EEE - 485

STATISTICAL LEARNING AND DATA ANALYTICS

PHASE II REPORT

KEREM AYÖZ - 21501569
MÜCAHİT FURKAN YILDIZ - 21400425

Retro Movie Recommender

1. Problem Description and Dataset

In Retro Movie Recommender Project, tried to implement a learning program to recommend movies to every user of our program. Watching movie is a very common and enjoyable event for idle times with multiple participant. Therefore, choosing movie is very important and hard mission. There are millions of films with different features such as categories, length, actors, tags. Choosing the right film among all them can be take a lot of time. A system that could give reasonable recommendation would be very helpful to people. As a movie lover, we thought that implementing such a program is reachable aim for us due to our knowledge on the topic. There are a lot of different ways to follow, however, we used some combination of the algorithms and tried to approach the problem from another aspects which described in the next section.

In this project, one of the important elements is dataset. We searched for dataset and founded different sets with different features. We have chosen MovieLens dataset at the proposal stage because of the enormous information that they contain such as millions of ratings, thousands of users and movies. However, we had to change it. We found another dataset from kaggle.com[1] which is also uses MovieLens movies. There is small and large user data but we used small one because of the huge size. Our rating matrix has 671 user and 9066 movies included with 100000 rating from 0 to 5 with resolution 0.5. In addition we have movie features data which we used to create movie feature vector. In that data for each movie there are adultness shows the adultness of the movie, belong to collection shows if the movie is in the collection, budget, genres, popularity shows the popularity value according to vote count, release date, spoken language, run time as minutes, vote count and vote average for every 9066 film. There are 20 different category for genre and 75 different language.

2. Review of the Methods

We used 3 different statistical learning methods for our project. Auto Encoder, Ridge Regression and Neural Network are the methods that we used.

2.1. User Feature Extraction

In our dataset, there are no features about the users in the dataset. In order to create a profile of a user, we used Autoencoder Neural Network. By using that structure, we obtained fixed size user-feature vector. The dimension of the vector is fixed number, which we can take it as a hyperparameter. We tried to generated a feature vector for each user.

We obtained the matrix where columns are representing movies and rows are representing users. The matrix entry Aij represents the rating of i th user on j th movie. After getting that matrix, we had a vector for each user which explains the movie taste of the user. Since the dimension of that vector is very large, we decided to use the Autoencoder Neural Network.

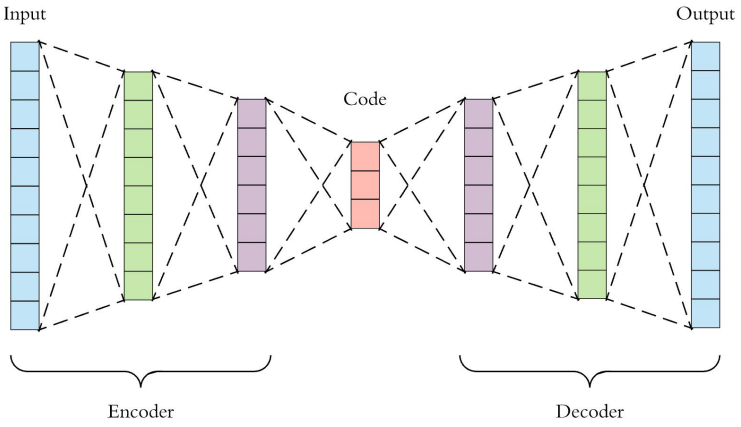

Fig.1. Autoencoder Neural Network Architecture

Autoencoder neural network simply tries to copy its input to output. After training, if we get the activation values of the code layer we obtain a input feature with much more less dimension than the input. In our project, we fed this structure with the user feature vectors and tried to get code layer activations. This gives us a lower dimensional user feature vector.

2.3. Ridge Regression to Make Feature Selection

The last step before we get into the rating estimation, we will estimate the importance of the features in the movie-feature. We have various features for each movie such as budget, genres, language and more other features. However; not all the features are equally important to determine the rating of the movie. In order to differentiate the importance of the features, we will put weights to each feature and will use a Ridge Regression to learn these weights. The label of the data will be the average rating of the movie. We will use the closed form solution of Ridge Regression which is as following;

$$\hat{\beta}^R = (\mathbf{X}^T\mathbf{X} + \lambda\mathbf{I})^{-1}\mathbf{X}^T\mathbf{y}$$

where rows of the X matrix represent the feature vector of each movie and y represents the average ratings of each movie. We put a threshold to select the features. We will select the features with the weights larger than the threshold. After learning the weights, we will build the reduced movie feature vector from the original one. The hyperparameter ƛ will be chosen at the test and validation stage of the project.

