ЛАБОРАТОРНА РОБОТА № 2

ПОРІВНЯННЯ МЕТОДІВ КЛАСИФІКАЦІЇ ДАНИХ

Мета роботи: використовуючи спеціалізовані бібліотеки та мову програмування Python дослідити різні методи класифікації даних та навчитися їх порівнювати.

Посилання на гітхаб: https://github.com/PanchukPetro/SShILabsPanchuk/tree/main/Lab2

Завдання 2.1. Класифікація за допомогою машин опорних векторів (SVM)

Набор даних в цьому завданні має такі ознаки:

1. **Age**: Вік особи (числовий, цілочисловий).
2. **Workclass**: Тип зайнятості (категоріальний).
3. **Fnlwgt**: Вага зразка — специфічний фактор ваги в популяційному аналізі (числовий, цілочисловий).
4. **Education**: Рівень освіти (категоріальний).
5. **Education-num**: Кількість років освіти (числовий, цілочисловий).
6. **Marital-status**: Сімейний статус (категоріальний)
7. **Occupation**: Рід діяльності (категоріальний).
8. **Relationship**: Сімейна роль (категоріальний).
9. **Race**: Раса особи (категоріальний).
10. **Sex**: Стать (категоріальний).
11. **Capital-gain**: Прибуток від капіталовкладень (числовий, цілочисловий).
12. **Capital-loss**: Збиток від капіталовкладень (числовий, цілочисловий).
13. **Hours-per-week**: Кількість робочих годин на тиждень (числовий, цілочисловий).
14. **Native-country**: Рідна країна (категоріальний).
15. **Income**: Клас доходу (категоріальний, дві категорії: "<=50K" або ">50K").

Результат виконання програми:

```
Accuracy: 79.56%
Precision: 79.26%
Recall: 79.56%
F1 Score: 75.75%
Predicted class: <=50K
```

Програмний код:

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn import preprocessing
from sklearn.metrics._classification import accuracy_score, precision_score, recall_score,
f1_score
from sklearn.svm import LinearSVC
from sklearn.multiclass import OneVsOneClassifier
from sklearn.model selection import train test split, cross val score

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

# Читання даних
X = []
Y = []
count_class1 = 0
count_class2 = 0
max datapoints = 25000

with open(input file, 'r') as f:
    for line in f.readlines():
        if count_class1 >= max_datapoints and count_class2 >= max_datapoints:
            break
        if '?' in line:
            continue
        data = line.strip().split(', ')
        if data[-1] == '<=50K' and count_class1 < max_datapoints:
            X.append(data[:-1])
            Y.append('<=50K')
            count class1 += 1
        elif data[-1] == '>50K' and count class2 < max datapoints:
            X.append(data[:-1])
            Y.append('>50K')
            count_class2 += 1

# Перетворення на масив numpy
X = np.array(X)
Y = np.array(Y)

# Перетворення рядкових даних на числові
label encoders = []
X encoded = np.empty(X.shape)

for i, item in enumerate(X[0]):
    if item.isdigit():
        X encoded[:, i] = X[:, i].astype(int)
    else:
        le = preprocessing.LabelEncoder()
        X_encoded[:, i] = le.fit_transform(X[:, i])
        label_encoders.append(le)

# Поділ на навчальні і тестові дані
X train, X test, y train, y test = train test split(X encoded, Y, test size=0.2, random state=5)

# Створення SVM-класифікатора
classifier = OneVsOneClassifier(LinearSVC(random state=0))

# Навчання класифікатора
classifier.fit(X_train, y_train)

# Передбачення для тестових даних
y test pred = classifier.predict(X test)

# Обчислення F-міри для SVM-класифікатора
accuracy = accuracy_score(y_test, y_test_pred)
precision = precision_score(y_test, y_test_pred, average='weighted')
recall = recall_score(y_test, y_test_pred, average='weighted')
f1 = f1 score(y test, y test pred, average='weighted')

print(f"Accuracy: {accuracy * 100:.2f}%")
print(f"Precision: {precision * 100:.2f}%")
print(f"Recall: {recall * 100:.2f}%")
print(f"F1 Score: {f1 * 100:.2f}%")

