# a6

March 24, 2018

## 1 CPSC 340 Assignment 6

```
In [7]: import os
        import numpy as np
        import pandas as pd
        import matplotlib.pyplot as plt
        %matplotlib inline

        from scipy.sparse import csr_matrix as sparse_matrix

        from sklearn.neighbors import NearestNeighbors
        from sklearn.decomposition import TruncatedSVD
```

### 1.1 Instructions

rubric={mechanics:5}

The above points are allocated for following the homework submission instructions.

### 1.2 Exercise 1: Finding similar items

For this question we'll be using the Amazon product data set. The author of the data set has asked for the following citations:

> Ups and downs: Modeling the visual evolution of fashion trends with one-class collaborative filtering. R. He, J. McAuley. WWW, 2016.
>
> Image-based recommendations on styles and substitutes. J. McAuley, C. Targett, J. Shi, A. van den Hengel. SIGIR, 2015.

We will focus on the "Patio, Lawn, and Garden" section. Download the ratings and place the file in the `data` directory with the original filename. Once you do that, the code below should load the data:

```
In [2]: filename = "ratings_Patio_Lawn_and_Garden.csv"

        with open(os.path.join("..", "data", filename), "rb") as f:
            ratings = pd.read_csv(f,names=("user","item","rating","timestamp"))
        ratings.head()
```

Out[2]:

| | user | item | rating | timestamp |
|---|---|---|---|---|
| 0 | A2VNYWOPJ13AFP | 0981850006 | 5.0 | 1259798400 |
| 1 | A2ODWVV8HML3AW | 0981850006 | 5.0 | 1371081600 |
| 2 | A3RVP3YBYYOPRH | 0981850006 | 5.0 | 1257984000 |
| 3 | A28XY55TP3Q90O | 0981850006 | 5.0 | 1314144000 |
| 4 | A3VZW1BGUQOOV3 | 0981850006 | 5.0 | 1308268800 |

We'd also like to construct the user-product matrix `X`. Let's see how big it would be:

```
In [3]: def get_stats(ratings, item_key="item", user_key="user"):
            print("Number of ratings:", len(ratings))
            print("The average rating:", np.mean(ratings["rating"]))

            d = len(set(ratings[item_key]))
            n = len(set(ratings[user_key]))
            print("Number of users:", n)
            print("Number of items:", d)
            print("Fraction nonzero:", len(ratings)/(n*d))
            print("Size of full X matrix: %.2f GB" % ((n*d)*8/1e9))

            return n,d

        n,d = get_stats(ratings)
```

```
Number of ratings: 993490
The average rating: 4.006400668350965
Number of users: 714791
Number of items: 105984
Fraction nonzero: 1.3114269915944552e-05
Size of full X matrix: 606.05 GB
```

600 GB! That is way too big. We don't want to create that matrix. On the other hand, we see that we only have about 1 million ratings, which would be around 8 MB ($10^6$ numbers $\times$ at 8 bytes per double precision floating point number). Much more manageable.

```
In [4]: def create_X(ratings,n,d,user_key="user",item_key="item"):
            user_mapper = dict(zip(np.unique(ratings[user_key]), list(range(n))))
            item_mapper = dict(zip(np.unique(ratings[item_key]), list(range(d))))

            user_inverse_mapper = dict(zip(list(range(n)), np.unique(ratings[user_key])))
            item_inverse_mapper = dict(zip(list(range(d)), np.unique(ratings[item_key])))

            user_ind = [user_mapper[i] for i in ratings[user_key]]
            item_ind = [item_mapper[i] for i in ratings[item_key]]

            X = sparse_matrix((ratings["rating"], (user_ind, item_ind)), shape=(n,d))

            return X, user_mapper, item_mapper, user_inverse_mapper, item_inverse_mapper, user
```

```
        X, user_mapper, item_mapper, user_inverse_mapper, item_inverse_mapper, user_ind, item_
```

In [5]:
```
# sanity check
print(X.shape) # should be number of users by number of items
print(X.nnz)   # number of nonzero elements -- should equal number of ratings
```

```
(714791, 105984)
993490
```

In [6]:
```
X.data.nbytes
```

Out[6]: 7947920

(Above: verifying our estimate of 8 MB to store sparse `X`)

#### 1.2.1 1.1

rubric={reasoning:2}

Find the following items:

1. the item with the most reviews
2. the item with the most total stars
3. the item with the highest average stars

Then, find the names of these items by looking them up with the url https://www.amazon.com/dp/ITEM_ID, where `ITEM_ID` is the id of the item.

