problemasdeclasificacion

September 10, 2024

[36]:
```python
import numpy as np

usecols = [0] + list(range(2, 128))

data = np.loadtxt(r"C:\Users\pfeli\Downloads\ACT2MACHINE\P1_5.txt",␣
↪delimiter='\t', usecols=usecols)

print(data[:2])
```

```
[[ 1.00000000e+00  6.77864488e+00  5.55481049e+00  4.93593256e+00
   5.63911404e+00  6.60346331e+00  6.30927847e+00  4.73882871e+00
   3.11004331e+00  2.16768246e+00  1.73461711e+00  1.70074842e+00
   2.35052930e+00  3.50076735e+00  4.17348336e+00  3.60129841e+00
   2.24598191e+00  1.25768327e+00  1.05769917e+00  1.07004257e+00
   8.05183980e-01  5.49397747e-01  7.32252928e-01  1.13320903e+00
   1.22713975e+00  1.02438236e+00  1.00171792e+00  1.25287461e+00
   1.21043665e+00  4.05059624e-01 -7.77141929e-01 -1.36373311e+00
  -8.44658594e-01  1.91734390e-01  6.15292087e-01 -6.85421779e-02
  -1.22317032e+00 -1.79026610e+00 -1.38311473e+00 -6.22125576e-01
  -4.50247795e-01 -1.13316081e+00 -1.99565095e+00 -2.20865846e+00
  -1.74388793e+00 -1.35768634e+00 -1.61972164e+00 -2.23885228e+00
  -2.54020102e+00 -2.38626222e+00 -2.24734024e+00 -2.40558162e+00
  -3.03471920e+00 -2.77938079e+00 -2.69713526e+00 -2.89331401e+00
  -2.99389066e+00 -2.81447866e+00 -2.57756118e+00 -2.43577240e+00
  -2.17549159e+00 -1.57361264e+00 -7.96674389e-01 -2.31803299e-01
  -5.16225566e-02 -1.23316455e-01 -2.64563699e-01 -4.62394198e-01
  -7.40166673e-01 -8.70541748e-01 -5.07675887e-01  2.53885590e-01
   7.62692522e-01  6.02376855e-01  2.26719603e-01  3.62126267e-01
   9.71732468e-01  1.33868636e+00  1.17237700e+00  1.04300212e+00
   1.45695489e+00  1.97172697e+00  1.75113274e+00  7.65384185e-01
  -1.35461199e-01 -3.31484898e-01 -8.05638571e-02  9.06039275e-02
   1.40973609e-01  2.92339379e-01  4.21667322e-01  1.78068702e-01
  -3.83725604e-01 -7.25730039e-01 -4.82455212e-01  1.11195346e-01
   5.60547918e-01  6.38932471e-01  4.42775182e-01  1.55963252e-01
  -3.52226954e-02  9.02042652e-04  1.38058250e-01 -3.59369229e+00
  -3.56157940e+00 -3.36776631e+00 -3.03752452e+00 -2.34591518e+00
  -1.41893210e+00 -7.87569326e-01 -6.69998499e-01 -6.18081902e-01
  -1.75003972e-01  4.22023282e-01  5.56799357e-01  8.95803993e-02
```

```
 -4.69966874e-01 -6.06846729e-01 -3.66854781e-01 -1.74912428e-01
 -2.66942131e-01 -4.23644741e-01 -2.38625920e-01  3.94337748e-01
  1.07648755e+00  1.25638253e+00  8.22914691e-01]
[ 1.00000000e+00 -3.54725174e-01 -2.10143440e-01 -3.37064661e-01
 -6.13977799e-01 -7.25108531e-01 -5.09282019e-01 -1.54145540e-01
  6.26510071e-02  1.02172384e-01  8.12152645e-02  7.50502223e-02
  2.43515558e-01  9.00655313e-01  2.01063259e+00  2.76404244e+00
  2.24797879e+00  6.39480323e-01 -7.51312425e-01 -8.50889989e-01
  1.55277001e-01  1.32523653e+00  1.99151769e+00  1.99677938e+00
  1.39295949e+00  4.53952973e-01 -2.63797950e-01 -4.74796282e-01
 -6.17868820e-01 -1.22666238e+00 -1.99136496e+00 -2.02920873e+00
 -1.13645868e+00 -1.97947396e-01 -1.29138711e-01 -8.49466837e-01
 -1.57947531e+00 -1.80392087e+00 -1.61103887e+00 -1.31643170e+00
 -1.08423056e+00 -8.28489826e-01 -3.34581265e-01  4.38138697e-01
  1.16302513e+00  1.47807165e+00  1.51871166e+00  1.72771550e+00
  2.05616266e+00  1.87689834e+00  9.92565025e-01  2.24455544e-01
 -2.45371203e-01 -1.55382158e-01  3.89297515e-01  1.08329468e+00
  1.12252122e+00  3.92803117e-01 -1.82852045e-01  1.57658630e-01
  9.39569130e-01  1.07226022e+00  2.80853323e-01 -6.81471456e-01
 -1.15523062e+00 -1.27942773e+00 -1.40165984e+00 -1.33472129e+00
 -6.68323882e-01  4.05962745e-01  1.10460728e+00  9.39058168e-01
  2.66096460e-01 -1.89062263e-01 -1.58033711e-01  5.50559335e-02
  1.54589956e-01  2.40520468e-01  5.08598011e-01  8.00625582e-01
  7.34255456e-01  2.33874373e-01 -3.25137324e-01 -5.95982645e-01
 -6.39542142e-01 -6.57018541e-01 -5.65582807e-01 -1.71358347e-01
  3.22661120e-01  3.87540712e-01 -1.76999744e-01 -9.58986988e-01
 -1.40853250e+00 -1.35963011e+00 -9.96919085e-01 -5.33243924e-01
 -1.02015108e-01  2.17245663e-01  4.75699058e-01  8.29400691e-01
  1.26931847e+00  1.53942534e+00  1.52447799e+00 -8.28550394e-01
 -3.53545307e-01  6.92506520e-02  4.22247925e-01  6.75601942e-01
  5.09527671e-01 -1.85811180e-01 -7.99394573e-01 -6.09410683e-01
  1.59701272e-01  4.94181699e-01 -6.70766793e-02 -8.15534322e-01
 -8.45637665e-01 -1.46853458e-01  6.36752585e-01  1.11270777e+00
  1.32821956e+00  1.31554324e+00  9.88432481e-01  4.20049137e-01
 -1.68787007e-01 -7.21103044e-01 -1.30752014e+00]]
```

