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

Tarea 3

Selección de características

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

Fecha de entrega: 16 de mayo de 2021
Santiago, Chile

Índice de Contenidos

Índice de Figuras

Índice de Códigos

1. Introducción

La utilización de clasificadores no sólo requiere de sofisticadas herramientas matemáticas, también es necesario conocer el "Mundo Real" ya que la utilización de estos requiere distinguir características relevantes para la distinción de clases, este problema aún no posee formulación matemática para resolverlo, si es que este posee solución.
En el presente informe se realizará un acercamiento a las estrategias de evaluación de características, para lo cual se utilizará un data set para un problema de clasificación de diabetes, utilizando este ejemplo para ejemplificar la importancia de esta área, debido al tiempo y dinero que se debe invertir para obtener todas las características del data set.
Esta experiencia considera 2 focos distintos para la reducción de características, como lo son Wrapper y Filtrado.
Para realizar la experiencia aparte del data set *Pima Indians Diabetes Database* se deben utilizar las bibliotecas scipy.io, pandas, numpy, seaborn, time y sklearn.
La experiencia consta de los siguientes bloques:

- Visualización de datos y contexto de cada característica
- Inicialización de clasificadores
- Clasificación variando número de características y estrategias
- Comparación de resultados

Los cuales osn explicados a medida se desarrolla el informe.

2. Desarrollo

2.1. Pruebas

Para esta experiencia se utiliza el dataset *Pima Indians Diabetes Database*,el que proviene de una base datos más grande. El dataset está enfocado en mujeres de mínimo 21 años de edad con herencia Pima India, de las cuales 268 poseen diabetes y 500 no. La mayoría de características del dataset requieren de un examen médico, por lo cual la reducción del número de características se hace importante.

2.1.1. a)

Las características del data set son las siguientes:

- preg: Número de veces embarazada
- plas: Concentración de glucosa en plasma a 2 horas en una prueba de tolerancia oral
- pres: Presión arterial diastólica [mmHg]
- skin: Espesor del pliegue cutáneo del tríceps [mm]
- insu: Suero de insulina, 2 horas [mu U / ml]
- mass Índice de masa corporal $[kg/m^2]$
- pedi: Función del pedigrí de la diabetes
- age: Edad en años
- class: Si posee o no diabetes

2.1.2. b)

Para resolver el problema lo primero es leer el archivo, subir las bibliotecas a utilizar, reemplazar las etiquetas de clase por 0 y 1 según corresponda y dividir en conjuntos de entrenamiento, validación y prueba, lo cual se realiza mediante el código:

Código 1: Subida de datos a Collab

```
#subir archivo diabetes.arff
from google.colab import files
uploaded= files.upload()

#Bibliotecas
from scipy.io import arff
 %matplotlib inline
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
```

```
import time
import sklearn

#p1 b
from sklearn.model_selection import train_test_split

data = arff.loadarff('diabetes.arff')
df= pd.DataFrame(data[0])
df['class'].replace(b'tested_positive',0, inplace = True)
df['class'].replace(b'tested_negative',1, inplace = True)

entrenamiento= train_test_split(df,test_size=0.60, random_state=7)[1].reset_index(drop = True)
validacion= train_test_split(df,test_size=0.2, random_state=7)[1].reset_index(drop = True)
test= train_test_split(df,test_size=0.2, random_state=7)[1].reset_index(drop = True)
```

2.1.3. c)

Una vez divididos los conjuntos se deben preprocesar las características, lo cual es ejecutado con el siguiente código:

Código 2: Preprocesamiento de características

```

#Parte c
from sklearn import preprocessing
scaler= preprocessing.StandardScaler()
scaler_entrenado=scaler.fit(entrenamiento.iloc[:,:9]) #Entrenamiento de standardscaler

#Aplicación de standardscaler a conjuntos
scaled_entrenamiento= scaler.transform(entrenamiento.iloc[:,:9])
scaled_validacion= scaler.transform(validacion.iloc[:,:9])
scaled_test= scaler.transform(test.iloc[:,:9])
```

2.1.4. d)

Se debe realizar una clasificación inicial usando un SVM lineal, el cual debe usar una grilla con parametro $C = [0.0001, 0.001, 0.1]$, posteriormente se debe utilizar el SVM lineal en el conjunto de validación, de esta forma es posible evaluar la calidad de sus hiperparámetros. El código utilizado es el siguiente:

Código 3: Clasificador SVM-l inicial

```

from sklearn import svm
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import PredefinedSplit

test_fold = [-1 for _ in range(len(validacion))] + [0 for _ in range(len(entrenamiento))]
y = np.concatenate([scaled_validacion, scaled_entrenamiento])
```

```
ps = PredefinedSplit(test_fold)

parametros= \{'C':[0.0001, 0.001, 0.1]\}
svmlineal = svm.SVC(kernel= 'linear', probability=False)
clfl = GridSearchCV(svmlineal, parametros, cv=ps)

scaled_concat = np.concatenate((scaled_entrenamiento,scaled_validacion))
conjuntos_concat= pd.concat([entrenamiento.iloc[:,8],validacion.iloc[:,8]])
conjuntos_concat= conjuntos_concat.reset_index(drop= True)

tiempo_svmlineal_inicio = time.time()
clfl.fit( scaled_concat , conjuntos_concat)#entrenamiento
tiempo_svmlineal_final = time.time()

print(clfl.best_params_)
y_pred_svml=clfl.predict(scaled_validacion)
print(f"Tiempo de entrenamiento: \{tiempo_svmlineal_final - tiempo_svmlineal_inicio\} [s]")
```

El cual al ser ejecutado da como resultado que el mejor hiperparámetro para C es $C = 0.1$ y un tiempo de entrenamiento de $0.02[s]$ aproximadamente.

2.1.5. e)

Para el clasificador recién creado se debe generar una matriz de confusión normalizada, ver su accuracy y una visualización de *heatmap*, esto se ejecuta de la siguiente manera:

Código 4: Matriz de confusión para clasificador inicial

```

matriz_confusion_svml = sklearn.metrics.confusion_matrix(conjuntos_concat.iloc[:,8], y_pred_svml
    ↪ , normalize='true')
print(f'accuracy = {(matriz_confusion_svml[0][0]+matriz_confusion_svml[1][1])/2}')

plt.figure()
sns.heatmap(matriz_confusion_svml, annot= True)
plt.title('Matriz de confusión SVM(lineal) preliminar')
plt.show()
```

El cual imprime en pantalla la siguiente matriz de confusión:

