Assignment Lesson 10 - 11

- Deep Learning & Recommendation Engine -

In this assignment, you will try to perform a simple Deep Learning process and make a recommendation engine using the provided datasets.

Part A: Deep Learning using MNIST Dataset

Prior Knowledge

Data Science Objective

Objective

The MNIST database (Modified National Institute of Standards and Technology database) is a large database of handwritten digits that is commonly used for training various image processing systems. The database is also widely used for training and testing in the field of machine learning.

The MNIST database contains 60,000 training images and 10,000 testing images. Half of the training set and half of the test set were taken from NIST's training dataset, while the other half of the training set and the other half of the test set were taken from NIST's testing dataset. The black and white images from NIST were normalized to fit into a 28x28 pixel bounding box and anti-aliased, which introduced grayscale levels. The goal is to **implement a Convolutional Neural Network to classify MNIST handwritten digit images**.
Source: Wikipedia

Link

Rapidminer

MNIST Dataset (46 MB)

This dataset contains two main folders: training and test. Both folders are further separated into 10 folders, each containing a handwritten file of a number.

Python

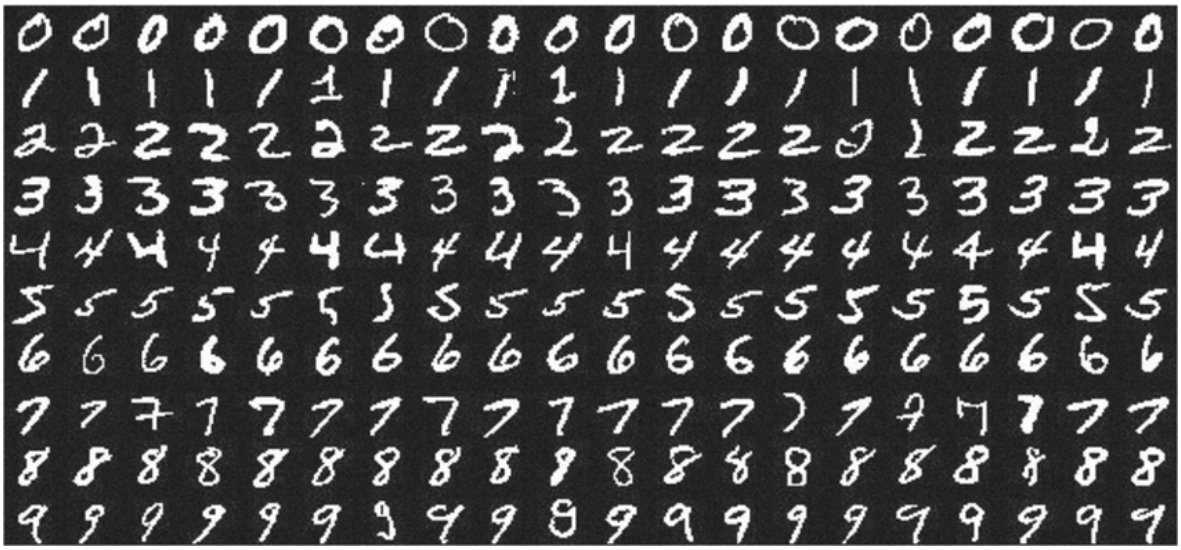
You can also use the following code to retrieve the dataset:

```
from keras.datasets import mnist
(x_train, y_train), (x_test, y_test) = mnist.load_data()
```

Data Preparation & Exploration

Dataset

This is the screenshot of some of the handwritten images in the dataset. Each image has a size of 28 x 28 pixels.



Classification Modeling

Deep Learning 24/30 10/10 9/10

You are free to make any deep learning model you like to solve this problem. Clearly show the network architecture with the parameters involved in each layer. The final goal is to make a model that could classify the test image with more than 99.00% accuracy.
Here are the criteria that will be evaluated:

- Number of layers used (Max 5 points) 4/5
- Type of layers used (Max 5 points) 3/5
- Parameters used (Max 5 points) 5/5
- Any accuracy that is more than 90% (Max 10 points) 8/10
- Extra Model Validation & Parameter Optimization (Max 5 points) 4/5

```
(x_train_full, y_train_full), (x_test, y_test) = mnist.load_data()
```

```
_BATCH = 128
_NB_SPLIT = int(x_train_full.shape[0]*0.1)
_SHAPE = x_train_full.shape[1]
_NORMALIZATION = 255.0
```

```
x_train, x_valid, y_train, y_valid = train_test_split(
    x_train_full, y_train_full, test_size=0.1, stratify=y_train_full.tolist()
)

for x in [x_train, x_test, x_valid]:
  x = x/_NORMALIZATION

keras.backend.clear_session()
```

```
model = keras.models.Sequential([
  keras.layers.Flatten(input_shape=[_SHAPE, _SHAPE]),
  keras.layers.Dense(300, activation='relu'),
  keras.layers.Dense(100, activation='relu'),
  keras.layers.Dense(10, activation='softmax')
])

model.compile(
    optimizer=tf.keras.optimizers.Adam(0.001),
    loss=tf.keras.losses.SparseCategoricalCrossentropy(),
    metrics=['accuracy']
)

history = model.fit(x_train,
          y_train,
          epochs=7,
          batch_size=_BATCH,
          shuffle=True,
          validation_data=(x_valid, y_valid))

print("\nEVALUATION")
model.evaluate(x_test, y_test)
```

Each epoch's Accuracy (if too many, just show the beginning, middle and end of the epoch progress):

```
Epoch 1/7
430/430 [==============================] - 4s 8ms/step - loss: 0.2827 - accuracy: 0.9170 - val_loss: 0.1310 - val_accuracy: 0.9624
Epoch 2/7
430/430 [==============================] - 4s 8ms/step - loss: 0.1035 - accuracy: 0.9689 - val_loss: 0.0849 - val_accuracy: 0.9740
Epoch 3/7
430/430 [==============================] - 3s 8ms/step - loss: 0.0668 - accuracy: 0.9803 - val_loss: 0.0798 - val_accuracy: 0.9764
Epoch 4/7
430/430 [==============================] - 3s 7ms/step - loss: 0.0461 - accuracy: 0.9860 - val_loss: 0.0757 - val_accuracy: 0.9772
Epoch 5/7
430/430 [==============================] - 3s 8ms/step - loss: 0.0331 - accuracy: 0.9899 - val_loss: 0.0674 - val_accuracy: 0.9798
Epoch 6/7
430/430 [==============================] - 3s 7ms/step - loss: 0.0264 - accuracy: 0.9920 - val_loss: 0.0685 - val_accuracy: 0.9794
Epoch 7/7
430/430 [==============================] - 5s 11ms/step - loss: 0.0213 - accuracy: 0.9933 - val_loss: 0.0696 - val_accuracy: 0.9816

EVALUATION
313/313 [==============================] - 1s 3ms/step - loss: 0.0719 - accuracy: 0.9802
```

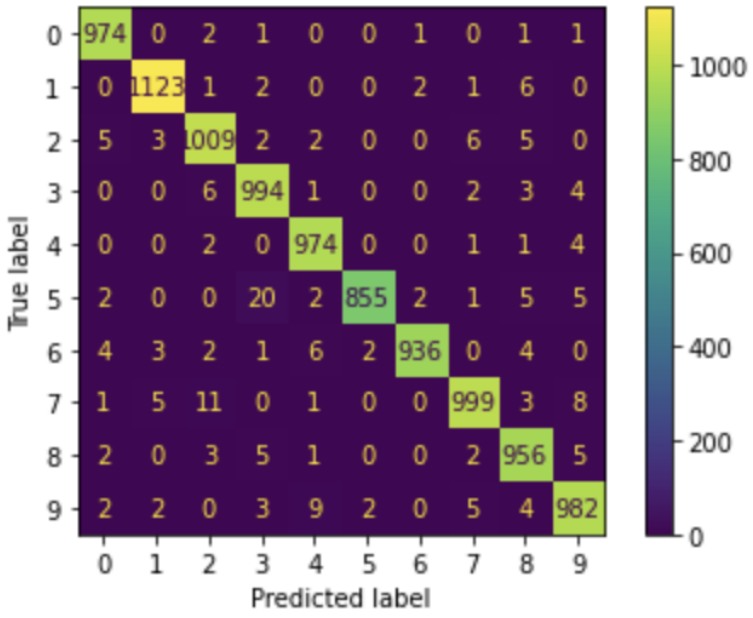
Final Accuracy:

Final Accuracy = 98.02%

Final Confusion Matrix (screenshot OK):

```
from sklearn.metrics import ConfusionMatrixDisplay

print("CONFUSION MATRIX")
y_pred = model.predict(x_test)
y_pred = np.argmax(y_pred,axis=1)
ConfusionMatrixDisplay.from_predictions(y_test, y_pred)
```

```
CONFUSION MATRIX
313/313 [==============================] - 2s 5ms/step
```



Analysis (100 - 200 words)

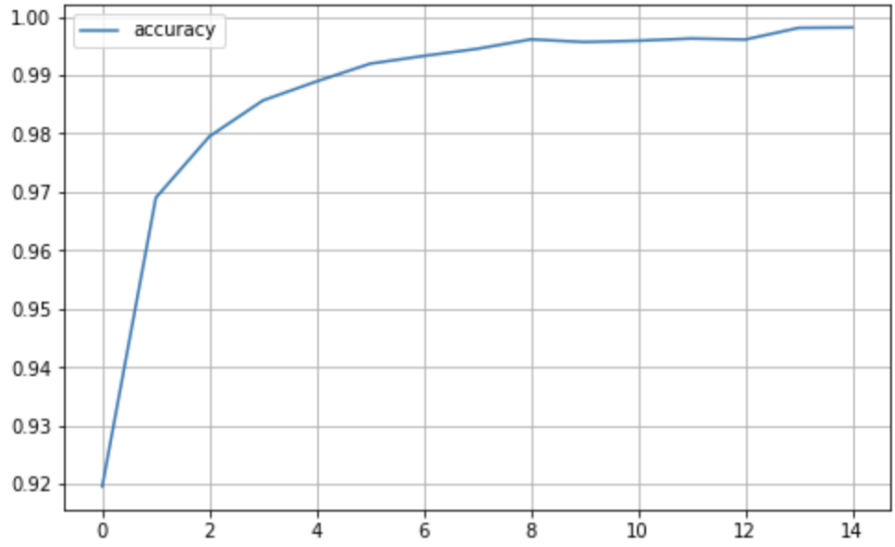
(Hint: Describe the process of your deep learning architecture. You may make some reference to outside sources since the current database is commonly used for deep learning practice.)

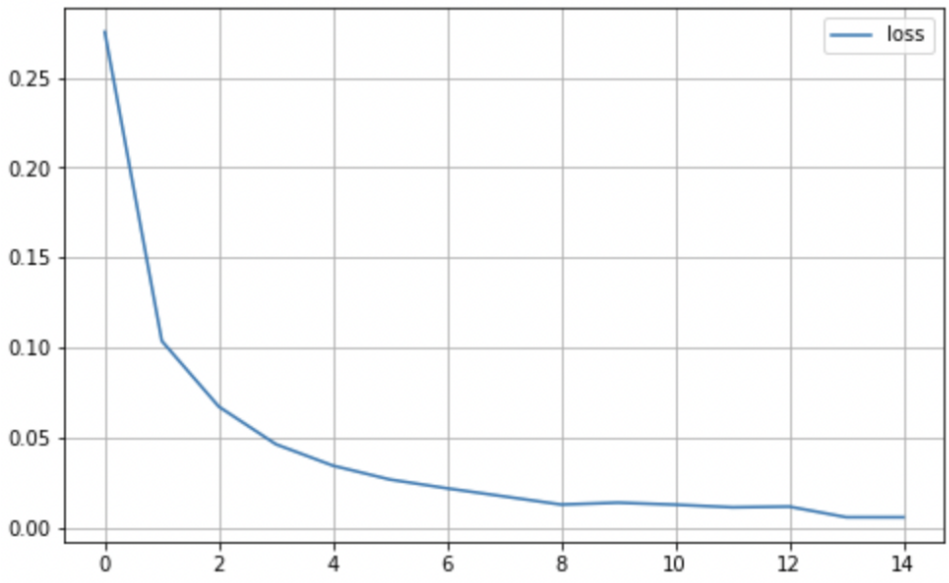
Our model is made of:

- A Flatten layer to flatten this input into a vector
- 2 hidden layers:
  - One dense layer with 300 nodes
  - One dense layer with 100 nodes
  - For both layers, we used the RELU function as it doesn't have the vanishing gradient problem and is fast to compute
- An output layer
  - That has 10 nodes as it is a classification problem and we have 10 classes (0 to 9)
  - As we want to obtain the probability distribution of each class, we used the softmax function as an activation function. Indeed, the softmax function normalizes its input.

We used the Adam optimizer as it is stable and the Categorical CrossEntropy as it is a classification problem.

