problemasdeclasificacion2

September 10, 2024

Paola Félix Torres A00227869

[1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
```

[2]:
```
data = np.loadtxt('C:\\Users\\pfeli\\Downloads\\ACT2MACHINE\\M_4.txt')
```

[7]:
```
x = data[:, 2:]  # Características
y = data[:, 0]  # Clases
```

[8]:
```
unique, counts = np.unique(y, return_counts=True)
class_distribution = dict(zip(unique, counts))

print("Distribución de clases:")
for cls, count in class_distribution.items():
    print(f"Clase {int(cls)}: {count} muestras")
```

```
Distribución de clases:
Clase 1: 90 muestras
Clase 2: 90 muestras
Clase 3: 90 muestras
Clase 4: 90 muestras
Clase 5: 90 muestras
Clase 6: 90 muestras
Clase 7: 90 muestras
```

No es necesario balancear los datos

[9]:
```
import numpy as np
from sklearn.model_selection import StratifiedKFold
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.ensemble import RandomForestClassifier
```

```python
from sklearn.metrics import classification_report

# 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 Lineal": GaussianNB(),
    "Random Forest": RandomForestClassifier()
}

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

# 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, :]
        y_train = y[train_index]

        # Entrenamiento
        clf.fit(x_train, y_train)

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

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

```
         1.0       0.91      0.94      0.93        90
         2.0       0.64      0.87      0.74        90
         3.0       0.92      0.81      0.86        90
         4.0       0.85      0.70      0.77        90
         5.0       0.94      0.84      0.89        90
         6.0       0.94      0.86      0.90        90
         7.0       0.88      0.97      0.92        90

    accuracy                           0.86       630
   macro avg       0.87      0.86      0.86       630
weighted avg       0.87      0.86      0.86       630

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

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

         1.0       0.79      0.71      0.75        90
         2.0       0.54      0.51      0.53        90
         3.0       0.71      0.72      0.72        90
         4.0       0.61      0.64      0.63        90
         5.0       0.63      0.66      0.64        90
         6.0       0.62      0.68      0.65        90
         7.0       0.86      0.83      0.85        90

    accuracy                           0.68       630
   macro avg       0.68      0.68      0.68       630
weighted avg       0.68      0.68      0.68       630

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

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

         1.0       0.97      0.97      0.97        90
         2.0       0.79      0.86      0.82        90
         3.0       0.93      0.88      0.90        90
         4.0       0.91      0.90      0.91        90
         5.0       0.92      0.93      0.93        90
         6.0       0.95      0.91      0.93        90
         7.0       0.90      0.91      0.91        90

    accuracy                           0.91       630
   macro avg       0.91      0.91      0.91       630
weighted avg       0.91      0.91      0.91       630
```

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

Evaluando LDA
Resultados para LDA
              precision    recall  f1-score   support

         1.0       0.78      0.80      0.79        90
         2.0       0.53      0.66      0.59        90
         3.0       0.55      0.60      0.57        90
         4.0       0.62      0.59      0.61        90
         5.0       0.72      0.68      0.70        90
         6.0       0.61      0.57      0.59        90
         7.0       0.95      0.79      0.86        90

    accuracy                           0.67       630
   macro avg       0.68      0.67      0.67       630
weighted avg       0.68      0.67      0.67       630

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

Evaluando QDA
```

```
C:\Users\pfeli\AppData\Local\Packages\PythonSoftwareFoundation.Python.3.10_qbz5n
2kfra8p0\LocalCache\local-packages\Python310\site-
packages\sklearn\discriminant_analysis.py:947: UserWarning: Variables are
collinear
  warnings.warn("Variables are collinear")
C:\Users\pfeli\AppData\Local\Packages\PythonSoftwareFoundation.Python.3.10_qbz5n
2kfra8p0\LocalCache\local-packages\Python310\site-
packages\sklearn\discriminant_analysis.py:947: UserWarning: Variables are
collinear
  warnings.warn("Variables are collinear")
C:\Users\pfeli\AppData\Local\Packages\PythonSoftwareFoundation.Python.3.10_qbz5n
2kfra8p0\LocalCache\local-packages\Python310\site-
packages\sklearn\discriminant_analysis.py:947: UserWarning: Variables are
collinear
  warnings.warn("Variables are collinear")
C:\Users\pfeli\AppData\Local\Packages\PythonSoftwareFoundation.Python.3.10_qbz5n
2kfra8p0\LocalCache\local-packages\Python310\site-
packages\sklearn\discriminant_analysis.py:947: UserWarning: Variables are
collinear
  warnings.warn("Variables are collinear")
C:\Users\pfeli\AppData\Local\Packages\PythonSoftwareFoundation.Python.3.10_qbz5n
2kfra8p0\LocalCache\local-packages\Python310\site-
packages\sklearn\discriminant_analysis.py:947: UserWarning: Variables are
collinear
  warnings.warn("Variables are collinear")
```

