Tarea 4 Final

July 15, 2023

Tarea 4 Diego Valdebenito LAB-MAA

Instrucciones

Los resultados de los ejercicios propuestos se deben entregar como un notebook por correo electronico a *juancaros@udec.cl* el dia 30/6 hasta las 21:00. Es importante considerar que el código debe poder ejecutarse en cualquier computadora con la data original del repositorio. Recordar la convencion para el nombre de archivo ademas de incluir en su documento titulos y encabezados por seccion. Utilizar la base de datos segun indicado en las preguntas.

Preguntas:

1. Utilizando el set de datos *junaeb2.csv* realice una regresion para predecir la variable *imce* con regularizacion via Lasso con cross-validation. Muestre que sus resultados son robustos a la seleccion de hiperparametros y compute una metrica de calidad de ajuste del modelo.
2. Utilizando el set de datos *charls2.csv* realice una clasificacion de la variable *retired* usando Random Forest sobre las demas variables del dataset con cross-validation. Muestre que sus resultados son sensibles a la seleccion de hiperparametros y compute una metrica de calidad de ajuste del modelo.
3. Repita el analisis de la Pregunta 2 usando Stacking, con tres modelos (Random Forest, Gradient Boosting y SVM). Muestre que sus resultados son robustos a la seleccion de hiperparametros y compute una metrica de calidad de ajuste del modelo.
4. Utilizando la base de datos *enia.csv* realice un analisis de cluster usando k-means incluyendo todas las variables excepto *tamano* y *ID*. Muestre que sus resultados son sensibles a la seleccion de hiperparametros y compute una metrica de calidad de ajuste del modelo.
5. Repita el analisis de la Pregunta 4 usando DBSCAN. Muestre que sus resultados son sensibles a la seleccion de hiperparametros y compute una metrica de calidad de ajuste del modelo.

[4]:
```
import numpy as np
import pandas as pd
import seaborn as sns
import eli5
from matplotlib import pyplot as plt
from numpy import mean
from numpy import std
from numpy import absolute

from sklearn.ensemble import RandomForestClassifier
```

```
from sklearn.metrics import accuracy_score
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import RepeatedKFold
from sklearn.linear_model import Lasso
from sklearn.linear_model import LassoCV
from sklearn.metrics import mean_squared_error
from sklearn.metrics import confusion_matrix
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.cluster import KMeans
from sklearn.cluster import DBSCAN
from sklearn.ensemble import GradientBoostingRegressor
from sklearn.model_selection import GridSearchCV
from sklearn.ensemble import StackingClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.pipeline import make_pipeline
from sklearn.svm import LinearSVC
from pywaffle import Waffle

import yellowbrick
from sklearn.cluster import KMeans, DBSCAN, AgglomerativeClustering
from sklearn.metrics import davies_bouldin_score, silhouette_score,␣
 ↪calinski_harabasz_score
from yellowbrick.cluster import KElbowVisualizer, SilhouetteVisualizer
from yellowbrick.style import set_palette
from yellowbrick.contrib.wrapper import wrap

%matplotlib inline
```

0.0.1 Pregunta 1

0.1 Lasso Regression (Regularization)

[5]:
```
df1=pd.read_csv('C:/Users/equipo/Downloads/Tarea 4/LAB-MAA-main/data/junaeb2.
 ↪csv')
df1 = df1[df1.vive_madre < 2]
df1 = df1[df1.vive_padre < 2]
df1 = df1[df1['madre_work'] != -1]
df1 = df1.dropna()
df1=df1.astype('int64')
df1.reset_index()
```

[5]:

	index	sexo	edad	imce	vive_padre	vive_madre	sk1	sk2	sk3	sk4	\
0	1	0	76	0	0	1	1	1	1	1	
1	4	0	86	1	1	1	1	1	1	1	
2	6	1	91	2	1	1	1	1	1	2	

```
3          7     1    92     1           1           1    1    3    1    1
4          8     1    69     1           1           1    1    2    3    2
…        …   …    …    …           …         …  …  …  …  …
23924  41848     1    79     1           1           0    1    1    2    1
23925  41850     1    79     2           1           1    1    1    2    2
23926  41851     0    78     2           1           1    1    1    1    1
23927  41852     1    78     0           1           1    1    1    1    1
23928  41853     0    78     0           0           1    2    2    1    1

       …  sk9  sk10  sk11  sk12  sk13  act_fisica  area  educm  educp  \
0      …    1     1     1     1     1           5     0      8      8
1      …    1     1     1     1     1           1     1     17     15
2      …    3     3     3     2     2           2     1     20     19
3      …    2     4     3     2     4           1     1     13     11
4      …    3     4     2     2     3           2     0     10      9
…    …  …   …   …   …   …   …         …   …     …     …
23924  …    1     1     2     1     2           2     1     17     17
23925  …    1     3     2     1     4           3     1     18     19
23926  …    1     3     1     1     1           3     1     13      9
23927  …    1     2     1     1     2           2     1     17     15
23928  …    1     1     1     1     1           1     1     18     11

       madre_work
0               1
1               0
2               1
3               0
4               1
…             …
23924           1
23925           0
23926           1
23927           1
23928           1

[23929 rows x 24 columns]
```

[7]:
```
target = df1.imce
features = df1.drop('imce', axis=1)
features.describe()
```

[7]:
```
              sexo          edad    vive_padre    vive_madre           sk1  \
count  23929.000000  23929.000000  23929.000000  23929.000000  23929.000000
mean       0.547286     82.907560      0.646997      0.978520      1.108738
std        0.497769      3.796739      0.477914      0.144982      0.375010
min        0.000000     62.000000      0.000000      0.000000      1.000000
25%        0.000000     81.000000      0.000000      1.000000      1.000000
```

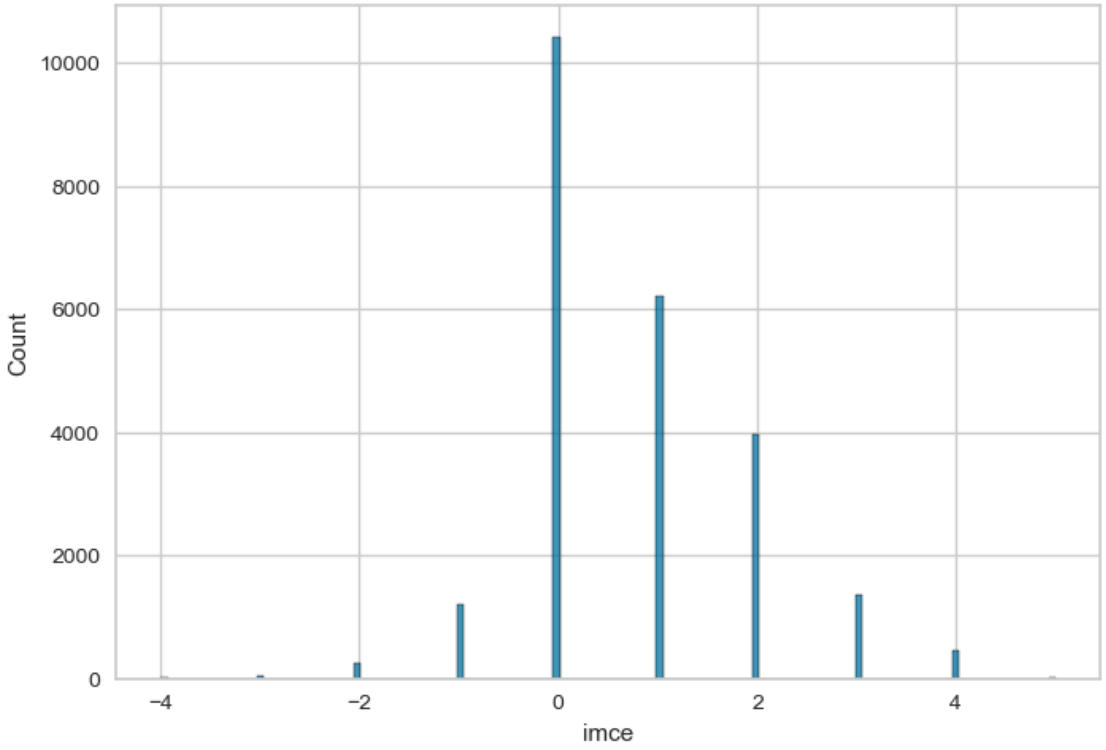
```
50%        1.000000     82.000000      1.000000      1.000000      1.000000
75%        1.000000     83.000000      1.000000      1.000000      1.000000
max        1.000000    107.000000      1.000000      1.000000      5.000000

                sk2           sk3           sk4           sk5           sk6  \
count  23929.000000  23929.000000  23929.000000  23929.000000  23929.000000
mean       1.382381      1.252288      1.245936      1.265953      1.481926
std        0.636408      0.564464      0.558799      0.555792      0.723248
min        1.000000      1.000000      1.000000      1.000000      1.000000
25%        1.000000      1.000000      1.000000      1.000000      1.000000
50%        1.000000      1.000000      1.000000      1.000000      1.000000
75%        2.000000      1.000000      1.000000      1.000000      2.000000
max        5.000000      5.000000      5.000000      5.000000      5.000000

       …           sk9           sk10          sk11          sk12  \
count  …  23929.000000  23929.000000  23929.000000  23929.000000
mean   …      1.321367      1.858540      1.376531      1.499854
std    …      0.641297      0.932845      0.652371      0.786840
min    …      1.000000      1.000000      1.000000      1.000000
25%    …      1.000000      1.000000      1.000000      1.000000
50%    …      1.000000      2.000000      1.000000      1.000000
75%    …      1.000000      2.000000      2.000000      2.000000
max    …      5.000000      5.000000      5.000000      5.000000

               sk13    act_fisica          area         educm         educp  \
count  23929.000000  23929.000000  23929.000000  23929.000000  23929.000000
mean       1.675624      2.537632      0.936061     13.735175     13.415730
std        0.968498      1.042565      0.244650      3.341749      3.395748
min        1.000000      1.000000      0.000000      0.000000      0.000000
25%        1.000000      2.000000      1.000000     13.000000     12.000000
50%        1.000000      2.000000      1.000000     13.000000     13.000000
75%        2.000000      3.000000      1.000000     16.000000     15.000000
max        5.000000      5.000000      1.000000     22.000000     22.000000

         madre_work
count  23929.000000
mean       0.827531
std        0.377795
min        0.000000
25%        1.000000
50%        1.000000
75%        1.000000
max        1.000000

[8 rows x 22 columns]
```

En las variables vive_madre y vive_padre existen datos con valor 2, lo que no tiene sentido ya que esas variables solo puede valer 0 o 1 por lo que se eliminan esos datos, y luego se eliminaron

los datos donde “madre_work” toma el valor de -1, ya que es una variable binaria. Además, se eliminaron los datos en donde las celdas de las variables “act_fisica” y “educm” esten vacias.

[30]:
```
sns.histplot(data=target)
```

[30]:
```
<AxesSubplot:xlabel='imce', ylabel='Count'>
```



Se puede observar que en la mayoria de las observaciones hay un valor medio del indice de masa corporal estandarizado “imce”. En general, se aprecia un sesgo negativo en los datos.

[9]:
```
# Split the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(features, target,␣
 ↪test_size=0.2, random_state=42)

# Standardize the features
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
```

Se estandarizaron los datos para poder ocupar correctamente el modelo de regresion Lasso.