[40]:

```python
import numpy as np

data = np.loadtxt(r"C:\Users\pfeli\Downloads\ACT2MACHINE\P1_5.txt",␣
 ↪delimiter='\t', usecols=usecols)

# Separar la clase (y) y las variables predictoras (x)
y = data[:, 0]
x = data[:, 1:]

print(y)
print(x)
```

```
[1. 1. 1. … 2. 2. 2.]
[[ 6.77864488  5.55481049  4.93593256 …  1.07648755  1.25638253
   0.82291469]
 [-0.35472517 -0.21014344 -0.33706466 … -0.16878701 -0.72110304
  -1.30752014]
 [ 1.7277155   2.05616266  1.87689834 …  0.84869694  0.64432822
  -0.13647387]
 …
 [ 2.07424806  2.32135436  1.79695161 … -1.15871772 -0.94376892
  -0.45962386]
 [-0.05645914  0.70658034  0.61369947 …  0.75021778  1.59629193
   2.28915839]
 [-0.04833282 -0.95008854 -0.84017033 … -1.35206151 -1.17221605
  -0.51677338]]
```

[41]:
```python
clases = data[:, 0]

unique, counts = np.unique(clases, return_counts=True)

for clase, count in zip(unique, counts):
    print(f"Clase {int(clase)}: {count} elementos")
```

```
Clase 1: 281 elementos
Clase 2: 1689 elementos
```

[46]:
```python
from sklearn.svm import SVC
from sklearn.model_selection import StratifiedKFold
from sklearn.metrics import classification_report
import numpy as np
import random

# Oversampling
print("----- Oversampling -----")
kf = StratifiedKFold(n_splits=5, shuffle=True)
clf = SVC(kernel='linear')

cv_y_test = []
cv_y_pred = []

for train_index, test_index in kf.split(x, y):
    x_train = x[train_index, :]
    y_train = y[train_index]

    # Separar las clases
    x1 = x_train[y_train == 1, :]
    y1 = y_train[y_train == 1]
    x2 = x_train[y_train == 2, :]
```

```
    y2 = y_train[y_train == 2]

    # Aplicar sobremuestreo a la clase minoritaria
    ind = random.choices(range(len(y1)), k=len(y2))
    x_sub = np.concatenate((x1[ind, :], x2), axis=0)
    y_sub = np.concatenate((y1[ind], y2), axis=0)

    unique, counts = np.unique(y_sub, return_counts=True)
    print(f"Distribución de clases después del sobremuestreo: {dict(zip(unique,␣
 ↪counts))}")

    # Entrenar el modelo con el conjunto sobremuestreado
    clf.fit(x_sub, y_sub)

    # Evaluar con los datos de prueba
    x_test = x[test_index, :]
    y_test = y[test_index]
    y_pred = clf.predict(x_test)

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

# Imprimir el informe de clasificación final
print(classification_report(np.concatenate(cv_y_test), np.
 ↪concatenate(cv_y_pred)))
```

```
----- Oversampling -----
Distribución de clases después del sobremuestreo: {1.0: 1352, 2.0: 1352}
Distribución de clases después del sobremuestreo: {1.0: 1351, 2.0: 1351}
Distribución de clases después del sobremuestreo: {1.0: 1351, 2.0: 1351}
Distribución de clases después del sobremuestreo: {1.0: 1351, 2.0: 1351}
Distribución de clases después del sobremuestreo: {1.0: 1351, 2.0: 1351}
              precision    recall  f1-score   support

         1.0       0.53      0.84      0.65       281
         2.0       0.97      0.88      0.92      1689

    accuracy                           0.87      1970
   macro avg       0.75      0.86      0.79      1970
weighted avg       0.91      0.87      0.88      1970
```