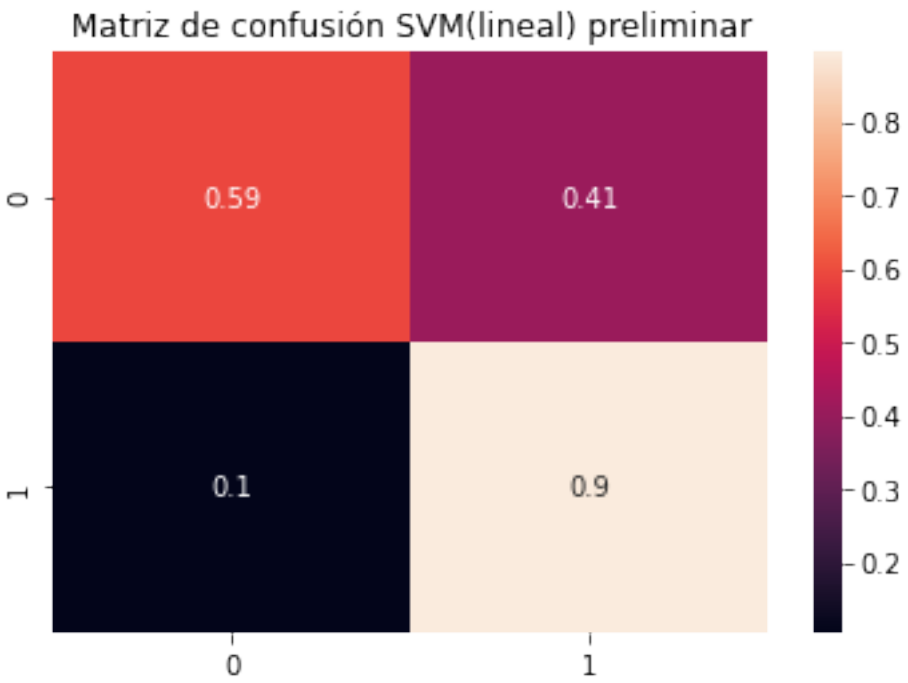

Figura 1: Matriz de confusión svm lineal con heatmap

Donde el accuracy obtenido es de 0.74 aproximadamente.

2.1.6. f)

Se debe realizar una selección de características con *SelectFromModel*, función que sirve para la selección de atributos mediante la estrategia *wrapper*. Para utilizar esta técnica se usa el mejor SVM encontrado en la parte anterior a través de la grilla. El código que realiza esto es el siguiente:

Código 5: Wrapper para SVM lineal

```
1
2  from sklearn.feature_selection import SelectFromModel
3
4  seleccion = SelectFromModel(estimator=svm.SVC(kernel= 'linear',C=0.1, probability=False)).fit(
       ↪ scaled_entrenamiento, entrenamiento.iloc[:,8])
5
6
7  caract= ['preg','plas','pres','skin','insu','mass','pedi','age']
8  seleccion.get_support()
9  L=[]
10 val = []
11 for i in range(len(caract)):
12     if seleccion.get_support()[i] == True:
13         L.append(caract[i])
14         val.append(seleccion.estimator_.coef_[0][i])
15     else: None
```

```
print(f"Las mejores características son: {L}")
```

El cual retorna como características más importantes 'plas' y 'mass'.

2.1.7. g)

Una vez obtenidas las características más relevantes, se debe entrenar un clasificador utilizando sólo estas características.Lo que se realiza mediante el código:

Código 6: Reduccion de caracteristicas mediante wrapper

```

car1= scaled_entrenamiento[:,1]
car2= scaled_entrenamiento[:,5]
cars= np.array([car1,car2]) # despues se concatenan las filas traspuestas para que sean columnas
tiempo_svmlineal2_inicio = time.time()
svml2 = svm.SVC(kernel= 'linear',C=0.1, probability=False).fit(cars.transpose(), entrenamiento.iloc
    ↪ [:,8])
tiempo_svmlineal2_final = time.time()
print(f"El tiempo de entrenamiento es {tiempo_svmlineal2_final - tiempo_svmlineal2_inicio} [s]")
```

Del cual se obtiene un tiempo de entrenamiento aproximado de $0.005[s]$.

2.1.8. h)

Una vez obtenido el nuevo clasificador se debe generar una matriz de confusión normalizada, ver su accuracy y una visualización de este en *heatmap*, esto se ejecuta de la siguiente manera:

Código 7: Matriz de confusión para SVM lineal con reducción de características

```

carv1= scaled_validacion[:,1]
carv2= scaled_validacion[:,5]
carsv= np.array([carv1,carv2])
y_pred_svml2 = svml2.predict(carsv.transpose()) #se concatenan las filas traspuestas para que sean
    ↪  columnas
matriz_confusion_svml2 = sklearn.metrics.confusion_matrix(validacion.iloc[:,8], y_pred_svml2,
    ↪ normalize='true')
print(f'accuracy = {(matriz_confusion_svml2[0][0]+matriz_confusion_svml2[1][1])/2}')

plt.figure()
sns.heatmap(matriz_confusion_svml2, annot= True)
plt.title('Matriz de confusión SVM(lineal) selección de características')
plt.show()
```

El que despliega la matriz normalizada con heatmap:

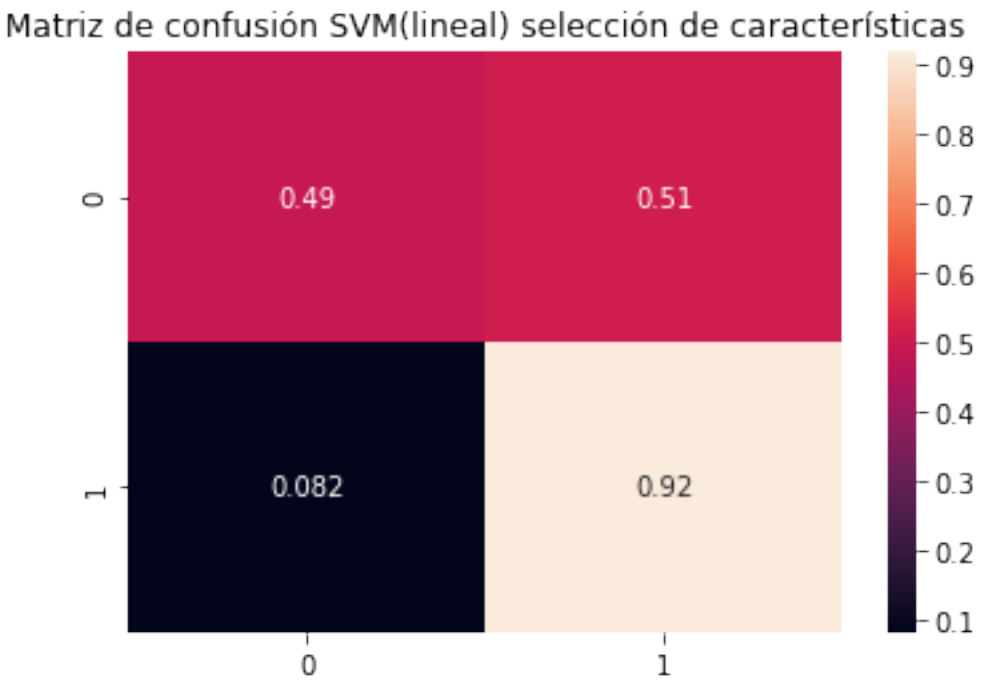

Figura 2: Matriz de confusión svm lineal, 2 características seleccionadas con wrapper

Del que se obtiene un accuracy de 0.70 aproximadamente.