# Передбачення результату для тестової точки даних
input_data = ['37', 'Private', '215646', 'HS-grad', '9', 'Never-married', 'Handlers-cleaners',
'Not-in-family', 'White', 'Male',
```

```
            '0', '0', '40', 'United-States']

# Кодування тестової точки даних
input_data_encoded = np.array([-1] * len(input_data))

count = 0
for i, item in enumerate(input data):
    if item.isdigit():
        input_data_encoded[i] = int(input_data[i])
    else:
        input_data_encoded[i] = label_encoders[count].transform([item])[0]
        count += 1

input data encoded = input data encoded.reshape(1, -1)

# Використання класифікатора для передбачення точки даних
predicted_class = classifier.predict(input_data_encoded)
print("Predicted class:", predicted_class[0])
```

Судячи по результату, тестова точка належить до групи що отримує меньше ніж 50 тисяч в рік

Завдання 2.2. Порівняння якості класифікаторів SVM з нелінійними ядрами

Код спільний для всіх класифікаторів:

```
import numpy as np
from sklearn import preprocessing
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score
from sklearn.svm import SVC

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

# Читання даних
X = []
Y = []
count class1 = 0
count_class2 = 0
max_datapoints = 25000

with open(input file, 'r') as f:
    for line in f.readlines():
        if count class1 >= max datapoints and count class2 >= max datapoints:
            break
        if '?' in line:
            continue
        data = line.strip().split(', ')
        if data[-1] == '<=50K' and count class1 < max datapoints:
            X.append(data[:-1])
            Y.append('<=50K')
            count class1 += 1
        elif data[-1] == '>50K' and count class2 < max datapoints:
            X.append(data[:-1])
            Y.append('>50K')
            count_class2 += 1

# Перетворення на масив numpy
X = np.array(X)
Y = np.array(Y)

# Перетворення рядкових даних на числові
label_encoders = []
X encoded = np.empty(X.shape)

for i, item in enumerate(X[0]):
    if item.isdigit():
        X_encoded[:, i] = X[:, i].astype(int)
    else:
        le = preprocessing.LabelEncoder()
        X encoded[:, i] = le.fit transform(X[:, i])
        label encoders.append(le)

# Поділ на навчальні і тестові дані
```

```
X_train, X_test, y_train, y_test = train_test_split(X_encoded, Y, test_size=0.2, random_state=5)


# Функція для оцінки та виведення метрик
def evaluate_classifier(classifier, X_train, X_test, y_train, y_test):
    classifier.fit(X_train, y_train)
    y test pred = classifier.predict(X test)

    accuracy = accuracy_score(y_test, y_test_pred)
    precision = precision_score(y_test, y_test_pred, average='weighted')
    recall = recall_score(y_test, y_test_pred, average='weighted')
    f1 = f1_score(y_test, y_test_pred, average='weighted')

    print(f"Accuracy: {accuracy * 100:.2f}%")
    print(f"Precision: {precision * 100:.2f}%")
    print(f"Recall: {recall * 100:.2f}%")
    print(f"F1 Score: {f1 * 100:.2f}%")
    print()


# Поліноміальне ядро
#Якщо я ставлю degree >2, воно тренує його 10 років. В degree 1 даже параметри кращі ніж в 2
print("Поліноміальне ядро:")
classifier poly = SVC(kernel='poly', degree=2)
evaluate_classifier(classifier_poly, X_train, X_test, y_train, y_test)
```

Поліноміальне ядро(task 2.1)

```
# Поліноміальне ядро
#Якщо я ставлю degree >2, воно тренує його 10 років. В degree 1 даже параметри кращі ніж в 2
print("Поліноміальне ядро:")
classifier poly = SVC(kernel='poly', degree=2)
evaluate_classifier(classifier_poly, X_train, X_test, y_train, y_test)
```

Гаусове ядро(task 2.2)