[10]:
```
# Train the Lasso regression model
lasso=Lasso(alpha=0.0009,
            max_iter=10000,
```

```
            tol=0.0001)  # You can adjust the alpha value to control the␣
 ↪regularization strength
lasso.fit(X_train_scaled, y_train)
```

[10]:
```
Lasso(alpha=0.0009, max_iter=10000)
```

[31]:
```
# Predict on the test set
y_pred = lasso.predict(X_test_scaled)

# Calculate the mean squared error
mse = mean_squared_error(y_test, y_pred)
rmse = np.sqrt(mse)
print("Root Mean Squared Error:", rmse)
```

```
Root Mean Squared Error: 1.153635008280782
```

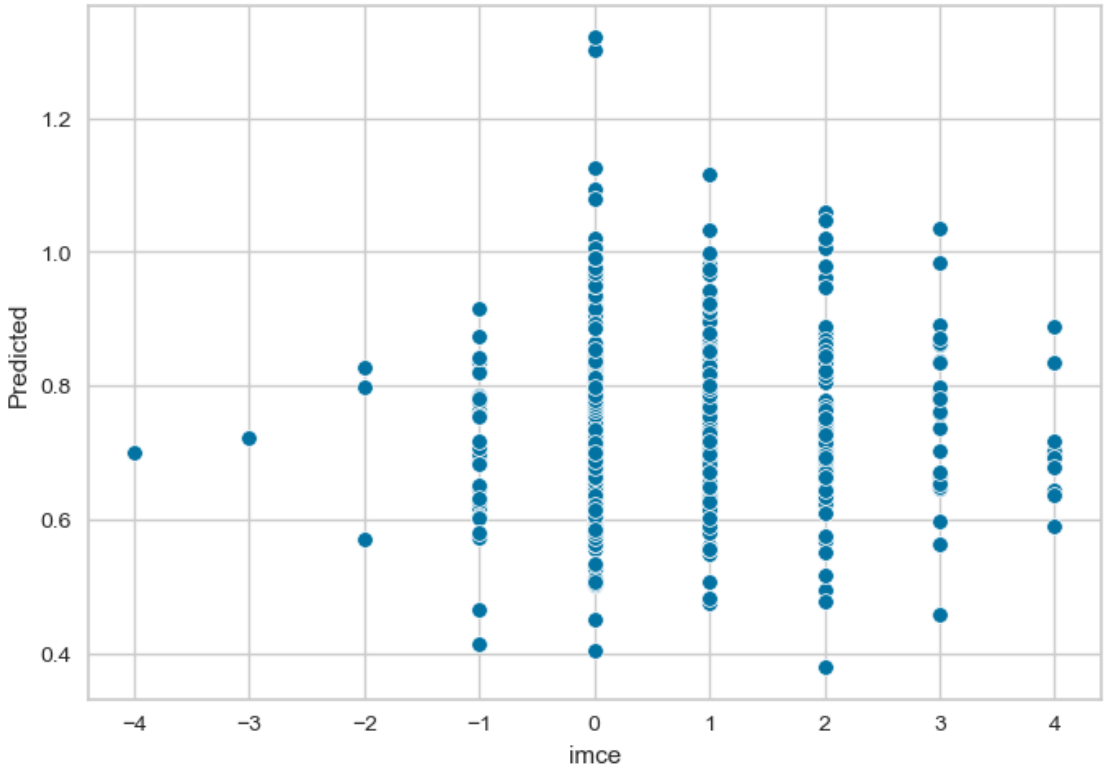
[32]:
```
# define model evaluation method
cv = RepeatedKFold(n_splits=100, n_repeats=3, random_state=1)
# cross validation scores
scores = cross_val_score(lasso, X_train_scaled, y_train, cv=cv, n_jobs=-1,␣
 ↪scoring='neg_root_mean_squared_error')
scores=absolute(scores)
print('Mean RMSE: %.3f (%.3f)' % (mean(scores), std(scores)))
```

```
Mean RMSE: 1.130 (0.068)
```

Los valores del RMSE y MRMSE indican que el modelo LASSO tiene un buen rendimiento en términos de precisión de predicción y que es estable en diferentes muestras de datos. En este caso, se puede tener confianza en las predicciones del modelo y considerarlo como un buen ajuste para los datos.

[13]:
```
y_pred=pd.DataFrame(y_pred)
y_pred.rename(columns={0 :'Predicted'}, inplace=True )
test = pd.concat([y_test, y_pred], axis=1, join='inner')
sns.scatterplot(data=test, x='imce', y='Predicted')
```

[13]:
```
<AxesSubplot:xlabel='imce', ylabel='Predicted'>
```



Se observa una prediccion bastante acertada, dada las caracteristicas del target de la base de datos.

Feature importance

[14]:
```
eli5.show_weights(lasso, top=-1, feature_names = X_train.columns.tolist())
```

[14]:
```
<IPython.core.display.HTML object>
```

Se observa que las variables más importantes son “sexo”, “sk5” (frecuencia categórica de si el estudiante juega con otros), “sk8” (frecuencia categórica de si el estudiante participa en juegos grupales), “sk7” (si el estudiante es agresivo) y “sk12” (Frecuencia categórica de si el estudiante juega a armar y desarmar cosas). Por ejemplo, la función objetivo o medida de desempeño mejora en 0.065 cuando se incluye la variable “sexo en el modelo”.

Tuning alpha

[15]:
```
# define model
model = LassoCV(n_alphas=100, cv=cv, n_jobs=-1, max_iter=10000)
# fit model
model.fit(X_train_scaled, y_train)
# summarize chosen configuration
print('alpha: %f' % model.alpha_)
```

```
alpha: 0.000967
```

A continuación, se volverá a aplicar el modelo para ver cómo reaccionan las medidas de precisión de este al cambiar el parámetro alpha.

[33]:
```
# Train the Lasso regression model
lasso2=Lasso(alpha=0.000967,
             max_iter=10000,
             tol=0.0001)  # You can adjust the alpha value to control the␣
 ↪regularization strength
lasso2.fit(X_train_scaled, y_train)
```

[33]:
```
Lasso(alpha=0.000967, max_iter=10000)
```

[34]:
```
# Predict on the test set
y_pred = lasso2.predict(X_test_scaled)

# Calculate the mean squared error
mse = mean_squared_error(y_test, y_pred)
rmse = np.sqrt(mse)
print("Root Mean Squared Error:", rmse)
```

```
Root Mean Squared Error: 1.153631593587831
```

[35]:
```
# define model evaluation method
cv = RepeatedKFold(n_splits=100, n_repeats=3, random_state=1)
# cross validation scores
scores = cross_val_score(lasso2, X_train_scaled, y_train, cv=cv, n_jobs=-1,␣
 ↪scoring='neg_root_mean_squared_error')
scores=absolute(scores)
print('Mean RMSE: %.3f (%.3f)' % (mean(scores), std(scores)))
```

```
Mean RMSE: 1.130 (0.068)
```

Se puede observar que al cambiar el hiperparámetro alpha los valores que indican la precisión del modelo son prácticamente idénticos al del modelo original, y ello se debe a que el alpha original es una buena aproximación al alpha óptimo.

0.1.1 Pregunta 2

0.2 Clasificación

0.2.1 Random Forest

Se ajustó drinkly como variable numérica y se transformó wealth a logaritmo, además de agregar una variable que indica cuando wealth no existe, dwealth. Además, se eliminaron los datos donde las filas de las variables “cesd” y “wealth” son vacías.

[24]:
```
df2=pd.read_csv('C:/Users/equipo/Downloads/Tarea 4/LAB-MAA-main/data/charls2.
 ↪csv')
df2 = df2.dropna(subset=['cesd', 'wealth'])
df2 = df2.replace({'.r': np.nan, '.m': np.nan, '.d': np.nan})
```

```
df2['drinkly'] = df2['drinkly'].astype(float)
df2['wealth']=df2['wealth']/100000
df2['dwealth']=0
df2.loc[df2['wealth'].isnull(), 'dwealth'] = 1
df2['lwealth']=np.log(df2['wealth']-df2['wealth'].min()+0.1)
df2.loc[df2['lwealth'].isnull(), 'lwealth'] = 0
df2.reset_index()
df2.describe()
```

[24]:

	age	cesd	child	drinkly	female
count	8080.000000	8080.000000	8080.000000	8080.000000	8080.000000
mean	58.164356	9.112376	2.780074	0.325990	0.535272
std	9.374956	6.481237	1.397316	0.468773	0.498785
min	21.000000	0.000000	0.000000	0.000000	0.000000
25%	51.000000	4.000000	2.000000	0.000000	0.000000
50%	57.000000	8.000000	3.000000	0.000000	1.000000
75%	64.000000	13.000000	4.000000	1.000000	1.000000
max	95.000000	30.000000	10.000000	1.000000	1.000000

	hrsusu	hsize	intmonth	married	retage
count	8080.000000	8080.000000	8080.000000	8080.000000	8080.000000
mean	2.566736	3.764233	7.506931	0.879084	1.480817
std	1.788298	1.823838	1.001893	0.326050	4.206412
min	0.000000	1.000000	1.000000	0.000000	0.000000
25%	0.000000	2.000000	7.000000	1.000000	0.000000
50%	3.496508	4.000000	7.000000	1.000000	0.000000
75%	4.025352	5.000000	8.000000	1.000000	0.000000
max	5.123964	16.000000	12.000000	1.000000	37.000000

	retin	retired	schadj	urban	wealth
count	8080.000000	8080.000000	8080.000000	8080.000000	8080.000000
mean	0.163861	0.184035	4.039851	0.212005	0.014795
std	0.370173	0.387536	3.545666	0.408754	0.434799
min	0.000000	0.000000	0.000000	0.000000	-10.000000
25%	0.000000	0.000000	0.000000	0.000000	0.000000
50%	0.000000	0.000000	4.000000	0.000000	0.004000
75%	0.000000	0.000000	8.000000	0.000000	0.022000
max	1.000000	1.000000	16.000000	1.000000	9.001000

	dwealth	lwealth
count	8080.0	8080.000000
mean	0.0	2.312239
std	0.0	0.084529
min	0.0	-2.302585
25%	0.0	2.312535
50%	0.0	2.312931
75%	0.0	2.314711

```
max        0.0     2.949741
```

[25]:
```
target = df2.retired
features = df2.drop(['retired', 'wealth'], axis=1)

# Split the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(features, target,␣
 ↪test_size=0.2, random_state=42)

# Standardize the features
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
```

Aquí se utiliza el 80% para entrenamiento 20% del tamaño de los datos para validación.

[78]:
```
rf = RandomForestClassifier(n_estimators=150,
                            random_state=42)

# Training the classifier
rf.fit(X_train_scaled, y_train)

# Making predictions on the test set
y_pred = rf.predict(X_test_scaled)

# Evaluating the accuracy of the classifier
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy:", accuracy)
```

```
Accuracy: 0.9090346534653465
```

Los resultados indican que el clasificador ha logrado predecir correctamente el 90.965% de las muestras en el conjunto de prueba (X_test), usando 150 árboles de decisión.

0.2.2 Matriz de confusión

[80]:
```
confusion_matrix(y_test, y_pred)
```

[80]:
```
array([[1223,   90],
       [  57,  246]], dtype=int64)
```

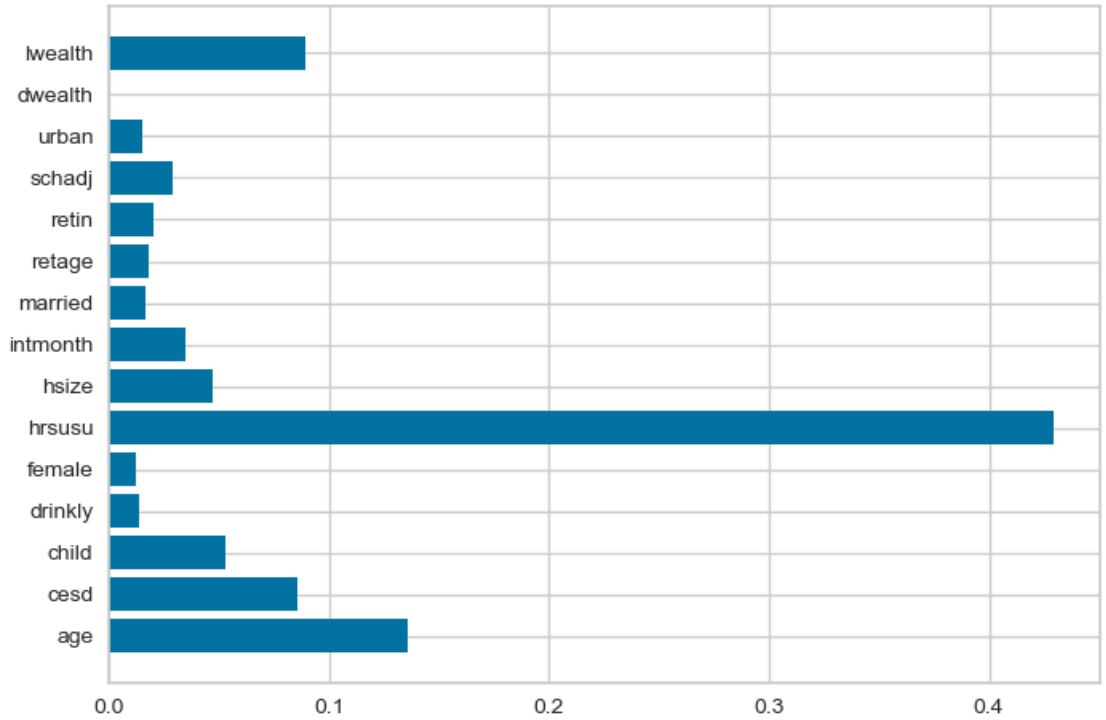
- Verdadero positivo (True Positive, TP): El número de muestras que realmente pertenecen a la clase positiva (Verdadero) y fueron correctamente clasificadas como positivas por el modelo es de 1223.
- Falso negativo (False Negative, FN): El número de muestras que realmente pertenecen a la clase positiva (Verdadero) pero fueron mal clasificadas, como negativas por el modelo es de 90.

