ЛАБОРАТОРНА РОБОТА № 1

ПОПЕРЕДНЯ ОБРОБКА ТА КОНТРОЛЬОВАНА КЛАСИФІКАЦІЯ ДАНИХ

Лукашевич Влада ІПЗ-21-1

https://github.com/vladalukashevych/artificial-intelligence-systems

Завдання 2.1. Попередня обробка даних

```
import numpy as np
from sklearn import preprocessing

input_data = np.array([[5.1, -2.9, 3.3], [-1.2, 7.8, -6.1], [3.9, 0.4, 2.1],
[7.3, -9.9, -4.5]])
data_binarized = preprocessing.Binarizer(threshold=2.1).transform(input_data)
print("\n Binarized data:\n", data_binarized)

# Виведення середнього значення та стандартного відхилення
print("\nBEFORE: ")
print("Mean =", input_data.mean(axis=0))
print("Std deviation =", input_data.std(axis=0))

# Виключення середнього
data_scaled = preprocessing.scale(input_data)
print("\nAFTER: ")
print("Mean =", data_scaled.mean(axis=0))
print("Std deviation =", data_scaled.std(axis=0))

# Масштабування MinMax
data_scaler_minmax = preprocessing.MinMaxScaler(feature_range=(0, 1))
data_scaled_minmax = data_scaler_minmax.fit_transform(input_data)
print("\nMin max scaled data:\n", data_scaled_minmax)

# Нормалізація даних
data_normalized_l1 = preprocessing.normalize(input_data, norm='l1')
data_normalized_l2 = preprocessing.normalize(input_data, norm='l2')
print("\nl1 normalized data:\n", data_normalized_l1)
print("\nl2 normalized data:\n", data_normalized_l2)
```

```
 Binarized data:
 [[1. 0. 1.]
 [0. 1. 0.]
 [1. 0. 0.]
 [1. 0. 0.]]

BEFORE:
Mean = [ 3.775 -1.15  -1.3  ]
Std deviation = [3.12039661 6.36651396 4.0620192 ]

AFTER:
Mean = [1.11022302e-16 0.00000000e+00 2.77555756e-17]
Std deviation = [1. 1. 1.]

Min max scaled data:
 [[0.74117647 0.39548023 1.        ]
 [0.         1.         0.        ]
 [0.6        0.5819209  0.87234043]
 [1.         0.         0.17021277]]
```

```
l1 normalized data:
 [[ 0.45132743 -0.25663717  0.2920354 ]
 [-0.0794702   0.51655629 -0.40397351]
 [ 0.609375    0.0625      0.328125  ]
 [ 0.33640553 -0.4562212  -0.20737327]]

l2 normalized data:
 [[ 0.75765788 -0.43082507  0.49024922]
 [-0.12030718  0.78199664 -0.61156148]
 [ 0.87690281  0.08993875  0.47217844]
 [ 0.55734935 -0.75585734 -0.34357152]]
```

L1-нормалізація і L2-нормалізація є двома різними методами, які використовуються для приведення векторів ознак до загальної шкали. Основна різниця між ними полягає в тому, як вони змінюють значення вектора. L1-нормалізація використовує метод найменших абсолютних відхилень, змінюючи значення так, щоб сума абсолютних значень дорівнювала 1. У свою чергу, L2-нормалізація застосовує метод найменших квадратів, забезпечуючи рівність 1 суми квадратів значень.

Ці методи також різняться чутливістю до викидів. L1-нормалізація є менш чутливою до викидів, оскільки мінімізує абсолютні відхилення, що робить її більш надійною у випадках, коли дані містять викиди. Це означає, що великі відхилення не мають значного впливу на нормалізовані значення. Натомість L2-нормалізація є більш чутливою до викидів, оскільки мінімізує квадрати