```
# Гаусове (RBF) ядро
print("Гаусове (RBF) ядро:")
classifier_rbf = SVC(kernel='rbf')
evaluate_classifier(classifier_rbf, X_train, X_test, y_train, y_test)
```

Сигмоїдальне ядро(task 2.3)

```
# Сигмоїдальне ядро
print("Сигмоїдальне ядро:")
classifier_sigmoid = SVC(kernel='sigmoid')
evaluate_classifier(classifier_sigmoid, X_train, X_test, y_train, y_test)
```

Результати:

```
Поліноміальне ядро:
Accuracy: 77.39%
Precision: 81.11%
Recall: 77.39%
F1 Score: 70.18%
```

```
Гаусове (RBF) ядро:
Accuracy: 78.19%
Precision: 82.82%
Recall: 78.19%
F1 Score: 71.51%
```

```
Сигмоїдальне ядро:
Accuracy: 60.47%
Precision: 60.64%
Recall: 60.47%
F1 Score: 60.55%
```

По результатам видно що Поліноміальне та Гаусове ядра виконують поставлену задачу майже на схожому рівні, з перевісом в +- 1% по кожній метриці для Гаусового ядра. Сигмоїдальне впоралось набагато гірше. Краще всього буде вибрати гаусове ядро

Завдання 2.3. Порівняння якості класифікаторів на прикладі класифікації сортів ірисів

Код для ознайомленням з датасетом:

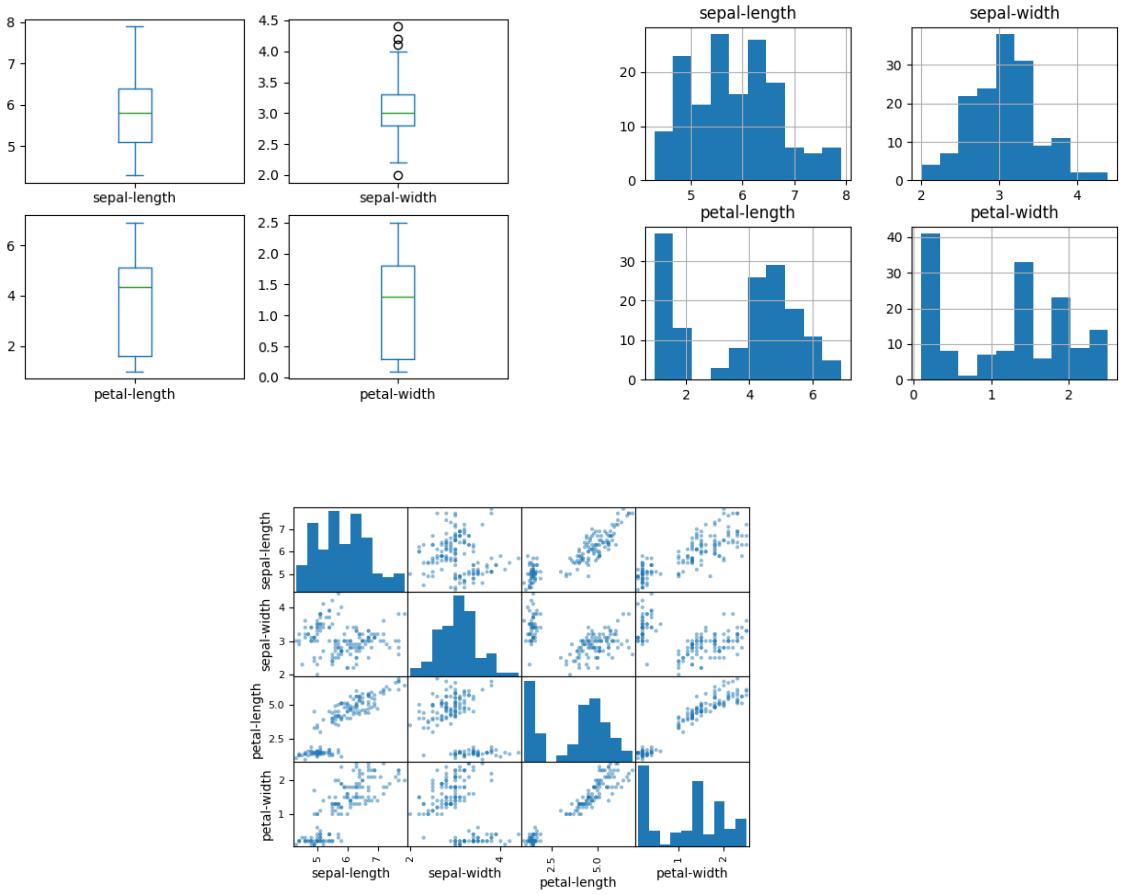
```
from sklearn.datasets import load_iris
iris_dataset = load_iris()