2.4. Rating Estimation with Neural Networks

After getting the user-feature vectors and reduced movie-feature vectors, we will train a model which takes these vectors as inputs. The label of that input data will be the actual rating that user gives to the movie. We will try to estimate the rating of the movie that the user gives.

We will have 2 neurons in the input layer and 1 neuron in the output layer. We will decide the number of hidden layers, number of neurons in that layers at the test and validation stage to choose the optimal numbers. A simple design of our model is as following;

After training the model, we now have a model that predicts the rating that user would give to a movie. Then we will take the movies with top 10 highest estimated rating and recommend these movies to the user. In training we will calculate the mean-squared error with that estimated value and actual rating that the user gives to that movie in training dataset. Also we will use the part of the dataset for testing.

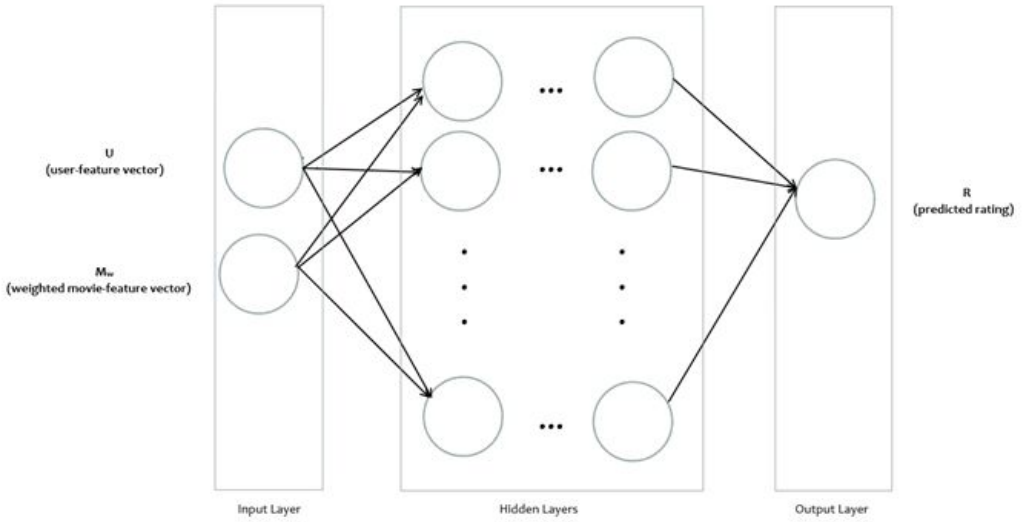

Fig.2. Rating predictor neural network architecture

3. Challenges

As we thought in the previous reports, we faced with some challenges that we need to spent times. Our first challenge processing the data. Some of the movies features was written wrongly in the .cvs file such as wrong date shape. Some of the entries blank and when we read it, categories and languages come as string and we need to encode to use algorithm to find the vector representation. We needed to create full category array and full language array and used t-bag encoder to store all the data. They are some time consuming challenges but the biggest one is occured at the end. When we coded autoencoder and the neural network, huge time problem is show up. We used stochastic gradient descent. However calculating the deltas took a lot of time even one datapoint took more than ten minutes. In literature, these calculations are generally made by gpu programming. However we are not able to gpu programming, our code was not that much efficient and time efficient.

4. Validation of Methods

We will use the k-fold cross validation method to increase the correctness of our model in the last step. The number k that we will use in our project was 5. This number makes us to use the %80 of the data for training and %20 of the data for testing. K-fold cross validation enabled us to use different parts of data for training and testing to obtain the best model. At the end we aim to reach very variation protected model.

5. Simulation

5.1. User Feature Selection

We implemented the Autoencoder Neural Network for obtaining the user feature vector. We used the user-movie rating matrix as input whose dimension is (671,9066). This indicates that we had 671 distinct users and these users rated 9066 distinct movies. User-movie matrix is a sparse matrix since on the average, a single user rated nearly 100 movies. Each row of that matrix is a training sample for the Autoencoder.