- Falso positivo (False Positive, FP): El número de muestras que realmente pertenecen a la clase negativa (Falso) pero fueron mal clasificadas, como positivas por el modelo es de 56.
- Verdadero negativo (True Negative, TN): El número de muestras que realmente pertenecen a la clase negativa (Falso) y fueron correctamente clasificadas como negativas por el modelo es de 247.

0.2.3 Importancia de los features (variables explicativas)

[28]:
```
plt.barh(features.columns, rf.feature_importances_)
```

[28]:
```
<BarContainer object of 15 artists>
```



El gráfico indica que por lejos, la variable que más importancia tiene para predecir si la persona se retirará o no corresponde a las horas en promedio de trabajo diario, siguiéndole (con mucha menos importancia) la edad a la que la persona fue encuestada.

Ahora, se realizará una variación de parametros.

[29]:
```
from sklearn.model_selection import GridSearchCV

# Create the parameter grid based on the results of random search
param_grid = {
    'bootstrap': [True],
```

```
    'max_depth': [4, 8, 16], #Cuanto pueden crecer máximo los arboles de␣
 ↪decisión
    'max_features': [6, 8], #Numero máximo de caracteristicas a considerar al␣
 ↪buscar la mejor división en un arbol
    'min_samples_leaf': [3, 4, 5], #Numero minimo de muestras requeridas en un␣
 ↪nodo terminal u hoja de un arbol
    'min_samples_split': [8, 10, 12], #Numero minimo de muestras necesarias␣
 ↪para dividir un nodo interno
    'n_estimators': [150, 200, 300] #Numero de arboles en el bosque
}

# Create a based model
rf = RandomForestClassifier()

# Instantiate the grid search model
grid_search = GridSearchCV(estimator = rf, param_grid = param_grid,
                          cv = 10, n_jobs = -1, #utilizar todos los nucleos
                           verbose = 2 # Mostrar mensajes mas detallados
                          )
```

[30]:
```
# Fit the grid search to the data
grid_search.fit(X_train_scaled, y_train)
#grid_search.best_params_
#best_grid = grid_search.best_estimator_
```

```
Fitting 10 folds for each of 162 candidates, totalling 1620 fits
```

[30]:
```
GridSearchCV(cv=10, estimator=RandomForestClassifier(), n_jobs=-1,
             param_grid={'bootstrap': [True], 'max_depth': [4, 8, 16],
                         'max_features': [6, 8], 'min_samples_leaf': [3, 4, 5],
                         'min_samples_split': [8, 10, 12],
                         'n_estimators': [150, 200, 300]},
             verbose=2)
```

[31]:
```
results_df = pd.DataFrame(grid_search.cv_results_)
results_df = results_df.sort_values(by=["rank_test_score"])
results_df = results_df.set_index(
    results_df["params"].apply(lambda x: "_".join(str(val) for val in x.
 ↪values()))
).rename_axis("kernel")
results_df[["params", "rank_test_score", "mean_test_score", "std_test_score"]]
```

[31]:
```
                                                                  params  \
kernel
True_16_6_5_10_200  {'bootstrap': True, 'max_depth': 16, 'max_feat…
True_8_6_4_10_150   {'bootstrap': True, 'max_depth': 8, 'max_featu…
True_8_6_3_10_300   {'bootstrap': True, 'max_depth': 8, 'max_featu…
```

```
True_8_6_3_12_200   {'bootstrap': True, 'max_depth': 8, 'max_featu…
True_16_6_3_8_300   {'bootstrap': True, 'max_depth': 16, 'max_feat…
…                                                               …
True_4_8_3_12_300   {'bootstrap': True, 'max_depth': 4, 'max_featu…
True_4_8_4_10_300   {'bootstrap': True, 'max_depth': 4, 'max_featu…
True_4_8_3_8_200    {'bootstrap': True, 'max_depth': 4, 'max_featu…
True_4_8_4_8_150    {'bootstrap': True, 'max_depth': 4, 'max_featu…
True_4_6_3_12_150   {'bootstrap': True, 'max_depth': 4, 'max_featu…

                    rank_test_score  mean_test_score  std_test_score
kernel
True_16_6_5_10_200                1         0.906248        0.010665
True_8_6_4_10_150                 2         0.905938        0.009650
True_8_6_3_10_300                 3         0.905784        0.010381
True_8_6_3_12_200                 4         0.905629        0.008665
True_16_6_3_8_300                 5         0.905474        0.011783
…                               …             …               …
True_4_8_3_12_300               158         0.902073        0.007129
True_4_8_4_10_300               158         0.902073        0.007129
True_4_8_3_8_200                160         0.901919        0.007421
True_4_8_4_8_150                161         0.901918        0.008680
True_4_6_3_12_150               162         0.901299        0.006270

[162 rows x 4 columns]
```

Se puede visualizar que en todos los casos en los que se realizaron distintas combinaciones de hiperparámentros, el ajuste es peor que el realizado en el modelo original, con solo 150 árboles de decisión.

0.2.4 Pregunta 3

0.2.5 Boosting

[82]:
```
gb = GradientBoostingClassifier(n_estimators=150, learning_rate=1.0, ␣
 ↪max_depth=1, random_state=42).fit(X_train, y_train)

# Making predictions on the test set
y_pred = gb.predict(X_test)

# Evaluating the accuracy of the classifier
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy:", accuracy)
```

```
Accuracy: 0.9053217821782178
```

Aquí se puede apreciar que el método Gradient Boosting básico que considera la misma cantidad de árboles que el método de Random Forest es menos preciso que este último, por lo que es peor. A continuación, se realizará una variación de parámetros para ver si existe otra combinación que incremente la precisión del método.

Tuning

[33]:
```
## Considerando Grid Search
gb = GradientBoostingClassifier()
parameters = {'learning_rate': [0.02,0.03,0.04],
                  'subsample'    : [0.9, 0.5],
                  'n_estimators' : [150,200,300],
                  'max_depth'    : [4,6,8]}

grid_GBR = GridSearchCV(estimator=gb, param_grid = parameters, cv = 8,␣
 ↪n_jobs=-1)
grid_GBR.fit(X_train_scaled, y_train)
```

[33]:
```
GridSearchCV(cv=8, estimator=GradientBoostingClassifier(), n_jobs=-1,
             param_grid={'learning_rate': [0.02, 0.03, 0.04],
                         'max_depth': [4, 6, 8],
                         'n_estimators': [150, 200, 300],
                         'subsample': [0.9, 0.5]})
```

[35]:
```
results_df = pd.DataFrame(grid_GBR.cv_results_)
results_df = results_df.sort_values(by=["rank_test_score"])
results_df = results_df.set_index(
    results_df["params"].apply(lambda x: "_".join(str(val) for val in x.
 ↪values()))
).rename_axis("kernel")
results_df[["params", "rank_test_score", "mean_test_score", "std_test_score"]]
```

[35]:
```
                                                           params  \
kernel
0.02_6_150_0.5  {'learning_rate': 0.02, 'max_depth': 6, 'n_est…
0.02_4_300_0.9  {'learning_rate': 0.02, 'max_depth': 4, 'n_est…
0.02_6_200_0.5  {'learning_rate': 0.02, 'max_depth': 6, 'n_est…
0.03_4_150_0.5  {'learning_rate': 0.03, 'max_depth': 4, 'n_est…
0.04_4_200_0.5  {'learning_rate': 0.04, 'max_depth': 4, 'n_est…
0.02_4_200_0.5  {'learning_rate': 0.02, 'max_depth': 4, 'n_est…
0.04_4_200_0.9  {'learning_rate': 0.04, 'max_depth': 4, 'n_est…
0.02_6_200_0.9  {'learning_rate': 0.02, 'max_depth': 6, 'n_est…
0.02_4_150_0.9  {'learning_rate': 0.02, 'max_depth': 4, 'n_est…
0.02_4_150_0.5  {'learning_rate': 0.02, 'max_depth': 4, 'n_est…
0.02_4_200_0.9  {'learning_rate': 0.02, 'max_depth': 4, 'n_est…
0.03_4_200_0.5  {'learning_rate': 0.03, 'max_depth': 4, 'n_est…
0.03_4_200_0.9  {'learning_rate': 0.03, 'max_depth': 4, 'n_est…
0.02_6_150_0.9  {'learning_rate': 0.02, 'max_depth': 6, 'n_est…
0.03_4_300_0.5  {'learning_rate': 0.03, 'max_depth': 4, 'n_est…
0.03_6_200_0.9  {'learning_rate': 0.03, 'max_depth': 6, 'n_est…
0.04_4_150_0.9  {'learning_rate': 0.04, 'max_depth': 4, 'n_est…
0.04_4_150_0.5  {'learning_rate': 0.04, 'max_depth': 4, 'n_est…
```

```
0.03_4_150_0.9  {'learning_rate': 0.03, 'max_depth': 4, 'n_est…
0.02_4_300_0.5  {'learning_rate': 0.02, 'max_depth': 4, 'n_est…
0.03_6_150_0.5  {'learning_rate': 0.03, 'max_depth': 6, 'n_est…
0.04_6_150_0.9  {'learning_rate': 0.04, 'max_depth': 6, 'n_est…
0.02_8_150_0.5  {'learning_rate': 0.02, 'max_depth': 8, 'n_est…
0.03_4_300_0.9  {'learning_rate': 0.03, 'max_depth': 4, 'n_est…
0.03_6_150_0.9  {'learning_rate': 0.03, 'max_depth': 6, 'n_est…
0.02_8_150_0.9  {'learning_rate': 0.02, 'max_depth': 8, 'n_est…
0.02_6_300_0.5  {'learning_rate': 0.02, 'max_depth': 6, 'n_est…
0.03_8_150_0.9  {'learning_rate': 0.03, 'max_depth': 8, 'n_est…
0.04_6_200_0.5  {'learning_rate': 0.04, 'max_depth': 6, 'n_est…
0.03_8_150_0.5  {'learning_rate': 0.03, 'max_depth': 8, 'n_est…
0.02_8_200_0.5  {'learning_rate': 0.02, 'max_depth': 8, 'n_est…
0.04_4_300_0.5  {'learning_rate': 0.04, 'max_depth': 4, 'n_est…
0.02_8_200_0.9  {'learning_rate': 0.02, 'max_depth': 8, 'n_est…
0.02_6_300_0.9  {'learning_rate': 0.02, 'max_depth': 6, 'n_est…
0.03_6_200_0.5  {'learning_rate': 0.03, 'max_depth': 6, 'n_est…
0.04_8_200_0.9  {'learning_rate': 0.04, 'max_depth': 8, 'n_est…
0.04_6_150_0.5  {'learning_rate': 0.04, 'max_depth': 6, 'n_est…
0.04_4_300_0.9  {'learning_rate': 0.04, 'max_depth': 4, 'n_est…
0.04_6_200_0.9  {'learning_rate': 0.04, 'max_depth': 6, 'n_est…
0.04_8_200_0.5  {'learning_rate': 0.04, 'max_depth': 8, 'n_est…
0.04_8_150_0.9  {'learning_rate': 0.04, 'max_depth': 8, 'n_est…
0.02_8_300_0.5  {'learning_rate': 0.02, 'max_depth': 8, 'n_est…
0.03_8_200_0.5  {'learning_rate': 0.03, 'max_depth': 8, 'n_est…
0.04_6_300_0.5  {'learning_rate': 0.04, 'max_depth': 6, 'n_est…
0.03_6_300_0.9  {'learning_rate': 0.03, 'max_depth': 6, 'n_est…
0.03_8_300_0.9  {'learning_rate': 0.03, 'max_depth': 8, 'n_est…
0.02_8_300_0.9  {'learning_rate': 0.02, 'max_depth': 8, 'n_est…
0.03_6_300_0.5  {'learning_rate': 0.03, 'max_depth': 6, 'n_est…
0.03_8_200_0.9  {'learning_rate': 0.03, 'max_depth': 8, 'n_est…
0.03_8_300_0.5  {'learning_rate': 0.03, 'max_depth': 8, 'n_est…
0.04_6_300_0.9  {'learning_rate': 0.04, 'max_depth': 6, 'n_est…
0.04_8_150_0.5  {'learning_rate': 0.04, 'max_depth': 8, 'n_est…
0.04_8_300_0.5  {'learning_rate': 0.04, 'max_depth': 8, 'n_est…
0.04_8_300_0.9  {'learning_rate': 0.04, 'max_depth': 8, 'n_est…

                rank_test_score  mean_test_score  std_test_score
kernel
0.02_6_150_0.5                1         0.903775        0.008919
0.02_4_300_0.9                2         0.903465        0.007704
0.02_6_200_0.5                3         0.903311        0.006903
0.03_4_150_0.5                4         0.903001        0.007947
0.04_4_200_0.5                5         0.902692        0.009383
0.02_4_200_0.5                6         0.902692        0.005175
0.04_4_200_0.9                6         0.902692        0.006999
0.02_6_200_0.9                8         0.902537        0.007623
```

