worksheet_17

May 1, 2024

1 Worksheet 17

Name: Honghao Zhao UID: U44266035

1.0.1 Topics

- Recommender Systems

1.0.2 Recommender Systems

In the example in class of recommending movies to users we used the movie rating as a measure of similarity between users and movies and thus the predicted rating for a user is a proxy for how highly a movie should be recommended. So the higher the predicted rating for a user, the higher a recommendation it would be.

a) Consider a streaming platform that only has "like" or "dislike" (no 1-5 rating). Describe how you would build a recommender system in this case.

1. Calculate the similarity between items (e.g., movies, songs) based on the pattern of likes and dislikes they receive. Recommend items that are similar to the items a user has liked.
2. Find users with similar liking patterns to the target user using the same binary similarity metrics, and recommend items that these similar users have liked but the target user hasn't yet rated.
3. Adapt matrix factorization methods to binary data. Techniques like logistic matrix factorization, which uses a logistic function to model the probability that a user likes an item, can be effective.
4. Implement neural network architectures that can learn from binary inputs, such as autoencoders, which can reconstruct a user's like/dislike profile and suggest new items based on learned representations.

b) Describe 3 challenges of building a recommender system

1. Most users only rate a small fraction of the total available items, leading to a sparse matrix of user-item interactions. This sparsity can make it difficult to find similar items or users and to accurately predict preferences.
2. As the number of users and items grows, the computational complexity of the recommender system can increase significantly. Efficiently processing and making recommendations in real-time can become challenging.

3. New users or items with few or no ratings present a challenge because there is insufficient data to make accurate recommendations. For new users, you don't know their preferences; for new items, you lack user feedback.

c) Why is SVD not an option for collaborative filtering?

Traditional SVD is not designed to handle missing data, which is a common issue in collaborative filtering where the user-item matrix is typically sparse. SVD requires a fully populated matrix or needs modifications to work around missing entries. Also, SVD can be computationally intensive, especially for very large matrices, making it less practical for real-time recommendation systems where the user-item matrix is frequently updated.

d) Use the code below to train a recommender system on a dataset of amazon movies

[4]:
```
!pip install findspark
!pip install pyspark
```

```
Requirement already satisfied: findspark in /usr/local/lib/python3.10/dist-
packages (2.0.1)
Collecting pyspark
  Downloading pyspark-3.5.1.tar.gz (317.0 MB)
                           317.0/317.0

MB 4.6 MB/s eta 0:00:00
  Preparing metadata (setup.py) … done
Requirement already satisfied: py4j==0.10.9.7 in /usr/local/lib/python3.10/dist-
packages (from pyspark) (0.10.9.7)
Building wheels for collected packages: pyspark
  Building wheel for pyspark (setup.py) … done
  Created wheel for pyspark: filename=pyspark-3.5.1-py2.py3-none-any.whl
size=317488491
sha256=b57f1008c0fed404633302dafb3e91e57b3a201827452dbe90de2bf4375c64c9
  Stored in directory: /root/.cache/pip/wheels/80/1d/60/2c256ed38dddce2fdd93be54
5214a63e02fbd8d74fb0b7f3a6
Successfully built pyspark
Installing collected packages: pyspark
Successfully installed pyspark-3.5.1
```

[8]:
```
from google.colab import files

uploaded = files.upload()

import pandas as pd
import io

df = pd.read_csv(io.BytesIO(uploaded['./train.csv']))
print(df.head())
```

```
<IPython.core.display.HTML object>
```

```
Saving train.csv to train (2).csv
```