We tried different epochs (between 1 and 15). We obtained the following accuracies and losses:





We decided that 7 was optimal as a number of epochs as after that number, the accuracy and the loss do not significantly improve. We observed the same behavior with the valid set.

We can label our model as a vanilla model as it doesn't have any convolutional layer or any batch normalization.
Even if our model did not reach the goal of 99% accuracy, it is quite close (98%), which shows that even vanilla models can obtain good results for this problem. However, this good performance might be the result of luck (good shuffle when fitting the model, specific split between validation and training sets…). When running this model 10 times, the average accuracy was 97.3%, with a maximum of 98.12%, which is still correct. It could be interesting to increase the number of repetitions.

Part B: Movie Recommendation

Prior Knowledge

Data Science Objective

Objective

MovieLens Dataset consists of
* 100,000 ratings (1-5) from 943 users on 1682 movies.
* Each user has rated at least 20 movies.
* Simple info for the movies (genres, title)
The data was collected through the MovieLens website (movielens.umn.edu).

The goal is to make a recommendation engine that could **recommend movies (those that are predicted to be having high ratings) to some of the users.**

Link

Rapidminer

MovieLens Dataset (Item Profile and Ratings)

Python

You can also use the following code to retrieve the dataset:

```
!curl https://dl.dropboxusercontent.com/s/khazvedpqrgrnet/ratings.csv?dl=0 -so ratings.csv
!curl https://dl.dropboxusercontent.com/s/vdpkvq8djxoprme/movie_profiles.csv?dl=0 -so movie_profiles.csv
```

Data Preparation & Exploration

User-user Rating Matrix 5/5

Build a user-user rating matrix from the ratings dataset. Give the screenshot of the first 20 rows and 20 columns.

Ratings of the first 20th movies from the first 20th users:

movie id	user id	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
0	1	5.000	3.000	4.000	3.000	3.000	5.000	4.000	1.000	5.000	3.000	2.000	5.000	5.000	5.000	5.000	5.000	3.000	4.000	5.000
1	2	4.000	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	2.000	NaN	NaN	4.000	4.000	NaN	NaN	NaN	NaN	3.000
2	3	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
3	4	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	4.000	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
4	5	4.000	3.000	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	4.000	NaN	NaN
5	6	4.000	NaN	NaN	NaN	NaN	NaN	2.000	4.000	4.000	NaN	NaN	4.000	2.000	5.000	3.000	NaN	NaN	NaN	4.000
6	7	NaN	NaN	NaN	5.000	NaN	NaN	5.000	5.000	5.000	4.000	3.000	5.000	NaN	NaN	NaN	NaN	NaN	NaN	NaN
7	8	NaN	NaN	NaN	NaN	NaN	NaN	3.000	NaN	NaN	NaN	3.000	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
8	9	NaN	NaN	NaN	NaN	NaN	5.000	4.000	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
9	10	4.000	NaN	NaN	4.000	NaN	NaN	4.000	NaN	4.000	NaN	4.000	5.000	3.000	NaN	NaN	4.000	NaN	NaN	NaN
10	11	NaN	NaN	NaN	NaN	NaN	NaN	NaN	4.000	5.000	NaN	2.000	2.000	NaN	NaN	5.000	NaN	NaN	NaN	NaN
11	12	NaN	NaN	NaN	5.000	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	5.000	NaN	NaN	NaN	NaN
12	13	3.000	3.000	NaN	5.000	1.000	NaN	2.000	4.000	3.000	NaN	1.000	5.000	5.000	4.000	NaN	NaN	1.000	NaN	NaN
13	14	NaN	NaN	NaN	NaN	NaN	NaN	5.000	NaN	4.000	NaN	NaN	5.000	4.000	3.000	4.000	NaN	NaN	3.000	5.000
14	15	1.000	NaN	NaN	NaN	NaN	NaN	1.000	NaN	4.000	NaN	NaN	NaN	1.000	4.000	4.000	NaN	NaN	1.000	NaN
15	16	5.000	NaN	NaN	5.000	NaN	NaN	5.000	5.000	5.000	NaN	5.000	5.000	NaN	NaN	5.000	NaN	NaN	NaN	NaN
16	17	4.000	NaN	NaN	NaN	NaN	NaN	4.000	NaN	3.000	NaN	NaN	NaN	3.000	NaN	NaN	NaN	NaN	NaN	NaN
17	18	5.000	NaN	NaN	3.000	NaN	5.000	NaN	5.000	5.000	NaN	NaN	5.000	5.000	5.000	4.000	NaN	NaN	NaN	3.000
18	19	NaN	NaN	NaN	4.000	NaN	NaN	NaN	5.000	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
19	20	3.000	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	2.000	NaN	NaN	NaN	4.000	NaN	NaN	NaN	NaN