0.02_4_150_0.9	9	0.902537	0.007673
0.02_4_150_0.5	10	0.902382	0.008266
0.02_4_200_0.9	10	0.902382	0.008055
0.03_4_200_0.5	10	0.902382	0.006819
0.03_4_200_0.9	13	0.902228	0.006946
0.02_6_150_0.9	14	0.902073	0.006045
0.03_4_300_0.5	15	0.901918	0.008724
0.03_6_200_0.9	16	0.901918	0.008768
0.04_4_150_0.9	17	0.901609	0.006188
0.04_4_150_0.5	18	0.901454	0.008091
0.03_4_150_0.9	19	0.901300	0.006118
0.02_4_300_0.5	19	0.901300	0.005526
0.03_6_150_0.5	19	0.901300	0.009339
0.04_6_150_0.9	22	0.901145	0.007514
0.02_8_150_0.5	23	0.900990	0.007949
0.03_4_300_0.9	23	0.900990	0.007001
0.03_6_150_0.9	25	0.900835	0.008449
0.02_8_150_0.9	26	0.900681	0.008273
0.02_6_300_0.5	26	0.900681	0.007445
0.03_8_150_0.9	28	0.900526	0.007653
0.04_6_200_0.5	28	0.900526	0.007628
0.03_8_150_0.5	30	0.900371	0.009282
0.02_8_200_0.5	30	0.900371	0.008663
0.04_4_300_0.5	32	0.900217	0.008595
0.02_8_200_0.9	32	0.900217	0.007501
0.02_6_300_0.9	32	0.900217	0.007603
0.03_6_200_0.5	35	0.899907	0.009495
0.04_8_200_0.9	36	0.899752	0.007900
0.04_6_150_0.5	37	0.899598	0.009812
0.04_4_300_0.9	38	0.899443	0.008133
0.04_6_200_0.9	39	0.899134	0.008989
0.04_8_200_0.5	40	0.898979	0.009545
0.04_8_150_0.9	40	0.898979	0.009861
0.02_8_300_0.5	42	0.898360	0.008126
0.03_8_200_0.5	43	0.898205	0.008547
0.04_6_300_0.5	44	0.898051	0.006124
0.03_6_300_0.9	44	0.898051	0.008278
0.03_8_300_0.9	46	0.897896	0.007296
0.02_8_300_0.9	47	0.897741	0.005966
0.03_6_300_0.5	48	0.897587	0.006715
0.03_8_200_0.9	49	0.897432	0.010491
0.03_8_300_0.5	50	0.897277	0.010299
0.04_6_300_0.9	51	0.896349	0.007673
0.04_8_150_0.5	52	0.894957	0.010120
0.04_8_300_0.5	53	0.894647	0.008673
0.04_8_300_0.9	54	0.894028	0.009625

Se puede apreciar que a pesar de haber variado los hiperparámetros, la precisión de Gradient

Boosting sigue siendo inferior a la de Random Forest en cualquiera de sus combinaciones, por lo que este método es peor que Random Forest al momento de clasificar esta base de datos.

0.2.6 Support Vector Machine

[36]:
```
from sklearn import svm

svm_m = svm.SVC()
svm_m.fit(X_train, y_train)

y_pred = svm_m.predict(X_test)

# Evaluating the accuracy of the classifier
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy:", accuracy)
```

```
Accuracy: 0.9053217821782178
```

Se puede visualizar que la precisión del método de Support Vector Machine a la hora de clasificar la base de datos es inferior a la de Random Forest pero similar a la de Gradient Boosting. A continuación, se realizará una variación de parámetros para ver si existe otra combinación que permita obtener una precisión superior.