We set the Autoencoder neural network neuron counts for input and output layers as 9066 since we want to learn the lower dimensional estimate of the user-movie matrix entries. As we observed from [5], we reduced the hidden layer neuron counts as following:

$$9066 \rightarrow 4533 \rightarrow 2266 \rightarrow \ldots \rightarrow 2266 \rightarrow 4533 \rightarrow 9066$$

We will apply tests for deciding the neuron count of the bottleneck layer. It is a hyperparameter of our model which we will try to optimize it.

Since the actual training of the user-movie matrix samples, we created a small dataset to simulate the autoencoder. It works well with the small data. We also checked the activation values, delta values and weight update procedures while implementing the Autoencoder. We also used different activation functions such as RELU, tanh and sigmoid. We get the best result with RELU since tanh and sigmoid force some activations of networks to become 0.

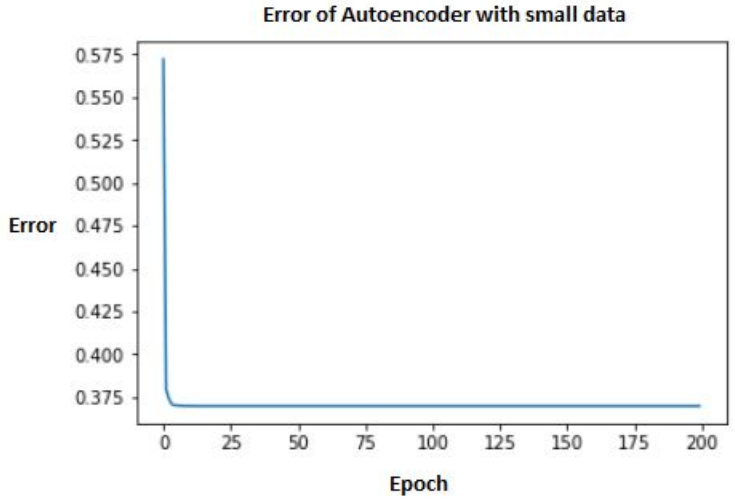

Fig.3. Error of small test data for Autoencoder

5.2. Movie Feature Selection

To select the features of movies, we decided to use ridge regression. Therefore, our first step is shaping the data. As described, our movieshave features such as adult, belong to collection, budget, genres, popularity, release date, spoken language, run time and vote count. To make ridge regression on the data, first we encoded it with t-bag encoding for the spoken languages and categories. After encoding, movies are represented with 102 dimensional vector whose first 20 dimensions used for showing the categories that movies include with ones and zeros that are not included. After categories there are 75 dimension for spoken languages with the same t-bag encoding ad 1 dimension for adult and inCollection.. At the end there are normalized budget, popularity, run time, vote count and date. Then use these vectors as data sample and make ridge regression and find the coefficients.