User-User Similarity Matrix:

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
0	1.000	0.873	0.912	0.971	1.000	0.959	0.943	0.894	0.950	0.919	0.943	0.904	0.930	0.947	0.984	0.962	0.938	0.854	0.954	0.966
1	0.873	1.000	0.848	0.836	0.926	0.932	0.851	0.940	0.859	0.889	0.892	0.854	0.938	0.947	0.956	0.887	0.673	0.915	0.922	0.905
2	0.912	0.848	1.000	0.901	0.862	0.973	0.987	0.960	0.947	0.971	0.951	0.925	0.947	0.973	0.899	0.943	0.980	0.944	0.933	0.933
3	0.971	0.836	0.901	1.000	1.000	0.946	0.931	0.937	0.940	0.951	0.944	0.927	0.945	0.957	0.912	0.957	0.959	0.942	0.948	0.950
4	1.000	0.926	0.862	1.000	1.000	0.822	0.000	0.980	0.878	0.906	0.876	0.971	0.911	0.950	0.932	0.741	0.000	0.997	1.000	0.904
5	0.959	0.932	0.973	0.946	0.822	1.000	0.938	0.971	0.953	0.958	0.956	0.961	0.977	0.975	0.966	0.970	0.968	0.965	0.974	0.956
6	0.943	0.851	0.987	0.931	0.000	0.938	1.000	0.920	0.944	0.962	0.960	0.921	0.931	0.969	0.819	0.956	0.919	0.938	0.954	0.937
7	0.894	0.940	0.960	0.937	0.980	0.971	0.920	1.000	0.940	0.952	0.955	0.959	0.986	0.979	0.961	0.969	0.939	0.951	0.967	0.962
8	0.950	0.859	0.947	0.940	0.878	0.953	0.944	0.940	1.000	0.944	0.923	0.937	0.924	0.958	0.914	0.941	0.937	0.923	0.942	0.939
9	0.919	0.889	0.971	0.951	0.906	0.958	0.962	0.952	0.944	1.000	0.959	0.926	0.954	0.964	0.890	0.960	0.965	0.959	0.957	0.963
10	0.943	0.892	0.951	0.944	0.876	0.956	0.960	0.955	0.923	0.959	1.000	0.918	0.930	0.961	0.918	0.952	0.952	0.931	0.943	0.941
11	0.904	0.854	0.925	0.927	0.971	0.961	0.921	0.959	0.937	0.926	0.918	1.000	0.955	0.955	0.912	0.940	0.918	0.934	0.923	0.940
12	0.930	0.938	0.947	0.945	0.911	0.977	0.931	0.986	0.924	0.954	0.930	0.955	1.000	0.983	0.964	0.951	0.945	0.940	0.958	0.956
13	0.947	0.947	0.973	0.957	0.950	0.975	0.969	0.979	0.958	0.964	0.961	0.955	0.983	1.000	0.932	0.975	0.975	0.954	0.979	0.966
14	0.984	0.956	0.899	0.912	0.932	0.966	0.819	0.961	0.914	0.890	0.918	0.912	0.964	0.932	1.000	0.918	0.897	0.928	0.920	0.946
15	0.962	0.887	0.943	0.957	0.741	0.970	0.956	0.969	0.941	0.960	0.952	0.940	0.951	0.975	0.918	1.000	0.950	0.947	0.960	0.955
16	0.938	0.673	0.980	0.959	0.000	0.968	0.919	0.939	0.937	0.965	0.952	0.918	0.945	0.975	0.897	0.950	1.000	0.927	0.932	0.936
17	0.854	0.915	0.944	0.942	0.997	0.965	0.938	0.951	0.923	0.959	0.931	0.934	0.940	0.954	0.928	0.947	0.927	1.000	0.958	0.940
18	0.954	0.922	0.933	0.948	1.000	0.974	0.954	0.967	0.942	0.957	0.943	0.923	0.958	0.979	0.920	0.960	0.932	0.958	1.000	0.958
19	0.966	0.905	0.933	0.950	0.904	0.956	0.937	0.962	0.939	0.963	0.941	0.940	0.956	0.966	0.946	0.955	0.936	0.940	0.958	1.000

