Clasificación Ejercicio 1

```
[1] # Importar las librerías necesarias
    import numpy as np
    import pandas as pd
    import matplotlib.pyplot as plt
    from sklearn.svm import SVC
    from sklearn.neighbors import KNeighborsClassifier
    from sklearn.tree import DecisionTreeClassifier
    from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
    from sklearn.discriminant_analysis import QuadraticDiscriminantAnalysis
    from sklearn.multiclass import OneVsRestClassifier

    from sklearn.preprocessing import StandardScaler

    from sklearn.feature_selection import SelectKBest, f_classif, SequentialFeatureSelector, RFE
    from sklearn.model_selection import train_test_split, StratifiedKFold
    from sklearn.metrics import recall_score, accuracy_score, classification_report
    from sklearn.utils.class_weight import compute_class_weight
```

```
[3] df = pd.read_csv('datos_ej1.txt', delimiter='\t', header=None)
    df = df.drop(columns=[155])
    df_clean = df.dropna()
    data = df_clean.to_numpy()

    # Separar las columnas: la primera es la clase, la segunda se ignora
    clases = data[:, 0]
    variables = data[:, 2:]
```

```
[4] print(clases)
    print(variables)
```

```
[1. 1. 1. ... 2. 2. 2.]
[[ 0.35064718 -0.42499289 -0.66086573 ... -0.31974841  0.59501498
   1.47385124]
 [ 1.58037485  1.17660703 -0.19197485 ...  0.05187984  0.10499566
   0.32481216]
 [-0.31600831  1.05618389  2.02518827 ...  1.44506729  1.67126541
   1.01422794]
 ...
 [ 0.83549938  0.21770378 -0.02633703 ... -0.69412959 -1.33083079
  -1.2422245 ]
 [-1.00218632 -1.44704626 -1.1893213  ... -0.2287077  -0.75036633
  -1.0083598 ]
 [-2.25450972 -1.34026227 -0.07863387 ...  0.38196512 -0.0196701
  -0.25582255]]
```

Vamos a checar si nuestos datos están balanceados o no

```
[5] n_class1 = np.sum(clases == 1)
    n_class2 = np.sum(clases == 2)

    print(f'Clase 1: {n_class1} ({(n_class1 * 100 / (n_class1 + n_class2)):.2f}%), Clase 2: {n_class2} ({(n_class2 * 100 / (n_class1 + n_class2)):.2f}%)')
```

```
Clase 1: 299 (25.04%), Clase 2: 895 (74.96%)
```

CrossValidation

```python
def CV_Upsampling(variables, classes, clf, kf, model):
    cv_y_test = []
    cv_y_pred = []

    for train_index, test_index in kf.split(variables, classes):

        # Fase de entrenamiento
        x_train = variables[train_index, :]
        y_train = classes[train_index]

        x1 = x_train[y_train == 1, :]
        y1 = y_train[y_train == 1]
        n1 = len(y1)

        x2 = x_train[y_train == 2, :]
        y2 = y_train[y_train == 2]
        n2 = len(y2)

        # Upsampling
        ind = np.random.choice(np.arange(n1), size=n2, replace=True)

        x_sub = np.concatenate((x1[ind, :], x2), axis=0)
        y_sub = np.concatenate((y1[ind], y2), axis=0)

        clf.fit(x_sub, y_sub)

        # Fase de prueba
        x_test = variables[test_index, :]
        y_test = classes[test_index]
        y_pred = clf.predict(x_test)

        cv_y_test.append(y_test)
        cv_y_pred.append(y_pred)

    # Resultados de la clasificación después del upsampling
    print(f"{model} Classification Report:")
    print(classification_report(np.concatenate(cv_y_test), np.concatenate(cv_y_pred), zero_division=0))
```

```
[7] def CV_Subsampling(variables, classes, clf, kf, model):
        cv_y_test = []
        cv_y_pred = []

        for train_index, test_index in kf.split(variables, classes):

            # Fase de entrenamiento
            x_train = variables[train_index, :]
            y_train = classes[train_index]

            x1 = x_train[y_train == 1, :]
            y1 = y_train[y_train == 1]
            n1 = len(y1)

            x2 = x_train[y_train == 2, :]
            y2 = y_train[y_train == 2]
            n2 = len(y2)

            # Subsampling
            ind = np.random.choice(range(n2), n1, replace=False)

            x_sub = np.concatenate((x1, x2[ind, :]), axis=0)
            y_sub = np.concatenate((y1, y2[ind]), axis=0)

            clf.fit(x_sub, y_sub)

            # Fase de prueba
            x_test = variables[test_index, :]
            y_test = classes[test_index]
            y_pred = clf.predict(x_test)

            cv_y_test.append(y_test)
            cv_y_pred.append(y_pred)

        # Resultados de la clasificación después del upsampling
        print(f"{model} Classification Report:")
        print(classification_report(np.concatenate(cv_y_test), np.concatenate(cv_y_pred), zero_division=0))
```

```
[8] def CV_Standard(variables, classes, clf, kf, model):
        cv_y_test = []
        cv_y_pred = []

        for train_index, test_index in kf.split(variables, classes):
            # Training phase
            x_train = variables[train_index, :]
            y_train = classes[train_index]

            clf.fit(x_train, y_train)

            # Test phase
            x_test = variables[test_index, :]
            y_test = classes[test_index]
            y_pred = clf.predict(x_test)

            cv_y_test.append(y_test)
            cv_y_pred.append(y_pred)

        # Classification results
        print(f"{model} Classification Report:")
        print(classification_report(np.concatenate(cv_y_test), np.concatenate(cv_y_pred), zero_division=0))
```