print("Ключі iris_dataset: \n{}".format(iris_dataset.keys()))
print(iris_dataset['DESCR'][:193] + "\n...")
print("Назви відповідей: {}".format(iris_dataset['target_names']))
print("Назва ознак: \n{}".format(iris_dataset['feature_names']))
print("Тип масиву data: {}".format(type(iris_dataset['data'])))
print("Форма масиву data:{}".format(iris_dataset['data'].shape))
print("Тип масиву target:{}".format(type(iris_dataset['target'])))
print("Відповіді:\n{}".format(iris_dataset['target']))
```

Результат:

```
Ключі iris_dataset:
dict_keys(['data', 'target', 'frame', 'target_names', 'DESCR', 'feature_names', 'filename', 'data_module'])
.. _iris_dataset:

Iris plants dataset
--------------------

**Data Set Characteristics:**

:Number of Instances: 150 (50 in each of three classes)
:Number of Attributes: 4 numeric, predictive
...
Назви відповідей: ['setosa' 'versicolor' 'virginica']
Назва ознак:
['sepal length (cm)', 'sepal width (cm)', 'petal length (cm)', 'petal width (cm)']
Тип масиву data: <class 'numpy.ndarray'>
Форма масиву data:(150, 4)
Тип масиву target:<class 'numpy.ndarray'>
Відповіді:
[0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
 2 2]
```

Код для візуалізації:

```
# Діаграма розмаху
dataset.plot(kind='box', subplots=True, layout=(2,2),
sharex=False, sharey=False)
pyplot.show()

# Гістограма розподілу атрибутів датасета
dataset.hist()
pyplot.show()

#Матриця діаграм розсіювання
scatter matrix(dataset)
pyplot.show()
```



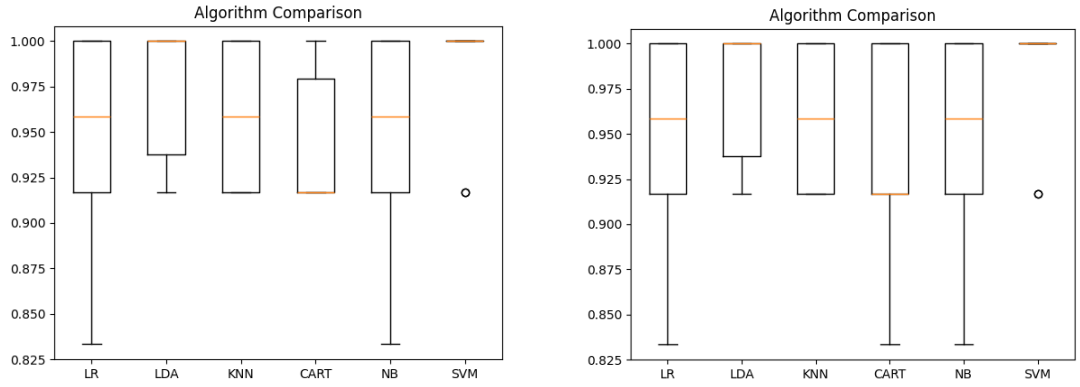
Отримані графіки

Код для перевірки точності алгоритмів