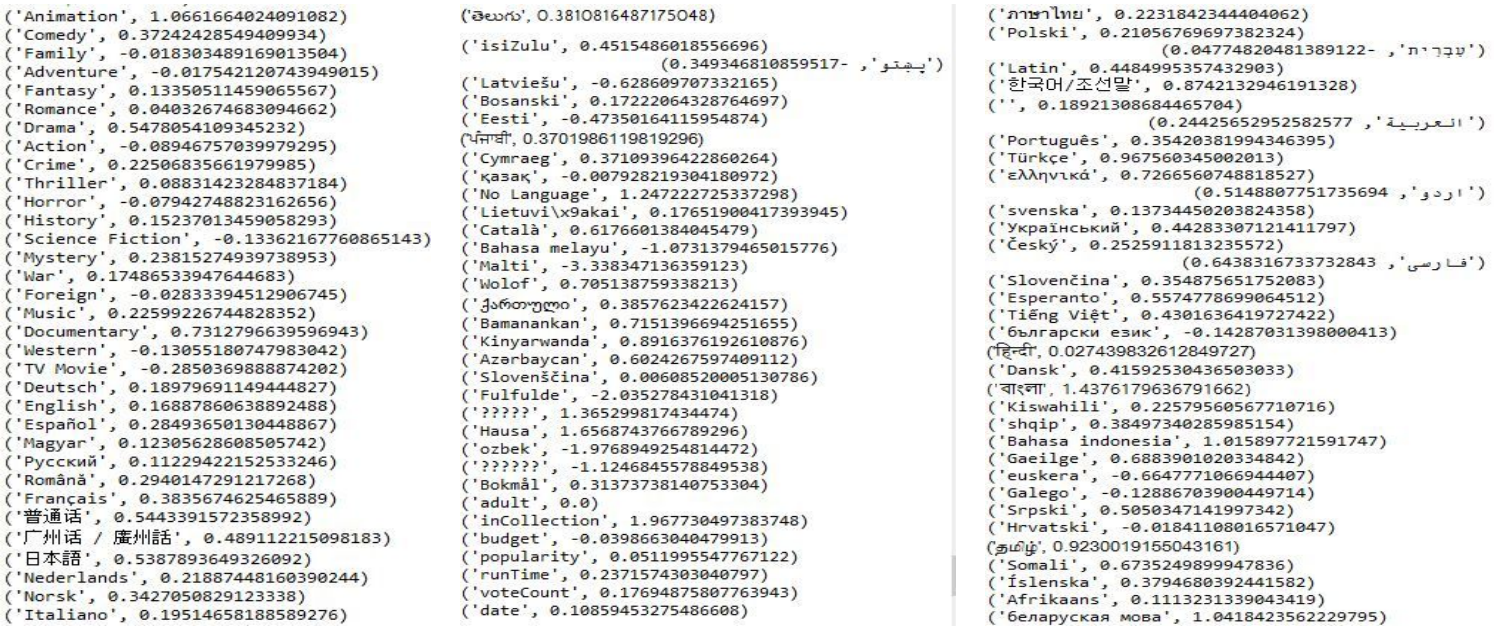

Fig.4. Weights of the various movie features

As selection we dropped the coefficients lower than 0.05. Then our training error will become 0.9667 which equal to the %19.334. %19 is not a good performance however by k-fold we believe that we can increase the performance of the our algorithm by optimizing the parameter in the future. Time usage of this part is around 10 sec. Therefore it is time efficient.

5.3. Neural Network Training

Since the computation of encoding tooks many time, we cannot train the neural network. It will also take many times as the same with the encoding part.

5. References

[1] "The Movies Dataset", *Kaggle.com*, 2018. [Online]. Available: https://www.kaggle.com/rounakbanik/the-movies-dataset/home. [Accessed: 05- Nov- 2018].

[2] F. Ricci, L. Rokach and B. Shapira, *Recommender Systems Handbook*. Boston, MA: Springer US, 2015.

[3] "Machine Learning for Recommender systems — Part 1 (algorithms, evaluation and cold start)", *Medium*, 2018. [Online]. Available: https://medium.com/recombee-blog/machine-learning-for-recommender-systems-part-1-algorithms-evaluation-and-cold-start-6f696683d0ed. [Accessed: 07- Nov- 2018].

[4] Towards Data Science. (2018). *Applied Deep Learning - Part 3: Autoencoders – Towards Data Science*. [online] Available at: https://towardsdatascience.com/applied-deep-learning-part-3-autoencoders-1c083af4d798 [Accessed 9 Dec. 2018].

[5] Lund, Jeffrey, and Yiu-Kai Ng. "Movie Recommendations Using the Deep Learning Approach." *2018 IEEE International Conference on Information Reuse and Integration (IRI)*. IEEE, 2018.