Maquinas de Soporte Vectorial

```
[9] # Maquinas de Soporte Vectorial Upsampled
def SVM(variables, classes):
  clf = SVC(kernel='linear')
  kf = StratifiedKFold(n_splits=5, shuffle=True)

  CV_Upsampling(variables, classes, clf, kf, 'SVM')

SVM(variables, clases)
```

```
SVM Classification Report:
              precision    recall  f1-score   support

         1.0       0.84      0.88      0.86       299
         2.0       0.96      0.95      0.95       895

    accuracy                           0.93      1194
   macro avg       0.90      0.91      0.91      1194
weighted avg       0.93      0.93      0.93      1194
```

```
[ ] # Maquinas de Soporte Vectorial SelfBalanced
def SVM_Balanced(variables, classes):
    clf = SVC(kernel='linear', class_weight='balanced')
    kf = StratifiedKFold(n_splits=5, shuffle=True)
    CV_Standard(variables, classes, clf, kf, 'SVM')

SVM_Balanced(variables, clases)
```

```
SVM Classification Report:
              precision    recall  f1-score   support

         1.0       0.82      0.89      0.85       299
         2.0       0.96      0.94      0.95       895

    accuracy                           0.92      1194
   macro avg       0.89      0.91      0.90      1194
weighted avg       0.93      0.92      0.92      1194
```

K-Vecinos

```
# K-Vecinos
def KNN(variables, classes, n_neighbors=5):
    clf = KNeighborsClassifier(n_neighbors=n_neighbors)
    kf = StratifiedKFold(n_splits=5, shuffle=True)

    CV_Upsampling(variables, classes, clf, kf, 'KNN')

KNN(variables, clases)
```

```
KNN Classification Report:
              precision    recall  f1-score   support

         1.0       0.73      0.89      0.80       299
         2.0       0.96      0.89      0.93       895

    accuracy                           0.89      1194
   macro avg       0.85      0.89      0.87      1194
weighted avg       0.90      0.89      0.90      1194
```

Discriminante Lineal

```
[ ] # Discriminante Lineal
def LDA(variables, classes):
    clf = LinearDiscriminantAnalysis()
    kf = StratifiedKFold(n_splits=5, shuffle=True)

    CV_Upsampling(variables, classes, clf, kf, 'Discriminante Lineal')

LDA(variables, clases)
```

```
Discriminante Lineal Classification Report:
              precision    recall  f1-score   support

         1.0       0.79      0.91      0.85       299
         2.0       0.97      0.92      0.94       895

    accuracy                           0.92      1194
   macro avg       0.88      0.91      0.90      1194
weighted avg       0.92      0.92      0.92      1194
```

Arboles de Decisión

```
[ ] # Decision Tree Upsampled
    def DecisionTree(variables, classes):
        clf = DecisionTreeClassifier()
        kf = StratifiedKFold(n_splits=5, shuffle=True)

        CV_Upsampling(variables, classes, clf, kf, 'Decision Tree')

    DecisionTree(variables, clases)
```

```
Decision Tree Classification Report:
              precision    recall  f1-score   support

         1.0       0.72      0.71      0.72       299
         2.0       0.90      0.91      0.91       895

    accuracy                           0.86      1194
   macro avg       0.81      0.81      0.81      1194
weighted avg       0.86      0.86      0.86      1194
```

```
[ ] # Decision Tree SelfBalanced
    def DecisionTree_Balanced(variables, classes):
        clf = DecisionTreeClassifier(class_weight='balanced')
        kf = StratifiedKFold(n_splits=5, shuffle=True)
        CV_Standard(variables, classes, clf, kf, 'Decision Tree')

    DecisionTree_Balanced(variables, clases)
```

```
Decision Tree Classification Report:
              precision    recall  f1-score   support

         1.0       0.73      0.69      0.71       299
         2.0       0.90      0.92      0.91       895

    accuracy                           0.86      1194
   macro avg       0.82      0.80      0.81      1194
weighted avg       0.86      0.86      0.86      1194
```