In [11]:
```
url_amazon = "https://www.amazon.com/dp/%s"

# example:
print(url_amazon % 'B00CFM0P7Y')
```

```
https://www.amazon.com/dp/B00CFM0P7Y
```

In [1]:
```
### YOUR CODE HERE ###
```

#### 1.2.2 1.2

rubric={reasoning:2}

Make the following histograms

1. The number of ratings per user
2. The number of ratings per item
3. The ratings themselves

For the first two, use

```
plt.yscale('log', nonposy='clip')
```

to put the histograms on a log-scale.

In [2]:
```
### YOUR CODE HERE ###
```

#### 1.2.3 1.3

rubric={reasoning:1}

Use scikit-learn's NearestNeighbors object (which uses Euclidean distance by default) to find the 5 items most similar to Brass Grill Brush 18 Inch Heavy Duty and Extra Strong, Solid Oak Handle.

The code block below grabs the row of `X` associated with the grill brush. The mappers take care of going back and forther between the IDs (like `B00CFM0P7Y`) and the indices of the sparse array (0,1,2,...).

Note: keep in mind that `NearestNeighbors` is for taking neighbors across rows, but here we're working across columns.

```
In [20]: grill_brush = "B00CFM0P7Y"
         grill_brush_ind = item_mapper[grill_brush]
         grill_brush_vec = X[:,grill_brush_ind]

         print(url_amazon % grill_brush)
```

```
https://www.amazon.com/dp/B00CFM0P7Y
```

```
In [3]: ### YOUR CODE HERE ###
```

#### 1.2.4 1.4

rubric={reasoning:1}

Using cosine similarity instead of Euclidean distance in `NearestNeighbors`, find the 5 products most similar to `B00CFM0P7Y`.

```
In [4]: ### YOUR CODE HERE ###
```

#### 1.2.5 1.5

rubric={reasoning:2}

For each of the two metrics, compute the compute the total popularity (total stars) of each of the 5 items and report it. Do the results make sense given what we discussed in class about Euclidean distance vs. cosine similarity?

```
In [5]: ### YOUR CODE HERE ###
```

#### 1.2.6 1.6

rubric={reasoning:3}

PCA gives us an approximation $X \approx ZW$ where the rows of $Z$ contain a length-$k$ latent feature vectors for each user and the columns of $W$ contain a length-$k$ latent feature vectors for each item.

Another strategy for finding similar items is to run PCA and then search for nearest neighbours with Euclidean distance in the latent feature space, which is hopefully more meaningful than the original "user rating space". In other words, we run nearest neighbors on the columns of $W$. Using $k = 10$ and scikit-learn's TruncatedSVD to perform the dimensionality reduction, find the 5 nearest

neighbours to the grill brush using this method. You can access $W$ via the `components_` field of the `TruncatedSVD` object, after you fit it to the data.

Briefly comment on your results.

Implementation note: when you call on `NearestNeighbors.kneighbors`, it expects the input to be a 2D array. There's some weirdness here because `X` is a scipy sparse matrix but your `W` will be a dense matrix, and they behave differently in subtle ways. If you get an error like "Expected 2D array, got 1D array instead" then this is your problem: a column of `W` is technically a 1D array but a column of `X` has dimension $1 \times n$, which is technically a 2D array. You can take a 1D numpy array and add an extra first dimension to it with `array[None]`.

Conceptual note 1: We are using the "truncated" rather than full SVD since a full SVD would involve dense $d \times d$ matrices, which we've already established are too big to deal with. And then we'd only use the first $k$ rows of it anyway. So a full SVD would be both impossible and pointless.

Conceptual note 2: as discussed in class, there is a problem here, which is that we're not ignoring the missing entries. You could get around this by optimizing the PCA objective with gradient descent, say using `findMin` from previous assignments. But we're just going to ignore that for now, as the assignment seems long enough as it is (or at least it's hard for me to judge how long it will take because it's new).

```
In [6]: ### YOUR CODE HERE ###
```

### 1.3 Exercise 2: putting it all together in a CPSC 340 "mini-project"

rubric={reasoning:25}

In this open-ended mini-project, you'll explore the UCI default of credit card clients data set. There are 30,000 examples and 24 features, and the goal is to estimate whether a person will default (fail to pay) their credit card bills; this column is labeled "default payment next month" in the data. The rest of the columns can be used as features.