```
results = []
names = []

for name, model in models:
    kfold = StratifiedKFold(n splits=10, random state=1, shuffle=True)
    cv results = cross val score(model, X train, Y train,cv=kfold, scoring='accuracy')
    results.append(cv_results)
    names.append(name)
    print('%s: %f (%f)' % (name, cv_results.mean(),cv_results.std()))
```

Код для візуалізації порівняння алгоритмів:

```
# Порівняння алгоритмів
pyplot.boxplot(results, tick labels=names)
pyplot.title('Algorithm Comparison')
pyplot.show()
```



Результати візуалізації

Дивлячись на графіки ми бачимо, що CART метод веде себе непередбачувано та постійно міняється після кожного запуску програми. Серед цих методів найбільш виділяється SVM, в ньому відсутній розподіл значень точності. Всі лінії знаходяться на рівні 1.0, що є ідеальним. Проте в ньому є один аутлаєр, який з'являється також стабільно. Я думаю що вибрати SVM буде найкращим вибором, outlier можна проігнорувати в даному випадку.

Також алгоритми показали таку загальну точність:

LR: 0.941667 (0.065085)

LDA: 0.975000 (0.038188)

KNN: 0.958333 (0.041667)

CART: 0.941667 (0.038188)

NB: 0.950000 (0.055277)

SVM: 0.983333 (0.033333)

Також SVM показує себе найкраще

```
# Прогнозування окремого входження за допомогою натренованої моделі
newdataset = np.array([[5, 2.9, 1, 0.2]])

svm_model = [model for name, model in models if name == 'SVM'][0]
svm_model.fit(X_train, Y_train)
prediction = svm_model.predict(newdataset)
print("////////////////////////////")
print("Прогноз: " + str(prediction[0]))
print("////////////////////////////")
```

Результат прогнозу:

```
////////////////////////////
Прогноз: Iris-setosa
////////////////////////////
```

Далі буде надано весь код програми та все виведення в консоль:

```
# Завантаження бібліотек
from fontTools.misc.textTools import tostr
from pandas import read csv
from pandas.plotting import scatter_matrix
from matplotlib import pyplot
from sklearn.model selection import train test split
from sklearn.model selection import cross val score
from sklearn.model selection import StratifiedKFold
from sklearn.metrics import classification_report
from sklearn.metrics import confusion_matrix
from sklearn.metrics import accuracy_score
from sklearn.linear model import LogisticRegression
from sklearn.multiclass import OneVsRestClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
from sklearn.naive_bayes import GaussianNB
from sklearn.svm import SVC
import numpy as np

url = "https://raw.githubusercontent.com/jbrownlee/Datasets/master/iris.csv"
names = ['sepal-length', 'sepal-width', 'petal-length', 'petal-width', 'class']
dataset = read_csv(url, names=names)

print(dataset.shape)

# Зріз даних head
print(dataset.head(20))
# Стастичні зведення методом describe
print(dataset.describe())
print(dataset.groupby('class').size())

# Діаграма розмаху
dataset.plot(kind='box', subplots=True, layout=(2,2),
sharex=False, sharey=False)
pyplot.show()

# Гістограма розподілу атрибутів датасета
dataset.hist()
pyplot.show()

#Матриця діаграм розсіювання
scatter_matrix(dataset)
pyplot.show()

# Розділення датасету на навчальну та контрольну вибірки
array = dataset.values
# Вибір перших 4-х стовпців
X = array[:,0:4]
# Вибір 5-го стовпця
y = array[:,4]
# Разделение X и y на обучающую и контрольную выборки
X train, X validation, Y train, Y validation = train test split(X, y,test size=0.20,
random_state=1)

# Завантажуємо алгоритми моделі
models = []
#models.append(('LR', LogisticRegression(solver='liblinear',multi class='ovr')))
models.append(('LR', OneVsRestClassifier(LogisticRegression()))) #LR multi_class був деприкейтед
говорить
models.append(('LDA', LinearDiscriminantAnalysis()))
models.append(('KNN', KNeighborsClassifier()))
models.append(('CART', DecisionTreeClassifier()))
models.append(('NB', GaussianNB()))
models.append(('SVM', SVC(gamma='auto')))
results = []
names = []

for name, model in models:
    kfold = StratifiedKFold(n splits=10, random state=1, shuffle=True)
    cv_results = cross_val_score(model, X_train, Y_train,cv=kfold, scoring='accuracy')
    results.append(cv_results)
    names.append(name)
    print('%s: %f (%f)' % (name, cv results.mean(),cv results.std()))

