Лабораторна робота №6

Посилання на гіт-хаб: <https://github.com/ViMIMercurysMight/python-ai>

Тема: Створення рекомендаційних систем

Завдання 1: Створення навчального конвеєра (конвеєра машинного навчання)

Необхідно створити конвеєр, призначений для вибору найбільш важливих ознак з вхідних даних і їх подальшої класифікації з використанням класифікатора на основі гранично випадкового лісу.

*Лістинг програми:*

from sklearn.datasets import \_samples\_generator  
from sklearn.feature\_selection import SelectKBest, f\_regression  
from sklearn.pipeline import Pipeline  
from sklearn.ensemble import ExtraTreesClassifier  
  
X, y = \_samples\_generator.make\_classification(n\_samples=150,  
 n\_features=25,  
 n\_classes=3,  
 n\_informative=6,  
 n\_redundant=0,  
 random\_state=7  
 )  
k\_best\_selector = SelectKBest(f\_regression, k=9)  
classifier = ExtraTreesClassifier(n\_estimators=60, max\_depth=4)  
process\_pipeline = Pipeline([('selector', k\_best\_selector), ('erf', classifier)])  
  
  
process\_pipeline.set\_params(selector\_\_k=7, erf\_\_n\_estimators=30)  
process\_pipeline.fit(X, y)  
output = process\_pipeline.predict(X)  
print("\nPredicted output:\n", output)  
  
print("\nScore:", process\_pipeline.score(X, y))  
status = process\_pipeline.named\_steps['selector'].get\_support()  
selected = [i for i, x in enumerate(status) if x]  
print("\nIndices of selected features: ", ", ".join([str(x) for x in selected]))

A screenshot of a computer

Description automatically generated with medium confidence

Рисунок 1. Результати виконання

Як можна побачити з виводу результату виконання (рис.1 ) В першому виводі данних міститься масив передбачення для усього набору даних, наступним рядком виведено оцінку навчання яка отримана шляхом трансформаціх данних, і останнім рядком вивденно вибрані шляхом аналізу вхідного набору дані для «рекомендації» їх.

Завдання 2: Пошук найближчих сусідів

*Лістинг програми:*

#Task 2  
import numpy as np  
import matplotlib.pyplot as plt  
from sklearn.neighbors import NearestNeighbors  
  
X = np.array([[2.1, 1.3], [1.3, 3.2], [2.9, 2.5], [2.7, 5.4],  
 [3.8, 0.9], [7.3, 2.1], [4.2, 6.5], [3.8, 3.7],  
 [2.5, 4.1], [3.4, 1.9], [5.7, 3.5], [6.1, 4.3],  
 [5.1, 2.2], [6.2, 1.1]])  
k = 5  
  
test\_datapoint = [4.3, 2.7]  
  
plt.figure()  
plt.title('Input data')  
plt.scatter(X[:, 0], X[:, 1], marker='o', s=75, color='black')  
  
  
knn\_model = NearestNeighbors(n\_neighbors=k, algorithm='ball\_tree').fit(X)  
  
distances, indices = knn\_model.kneighbors([test\_datapoint])  
  
print("\nK Nearest Neighbors: ")  
for rank, index in enumerate(indices[0][:k], start=1):  
 print(str(rank) + " ==>", X[index])  
plt.figure()  
plt.title('Nearest Neighbors')  
plt.scatter(X[:, 0], X[:, 1], marker='o', s=75, color='k')  
plt.scatter(X[indices][0][:][:,0], X[indices][0][:][:,1],  
 marker='o', s=250, color='k', facecolors='none')  
plt.scatter(test\_datapoint[0], test\_datapoint[1], marker='x', s=75, color='k')  
plt.show()

Chart, scatter chart

Description automatically generated

Рисунок 2. Результат виконання програми

Як можна бачити на першому графыку зображені усі сусіди, на другому визначені найближчі у середньому сусіди. В вікно терміналу було виведено дані про (координати тестової точки) цих п’яти найближчих сусідів

