Міністерство освіти і науки України
НТУУ «КПІ ім. Ігоря Сікорського»
Навчально-науковий інститут атомної та теплової енергетики
Кафедра цифрових технологій в енергетиці

Лабораторна робота №4

з дисципліни «Вступ до інтелектуального аналізу даних»

Тема «Машинне навчання в Python»

Варіант №19

Студента 3-го курсу НН ІАТЕ гр. ТР-12
Ковальова Олександра
Перевірив: д.т.н., проф. Путренко В. В.

КИЇВ 2024

Мета: Опрацювати приклад роботи з машинним навчанням, використовуючи Jupyter Notebook. Виконати поставлене завдання.

Хід роботи

Метод K-Means:

In [1]:
```
from sklearn.datasets import load_wine
from scipy.cluster.vq import kmeans2, whiten

[data, target] = load_wine(return_X_y=True)
whitened = whiten(data)

start = [whitened[0],
         whitened[int(len(whitened)/2)],
         whitened[len(whitened)-1]]
[centroid, label] = kmeans2(whitened, start)

errors = 0.0
for i in range(len(label)):
    if target[i] != label[i]:
        errors += 1
        acc = ((len(data) - errors)
               /len(data))
        print (acc)
```

Цей код використовує алгоритм K-means для кластеризації даних, що стосуються вин. Для реалізації цього використовуються бібліотеки scikit-learn та scipy.

Найперше, дані про вина завантажуються з набору даних за допомогою load_wine з scikit-learn, а саме load_wine(return_X_y=True) повертає дані та мітки класів. Далі, за допомогою функції whiten з бібліотеки scipy, дані "відбілюються", тобто стандартизуються для полегшення процесу кластеризації.

Початкові центри кластерів обираються з відбіленого набору даних, і алгоритм K-means використовується для кластеризації. Результати цього процесу - центроїди кластерів та мітки для кожного зразка - зберігаються у відповідних змінних.

Після кластеризації виконується порівняння справжніх міток класів (з target) і отриманих міток від K-means (з label). Якщо вони не співпадають, збільшується лічильник помилок. Далі, розраховується точність кластеризації (acc), яка представляє собою відношення кількості правильно класифікованих зразків до загальної кількості зразків. Отримані точності виводяться на екран.

Цей код не лише кластеризує дані про вина, але й відображає результати кластеризації та точність в порівнянні зі справжніми мітками класів. Такий підхід дозволяє оцінити ефективність алгоритму K-means для конкретного набору даних.

Результат:

```
0.9943820224719101
0.9887640449438202
0.9831460674157303
0.9775280898876404
0.9719101123595506
0.9662921348314607
0.9606741573033708
0.9550561797752809
```

Побудова моделі класифікації за допомогою дерева рішень:

```
In [3]: from sklearn.datasets import load_wine
        from sklearn import tree

        [data, target] = load_wine(return_X_y=True)

        data_train = data[0:len(data)-2]
        target_train = target[0:len(data)-2]

        dtc = tree.DecisionTreeClassifier()
        dtc = dtc.fit(data_train,
                      target_train)
        print (dtc.predict(
            data[len(data)-1]
            .reshape(1,-1)))
```

У цьому коді використовуються бібліотеки scikit-learn для завантаження даних про вина (wine dataset) та побудови моделі класифікації з використанням дерева рішень.

Спочатку, дані та їх мітки завантажуються з набору даних про вина за допомогою load_wine(return_X_y=True). Потім дані розділяються на тренувальний набір (data_train, target_train) та тестовий зразок. У цьому випадку, останні два зразки відокремлюються для тестування.

Після цього створюється модель класифікатора за допомогою дерева рішень (DecisionTreeClassifier()), яка потім навчається на тренувальних даних (fit(data_train, target_train)).

Остання стрічка коду використовує навчену модель для передбачення класу останнього зразка у тестовому наборі. Результат передбачення виводиться на екран за допомогою print(dtc.predict(data[len(data)-1].reshape(1, -1))).

Цей код демонструє простий приклад використання дерева рішень для класифікації даних про вина. Дерево рішень розділяє признаки у вузли так, щоб максимізувати чистоту класів у кожному листі, що дозволяє ефективно класифікувати нові зразки. Результат передбачення показує клас, до якого належить останній зразок у тестовому наборі.

Результат:

```
[2]
```

Побудова моделі класифікації з використанням методу опорних векторів (SVM):

```
In [8]: from sklearn.datasets import load_wine
        from sklearn import svm

        [data, target] = load_wine(return_X_y=True)
        data_train = data[0:len(data)-2]
        target_train = target[0:len(target)-2]

        # clf = svm.LinearSVC()
        clf = svm.LinearSVC(dual="auto")
        clf.fit(data_train, target_train)
        print (clf.predict(
            data[len(data)-1]
            .reshape(1,-1)))
```

Спочатку, дані та їх мітки завантажуються з набору даних про вина за допомогою load_wine(return_X_y=True). Далі дані розділяються на тренувальний набір (data_train, target_train). У цьому випадку, останні два зразки відокремлюються для тестування.

Модель класифікатора створюється за допомогою методу опорних векторів (SVM) з використанням класу LinearSVC. Опція dual="auto" вказує системі самостійно вибрати оптимальний метод для розв'язку двоїстої задачі оптимізації.

Навчання моделі виконується за допомогою методу fit(data_train, target_train), де дані навчаються на тренувальному наборі.

Остання стрічка коду використовує навчену модель для передбачення класу останнього зразка у тестовому наборі. Результат передбачення виводиться на екран за допомогою print(clf.predict(data[len(data)-1].reshape(1, -1))).

Метод опорних векторів (SVM) використовує геометричний підхід для класифікації даних. У випадку лінійного ядра, як в даному випадку з параметром LinearSVC, модель намагається розділити класи гіперплощиною так, щоб максимізувати відстань між класами. Результат передбачення вказує на клас, до якого належить останній зразок у тестовому наборі.

Результат:

```
[2]
```

Висновок: Під час виконання лабораторної роботи були набуті практичні навички роботи з методами, які є складовими сімейства алгоритмів машинного навчання та використовуються для різних завдань, таких як кластеризація, класифікація та регресія, а саме: K-means, SVM (Support Vector Machine) та дерево рішень.

Програмний код

Notebook.ipynb:

```
from sklearn.datasets import load_wine
from scipy.cluster.vq import kmeans2, whiten

[data, target] = load_wine(return_X_y=True)
whitened = whiten(data)

start = [whitened[0],
         whitened[int(len(whitened)/2)],
         whitened[len(whitened)-1]]
[centroid, label] = kmeans2(whitened, start)

errors = 0.0
for i in range(len(label)):
    if target[i] != label[i]:
        errors += 1
        acc = ((len(data) - errors)
               /len(data))
        print (acc)

from sklearn.datasets import load_wine
from sklearn import tree

[data, target] = load_wine(return_X_y=True)

data_train = data[0:len(data)-2]
target_train = target[0:len(data)-2]

dtc = tree.DecisionTreeClassifier()
dtc = dtc.fit(data_train,
              target_train)
print (dtc.predict(
    data[len(data)-1]
    .reshape(1,-1)))

from sklearn.datasets import load_wine
from sklearn import svm

[data, target] = load_wine(return_X_y=True)
data_train = data[0:len(data)-2]
target_train = target[0:len(target)-2]

# clf = svm.LinearSVC()
clf = svm.LinearSVC(dual="auto")
clf.fit(data_train, target_train)
print (clf.predict(
    data[len(data)-1]
    .reshape(1,-1)))
```