[68]:
```
#Evalúa al menos 8 modelos de clasificación distintos utilizando validación␣
 ↪cruzada, y determina cuál de ellos es el más efectivo.

import numpy as np
from sklearn.model_selection import StratifiedKFold
```

```python
from sklearn.neighbors import KNeighborsClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis,␣
    ↪QuadraticDiscriminantAnalysis
from sklearn.svm import SVC
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import classification_report
import random

# Lista de clasificadores a evaluar
classifiers = {
    "KNN": KNeighborsClassifier(),
    "Árbol de Decisión": DecisionTreeClassifier(),
    "Regresión Logística": LogisticRegression(max_iter=1000),
    "LDA": LinearDiscriminantAnalysis(),
    "QDA": QuadraticDiscriminantAnalysis(),
    "SVM": SVC(kernel='linear'),
    "Bayesiano Ingenuo Cuadrático": QuadraticDiscriminantAnalysis(),
    "Bayesiano Ingenuo Lineal": LinearDiscriminantAnalysis(),
}

# Validación cruzada estratificada
kf = StratifiedKFold(n_splits=5, shuffle=True)

# Sobremuestreo
def sobremuestreo(x_train, y_train):
    x1 = x_train[y_train == 1, :]
    y1 = y_train[y_train == 1]
    x2 = x_train[y_train == 2, :]
    y2 = y_train[y_train == 2]
    ind = random.choices(range(len(y1)), k=len(y2))
    x_sub = np.concatenate((x1[ind, :], x2), axis=0)
    y_sub = np.concatenate((y1[ind], y2), axis=0)
    return x_sub, y_sub

# Almacenar resultados
results = {}

# Evaluación de modelos
for name, clf in classifiers.items():
    print(f"Evaluando {name}")
    cv_y_test = []
    cv_y_pred = []

    for train_index, test_index in kf.split(x, y):
        x_train = x[train_index, :]
```

```python
        y_train = y[train_index]

        # Sobremuestreo
        x_sub, y_sub = sobremuestreo(x_train, y_train)

        # Entrenamiento
        clf.fit(x_sub, y_sub)

        # Predicción
        x_test = x[test_index, :]
        y_test = y[test_index]
        y_pred = clf.predict(x_test)

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

    # Resultados
    report = classification_report(np.concatenate(cv_y_test), np.
 ↪concatenate(cv_y_pred), output_dict=True)
    results[name] = report

    print(f"Resultados para {name}")
    print(classification_report(np.concatenate(cv_y_test), np.
 ↪concatenate(cv_y_pred)))
    print("--------------------------------------------------\n")

for name, metrics in results.items():
    print(f"Modelo: {name}")
    print(f"Accuracy: {metrics['accuracy']:.4f}")
    print(f"Macro Avg F1-Score: {metrics['macro avg']['f1-score']:.4f}")
    print(f"Weighted Avg F1-Score: {metrics['weighted avg']['f1-score']:.4f}")
    print("--------------------------------------------------\n")
```

```
Evaluando KNN
Resultados para KNN
              precision    recall  f1-score   support

         1.0       0.29      0.61      0.39       281
         2.0       0.92      0.75      0.83      1689

    accuracy                           0.73      1970
   macro avg       0.61      0.68      0.61      1970
weighted avg       0.83      0.73      0.77      1970

--------------------------------------------------

Evaluando Árbol de Decisión
```

```
Resultados para Árbol de Decisión
              precision    recall  f1-score   support

         1.0       0.45      0.44      0.45       281
         2.0       0.91      0.91      0.91      1689

    accuracy                           0.84      1970
   macro avg       0.68      0.68      0.68      1970
weighted avg       0.84      0.84      0.84      1970

--------------------------------------------------

Evaluando Regresión Logística
Resultados para Regresión Logística
              precision    recall  f1-score   support

         1.0       0.52      0.80      0.63       281
         2.0       0.96      0.88      0.92      1689

    accuracy                           0.87      1970
   macro avg       0.74      0.84      0.77      1970
weighted avg       0.90      0.87      0.88      1970

--------------------------------------------------

Evaluando LDA
Resultados para LDA
              precision    recall  f1-score   support

         1.0       0.49      0.77      0.60       281
         2.0       0.96      0.87      0.91      1689

    accuracy                           0.85      1970
   macro avg       0.72      0.82      0.75      1970
weighted avg       0.89      0.85      0.87      1970

--------------------------------------------------

Evaluando QDA
```

```
C:\Users\pfeli\AppData\Local\Packages\PythonSoftwareFoundation.Python.3.10_qbz5n
2kfra8p0\LocalCache\local-packages\Python310\site-
packages\sklearn\metrics\_classification.py:1531: UndefinedMetricWarning:
Precision is ill-defined and being set to 0.0 in labels with no predicted
samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
C:\Users\pfeli\AppData\Local\Packages\PythonSoftwareFoundation.Python.3.10_qbz5n
2kfra8p0\LocalCache\local-packages\Python310\site-
```

```
packages\sklearn\metrics\_classification.py:1531: UndefinedMetricWarning:
Precision is ill-defined and being set to 0.0 in labels with no predicted
samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
C:\Users\pfeli\AppData\Local\Packages\PythonSoftwareFoundation.Python.3.10_qbz5n
2kfra8p0\LocalCache\local-packages\Python310\site-
packages\sklearn\metrics\_classification.py:1531: UndefinedMetricWarning:
Precision is ill-defined and being set to 0.0 in labels with no predicted
samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
C:\Users\pfeli\AppData\Local\Packages\PythonSoftwareFoundation.Python.3.10_qbz5n
2kfra8p0\LocalCache\local-packages\Python310\site-
packages\sklearn\metrics\_classification.py:1531: UndefinedMetricWarning:
Precision is ill-defined and being set to 0.0 in labels with no predicted
samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
C:\Users\pfeli\AppData\Local\Packages\PythonSoftwareFoundation.Python.3.10_qbz5n
2kfra8p0\LocalCache\local-packages\Python310\site-
packages\sklearn\metrics\_classification.py:1531: UndefinedMetricWarning:
Precision is ill-defined and being set to 0.0 in labels with no predicted
samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
C:\Users\pfeli\AppData\Local\Packages\PythonSoftwareFoundation.Python.3.10_qbz5n
2kfra8p0\LocalCache\local-packages\Python310\site-
packages\sklearn\metrics\_classification.py:1531: UndefinedMetricWarning:
Precision is ill-defined and being set to 0.0 in labels with no predicted
samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))

Resultados para QDA
              precision    recall  f1-score   support

         1.0       0.00      0.00      0.00       281
         2.0       0.86      1.00      0.92      1689

    accuracy                           0.86      1970
   macro avg       0.43      0.50      0.46      1970
weighted avg       0.74      0.86      0.79      1970

-------------------------------------------------

Evaluando SVM
Resultados para SVM
              precision    recall  f1-score   support

         1.0       0.51      0.83      0.63       281
         2.0       0.97      0.87      0.92      1689
```

```
    accuracy                           0.86      1970
   macro avg       0.74      0.85      0.78      1970
weighted avg       0.90      0.86      0.88      1970

--------------------------------------------------

Evaluando Bayesiano Ingenuo Cuadrático
Resultados para Bayesiano Ingenuo Cuadrático
              precision    recall  f1-score   support

         1.0       1.00      0.00      0.01       281
         2.0       0.86      1.00      0.92      1689

    accuracy                           0.86      1970
   macro avg       0.93      0.50      0.47      1970
weighted avg       0.88      0.86      0.79      1970

--------------------------------------------------

Evaluando Bayesiano Ingenuo Lineal
Resultados para Bayesiano Ingenuo Lineal
              precision    recall  f1-score   support

         1.0       0.51      0.79      0.62       281
         2.0       0.96      0.88      0.92      1689

    accuracy                           0.86      1970
   macro avg       0.74      0.83      0.77      1970
weighted avg       0.90      0.86      0.87      1970

--------------------------------------------------

Modelo: KNN
Accuracy: 0.7325
Macro Avg F1-Score: 0.6116
Weighted Avg F1-Score: 0.7665
--------------------------------------------------

Modelo: Árbol de Decisión
Accuracy: 0.8447
Macro Avg F1-Score: 0.6777
Weighted Avg F1-Score: 0.8435
--------------------------------------------------

Modelo: Regresión Logística
Accuracy: 0.8660
Macro Avg F1-Score: 0.7747
Weighted Avg F1-Score: 0.8772
```

```
--------------------------------------------------

Modelo: LDA
Accuracy: 0.8528
Macro Avg F1-Score: 0.7546
Weighted Avg F1-Score: 0.8656
--------------------------------------------------

Modelo: QDA
Accuracy: 0.8574
Macro Avg F1-Score: 0.4616
Weighted Avg F1-Score: 0.7915
--------------------------------------------------

Modelo: SVM
Accuracy: 0.8640
Macro Avg F1-Score: 0.7752
Weighted Avg F1-Score: 0.8762
--------------------------------------------------

Modelo: Bayesiano Ingenuo Cuadrático
Accuracy: 0.8579
Macro Avg F1-Score: 0.4653
Weighted Avg F1-Score: 0.7927
--------------------------------------------------

Modelo: Bayesiano Ingenuo Lineal
Accuracy: 0.8629
Macro Avg F1-Score: 0.7686
Weighted Avg F1-Score: 0.8742
--------------------------------------------------
```