```
Resultados para QDA
              precision    recall  f1-score   support

         1.0       0.16      0.12      0.14        90
         2.0       0.13      0.12      0.12        90
         3.0       0.15      0.23      0.19        90
         4.0       0.29      0.18      0.22        90
         5.0       0.21      0.20      0.21        90
         6.0       0.23      0.17      0.19        90
         7.0       0.39      0.58      0.47        90

    accuracy                           0.23       630
   macro avg       0.22      0.23      0.22       630
weighted avg       0.22      0.23      0.22       630

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

Evaluando SVM
Resultados para SVM
              precision    recall  f1-score   support

         1.0       0.96      0.99      0.97        90
         2.0       0.79      0.90      0.84        90
         3.0       0.95      0.87      0.91        90
         4.0       0.95      0.90      0.93        90
         5.0       0.92      0.90      0.91        90
         6.0       0.96      0.89      0.92        90
         7.0       0.90      0.96      0.92        90

    accuracy                           0.91       630
   macro avg       0.92      0.91      0.91       630
weighted avg       0.92      0.91      0.91       630

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

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

         1.0       0.93      0.70      0.80        90
         2.0       0.34      0.62      0.44        90
         3.0       0.73      0.66      0.69        90
         4.0       0.51      0.61      0.56        90
         5.0       0.66      0.49      0.56        90
         6.0       0.85      0.38      0.52        90
         7.0       0.88      0.99      0.93        90

    accuracy                           0.63       630
```

```
   macro avg       0.70      0.63      0.64       630
weighted avg       0.70      0.63      0.64       630

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

Evaluando Random Forest
Resultados para Random Forest
              precision    recall  f1-score   support

         1.0       0.91      0.89      0.90        90
         2.0       0.76      0.72      0.74        90
         3.0       0.92      0.81      0.86        90
         4.0       0.79      0.86      0.82        90
         5.0       0.90      0.93      0.92        90
         6.0       0.85      0.81      0.83        90
         7.0       0.89      0.99      0.94        90

    accuracy                           0.86       630
   macro avg       0.86      0.86      0.86       630
weighted avg       0.86      0.86      0.86       630