# Порівняння алгоритмів
pyplot.boxplot(results, tick_labels=names)
```

```
pyplot.title('Algorithm Comparison')
pyplot.show()


# Створюємо прогноз на контрольній вибірці
model = SVC(gamma='auto')
model.fit(X train, Y train)
predictions = model.predict(X validation)
# Оцінюємо прогноз
print("Оцінка прогноза:")
print(accuracy_score(Y_validation, predictions))
print(confusion_matrix(Y_validation, predictions))
print(classification report(Y validation, predictions))

# Прогнозування окремого входження за допомогою натренованої моделі
newdataset = np.array([[5, 2.9, 1, 0.2]])

svm_model = [model for name, model in models if name == 'SVM'][0]
svm model.fit(X train, Y train)
prediction = svm model.predict(newdataset)
print("/////////////////////////////")
print("Прогноз: " + str(prediction[0]))
print("/////////////////////////////")
```

```
(150, 5)
    sepal-length  sepal-width  petal-length  petal-width        class
0            5.1          3.5           1.4          0.2  Iris-setosa
1            4.9          3.0           1.4          0.2  Iris-setosa
2            4.7          3.2           1.3          0.2  Iris-setosa
3            4.6          3.1           1.5          0.2  Iris-setosa
4            5.0          3.6           1.4          0.2  Iris-setosa
5            5.4          3.9           1.7          0.4  Iris-setosa
6            4.6          3.4           1.4          0.3  Iris-setosa
7            5.0          3.4           1.5          0.2  Iris-setosa
8            4.4          2.9           1.4          0.2  Iris-setosa
9            4.9          3.1           1.5          0.1  Iris-setosa
10           5.4          3.7           1.5          0.2  Iris-setosa
11           4.8          3.4           1.6          0.2  Iris-setosa
12           4.8          3.0           1.4          0.1  Iris-setosa
13           4.3          3.0           1.1          0.1  Iris-setosa
14           5.8          4.0           1.2          0.2  Iris-setosa
15           5.7          4.4           1.5          0.4  Iris-setosa
16           5.4          3.9           1.3          0.4  Iris-setosa
17           5.1          3.5           1.4          0.3  Iris-setosa
18           5.7          3.8           1.7          0.3  Iris-setosa
19           5.1          3.8           1.5          0.3  Iris-setosa
       sepal-length  sepal-width  petal-length  petal-width
count    150.000000   150.000000    150.000000   150.000000
mean       5.843333     3.054000      3.758667     1.198667
std        0.828066     0.433594      1.764420     0.763161
min        4.300000     2.000000      1.000000     0.100000
25%        5.100000     2.800000      1.600000     0.300000
50%        5.800000     3.000000      4.350000     1.300000
75%        6.400000     3.300000      5.100000     1.800000
max        7.900000     4.400000      6.900000     2.500000
class
Iris-setosa        50
Iris-versicolor    50
Iris-virginica     50
dtype: int64
LR: 0.941667 (0.065085)
LDA: 0.975000 (0.038188)
KNN: 0.958333 (0.041667)
CART: 0.941667 (0.038188)
NB: 0.950000 (0.055277)
SVM: 0.983333 (0.033333)
Оцінка прогноза:
0.9666666666666667
[[11  0  0]
 [ 0 12  1]
 [ 0  0  6]]
```

Виведення в консоль частина 1

```
[[11  0  0]
 [ 0 12  1]
 [ 0  0  6]]
                precision    recall  f1-score   support

    Iris-setosa       1.00      1.00      1.00        11
Iris-versicolor       1.00      0.92      0.96        13
 Iris-virginica       0.86      1.00      0.92         6

       accuracy                           0.97        30
      macro avg       0.95      0.97      0.96        30
   weighted avg       0.97      0.97      0.97        30