Discriminante Multiclase

```
[ ] # Discriminante Multiclase
    def Multiclase(variables, classes):
        clf = OneVsRestClassifier(SVC(kernel='linear'))
        kf = StratifiedKFold(n_splits=5, shuffle=True)

        CV_Upsampling(variables, classes, clf, kf, 'Multiclase')

    Multiclase(variables, clases)
```

```
Multiclase Classification Report:
              precision    recall  f1-score   support

         1.0       0.85      0.87      0.86       299
         2.0       0.96      0.95      0.95       895

    accuracy                           0.93      1194
   macro avg       0.90      0.91      0.91      1194
weighted avg       0.93      0.93      0.93      1194
```

```
[ ] # Discriminante Multiclase SelfBalanced
    def Multiclase_Balanced(variables, classes):
        clf = OneVsRestClassifier(SVC(kernel='linear', class_weight='balanced'))
        kf = StratifiedKFold(n_splits=5, shuffle=True)
        CV_Standard(variables, classes, clf, kf, 'Multiclase')

    Multiclase_Balanced(variables, clases)
```

```
Multiclase Classification Report:
              precision    recall  f1-score   support

         1.0       0.81      0.89      0.84       299
         2.0       0.96      0.93      0.94       895

    accuracy                           0.92      1194
   macro avg       0.88      0.91      0.89      1194
weighted avg       0.92      0.92      0.92      1194
```

Discriminante Cuadrático

```
[ ] # Discriminante Cuadrático
    def QDA(variables, classes):
        clf = QuadraticDiscriminantAnalysis()
        kf = StratifiedKFold(n_splits=5, shuffle=True)

        CV_Upsampling(variables, classes, clf, kf, 'Discriminante Cuadrático')

    QDA(variables, clases)
```

```
Discriminante Cuadrático Classification Report:
              precision    recall  f1-score   support

         1.0       0.00      0.00      0.00       299
         2.0       0.75      1.00      0.86       895

    accuracy                           0.75      1194
   macro avg       0.37      0.50      0.43      1194
weighted avg       0.56      0.75      0.64      1194
```