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

[12]:
```python
import numpy as np
import matplotlib.pyplot as plt
from sklearn import datasets
from sklearn.model_selection import StratifiedKFold, GridSearchCV
from sklearn.neighbors import KNeighborsClassifier
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score

#---------------------------------------------------------------------------------------------
# KNN classifier - Hyperparameter tuning
#---------------------------------------------------------------------------------------------

print("----- KNN classifier - K parameter -----")

# Definición del rango de valores para k
kk = np.arange(1, 140)
acc = []

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

for k in kk:
    print('---- k =', k)
```

```
    acc_cv = []

    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]

        clf_cv = KNeighborsClassifier(n_neighbors=k)
        clf_cv.fit(x_train, y_train)
        y_pred = clf_cv.predict(x_test)
        acc_i = accuracy_score(y_test, y_pred)
        acc_cv.append(acc_i)

    acc_hyp = np.mean(acc_cv)
    acc.append(acc_hyp)
    print('ACC:', acc_hyp)

opt_index = np.argmax(acc)
opt_hyperparameter = kk[opt_index]
print("Optimal k: ", opt_hyperparameter)

plt.plot(kk, acc)
plt.xlabel("k")
plt.ylabel("Accuracy")
plt.title("Accuracy vs K for KNN")
plt.show()

# Fit model with optimal hyperparameters
clf_knn = KNeighborsClassifier(n_neighbors=opt_hyperparameter)
clf_knn.fit(X, y)

#---------------------------------------------------------------------------------------------
# SVM classifier - Regularization parameter tuning
#---------------------------------------------------------------------------------------------

print("----- SVM classifier - Regularization parameter -----")

cc = np.logspace(-3, 1, 100)
acc = []

for c in cc:
    print('---- C =', c)

    acc_cv = []

    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]
```

```
        clf_cv = SVC(C=c, kernel='linear')
        clf_cv.fit(x_train, y_train)
        y_pred = clf_cv.predict(x_test)
        acc_i = accuracy_score(y_test, y_pred)
        acc_cv.append(acc_i)

    acc_hyp = np.mean(acc_cv)
    acc.append(acc_hyp)
    print('ACC:', acc_hyp)

opt_index = np.argmax(acc)
opt_hyperparameter = cc[opt_index]
print("Optimal C: ", opt_hyperparameter)

plt.plot(cc, acc)
plt.xscale('log')
plt.xlabel("C")
plt.ylabel("Accuracy")
plt.title("Accuracy vs C for SVM")
plt.show()

# Fit model with optimal hyperparameters
clf_svm = SVC(C=opt_hyperparameter, kernel='linear')
clf_svm.fit(X, y)

#----------------------------------------------------------------------------------------------
# Final results
#----------------------------------------------------------------------------------------------

print("Evaluación final de modelos")

# Evaluar KNN
cv_results_knn = []
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]
    clf_knn.fit(x_train, y_train)
    y_pred = clf_knn.predict(x_test)
    cv_results_knn.append(accuracy_score(y_test, y_pred))

print(f"Precisión media en validación cruzada para KNN: {np.
 ↪mean(cv_results_knn)}")

# Evaluar SVM
cv_results_svm = []
```

```
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]
    clf_svm.fit(x_train, y_train)
    y_pred = clf_svm.predict(x_test)
    cv_results_svm.append(accuracy_score(y_test, y_pred))

print(f"Precisión media en validación cruzada para SVM: {np.
 ↪mean(cv_results_svm)}")
```

```
----- KNN classifier - K parameter -----
---- k = 1
ACC: 0.8444444444444444
---- k = 2
ACC: 0.7873015873015873
---- k = 3
ACC: 0.8555555555555555
---- k = 4
ACC: 0.8587301587301587
---- k = 5
ACC: 0.8603174603174603
---- k = 6
ACC: 0.8666666666666666
---- k = 7
ACC: 0.8761904761904763
---- k = 8
ACC: 0.8761904761904763
---- k = 9
ACC: 0.8857142857142856
---- k = 10
ACC: 0.8746031746031747
---- k = 11
ACC: 0.8730158730158731
---- k = 12
ACC: 0.8682539682539682
---- k = 13
ACC: 0.8746031746031747
---- k = 14
ACC: 0.8714285714285713
---- k = 15
ACC: 0.8777777777777779
---- k = 16
ACC: 0.884126984126984
---- k = 17
ACC: 0.8841269841269842
---- k = 18
```

```
ACC: 0.8841269841269842
---- k = 19
ACC: 0.8793650793650792
---- k = 20
ACC: 0.8777777777777779
---- k = 21
ACC: 0.8714285714285713
---- k = 22
ACC: 0.8587301587301589
---- k = 23
ACC: 0.880952380952381
---- k = 24
ACC: 0.8714285714285716
---- k = 25
ACC: 0.8777777777777779
---- k = 26
ACC: 0.8682539682539684
---- k = 27
ACC: 0.8587301587301587
---- k = 28
ACC: 0.8682539682539682
---- k = 29
ACC: 0.8619047619047618
---- k = 30
ACC: 0.8634920634920636
---- k = 31
ACC: 0.8587301587301587
---- k = 32
ACC: 0.8619047619047618
---- k = 33
ACC: 0.8428571428571429
---- k = 34
ACC: 0.8523809523809524
---- k = 35
ACC: 0.8587301587301587
---- k = 36
ACC: 0.8555555555555555
---- k = 37
ACC: 0.846031746031746
---- k = 38
ACC: 0.8349206349206348
---- k = 39
ACC: 0.8285714285714286
---- k = 40
ACC: 0.846031746031746
---- k = 41
ACC: 0.8365079365079365
---- k = 42
```