відхилень, і великі значення можуть суттєво вплинути на результат нормалізації.

```
import numpy as np
from sklearn import preprocessing

# Надання позначок вхідних даних
input_labels = ['red', 'Ыack', 'red', 'green', 'Ьlack', 'yellow', 'white']

# Створення кодувальника та встановлення відповідності
# між мітками та числами
encoder = preprocessing.LabelEncoder()
encoder.fit(input_labels)

# Виведення відображення
print("\nLabel mapping:")
for i, item in enumerate(encoder.classes_) : print(item, '-->', i)

# перетворення міток за допомогою кодувальника
test_labels = ['green', 'red', 'Ыack']
encoded_values = encoder.transform(test_labels )
print("\nLabels =", test_labels)
print("Encoded values =", list (encoded_values ) )

# Декодування набору чисел за допомогою декодера
encoded_values = [3, 0, 4, 1]
decoded_list = encoder.inverse_transform(encoded_values)
print("\nEncoded values =", encoded_values)
print("Decoded labels =", list (decoded_list ) )
```

```
Label mapping:
green --> 0
red --> 1
white --> 2
yellow --> 3
Ьlack --> 4
black --> 5

Labels = ['green', 'red', 'Ыack']
Encoded values = [np.int64(0), np.int64(1), np.int64(4)]

Encoded values = [3, 0, 4, 1]
Decoded labels = [np.str_('yellow'), np.str_('green'), np.str_('Ыack'), np.str_('red')]
```

Завдання 2.2. Попередня обробка нових даних

12.	-1.3	3.9	4.5	-5.3	-4.2	3.3	-5.2	-6.5	-1.1	-5.2	2.6	-2.2	1.8

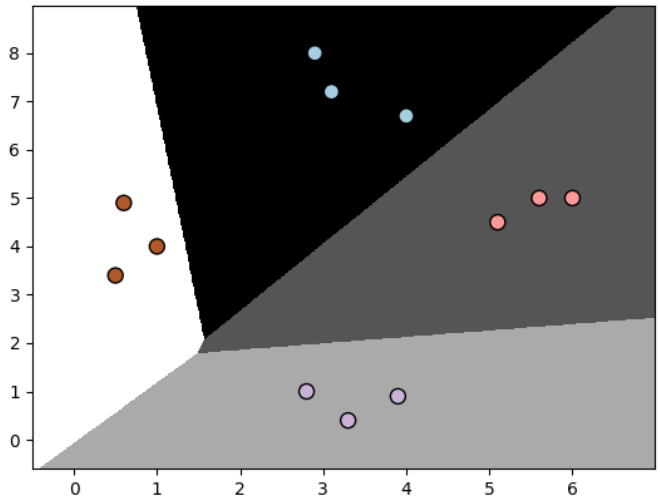
```
import numpy as np
from sklearn import preprocessing

input_data = np.array([[-1.3, 3.9, 4.5], [-5.3, -4.2, 3.3], [-5.2, -6.5, -1.1],
[-5.2, 2.6, -2.2]])
data_binarized = preprocessing.Binarizer(threshold=1.8).transform(input_data)
print("\n Binarized data:\n", data_binarized)

# Виведення середнього значення та стандартного відхилення
print("\nBEFORE: ")
print("Mean =", input_data.mean(axis=0))
print("Std deviation =", input_data.std(axis=0))

# Виключення середнього
data_scaled = preprocessing.scale(input_data)
print("\nAFTER: ")
print("Mean =", data_scaled.mean(axis=0))
print("Std deviation =", data_scaled.std(axis=0))

# Масштабування MinMax
data_scaler_minmax = preprocessing.MinMaxScaler(feature_range=(0, 1))
data_scaled_minmax = data_scaler_minmax.fit_transform(input_data)
print("\nMin max scaled data:\n", data_scaled_minmax)

# Нормалізація даних
data_normalized_l1 = preprocessing.normalize(input_data, norm='l1')
data_normalized_l2 = preprocessing.normalize(input_data, norm='l2')
print("\nl1 normalized data:\n", data_normalized_l1)
print("\nl2 normalized data:\n", data_normalized_l2)
```

```
 Binarized data:
 [[0. 1. 1.]
 [0. 0. 1.]
 [0. 0. 0.]
 [0. 1. 0.]]

BEFORE:
Mean = [-4.25  -1.05   1.125]
Std deviation = [1.7036725  4.40028408 2.83405628]

AFTER:
Mean = [0.00000000e+00 5.55111512e-17 0.00000000e+00]
Std deviation = [1. 1. 1.]

Min max scaled data:
 [[1.         1.         1.        ]
 [0.         0.22115385 0.82089552]
 [0.025      0.         0.1641791 ]
 [0.025      0.875      0.        ]]
```

```
l1 normalized data:
 [[-0.13402062  0.40206186  0.46391753]
 [-0.4140625  -0.328125    0.2578125 ]
 [-0.40625    -0.5078125  -0.0859375 ]
 [-0.52        0.26       -0.22      ]]

l2 normalized data:
 [[-0.21328678  0.63986035  0.7383004 ]
 [-0.70435392 -0.55816726  0.43855999]
 [-0.61931099 -0.77413873 -0.13100809]
 [-0.83653629  0.41826814 -0.3539192 ]]
```