SVM

[63]:
```python
import numpy as np
from sklearn.model_selection import StratifiedKFold
from sklearn.metrics import classification_report
from sklearn.preprocessing import StandardScaler

# Función sigmoide
def sigmoid(z):
    return 1 / (1 + np.exp(-z))

# Regresión Logística
class LogisticRegression:
    def __init__(self, learning_rate=0.01, n_iterations=1000):
        self.learning_rate = learning_rate
```

```python
        self.n_iterations = n_iterations
        self.weights = None
        self.bias = None

    def fit(self, X, y):
        n_samples, n_features = X.shape
        self.weights = np.zeros(n_features)
        self.bias = 0

        for _ in range(self.n_iterations):
            # Predicciones
            linear_model = np.dot(X, self.weights) + self.bias
            y_pred = sigmoid(linear_model)

            # Gradientes
            dw = (1 / n_samples) * np.dot(X.T, (y_pred - y))
            db = (1 / n_samples) * np.sum(y_pred - y)

            # Actualización de parámetros
            self.weights -= self.learning_rate * dw
            self.bias -= self.learning_rate * db

    def predict(self, X):
        linear_model = np.dot(X, self.weights) + self.bias
        y_pred = sigmoid(linear_model)
        return np.round(y_pred)

# Normalizar datos
scaler = StandardScaler()
x = scaler.fit_transform(x)

# Validación cruzada estratificada
kf = StratifiedKFold(n_splits=5, shuffle=True)

# Evaluación de modelo
print("Evaluando Regresión Logística desde cero")
cv_y_test = []
cv_y_pred = []

for train_index, test_index in kf.split(x, y):
    x_train, x_test = x[train_index], x[test_index]
    y_train, y_test = y[train_index], y[test_index]

    # Inicializar y entrenar el modelo
    model = LogisticRegression(learning_rate=0.01, n_iterations=1000)
    model.fit(x_train, y_train)
```

```
    # Predicción
    y_pred = model.predict(x_test)

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

# Resultados
print("Resultados para Regresión Logística desde cero")
print(classification_report(np.concatenate(cv_y_test), np.
 ↪concatenate(cv_y_pred)))
print("--------------------------------------------------\n")
```

```
Evaluando Regresión Logística desde cero
Resultados para Regresión Logística desde cero
              precision    recall  f1-score   support

         0.0       0.00      0.00      0.00         0
         1.0       0.13      0.88      0.22       281
         2.0       0.00      0.00      0.00      1689

    accuracy                           0.13      1970
   macro avg       0.04      0.29      0.07      1970
weighted avg       0.02      0.13      0.03      1970

--------------------------------------------------


C:\Users\pfeli\AppData\Local\Packages\PythonSoftwareFoundation.Python.3.10_qbz5n
2kfra8p0\LocalCache\local-packages\Python310\site-
packages\sklearn\metrics\_classification.py:1531: UndefinedMetricWarning:
Precision is ill-defined and being set to 0.0 in labels with no predicted
samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
C:\Users\pfeli\AppData\Local\Packages\PythonSoftwareFoundation.Python.3.10_qbz5n
2kfra8p0\LocalCache\local-packages\Python310\site-
packages\sklearn\metrics\_classification.py:1531: UndefinedMetricWarning: Recall
is ill-defined and being set to 0.0 in labels with no true samples. Use
`zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
C:\Users\pfeli\AppData\Local\Packages\PythonSoftwareFoundation.Python.3.10_qbz5n
2kfra8p0\LocalCache\local-packages\Python310\site-
packages\sklearn\metrics\_classification.py:1531: UndefinedMetricWarning:
Precision is ill-defined and being set to 0.0 in labels with no predicted
samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
C:\Users\pfeli\AppData\Local\Packages\PythonSoftwareFoundation.Python.3.10_qbz5n
2kfra8p0\LocalCache\local-packages\Python310\site-
packages\sklearn\metrics\_classification.py:1531: UndefinedMetricWarning: Recall
```

```
is ill-defined and being set to 0.0 in labels with no true samples. Use
`zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
C:\Users\pfeli\AppData\Local\Packages\PythonSoftwareFoundation.Python.3.10_qbz5n
2kfra8p0\LocalCache\local-packages\Python310\site-
packages\sklearn\metrics\_classification.py:1531: UndefinedMetricWarning:
Precision is ill-defined and being set to 0.0 in labels with no predicted
samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
C:\Users\pfeli\AppData\Local\Packages\PythonSoftwareFoundation.Python.3.10_qbz5n
2kfra8p0\LocalCache\local-packages\Python310\site-
packages\sklearn\metrics\_classification.py:1531: UndefinedMetricWarning: Recall
is ill-defined and being set to 0.0 in labels with no true samples. Use
`zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
```