Завдання 3: Створити класифікатор методом k найближчих сусідів

*Лістинг програми:*

#Task 3  
import numpy as np  
import matplotlib.pyplot as plt  
import matplotlib.cm as cm  
from sklearn import neighbors, datasets  
  
input\_file = ‘data.txt’  
data = np.loadtxt(input\_file, delimiter=’,’)  
X, y = data[:, :-1], data[:, -1].astype(np.int)  
  
plt.figure()  
plt.title(“InputData”)  
marker\_shapes = ‘v^os’  
mapper = [marker\_shapes[i] for I in y]  
for I in range(X.shape[0]):  
 plt.scatter(X[I, 0],  
 X[I, 1],  
 marker=mapper[i],  
 s=75,  
 edgecolors=’black’,  
 facecolors=’none’)  
  
num\_neighbors = 12  
step\_size = 0.01  
  
#-----  
Classifier = neighbors.KneighborsClassifier(num\_neighbors,  
 weights=’distance’)  
Classifier.fit(X, y)  
  
x\_min, x\_max = X[:, 0].min() – 1, X[:, 0].max() + 1  
y\_min, y\_max = X[:, 1].min() – 1, X[:, 1].max() + 1  
x\_values, y\_values = np.meshgrid(np.arange(x\_min, x\_max, step\_size),  
 np.arange(y\_min, y\_max, step\_size))  
  
output = Classifier.predict(np.c\_[x\_values.ravel(),  
 y\_values.ravel()])  
  
output = output.reshape(x\_values.shape)  
plt.figure()  
plt.pcolormesh(x\_values, y\_values, output, cmap=cm.Paired)  
  
for I in range(X.shape[0]):  
 plt.scatter(X[I, 0], X[I, 1],  
 marker=mapper[i],  
 s=50,  
 edgecolors=’black’,  
 facecolors=’none’)  
  
plt.xlim(x\_values.min(), x\_values.max())  
plt.ylim(y\_values.min(), y\_values.max())  
plt.title(‘Edges model classifier on base K nearest neighbors’)  
  
test\_datapoint = [5.1, 3.6]  
plt.figure()  
plt.title(‘Test data point’)  
for I in range(X.shape[0]):  
 plt.scatter(X[I, 0], X[I, 1], marker=mapper[i],  
 s=75, edgecolors=’black’, facecolors=’none’)  
  
plt.scatter(test\_datapoint[0], test\_datapoint[1], marker=’x’, linewidths=6,  
 s=200, facecolors=’black’)  
  
\_, indices = Classifier.kneighbors([test\_datapoint])  
indices = indices.astype(np.int)[0]  
  
plt.figure()  
plt.title(‘K nearest neighbors’)  
for I in indices:  
 plt.scatter(X[I, 0],  
 X[I, 1],  
 marker=mapper[y[i]],  
 linewidths=3,  
 s=100,  
 facecolors=’black’)  
plt.scatter(test\_datapoint[0], test\_datapoint[1], marker=’x’,  
 linewidths=6, s=200, facecolors=’black’)  
  
for I in range(X.shape[0]):  
 plt.scatter(X[I, 0], X[I, 1], marker=mapper[i], s=75,  
 edgecolors=’black’, facecolors=’none’)  
  
print(‘Predicted output:’, Classifier.predict([test\_datapoint])[0])  
plt.show()

Scatter chart

Description automatically generated



Рисунок 4. Результато виконання

Результато виконання програми стало чотири діагрми. На першій було нанесено усі точки віднесені до різних типів. На другому визначено центр, на третьому ці точки було віднесено до чотирох кластерів з позначенням їх меж, на останньому було визначено усі найближчі до центра точки й зафарбовано їх чорним.

Сама тестова точка відноситься до класу 1

Завдання 4: Обчислення оцінки подібності

*Лістинг програми*:

#Task 4  
  
import argparse  
import json  
import numpy as np  
  
def build\_arg\_parser() :  
 parser = argparse.ArgumentParser(description='Compute similarity score')  
 parser.add\_argument('--user1', dest='user1', required=True, help='First user')  
 parser.add\_argument('--user2', dest='user2', required=True, help='Second user')  
 parser.add\_argument('--score-type',  
 dest='score\_type',  
 required=True,  
 choices=['Euclidean', 'Pearson'],  
 help='Similarity metric to be user')  
 return parser  
  
# Обчислення оцінки евклідова відстані між  
# користувачами userl та user2  
def euclidean\_score(dataset, user1, user2):  
 if user1 not in dataset:  
 raise TypeError('Cannot find ' + user1 + ' in the dataset')  
 if user2 not in dataset:  
 raise TypeError('Cannot find ' + user2 + ' in the dataset')  
 # Фільми, оцінені обома користувачами, userl та user2  
 common\_movies = {}  
  
 for item in dataset[user1]:  
 if item in dataset[user2]:  
 common\_movies[item] = 1  
  