Завдання 2.3. Класифікація логістичною регресією або логістичний класифікатор

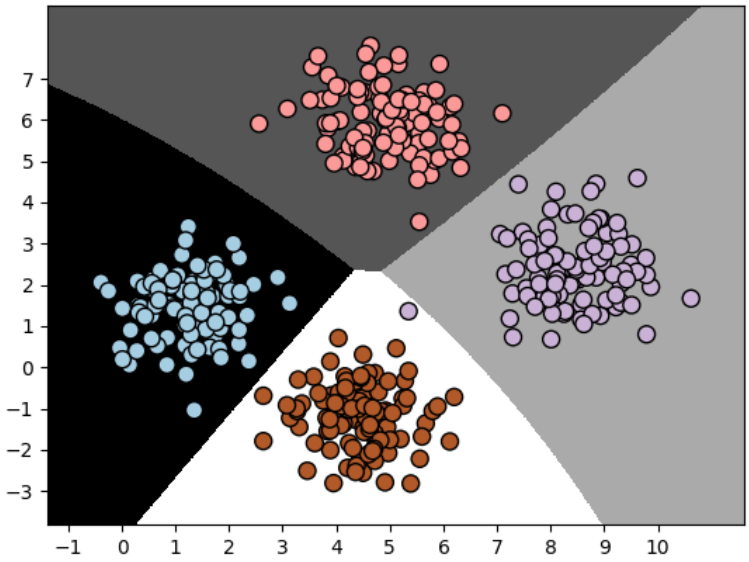
```
import numpy as np
from sklearn import linear_model
import matplotlib.pyplot as plt
from utilities import visualize_classifier

# Визначення зразка вхідних даних
X = np.array([[3.1, 7.2], [4, 6.7], [2.9, 8], [5.1, 4.5], [6, 5], [5.6, 5],
[3.3, 0.4], [3.9, 0.9], [2.8, 1], [0.5, 3.4], [1, 4], [0.6, 4.9]])
y = np.array([0, 0, 0, 1, 1, 1, 2, 2, 2, 3, 3, 3])

# Створення логістичного класифікатора
classifier = linear model.LogisticRegression(solver='liblinear',C=1)

# Тренування класифікатора
classifier.fit(X, y)

visualize_classifier(classifier, X, y)
```



Завдання 2.4. Класифікація наївним байєсовським класифікатором

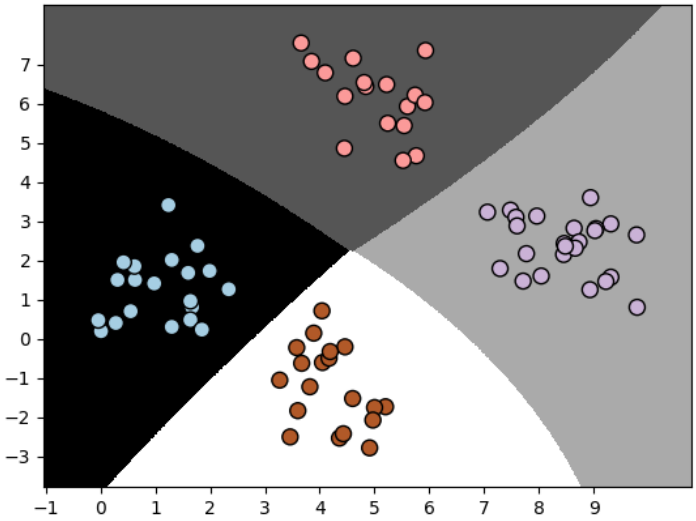
```python
import numpy as np
import matplotlib.pyplot as plt
from sklearn.naive_bayes import GaussianNB
from sklearn.model_selection import train_test_split
from utilities import visualize_classifier

# Вхідний файл, який містить дані
input_file = 'data_multivar_nb.txt'

# Завантаження даних із вхідного файлу
data = np.loadtxt(input_file, delimiter=',')
X, y = data[:, :-1], data[:, -1]

# Створення наївного байєсовського класифікатора
classifier = GaussianNB()

# Тренування класифікатора
classifier.fit(X, y)

# Прогнозування значень для тренувальних даних
y_pred = classifier.predict(X)

# Обчислення якості класифікатора
accuracy = 100.0 * (y == y_pred).sum() / X.shape[0]
print("Accuracy of Naive Bayes classifier =", round(accuracy, 2), "%")

# Візуалізація результатів роботи класифікатора
visualize_classifier(classifier, X, y)
```

