Song Recommendation system

April 25, 2024

1 Assignment 5 - Classic rock radio songs Recommender System and build Neural Networks only with numpy

2 1. Collaborative Filtering

[17]:
```
import pandas as pd
import numpy as np
from scipy.spatial.distance import cosine

# Loading the user-item matrix.
user_item_matrix = pd.read_csv("radio_songs.csv", index_col=0)

user_item_matrix = user_item_matrix + 1e-8 # Adding a small value to avoid␣
↪division by zero

# Defining function:
def item_item_cosine_similarity(item1, item2):
    return 1 - cosine(item1, item2)

# Computing similarity between all songs using cosine similarity
song_similarity = {}
for song1 in user_item_matrix.columns:
    song_similarity[song1] = {}
    for song2 in user_item_matrix.columns:
        if song1 != song2:
            similarity = item_item_cosine_similarity(user_item_matrix[song1],␣
↪user_item_matrix[song2])
            song_similarity[song1][song2] = similarity

# Finding top 10 similar songs to 'u2' and 'pink floyd'
similarities = song_similarity['u2']
sorted_similarities = sorted(similarities.items(), key=lambda x: x[1],␣
↪reverse=True)
recommended_songs = [song[0] for song in sorted_similarities[:10]]

print("Top 10 recommended songs for users who have listened to 'U2' and 'Pink␣
↪Floyd':")
```

```
print(recommended_songs)
```

```
Top 10 recommended songs for users who have listened to 'U2' and 'Pink Floyd':
['misfits', 'robbie williams', 'green day', 'depeche mode', 'peter fox', 'dire
straits', 'madonna', 'enter shikari', 'kelly clarkson', 'johnny cash']
```

#B.Find user most similar to user 1606. Use user-user collaborative filtering with cosine similarity. #List the recommended songs for user 1606 (Hint: find the songs listened to by the most similar user).

[2]:
```
import pandas as pd
import numpy as np
from scipy.spatial.distance import cosine

# Loading the user-item matrix from the CSV file
user_item_matrix = pd.read_csv('radio_songs.csv', index_col=0)

# Replacing NaN values with 0
user_item_matrix.fillna(0, inplace=True)

# Ensuring there are no rows with all zeros
user_item_matrix = user_item_matrix.loc[(user_item_matrix != 0).any(axis=1)]

# Defining a function to calculate cosine similarity between two users
def calculate_similarity(user1, user2):
    return 1 - cosine(user_item_matrix.loc[user1], user_item_matrix.loc[user2])

# Calculating cosine similarity between user 1606 and all other users
similarities = {}
for user in user_item_matrix.index:
    if user != 1606:  # Exclude user 1606 itself
        similarity = calculate_similarity(1606, user)
        similarities[user] = similarity

# Finding the user with the highest cosine similarity to user 1606
most_similar_user = max(similarities, key=similarities.get)

# Retrieving the songs listened to by the most similar user
recommended_songs = user_item_matrix.loc[most_similar_user][user_item_matrix.
 ↪loc[1606] == 0]

# Printing the recommended songs for user 1606
print("Recommended songs for user 1606 based on user-user collaborative␣
 ↪filtering:")
print(recommended_songs.head(10))  # Print the top 10 recommended songs
```

```
Recommended songs for user 1606 based on user-user collaborative filtering:
ac/dc                    0
adam green               0
```

```
aerosmith            0
afi                  0
air                  0
alanis morissette    0
alexisonfire         0
alicia keys          0
all that remains     0
amon amarth          0
Name: 1144, dtype: int64
```

#C. How many of the recommended songs has already been listened to by user 1606?

[3]:
```python
# Finding the songs listened to by user 1606
listened_songs_1606 = user_item_matrix.loc[1606][user_item_matrix.loc[1606] !=␣
↪0].index

# Counting the number of recommended songs that user 1606 has already listened␣
↪to
num_already_listened = recommended_songs.index.isin(listened_songs_1606).sum()

print("Number of recommended songs already listened to by user 1606:",␣
↪num_already_listened)
```

```
Number of recommended songs already listened to by user 1606: 0
```

[ ]:
```python
# D.Use a combination of user-item approach to build a recommendation score for␣
↪each song for each user
#using the following steps for each user-
```