Tuning

[37]:
```
# defining parameter range
param_grid = {'C': [0.01, 0.1, 1, 100, 1000],
              'gamma': [1, 0.1, 0.01, 0.001, 0.0001, 0.00001],
              'kernel': ['rbf']}

grid_SVM = GridSearchCV(svm_m, param_grid, refit = True, verbose = 3)

# fitting the model for grid search
grid_SVM.fit(X_train, y_train)
```

```
Fitting 5 folds for each of 30 candidates, totalling 150 fits
[CV 1/5] END …C=0.01, gamma=1, kernel=rbf;, score=0.817 total time=   2.6s
[CV 2/5] END …C=0.01, gamma=1, kernel=rbf;, score=0.817 total time=   3.0s
[CV 3/5] END …C=0.01, gamma=1, kernel=rbf;, score=0.817 total time=   2.4s
[CV 4/5] END …C=0.01, gamma=1, kernel=rbf;, score=0.817 total time=   2.3s
[CV 5/5] END …C=0.01, gamma=1, kernel=rbf;, score=0.817 total time=   2.3s
[CV 1/5] END …C=0.01, gamma=0.1, kernel=rbf;, score=0.817 total time=   1.2s
[CV 2/5] END …C=0.01, gamma=0.1, kernel=rbf;, score=0.817 total time=   1.2s
[CV 3/5] END …C=0.01, gamma=0.1, kernel=rbf;, score=0.817 total time=   1.2s
[CV 4/5] END …C=0.01, gamma=0.1, kernel=rbf;, score=0.817 total time=   1.2s
[CV 5/5] END …C=0.01, gamma=0.1, kernel=rbf;, score=0.817 total time=   1.6s
[CV 1/5] END …C=0.01, gamma=0.01, kernel=rbf;, score=0.817 total time=   1.1s
[CV 2/5] END …C=0.01, gamma=0.01, kernel=rbf;, score=0.817 total time=   1.1s
[CV 3/5] END …C=0.01, gamma=0.01, kernel=rbf;, score=0.817 total time=   0.9s
[CV 4/5] END …C=0.01, gamma=0.01, kernel=rbf;, score=0.817 total time=   0.8s
```

```
[CV 5/5] END …C=0.01, gamma=0.01, kernel=rbf;, score=0.817 total time=   0.8s
[CV 1/5] END …C=0.01, gamma=0.001, kernel=rbf;, score=0.817 total time=   0.7s
[CV 2/5] END …C=0.01, gamma=0.001, kernel=rbf;, score=0.817 total time=   0.8s
[CV 3/5] END …C=0.01, gamma=0.001, kernel=rbf;, score=0.817 total time=   0.8s
[CV 4/5] END …C=0.01, gamma=0.001, kernel=rbf;, score=0.817 total time=   0.7s
[CV 5/5] END …C=0.01, gamma=0.001, kernel=rbf;, score=0.817 total time=   0.7s
[CV 1/5] END ..C=0.01, gamma=0.0001, kernel=rbf;, score=0.817 total time=   0.8s
[CV 2/5] END ..C=0.01, gamma=0.0001, kernel=rbf;, score=0.817 total time=   0.7s
[CV 3/5] END ..C=0.01, gamma=0.0001, kernel=rbf;, score=0.817 total time=   0.7s
[CV 4/5] END ..C=0.01, gamma=0.0001, kernel=rbf;, score=0.817 total time=   0.7s
[CV 5/5] END ..C=0.01, gamma=0.0001, kernel=rbf;, score=0.817 total time=   0.9s
[CV 1/5] END …C=0.01, gamma=1e-05, kernel=rbf;, score=0.817 total time=   0.9s
[CV 2/5] END …C=0.01, gamma=1e-05, kernel=rbf;, score=0.817 total time=   1.1s
[CV 3/5] END …C=0.01, gamma=1e-05, kernel=rbf;, score=0.817 total time=   1.1s
[CV 4/5] END …C=0.01, gamma=1e-05, kernel=rbf;, score=0.817 total time=   1.0s
[CV 5/5] END …C=0.01, gamma=1e-05, kernel=rbf;, score=0.817 total time=   0.8s
[CV 1/5] END …C=0.1, gamma=1, kernel=rbf;, score=0.817 total time=   2.4s
[CV 2/5] END …C=0.1, gamma=1, kernel=rbf;, score=0.817 total time=   2.4s
[CV 3/5] END …C=0.1, gamma=1, kernel=rbf;, score=0.817 total time=   2.3s
[CV 4/5] END …C=0.1, gamma=1, kernel=rbf;, score=0.817 total time=   2.2s
[CV 5/5] END …C=0.1, gamma=1, kernel=rbf;, score=0.817 total time=   2.5s
[CV 1/5] END …C=0.1, gamma=0.1, kernel=rbf;, score=0.818 total time=   1.7s
[CV 2/5] END …C=0.1, gamma=0.1, kernel=rbf;, score=0.817 total time=   1.5s
[CV 3/5] END …C=0.1, gamma=0.1, kernel=rbf;, score=0.817 total time=   1.3s
[CV 4/5] END …C=0.1, gamma=0.1, kernel=rbf;, score=0.820 total time=   1.3s
[CV 5/5] END …C=0.1, gamma=0.1, kernel=rbf;, score=0.817 total time=   1.3s
[CV 1/5] END …C=0.1, gamma=0.01, kernel=rbf;, score=0.875 total time=   0.8s
[CV 2/5] END …C=0.1, gamma=0.01, kernel=rbf;, score=0.874 total time=   0.7s
[CV 3/5] END …C=0.1, gamma=0.01, kernel=rbf;, score=0.877 total time=   0.8s
[CV 4/5] END …C=0.1, gamma=0.01, kernel=rbf;, score=0.873 total time=   0.7s
[CV 5/5] END …C=0.1, gamma=0.01, kernel=rbf;, score=0.872 total time=   0.7s
[CV 1/5] END …C=0.1, gamma=0.001, kernel=rbf;, score=0.845 total time=   0.7s
[CV 2/5] END …C=0.1, gamma=0.001, kernel=rbf;, score=0.847 total time=   0.7s
[CV 3/5] END …C=0.1, gamma=0.001, kernel=rbf;, score=0.848 total time=   0.8s
[CV 4/5] END …C=0.1, gamma=0.001, kernel=rbf;, score=0.834 total time=   0.9s
[CV 5/5] END …C=0.1, gamma=0.001, kernel=rbf;, score=0.841 total time=   1.1s
[CV 1/5] END …C=0.1, gamma=0.0001, kernel=rbf;, score=0.817 total time=   1.0s
[CV 2/5] END …C=0.1, gamma=0.0001, kernel=rbf;, score=0.817 total time=   1.1s
[CV 3/5] END …C=0.1, gamma=0.0001, kernel=rbf;, score=0.817 total time=   0.7s
[CV 4/5] END …C=0.1, gamma=0.0001, kernel=rbf;, score=0.817 total time=   0.8s
[CV 5/5] END …C=0.1, gamma=0.0001, kernel=rbf;, score=0.817 total time=   0.8s
[CV 1/5] END …C=0.1, gamma=1e-05, kernel=rbf;, score=0.817 total time=   0.7s
[CV 2/5] END …C=0.1, gamma=1e-05, kernel=rbf;, score=0.817 total time=   0.8s
[CV 3/5] END …C=0.1, gamma=1e-05, kernel=rbf;, score=0.817 total time=   0.7s
[CV 4/5] END …C=0.1, gamma=1e-05, kernel=rbf;, score=0.817 total time=   0.7s
[CV 5/5] END …C=0.1, gamma=1e-05, kernel=rbf;, score=0.817 total time=   0.7s
[CV 1/5] END …C=1, gamma=1, kernel=rbf;, score=0.817 total time=   2.5s
[CV 2/5] END …C=1, gamma=1, kernel=rbf;, score=0.817 total time=   2.5s
```

```
[CV 3/5] END …C=1, gamma=1, kernel=rbf;, score=0.817 total time=   3.5s
[CV 4/5] END …C=1, gamma=1, kernel=rbf;, score=0.817 total time=   2.6s
[CV 5/5] END …C=1, gamma=1, kernel=rbf;, score=0.818 total time=   2.6s
[CV 1/5] END …C=1, gamma=0.1, kernel=rbf;, score=0.865 total time=   1.3s
[CV 2/5] END …C=1, gamma=0.1, kernel=rbf;, score=0.873 total time=   1.3s
[CV 3/5] END …C=1, gamma=0.1, kernel=rbf;, score=0.879 total time=   1.3s
[CV 4/5] END …C=1, gamma=0.1, kernel=rbf;, score=0.858 total time=   1.3s
[CV 5/5] END …C=1, gamma=0.1, kernel=rbf;, score=0.874 total time=   1.4s
[CV 1/5] END …C=1, gamma=0.01, kernel=rbf;, score=0.899 total time=   0.9s
[CV 2/5] END …C=1, gamma=0.01, kernel=rbf;, score=0.892 total time=   0.9s
[CV 3/5] END …C=1, gamma=0.01, kernel=rbf;, score=0.889 total time=   1.0s
[CV 4/5] END …C=1, gamma=0.01, kernel=rbf;, score=0.892 total time=   0.9s
[CV 5/5] END …C=1, gamma=0.01, kernel=rbf;, score=0.896 total time=   0.7s
[CV 1/5] END …C=1, gamma=0.001, kernel=rbf;, score=0.895 total time=   0.6s
[CV 2/5] END …C=1, gamma=0.001, kernel=rbf;, score=0.897 total time=   0.6s
[CV 3/5] END …C=1, gamma=0.001, kernel=rbf;, score=0.890 total time=   0.6s
[CV 4/5] END …C=1, gamma=0.001, kernel=rbf;, score=0.886 total time=   0.7s
[CV 5/5] END …C=1, gamma=0.001, kernel=rbf;, score=0.894 total time=   0.6s
[CV 1/5] END …C=1, gamma=0.0001, kernel=rbf;, score=0.859 total time=   0.7s
[CV 2/5] END …C=1, gamma=0.0001, kernel=rbf;, score=0.855 total time=   0.7s
[CV 3/5] END …C=1, gamma=0.0001, kernel=rbf;, score=0.859 total time=   0.7s
[CV 4/5] END …C=1, gamma=0.0001, kernel=rbf;, score=0.856 total time=   0.7s
[CV 5/5] END …C=1, gamma=0.0001, kernel=rbf;, score=0.846 total time=   0.7s
[CV 1/5] END …C=1, gamma=1e-05, kernel=rbf;, score=0.817 total time=   0.7s
[CV 2/5] END …C=1, gamma=1e-05, kernel=rbf;, score=0.817 total time=   0.7s
[CV 3/5] END …C=1, gamma=1e-05, kernel=rbf;, score=0.817 total time=   0.7s
[CV 4/5] END …C=1, gamma=1e-05, kernel=rbf;, score=0.817 total time=   0.8s
[CV 5/5] END …C=1, gamma=1e-05, kernel=rbf;, score=0.817 total time=   1.0s
[CV 1/5] END …C=100, gamma=1, kernel=rbf;, score=0.817 total time=   3.8s
[CV 2/5] END …C=100, gamma=1, kernel=rbf;, score=0.818 total time=   2.8s
[CV 3/5] END …C=100, gamma=1, kernel=rbf;, score=0.817 total time=   2.9s
[CV 4/5] END …C=100, gamma=1, kernel=rbf;, score=0.822 total time=   2.9s
[CV 5/5] END …C=100, gamma=1, kernel=rbf;, score=0.820 total time=   3.3s
[CV 1/5] END …C=100, gamma=0.1, kernel=rbf;, score=0.860 total time=   1.9s
[CV 2/5] END …C=100, gamma=0.1, kernel=rbf;, score=0.867 total time=   1.9s
[CV 3/5] END …C=100, gamma=0.1, kernel=rbf;, score=0.871 total time=   1.7s
[CV 4/5] END …C=100, gamma=0.1, kernel=rbf;, score=0.855 total time=   1.3s
[CV 5/5] END …C=100, gamma=0.1, kernel=rbf;, score=0.865 total time=   1.7s
[CV 1/5] END …C=100, gamma=0.01, kernel=rbf;, score=0.889 total time=   0.9s
[CV 2/5] END …C=100, gamma=0.01, kernel=rbf;, score=0.885 total time=   0.8s
[CV 3/5] END …C=100, gamma=0.01, kernel=rbf;, score=0.892 total time=   0.7s
[CV 4/5] END …C=100, gamma=0.01, kernel=rbf;, score=0.881 total time=   0.8s
[CV 5/5] END …C=100, gamma=0.01, kernel=rbf;, score=0.893 total time=   0.8s
[CV 1/5] END …C=100, gamma=0.001, kernel=rbf;, score=0.906 total time=   0.5s
[CV 2/5] END …C=100, gamma=0.001, kernel=rbf;, score=0.904 total time=   0.6s
[CV 3/5] END …C=100, gamma=0.001, kernel=rbf;, score=0.896 total time=   0.7s
[CV 4/5] END …C=100, gamma=0.001, kernel=rbf;, score=0.903 total time=   0.8s
[CV 5/5] END …C=100, gamma=0.001, kernel=rbf;, score=0.907 total time=   0.8s
```

```
[CV 1/5] END …C=100, gamma=0.0001, kernel=rbf;, score=0.894 total time=   0.7s
[CV 2/5] END …C=100, gamma=0.0001, kernel=rbf;, score=0.907 total time=   0.6s
[CV 3/5] END …C=100, gamma=0.0001, kernel=rbf;, score=0.898 total time=   0.5s
[CV 4/5] END …C=100, gamma=0.0001, kernel=rbf;, score=0.896 total time=   0.5s
[CV 5/5] END …C=100, gamma=0.0001, kernel=rbf;, score=0.905 total time=   0.5s
[CV 1/5] END …C=100, gamma=1e-05, kernel=rbf;, score=0.892 total time=   0.6s
[CV 2/5] END …C=100, gamma=1e-05, kernel=rbf;, score=0.898 total time=   0.6s
[CV 3/5] END …C=100, gamma=1e-05, kernel=rbf;, score=0.897 total time=   0.6s
[CV 4/5] END …C=100, gamma=1e-05, kernel=rbf;, score=0.891 total time=   0.6s
[CV 5/5] END …C=100, gamma=1e-05, kernel=rbf;, score=0.899 total time=   0.6s
[CV 1/5] END …C=1000, gamma=1, kernel=rbf;, score=0.817 total time=   2.7s
[CV 2/5] END …C=1000, gamma=1, kernel=rbf;, score=0.818 total time=   3.0s
[CV 3/5] END …C=1000, gamma=1, kernel=rbf;, score=0.817 total time=   3.9s
[CV 4/5] END …C=1000, gamma=1, kernel=rbf;, score=0.822 total time=   3.2s
[CV 5/5] END …C=1000, gamma=1, kernel=rbf;, score=0.820 total time=   2.8s
[CV 1/5] END …C=1000, gamma=0.1, kernel=rbf;, score=0.860 total time=   1.4s
[CV 2/5] END …C=1000, gamma=0.1, kernel=rbf;, score=0.867 total time=   1.8s
[CV 3/5] END …C=1000, gamma=0.1, kernel=rbf;, score=0.871 total time=   1.6s
[CV 4/5] END …C=1000, gamma=0.1, kernel=rbf;, score=0.855 total time=   1.3s
[CV 5/5] END …C=1000, gamma=0.1, kernel=rbf;, score=0.865 total time=   2.7s
[CV 1/5] END …C=1000, gamma=0.01, kernel=rbf;, score=0.866 total time=   2.4s
[CV 2/5] END …C=1000, gamma=0.01, kernel=rbf;, score=0.889 total time=   2.2s
[CV 3/5] END …C=1000, gamma=0.01, kernel=rbf;, score=0.891 total time=   2.3s
[CV 4/5] END …C=1000, gamma=0.01, kernel=rbf;, score=0.880 total time=   2.2s
[CV 5/5] END …C=1000, gamma=0.01, kernel=rbf;, score=0.880 total time=   2.1s
[CV 1/5] END …C=1000, gamma=0.001, kernel=rbf;, score=0.914 total time=   1.0s
[CV 2/5] END …C=1000, gamma=0.001, kernel=rbf;, score=0.903 total time=   1.2s
[CV 3/5] END …C=1000, gamma=0.001, kernel=rbf;, score=0.897 total time=   1.3s
[CV 4/5] END …C=1000, gamma=0.001, kernel=rbf;, score=0.906 total time=   1.5s
[CV 5/5] END …C=1000, gamma=0.001, kernel=rbf;, score=0.909 total time=   1.0s
[CV 1/5] END ..C=1000, gamma=0.0001, kernel=rbf;, score=0.903 total time=   0.6s
[CV 2/5] END ..C=1000, gamma=0.0001, kernel=rbf;, score=0.909 total time=   0.6s
[CV 3/5] END ..C=1000, gamma=0.0001, kernel=rbf;, score=0.899 total time=   0.5s
[CV 4/5] END ..C=1000, gamma=0.0001, kernel=rbf;, score=0.899 total time=   0.6s
[CV 5/5] END ..C=1000, gamma=0.0001, kernel=rbf;, score=0.909 total time=   0.6s
[CV 1/5] END …C=1000, gamma=1e-05, kernel=rbf;, score=0.896 total time=   0.5s
[CV 2/5] END …C=1000, gamma=1e-05, kernel=rbf;, score=0.906 total time=   0.5s
[CV 3/5] END …C=1000, gamma=1e-05, kernel=rbf;, score=0.894 total time=   0.5s
[CV 4/5] END …C=1000, gamma=1e-05, kernel=rbf;, score=0.896 total time=   0.5s
[CV 5/5] END …C=1000, gamma=1e-05, kernel=rbf;, score=0.904 total time=   0.5s
```

[37]:
```
GridSearchCV(estimator=SVC(),
             param_grid={'C': [0.01, 0.1, 1, 100, 1000],
                         'gamma': [1, 0.1, 0.01, 0.001, 0.0001, 1e-05],
                         'kernel': ['rbf']},
             verbose=3)
```

[38]:
```
results_df = pd.DataFrame(grid_SVM.cv_results_)
results_df = results_df.sort_values(by=["rank_test_score"])
results_df = results_df.set_index(
    results_df["params"].apply(lambda x: "_".join(str(val) for val in x.
 ↪values()))
).rename_axis("kernel")
results_df[["params", "rank_test_score", "mean_test_score", "std_test_score"]]
```