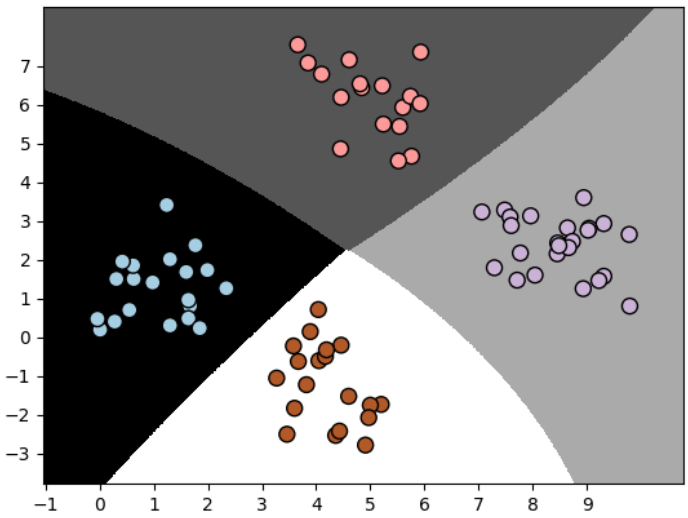
```
Accuracy of Naive Bayes classifier = 99.75 %
```



```
# Розбивка даних на навчальний та тестовий набори
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=3)
classifier_new = GaussianNB()
classifier_new.fit(X_train, y_train)
y_test_pred = classifier_new.predict(X_test)

# Обчислення якості класифікатора
accuracy = 100.0 * (y_test == y_test_pred).sum() / X_test.shape[0]
print("Accuracy of the new classifier =", round(accuracy, 2), "%")

# Візуалізація роботи класифікатора
visualize_classifier(classifier_new, X_test, y_test)

num_folds = 3
accuracy_values = cross_val_score(classifier, X, y, scoring='accuracy',
cv=num_folds)
print("Accuracy: " + str(round(100 * accuracy_values.mean(), 2)) + "%")
precision_values = cross_val_score(classifier, X, y,
scoring='precision_weighted', cv=num_folds)
print("Precision: " + str(round(100 * precision_values.mean(), 2)) + "%")
recall_values = cross_val_score(classifier, X, y, scoring='recall_weighted',
cv=num_folds)
print("Recall: " + str(round(100 * recall_values.mean(), 2)) + "%")
f1_values = cross_val_score(classifier, X, y, scoring='f1_weighted',
cv=num_folds)
print("F1: " + str(round(100 * f1_values.mean(), 2)) + "%")
```

```
Accuracy of the new classifier = 100.0 %
Accuracy: 99.75%
Precision: 99.76%
Recall: 99.75%
F1: 99.75%
```