Appendix A - auto_encoder.py

```
import numpy as np
import matplotlib.pyplot as plt
import csv
import os
import random
from numpy import array
import math

get_ipython().run_line_magic('matplotlib', 'notebook')

def tanh(x, derivative=False):
    if (derivative == True):
        return (1 - (np.tanh(x) ** 2))
    return np.tanh(x)

def relu(x, derivative=False):
    if (derivative == True):
        return 1. * (x > 0)
    return x * (x > 0)

def sigmoid(x, derivative=False):
    sigm = 1. / (1. + np.exp(-x))
    if derivative:
        return sigm * (1. - sigm)
    return sigm

def activation_function(x, derivative=False):
    return relu(x, derivative)

class Autoencoder:

    def __init__(self, neuron_list):
        self.layer_count = len(neuron_list)
        self.neuron_list = neuron_list
        self.weights = [np.random.randn(y, x) for x, y in zip(neuron_list[:-1], neuron_list[1:])]
        self.biases = [np.zeros((y, 1), dtype=float) for y in neuron_list[1:]]
        self.activations = [np.zeros((x)) for x in neuron_list]
        self.deltas = [np.zeros((x)) for x in neuron_list]
        self.learning_rate = 0.01

    def forward_propogation(self, x):
        self.activations[0] = x
        for i in range(self.layer_count - 1):
                self.activations[i + 1] = activation_function(np.dot(self.weights[i], self.activations[i]) +
self.biases[i])
        return self.activations[-1]
```

```
    def compute_deltas(self, output_labels):
        # Compute last layers' activations
        for i in range(self.neuron_list[-1]):
            self.deltas[-1][i] = 2 * activation_function(self.activations[-1][i], True) * (output_labels[i] -
activation_function(self.activations[-1][i]))

        # Compute all deltas in all layers
        # l is layer starting from L-1, ending at 1
        for l in range(self.layer_count - 2, 0, -1):
            # i is representing neuron count in the layer l
            for i in range(len(self.activations[l])):
                # j is representing next layer's neurons
                for j in range(len(self.activations[l + 1])):
                                        self.deltas[l][i] = self.deltas[l + 1][j] * self.weights[l][j][i] *
activation_function(self.activations[l][i], True)

    def back_propogation(self, output_labels):
        # Compute deltas
        self.compute_deltas(output_labels)
        # Update weights
        # l is layer starting from L-1, ending at 1
        for l in range(0, self.layer_count - 1):
            # i is representing neuron count in the layer l
            for i in range(len(self.activations[l])):
                # j is representing next layer's neurons
                for j in range(len(self.activations[l + 1])):
                    self.weights[l][j][i] += self.learning_rate * self.deltas[l + 1][j] * self.activations[l][i]

    def train(self, x, y, epoch):
        error = []

        for e in range(epoch):
            pass_error = 0
            print("Epoch: " + str(e))
            for i in range(len(x)):
                estimation = self.forward_propogation(x[i])
                print("Est: " + str(np.transpose(estimation)))
                print("Out: " + str(np.transpose(y[i])))
                print("")
                pass_error += np.sum((estimation - y[i])**2) / len(x[i])
                self.back_propogation(y[i])
            error.append(pass_error / len(x))
        return error
```

