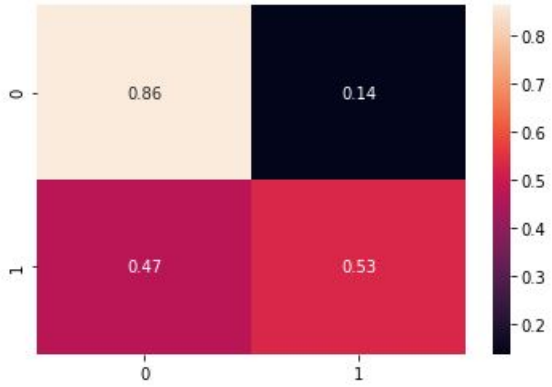

Figura 14: Matriz de confusión del Random Forest al usar una estrategia de *Wrapper* en el conjunto de prueba.

- **Random Forest al aplicar en método tipo filtro para conseguir 4 características:** Ante este método de reducción de características se obtuvo el accuracy del código 34 y la matriz de confusión de la figura 15.

Código 34: Métricas de un Random Forest al aplicar un método de tipo filtro en el conjunto de prueba.

```
Para el caso del Random Forest, usando un filtro para las caracteristicas escogiendo
las 4 mejores, tomando el mejor clasificador y evaluarlo sobre el conjunto de Test
se obtuvo lo siguiente:

El accuracy sobre el conjunto de test es: 0.7467532467532467
La matriz de confusión es:

```

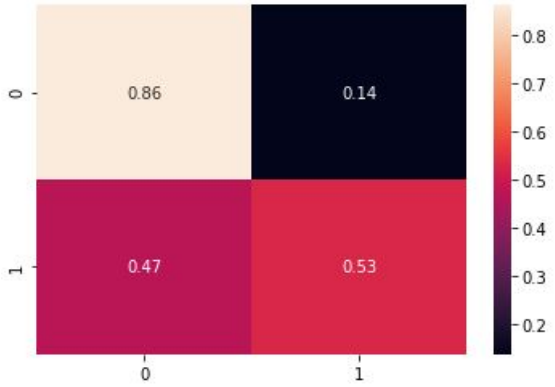

Figura 15: Matriz de confusión del Random Forest con características reducidas a 4 (método de filtro) en el conjunto de prueba.

- **Random Forest al aplicar en método tipo filtro para conseguir 2 características:** Ante esta situación se obtuvo el accuracy del código 35 y la matriz de confusión de la figura 16.

Código 35: Métricas de un Random Forest al aplicar un método de tipo filtro en el conjunto de pruebas.

```
Para el caso del Random Forest, usando un filtro para las caracteristicas escogiendo
las 2 mejores, tomando el mejor clasificador y evaluarlo sobre el conjunto de Test
se obtuvo lo siguiente:

El accuracy sobre el conjunto de test es: 0.7727272727272727
La matriz de confusión es:

```

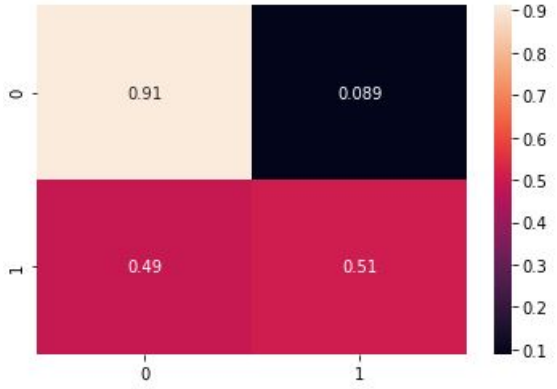

Figura 16: Matriz de confusión del Random Forest con características reducidas a 2 (método de filtro) en el conjunto de prueba.

2.2. Análisis

2.2.1. Actividad A

De los casos observados durante la actividad O de la subsección anterior se logra apreciar que el clasificador SVM lineal que entregaba un mayor accuracy corresponde al caso donde no se realizó selección de características. Al observar la matriz de confusión se logra apreciar que el clasificador predice mucho mejor los casos positivos, esto se respalda en el hecho de que presenta una tasa de verdaderos positivos (Predice positivo y era positivo) muy alta, de un 0.9, mientras que presenta una tasa de falsos positivos (predice positivo y era negativo) relativamente pequeña, de un 0.1. Por el contrario se aprecia que tiene una tasa de verdaderos negativos (dice negativo y es negativo) muy similar a la tasa de falsos negativos (dice negativo y es positivo), de esto se desprende que tiene una mayor dificultad discerniendo los casos negativos

Para el caso de los Random Forest se logra apreciar que se tiene un mayor accuracy cuando se utiliza una estrategia de tipo filtro para reducir las características a 2. Si se observa la matriz de confusión se observa lo mismo que para el caso del SVM lineal sin reducir características, es decir, el clasificador clasifica mejor a los casos positivos que negativos.