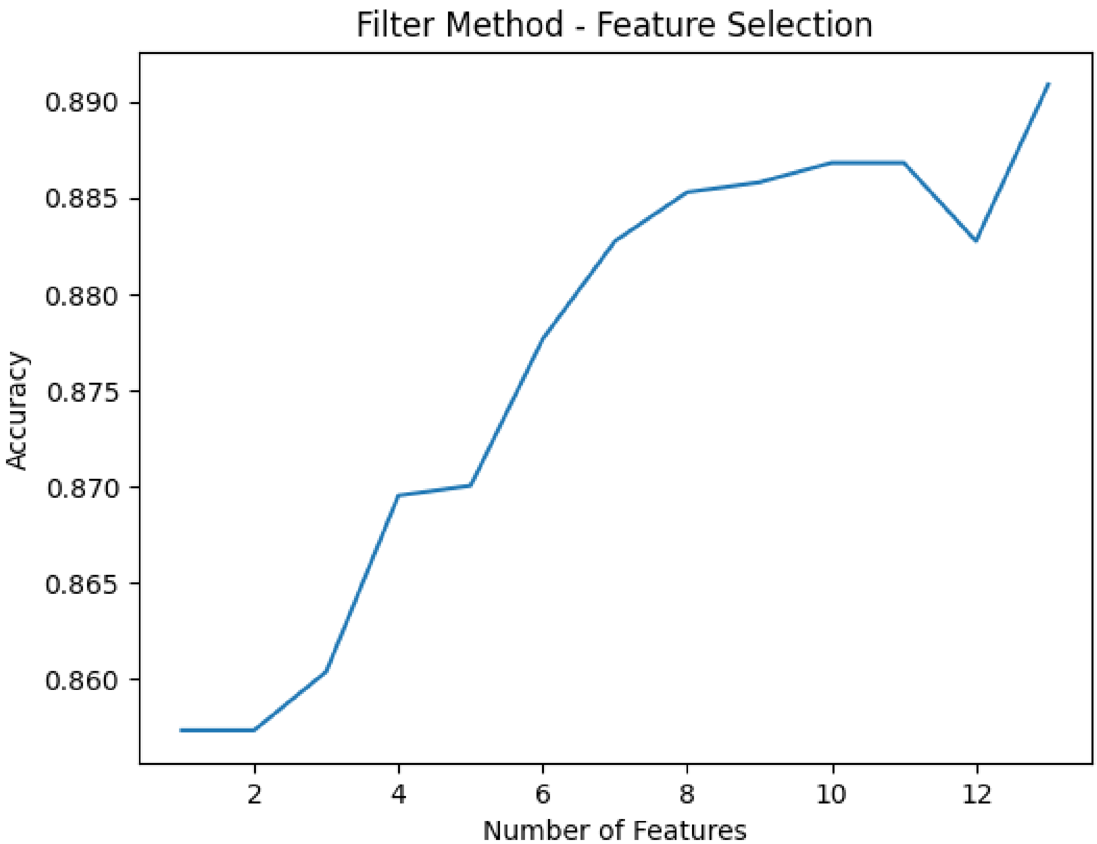
[64]:

```python
import numpy as np
from sklearn.model_selection import StratifiedKFold
from sklearn.svm import SVC
from sklearn.feature_selection import SelectKBest, f_classif
from sklearn.metrics import accuracy_score
import matplotlib.pyplot as plt


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

n_feats = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13]
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):
        x_train = x[train_index, :]
        y_train = y[train_index]

        clf_cv = SVC(kernel='linear')
        fselection_cv = SelectKBest(f_classif, k=n_feat)
        x_train_selected = fselection_cv.fit_transform(x_train, y_train)
        clf_cv.fit(x_train_selected, y_train)

        x_test = x[test_index, :]
        y_test = y[test_index]
        x_test_selected = fselection_cv.transform(x_test)
```

```
        y_pred = clf_cv.predict(x_test_selected)

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

    acc = np.mean(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("Number of Features")
plt.ylabel("Accuracy")
plt.title("Filter Method - Feature Selection")
plt.show()

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

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

```
----- Optimal selection of number of features -----
---- n features = 1
ACC: 0.8573604060913705
---- n features = 2
ACC: 0.8573604060913705
---- n features = 3
ACC: 0.8604060913705585
---- n features = 4
ACC: 0.8695431472081218
---- n features = 5
ACC: 0.8700507614213198
---- n features = 6
ACC: 0.8776649746192893
---- n features = 7
ACC: 0.882741116751269
---- n features = 8
ACC: 0.8852791878172589
---- n features = 9
ACC: 0.8857868020304569
```

