Lab 6

CS 216, Everything Data, Spring 2020

In this lab exercise, you will participate in a prediction challenge. You should work in a group of 3-5 students of your choice; you will only need to submit once as a group (see submission instructions below). Remember that labs are graded for sincere effort. In addition, the top three teams for today's challenge will receive bonus points for today's lab.

Part 1: The Challenge

Until know, you have primarily experienced prediction with very structured data that was already formatted in a convenient way for applying machine learning algorithms. In practice, however, we often get a variety of data from different sources, and it is not immediately clear what information is useful for the prediction task, or how to use it. In this prediction challenge, we will work with the MovieLens dataset from HW 4. Recall that it contained about 100,000 ratings (on a 1 to 5 scale) of 1,682 movies by 943 users. The information we are given is contained in three tables or dataframes:

- `df_users` (the `u.user` file) contains demographic information about each of the 943 users.
- `df_movies` (the `u.item` file) contains basic identifying and genre information for each of the 1,682 movies.
- `df_train_ratings` (the `train_ratings.csv` file) contains 70,050 ratings: each row identifies a particular user who rated a particular movie with a particular score.

Our goal is to use this information to predict the 29,950 ratings in the `test_ratings.csv` file (which will be provided 20 minutes before the end of lab). Just as in `df_train_ratings`, for each rating we want to predict, we will be told the `user_id` and `movie_id` and asked to predict how that particular user would rate that particular movie. You may assume that all users and movies in the test set also appear in `df_users` and `df_movies`.

We will measure the error of our predicted ratings as the root-mean-square error (see https://en.wikipedia.org/wiki/Root-mean-square_deviation (https://en.wikipedia.org/wiki/Root-mean-square_deviation)), and maintain a leaderboard of the teams with the lowest error so far. The three teams with the lowest error predictions on the leaderboard with 5 minutes before the end of lab will receive bonus points for today's lab.

You may use anything in standard Python, Numpy, Pandas, scikit-learn, and your own previous labs and homeworks to help you, but you may not use code from outside of these sources. Code showing how to measure error is provided below, good luck!

In [1]:
```
# Feel free to import whatever else
# you want from sklearn
from sklearn import metrics
import numpy as np
import pandas as pd
```

In [ ]:
```
genres = ['unknown', 'action', 'adventure', 'animation', 'children',
          'comedy', 'crime', 'documentary', 'drama', 'fantasy',
          'film_noir', 'horror', 'musical', 'mystery', 'romance',
          'sci_fi','thriller', 'war', 'western']
total = len(df_users) #943 users
#our goal is for each user, predict how the particular user would rate that pa
rticular movie given a movie and the user id
#for each user in the dfusers
#create table of average score for each genre of movie based off previously ra
ted movies
#Do this by filtering the user id in df_movies
#print(df_users['user_id'])
for user in range(len(df_users)):
    ratinglist = []
    for genre in genres:
        currating = []
        for rating in range(len(df_movies)):
            if df_train_ratings['user_id'][user] == user:
                if df_train_ratings['genre']
                currating.append(df_train_ratings['rating'][user])
#for genre in genres:
#    total = 0
#    count = 0
#    for x in range(len(df_movies['movie_id'])):
#        if df_movies[genre][x] == 1:
#            for movie in range(len(df_train_ratings['movie_id'])):
#                if df_train_ratings['movie_id'][movie] == df_movies['movie_i
d'][x]:
#                    count += 1
#                    total += df_train_ratings['rating'][movie]
#    print(genre, total/count)
```

In [2]:
```
df_users = pd.read_table('u.user', sep='|', names = ['user_id', 'age', 'sex',
'occupation', 'zip'])
print(df_users.shape)
df_users.head()
```

```
(943, 5)
```

Out[2]:

	user_id	age	sex	occupation	zip
0	1	24	M	technician	85711
1	2	53	F	other	94043
2	3	23	M	writer	32067
3	4	24	M	technician	43537
4	5	33	F	other	15213