Regresión Logística

```
[ ]
class CustomLogisticRegression:
    def __init__(self, learning_rate=0.001, n_iters=1000):
        self.lr = learning_rate
        self.n_iters = n_iters
        self.weights = None
        self.bias = None
        self.losses = []

    def _sigmoid(self, x):
        return 1 / (1 + np.exp(-x))

    def compute_loss(self, y_true, y_pred):
        epsilon = 1e-9
        y1 = y_true * np.log(y_pred + epsilon)
        y2 = (1-y_true) * np.log(1 - y_pred + epsilon)
        return -np.mean(y1 + y2)

    def feed_forward(self, X):
        z = np.dot(X, self.weights) + self.bias
        A = self._sigmoid(z)
        return A

    def fit(self, X, y):
        n_samples, n_features = X.shape
        self.weights = np.random.randn(n_features) * 0.01
        self.bias = 0

        for _ in range(self.n_iters):
            A = self.feed_forward(X)
            self.losses.append(self.compute_loss(y, A))
            dz = A - y
            dw = (1 / n_samples) * np.dot(X.T, dz)
            db = (1 / n_samples) * np.sum(dz)
            self.weights -= self.lr * dw
            self.bias -= self.lr * db

    def predict(self, X):
        threshold = 0.5
        y_hat = np.dot(X, self.weights) + self.bias
        y_predicted = self._sigmoid(y_hat)
        y_predicted_cls = [1 if i > threshold else 0 for i in y_predicted]
        return np.array(y_predicted_cls)
```

```
[ ] # Regresión Logística
    def LogisticRegression(variables, classes):
        clf = CustomLogisticRegression()
        kf = StratifiedKFold(n_splits=5, shuffle=True)

        CV_Standard(variables, classes, clf, kf, 'Regresión Logística')

    LogisticRegression(variables, clases)
```

```
Regresión Logística Classification Report:
              precision    recall  f1-score   support

         0.0       0.00      0.00      0.00         0
         1.0       0.22      0.79      0.34       299
         2.0       0.00      0.00      0.00       895

    accuracy                           0.20      1194
   macro avg       0.07      0.26      0.11      1194
weighted avg       0.05      0.20      0.08      1194
```

```
[ ] # Regresión Logística
    from sklearn.linear_model import LogisticRegression

    def SKLogisticRegression(variables, classes):
        clf = LogisticRegression(class_weight='balanced')
        kf = StratifiedKFold(n_splits=5, shuffle=True)

        CV_Standard(variables, classes, clf, kf, 'Regresión Logística')

    SKLogisticRegression(variables, clases)
```

```
Regresión Logística Classification Report:
              precision    recall  f1-score   support

         1.0       0.84      0.90      0.87       299
         2.0       0.97      0.94      0.95       895

    accuracy                           0.93      1194
   macro avg       0.90      0.92      0.91      1194
weighted avg       0.93      0.93      0.93      1194
```

```
[ ] def FilterFeatureSelection(x, y):
        print("----- Feature selection using 50% of predictors -----")

        # Select features
        fselection = SelectKBest(f_classif, k = 6)
        fselection.fit(x, y)

        print("Selected features: ", fselection.get_feature_names_out())

        # Fit model using the new dataset
        clf = SVC(kernel = 'linear')
        x_transformed = fselection.transform(x)
        clf.fit(x_transformed, y)

        # Evaluate model using cross validation
        cv_y_test = []
        cv_y_pred = []

        kf = StratifiedKFold(n_splits=5, shuffle = True)

        for train_index, test_index in kf.split(x, y):

            # Training phase
            x_train = x[train_index, :]
            y_train = y[train_index]

            clf_cv = SVC(kernel = 'linear')

            fselection_cv = SelectKBest(f_classif, k = 6)
            fselection_cv.fit(x_train, y_train)
            x_train = fselection_cv.transform(x_train)

            clf_cv.fit(x_train, y_train)

            # Test phase
            x_test = fselection_cv.transform(x[test_index, :])
            y_test = y[test_index]
            y_pred = clf_cv.predict(x_test)

            cv_y_test.append(y_test)
            cv_y_pred.append(y_pred)

        print(classification_report(np.concatenate(cv_y_test), np.concatenate(cv_y_pred)))
```

```python
print("----- Optimal selection of number of features -----")

n_feats = range(1, variables.shape[1] + 1)

acc_nfeat = []

for n_feat in n_feats:
    print('---- n features =', n_feat)

    acc_cv = []

    kf = StratifiedKFold(n_splits=5, shuffle = True)

    for train_index, test_index in kf.split(x, y):

        # Training phase
        x_train = x[train_index, :]
        y_train = y[train_index]

        clf_cv = SVC(kernel = 'linear')

        fselection_cv = SelectKBest(f_classif, k = n_feat)
        fselection_cv.fit(x_train, y_train)
        x_train = fselection_cv.transform(x_train)

        clf_cv.fit(x_train, y_train)

        # Test phase
        x_test = fselection_cv.transform(x[test_index, :])
        y_test = y[test_index]
        y_pred = clf_cv.predict(x_test)


        acc_i = accuracy_score(y_test, y_pred)
        acc_cv.append(acc_i)

    acc = np.average(acc_cv)
    acc_nfeat.append(acc)

    print('ACC:', acc)

opt_index = np.argmax(acc_nfeat)
opt_features = n_feats[opt_index]
print("Optimal number of features: ", opt_features)

plt.plot(n_feats, acc_nfeat)
plt.xlabel("features")
plt.ylabel("Accuracy")

plt.show()
```

```
        # Fit model with optimal number of features
        clf = SVC(kernel = 'linear')
        fselection = SelectKBest(f_classif, k = opt_features)
        fselection.fit(x, y)

        print("Selected features: ", fselection.get_feature_names_out())

        x_transformed = fselection.transform(x)
        clf.fit(x_transformed, y)

    FilterFeatureSelection(variables, clases)
```

