EEE - 485

STATISTICAL LEARNING AND DATA ANALYTICS

PHASE III 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 other 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 categories 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 generate 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.

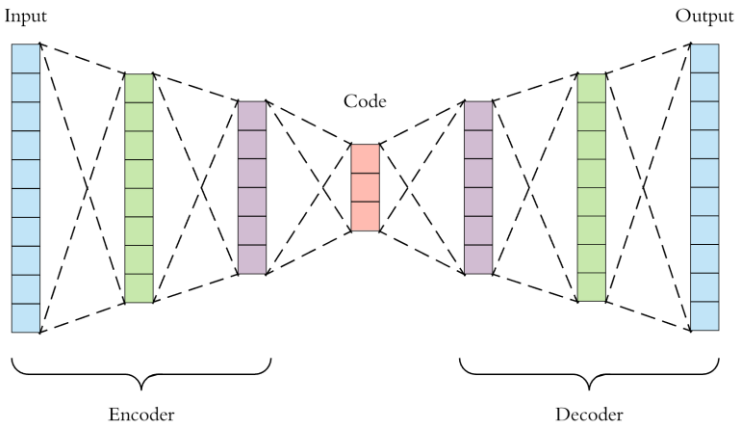

Fig.1. Autoencoder Neural Network Architecture

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 an 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;

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

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. A simple design of our model is as following;

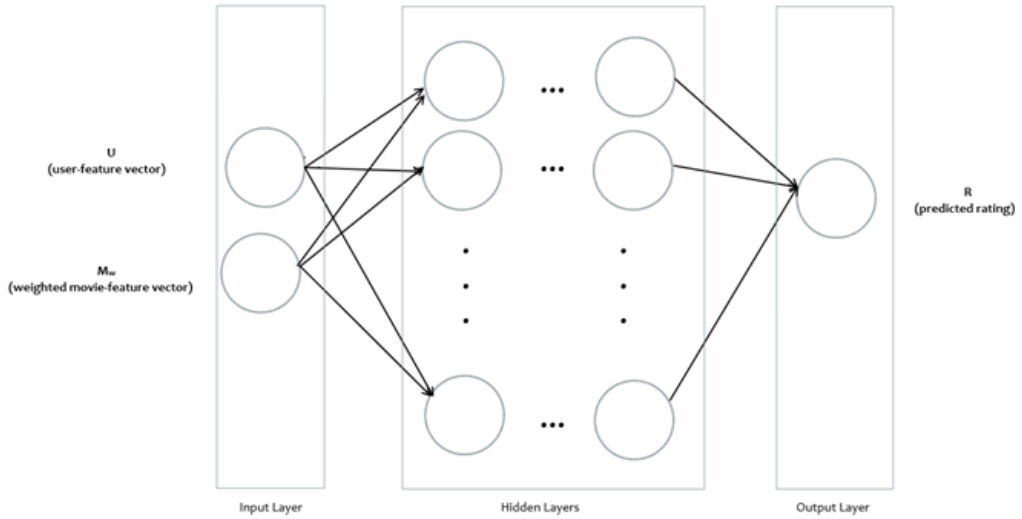

Fig.2. Rating predictor neural network architecture

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.

3. Challenges

As we thought in the previous reports, we faced with some challenges that we need to spend 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 occurred at the end. When we coded autoencoder and the neural network, huge time problem is show up. We used stochastic gradient descent. 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. In development process, we tried to optimize our code as much as possible to get output faster.

After Phase II, we fastened our Autoencoder and Neural Network's training by factor of 500. By vectorizing calculations and using the NumPy's fast matrix multiplication methods, we built a fast Neural Network and Autoencoder structures. Also, we optimized the data pre-processing stage by saving the important variables that takes time to calculate again and again using the Pickle library of Python. These improvements helped us a lot during the development process.