[16]:
```python
def recommend_songs(user_id, top_n=5):
    user_row = pd.Series(user_item_matrix.loc[user_id].values,␣
↪index=user_item_matrix.columns)
    similarities = cosine_similarity([user_row], user_item_matrix.values)
    most_similar_user_index = np.argsort(similarities[0])[-2]
    most_similar_user_row = pd.Series(user_item_matrix.
↪iloc[most_similar_user_index].values, index=user_item_matrix.columns)
    recommended_songs = most_similar_user_row[most_similar_user_row == 1].index.
↪difference(user_row[user_row == 1].index)

    # Calculating recommendation score for each song
    recommendation_scores = {}
    for song in recommended_songs:
        # Get the similarity score for the song
        similarity_score = most_similar_user_row[song]
        # Calculate the recommendation score for the song
        recommendation_score = similarity_score / np.sum(similarities)
        recommendation_scores[song] = recommendation_score
```

```
    # Sortinge songs based on recommendation score
    sorted_recommendations = sorted(recommendation_scores.items(), key=lambda x:
 ↪ x[1], reverse=True)

    # Returninge top recommended songs
    return [song[0] for song in sorted_recommendations[:top_n]]

 # Sortinge songs based on recommendation score
    sorted_recommendations = sorted(recommendation_scores.items(), key=lambda x:
 ↪ x[1], reverse=True)

    # Returninge top recommended songs
    return [song[0] for song in sorted_recommendations[:top_n]]

# Getting 5 songs recommendation:
top_recommendations_1606 = recommend_songs(1606)
print("Top 5 song recommendations for user 1606:")
print(top_recommendations_1606)
```

```
Top 5 song recommendations for user 1606:
['beastie boys', 'bob dylan', 'bob marley & the wailers', 'david bowie', 'eric
clapton']
```

3 2. Conceptual questions:

[ ]:
```
#1.Pearson correlation coefficient and Jaccard similarity coefficient

#2.To build a Content-Based Recommender System, you need data on the items to␣
 ↪be recommended
#along with their detailed characteristics, such as text, metadata, or feature␣
 ↪vectors.

#3. Accuracy Metrics ( f-1 score, precision,recall and accuracy) and User␣
 ↪Studies
```

4 3. Neural Network using numpy:

[31]:
```
from sklearn.datasets import make_gaussian_quantiles
import matplotlib.pyplot as plt

samples = 2000

def load_extra_datasets(N):
    gaussian_quantiles = make_gaussian_quantiles(mean=None,
                                                  cov=0.7,
                                                  n_samples=N,
```

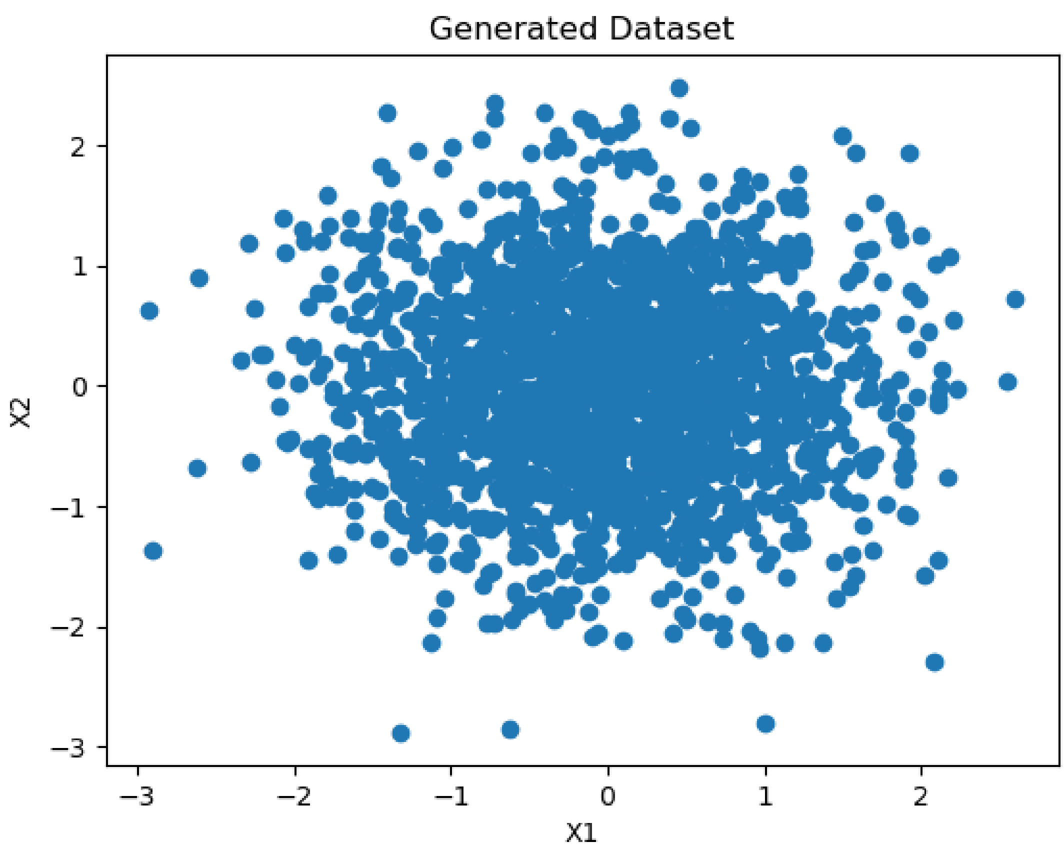
```
                                                    n_features=2,
                                                    n_classes=2,
                                                    shuffle=True,
                                                    random_state=None)
    return  gaussian_quantiles

gaussian_quantiles = load_extra_datasets(samples)

X, Y = gaussian_quantiles

X, Y = X.T, Y.reshape(1, Y.shape[0])

plt.scatter(X[0, :], X[1, :])
plt.xlabel('X1')
plt.ylabel('X2')
plt.title('Generated Dataset')
plt.show()
```