Add Markdown Cell

User-feature Rating Matrix 5/5

Build a user-feature rating matrix from the ratings and item profile dataset. Give the screenshot of the first 20 rows and 10 columns.

user id	movie id	unknown	Action	Adventure	Animation	Childrens	Comedy	Crime	Documentary	Drama
1	136.500000	0.002941	0.183824	0.090441	0.029412	0.040441	0.232353	0.063235	0.017647	0.308824
2	249.500000	0.000000	0.122581	0.041935	0.012903	0.038710	0.196774	0.109677	0.000000	0.432258
3	318.814815	0.000000	0.144444	0.051852	0.000000	0.000000	0.114815	0.111111	0.018519	0.237037
4	291.041667	0.000000	0.258333	0.116667	0.000000	0.000000	0.166667	0.158333	0.041667	0.225000
5	291.291429	0.004571	0.201143	0.122286	0.060571	0.081143	0.281143	0.040000	0.000000	0.082286
6	312.018957	0.000000	0.079621	0.072038	0.032227	0.057820	0.218957	0.049289	0.003791	0.367773
7	392.779156	0.000000	0.183127	0.115136	0.028784	0.072457	0.167246	0.063524	0.010422	0.305211
8	301.322034	0.000000	0.538983	0.220339	0.000000	0.013559	0.081356	0.125424	0.000000	0.244068
9	370.818182	0.000000	0.181818	0.136364	0.000000	0.000000	0.245455	0.000000	0.000000	0.400000
10	361.201087	0.000000	0.116304	0.066304	0.031522	0.036957	0.201087	0.078261	0.013043	0.376087
11	392.806630	0.000000	0.111602	0.055249	0.000000	0.022099	0.279558	0.051934	0.000000	0.340331
12	258.725490	0.000000	0.239216	0.121569	0.015686	0.047059	0.235294	0.031373	0.000000	0.427451
13	486.605346	0.000000	0.149686	0.071069	0.018553	0.047484	0.179560	0.044340	0.014151	0.242453
14	346.285714	0.000000	0.140816	0.132653	0.051020	0.048980	0.287755	0.055102	0.014286	0.324490
15	444.740385	0.000000	0.113462	0.069231	0.003846	0.025000	0.144231	0.028846	0.000000	0.294231
16	338.114286	0.000000	0.197143	0.080000	0.055714	0.074286	0.222857	0.128571	0.000000	0.350000
17	264.571429	0.000000	0.064286	0.100000	0.028571	0.064286	0.200000	0.035714	0.000000	0.364286
18	403.902527	0.000000	0.068592	0.049097	0.036823	0.054152	0.272924	0.036823	0.008664	0.415884
19	320.750000	0.000000	0.170000	0.090000	0.000000	0.050000	0.450000	0.000000	0.000000	0.380000
20	316.895833	0.000000	0.279167	0.191667	0.050000	0.095833	0.145833	0.020833	0.000000	0.195833

Building Recommendation Engine

Collaborative-based Method 5/5

Make a recommendation engine using collaborative-based method to predict the user rating. Attach the screenshots of your code/architecture & parameters.

```
_PATH = "10_11_Data"
_MOVIE_ID = 'movie id'
_USER_ID = 'user id'
_RATING = 'rating'
_TITLE = ' movie title '
_MAX_RATING = 5
```

```
_movie_pivot = _ratings.pivot(index=_USER_ID, columns=_MOVIE_ID, values=_RATING).reset_index()
_movie_depivot = _movie_pivot.melt(id_vars=_USER_ID,value_name=_RATING)
_movie_train = _ratings
_movie_test = _movie_depivot[_movie_depivot[_RATING].isna()]
```

```
# Setting the dataset in surprise format (user_id, item_id, rating)
_reader = Reader(rating_scale=(1, 5))
_trainset_model_surprise = Dataset.load_from_df(_movie_train, _reader).build_full_trainset()
_testset_model_surprise = Dataset.load_from_df(_movie_test, _reader).build_full_trainset().build_testset()

# User-based kNN Regression
_model = KNNBasic(k=3,
                  sim_options = {"name": "cosine", "user_based": True} # similarity
                  )
predictions = _model.fit(_trainset_model_surprise).test(_testset_model_surprise)
```