**Your tasks:**

1. Download the data set and load it in. Since the data comes as an MS Excel file, I suggest using `pandas.read_excel` to read it in. See Lecture 2 for an example of using pandas.
2. Perform exploratory data analysis on the data set. Include at least two summary statistics and two visualizations that you find useful, and accompany each one with a sentence explaining it.
3. Randomly split the data into train, validation, test sets. The validation set will be used for your experiments. The test set should be saved until the end, to make sure you didn't overfit on the validation set. You are welcome to use scikit-learn's train_test_split, which takes care of both shuffling and splitting.
4. Try scikit-learn's DummyClassifier as a baseline model.
5. Try logistic regression as a first real attempt. Make a plot of train/validation error vs. regularization strength. What's the lowest validation error you can get?
6. Explore the features, which are described on the UCI site. Explore preprocessing the features, in terms of transforming non-numerical variables, feature scaling, change of basis, etc. Did this improve your results?
7. Try 3 other models aside from logistic regression, at least one of which is a neural network. Can you beat logistic regression? (For the neural net(s), the simplest choice would probably be to use scikit-learn's MLPClassifier, but you are welcome to use any software you wish. )

8. Make some attempts to optimize hyperparameters for the models you've tried and summarize your results. In at least one case you should be optimizing multiple hyperparameters for a single model. I won't make it a strict requirement, but I recommend checking out one of the following (the first two are simple scikit-learn tools, the latter two are much more sophisticated algorithms and require installing new packages):

- GridSearchCV

- RandomizedSearchCV
- hyperopt-sklearn
- scikit-optimize

9. Explore feature selection for this problem. What are some particularly relevant and irrelevant features? Can you improve on your original logistic regression model if you first remove some irrelevant features?
10. Take your best model overall. Train it on the combined train/validation set and run it on the test set once. Does the test error agree fairly well with the validation error from before? Do you think you've had issues with optimization bias? Report your final test error directly in your README.md file as well as in your report.

**Submission format:** Your submission should take the form of a "report" that includes both code and an explanation of what you've done. You don't have to include everything you ever tried - it's fine just to have your final code - but it should be reproducible. For example, if you chose your hyperparameters based on some hyperparameter optimization experiment, you should leave in the code for that experiment so that someone else could re-run it and obtain the same hyperparameters, rather than mysteriously just setting the hyperparameters to some (carefully chosen) values in your code.

**Assessment:** We plan to grade and fairly leniently. We don't have some secret target accuracy that you need to achieve to get a good grade. You'll be assessed on demonstration of mastery of course topics, clear presentation, and the quality of your analysis and results. For example, if you write something like, "And then I noticed the model was overfitting, so I decided to stop using regularization" - then, well, that's not good. If you just have a bunch of code and no text or figures, that's not good. If you do a bunch of sane things and get a lower accuracy than your friend, don't sweat it.

**And...** This style of this "project" question is different from other assignments. It'll be up to you to decide when you're "done" -- in fact, this is one of the hardest parts of real projects. But please don't spend WAY too much time on this... perhaps "a few hours" (2-6 hours???) is a good guideline for a typical submission. Of course if you're having fun you're welcome to spend as much time as you want! But, if so, don't do it out of perfectionism... do it because you're learning and enjoying it.

```
In [ ]: # YOUR CODE AND REPORT HERE, IN A SENSIBLE FORMAT
```

### 1.4 Exercise 3: Very short answer questions

rubric={reasoning:7}

1. Why is it difficult for a standard collaborative filtering model to make good predictions for new items?

2. Consider a fully connected neural network with layer sizes (10,20,20,5); that is, the input dimensionality is 10, there are two hidden layers each of size 20, and the output dimensionality is 5. How many parameters does the network have, including biases?
3. Why do we need nonlinear activation functions in neural networks?
4. Assuming we could globally minimize the neural network objectve, how does the depth of a neural network affect the fundamental trade-off?
5. List 3 forms of regularization we use to prevent overfitting in neural networks.
6. Assuming we could globally minimize the neural network objectve, how would the size of the filters in a convolutational neural network affect the fundamental trade-off?
7. Why do people say convolutional neural networks just a special case of a fully-connected (regular) neural networks? What does this imply about the number of learned parameters?

*answer here*