4. Validation of Methods

We will use the k-fold cross validation method to increase the correctness of our model in the last step in Neural Network. 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 (1000,499). This indicates that we had 1000 distinct users and these users rated 499 distinct movies. We eliminated the users whose total vote count is small. Also, we eliminated the movies with a smaller number of votes. Then we build that user-movie matrix. For empty entries, we write the average rating of the movie. In beginning, if user i did not watch the movie j, we make Aij=0. However, we thought that we were penalizing the user i for movie j if he did not watch it yet. We were assuming he will not like that movie. So, we make Aij=average_rating_of_movie_j. With that we assume if user did not watch the movie, he will probably like it as much as average.

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 499 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:

$$499 \rightarrow 350 \rightarrow 250 \rightarrow 180 \rightarrow 100 \rightarrow 75 \rightarrow 100 \rightarrow 180 \rightarrow 250 \rightarrow 350 \rightarrow 499$$

We decided neuron count decreasing rate and bottleneck layer neuron count by experimenting. We observed the results and get approximately %20 error at epoch 5 with that setup. We are able to represent each user with 75 dimensions instead of 499. Here is the error plot of the Autoencoder for extracting the user features:

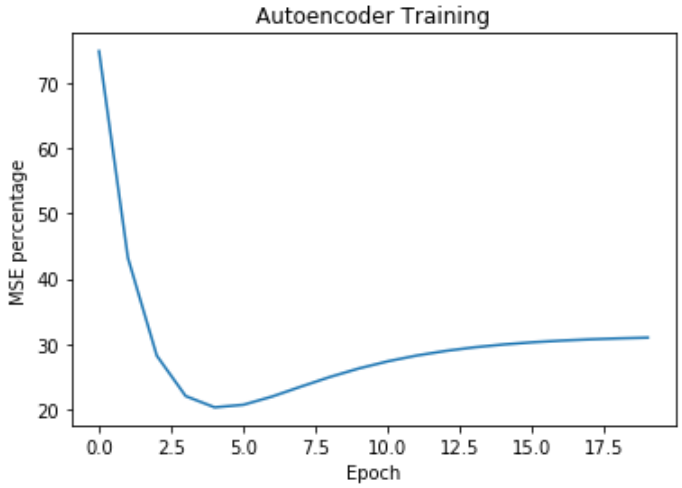

Fig.3. Error of user feature data for Autoencoder

Since the actual training of the user-movie matrix samples might give high error due to uncorrelation, we created a small dataset to simulate the autoencoder. This dataset contains correlated input data, so it can be reconstructed by Autoencoder with low error. It works well with the small data with %4 error. 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. Tests shows us that our Autoencoder works fine with correlated data.

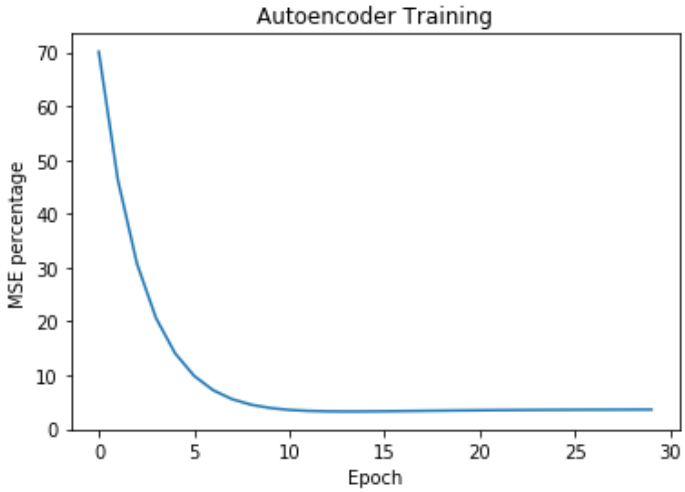

Fig.4. Error of small test data for Autoencoder

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 movies have 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 dimensions for spoken languages with the same t-bag encoding ad 1 dimension for adult and in Collection. 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.