[38]:

kernel	params	rank_test_score	mean_test_score	std_test_score
1000_0.001_rbf	{'C': 1000, 'gamma': 0.001, 'kernel': 'rbf'}	1	0.905941	0.005729
1000_0.0001_rbf	{'C': 1000, 'gamma': 0.0001, 'kernel': 'rbf'}	2	0.903930	0.004146
100_0.001_rbf	{'C': 100, 'gamma': 0.001, 'kernel': 'rbf'}	3	0.903157	0.004001
100_0.0001_rbf	{'C': 100, 'gamma': 0.0001, 'kernel': 'rbf'}	4	0.899908	0.005176
1000_1e-05_rbf	{'C': 1000, 'gamma': 1e-05, 'kernel': 'rbf'}			
100_1e-05_rbf	{'C': 100, 'gamma': 1e-05, 'kernel': 'rbf'}			
1_0.01_rbf	{'C': 1, 'gamma': 0.01, 'kernel': 'rbf'}			
1_0.001_rbf	{'C': 1, 'gamma': 0.001, 'kernel': 'rbf'}			
100_0.01_rbf	{'C': 100, 'gamma': 0.01, 'kernel': 'rbf'}			
1000_0.01_rbf	{'C': 1000, 'gamma': 0.01, 'kernel': 'rbf'}			
0.1_0.01_rbf	{'C': 0.1, 'gamma': 0.01, 'kernel': 'rbf'}			
1_0.1_rbf	{'C': 1, 'gamma': 0.1, 'kernel': 'rbf'}			
1000_0.1_rbf	{'C': 1000, 'gamma': 0.1, 'kernel': 'rbf'}			
100_0.1_rbf	{'C': 100, 'gamma': 0.1, 'kernel': 'rbf'}			
1_0.0001_rbf	{'C': 1, 'gamma': 0.0001, 'kernel': 'rbf'}			
0.1_0.001_rbf	{'C': 0.1, 'gamma': 0.001, 'kernel': 'rbf'}			
100_1_rbf	{'C': 100, 'gamma': 1, 'kernel': 'rbf'}			
1000_1_rbf	{'C': 1000, 'gamma': 1, 'kernel': 'rbf'}			
0.1_0.1_rbf	{'C': 0.1, 'gamma': 0.1, 'kernel': 'rbf'}			
1_1_rbf	{'C': 1, 'gamma': 1, 'kernel': 'rbf'}			
1_1e-05_rbf	{'C': 1, 'gamma': 1e-05, 'kernel': 'rbf'}			
0.01_1e-05_rbf	{'C': 0.01, 'gamma': 1e-05, 'kernel': 'rbf'}			
0.01_0.0001_rbf	{'C': 0.01, 'gamma': 0.0001, 'kernel': 'rbf'}			
0.01_0.001_rbf	{'C': 0.01, 'gamma': 0.001, 'kernel': 'rbf'}			
0.01_0.01_rbf	{'C': 0.01, 'gamma': 0.01, 'kernel': 'rbf'}			
0.1_1e-05_rbf	{'C': 0.1, 'gamma': 1e-05, 'kernel': 'rbf'}			
0.1_0.0001_rbf	{'C': 0.1, 'gamma': 0.0001, 'kernel': 'rbf'}			
0.01_0.1_rbf	{'C': 0.01, 'gamma': 0.1, 'kernel': 'rbf'}			
0.1_1_rbf	{'C': 0.1, 'gamma': 1, 'kernel': 'rbf'}			
0.01_1_rbf	{'C': 0.01, 'gamma': 1, 'kernel': 'rbf'}			

1000_1e-05_rbf	5	0.899289	0.004960
100_1e-05_rbf	6	0.895421	0.003093
1_0.01_rbf	7	0.893410	0.003285
1_0.001_rbf	8	0.892327	0.004068
100_0.01_rbf	9	0.887841	0.004516
1000_0.01_rbf	10	0.881343	0.008829
0.1_0.01_rbf	11	0.874381	0.001690
1_0.1_rbf	12	0.869895	0.007542
1000_0.1_rbf	13	0.863552	0.005668
100_0.1_rbf	13	0.863552	0.005668
1_0.0001_rbf	15	0.855197	0.004872
0.1_0.001_rbf	16	0.842821	0.005265
100_1_rbf	17	0.818843	0.001907
1000_1_rbf	17	0.818843	0.001907
0.1_0.1_rbf	19	0.818069	0.000923
1_1_rbf	20	0.817451	0.000446
1_1e-05_rbf	21	0.816832	0.000253
0.01_1e-05_rbf	21	0.816832	0.000253
0.01_0.0001_rbf	21	0.816832	0.000253
0.01_0.001_rbf	21	0.816832	0.000253
0.01_0.01_rbf	21	0.816832	0.000253
0.1_1e-05_rbf	21	0.816832	0.000253
0.1_0.0001_rbf	21	0.816832	0.000253
0.01_0.1_rbf	21	0.816832	0.000253
0.1_1_rbf	21	0.816832	0.000253
0.01_1_rbf	21	0.816832	0.000253

Se puede visualizar que al fijar el hiperparámetro "C" en 1000, se obtiene una precisión superior a la del modelo Support Vector Machine original, pero que al mismo tiempo es inferior a la de Random Forest, por lo cual se considera en este caso que el mejor método entre Random Forest, Gradient Boosting y Support Vector Machine para clasificar la base de datos es Random Forest.

0.2.7 Stacking

[39]:
```
estimators = [
    ('rf', RandomForestClassifier(n_estimators=150, max_depth=16,␣
 ↪max_features=8, min_samples_split=8, random_state=42)),
    ('boost', GradientBoostingClassifier(n_estimators=300, learning_rate=0.04,␣
 ↪max_depth=8, random_state=42)),
    ('svc', svm.SVC(C=1, gamma=0.0001))]

stack = StackingClassifier(estimators=estimators,␣
 ↪final_estimator=LogisticRegression())

stack.fit(X_train, y_train).score(X_test, y_test)
```

[39]:
```
0.9102722772277227
```

[40]:
```
cv = cross_val_score(stack, X_train, y_train, cv=10, scoring='accuracy',␣
↪n_jobs=-1)
print('Accuracy: %.6f' % round(np.mean(cv),6))
```

```
Accuracy: 0.904854
```

Se puede apreciar que al utilizar un ensamblador de clasificadores se obtiene una precisión del 90.4%, que es inferior a la de Random Forest, superior a la de Gradient Boosting e inferior a la de SVM al cambiar los hiperparámetros. Por ello, es posible afirmar que una combinación lineal de los modelos es menos precisa que el modelo Random Forest original.

0.2.8 Pregunta 4

Se eliminaron las filas que contienen valores faltantes en la base de datos, se convirtió la variable "export" en entera, se agruparon los valores de la variable "year" de la base de datos y se cuenta cuántas veces aparece cada valor correspondiente al año. El resultado se guarda en la variable "bycount", se agruparon los valores de la columna "ID" de la base de datos y se cuenta cuántas veces aparece cada valor. El resultado se guarda en la variable "bIDcount", se calcula el logaritmo natural de la variable "utilidades" de la base, después de restar el valor mínimo y sumar 0.1 para evitar el logaritmo de cero o valores negativos, se filtra la base de datos de forma que se mantienen solo las filas donde el valor de la variable "utilidades" sea menor a 10000.

[16]:
```
enia=pd.read_csv('C:/Users/equipo/Downloads/Tarea 4/LAB-MAA-main/data/enia.csv')
enia.dropna(inplace=True)
enia.export = enia.export.astype(int)
Xa = enia
bycount = enia['year'].groupby(enia['year']).count()
bIDcount = enia['ID'].groupby(enia['ID']).count()
enia['utilidades']=np.log(enia['utilidades']-enia['utilidades'].min()+0.1)
enia = enia[enia["utilidades"] < 10000]
enia.describe()
```

[16]:

	ID	year	tamano	sales	age
count	39104.000000	39104.000000	39104.000000	39104.000000	39104.000000
mean	218089.455554	2011.787183	2.248773	3.574172	15.305084
std	128228.474792	3.781237	1.153089	1.692742	12.488330
min	100000.000000	2007.000000	1.000000	0.000000	0.000000
25%	105409.000000	2007.000000	1.000000	2.337643	7.000000
50%	200994.500000	2013.000000	2.000000	3.553321	14.000000
75%	302466.250000	2015.000000	3.000000	4.539098	20.000000
max	507526.000000	2017.000000	4.000000	10.309005	190.000000

	foreign	export	workers	fomento	iyd
count	39104.000000	39104.000000	39104.000000	39104.000000	39104.000000
mean	0.081859	0.111191	1.757726	0.076105	0.224887
std	0.274153	0.314372	1.186507	0.265169	0.417514
min	0.000000	0.000000	0.000000	0.000000	0.000000
25%	0.000000	0.000000	0.778151	0.000000	0.000000

```
50%        0.000000      0.000000      1.785330      0.000000      0.000000
75%        0.000000      0.000000      2.661813      0.000000      0.000000
max        1.000000      1.000000      5.845915      1.000000      1.000000

          impuestos    utilidades
count  39104.000000  39104.000000
mean       0.203856      5.500845
std       15.869466      0.060706
min     -180.992528     -2.302585
25%        0.000000      5.499092
50%        0.000007      5.499092
75%        0.000167      5.499097
max     2981.494528     10.729529
```

A continuación, se define a "enia2" como un objeto correspondiente a la base de datos "enia" pero excluyendo a las variables de "ID" y "tamano", tal como lo pide el enunciado. Luego, se normaliza esta base de datos, con el objetivo de poder utilizar el método "K-means".

[17]:
```
enia2 = enia.drop(['tamano', 'ID'], axis=1)
enia2
```

[17]:
```
       year     sales  age  foreign  export   workers  fomento  iyd  \
0      2007  7.046558   22        1       1  3.486855        0    1
1      2009  7.875563   24        1       1  3.504607        0    0
2      2013  7.437399   23        1       1  4.691621        1    1
3      2015  7.356472   30        1       1  4.682614        1    0
4      2017  4.014772   32        1       1  4.611691        1    0
…     …        …  …      …     …       …      …  …
39101  2017  1.401027   17        0       0  0.000000        0    0
39102  2017  1.528097   22        0       0  0.000000        0    0
39103  2017  1.679748   21        0       0  1.397940        0    0
39104  2017  1.285538    9        0       0  1.568202        0    0
39105  2017  1.441296   12        0       0  0.000000        0    0

       impuestos  utilidades
0       1.231345    5.527776
1       8.762233    5.629224
2       0.001886    5.511843
3       0.411670    5.504389
4       0.000721    5.499100
…          …         …
39101   0.000001    5.499092
39102   0.000000    5.499092
39103   0.000003    5.499092
39104   0.000000    5.499092
39105   0.000002    5.499092

[39104 rows x 10 columns]
```

[18]:
```
enia2_norm = (enia2-enia2.min())/(enia2.max()-enia2.min())
enia2_norm
```

[18]:
```
        year     sales       age  foreign  export   workers  fomento  iyd  \
0        0.0  0.683534  0.115789      1.0     1.0  0.596460      0.0  1.0
1        0.2  0.763950  0.126316      1.0     1.0  0.599497      0.0  0.0
2        0.6  0.721447  0.121053      1.0     1.0  0.802547      1.0  1.0
3        0.8  0.713597  0.157895      1.0     1.0  0.801006      1.0  0.0
4        1.0  0.389443  0.168421      1.0     1.0  0.788874      1.0  0.0
…      …        …         …        …       …        …  …
39101    1.0  0.135903  0.089474      0.0     0.0  0.000000      0.0  0.0
39102    1.0  0.148229  0.115789      0.0     0.0  0.000000      0.0  0.0
39103    1.0  0.162940  0.110526      0.0     0.0  0.239131      0.0  0.0
39104    1.0  0.124700  0.047368      0.0     0.0  0.268256      0.0  0.0
39105    1.0  0.139809  0.063158      0.0     0.0  0.000000      0.0  0.0

        impuestos  utilidades
0        0.057620    0.600851
1        0.060002    0.608636
2        0.057232    0.599629
3        0.057361    0.599057
4        0.057231    0.598651
…           …          …
39101    0.057231    0.598650
39102    0.057231    0.598650
39103    0.057231    0.598650
39104    0.057231    0.598650
39105    0.057231    0.598650

[39104 rows x 10 columns]
```