✓ 1m 12.8s

```
Computing the cosine similarity matrix...
Done computing similarity matrix.
```

Content-based Method 5/5

Make a recommendation engine using content-based method to predict the user rating. Attach the screenshots of your code/architecture & parameters.

```
_PATH = "10_11_Data"
_MOVIE_ID = 'movie id'
_USER_ID = 'user id'
_RATING = 'rating'
_TITLE = ' movie title '
_MAX_RATING = 5
```

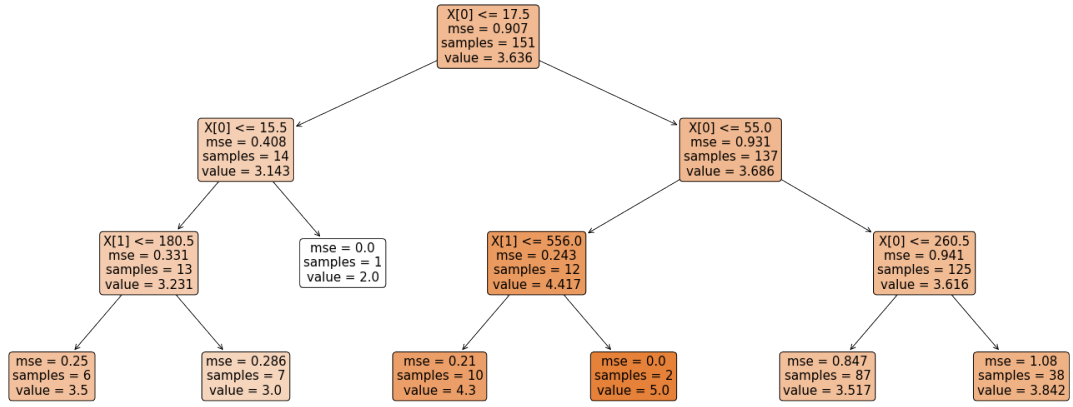
```
_user_feature = _ratings.merge(_movies, on=_MOVIE_ID, how='left')
_user_feature.iloc[:,8:] = _user_feature.iloc[:,8:].mul(_user_feature.iloc[:,2]/_MAX_RATING, axis='index')
_DROP_COL = [_RATING, _TITLE]
_user_feature = _user_feature.drop(_DROP_COL, axis=1).groupby(_USER_ID).mean()

_sim_matrix = cosine_similarity(_user_feature, _movies.drop(_TITLE, axis=1))

_sim_matrix = pd.DataFrame(_sim_matrix,
                           index=_user_feature.index,
                           columns=_movies.drop(_TITLE, axis=1).index).reset_index()

_sim_matrix.melt(id_vars=_USER_ID, value_name='similarity')\
    .sort_values(by=[_USER_ID,'similarity'], ascending=False)
```

Decision Tree Method 5/5

Build a decision tree for predicting the user rating (you can use any user ID as a sample). Attach the screenshots of your code/architecture & parameters.

```
_UID = 23

user_labels = _movie_pivot.T
user_labels = user_labels.rename(columns=user_labels.iloc[0]).iloc[1:]
one_user_feature = pd.concat([_ratings, user_labels[_UID]],axis=1)

train_set = one_user_feature.dropna()
test_set = one_user_feature[one_user_feature[_UID].isna()]

regular_att_train = train_set.drop(_UID, axis=1).values
label_att_train = train_set[_UID].values

regular_att_test = test_set.drop(_UID, axis=1).values
label_att_test = test_set[_UID].values

regressor = DecisionTreeRegressor(
    max_depth=3,
)
regressor.fit(regular_att_train, label_att_train)
prediction = regressor.predict(regular_att_test)
prediction
```

Decision Tree:



Deployment

• Collaborative-based Method 5/5

Show the **5 recommended movies** for the given users using the model you have made. Write down also the **title of those movies** (not just the movie id). You can also just use software/code screenshots instead of typing it directly.