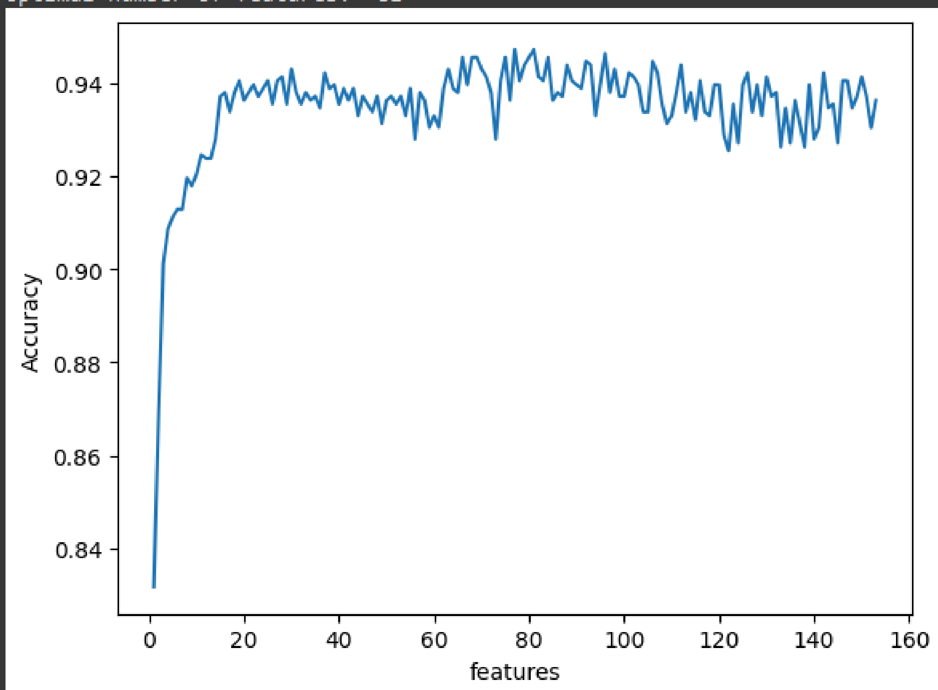
```
----- Feature selection using 50% of predictors -----
Selected features:  ['x17' 'x18' 'x19' 'x25' 'x26' 'x27']
              precision    recall  f1-score   support

         1.0       0.86      0.80      0.82       299
         2.0       0.93      0.96      0.94       895

    accuracy                           0.92      1194
   macro avg       0.89      0.88      0.88      1194
weighted avg       0.91      0.92      0.91      1194

----- Optimal selection of number of features -----
---- n features = 1
ACC: 0.8316690693013606
---- n features = 2
ACC: 0.8685172813895432
---- n features = 3
ACC: 0.901177877008544
---- n features = 4
ACC: 0.9087092577616822
---- n features = 5
ACC: 0.9112408143173587
---- n features = 6
ACC: 0.9129003902816357
---- n features = 7
ACC: 0.9128863260785487
---- n features = 8
ACC: 0.9196019830526353
---- n features = 9
ACC: 0.9179353749868149
---- n features = 10
ACC: 0.9204352870855456
---- n features = 11
ACC: 0.9246264196054991
---- n features = 12
ACC: 0.9238001476741324
---- n features = 13
ACC: 0.9238106958264478
---- n features = 14
ACC: 0.927973699940227
---- n features = 15
```

```
ACC: 0.9405435814493162
---- n features = 147
ACC: 0.9405189690939137
---- n features = 148
ACC: 0.9346647445589115
---- n features = 149
ACC: 0.9371646566576421
---- n features = 150
ACC: 0.9413768854822264
---- n features = 151
ACC: 0.9371892690130446
---- n features = 152
ACC: 0.9304665799374143
---- n features = 153
ACC: 0.9363594810309062
Optimal number of features:  81
```




```
Selected features:  ['x0' 'x1' 'x2' 'x7' 'x8' 'x10' 'x11' 'x12' 'x15' 'x16' 'x17' 'x18' 'x19'
 'x20' 'x22' 'x23' 'x24' 'x25' 'x26' 'x27' 'x28' 'x29' 'x30' 'x32' 'x33'
 'x34' 'x35' 'x53' 'x54' 'x61' 'x62' 'x63' 'x65' 'x66' 'x67' 'x68' 'x69'
 'x72' 'x73' 'x74' 'x75' 'x76' 'x78' 'x79' 'x80' 'x81' 'x82' 'x83' 'x86'
 'x87' 'x88' 'x89' 'x90' 'x91' 'x92' 'x93' 'x100' 'x101' 'x109' 'x118'
 'x119' 'x120' 'x121' 'x124' 'x125' 'x126' 'x127' 'x128' 'x133' 'x134'
 'x135' 'x136' 'x137' 'x138' 'x139' 'x140' 'x141' 'x142' 'x150' 'x151'
 'x152']
```

```python
def SecuentialFeatureSelection(x, y):
    print("----- Feature selection using 50% of predictors -----")

    # Select features
    clf = SVC(kernel = 'linear')
    fselection = SequentialFeatureSelector(clf, n_features_to_select = 0.5)
    fselection.fit(x, y)

    print("Selected features: ", fselection.get_feature_names_out())

    # Fit model using the new dataset
    x_transformed = fselection.transform(x)
    clf.fit(x_transformed, y)

    # Evaluate model using cross validation
    cv_y_test = []
    cv_y_pred = []

    kf = StratifiedKFold(n_splits=5, shuffle = True)

    for train_index, test_index in kf.split(x, y):

        # Training phase
        x_train = x[train_index, :]
        y_train = y[train_index]

        clf_cv = SVC(kernel = 'linear')

        fselection_cv = SequentialFeatureSelector(clf_cv, n_features_to_select=0.5)
        fselection_cv.fit(x_train, y_train)
        x_train = fselection_cv.transform(x_train)

        clf_cv.fit(x_train, y_train)

        # Test phase
        x_test = fselection_cv.transform(x[test_index, :])
        y_test = y[test_index]
        y_pred = clf_cv.predict(x_test)

        cv_y_test.append(y_test)
        cv_y_pred.append(y_pred)

    print(classification_report(np.concatenate(cv_y_test), np.concatenate(cv_y_pred)))
```

```
print("----- Optimal selection of number of features -----")

n_feats = range(1, variables.shape[1] + 1)

acc_nfeat = []

for n_feat in n_feats:
    print('---- n features =', n_feat)

    acc_cv = []

    kf = StratifiedKFold(n_splits=5, shuffle = True)

    for train_index, test_index in kf.split(x, y):

        # Training phase
        x_train = x[train_index, :]
        y_train = y[train_index]

        clf_cv = SVC(kernel = 'linear')

        fselection_cv = SequentialFeatureSelector(clf_cv, n_features_to_select=n_feat)
        fselection_cv.fit(x_train, y_train)
        x_train = fselection_cv.transform(x_train)

        clf_cv.fit(x_train, y_train)

        # Test phase
        x_test = fselection_cv.transform(x[test_index, :])
        y_test = y[test_index]
        y_pred = clf_cv.predict(x_test)


        acc_i = accuracy_score(y_test, y_pred)