2.1.9. i)

Se debe repetir el proceso de selección de características, esta vez mediante la estrategia de filtrado. Utilizando la función *SelectKBest* se obtienen las 4 mejores caracteristicas y posteriormente se entrena un SVM lineal. El código que ejecuta lo anterior es el siguiente:

Código 8: Filtro 4 características SVM lineal

```
1
2  from sklearn.feature_selection import SelectKBest
3  seleccion2 = SelectKBest(k=4).fit_transform(scaled_entrenamiento, entrenamiento.iloc[:,8])
4  caract= ['preg','plas','pres','skin','insu','mass','pedi','age']
5  L=[]
6  for i in range(len(caract)): ##No pude obtnerlo mediante get_param asi que se improviso con una
       ↪ revision por fuerza bruta
7      if scaled_entrenamiento[0][i] in seleccion2[0] :
8          L.append(caract[i])
9      else: None
10 print(f"Las mejores 4 características son: {L}")
11 ##########################################################
12 #g
13 car1i= scaled_entrenamiento[:,0]
14 car2i= scaled_entrenamiento[:,1]
15 car3i= scaled_entrenamiento[:,5]
```

```
car4i= scaled_entrenamiento[:,7]
carsi= np.array([car1i,car2i,car3i,car4i])
tiempo_svmlineali_inicio = time.time()
svmli = svm.SVC(kernel= 'linear',C=0.1, probability=False).fit(carsi.transpose(), entrenamiento.iloc
    ↪ [:,8])
tiempo_svmlineali_final = time.time()
print(f"El tiempo de entrenamiento es {tiempo_svmlineali_final - tiempo_svmlineali_inicio} [s]")
#########################################
#h
carv1i= scaled_validacion[:,0]
carv2i= scaled_validacion[:,1]
carv3i= scaled_validacion[:,5]
carv4i= scaled_validacion[:,7]
carsvi= np.array([carv1i,carv2i,carv3i,carv4i])
y_pred_svmli = svmli.predict(carsvi.transpose()) #se concatenan las filas traspuestas para que sean
    ↪ columnas
matriz_confusion_svmli = sklearn.metrics.confusion_matrix(validacion.iloc[:,8], y_pred_svmli,
    ↪ normalize='true')
print(f'accuracy = {(matriz_confusion_svmli[0][0]+matriz_confusion_svmli[1][1])/2}')

plt.figure()
sns.heatmap(matriz_confusion_svmli,annot=True)
plt.title('Matriz de confusión SVM(lineal) selección de características por filtro')
plt.show()
```

Donde las 4 mejores características según este método son ['preg', 'plas', 'mass', 'age'], al utilizar estas características para el entrenamiento este demora aproximadamente $0.005[s]$. La matriz desplegada por el código es la siguiente:

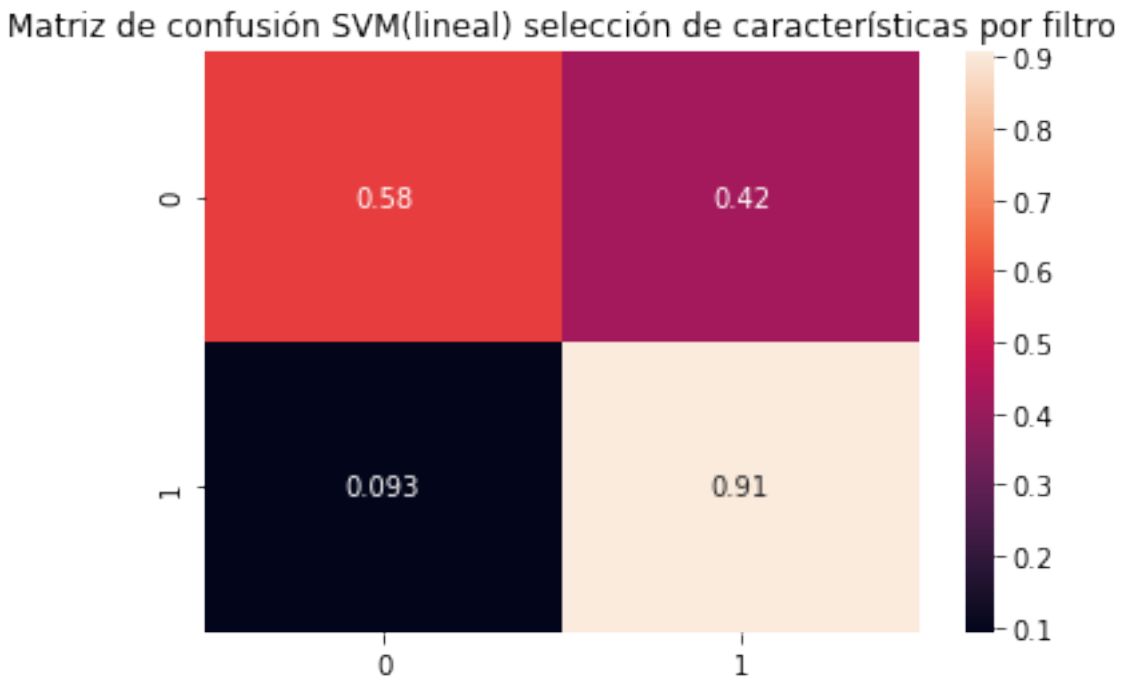

Figura 3: Matriz de confusión svm lineal, 4 caracterísrticas seleccionadas con filtrado

De la cual su accuracy es de 0.74 aproximadamente.

2.1.10. j)

Se debe repetir la estrategia de filtrado, pero esta vez sólo sobre las 2 mejores características, lo cual es ejecutado como sigue:

Código 9: Filtro 2 características SVM lineal

```
1
2  from sklearn.feature_selection import SelectKBest
3  seleccionj = SelectKBest(k=2).fit_transform(scaled_entrenamiento, entrenamiento.iloc[:,8])
4  caract= ['preg','plas','pres','skin','insu','mass','pedi','age']
5  L=[]
6  for i in range(len(caract)): ##No pude obtnerlo mediante get_param asi que se improviso con una
       ↪ revision por fuerza bruta
7      if scaled_entrenamiento[0][i] in seleccionj[0] :
8          L.append(caract[i])
9      else: None
10 print(f"Las mejores 2 características son: {L}")
11
12
13 ##########################################################
14 #g
15 car1j= scaled_entrenamiento[:,1]
16 car2j= scaled_entrenamiento[:,5]
```

```
carsj= np.array([car1j,car2j])
tiempo_svmlinealj_inicio = time.time()
svmlj = svm.SVC(kernel= 'linear',C=0.1, probability=False).fit(carsj.transpose(), entrenamiento.iloc
    ↪ [:,8])
tiempo_svmlinealj_final = time.time()
print(f"El tiempo de entrenamiento es {tiempo_svmlinealj_final - tiempo_svmlinealj_inicio} [s]")
##########################################
#h
carv1j= scaled_validacion[:,1]
carv2j= scaled_validacion[:,5]
carsvj= np.array([carv1j,carv2j])
y_pred_svmlj = svmlj.predict(carsvj.transpose()) #se concatenan las filas traspuestas para que sean
    ↪ columnas
matriz_confusion_svmlj = sklearn.metrics.confusion_matrix(validacion.iloc[:,8], y_pred_svmlj,
    ↪ normalize='true')
print(f'accuracy = {(matriz_confusion_svmlj[0][0]+matriz_confusion_svmlj[1][1])/2}')

plt.figure()
sns.heatmap(matriz_confusion_svmlj, annot= True)
plt.title('Matriz de confusión SVM(lineal) selección de 2 características por filtro')
plt.show()
```