[32]:
```
n_x = X.shape[0] # size of input layer`
n_h = 4
n_y = Y.shape[0] # size of output layer

print(n_x, n_y)
```

```
2 1
```

[33]:
```
import numpy as np

W1 = np.random.randn(n_h,n_x) * 0.01
b1 = np.zeros(shape=(n_h, 1))
W2 = np.random.randn(n_y,n_h) * 0.01
b2 = np.zeros(shape=(n_y, 1))

print("W1\n", W1)
print("b1\n", b1)
print("W2\n", W2)
print("b2\n", b2)
```

```
W1
 [[ 0.01483774  0.0005294 ]
 [-0.01164417 -0.01091096]
 [-0.01316516 -0.00801637]
 [ 0.00588233  0.00389455]]
b1
 [[0.]
 [0.]
 [0.]
 [0.]]
W2
 [[-0.01282769 -0.00274775  0.0022777  -0.00076836]]
b2
 [[0.]]
```

[34]:
```
def sigmoid(x):
    return 1 / (1 + np.e ** -x)

total_cost = -9999
```

[35]:
```
# Implement Forward Propagation to calculate A2 (probabilities)
Z1 = np.dot(W1,X) + b1
A1 = np.tanh(Z1)
Z2 = np.dot(W2,A1) + b2
A2 = sigmoid(Z2) # Final output prediction

print(b2)
```

```
[[0.]]
```

[36]:
```
# Compute the cross-entropy cost
old_total_cost = total_cost
cost_function = np.multiply(np.log(A2), Y) + np.multiply((1 - Y), np.log(1 -␣
↪A2)) #J(theta)
total_cost = -np.sum(cost_function) / samples

print("cost=", total_cost)
print("cost delta=", np.subtract(total_cost, old_total_cost))
```

```
cost= 0.693146168495335
cost delta= 9999.693146168496
```

[37]:
```
print(Z1.shape)
print(A1.shape)
print(Z2.shape)
print(A2.shape)
print(cost_function.shape)
```

```
(4, 2000)
(4, 2000)
(1, 2000)
(1, 2000)
(1, 2000)
```

[38]:
```
dJdZ2 = A2 - Y
dJdW2 = (1 / samples) * np.dot(dJdZ2, A1.T)
dJdb2 = (1 / samples) * np.sum(dJdZ2, axis=1, keepdims=True)
# since activation function is tanh(Z1) = A1
# first derivative of d/dz tanh(z) = 1 - tanh(z) ^ 2 = 1 - A1 ^ 2
dJdZ1 = np.multiply(np.dot(W2.T, dJdZ2), 1 - np.power(A1, 2))
dJdW1 = (1 / samples) * np.dot(dJdZ1, X.T)
dJdb1 = (1 / samples) * np.sum(dJdZ1, axis=1, keepdims=True)

print("dJdZ2=", dJdZ2)
print("dJdW2=", dJdW2)
print("dJdb2=", dJdb2)
print("dJdW1=", dJdW1)
print("dJdb1=", dJdb1)
```

```
dJdZ2= [[ 0.50003958  0.49999537  0.50002254 … -0.49998566 -0.49998431
  -0.50003651]]
dJdW2= [[ 7.72011306e-05 -5.42885894e-05 -6.39720552e-05  2.84047825e-05]]
dJdb2= [[1.25798911e-06]]
dJdW1= [[-6.69612547e-05  7.69601079e-06]
 [-1.43495333e-05  1.65323284e-06]
 [ 1.18927832e-05 -1.36952626e-06]
 [-4.01512936e-06  4.61370399e-07]]
dJdb1= [[-7.53034123e-07]
 [-1.68314453e-07]
```

```
 [ 1.35129378e-07]
 [-1.02548355e-08]]
```

[39]:
```
learning_rates = [0.0001, 0.01, 1]

for learning_rate in learning_rates:
    # Making a copy of b2
    b2_old = b2.copy()

    # Updating the weights and biases
    W1 -= learning_rate * dJdW1
    b1 -= learning_rate * dJdb1
    W2 -= learning_rate * dJdW2
    b2 -= learning_rate * dJdb2

    # Printing the delta change in b2 after 10 iterations
    print(f"For learning rate {learning_rate}:")
    print(f"Delta change in b2: {np.subtract(b2, b2_old)}")

    # Reassigning b2_old for the next iteration
    b2_old = b2.copy()

    print("b2 before=", b2_old)
```

```
For learning rate 0.0001:
Delta change in b2: [[-1.25798911e-10]]
b2 before= [[-1.25798911e-10]]
For learning rate 0.01:
Delta change in b2: [[-1.25798911e-08]]
b2 before= [[-1.270569e-08]]
For learning rate 1:
Delta change in b2: [[-1.25798911e-06]]
b2 before= [[-1.2706948e-06]]
```

[40]:
```
#As the learning rate increases, the delta change in weight b2 also increases.
#Higher learning rates lead to larger updates in the weights, causing more␣
↪significant changes in b2.
```