Завдання 2.5. Вивчити метрики якості класифікації

```
import pandas as pd

df = pd.read_csv('data_metrics.csv')
print(df.head())

thresh = 0.5
df['predicted_RF'] = (df.model_RF >= 0.5).astype('int')
df['predicted_LR'] = (df.model_LR >= 0.5).astype('int')
print(df.head())

from sklearn.metrics import confusion_matrix
print("\nconfusion_matrix function:")
print(confusion_matrix(df.actual_label.values, df.predicted_RF.values))

#---------------------

def find_TP(y_true, y_pred):
    return sum((y_true == 1) & (y_pred == 1))

def find_FN(y_true, y_pred):
    return sum((y_true == 1) & (y_pred == 0))

def find_FP(y_true, y_pred):
    return sum((y_true == 0) & (y_pred == 1))

def find_TN(y_true, y_pred):
    return sum((y_true == 0) & (y_pred == 0))

print('TP:', find_TP(df.actual_label.values, df.predicted_RF.values))
print('FN:', find_FN(df.actual_label.values, df.predicted_RF.values))
print('FP:', find_FP(df.actual_label.values, df.predicted_RF.values))
print('TN:', find_TN(df.actual_label.values, df.predicted_RF.values))

#-----------------------

import numpy as np
def find_conf_matrix_values(y_true,y_pred):
# calculate TP, FN, FP, TN
```

```
    TP = find_TP(y_true,y_pred)
    FN = find_FN(y_true,y_pred)
    FP = find_FP(y_true,y_pred)
    TN = find_TN(y_true,y_pred)
    return TP,FN,FP,TN

def lukashevych_confusion_matrix(y_true, y_pred):
    TP,FN,FP,TN = find_conf_matrix_values(y_true,y_pred)
    return np.array([[TN,FP],[FN,TP]])

print("\nlukashevych_confusion_matrix function:")
print(lukashevych_confusion_matrix(df.actual_label.values,
df.predicted_RF.values))
print(lukashevych_confusion_matrix(df.actual_label.values,
df.predicted_RF.values))


assert
np.array_equal(lukashevych_confusion_matrix(df.actual_label.values,df.predicted_
RF.values),
                      confusion_matrix(df.actual_label.values,
df.predicted_RF.values)), \
    'lukashevych_confusion_matrix() is not correct for RF'
assert np.array_equal(lukashevych_confusion_matrix(df.actual_label.values,
df.predicted_LR.values),
                      confusion_matrix(df.actual_label.values,
df.predicted_LR.values)), \
    'lukashevych_confusion_matrix() is not correct for LR'

#--------------------------

from sklearn.metrics import accuracy_score
print("\naccuracy_score:")
print(accuracy_score(df.actual_label.values, df.predicted_RF.values))


def lukashevych_accuracy_score(y_true, y_pred):
# calculates the fraction of samples
    TP,FN,FP,TN = find_conf_matrix_values(y_true,y_pred)
    return (TP + TN)/(TP + TN + FP + FN)


assert lukashevych_accuracy_score(df.actual_label.values,
df.predicted_RF.values) == \
       accuracy_score(df.actual_label.values, df.predicted_RF.values), \
    'lukashevych_accuracy_score failed on RF'
assert lukashevych_accuracy_score(df.actual_label.values,
df.predicted_LR.values) == \
       accuracy_score(df.actual_label.values, df.predicted_LR.values), \
    'lukashevych_accuracy_score failed on LR'

print("\nlukashevych_accuracy_score:")
print('Accuracy RF: %.3f'%(lukashevych_accuracy_score(df.actual_label.values,
df.predicted_RF.values)))
print('Accuracy LR: %.3f'%(lukashevych_accuracy_score(df.actual_label.values,
df.predicted_LR.values)))

#-------------------------

from sklearn.metrics import recall_score
print("\nrecall_score:")
print(recall_score(df.actual_label.values, df.predicted_RF.values))



def lukashevych_recall_score(y_true, y_pred):
```