Donde las 2 mejores características son ['plas', 'mass'], el tiempo de entrenamiento del SVM lineal ronda los $0.003[s]$, y la matriz de confusión que se imprime es la siguiente:

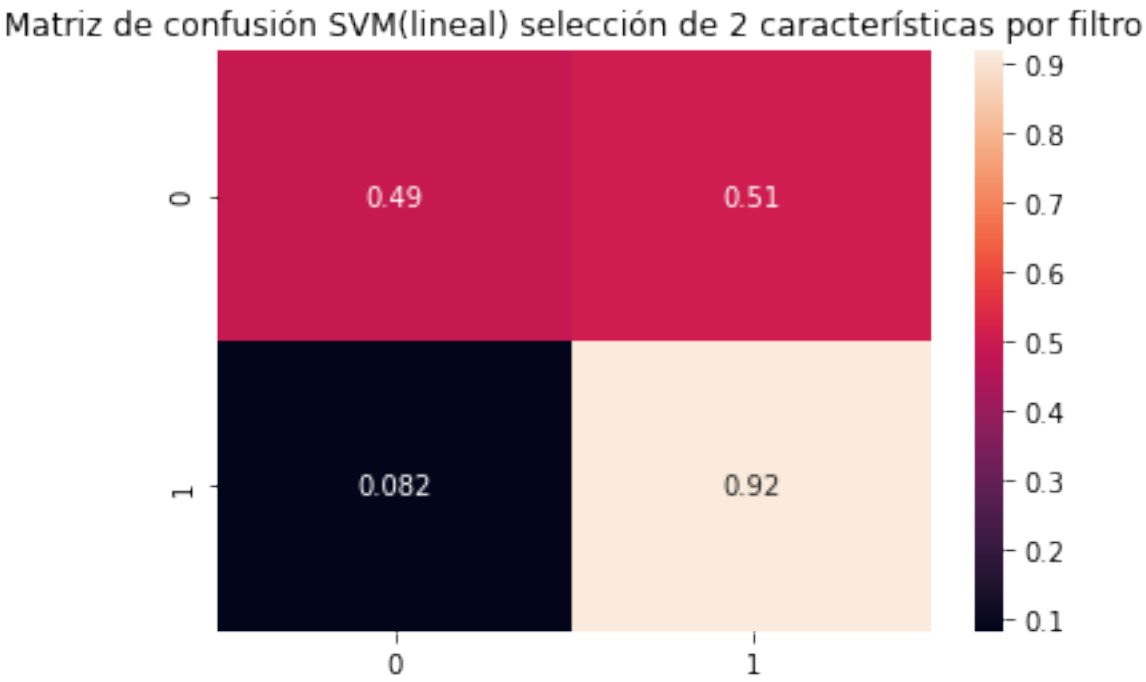

Figura 4: Matriz de confusión svm lineal, 2 caracterísrticas seleccionadas con filtrado

Obteniendo de accuracy 0.70 aprox.

2.1.11. k)

Se debe crear un nuevo clasificador inicial, esta vez Random Forest, al cual se le debe ajustar una grilla de la profundidad de sus árboles, utilizando el conjunto de validación para ver la calidad de los parámetros, posteriormente se debe desplegar su matriz de confusión ,accuracy y tiempo de entrenamiento.
Para realizar esta tarea se utiliza el código:

Código 10: Random Forest inicializador

```
1
2  from sklearn.ensemble import RandomForestClassifier
3  from sklearn.model_selection import GridSearchCV
4  from sklearn.model_selection import PredefinedSplit
5  ##########################
6  #Parte d
7  test_fold = [-1 for _ in range(len(validacion))] + [0 for _ in range(len(entrenamiento))]
8  y = np.concatenate([scaled_validacion, scaled_entrenamiento])
9  ps = PredefinedSplit(test_fold)
10
11 parametros= {'n_estimators' : [50, 100, 150, 200, 250]}
12 rf = RandomForestClassifier(max_depth=3, random_state=0)
13 rfk = GridSearchCV(rf, parametros, cv=ps)
14
15 scaled_concat = np.concatenate((scaled_entrenamiento,scaled_validacion))
16 conjuntos_concat= pd.concat([entrenamiento,validacion])
17 conjuntos_concat= conjuntos_concat.reset_index(drop= True)
18
19 tiempo_rfk_inicio = time.time()
20 rfk.fit( scaled_concat , conjuntos_concat.iloc[:,8])#entrenamiento
21
22 tiempo_rfk_final = time.time()
23
24 print(rfk.best_params_)
25 y_pred_rfk= rfk.predict(scaled_concat)
26 print(f"Tiempo de entrenamiento: {tiempo_rfk_final - tiempo_rfk_inicio} [s]")
27
28 ##########################
29 #Parte e
30 matriz_confusion_rfk = sklearn.metrics.confusion_matrix(conjuntos_concat.iloc[:,8], y_pred_rfk,
       ↪ normalize='true')
31 print(f'accuracy = {(matriz_confusion_rfk[0][0]+matriz_confusion_rfk[1][1])/2}')
32
33 plt.figure()
34 sns.heatmap(matriz_confusion_rfk, annot= True)
35 plt.title('Matriz de confusión Random Forest preliminar')
36 plt.show()
```

Del cual se obtiene como mejor parámetro una profundidad de 200, el tiempo de entrenamiento es de 1.29$[s]$ aprox, su matriz de confusión es la siguiente:

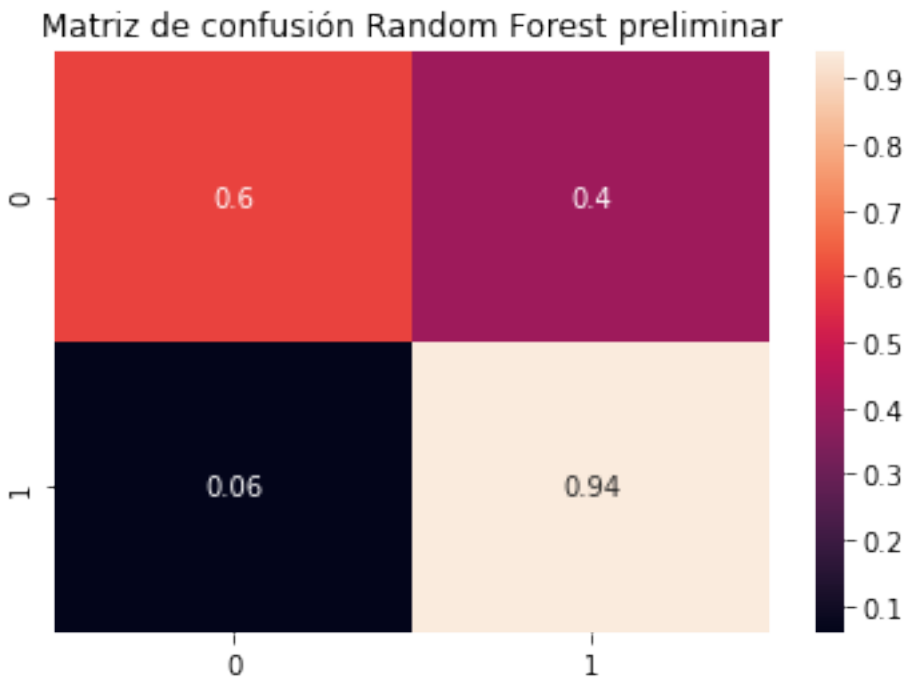

Figura 5: Matriz de confusión Random Forest, clasificador inicial con heatmap

Donde su accuracy es de 0.77 aprox.