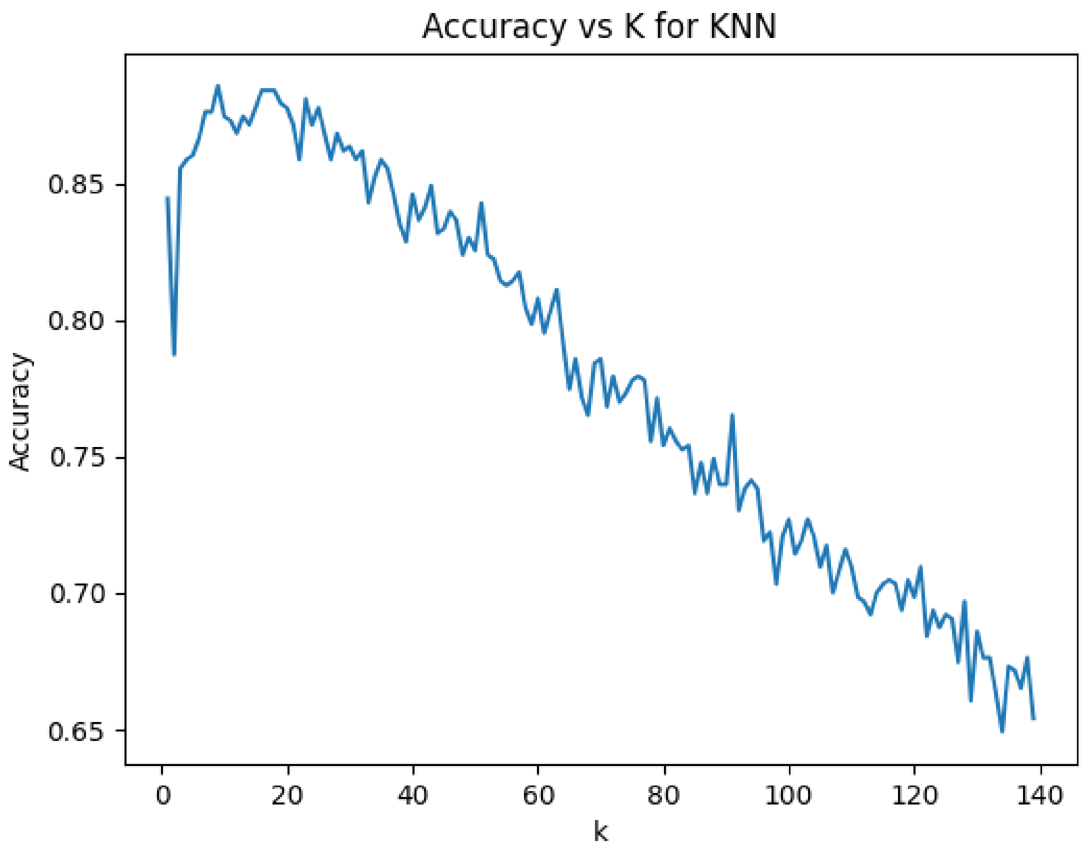
```
ACC: 0.8412698412698413
---- k = 43
ACC: 0.8492063492063492
---- k = 44
ACC: 0.8317460317460317
---- k = 45
ACC: 0.8333333333333333
---- k = 46
ACC: 0.8396825396825396
---- k = 47
ACC: 0.8365079365079365
---- k = 48
ACC: 0.8238095238095239
---- k = 49
ACC: 0.8301587301587301
---- k = 50
ACC: 0.8253968253968255
---- k = 51
ACC: 0.8428571428571429
---- k = 52
ACC: 0.8238095238095239
---- k = 53
ACC: 0.8222222222222223
---- k = 54
ACC: 0.8142857142857143
---- k = 55
ACC: 0.8126984126984127
---- k = 56
ACC: 0.8142857142857143
---- k = 57
ACC: 0.8174603174603174
---- k = 58
ACC: 0.8047619047619048
---- k = 59
ACC: 0.7984126984126985
---- k = 60
ACC: 0.807936507936508
---- k = 61
ACC: 0.7952380952380953
---- k = 62
ACC: 0.8031746031746032
---- k = 63
ACC: 0.8111111111111111
---- k = 64
ACC: 0.7920634920634921
---- k = 65
ACC: 0.7746031746031747
---- k = 66
```

```
ACC: 0.7857142857142857
---- k = 67
ACC: 0.7714285714285715
---- k = 68
ACC: 0.765079365079365
---- k = 69
ACC: 0.7841269841269841
---- k = 70
ACC: 0.7857142857142858
---- k = 71
ACC: 0.7682539682539682
---- k = 72
ACC: 0.7793650793650793
---- k = 73
ACC: 0.7698412698412698
---- k = 74
ACC: 0.7730158730158729
---- k = 75
ACC: 0.7777777777777778
---- k = 76
ACC: 0.7793650793650794
---- k = 77
ACC: 0.7777777777777778
---- k = 78
ACC: 0.7555555555555555
---- k = 79
ACC: 0.7714285714285714
---- k = 80
ACC: 0.753968253968254
---- k = 81
ACC: 0.7603174603174603
---- k = 82
ACC: 0.7555555555555555
---- k = 83
ACC: 0.7523809523809524
---- k = 84
ACC: 0.753968253968254
---- k = 85
ACC: 0.7365079365079366
---- k = 86
ACC: 0.7476190476190476
---- k = 87
ACC: 0.7365079365079366
---- k = 88
ACC: 0.7492063492063492
---- k = 89
ACC: 0.7396825396825396
---- k = 90
```

```
ACC: 0.7396825396825397