        acc_cv.append(acc_i)

    acc = np.average(acc_cv)
    acc_nfeat.append(acc)

    print('ACC:', acc)

opt_index = np.argmax(acc_nfeat)
opt_features = n_feats[opt_index]
print("Optimal number of features: ", opt_features)

plt.plot(n_feats, acc_nfeat)
plt.xlabel("features")
plt.ylabel("Accuracy")

plt.show()
```

```
    # Fit model with optimal number of features
    clf = SVC(kernel = 'linear')
    fselection = SequentialFeatureSelector(clf, n_features_to_select = opt_features)
    fselection.fit(x, y)

    print("Selected features: ", fselection.get_feature_names_out())

    x_transformed = fselection.transform(x)
    clf.fit(x_transformed, y)

SecuentialFeatureSelection(variables, clases)
```

```
    SecuentialFeatureSelection(variables, clases)
[23]   250m 15.3s
...
ACC: 0.9221124433036814
---- n features = 10
ACC: 0.9187581308674098
---- n features = 11
ACC: 0.9162476706163638
---- n features = 12
ACC: 0.9187651629689533
---- n features = 13
ACC: 0.9304630638866426
---- n features = 14
ACC: 0.9262754474174608
---- n features = 15
ACC: 0.9170880067508176
---- n features = 16
ACC: 0.9262930276713195
---- n features = 17
ACC: 0.9221265075067684
---- n features = 18
ACC: 0.9313280123764989
---- n features = 19
ACC: 0.9246229035547273
---- n features = 20
ACC: 0.930473612038958
---- n features = 21
ACC: 0.9254456594353222
---- n features = 22
ACC: 0.9103828979290463
---- n features = 23
ACC: 0.9338244084244577
---- n features = 24
```

```python
def RecursiveFeatureSelector(x, y):
    print("----- Feature selection using 50% of predictors -----")

    # Select features
    clf = SVC(kernel = 'linear')
    fselection = RFE(clf, n_features_to_select = 0.5)
    fselection.fit(x, y)

    print("Selected features: ", fselection.get_feature_names_out())

    # Fit model using the new dataset
    x_transformed = fselection.transform(x)
    clf.fit(x_transformed, y)

    # Evaluate model using cross validation
    cv_y_test = []
    cv_y_pred = []

    kf = StratifiedKFold(n_splits=5, shuffle = True)

    for train_index, test_index in kf.split(x, y):

        # Training phase
        x_train = x[train_index, :]
        y_train = y[train_index]

        clf_cv = SVC(kernel = 'linear')

        fselection_cv = RFE(clf_cv, n_features_to_select=0.5)
        fselection_cv.fit(x_train, y_train)
        x_train = fselection_cv.transform(x_train)

        clf_cv.fit(x_train, y_train)

        # Test phase
        x_test = fselection_cv.transform(x[test_index, :])
        y_test = y[test_index]
        y_pred = clf_cv.predict(x_test)

        cv_y_test.append(y_test)
        cv_y_pred.append(y_pred)

    print(classification_report(np.concatenate(cv_y_test), np.concatenate(cv_y_pred)))
```

```python
print("----- Optimal selection of number of features -----")

n_feats = range(1, variables.shape[1] + 1)

acc_nfeat = []

for n_feat in n_feats:
    print('---- n features =', n_feat)

    acc_cv = []

    kf = StratifiedKFold(n_splits=5, shuffle = True)

    for train_index, test_index in kf.split(x, y):

        # Training phase
        x_train = x[train_index, :]
        y_train = y[train_index]

        clf_cv = SVC(kernel = 'linear')

        fselection_cv = RFE(clf_cv, n_features_to_select=n_feat)
        fselection_cv.fit(x_train, y_train)
        x_train = fselection_cv.transform(x_train)

        clf_cv.fit(x_train, y_train)

        # Test phase
        x_test = fselection_cv.transform(x[test_index, :])
        y_test = y[test_index]
        y_pred = clf_cv.predict(x_test)


        acc_i = accuracy_score(y_test, y_pred)
        acc_cv.append(acc_i)

    acc = np.average(acc_cv)
    acc_nfeat.append(acc)

    print('ACC:', acc)

opt_index = np.argmax(acc_nfeat)
opt_features = n_feats[opt_index]
print("Optimal number of features: ", opt_features)

plt.plot(n_feats, acc_nfeat)
plt.xlabel("features")
plt.ylabel("Accuracy")

plt.show()
```

```
    # Fit model with optimal number of features
    clf = SVC(kernel = 'linear')
    fselection = RFE(clf, n_features_to_select = opt_features)
    fselection.fit(x, y)

    print("Selected features: ", fselection.get_feature_names_out())

    x_transformed = fselection.transform(x)
    clf.fit(x_transformed, y)

RecursiveFeatureSelector(variables, clases)
```