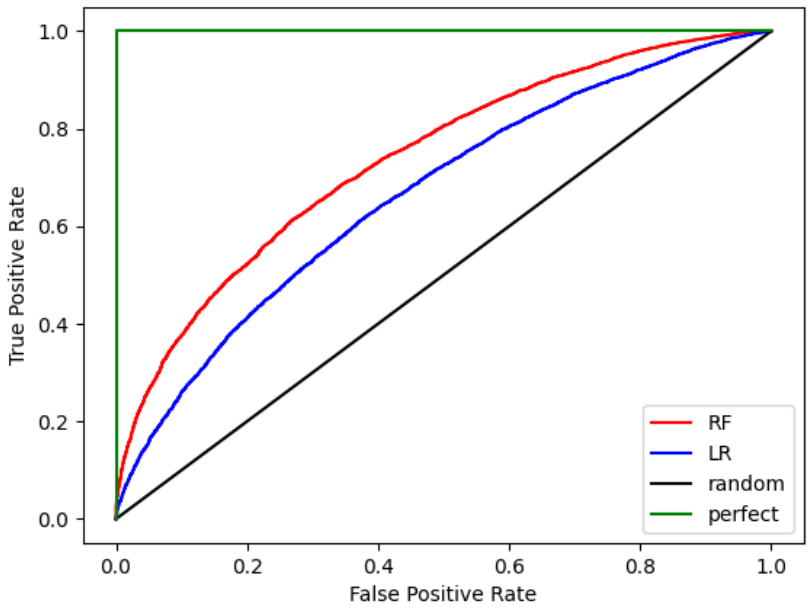
```
# calculates the fraction of positive samples predicted correctly
    TP,FN,FP,TN = find_conf_matrix_values(y_true,y_pred)
    return TP / (TP + FN)

assert lukashevych_recall_score(df.actual_label.values, df.predicted_RF.values)
== \
       recall_score(df.actual_label.values, df.predicted_RF.values),\
    'lukashevych_recall_score failed on RF'

assert lukashevych_recall_score(df.actual_label.values, df.predicted_LR.values)
== \
       recall_score(df.actual_label.values, df.predicted_LR.values), \
    'lukashevych_recall_score failed on LR'

print("\nlukashevych_recall_score:")
print('Recall RF: %.3f'%(lukashevych_recall_score(df.actual_label.values,
df.predicted_RF.values)))
print('Recall LR: %.3f'%(lukashevych_recall_score(df.actual_label.values,
df.predicted_LR.values)))

#----------------------------

from sklearn.metrics import precision_score
print("\nprecision_score:")
print(precision_score(df.actual_label.values, df.predicted_RF.values))


def lukashevych_precision_score(y_true, y_pred):
# calculates the fraction of predicted positives samples that are actually
positive
    TP,FN,FP,TN = find_conf_matrix_values(y_true,y_pred)
    return TP / (TP + FP)

assert lukashevych_precision_score(df.actual_label.values,
df.predicted_RF.values) == \
       precision_score(df.actual_label.values, df.predicted_RF.values), \
    'lukashevych_precision_score failed on RF'
assert lukashevych_precision_score(df.actual_label.values,
df.predicted_LR.values) == \
       precision_score(df.actual_label.values, df.predicted_LR.values), \
    'lukashevych_precision_score failed on LR'

print("\nlukashevych_precision_score:")
print('Precision RF: %.3f'%(lukashevych_precision_score(df.actual_label.values,
df.predicted_RF.values)))
print('Precision LR: %.3f'%(lukashevych_precision_score(df.actual_label.values,
df.predicted_LR.values)))

from sklearn.metrics import f1_score
print("\nf1_score:")
print(f1_score(df.actual_label.values, df.predicted_RF.values))

def lukashevych_f1_score(y_true, y_pred):
# calculates the F1 score
    recall = lukashevych_recall_score(y_true,y_pred)
    precision = lukashevych_precision_score(y_true,y_pred)
    return 2 / (1 / precision + 1 / recall)
    # return (2 * precision * recall) / (precision + recall)

# print(lukashevych_f1_score(df.actual_label.values, df.predicted_RF.values))
# print(f1_score(df.actual_label.values, df.predicted_RF.values))
# assert lukashevych_f1_score(df.actual_label.values, df.predicted_RF.values) ==
\
#        f1_score(df.actual_label.values, df.predicted_RF.values), \
#     'lukashevych_f1_score failed on RF'
```

```
# print(lukashevych_f1_score(df.actual_label.values, df.predicted_LR.values))
# print(f1_score(df.actual_label.values, df.predicted_LR.values))
assert lukashevych_f1_score(df.actual_label.values, df.predicted_LR.values) == \
       f1_score(df.actual_label.values, df.predicted_LR.values), \
    'lukashevych_f1_score failed on LR'

print("\nlukashevych_f1_score:")
print('F1 RF: %.3f' % (lukashevych_f1_score(df.actual_label.values,
df.predicted_RF.values)))
print('F1 LR: %.3f' % (lukashevych_f1_score(df.actual_label.values,
df.predicted_LR.values)))

#---------------------------

print('\nscores with threshold = 0.5')
print('Accuracy RF: %.3f'%(lukashevych_accuracy_score(df.actual_label.values,
df.predicted_RF.values)))
print('Recall RF: %.3f'%(lukashevych_recall_score(df.actual_label.values,
df.predicted_RF.values)))
print('Precision RF: %.3f'%(lukashevych_precision_score(df.actual_label.values,
df.predicted_RF.values)))
print('F1 RF: %.3f'%(lukashevych_f1_score(df.actual_label.values,
df.predicted_RF.values)))
print('')
print('scores with threshold = 0.25')
print('Accuracy RF: %.3f'%(lukashevych_accuracy_score(df.actual_label.values,
(df.model_RF >= 0.25).astype('int').values)))
print('Recall RF: %.3f'%(lukashevych_recall_score(df.actual_label.values,
(df.model_RF >= 0.25).astype('int').values)))
print('Precision RF: %.3f'%(lukashevych_precision_score(df.actual_label.values,
(df.model_RF >= 0.25).astype('int').values)))
print('F1 RF: %.3f'%(lukashevych_f1_score(df.actual_label.values, (df.model_RF
>= 0.25).astype('int').values)))
```

Зменшивши поріг з 0.5 до 0.25, ми стаємо менш строгими у передбаченні класу 1, тобто більше прикладів можуть бути віднесені до позитивного класу навіть з нижчою ймовірністю. Це призводить до зростання повноти, оскільки модель тепер знаходить більше позитивних прикладів. Проте це часто відбувається за рахунок зниження точності, оскільки більше передбачень можуть виявитися хибно позитивними. За таких умов оцінка F1 може допомогти оцінити баланс між точністю і повнотою, відображаючи наскільки ефективно модель справляється з хибними позитивами.