```
---- n features = 10
ACC: 0.8868020304568528
---- n features = 11
ACC: 0.8868020304568528
---- n features = 12
ACC: 0.882741116751269
---- n features = 13
ACC: 0.8908629441624365
Optimal number of features:  13
```

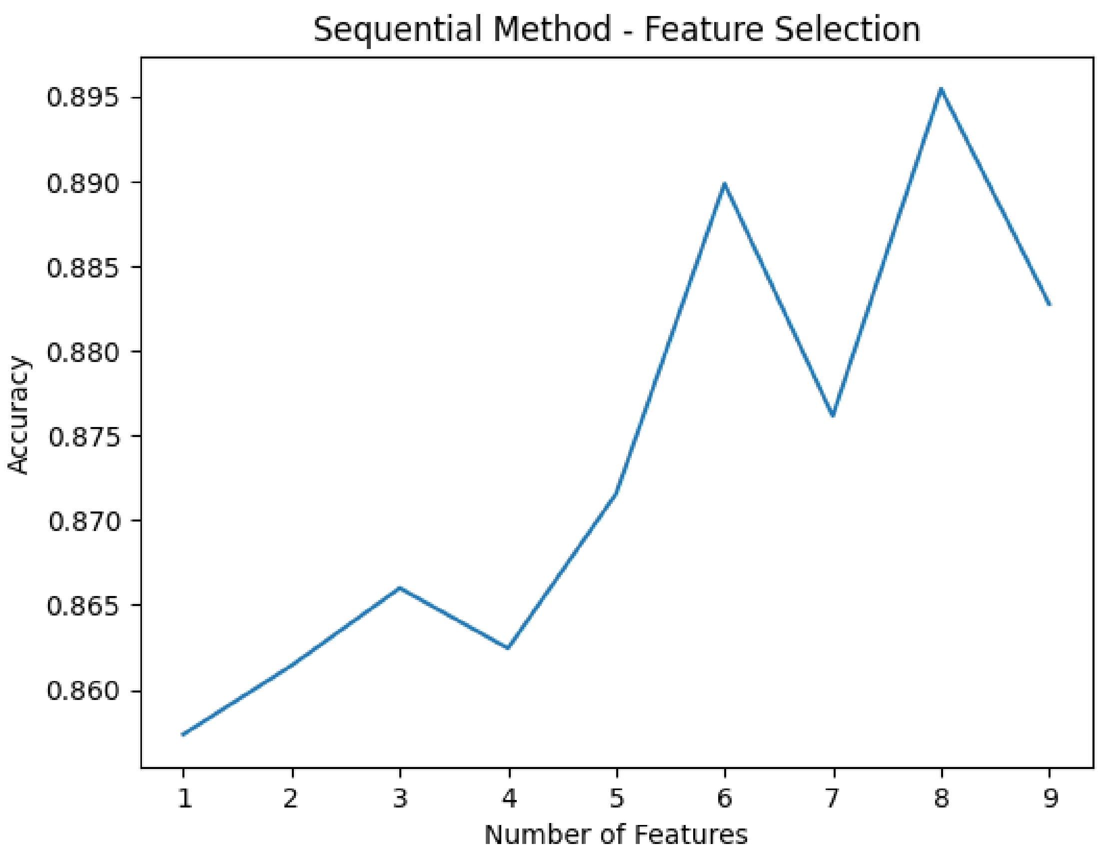



```
Selected features:  ['x8' 'x9' 'x10' 'x11' 'x75' 'x76' 'x77' 'x78' 'x79' 'x87'
'x88' 'x89'
 'x90']
```

[65]:
```python
from sklearn.feature_selection import SequentialFeatureSelector

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

n_feats = [1, 2, 3, 4, 5, 6, 7, 8, 9]
acc_nfeat = []
```

```python
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):
        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)
        x_train_selected = fselection_cv.fit_transform(x_train, y_train)
        clf_cv.fit(x_train_selected, y_train)

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

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

    acc = np.mean(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("Number of Features")
plt.ylabel("Accuracy")
plt.title("Sequential Method - Feature Selection")
plt.show()

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

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

```
----- Optimal selection of number of features -----
---- n features = 1
ACC: 0.8573604060913705
---- n features = 2
ACC: 0.8614213197969542
---- n features = 3
ACC: 0.865989847715736
---- n features = 4
ACC: 0.8624365482233503
---- n features = 5
ACC: 0.8715736040609137
---- n features = 6
ACC: 0.8898477157360407
---- n features = 7
ACC: 0.8761421319796954
---- n features = 8
ACC: 0.8954314720812182
---- n features = 9
ACC: 0.882741116751269
Optimal number of features:  8
```