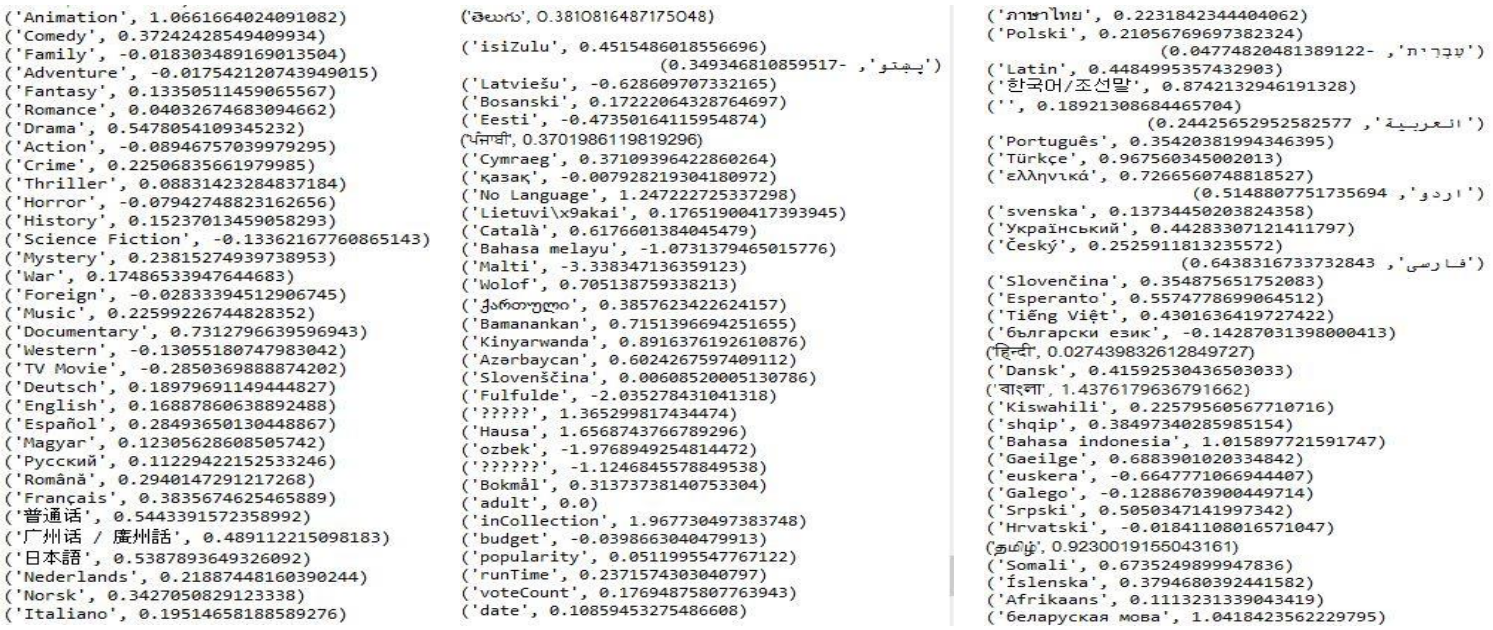

Fig.4. Weights of the various movie features

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 our algorithm by optimizing the parameter in the future. Time usage of this part is around 10 sec. Therefore, it is time efficient.

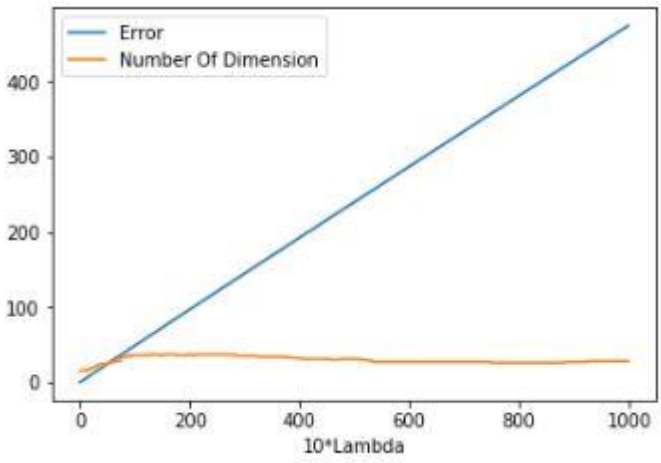

Fig.5. Lamda Selection Process

In ridge regression, our aim to drop some features of the movies which are uncorrelated, and the number of the captured dimension is changing with the lambda. To find optimal lambda we tried various values. According to that graph at first captured dimension is increasing with the lambda. Also, as expected, with lambda error is increasing because ridge punishes the coefficients. At optimal point lambda = 0.5 captured dimension is high while the error is not much. It means that without losing relations in dimensions we can drop some features.