User ID: 26

```
  User ID: 26
                                        movie_title  movie_id
                                Braveheart (1995)         22
                            North by Northwest (1959)        480
                                Prefontaine (1997)       1189
Marlene Dietrich: Shadow and Light (1996)        1201
                            Santa with Muscles (1996)       1500
```

User ID: 50

```
  User ID: 50
                                movie_title  movie_id
                            Shadowlands (1993)       736
                Great Day in Harlem, A (1994)       814
City of Lost Children, The (1995)       919
               They Made Me a Criminal (1939)      1122
                            Prefontaine (1997)      1189
```

User ID: 101

```
  User ID: 101
                                    movie_title  movie_id
One Flew Over the Cuckoos Nest (1975)       357
                            Postman, The (1997)       898
                            Prefontaine (1997)      1189
                        Santa with Muscles (1996)      1500
                            Aiqing wansui (1994)      1536
```

- **Content-based Method 5/5**

Show the **5 recommended movies** for the given users using the model you have made. Write down also the **title of those movies** (not just the movie id). You can also just use software/code screenshots instead of typing it directly.

User ID: 26

User ID: 26

movie_title	movie_id
Bird of Prey (1996)	1364
Mighty, The (1998)	1432
Fausto (1993)	1490
Secret Adventures of Tom Thumb, The (1993)	1555
Aparajito (1956)	1558

User ID: 50

User ID: 50

movie_title	movie_id
Up in Smoke (1978)	1118
Safe (1995)	1131
Ghosts of Mississippi (1996)	1136
Stalker (1979)	1159
Women, The (1939)	1172

User ID: 101

User ID: 101

movie_title	movie_id
Man in the Iron Mask, The (1998)	1483
Trial by Jury (1994)	1522
Glass Shield, The (1994)	1551
Butcher Boy, The (1998)	1645
Little City (1998)	1656

Decision Tree Method 3/5

By referencing the nodes in the decision tree, describe what **kind of genres** the following users like and dislike. (The number of likes/dislikes are decided based on the tree you build)

User ID: 26 (Genres)

Decision Tree:

Likes the genre:

- Comedy
- Romance

Dislikes the genre:

- Horror
- Thriller

Recommended Movies:

```
   Likes:
             movie_title  movie_id
     Chasing Amy (1997)       268
         Tin Cup (1996)       284
 Fierce Creatures (1997)       290
        Rosewood (1997)       292


   Dislikes:
                movie_title  movie_id
     Time to Kill, A (1996)       282
             Tin Cup (1996)       284
 English Patient, The (1996)       286
        Marvins Room (1996)       287
```

User ID: 50 (Genres)

Decision Tree:

Likes the genre:

- Comedy
- Childrens

Dislikes the genre:

- Action
- Sci-fi

Recommended Movies:

Likes:

movie_title	movie_id
Stargate (1994)	62
101 Dalmatians (1996)	225
Ace Ventura: When Nature Calls (1995)	364
Flubber (1997)	892
Gridlockd (1997)	1245

Dislikes:

movie_title	movie_id
Dances with Wolves (1990)	97
Godfather, The (1972)	127
Bound (1996)	129
Alien (1979)	183
Bridge on the River Kwai, The (1957)	199

User ID: 101 (Genres)

Decision Tree:

Likes the genre:

- Comedy
- Romance

Dislikes the genre:

- Crime
- Thriller

Recommended Movies:

```
  Likes:
         movie_title  movie_id
     Tin Cup (1996)       284
 Rear Window (1954)       603
        Dave (1993)       732
  Booty Call (1997)       948
Cool Runnings (1993)      1035

  Dislikes:
                        movie_title  movie_id
      Much Ado About Nothing (1993)        83
               Reservoir Dogs (1992)       156
     Godfather: Part II, The (1974)       187
                    Amadeus (1984)       191
One Flew Over the Cuckoos Nest (1975)       357
```

Final Analysis (100 - 200 words) 10/10

(Hint: Which recommendation engine is the best used for this case? What's your opinion on the advantages/disadvantages of each method?)

The decision tree model is a simple, interpretable method that can be used to recommend movies based on a set of features or attributes. However, decision tree models can struggle with handling large amounts of data and may not be able to capture complex relationships between movies. We set the max depth to 3 for readability reasons.

The collaborative-based method uses the past viewing history of all users to recommend movies to a specific user. This method is effective at recommending new, unseen movies to users, as it takes into account the viewing history of many users. However, it may struggle to recommend movies to users who have very unique or niche tastes, as it relies on the viewing history of a large number of users. We can see in our case that the same movie Prefontaine (1997) is recommended for the three users.

The content-based method is effective at recommending movies to users based on their past viewing history, as it identifies patterns in the content of movies that a user has previously enjoyed. It seems to be most suited to this situation.

However, a Hybrid method of Content-based and Collaborative-based might be more accurate as it will take into account both the content of the movie and the viewing history of other users. It could