0.3 Clustering

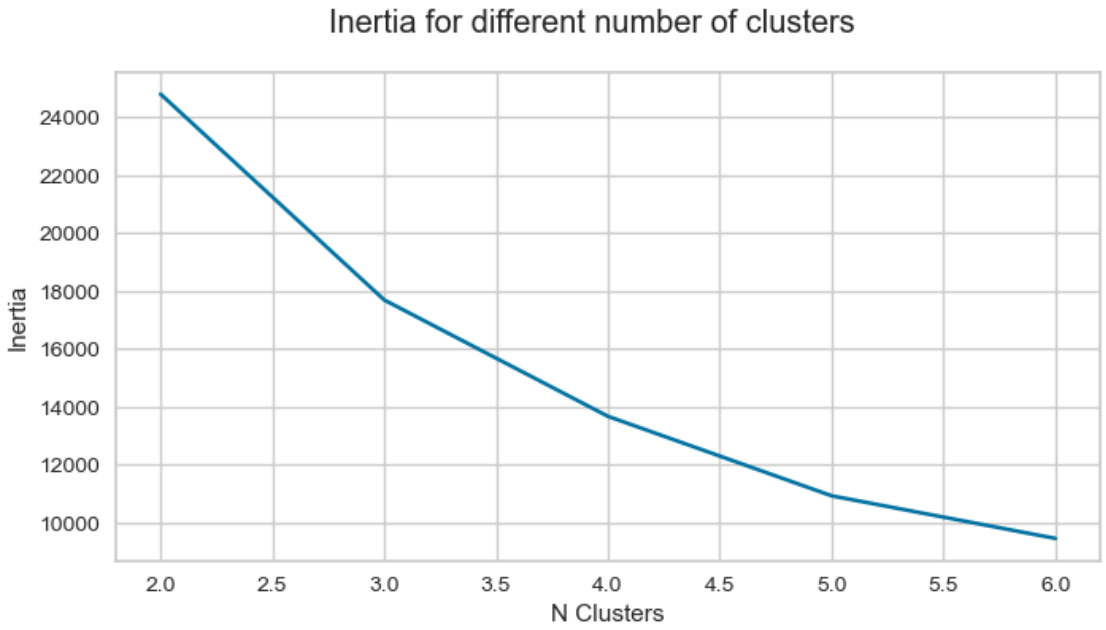
0.3.1 K-means

[19]:
```
total_clusters = 5
param_search = pd.DataFrame({
    "N Clusters": np.arange(2,total_clusters+2,dtype=int),
    "Inertia": [0]*total_clusters
})
for n_clusters in range(total_clusters):
    param_search.loc[n_clusters, "Inertia"] = KMeans(n_clusters=n_clusters+1,␣
 ↪random_state=1).fit(enia2_norm).inertia_

fig, ax = plt.subplots(1,1, figsize=(8,4))
sns.lineplot(data=param_search, x="N Clusters", y="Inertia", ax=ax)
fig.suptitle("Inertia for different number of clusters")
fig.show()
```

```
n_clusters = 4
cluster_labels = pd.Series(KMeans(n_clusters=n_clusters, random_state=1).
 ↪fit(enia2_norm).labels_)
cluster_labels.value_counts()
```

```
C:\Users\equipo\AppData\Local\Temp\ipykernel_9676\3258729259.py:12: UserWarning:
Matplotlib is currently using module://matplotlib_inline.backend_inline, which
is a non-GUI backend, so cannot show the figure.
  fig.show()
```

[19]:
```
3    15611
0    12190
1     6955
2     4348
dtype: int64
```



Se observa en el gráfico que la última caída abrupta de la inercia (que indica qué tan bien estan conformados los clusters) fue de 2 a 3 clusters, por lo que se considera que el número óptimo de clusters para esta base de datos es de 3.

[20]:
```
kmeans = KMeans(n_clusters=3, random_state=32, max_iter=500, n_init=5)
kmeans.fit(enia2_norm)
```

[20]:
```
KMeans(max_iter=500, n_clusters=3, n_init=5, random_state=32)
```

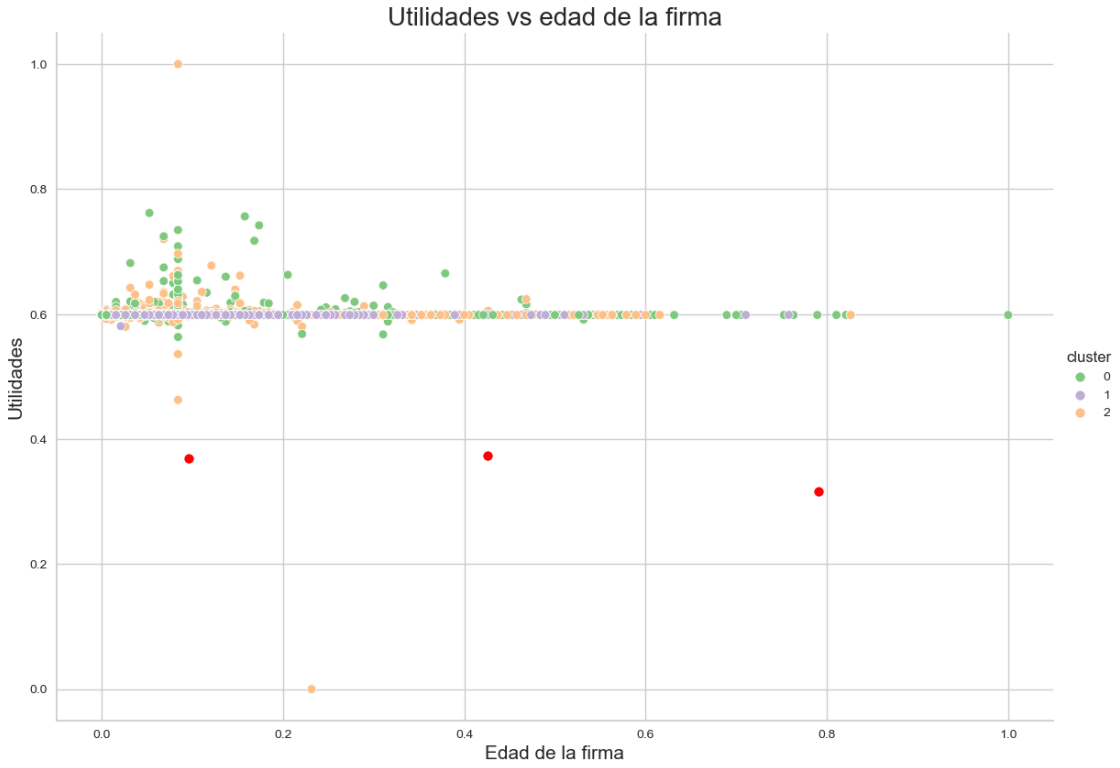
[21]:
```
enia2_norm['Kmeans_Clusters'] = kmeans.labels_
enia2_norm.head
```

[21]:
```
<bound method NDFrame.head of        year     sales       age  foreign  export
workers  fomento  iyd  \
0       0.0  0.683534  0.115789      1.0     1.0  0.596460      0.0  1.0
1       0.2  0.763950  0.126316      1.0     1.0  0.599497      0.0  0.0
2       0.6  0.721447  0.121053      1.0     1.0  0.802547      1.0  1.0
3       0.8  0.713597  0.157895      1.0     1.0  0.801006      1.0  0.0
4       1.0  0.389443  0.168421      1.0     1.0  0.788874      1.0  0.0
…     …        …       …      …     …       …      …  …
39101   1.0  0.135903  0.089474      0.0     0.0  0.000000      0.0  0.0
39102   1.0  0.148229  0.115789      0.0     0.0  0.000000      0.0  0.0
39103   1.0  0.162940  0.110526      0.0     0.0  0.239131      0.0  0.0
39104   1.0  0.124700  0.047368      0.0     0.0  0.268256      0.0  0.0
39105   1.0  0.139809  0.063158      0.0     0.0  0.000000      0.0  0.0

       impuestos  utilidades  Kmeans_Clusters
0       0.057620    0.600851                0
1       0.060002    0.608636                2
2       0.057232    0.599629                0
3       0.057361    0.599057                1
4       0.057231    0.598651                1
…          …         …                 …
39101   0.057231    0.598650                1
39102   0.057231    0.598650                1
39103   0.057231    0.598650                1
39104   0.057231    0.598650                1
39105   0.057231    0.598650                1

[39104 rows x 11 columns]>
```

[22]:
```
kmeans = KMeans(n_clusters=3, random_state=32, max_iter=500,n_init=5)
y_kmeans = kmeans.fit_predict(enia2_norm)

kmeans = KMeans(n_clusters=3,init= "random", random_state = 1).fit(enia2_norm)
centroids = kmeans.cluster_centers_
plt.figure(figsize=(15,8))
Data_kmean = enia2_norm.copy()
Data_kmean['cluster'] = kmeans.labels_
sns.relplot(data = Data_kmean ,x='age' , y  ='utilidades', hue='cluster',␣
 ↪palette='Accent' ,kind='scatter', height=8.27, aspect = 11.7/8.27)
plt.scatter(centroids[:, 0], centroids[:,1], c='red', s=50)
plt.title("Utilidades vs edad de la firma", fontsize = 20)
plt.xlabel("Edad de la firma",fontsize=15)
plt.ylabel("Utilidades",fontsize=15)
```

[22]: Text(35.431541989817454, 0.5, 'Utilidades')

<Figure size 1500x800 with 0 Axes>

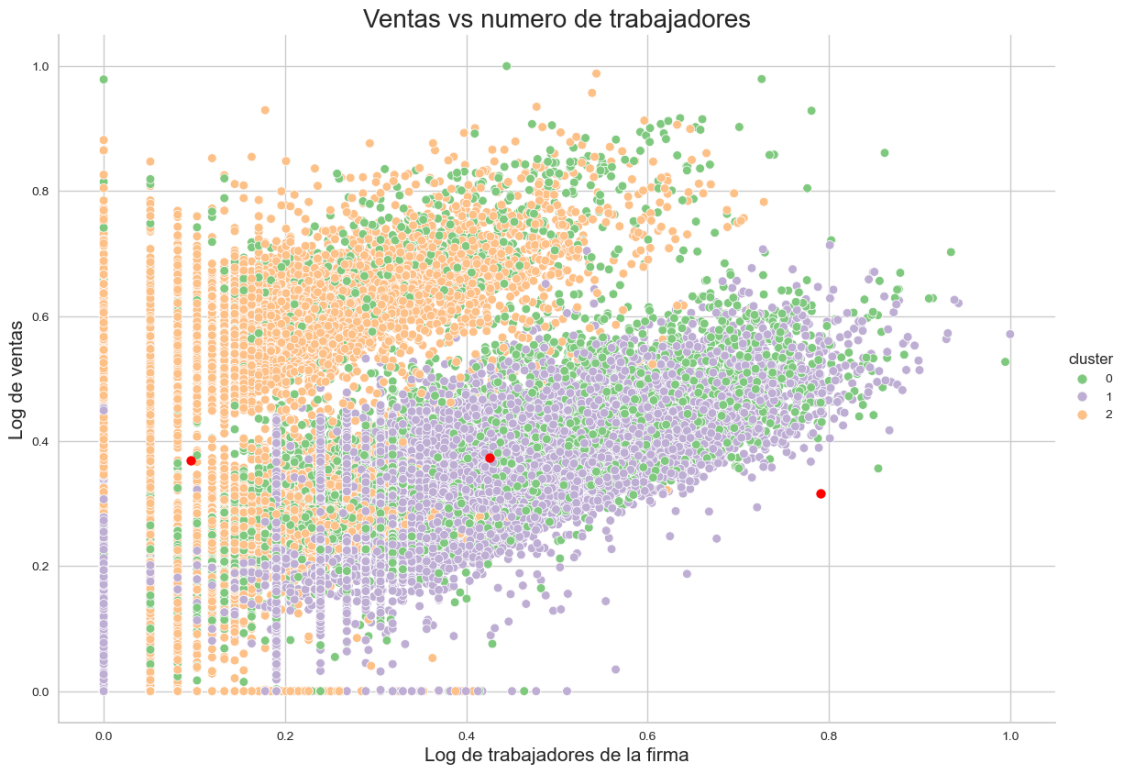


En el gráfico, se observa que las variables correspondientes al logaritmo de las utilidades de la firma y la edad no son dominantes, debido a que no es clara una separación agrupada en función de sus ejes.