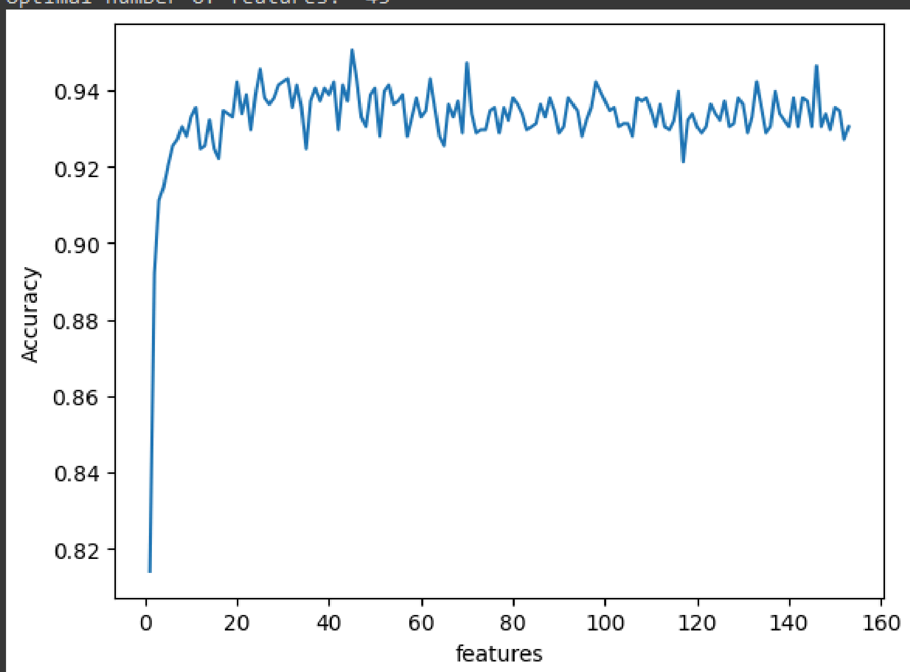
```
----- Feature selection using 50% of predictors -----
Selected features:  ['x0' 'x2' 'x5' 'x7' 'x8' 'x9' 'x10' 'x11' 'x12' 'x15' 'x16' 'x18' 'x19'
 'x20' 'x23' 'x24' 'x25' 'x26' 'x27' 'x28' 'x29' 'x30' 'x32' 'x35' 'x37'
 'x38' 'x41' 'x42' 'x44' 'x46' 'x47' 'x48' 'x50' 'x51' 'x52' 'x56' 'x59'
 'x61' 'x63' 'x65' 'x67' 'x73' 'x75' 'x76' 'x78' 'x79' 'x80' 'x81' 'x82'
 'x83' 'x85' 'x90' 'x91' 'x93' 'x97' 'x101' 'x102' 'x106' 'x109' 'x110'
 'x111' 'x119' 'x121' 'x123' 'x124' 'x128' 'x129' 'x130' 'x132' 'x135'
 'x137' 'x138' 'x139' 'x144' 'x146' 'x151']
              precision    recall  f1-score   support

         1.0       0.86      0.87      0.87       299
         2.0       0.96      0.95      0.96       895

    accuracy                           0.93      1194
   macro avg       0.91      0.91      0.91      1194
weighted avg       0.93      0.93      0.93      1194

----- Optimal selection of number of features -----
---- n features = 1
ACC: 0.8140642030870925
---- n features = 2
ACC: 0.8919798881895854
---- n features = 3
ACC: 0.9112337822158152
---- n features = 4
ACC: 0.9145880946520869
---- n features = 5
ACC: 0.9204423191870891
---- n features = 6
ACC: 0.9254562075876376
---- n features = 7
ACC: 0.9271474280088604
---- n features = 8
ACC: 0.9304911922928166
---- n features = 9
ACC: 0.9279631517879118
---- n features = 10
ACC: 0.9329911043915475
---- n features = 11
ACC: 0.9355191448964522
---- n features = 12
ACC: 0.9246229035547273
---- n features = 13
ACC: 0.925435111283007
---- n features = 14
ACC: 0.9321683485109524
---- n features = 15
```

```
---- n features = 144
ACC: 0.9371787208607293
---- n features = 145
ACC: 0.9304806441405014
---- n features = 146
ACC: 0.9464118701874055
---- n features = 147
ACC: 0.9304806441405015
---- n features = 148
ACC: 0.9338173763229142
---- n features = 149
ACC: 0.9296543722091346
---- n features = 150
ACC: 0.9355191448964524
---- n features = 151
ACC: 0.9346752927112268
---- n features = 152
ACC: 0.9271298477550017
---- n features = 153
ACC: 0.9304841601912731
Optimal number of features:  45
```




```
Selected features:  ['x0' 'x2' 'x5' 'x7' 'x8' 'x9' 'x10' 'x11' 'x12' 'x16' 'x18' 'x19' 'x24'
 'x26' 'x27' 'x28' 'x29' 'x35' 'x37' 'x41' 'x46' 'x50' 'x56' 'x59' 'x61'
 'x65' 'x67' 'x73' 'x75' 'x79' 'x80' 'x82' 'x85' 'x91' 'x97' 'x101' 'x102'
 'x106' 'x109' 'x111' 'x119' 'x121' 'x132' 'x135' 'x138']
```