In [3]:
```
df_movies = pd.read_table('u.item', sep='|', encoding='latin',\
                          names = ['movie_id', 'movie_title', 'release_date',
'video_release_date',\
                                   'imdb_url', 'unknown', 'action', 'adventur
e', 'animation',\
                                   'children', 'comedy', 'crime', 'documentar
y', 'drama', 'fantasy',\
                                   'film_noir', 'horror', 'musical', 'myster
y', 'romance', 'sci_fi',\
                                   'thriller', 'war', 'western'])
print(df_movies.shape)
df_movies.head()
```

```
(1682, 24)
```

Out[3]:

	movie_id	movie_title	release_date	video_release_date	imdb_url	unknown	act
0	1	Toy Story (1995)	01-Jan-1995	NaN	http://us.imdb.com/M/title-exact?Toy%20Story%2...	0	
1	2	GoldenEye (1995)	01-Jan-1995	NaN	http://us.imdb.com/M/title-exact?GoldenEye%20(...	0	
2	3	Four Rooms (1995)	01-Jan-1995	NaN	http://us.imdb.com/M/title-exact?Four%20Rooms%...	0	
3	4	Get Shorty (1995)	01-Jan-1995	NaN	http://us.imdb.com/M/title-exact?Get%20Shorty%...	0	
4	5	Copycat (1995)	01-Jan-1995	NaN	http://us.imdb.com/M/title-exact?Copycat%20(1995)	0	

5 rows × 24 columns

In [4]:
```
df_train_ratings = pd.read_csv('train_ratings.csv')
print(df_train_ratings.shape)
df_train_ratings.head()
```

```
(70050, 4)
```

Out[4]:

	user_id	movie_id	rating	timestamp
0	196	242	3	881250949
1	22	377	1	878887116
2	244	51	2	880606923
3	166	346	1	886397596
4	298	474	4	884182806

In [10]:
```
y_train = df_train_ratings['rating'].values
```

Part 2: Feature Engineering, Modeling, and Training

Until 20 minutes left in lab, you will only have access to the data above to build and test your model. Remember, the test dataset will be a table formatted exactly like `df_train_ratings` except with 29,950 rows instead of 70,050. Below, we go through a very simple example of making a "prediction" (we just randomly guess a number between 1 and 5 for each of the 70,050 training ratings) and measuring the root-mean-squre error of that prediction with respect to `y_train`, the actual ratings for the training data.

In [9]:
```
np.random.seed(1)
random_guessing = np.random.randint(low=1, high=6, size=y_train.size)
```

In [13]:
```
print("root-mean-square error of random guessing: ",
      metrics.mean_squared_error(y_train, random_guessing, squared=False))
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-13-36002ae8d9a4> in <module>
      1 print("root-mean-square error of random guessing: ",
----> 2       metrics.mean_squared_error(y_train, random_guessing, squared=Fa
lse))

TypeError: mean_squared_error() got an unexpected keyword argument 'squared'
```

Not surprisingly, random guessing is not incredibly accurate. To measure the root-mean-square error using scikit-learn, we simply use the metrics.mean_squared_error function (recall HW 6) but pass the parameter `squared=False`. This means that we take the square root of the mean-square error, which makes the error on the same scale as te data. So the error of about 1.89 above means that on average, our random guesses were about 1.89 off from the true rating. Try to build a model that does better by using the data in the training set. Be careful though not to overfit; the final measurement of error will be on the test data set you haven't seen yet. Feel free to make fractional predictions (i.e., you can predict a rating of 3.64 for a movie if you like).

In [12]:
```
prediction = np.random.randint(low=1, high=6, size=y_train.size)
print("root-mean-square error of random guessing: ",
      metrics.mean_squared_error(y_train, random_guessing))
```

```
root-mean-square error of random guessing:  3.5627266238401143
```

Part 3: Testing

Note You can cannot begin this part until the test data is released 20 minutes before the end of lab. By that time, you should hopefully have built a predictive model based on the training ratings in Part 2. Now it's time to test that model on the held out test data. With 5 minutes left in lab, the team with the lowest root-mean-square error on the test data will be the winner of the challenge.