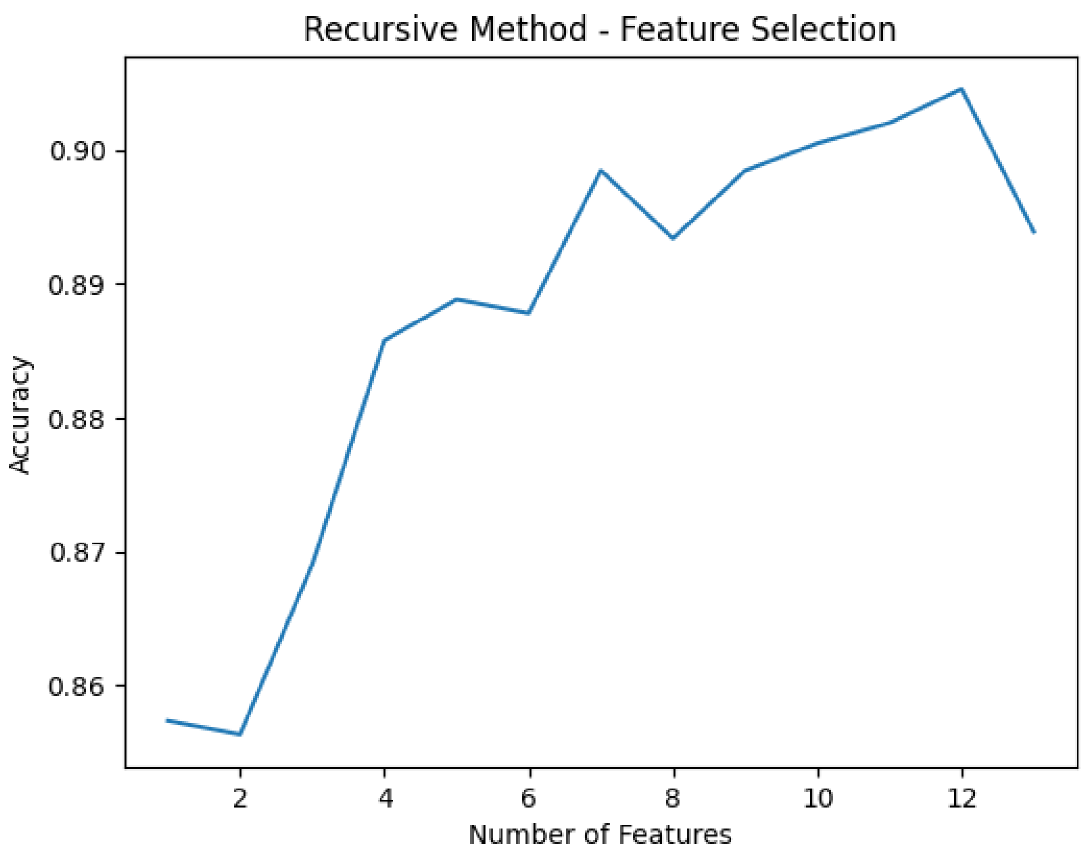


```
Selected features:  ['x0' 'x1' 'x2' 'x9' 'x10' 'x14' 'x23' 'x88']
```

[66]:
```
from sklearn.feature_selection import RFE

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

n_feats = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13]
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):
        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)
        x_train_selected = fselection_cv.fit_transform(x_train, y_train)
        clf_cv.fit(x_train_selected, y_train)

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

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

    acc = np.mean(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("Number of Features")
plt.ylabel("Accuracy")
plt.title("Recursive Method - Feature Selection")
plt.show()
```

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

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

```
----- Optimal selection of number of features -----
---- n features = 1
ACC: 0.8573604060913705
---- n features = 2
ACC: 0.8563451776649746
---- n features = 3
ACC: 0.8690355329949238
---- n features = 4
ACC: 0.8857868020304569
---- n features = 5
ACC: 0.8888324873096447
---- n features = 6
ACC: 0.8878172588832488
---- n features = 7
ACC: 0.898477157360406
---- n features = 8
ACC: 0.8934010152284262
---- n features = 9
ACC: 0.898477157360406
---- n features = 10
ACC: 0.900507614213198
---- n features = 11
ACC: 0.9020304568527919
---- n features = 12
ACC: 0.9045685279187816
---- n features = 13
ACC: 0.8939086294416242
Optimal number of features:  12
```



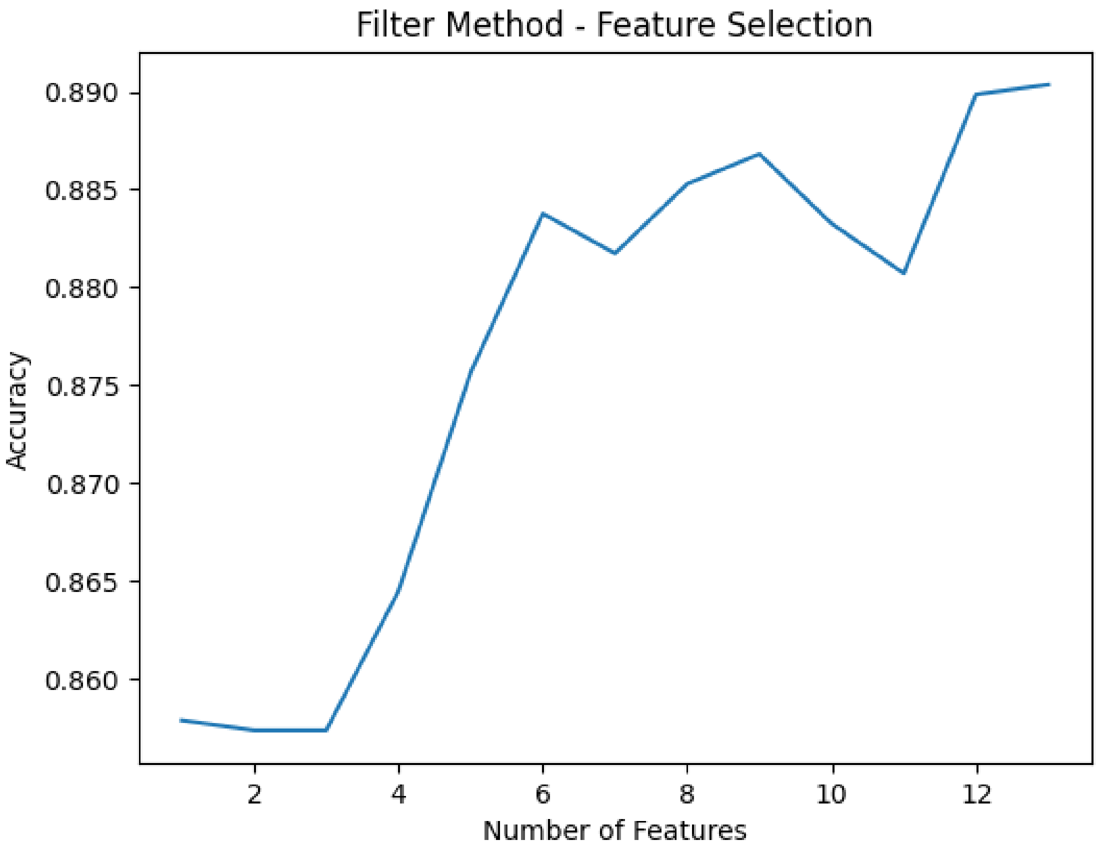

```
Selected features:  ['x0' 'x3' 'x9' 'x14' 'x23' 'x31' 'x41' 'x63' 'x65' 'x89'
'x92' 'x121']
```