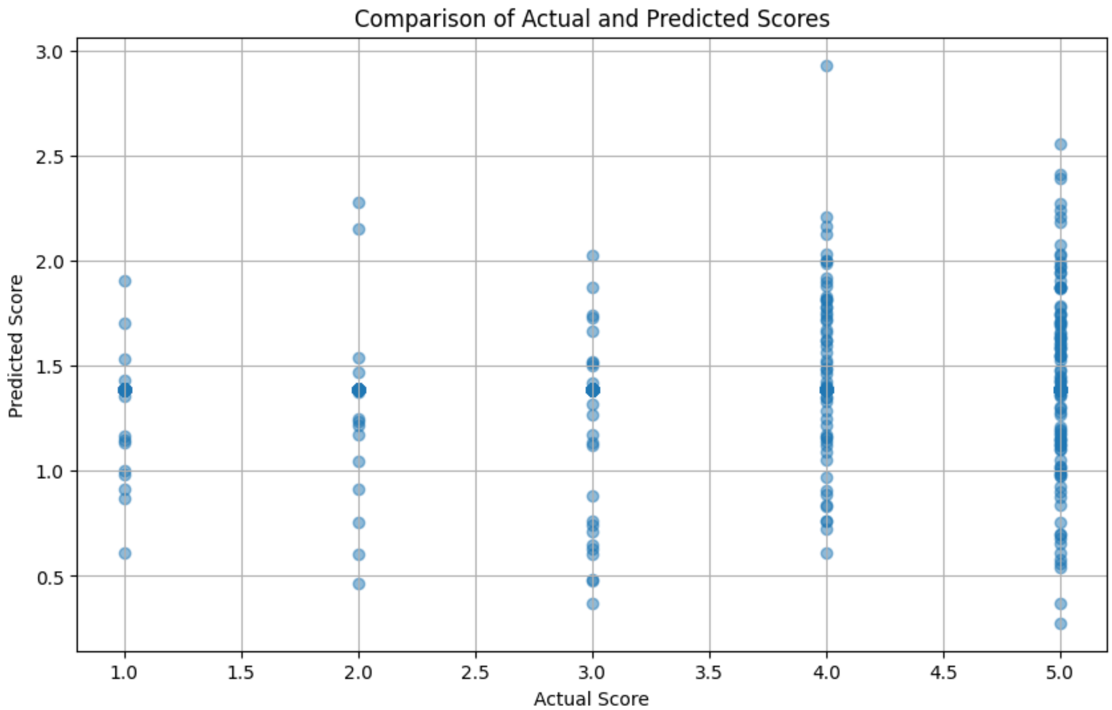
```
---------------------------------------------------------------------------
KeyError                                  Traceback (most recent call last)
<ipython-input-8-708bf444bb95> in <cell line: 8>()
      6 import io
      7
----> 8 df = pd.read_csv(io.BytesIO(uploaded['./train.csv']))
      9 print(df.head())

KeyError: './train.csv'
```

[14]:
```python
# Note: requires py3.10
import findspark
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt

from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error, confusion_matrix

from pyspark.sql import SparkSession
from pyspark import SparkConf, SparkContext
from pyspark.ml.recommendation import ALS
from pyspark.ml.tuning import ParamGridBuilder, CrossValidator

findspark.init()
conf = SparkConf()
conf.set("spark.executor.memory","28g")
conf.set("spark.driver.memory", "28g")
conf.set("spark.driver.cores", "8")
sc = SparkContext.getOrCreate(conf)
spark = SparkSession.builder.getOrCreate()

init_df = pd.read_csv("./train.csv").dropna()
init_df['UserId_fact'] = init_df['UserId'].astype('category').cat.codes
init_df['ProductId_fact'] = init_df['ProductId'].astype('category').cat.codes

# Split training set into training and testing set
X_train_processed, X_test_processed, Y_train, Y_test = train_test_split(
        init_df.drop(['Score'], axis=1),
        init_df['Score'],
        test_size=1/4.0,
        random_state=0
    )

X_train_processed['Score'] = Y_train
```

```python
df = spark.createDataFrame(X_train_processed[['UserId_fact', 'ProductId_fact', 'Score']])
als = ALS(
    userCol="UserId_fact",
    itemCol="ProductId_fact",
    ratingCol="Score",
    coldStartStrategy="drop",
    nonnegative=True,
    rank=100
)
# param_grid = ParamGridBuilder().addGrid(
        # als.rank, [10, 50]).addGrid(
        # als.regParam, [.1]).addGrid(
        # # als.maxIter, [10]).build()
# evaluator = RegressionEvaluator(
        # metricName="rmse",
        # labelCol="Score",
        # # predictionCol="prediction")
# cv = CrossValidator(estimator=als, estimatorParamMaps=param_grid, evaluator=evaluator, numFolds=3, parallelism = 6)
# cv_fit = cv.fit(df)
# rec_sys = cv_fit.bestModel

rec_sys = als.fit(df)
# rec_sys.save('rec_sys.obj') # so we don't have to re-train it
rec = rec_sys.transform(spark.createDataFrame(X_test_processed[['UserId_fact', 'ProductId_fact']])).toPandas()
predictions_pd = rec_sys.transform(spark.createDataFrame(X_test_processed[['UserId_fact', 'ProductId_fact']])).toPandas()

X_test_processed = X_test_processed.merge(predictions_pd, on=['UserId_fact', 'ProductId_fact'], how='left')
X_test_processed['prediction'].fillna(X_test_processed['prediction'].mean(), inplace=True)
X_test_processed['Score'] = X_test_processed['prediction']
X_test_processed.drop(columns=['prediction'], inplace=True)

print("Kaggle RMSE = ", mean_squared_error(X_test_processed['Score'], Y_test, squared=False))

plt.figure(figsize=(10, 6))
plt.scatter(Y_test, X_test_processed['Score'], alpha=0.5)
plt.title('Comparison of Actual and Predicted Scores')
plt.xlabel('Actual Score')
plt.ylabel('Predicted Score')
```

```
plt.grid(True)
plt.show()
```

```
Kaggle RMSE =  2.9769618266414257
```



2 New Section

```
[ ]:
```