2.1.12. l)

Utilizando la técnica de wrapper para reducir características se debe implementar un clasificador Random Forest y obetener información relevante como su tiempo de entrenamiento y accuracy.

Código 11: Wrapper Random Forest

```
1
2  from sklearn.feature_selection import SelectFromModel
3  seleccionl = SelectFromModel(estimator=RandomForestClassifier(max_depth=3, random_state=0,
       ↪ n_estimators=200)).fit(scaled_entrenamiento, entrenamiento.iloc[:,8])
4
5
6  caract= ['preg','plas','pres','skin','insu','mass','pedi','age']
7  seleccionl.get_support()
8  Ll=[]
9  vall = []
10 for i in range(len(caract)):
11     if seleccion.get_support()[i] == True:
12         Ll.append(caract[i])
13         vall.append(seleccion.estimator_.coef_[0][i])
14     else: None
15 print(f"Las mejores características son: {Ll}") #index 1 y 5
16
```

```
17 ########################################
18 #Parte g
19 car1l= scaled_entrenamiento[:,1]
20 car2l= scaled_entrenamiento[:,5]
21 carsl= np.array([car1l,car2l])
22 tiempo_rfl_inicio = time.time()
23 rfl = RandomForestClassifier(max_depth=3, random_state=0,n_estimators=200).fit(cars.transpose
       ↪ (), entrenamiento.iloc[:,8])
24 tiempo_rfl_final = time.time()
25 print(f"El tiempo de entrenamiento es {tiempo_rfl_final - tiempo_rfl_inicio} [s]")
26
27 ########################################
28 #Parte h
29 carv1l= scaled_validacion[:,1]
30 carv2l= scaled_validacion[:,5]
31 carsvl= np.array([carv1l,carv2l])
32 y_pred_rfl = rfl.predict(carsvl.transpose()) #se concatenan las filas traspuestas para que sean
       ↪ columnas
33 matriz_confusion_rfl = sklearn.metrics.confusion_matrix(validacion.iloc[:,8], y_pred_rfl, normalize=
       ↪ 'true')
34 print(f'accuracy = {(matriz_confusion_rfl[0][0]+matriz_confusion_rfl[1][1])/2}')
35
36 plt.figure()
37 sns.heatmap(matriz_confusion_rfl, annot= True)
38 plt.title('Matriz de confusión Random Forest selección de características wrapper')
39 plt.show()
```

El código despliega como características importantes ['plas', 'mass'], el tiempo de entrenamiento para el clasificador es de $0.3[s]$ aprox, la matriz de confusión es la siguiente:

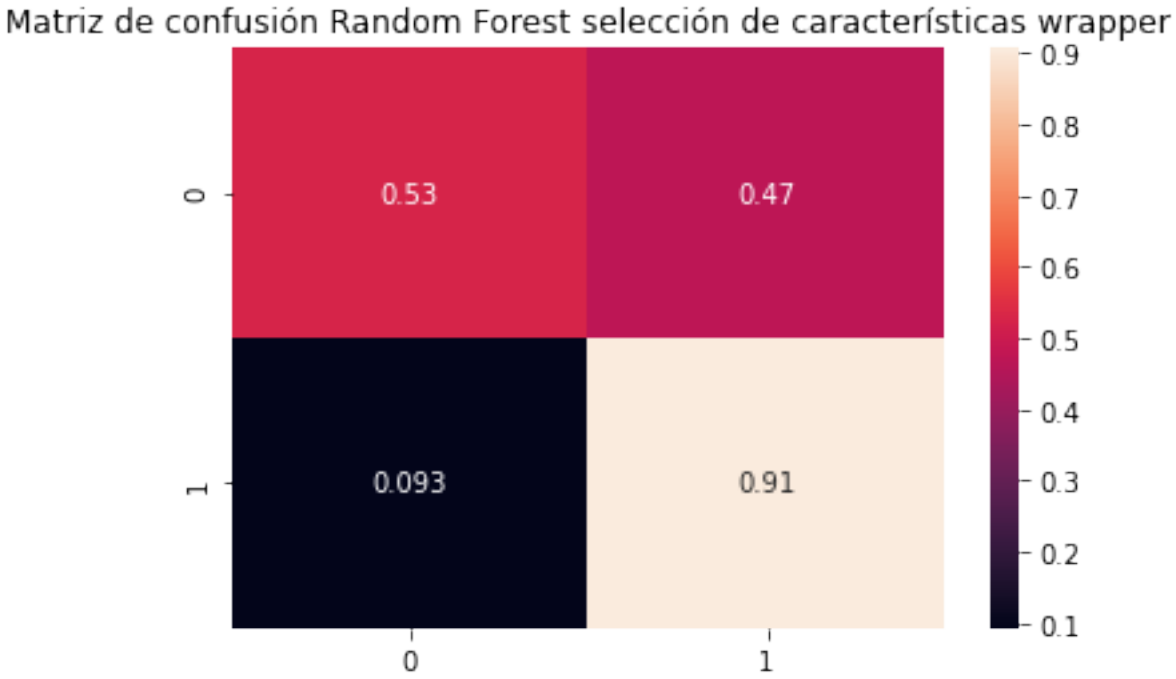

Figura 6: Matriz de confusión para Random Forest con reducción de característica mediante wrapper

Donde su accuracy es de 0.72 aprox.