[72]:
```
import numpy as np
from sklearn.model_selection import StratifiedKFold, cross_val_score
from sklearn.svm import SVC
from sklearn.feature_selection import SelectKBest, f_classif
from sklearn.metrics import accuracy_score, classification_report,␣
 ↪confusion_matrix
import matplotlib.pyplot as plt

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

n_feats = list(range(1, 14))
acc_nfeat = []

for n_feat in n_feats:
    print(f'---- n features = {n_feat}')

    acc_cv = []
```

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

    for train_index, test_index in kf.split(x, y):
        x_train, x_test = x[train_index, :], x[test_index, :]
        y_train, y_test = y[train_index], y[test_index]

        fselection_cv = SelectKBest(f_classif, k=n_feat)
        x_train_selected = fselection_cv.fit_transform(x_train, y_train)

        clf_cv = SVC(kernel='linear')
        clf_cv.fit(x_train_selected, y_train)

        x_test_selected = fselection_cv.transform(x_test)
        y_pred = clf_cv.predict(x_test_selected)

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

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

    print(f'ACC: {acc}')

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

plt.plot(n_feats, acc_nfeat)
plt.xlabel("Number of Features")
plt.ylabel("Accuracy")
plt.title("Filter Method - Feature Selection")
plt.show()

print("----- Adjusting the model with selected features -----")

fselection = SelectKBest(f_classif, k=opt_features)
x_transformed = fselection.fit_transform(x, y)

clf = SVC(kernel='linear')
clf.fit(x_transformed, y)

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

print("----- Evaluating the final model -----")
kf = StratifiedKFold(n_splits=5, shuffle=True)
scores = cross_val_score(clf, x_transformed, y, cv=kf, scoring='accuracy')
```

```python
print(f"Mean cross-validated accuracy: {np.mean(scores):.4f}")

y_pred = cross_val_score(clf, x_transformed, y, cv=kf, scoring='accuracy')
print("\nClassification Report:\n", classification_report(y, clf.
 ↪predict(x_transformed)))
```

```
----- Optimal selection of number of features -----
---- n features = 1
ACC: 0.8578680203045685
---- n features = 2
ACC: 0.8573604060913705
---- n features = 3
ACC: 0.8573604060913705
---- n features = 4
ACC: 0.8644670050761422
---- n features = 5
ACC: 0.8756345177664974
---- n features = 6
ACC: 0.8837563451776651
---- n features = 7
ACC: 0.881725888324873
---- n features = 8
ACC: 0.8852791878172589
---- n features = 9
ACC: 0.8868020304568528
---- n features = 10
ACC: 0.883248730964467
---- n features = 11
ACC: 0.8807106598984772
---- n features = 12
ACC: 0.8898477157360405
---- n features = 13
ACC: 0.8903553299492385
Optimal number of features: 13
```




```
----- Adjusting the model with selected features -----
Selected features:  ['x8' 'x9' 'x10' 'x11' 'x75' 'x76' 'x77' 'x78' 'x79' 'x87'
'x88' 'x89'
 'x90']
----- Evaluating the final model -----
Mean cross-validated accuracy: 0.8848

Classification Report:
              precision    recall  f1-score   support

         1.0       0.86      0.25      0.38       281
         2.0       0.89      0.99      0.94      1689

    accuracy                           0.89      1970
   macro avg       0.88      0.62      0.66      1970
weighted avg       0.88      0.89      0.86      1970
```

¿Qué pasa si no se considera el problema de tener datos desbalanceados para este caso? ¿Por qué? De todos los clasificadores, ¿cuál o cuales consideras que son adecuados para los datos? ¿Qué propiedades tienen dichos modelos que los hacen apropiados para los datos? Argumenta tu respuesta. ¿Es posibles reducir la dimensionalidad del problema sin perder rendimiento en el modelo? ¿Por qué? ¿Qué método de selección de características consideras el más adecuado para este caso? ¿Por qué? Si quisieras mejorar el rendimiento de tus modelos, ¿qué más se podría hacer?

Si no se considera el problema de tener datos desbalanceados para este caso, lo que pasaría es que el modelo clasificaría incorrectamente la clase minoritaria y favorecería a la clase mayoritaria, ya que los algoritmos de clasificación tienden a priorizar la precisión y en este caso aprendería a predecir casi siempre la clase mayoritaria para lograr alcanzar esta precisión.

De todos los modelos de clasificación evaluados, los más adecuados para los datos serían SVM y regresión logística, ya que ambos tienen un buen equilibrio entre la precisión y el f1-score, lo que los hacen apropiados para los datos es que estos modelos tienen la capacidad de generalizar bien con diferentes tipos de distribuciones.

Si es posible reducir la dimensionalidad, pues utilizando las tecnicas adecuadas se puede identificar el número óptimo de características y así eliminar aquellas características irrelevantes o redundantes y al mismo tiempo lograr un modelo más sencillo, preciso y menos propenso a sobreajustarse.

De los tres métodos de selección de características considero que el recursivo es el más adecuado, ya que este logró una precisión de 0.9, sin embargo fue un poco tardado de correr, pues se tardó un poco más de media hora.

Otra maneras de mejorar el rendimiento de los modelos además de la mencionada anteriormente, podría ser la optimización de hiperparámetros.