Pese a lo anterior se debe hacer hincapié en el hecho de que la explicación dada anteriormente no es certera en todas las circunstancias en que se ejecuta el código, esto se debe al uso de la función *train_test_split()* que provoca que los conjuntos de entrenamiento, validación y prueba no tengan siempre las mismas muestras, esto provoca que cada vez que se ejecute el código varíen los accuracy, las matrices de confusión y para el caso de seleccionar características mediante la estrategia tipo *Wrapper* varían también las características seleccionadas, en el caso expuesto durante el informe se aprecia que en el SVM lineal se eligieron 2 características y en el Random Forest se eligen 3 pero existen ocasiones donde pueden llegar a obtenerse hasta 4 características.

2.2.2. Actividad B

El hecho de considerar menos características influye directamente en el accuracy, donde por ejemplo, en el caso del SVM lineal se observa que al reducir características los accuracy disminuyen, sin

embargo, se debe notar que esta disminución no es muy drástica, el accuracy a lo más disminuyó en 2 unidades.

Para el caso del Random Forest se observa que si se reducen características el accuracy a lo más disminuye en 1 unidad o aumenta en 3, esto arroja luz de que a diferencia del SVM lineal, el Random Forest tiene un mayor margen de mejora.

Sin embargo, y al igual que para la actividad A de esta sección, la explicación dada es válida para el caso expuesto en el informe, más puede no ser "universal" y preservarse ante todas las ejecuciones del código.

2.2.3. Actividad C

Para el caso del SVM lineal se logra observar que la reducción de características sirve para reducir el tiempo de entrenamiento, pues en un inicio se tenía un tiempo de entrenamiento de 0.02456 y al reducir características se redujo este tiempo a 0.01564 (método *Wrapper*), 0.02055 (Método tipo filtro, 4 características) y 0.01539 (método tipo filtro, 2 características). Es decir, el trade-off para el caso del SVM lineal es: Si se disminuyen características el tiempo de entrenamiento disminuye pero el accuracy empeora un poco.

Para el caso del Random Forest se aprecia que el reducir características influye en el tiempo de entrenamiento de manera negativa pues lo aumenta, esto se observa del hecho de que en un inicio se tenía un tiempo de entrenamiento de 1.17151 y al reducir características el tiempo aumenta a 1.17832 (método *Wrapper*), 1.18046 (método tipo filtro, 4 características) y 1.3751 (método tipo filtro, 2 características). De lo anterior se observa algo interesante, pues el Random Forest que tiene un mayor tiempo de entrenamiento es a su vez el que tiene un mayor accuracy en el conjunto de prueba, de esto se infiere que para este caso el trade-off es: Si se disminuyen características se puede llegar a mejorar el accuracy pero el tiempo de entrenamiento aumentará.

Nuevamente se debe hacer hincapié en que la explicación dada podría llegar a variar en alguna ejecución del código.

3. Conclusión

A modo de conclusión se puede afirmar el cumplimiento de los objetivos impuestos durante la tarea 3 del curso EL4106 Inteligencia computacional donde se logró implementar todas y cada una de las actividades solicitadas por el cuerpo docente de una manera satisfactoria.

Dentro de los aprendizajes obtenidos se destaca el uso de nuevas funciones como *SelectFromModel()*, *SelectKBest()*, *RandomForestClassifier()* y *PredifinedSplit()*, funciones que proveen herramientas muy útiles a la hora de trabajar con clasificadores, grillas y reducción de características. Además se destaca la aplicación de conocimiento vistos en clases en un trabajo más aterrizado a la vida real, donde se aprecia de manera más tangible como la reducción de características puede repercutir en los accuracy, matrices de confusión y tiempos de entrenamiento de los clasificadores implementados.

Dentro de los resultados obtenidos se destaca principalmente el hecho de que debido al uso de *train_test_split()* los resultados obtenidos no son universales y que pueden variar de ejecución en ejecución. Sin embargo se espera que al reducir características el trade-off de un SVM lineal, a menos características menos tiempo de entrenamiento pero peor accuracy, y el de un Random Forest, a menos caracteristicas más tiempo de entrenamiento pero un margen de mejora en el accuracy, se mantenga.

Una forma de optimizar el código implementado podría ser no realizando tantos clasificadores y grillas para los entrenamientos, sino que usar solo una grilla y un clasificador (uno para el caso del SVM lineal y uno para el caso del Random Forest), de esta manera se lograría un ahorro de memoria.