////////////////////////////
Прогноз: Iris-setosa
////////////////////////////
```

Виведення в консоль частина 2

Висновки: Нам вдалось добитись якості класифікації 0.966, а спрогнозовано було, що описана нами квітка входить до виду “Iris-setosa”

Завдання 2.4. Порівняння якості класифікаторів для набору даних завдання 2.1

Програмний код:

```
input_file = 'income_data.txt'

# Читання даних
X = []
y = []
count_class1 = 0
count_class2 = 0
max datapoints = 25000

with open(input_file, 'r') as f:
    for line in f.readlines():
        if count_class1 >= max_datapoints and count_class2 >= max_datapoints:
            break
        if '?' in line:
            continue
        data = line.strip().split(', ')
        if data[-1] == '<=50K' and count_class1 < max_datapoints:
            X.append(data[:-1])
            y.append('<=50K')
            count class1 += 1
        elif data[-1] == '>50K' and count_class2 < max_datapoints:
            X.append(data[:-1])
            y.append('>50K')
            count class2 += 1

# Перетворення на масив numpy
X = np.array(X)
y = np.array(y)

# Перетворення рядкових даних на числові
label encoders = []
X_encoded = np.empty(X.shape)

for i, item in enumerate(X[0]):
    if item.isdigit():
        X encoded[:, i] = X[:, i].astype(int)
    else:
        le = preprocessing.LabelEncoder()
        X_encoded[:, i] = le.fit_transform(X[:, i])
        label encoders.append(le)

# Поділ на навчальні і тестові дані
X_train, X_validation, y_train, y_validation = train_test_split(X_encoded, y, test_size=0.2,
random_state=5)

# Завантажуємо алгоритми моделі
models = []
#models.append(('LR', LogisticRegression(solver='liblinear',multi class='ovr')))
models.append(('LR', OneVsRestClassifier(LogisticRegression()))) #LR multi_class був деприкейтед
говорить
models.append(('LDA', LinearDiscriminantAnalysis()))
models.append(('KNN', KNeighborsClassifier()))
models.append(('CART', DecisionTreeClassifier()))
models.append(('NB', GaussianNB()))
models.append(('SVM', SVC(kernel='poly', degree=2)))
results = []
names = []

for name, model in models:
    kfold = StratifiedKFold(n_splits=10, random_state=1, shuffle=True)
    cv_results = cross_val_score(model, X_train, y_train,cv=kfold, scoring='accuracy')
    results.append(cv_results)
    names.append(name)
    print('%s: %f (%f)' % (name, cv_results.mean(),cv_results.std()))
```

```
LR: 0.784699 (0.004054)
LDA: 0.811637 (0.005701)
KNN: 0.767748 (0.003026)
CART: 0.806706 (0.007076)
NB: 0.789133 (0.006934)
SVM: 0.781176 (0.003242)
```

Результат порівняння

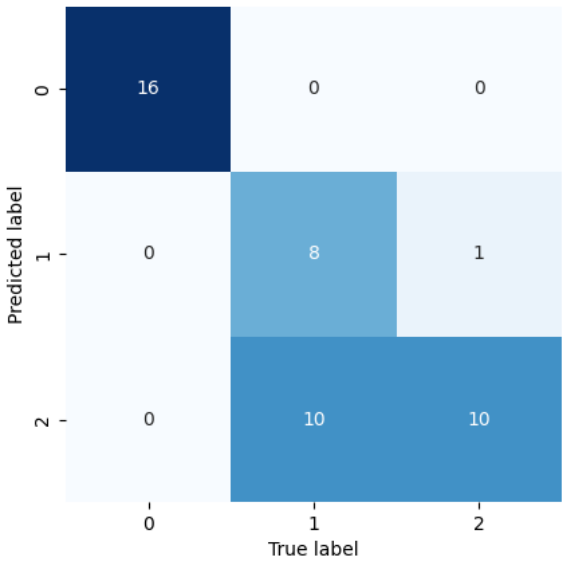
Завдання 2.5. Класифікація даних лінійним класифікатором Ridge

Програмний код:

```
# ======================================================================
# Приклад класифікатора Ridge
# ======================================================================
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.datasets import load_iris
from sklearn.linear_model import RidgeClassifier
from sklearn.model_selection import train_test_split
from sklearn import metrics
from io import BytesIO

# Завантаження даних Iris
iris = load iris()
X, y = iris.data, iris.target

# Розділення на тренувальну і тестову вибірки
Xtrain, Xtest, ytrain, ytest = train test split(X, y, test size=0.3, random state=0)

# Створення та навчання класифікатора Ridge
clf = RidgeClassifier(tol=1e-2, solver="sag")
clf.fit(Xtrain, ytrain)

# Прогнозування
ypred = clf.predict(Xtest)

# Оцінка якості моделі
print('Accuracy:', np.round(metrics.accuracy_score(ytest, ypred), 4))
print('Precision:', np.round(metrics.precision score(ytest, ypred, average='weighted'), 4))
print('Recall:', np.round(metrics.recall score(ytest, ypred, average='weighted'), 4))
print('F1 Score:', np.round(metrics.f1 score(ytest, ypred, average='weighted'), 4))
print('Cohen Kappa Score:', np.round(metrics.cohen_kappa_score(ytest, ypred), 4))
print('Matthews Corrcoef:', np.round(metrics.matthews_corrcoef(ytest, ypred), 4))

# Звіт про класифікацію
print('\t\tClassification Report:\n', metrics.classification report(ytest, ypred))

# Побудова матриці плутанини
mat = metrics.confusion_matrix(ytest, ypred)
sns.heatmap(mat.T, square=True, annot=True, fmt='d', cbar=False, cmap="Blues")
plt.xlabel('True label')
plt.ylabel('Predicted label')

# Збереження зображення у форматі JPG
plt.savefig("Confusion.jpg")

# Збереження зображення у форматі SVG у віртуальний файл
f = BytesIO()
plt.savefig(f, format="svg")
plt.show()
```

```
Accuracy: 0.7556
Precision: 0.8333
Recall: 0.7556
F1 Score: 0.7503
Cohen Kappa Score: 0.6431
Matthews Corrcoef: 0.6831
        Classification Report:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        16
           1       0.89      0.44      0.59        18
           2       0.50      0.91      0.65        11

    accuracy                           0.76        45
   macro avg       0.80      0.78      0.75        45
weighted avg       0.83      0.76      0.75        45
```

Виведення в консоль



Матриця плутанини

Налаштування класифікатора Ridge

1. **tol=1e-2**: Параметр толерантності для критеріїв зупинки оптимізації. Значення 1e-2 означає, що алгоритм припиняє оптимізацію, коли зміна значення функції втрат між ітераціями стає меншою за 10−210^{-2}10−2.
2. **solver="sag"**: Використовується алгоритм **SAG** (Stochastic Average Gradient Descent), який добре підходить для великих датасетів. Він обчислює середнє градієнтів з урахуванням лише частини даних.

Використовуються такі показники якості:

1. **Accuracy** (Точність):
   Результат: 0.7556
2. **Precision** (Прецизійність):
   Результат: 0.8333
3. **Recall** (Чутливість):
   Результат: 0.7556
4. **F1 Score**:
   Результат: 0.7503
5. **Cohen Kappa Score** (Коефіцієнт Коена Каппа):
   Результат: 0.6431
6. **Matthews Corrcoef** (Коефіцієнт кореляції Метьюза):
   Результат: 0.6831

Коефіцієнт Коена Каппа використовується для оцінки узгодженості між двома оцінювачами або моделями, коли вони класифікують об'єкти в категорії. Він враховує узгодженість, яка могла статися випадково, і є більш надійним показником, ніж проста точність у класифікаційних завданнях.

Коефіцієнт кореляції Метьюза використовується для оцінки якості двокласової класифікації і є більш збалансованим показником, особливо коли класи дисбалансовані. Він враховує всі чотири комірки матриці невідповідності: TP, TN, FP, FN.