---- k = 91
ACC: 0.765079365079365
---- k = 92
ACC: 0.7301587301587302
---- k = 93
ACC: 0.7380952380952381
---- k = 94
ACC: 0.7412698412698413
---- k = 95
ACC: 0.738095238095238
---- k = 96
ACC: 0.719047619047619
---- k = 97
ACC: 0.7222222222222222
---- k = 98
ACC: 0.7031746031746031
---- k = 99
ACC: 0.7206349206349206
---- k = 100
ACC: 0.726984126984127
---- k = 101
ACC: 0.7142857142857143
---- k = 102
ACC: 0.7190476190476192
---- k = 103
ACC: 0.7269841269841271
---- k = 104
ACC: 0.7206349206349206
---- k = 105
ACC: 0.7095238095238094
---- k = 106
ACC: 0.7174603174603176
---- k = 107
ACC: 0.7
---- k = 108
ACC: 0.7079365079365079
---- k = 109
ACC: 0.7158730158730158
---- k = 110
ACC: 0.7095238095238094
---- k = 111
ACC: 0.6984126984126984
---- k = 112
ACC: 0.6968253968253968
---- k = 113
ACC: 0.692063492063492
---- k = 114
```

```
ACC: 0.7
---- k = 115
ACC: 0.7031746031746031
---- k = 116
ACC: 0.7047619047619047
---- k = 117
ACC: 0.7031746031746031
---- k = 118
ACC: 0.6936507936507936
---- k = 119
ACC: 0.7047619047619047
---- k = 120
ACC: 0.6984126984126984
---- k = 121
ACC: 0.7095238095238096
---- k = 122
ACC: 0.6841269841269841
---- k = 123
ACC: 0.6936507936507936
---- k = 124
ACC: 0.6873015873015873
---- k = 125
ACC: 0.692063492063492
---- k = 126
ACC: 0.6904761904761905
---- k = 127
ACC: 0.6746031746031746
---- k = 128
ACC: 0.6968253968253968
---- k = 129
ACC: 0.6603174603174603
---- k = 130
ACC: 0.6857142857142857
---- k = 131
ACC: 0.6761904761904762
---- k = 132
ACC: 0.6761904761904762
---- k = 133
ACC: 0.6634920634920635
---- k = 134
ACC: 0.6492063492063492
---- k = 135
ACC: 0.6730158730158731
---- k = 136
ACC: 0.6714285714285714
---- k = 137
ACC: 0.6650793650793652
---- k = 138
```

```
ACC: 0.6761904761904762
---- k = 139
ACC: 0.653968253968254
Optimal k:  9
```