1. ¿Qué pasa si no se considera el problema de tener datos desbalanceados para este caso? ¿Por qué?

Si no se considera el problema de desbalanceo, el modelo va a tener una gran tendencia a predecir la clase más representada, y a pesar de que va a tener una gran exactitud, puede tender a ignorar la clase que es minoria, y en temas como enfermedades, esta minoria puede ser en realidad el enfoque de nuetro modelo, por lo que de no balancear los datos el modelo aunque exacto no nos serviríá.

2. De todos los clasificadores, ¿cuál o cuáles consideras que son adecuados para los datos? ¿Qué propiedades tienen dichos modelos que los hacen apropiados para los datos? Argumenta tu respuesta.

Las SVMs, los discriminantes, tanto lineal como multiclase, y la regresión logistica (de scikitlearn) fueron los que ofrecieron mejores resultados.

3. ¿Es posible reducir la dimensionalidad del problema sin perder rendimiento en el modelo? ¿Por qué?

Sí, mediante una selección de características adecuada al problema. Evitando las cracterísticas con menor o nulo impacto en el resultado del modelo.

4. ¿Qué método de selección de características consideras el más adecuado para este caso? ¿Por qué?

El metodo secuencial me parece mas apropiado devido a la manera en la que va agregando y/o eliminando caracteristicas. De la misma manera el metodo recursivo me parece buena elección. Y a pesar de que el metodo filter es el más sencillo, da buena presición de resultados a una fracción del tiempo de ejecución.

5. Si quisieras mejorar el rendimiento de tus modelos, ¿qué más se podría hacer?

Realizar un ajuste de hiperparametros (ej.2). Reutilizar codigo en la medida de lo posible. Implementar mejores algoritmos de balanceo.