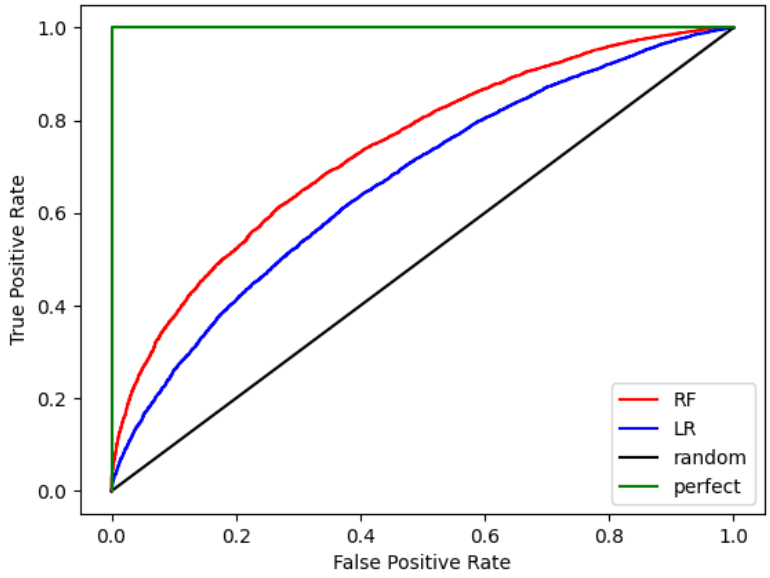
Зміна порогу дозволяє надавати перевагу точності або повноті в залежності від завдань моделі. Вибір порогу залежить від того, чи важливо виявити більше позитивних значень (більша повнота), чи мінімізувати хибно-позитивні передбачення (більша точність).

```
from sklearn.metrics import roc_curve
fpr_RF, tpr_RF, thresholds_RF = roc_curve(df.actual_label.values,
df.model_RF.values)
fpr_LR, tpr_LR, thresholds_LR = roc_curve(df.actual_label.values,
df.model_LR.values)

import matplotlib.pyplot as plt
plt.plot(fpr_RF, tpr_RF,'r-',label = 'RF')
plt.plot(fpr_LR,tpr_LR,'b-', label= 'LR')
plt.plot([0,1],[0,1],'k-',label='random')
plt.plot([0,0,1,1],[0,1,1,1],'g-',label='perfect')
plt.legend()
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.show()
```




```
from sklearn.metrics import roc_auc_score
auc_RF = roc_auc_score(df.actual_label.values, df.model_RF.values)
auc_LR = roc_auc_score(df.actual_label.values, df.model_LR.values)
print('\nAUC RF: %.3f'% auc_RF)
print('AUC LR: %.3f'% auc_LR)

plt.plot(fpr_RF, tpr_RF,'r-',label = 'RF AUC: %.3f'%auc_RF)
plt.plot(fpr_LR,tpr_LR,'b-', label= 'LR AUC: %.3f'%auc_LR)
plt.plot([0,1],[0,1],'k-',label='random')
plt.plot([0,0,1,1],[0,1,1,1],'g-',label='perfect')
plt.legend()
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.show()
```

```
AUC RF: 0.738
AUC LR: 0.666
```



На основі попередньо отриманих результатів можна здійснити порівняння моделей Random Forest (RF) та логістичної регресії (LR).

Модель RF має оцінку якості (accuracy) 0.671, тоді як LR - 0.616. Це свідчить про те, що модель RF має кращу загальну точність порівняно з LR.

Модель RF демонструє оцінку повноти (recall) 0.641, в той час як LR має 0.543. Це означає, що RF краще ідентифікує справжні позитивні випадки.

Точність моделі RF становить 0.681, а у моделі LR - 0.636. Модель RF має меншу кількість хибних позитивних випадків, що вказує на її кращу точність.