[137]:
```
kmeans = KMeans(n_clusters=3,init= "random", random_state = 1).fit(enia2_norm)
centroids = kmeans.cluster_centers_
plt.figure(figsize=(15,8))
Data_kmean = enia2_norm.copy()
Data_kmean['cluster'] = kmeans.labels_
sns.relplot(data = Data_kmean ,x='workers' , y  ='sales', hue='cluster',␣
 ↪palette='Accent' ,kind='scatter', height=8.27, aspect = 11.7/8.27)
plt.scatter(centroids[:, 0], centroids[:,1], c='red', s=50)
plt.title("Ventas vs numero de trabajadores", fontsize = 20)
plt.xlabel("Log de trabajadores de la firma",fontsize=15)
plt.ylabel("Log de ventas",fontsize=15)
```

[137]: Text(35.431541989817454, 0.5, 'Log de ventas')

<Figure size 1500x800 with 0 Axes>

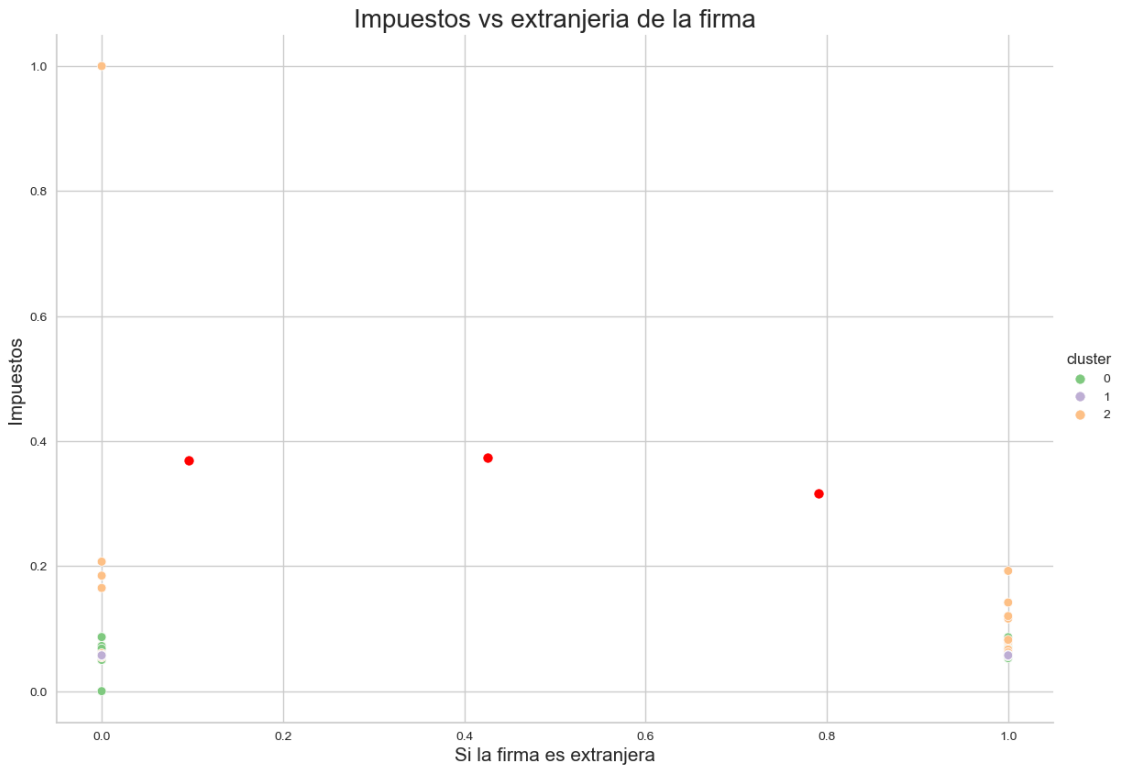


En este caso se puede visualizar una dominancia de la variable asociada a las ventas de la firma, ya que en función del eje de las ordenadas se observa una clara separación de clusters, mientras que en el eje x no es clara la dependencia de este eje para la formación de clusters, por lo que la variable asociada al número de trabajadores no es dominante.

[144]:
```
kmeans = KMeans(n_clusters=3,init= "random", random_state = 1).fit(enia2_norm)
centroids = kmeans.cluster_centers_
plt.figure(figsize=(15,8))
Data_kmean = enia2_norm.copy()
Data_kmean['cluster'] = kmeans.labels_
sns.relplot(data = Data_kmean ,x='foreign' , y  ='impuestos', hue='cluster',␣
 ↪palette='Accent' ,kind='scatter', height=8.27, aspect = 11.7/8.27)
plt.scatter(centroids[:, 0], centroids[:,1], c='red', s=50)
plt.title("Impuestos vs extranjeria de la firma", fontsize = 20)
plt.xlabel("Si la firma es extranjera",fontsize=15)
plt.ylabel("Impuestos",fontsize=15)
```

[144]:
```
Text(35.431541989817454, 0.5, 'Impuestos')
```

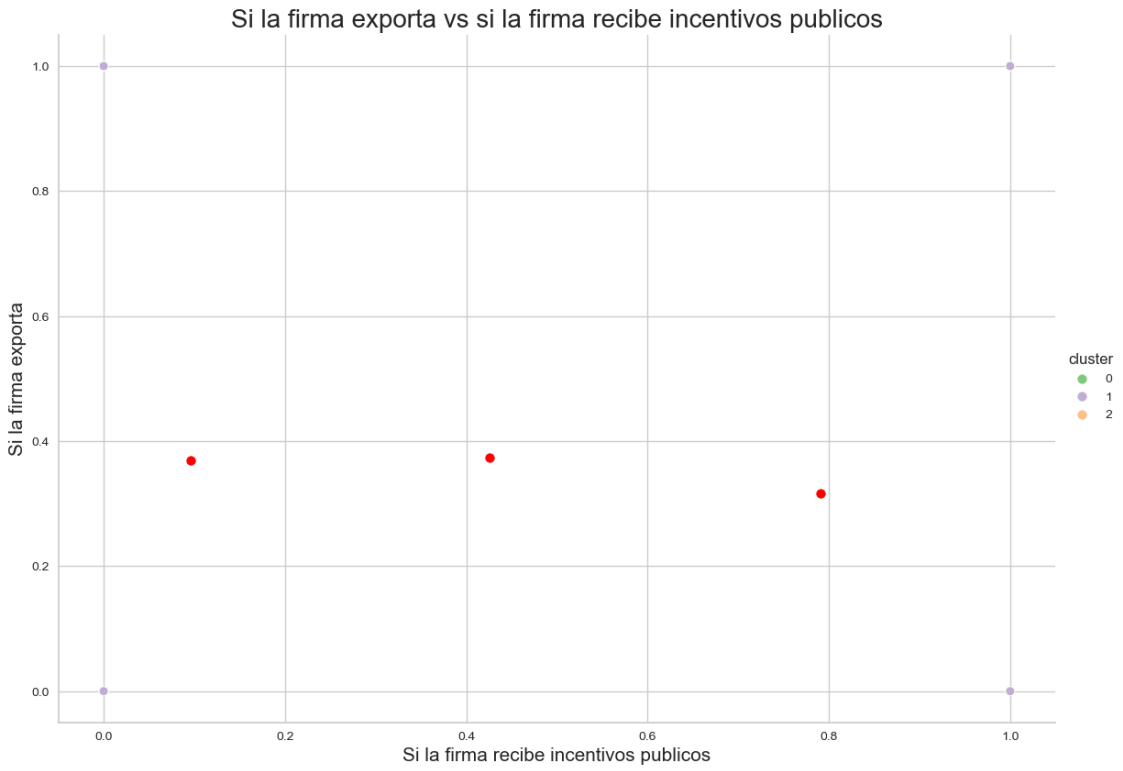
```
<Figure size 1500x800 with 0 Axes>
```



No se ve clara una formacion de clusters cuando las variables de impuestos y de si la firma es extranjera cambian de valor.

[145]:
```
kmeans = KMeans(n_clusters=3,init= "random", random_state = 1).fit(enia2_norm)
centroids = kmeans.cluster_centers_
plt.figure(figsize=(15,8))
Data_kmean = enia2_norm.copy()
Data_kmean['cluster'] = kmeans.labels_
sns.relplot(data = Data_kmean ,x='fomento' , y  ='export', hue='cluster',␣
 ↪palette='Accent' ,kind='scatter', height=8.27, aspect = 11.7/8.27)
plt.scatter(centroids[:, 0], centroids[:,1], c='red', s=50)
plt.title("Si la firma exporta vs si la firma recibe incentivos publicos",␣
 ↪fontsize = 20)
plt.xlabel("Si la firma recibe incentivos publicos",fontsize=15)
plt.ylabel("Si la firma exporta",fontsize=15)
```

[145]:
```
Text(35.431541989817454, 0.5, 'Si la firma exporta')

<Figure size 1500x800 with 0 Axes>
```



A pesar de que se esperaban claras separaciones dadas las distintas combinaciones de las variables binarias, no se encontro evidencia para afirmar que las variables de si la firma exporta o si recibe incentivos publicos puedan formar agrupaciones distintas.

[23]:
```
from sklearn.cluster import KMeans

kmeans = KMeans(n_clusters=3, random_state=32, max_iter=500, n_init=5)
kmeans.fit(enia2_norm)

inercia = kmeans.inertia_
print("Inercia:", inercia)
```

```
Inercia: 13670.741417628758
```

[25]:
```
from sklearn.cluster import KMeans

kmeans2 = KMeans(n_clusters=2, random_state=32, max_iter=500, n_init=5)
kmeans2.fit(enia2_norm)

inercia = kmeans2.inertia_
print("Inercia:", inercia)
```

```
Inercia: 25104.6349175832
```

[27]:
```
from sklearn.cluster import KMeans

kmeans3 = KMeans(n_clusters=4, random_state=32, max_iter=1000, n_init=5)
kmeans3.fit(enia2_norm)

inercia = kmeans3.inertia_
print("Inercia:", inercia)
```

```
Inercia: 11867.768674452846
```

[28]:
```
kmeans4 = KMeans(n_clusters=3, random_state=32, max_iter=1000, n_init=5)
kmeans4.fit(enia2_norm)

inercia = kmeans4.inertia_
print("Inercia:", inercia)
```

```
Inercia: 13670.741417628758
```

Se puede apreciar que al considerar 3 clusters en lugar de 2 la inercia disminuye significativamente, es decir, el modelo K-means es mejor cuando se tienen 3 clusters y es levemente mejor cuando se tienen 4 clusters, pero debido a que la diferencia de inercia entre 3 y 4 clusters considerados no es demasiado alta, se considera que el modelo con 3 clusters es mejor. Luego, al incrementar el número de iteraciones el modelo no varía en nada su medida de calidad de ajuste.

0.3.2 Pregunta 5

0.3.3 DBSCAN

[146]:
```
from sklearn.neighbors import NearestNeighbors
neighbors = NearestNeighbors(n_neighbors=5)
neighbors_fit = neighbors.fit(enia2)
distances, indices = neighbors_fit.kneighbors(enia2)
```

[ ]:
```
#dbscan = DBSCAN(eps=2100, min_samples=100)
# Fit the DBSCAN model to the data
#dbscan.fit(enia2)
# Extract the cluster labels
#labels = dbscan.labels_
# Print the number of clusters
#n_clusters = len(np.unique(labels))
#print(f"Number of clusters: {n_clusters}")

#data_dbscan = enia2.copy()
#data_dbscan['cluster'] = dbscan.labels_
#sns.relplot(data = data_dbscan ,x='age' , y  ='utilidades', hue='cluster',␣
 ↪palette='Accent' ,kind='scatter', height=8.27, aspect = 11.7/8.27)
```

Mi computador casi muere al ejecutar este codigo, por lo cual dejo la pregunta 5 en blanco y para evitar perder mis datos otra vez, decidi dejar el codigo marcado como comentarios.