2.1.13. m)

Se debe repetir la utilización del clasificador Random Forest, escogiendo las 4 mejores características mediante filtrado, lo cual se implementa como sigue:

Código 12: Filtro 4 características Random Forest

```
1
2  from sklearn.feature_selection import SelectKBest
3  seleccionm = SelectKBest(k=4).fit_transform(scaled_entrenamiento, entrenamiento.iloc[:,8])
4  caract= ['preg','plas','pres','skin','insu','mass','pedi','age']
5  L=[]
6  for i in range(len(caract)): ##No pude obtnerlo mediante get_param asi que se improviso con una
       ↪ revision por fuerza bruta
7      if scaled_entrenamiento[0][i] in seleccionm[0] :
8          L.append(caract[i])
9          print(i)
10     else: None
11 print(f"Las mejores 4 características son: {L}")
12 ##########################################################
13 #g
14 car1m= scaled_entrenamiento[:,0]
15 car2m= scaled_entrenamiento[:,1]
```

```
car3m= scaled_entrenamiento[:,5]
car4m= scaled_entrenamiento[:,7]
carsm= np.array([car1m,car2m,car3m,car4m])
tiempo_rfm_inicio = time.time()
rfm = RandomForestClassifier(max_depth=3, random_state=0,n_estimators=200).fit(carsm.
    ↪ transpose(), entrenamiento.iloc[:,8])
tiempo_rfm_final = time.time()
print(f"El tiempo de entrenamiento es {tiempo_rfm_final - tiempo_rfm_inicio} [s]")
##########################################
#h
carv1m= scaled_validacion[:,0]
carv2m= scaled_validacion[:,1]
carv3m= scaled_validacion[:,5]
carv4m= scaled_validacion[:,7]
carsvm= np.array([carv1m,carv2m,carv3m,carv4m])
y_pred_rfm = rfm.predict(carsvm.transpose()) #se concatenan las filas traspuestas para que sean
    ↪ columnas
matriz_confusion_rfm = sklearn.metrics.confusion_matrix(validacion.iloc[:,8], y_pred_rfm,
    ↪ normalize='true')
print(f'accuracy = {(matriz_confusion_rfm[0][0]+matriz_confusion_rfm[1][1])/2}')

plt.figure()
sns.heatmap(matriz_confusion_rfm, annot= True)
plt.title('Matriz de confusión Random Forest selección de 4 mejores características por filtro')
plt.show()
```

Una vez ejecutado el código las 4 características más importantes por metodo de filtro son ['preg', 'plas', 'mass', 'age'], las que al ser utilizadas para entrenar al clasificador demoran alrededor de $0.28[s]$, este clasificador posee la siguiente matriz de confusión:

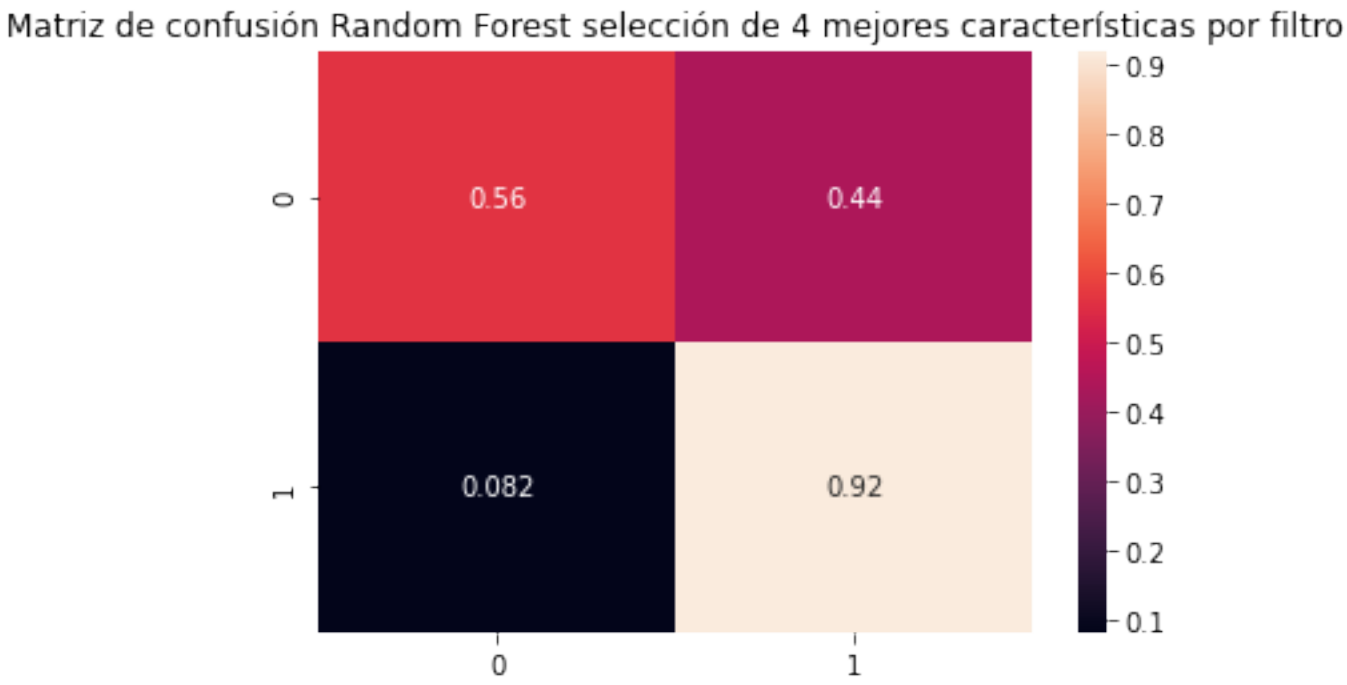

Figura 7: Matriz de confusión para clasificador Random Forest escogiendo las 4 mejores caracteristicas por filtrado

Siendo su accuracy 0.74.

2.1.14. n)

Ahora se realiza exactamente lo mismo que en la parte anterior con la salvedad de utilizar las 2 mejores características.

Código 13: Filtro 2 características Random Forest

```
1
2  from sklearn.feature_selection import SelectKBest
3  seleccionn = SelectKBest(k=2).fit_transform(scaled_entrenamiento, entrenamiento.iloc[:,8])
4  caract= ['preg','plas','pres','skin','insu','mass','pedi','age']
5  L=[]
6  for i in range(len(caract)): ##No pude obtnerlo mediante get_param asi que se improviso con una
       ↪ revision por fuerza bruta
7      if scaled_entrenamiento[0][i] in seleccionn[0] :
8          L.append(caract[i])
9      else: None
10 print(f"Las mejores 2 características son: {L}")
11
12
13 ##########################################################
14 #g
15 car1n= scaled_entrenamiento[:,1]
16 car2n= scaled_entrenamiento[:,5]
```

```
carsn= np.array([car1n,car2n])
tiempo_rfn_inicio = time.time()
rfn = RandomForestClassifier(max_depth=3, random_state=0,n_estimators=200).fit(carsn.
    ↪ transpose(), entrenamiento.iloc[:,8])
tiempo_rfn_final = time.time()
print(f"El tiempo de entrenamiento es {tiempo_rfn_final - tiempo_rfn_inicio} [s]")
########################################
#h
carv1n= scaled_validacion[:,1]
carv2n= scaled_validacion[:,5]
carsvn= np.array([carv1n,carv2n])
y_pred_rfn = rfn.predict(carsvn.transpose()) #se concatenan las filas traspuestas para que sean
    ↪ columnas
matriz_confusion_rfn = sklearn.metrics.confusion_matrix(validacion.iloc[:,8], y_pred_rfn, normalize
    ↪ ='true')
print(f'accuracy = {(matriz_confusion_rfn[0][0]+matriz_confusion_rfn[1][1])/2}')

plt.figure()
sns.heatmap(matriz_confusion_rfn, annot= True)
plt.title('Matriz de confusión Random Forest, selección de 2 características por filtro')
plt.show()
```