Оцінка F1 для RF складає 0.660, тоді як для LR - 0.586. Це показує, що модель RF має кращий баланс між точністю та повнотою.

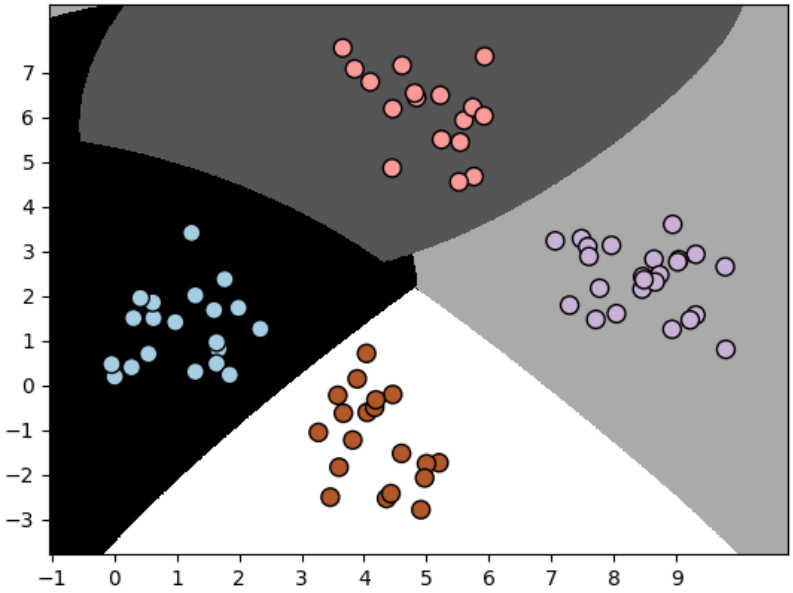
AUC для RF дорівнює 0.738, а для LR - 0.666. Більше значення AUC моделі RF свідчить про її кращу загальну ефективність у розрізненні між класами.

Згідно з даними метриками, модель Random Forest переважає над логістичною регресією для цього набору даних. Модель RF показує кращі результати за всіма основними характеристиками, що робить її ліпшим вибором для цієї задачі.

Завдання 2.6. Розробіть програму класифікації даних в файлі data_multivar_nb.txt за допомогою машини опорних векторів (Support Vector Machine - SVM). Розрахуйте показники якості класифікації. Порівняйте їх з показниками наївного байєсівського класифікатора. Зробіть висновки яку модель класифікації краще обрати і чому.

```
import numpy as np
from sklearn import svm
from utilities import visualize_classifier
from sklearn.model_selection import train_test_split
from sklearn.model_selection import cross_val_score

input_file = 'data_multivar_nb.txt'
data = np.loadtxt(input_file, delimiter=',')
X, y = data[:, :-1], data[:, -1]

classifier = svm.SVC()

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=3)
classifier.fit(X_train, y_train)
y_test_pred = classifier.predict(X_test)

accuracy = 100.0 * (y_test == y_test_pred).sum() / X_test.shape[0]
print("Accuracy of SVM classifier =", round(accuracy, 2), "%")

visualize_classifier(classifier, X_test, y_test)

num_folds = 3
accuracy_values = cross_val_score(classifier, X, y, scoring='accuracy',
cv=num_folds)
print("Accuracy: " + str(round(100 * accuracy_values.mean(), 2)) + "%")
precision_values = cross_val_score(classifier, X, y,
scoring='precision_weighted', cv=num_folds)
print("Precision: " + str(round(100 * precision_values.mean(), 2)) + "%")
recall_values = cross_val_score(classifier, X, y, scoring='recall_weighted',
cv=num_folds)
print("Recall: " + str(round(100 * recall_values.mean(), 2)) + "%")
f1_values = cross_val_score(classifier, X, y, scoring='f1_weighted',
cv=num_folds)
print("F1: " + str(round(100 * f1_values.mean(), 2)) + "%")
```

```
Accuracy of SVM classifier = 100.0 %
Accuracy: 99.75%
Precision: 99.76%
Recall: 99.75%
F1: 99.75%
```



Порівнявши показники якості класифікації за допомогою машини опорних векторів з показниками наївного байєсівського класифікатора (результати завдання 2.4), можемо побачити, що після потрійної перехресної перевірки показники якості обох класифікаторів повністю збігаються. З цього можемо зробити висновок, що для цієї задачі однаково підходять обидва методи, без переваги один над іншим.