 # За відсутності фільмів, оцінених обома користувачами,  
 # оцінка приймається рівною 0  
 if len(common\_movies) == 0:  
 return 0  
  
 squared\_diff = []  
  
 for item in dataset[user1]:  
 if item in dataset[user2]:  
 squared\_diff.append(np.square(dataset[user1][item]  
 - dataset[user2][item]))  
 return 1 / (1 + np.sqrt(np.sum(squared\_diff)))  
  
#Коеф. кореляції Пірсона  
def pearson\_score(dataset, user1, user2):  
 if user1 not in dataset:  
 raise TypeError('Cannot find ' + user1 + ' in the dataset')  
 if user2 not in dataset:  
 raise TypeError('Cannot find ' + user2 + ' in the dataset')  
  
 common\_movies = {}  
  
 for item in dataset[user1]:  
 if item in dataset[user2]:  
 common\_movies[item] = 1  
  
 num\_ratings = len(common\_movies)  
  
 # За відсутності фільмів, оцінених обома користувачами,  
 # оцінка приймається рівною 0  
 if num\_ratings == 0:  
 return 0  
  
 # Обчислення суми рейтингових оцінок усіх фільмів,  
 # оцінених обома користувачами  
 user1\_sum = np.sum([dataset[user1][item] for item in common\_movies])  
 user2\_sum = np.sum([dataset[user2][item] for item in common\_movies])  
  
 # Обчислення Суми квадратів рейтингових оцінок всіх  
 # фільмів, оцінених обома користувачами  
 user1\_squared\_sum = np.sum([np.square(dataset[user1][item])  
 for item in common\_movies])  
 user2\_squared\_sum = np.sum([np.square(dataset[user2][item])  
 for item in common\_movies])  
 # Обчислення суми творів рейтингових оцінок всіх  
 # фільмів, оцінених обома користувачами  
 sum\_of\_products = np.sum([dataset[user1][item] \*  
 dataset[user2][item] for item in common\_movies])  
  
 # Обчислення коефіцієнта кореляції Пірсона  
 Sxy = sum\_of\_products - (user1\_sum \* user2\_sum / num\_ratings)  
 Sxx = user1\_squared\_sum - np.square(user1\_sum) / num\_ratings  
 Syy = user2\_squared\_sum - np.square(user2\_sum) / num\_ratings  
  
 if Sxx \* Syy == 0:  
 return 0  
  
 return Sxy / np.sqrt(Sxx \* Syy)  
  
if \_\_name\_\_=='\_\_main\_\_':  
 args = build\_arg\_parser().parse\_args()  
 user1 = args.user1  
 user2 = args.user2  
 score\_type = args.score\_type  
  
ratings\_file = 'ratings.json'  
  
with open(ratings\_file, 'r') as f:  
 data = json.loads(f.read())  
  
if score\_type == 'Euclidean':  
 print("\nEuclidean score:")  
 print(euclidean\_score(data, user1, user2))  
else:  
 print("\nPearson score:")  
 print(pearson\_score(data, user1, user2))

Проведемо тестові запуски программи для користувачів David Smith та Bill Duffy (рис.3)

Text

Description automatically generated with medium confidenceText

Description automatically generated

Рисунок 3. Результат запиту програми

Проведемо запити программи для пар: David Smith та Brenda Peterson, David Smith та Samuel Miller, David Smith та Julie Hammel, David Smith та Clarissa Jackson , David Smith та Adam Cohen

David Smith та Chris Duncan (рис 4)

Text

Description automatically generated

Рисунок 4. Результат запиту для багатьох користувачів

Завдання 5: Пошук користувачів зі схожими уподобаннями методом колаборативної фільтрації

Для полегшення тестування програми, код оцінювання з попереднього завдання було імплементовано напрюму в поточне

*Лістинг програми*:

#Task 5  
import argparse  
import json  
import numpy as np  
  
  
def euclidean\_score(dataset, user1, user2):  
 if user1 not in dataset:  
 raise TypeError('Cannot find ' + user1 + ' in the dataset')  
 if user2 not in dataset:  
 raise TypeError('Cannot find ' + user2 + ' in the dataset')  
  
 common\_movies = {}  
  
 for item in dataset[user1]:  
 if item in dataset[user2]:  
 common\_movies[item] = 1  
  
 if len(common\_movies) == 0:  
 return 0  
  
 squared\_diff = []  
  
 for item in dataset[user1]:  
 if item in dataset[user2]:  
 squared\_diff.append(np.square(dataset[user1][item]  
 - dataset[user2][item]))  
 return 1 / (1 + np.sqrt(np.sum(squared\_diff)))  
  