Las 2 mejores características son ['plas', 'mass'], el tiempo de entrenamiento es de 0.27$[s]$ aprox y su matriz de consufión:

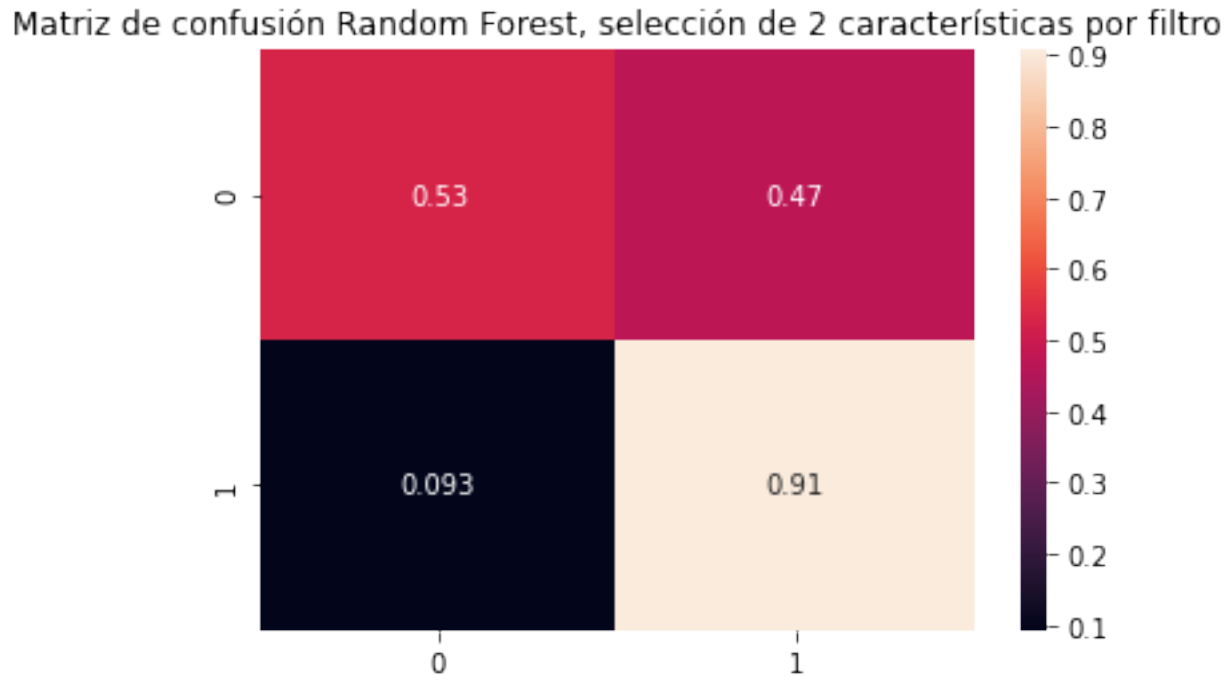

Figura 8: Matriz de confusión Random Forest 2 mejores características por filtrado

Obteniendo un accuracy de 0.72

2.1.15. o)

Una vez obtenidos todos los valores, al comparar los SVM lineales, el con mayor accuracy y por ende el mejor SVM lineal es el utilizado en parte i), ese decir, el de mejores 4 características mediante filtrado, con un accuracy de 0.79.De igual manera, al realizar la comparación con los Random Forest, el mejor clasificador está dado por la implementación de parte m), es decir, mediante filtrado con las 4 mejores características, con un accuracy de 0.79. Una vez vislumbrados los mejores clasificadores considerando como criterio su accuracy se procede a evaluar en conjunto de pruebas, este paso se realiza mediante el código:

Código 14: Evaluación mejor SVM lineal y Random Forest a conjunto de pruebas

```
1
2  cart1= scaled_test[:,0]
3  cart2= scaled_test[:,1]
4  cart3= scaled_test[:,5]
5  cart4= scaled_test[:,7]
6  carst= np.array([cart1,cart2,cart3,cart4])
7
8  y_pred_svmlo = svmli.predict(carst.transpose()) #se concatenan las filas traspuestas para que sean
       ↪ columnas
9  y_pred_rfo = rfm.predict(carst.transpose())
10
11 matriz_confusion_svmlo = sklearn.metrics.confusion_matrix(test.iloc[:,8], y_pred_svmlo, normalize
       ↪ ='true')
12 matriz_confusion_rfo = sklearn.metrics.confusion_matrix(test.iloc[:,8], y_pred_rfo, normalize='true
       ↪ ')
13
14
15 print(f'accuracy svm lineal = {(matriz_confusion_svmlo[0][0]+matriz_confusion_svmlo[1][1])/2}')
16 print(f'accuracy random forest = {(matriz_confusion_rfo[0][0]+matriz_confusion_rfo[1][1])/2}')
17
18 plt.figure()
19 sns.heatmap(matriz_confusion_svmlo, annot= True)
20 plt.title('Matriz de confusión SVM(lineal) a test selección de 4 mejores características por filtro')
21 plt.figure()
22 sns.heatmap(matriz_confusion_rfo, annot= True)
23 plt.title('Matriz de confusión Random Forest a test selección de 4 mejores características por filtro')
24 plt.show()
```

Del cual se obtienen las matrices de confusion:

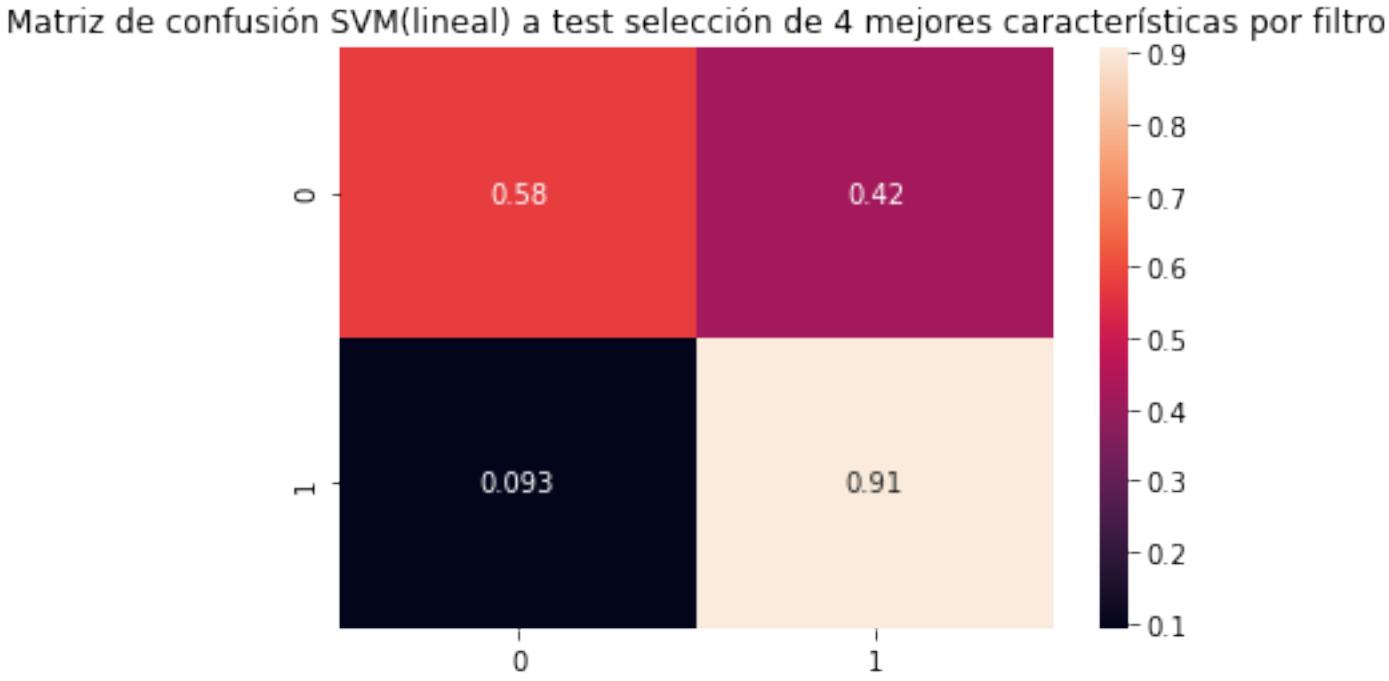

Figura 9: Matriz de confusión SVM lineal, de 4 características a test

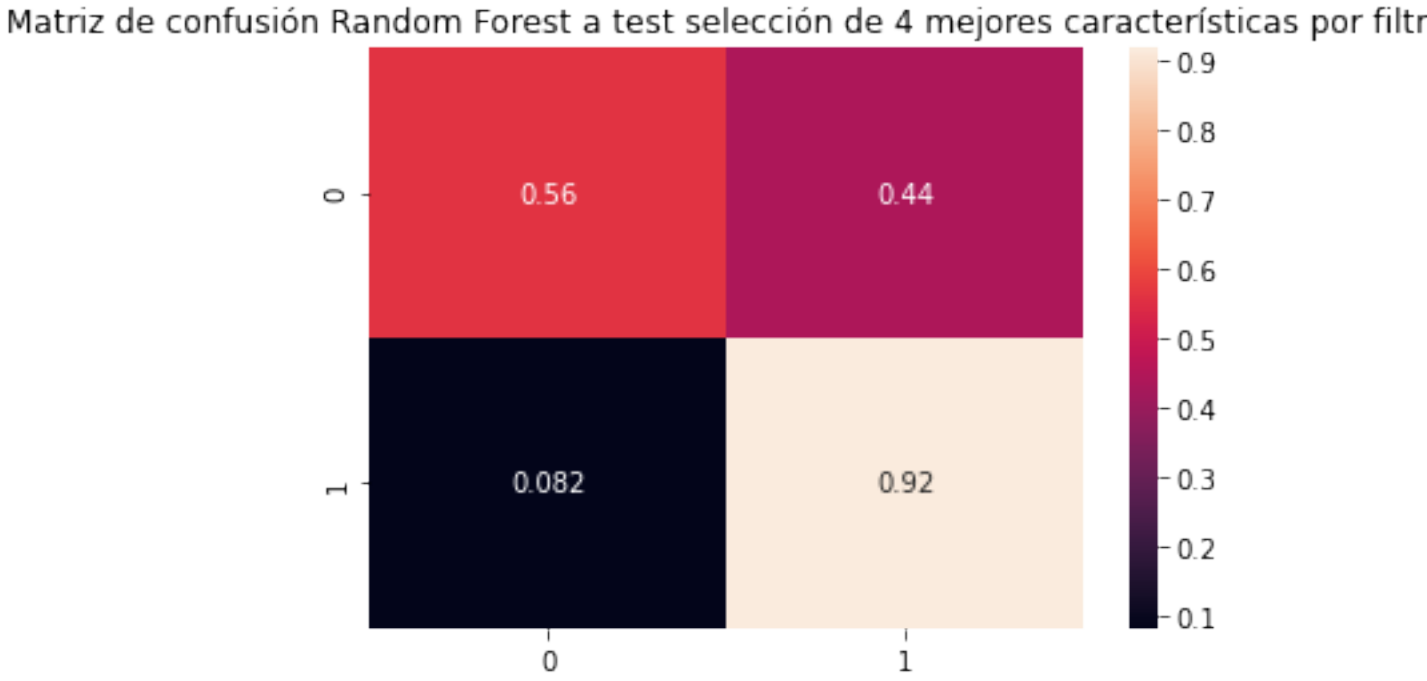

Figura 10: Matriz de confusión Random Forest de 4 características a test

Donde la primera tiene un accuracy de 0.743 y la segunda uno de 0.739.

3. Análisis

3.1. a)

Al comparar los resultados mediante su eficacia, la cual está medida de forma cuantitativa con el valor de accuracies, en cada caso análogo el mejor desempeño fue realizado por el clasificador Random Forest, esto podría deberse a su funcionamiento interno, debido a que posee una gran cantidad de clasificadores dentro de su estructura, esto sumado a la utilización de Bagging genera aleatoreidad, lo cual lo hace poseer menos sobreajuste que un solo clasificador.

3.2. b)

Al reducir las características el entrenamiento se hace de forma más rápida, pero existe un límite, ya que en ambos casos se hace evidente que una menor cantidad de caracterísitcas generan un sobreajuste o preferencia a una clase, por lo cual el calibrar de manera correct se hace indispensable.

3.3. c)

Debido a que en este caso la reducción de características compromete muy poco la fidelidad de los resultados comparado con el clasificador que inicializa, el cual podemos tomar como referencia, la reducción de caracteristicas aportaría enormemente en esta aplicación, debido al dinero y tiempo ahorrados.Lo cual se puede extrapolar con otras aplicaciones donde los resultados no se vean enormemente comprometidos.

4. Conclusiones

Al realizar la experiencia se obtiene un acercamiento mayor a la utilización de estrategias de evaluación para características y cómo estas se implementan en código, por otra parte los resultados a medida que las características van despareciendo dejan en evidencia que el abuso de estas estrategias pueden empeorar el rendimiento de los clasificadores.
Debido a la nula experiencia en este tema el entendimiento de librerías se hace complicado.
Los resultados que se obtienen concuerdan con lo esperado teóricamente, aunque cabe destacar que dependiendo del caso la diferencia puede hacerse más o menos notoria, ya que esta estrategia tiene ligada la idea de caracterísiticas representativas de una clase. En este caso la mejor solución sin considerar el clasificador era la utilización de las mejores 4 características, siendo el mejor clasificador en esta tarea el Random Forest.
Para mejorar el desempeño de las estrategias se podrían generar mayor numero de permutaciones utilizando el mismo clasificador, es decir, para 1 a n características y evaluar su desempeño y de igual forma utilizar otros clasificadores.