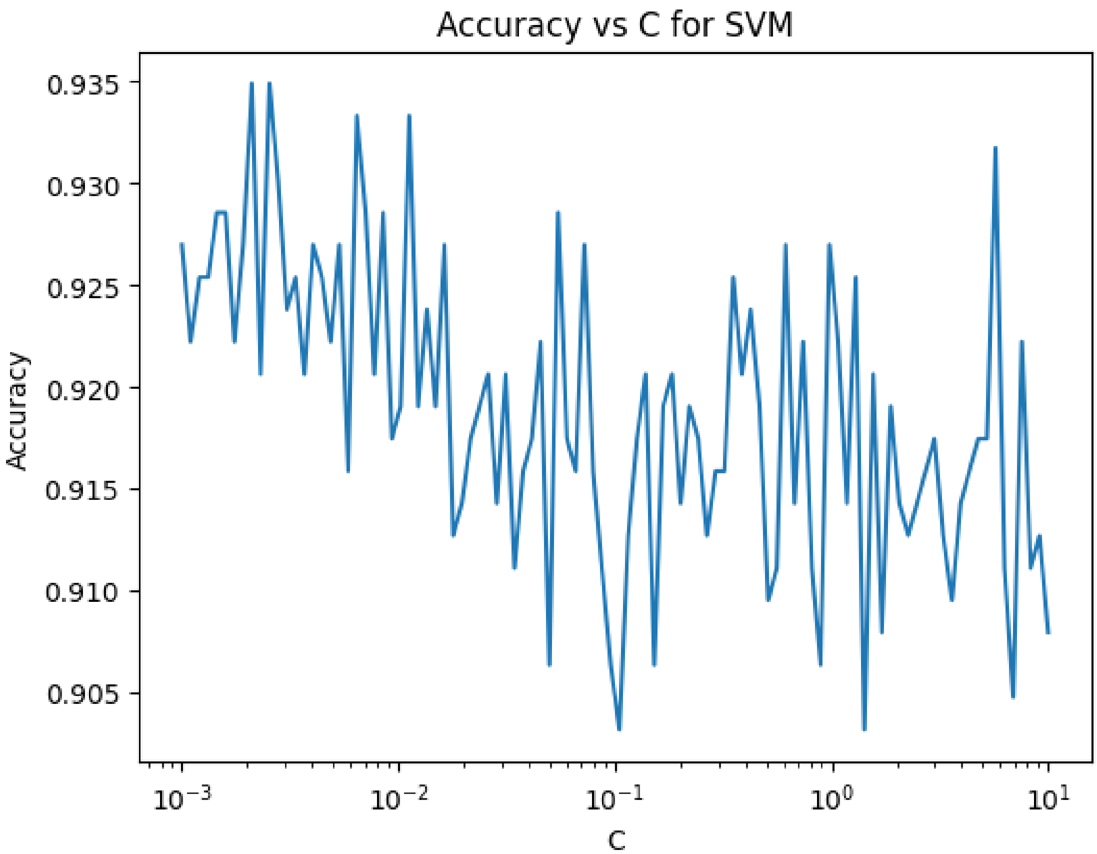

```
----- SVM classifier - Regularization parameter -----
---- C = 0.001
ACC: 0.9269841269841269
---- C = 0.0010974987654930556
ACC: 0.9222222222222222
---- C = 0.0012045035402587824
ACC: 0.9253968253968253
---- C = 0.0013219411484660286
ACC: 0.9253968253968254
---- C = 0.0014508287784959402
ACC: 0.9285714285714286
---- C = 0.0015922827933410922
ACC: 0.9285714285714286
---- C = 0.001747528400007683
ACC: 0.9222222222222222
```

```
---- C = 0.0019179102616724887
ACC: 0.926984126984127
---- C = 0.00210490414451202
ACC: 0.9349206349206349
---- C = 0.0023101297000831605
ACC: 0.9206349206349207
---- C = 0.0025353644939701114
ACC: 0.9349206349206349
---- C = 0.0027825594022071257
ACC: 0.9301587301587301
---- C = 0.0030538555088334154
ACC: 0.9238095238095237
---- C = 0.003351602650938841
ACC: 0.9253968253968253
---- C = 0.0036783797718286343
ACC: 0.9206349206349206
---- C = 0.004037017258596553
ACC: 0.9269841269841269
---- C = 0.004430621457583882
ACC: 0.9253968253968254
---- C = 0.004862601580065354
ACC: 0.9222222222222222
---- C = 0.005336699231206312
ACC: 0.9269841269841269
---- C = 0.005857020818056668
ACC: 0.9158730158730158
---- C = 0.006428073117284319
ACC: 0.9333333333333333
---- C = 0.007054802310718645
ACC: 0.9285714285714286
---- C = 0.007742636826811269
ACC: 0.9206349206349207
---- C = 0.008497534359086447
ACC: 0.9285714285714285
---- C = 0.0093260334688322
ACC: 0.9174603174603174
---- C = 0.010235310218990263
ACC: 0.919047619047619
---- C = 0.011233240329780276
ACC: 0.9333333333333333
---- C = 0.012328467394420665
ACC: 0.919047619047619
---- C = 0.013530477745798075
ACC: 0.9238095238095237
---- C = 0.01484968262254465
ACC: 0.919047619047619
---- C = 0.016297508346206444
ACC: 0.926984126984127
```

```
---- C = 0.01788649529057435
ACC: 0.9126984126984128
---- C = 0.019630406500402715
ACC: 0.9142857142857144
---- C = 0.021544346900318846
ACC: 0.9174603174603174
---- C = 0.023644894126454083
ACC: 0.9190476190476191
---- C = 0.025950242113997372
ACC: 0.9206349206349206
---- C = 0.02848035868435802
ACC: 0.9142857142857143
---- C = 0.03125715849688237
ACC: 0.9206349206349206
---- C = 0.03430469286314919
ACC: 0.9111111111111111
---- C = 0.037649358067924694
ACC: 0.915873015873016
---- C = 0.04132012400115339
ACC: 0.9174603174603175
---- C = 0.04534878508128584
ACC: 0.9222222222222223
---- C = 0.049770235643321115
ACC: 0.9063492063492063
---- C = 0.05462277217684343
ACC: 0.9285714285714285
---- C = 0.05994842503189412
ACC: 0.9174603174603175
---- C = 0.06579332246575682
ACC: 0.9158730158730158
---- C = 0.07220809018385467
ACC: 0.9269841269841269
---- C = 0.07924828983539177
ACC: 0.9158730158730158
---- C = 0.08697490026177834
ACC: 0.9111111111111111
---- C = 0.09545484566618342
ACC: 0.9063492063492063