  
def pearson\_score(dataset, user1, user2):  
 if user1 not in dataset:  
 raise TypeError('Cannot find ' + user1 + ' in the dataset')  
 if user2 not in dataset:  
 raise TypeError('Cannot find ' + user2 + ' in the dataset')  
  
 common\_movies = {}  
  
 for item in dataset[user1]:  
 if item in dataset[user2]:  
 common\_movies[item] = 1  
  
 num\_ratings = len(common\_movies)  
 if num\_ratings == 0:  
 return 0  
  
 user1\_sum = np.sum([dataset[user1][item] for item in common\_movies])  
 user2\_sum = np.sum([dataset[user2][item] for item in common\_movies])  
  
 user1\_squared\_sum = np.sum([np.square(dataset[user1][item])  
 for item in common\_movies])  
 user2\_squared\_sum = np.sum([np.square(dataset[user2][item])  
 for item in common\_movies])  
  
 sum\_of\_products = np.sum([dataset[user1][item] \*  
 dataset[user2][item] for item in common\_movies])  
 Sxy = sum\_of\_products - (user1\_sum \* user2\_sum / num\_ratings)  
 Sxx = user1\_squared\_sum - np.square(user1\_sum) / num\_ratings  
 Syy = user2\_squared\_sum - np.square(user2\_sum) / num\_ratings  
  
 if Sxx \* Syy == 0:  
 return 0  
  
 return Sxy / np.sqrt(Sxx \* Syy)  
  
  
def pearson\_score(dataset, user1, user2):  
 if user1 not in dataset:  
 raise TypeError('Cannot find ' + user1 + ' in the dataset')  
 if user2 not in dataset:  
 raise TypeError('Cannot find ' + user2 + ' in the dataset')  
  
 common\_movies = {}  
  
 for item in dataset[user1]:  
 if item in dataset[user2]:  
 common\_movies[item] = 1  
  
 num\_ratings = len(common\_movies)  
 if num\_ratings == 0:  
 return 0  
  
 user1\_sum = np.sum([dataset[user1][item] for item in common\_movies])  
 user2\_sum = np.sum([dataset[user2][item] for item in common\_movies])  
  
 user1\_squared\_sum = np.sum([np.square(dataset[user1][item])  
 for item in common\_movies])  
 user2\_squared\_sum = np.sum([np.square(dataset[user2][item])  
 for item in common\_movies])  
  
 sum\_of\_products = np.sum([dataset[user1][item] \*  
 dataset[user2][item] for item in common\_movies])  
 Sxy = sum\_of\_products - (user1\_sum \* user2\_sum / num\_ratings)  
 Sxx = user1\_squared\_sum - np.square(user1\_sum) / num\_ratings  
 Syy = user2\_squared\_sum - np.square(user2\_sum) / num\_ratings  
  
 if Sxx \* Syy == 0:  
 return 0  
  
 return Sxy / np.sqrt(Sxx \* Syy)  
  
  
def find\_similar\_users(dataset, user, num\_users):  
 if user not in dataset:  
 raise TypeError('Cannot find ' + user + ' in the dataset')  
 # Обчислення оцінки подібності за Пірсоном між  
 # вказаним користувачем та всіма іншими  
 # користувачами в наборі даних  
 scores = np.array([[x, pearson\_score(dataset, user, x)] for x in dataset if x != user])  
 # Сортування оцінок за спаданням  
 scores\_sorted = np.argsort(scores[:, 1])[::-1]  
 # Вилучення оцінок перших 'num\_users' користувачів  
 top\_users = scores\_sorted[:num\_users]  
 return scores[top\_users]  
  
user = "Bill Duffy"  
#user = "Clarissa Jackson"  
ratings\_file = 'ratings.json'  
  
with open(ratings\_file, 'r') as f:  
 data = json.loads(f.read())  
  
print('\nUSers similar to ' + user + ':\n')  
similar\_users = find\_similar\_users(data, user, 3)  
print('User\t\t\tSimilarity score')  
print('-'\*41)  
  
for item in similar\_users:  
 print(item[0], '\t\t', round(float(item[1]), 2))

Text

Description automatically generatedText

Description automatically generated with medium confidence

Рисунок 5. Результат виконання роботи

Шляхом використання методу колаборативної фільтрації було знайдено користувачів зі схожими вподобаннями.

Завдання 6: Створення рекомендаційної системи фільмів

Подібно до попереднього завдання код попереднів прикладів було імплементовано в код програми напряму без імпорутвання.