Appendix B - movie_feature_selection.py

```
import pandas
import ast
import numpy as np
from sklearn.model_selection import train_test_split

df = pandas.read_csv('movies_metadata.csv')

#_____________________
#_____________________
t = ['Animation', 'Comedy', 'Family', 'Adventure', 'Fantasy', 'Romance', 'Drama', 'Action', 'Crime',
'Thriller', 'Horror', 'History', 'Science Fiction', 'Mystery', 'War', 'Foreign', 'Music', 'Documentary',
'Western', 'TV Movie']
categories = np.zeros((45457,20), dtype=int)
for y in range (45383):
    s=df['genres'][y]
    if not s == '[]':
        s=s[1:-1]
        s = ast.literal_eval(s)
        if not isinstance(s, dict):
            for m in range(len(s)):
                if s[m]['name'] in t:
                    categories[y][t.index(s[m]['name'])]=1
        else:
            categories[y][t.index(s['name'])]=1

#_____________________
#_____________________
c = []
for y in range (45383):
    s=df['spoken_languages'][y]
    if not s == '[]':
        s=s[1:-1]
        s = ast.literal_eval(s)
        if not isinstance(s, dict):
            for m in range(len(s)):
                if s[m]['name'] not in c:
                    c.append((s[m]['name']))
c.append('Bokmål')
languages = np.zeros((45383,75), dtype=int)
for y in range (45383):
    s=df['spoken_languages'][y]
    if not s == '[]':
        s=s[1:-1]
        s = ast.literal_eval(s)
        if not isinstance(s, dict):
            for m in range(len(s)):
```

```
            if s[m]['name'] in c:
                languages[y][c.index(s[m]['name'])]=1
        else:
            languages[y][c.index(s['name'])]=1
#_____________________________
#_____________________________
adult = np.zeros((45383),dtype=int)
for y in range (45383):
    s=df['adult'][y]
    if s == 'True':
        adult[y]=1
#_____________________________
#_____________________________
inCollection = np.zeros((45383),dtype=int)
for y in range (45383):
    s=df['belongs_to_collection'][y]
    if not s == 'nan':
        inCollection[y]=1
#_____________________________
#_____________________________
budget = np.zeros((45383),dtype=float)
for y in range (45383):
    s=df['budget'][y]
    budget[y] = float(s)
norm = np.std(budget)
budget  = budget/norm
#_____________________________
#_____________________________
popularity = np.zeros((45383),dtype=float)
for y in range (45383):
    s=df['popularity'][y]
    popularity[y] = float(s)
mean = np.mean(popularity)
popularity = popularity - mean
norm = np.std(popularity)
popularity  = popularity/norm
#_____________________________
#_____________________________
runTime = np.zeros((45383),dtype=float)
for y in range (45383):
    s=df['runtime'][y]
    if not np.isnan(s):
        runTime[y] = float(s)
norm = np.std(runTime)
runTime  = runTime/norm
#_____________________________
#_____________________________
voteCount = np.zeros((45383),dtype=float)
```

```
for y in range (45383):
    s=df['vote_count'][y]
    if not np.isnan(s):
        voteCount[y] = float(s)
norm = np.std(voteCount)
voteCount  = voteCount/norm
#_____________________________
#_____________________________
average = np.zeros((45383),dtype=float)
for y in range (45383):
    s=df['vote_average'][y]
    if not np.isnan(s):
        average[y] = float(s)
#_____________________________
#_____________________________

date = np.zeros((45383),dtype=float)
for y in range (45383):
    s=df['release_date'][y]
    if not isinstance(s, float):
        s = s[-4:]
        date[y] = float(s)
norm = np.std(date)
date  = date/norm

x = np.zeros((45383,102),dtype=int)

for Id in range (45383):
    for cat in range(20):
        x[Id][cat]=categories[Id][cat]
    for lan in range(75):
        x[Id][lan+20]=languages[Id][lan]
    x[Id][95]=adult[Id]
    x[Id][96]=inCollection[Id]
    x[Id][97]=budget[Id]
    x[Id][98]=popularity[Id]
    x[Id][99]=runTime[Id]
    x[Id][100]=voteCount[Id]
    x[Id][101]=date[Id]


x,x2, average, average2 = train_test_split(x, average, test_size=0.2)


tX = np.transpose(x)
xTx = np.dot(tX,x)
left = np.add(xTx,0.5*np.identity(102))
b = np.dot(np.linalg.inv(left),tX)
```

```
b = np.dot(b,average)

t += c
t.append('adult')
t.append('inCollection')
t.append('budget')
t.append('popularity')
t.append('runTime')
t.append('voteCount')
t.append('date')

print(np.sum(average - np.dot(x,b)))
```

Appendix C -data_process.py

```
import numpy as np
import matplotlib.pyplot as plt
import csv
import os
from numpy import array

from auto_encoder import NeuralNetwork


def column(matrix, i):
    return [row[i] for row in matrix]


project_root = os.path.dirname(os.path.abspath(""))
data_path = os.path.join(project_root, "data\\ratings_small.csv")
data = list(csv.reader(open(data_path)))

data = data[1:]
data = np.array(data)
data[1]

movies = []
for i in range(len(column(data, 1))):
    movies.append(data[i][1])
movies = set(movies)
movies = sorted(movies, key=lambda x: int(x), reverse=False)

users = []
for i in range(len(column(data, 0))):
    users.append(data[i][0])
users = set(users)
users = sorted(users, key=lambda x: int(x), reverse=False)

user_movie = np.zeros((len(users), len(movies)), dtype=float)

for i in range(len(data)):
    user_movie[int(data[i][0])][movies.index(data[i][1])] = data[i][2]

neuron_list = [9066, 4533, 2266, 2266, 4533, 9066]
aen = Autoencoder(neuron_list)
user_movie = np.transpose([user_movie])
err = aen.train(user_movie, user_movie, 10)
plt.plot(err)
plt.show()
print(err)
```