---- C = 0.10476157527896651
ACC: 0.9031746031746032
---- C = 0.11497569953977368
ACC: 0.9126984126984127
---- C = 0.1261856883066021
ACC: 0.9174603174603174
---- C = 0.1384886371393873
ACC: 0.9206349206349206
---- C = 0.15199110829529347
ACC: 0.9063492063492063
```

```
---- C = 0.1668100537200059
ACC: 0.919047619047619
---- C = 0.18307382802953698
ACC: 0.9206349206349206
---- C = 0.2009233002565048
ACC: 0.9142857142857144
---- C = 0.22051307399030456
ACC: 0.9190476190476191
---- C = 0.24201282647943834
ACC: 0.9174603174603175
---- C = 0.26560877829466867
ACC: 0.9126984126984127
---- C = 0.29150530628251786
ACC: 0.9158730158730158
---- C = 0.31992671377973847
ACC: 0.9158730158730158
---- C = 0.35111917342151344
ACC: 0.9253968253968254
---- C = 0.3853528593710531
ACC: 0.9206349206349207
---- C = 0.4229242874389499
ACC: 0.9238095238095239
---- C = 0.4641588833612782
ACC: 0.9190476190476191
---- C = 0.5094138014816381
ACC: 0.9095238095238095
---- C = 0.5590810182512228
ACC: 0.9111111111111111
---- C = 0.6135907273413176
ACC: 0.926984126984127
---- C = 0.6734150657750828
ACC: 0.9142857142857143
---- C = 0.7390722033525783
ACC: 0.9222222222222223
---- C = 0.8111308307896873
ACC: 0.9111111111111111
---- C = 0.8902150854450392
ACC: 0.9063492063492063
---- C = 0.9770099572992257
ACC: 0.9269841269841269
---- C = 1.0722672220103242
ACC: 0.9222222222222223
---- C = 1.1768119524349991
ACC: 0.9142857142857143
---- C = 1.291549665014884
ACC: 0.9253968253968253
---- C = 1.4174741629268062
ACC: 0.9031746031746033
```

```
---- C = 1.5556761439304723
ACC: 0.9206349206349207
---- C = 1.7073526474706922
ACC: 0.9079365079365079
---- C = 1.873817422860385
ACC: 0.9190476190476191
---- C = 2.0565123083486534
ACC: 0.9142857142857143
---- C = 2.2570197196339215
ACC: 0.9126984126984127
---- C = 2.4770763559917115
ACC: 0.9142857142857143
---- C = 2.718588242732943
ACC: 0.9158730158730158
---- C = 2.9836472402833403
ACC: 0.9174603174603174
---- C = 3.2745491628777317
ACC: 0.9126984126984127
---- C = 3.5938136638046294
ACC: 0.9095238095238095
---- C = 3.94420605943766
ACC: 0.9142857142857143
---- C = 4.328761281083062
ACC: 0.9158730158730158
---- C = 4.750810162102798
ACC: 0.9174603174603174
---- C = 5.21400828799969
ACC: 0.9174603174603175
---- C = 5.72236765935022
ACC: 0.9317460317460318
---- C = 6.280291441834259
ACC: 0.9111111111111111
---- C = 6.892612104349702
ACC: 0.9047619047619048
---- C = 7.56463327554629
ACC: 0.9222222222222223
---- C = 8.302175681319753
ACC: 0.9111111111111111
---- C = 9.111627561154895
ACC: 0.9126984126984127
---- C = 10.0
ACC: 0.9079365079365079
Optimal C:  0.00210490414451202
```




```
Evaluación final de modelos
Precisión media en validación cruzada para KNN: 0.880952380952381
Precisión media en validación cruzada para SVM: 0.9238095238095239
```