*Лістинг програми*

#Task 6  
import argparse  
import json  
import numpy as np  
  
  
def pearson\_score(dataset, user1, user2):  
 if user1 not in dataset:  
 raise TypeError('Cannot find ' + user1 + ' in the dataset')  
 if user2 not in dataset:  
 raise TypeError('Cannot find ' + user2 + ' in the dataset')  
  
 common\_movies = {}  
  
 for item in dataset[user1]:  
 if item in dataset[user2]:  
 common\_movies[item] = 1  
  
 num\_ratings = len(common\_movies)  
 if num\_ratings == 0:  
 return 0  
  
 user1\_sum = np.sum([dataset[user1][item] for item in common\_movies])  
 user2\_sum = np.sum([dataset[user2][item] for item in common\_movies])  
  
 user1\_squared\_sum = np.sum([np.square(dataset[user1][item])  
 for item in common\_movies])  
 user2\_squared\_sum = np.sum([np.square(dataset[user2][item])  
 for item in common\_movies])  
  
 sum\_of\_products = np.sum([dataset[user1][item] \*  
 dataset[user2][item] for item in common\_movies])  
 Sxy = sum\_of\_products - (user1\_sum \* user2\_sum / num\_ratings)  
 Sxx = user1\_squared\_sum - np.square(user1\_sum) / num\_ratings  
 Syy = user2\_squared\_sum - np.square(user2\_sum) / num\_ratings  
  
 if Sxx \* Syy == 0:  
 return 0  
  
 return Sxy / np.sqrt(Sxx \* Syy)  
  
  
def build\_arg\_parser():  
 parser = argparse.ArgumentParser(description=  
 'Find users who are similar to the input user ')  
 parser.add\_argument('--user', dest='user', required=True, help='Input user')  
 return parser  
  
  
def find\_similar\_users(dataset, user, num\_users):  
 if user not in dataset:  
 raise TypeError('Cannot find ' + user + ' in the dataset')  
 scores = np.array([[x, pearson\_score(dataset, user, x)] for x in dataset if x != user])  
 scores\_sorted = np.argsort(scores[:, 1])[::-1]  
 top\_users = scores\_sorted[:num\_users]  
 return scores[top\_users]  
  
# Отримати рекомендації щодо фільмів  
# для вказаного користувача  
def get\_recommendations(dataset, input\_user):  
 if input\_user not in dataset:  
 raise TypeError('Cannot find ' + input\_user + ' in the dataset')  
  
 overall\_scores = {}  
 similarity\_scores = {}  
  
 for user in [x for x in dataset if x != input\_user]:  
 similarity\_score = pearson\_score(dataset, input\_user, user)  
 if similarity\_score <= 0:  
 continue  
 filtered\_list = [x for x in dataset[user] if x not in \  
 dataset[input\_user] or dataset[input\_user][x] == 0]  
  
 for item in filtered\_list:  
 overall\_scores.update({item: dataset[user][item] \* similarity\_score})  
 similarity\_scores.update({item: similarity\_score})  
  
 if len(overall\_scores) == 0:  
 return ["No recommendations possible"]  
 # Генерація рейтингів фільмів за допомогою їх нормалізації  
 movie\_scores = np.array([[score/similarity\_scores[item],  
 item] for item, score in overall\_scores.items()])  
 # Сортування за спаданням  
 movie\_scores = movie\_scores[np.argsort(movie\_scores[:, 0])[::-1]]  
 # Вилучення рекомендацій фільмів  
 movie\_recommendations = [movie for \_, movie in movie\_scores]  
 return movie\_recommendations  
  
user = "Chris Duncan"  
#user = "Julie Hammel"  
  
ratings\_file = 'ratings.json'  
with open(ratings\_file, 'r') as f:  
 data = json.loads(f.read())  
  
print("\n Movie recommendations for " + user + ":")  
movies = get\_recommendations(data, user)  
for i, movie in enumerate(movies):  
 print(str(i+1) + '. ' + movie)

Text

Description automatically generatedGraphical user interface, text

Description automatically generated

Рисунок 5. Результат виконання програми

Процес пошуку рекомендацій в фільмах в деякому сенсі подібний до процесу пошуку корисувачів зі схожими вподобаннями, але в якості основи було обрано оцінки фільмів. Для розширення можливостей системи можливо розширити фільтр для використання таких критеріїв як жанр, режисер тощо, що в цілому не є складною задачею.

Висновок: Розглянули способи побудови рекомендаційних систем, та оцінювання.