In [17]:
```
df_test_ratings = pd.read_csv('test_ratings.csv')
print(df_test_ratings.shape)
df_test_ratings.head()
```

```
---------------------------------------------------------------------------
FileNotFoundError                         Traceback (most recent call last)
<ipython-input-17-670c861cc021> in <module>
----> 1 df_test_ratings = pd.read_csv('test_ratings.csv')
      2 print(df_test_ratings.shape)
      3 df_test_ratings.head()

~\Anaconda3\lib\site-packages\pandas\io\parsers.py in parser_f(filepath_or_buffer, sep, delimiter, header, names, index_col, usecols, squeeze, prefix, mangle_dupe_cols, dtype, engine, converters, true_values, false_values, skipinitialspace, skiprows, skipfooter, nrows, na_values, keep_default_na, na_filter, verbose, skip_blank_lines, parse_dates, infer_datetime_format, keep_date_col, date_parser, dayfirst, cache_dates, iterator, chunksize, compression, thousands, decimal, lineterminator, quotechar, quoting, doublequote, escapechar, comment, encoding, dialect, error_bad_lines, warn_bad_lines, delim_whitespace, low_memory, memory_map, float_precision)
    683         )
    684
--> 685         return _read(filepath_or_buffer, kwds)
    686
    687     parser_f.__name__ = name

~\Anaconda3\lib\site-packages\pandas\io\parsers.py in _read(filepath_or_buffer, kwds)
    455
    456     # Create the parser.
--> 457     parser = TextFileReader(fp_or_buf, **kwds)
    458
    459     if chunksize or iterator:

~\Anaconda3\lib\site-packages\pandas\io\parsers.py in __init__(self, f, engine, **kwds)
    893             self.options["has_index_names"] = kwds["has_index_names"]
    894
--> 895         self._make_engine(self.engine)
    896
    897     def close(self):

~\Anaconda3\lib\site-packages\pandas\io\parsers.py in _make_engine(self, engine)
   1133     def _make_engine(self, engine="c"):
   1134         if engine == "c":
-> 1135             self._engine = CParserWrapper(self.f, **self.options)
   1136         else:
   1137             if engine == "python":

~\Anaconda3\lib\site-packages\pandas\io\parsers.py in __init__(self, src, **kwds)
   1915         kwds["usecols"] = self.usecols
   1916
-> 1917         self._reader = parsers.TextReader(src, **kwds)
   1918         self.unnamed_cols = self._reader.unnamed_cols
   1919

pandas\_libs\parsers.pyx in pandas._libs.parsers.TextReader.__cinit__()

pandas\_libs\parsers.pyx in pandas._libs.parsers.TextReader._setup_parser_sou
```

```
rce()

FileNotFoundError: [Errno 2] File b'test_ratings.csv' does not exist: b'test_
ratings.csv'
```

In [15]:
```
y_test = df_test_ratings['rating'].values
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-15-38562b9480e4> in <module>
----> 1 y_test = df_test_ratings['rating'].values

NameError: name 'df_test_ratings' is not defined
```

In [16]:
```
# Write code for Part 3 here.
# Add additional code blocks as necessary
# to keep your work organized.
# Refer to the sample code provided in Part 2
# to see how to measure your test error.
```

Submitting Lab 6

1. Double check that you have written all of your answers along with your supporting work in this notebook. Make sure you save the complete notebook.
2. Double check that your entire notebook runs correctly and generates the expected output. To do so, you can simply select Kernel -> Restart and Run All.
3. You will download two versions of your notebook to submit, a .pdf and a .py. To create a PDF, we reccomend that you select File --> Download as --> HTML (.html). Open the downloaded .html file; it should open in your web broser. Double check that it looks like your notebook, then print a .pdf using your web browser (you should be able to select to print to a pdf on most major web browsers and operating systems). Check your .pdf for readability: If some long cells are being cut off, go back to your notebook and split them into multiple smaller cells. Also, make sure that it is a reasonable length; print statements which are truncated inside of the notebook may come to many pages in the pdf. To get the .py file from your notebook, simply select File -> Download as -> Python (.py).
4. Upload the .pdf to gradescope under lab 6 report and the .py to gradescope under lab 6 code. Only submit once per group, but be sure to add your partner using the group feature on gradescope (https://www.gradescope.com/help#help-center-item-student-group-members).