¿Observas un problema en cuanto al balanceo de las clases? ¿Por qué? ¿Qué modelo o modelos fueron efectivos para clasificar tus datos? ¿Observas algo especial sobre los modelos? Argumenta tu respuesta. ¿Observas alguna mejora importante al optimizar hiperparámetros? ¿Es el resultado que esperabas? Argumenta tu respuesta. ¿Qué inconvenientes hay al encontrar hiperparámetros? ¿Por qué?

No observo ningun problema en cuanto al balanceo de las clases, ya que el conjunto de datos está perfectamente balanceado, con 90 muestras en cada clase.

Los modelos más efectivos para clasificar los datos fueron SVM con una precisión media de 0.92 y K-Vecinos con una precisión media de 0.87. Mientras SVM tiene una buena capacidad para manejar la separación de clases y encontrar un margen óptimo, KNN se beneficia de su simplicidad y capacidad de ajustar el número de vecinos.

Una vez realizada la optimización de los hiperparámetros, se volvió a evaluar el rendimiento de los modelos SVM y KNN. Sin embargo, ninguno de los dos mostró una mejora significativa, la precisión solo aumentó en algunas centésimas, lo que no era lo que esperaba. Esto podría indicar que la técnica de optimización de hiperparámetros no es tan efectiva para modelos ya bien ajustados como estos.

Además, al encontrar los hiperparámetros óptimos, se pueden presentar varios inconvenientes. El modelo podría sobreajustarse a los datos de entrenamiento, lo que afecta su capacidad para generalizar a nuevos datos. También, el proceso de búsqueda de hiperparámetros puede resultar muy costoso en términos computacionales, ya que requiere evaluar múltiples combinaciones y puede ser intensivo en recursos.