5.3. Neural Network Training

After getting the results of the Autoencoder and Ridge Regression, we obtained 75-dimensional user features for each user and 64-dimensional movie features for each movie. So, our neural network has 140 input neurons (We also add 1 to input as bias). We trained our model with different neuron counts and different hidden layer counts. We used RELU as activation function except for output layer. In output layer we used sigmoid function as an activation function.

However, we could not get a percentage MSE less than %44.6 while training the Neural Network. We tried different neuron counts with different layer counts, different learning rates, different weight initializations. We get the best result with the following structure:

- Learning rate = 0.05
- Weights initialized randomly in interval [0, 0.01]
- Neuron Counts in each layer = [140,200,160,120,80,40,1]
- No shuffling

This error could be caused by uncorrelated data or wrong Neural Network setup. We tried too many options to find the cause of the problem, however we were not able to fix that problem. Because of that problem, we cannot get proper recommendations from the system. We will try to lower that error rate in the future versions and we will try to make better recommendations after getting a reasonable training error. The error plot of k-fold validation training of Neural Network is as follows:

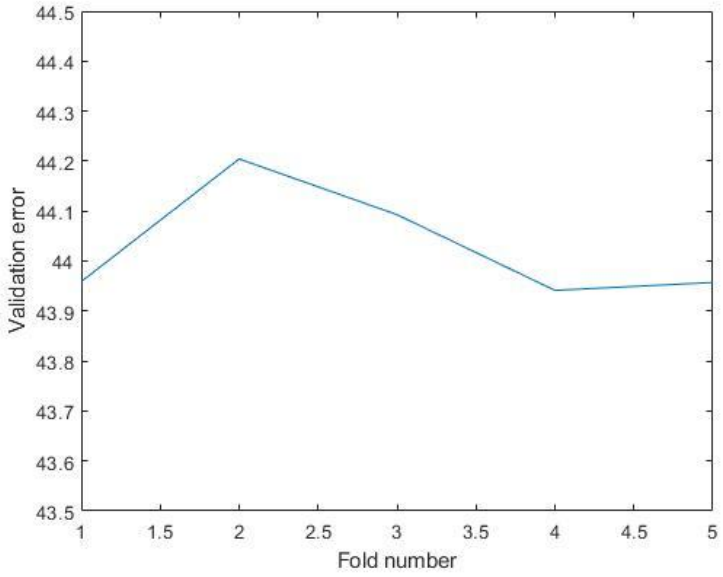

Fig.6. K-Fold Training Error for Neural Network

6. Conclusion

As a result, we experienced the development process of recommender system which uses different Machine Learning algorithms. We faced with many difficulties and some of them took too much time to overcome and some of them were impossible to solve, however we tried our best to achieve the best result. Especially, coding a Neural Network structure efficiently was a challenging task for us, however we overcome that problem. Also, hyperparameter optimization for Autoencoder and Neural Network were difficult since we could not try all possible options, we made a brain-storming activity to achieve best results.

Other thing that challenges us the data pre-processing part of the project. In our case, some of the movies does not have a single vote, some of them have different format for some features, some users have not voted a single movie. We searched through all of them and eliminated the unwanted data instances. Then we created the modified dataset and saved them as variables to access them faster without extra processing. This part of the project also took a lot of time for us.

We obtained user features with the %20 Autoencoder error. As we searched through internet, this error might not be a problem since Autoencoder can differentiate the users even with an error like %20. Our goal here is representing each user with less dimension to differentiate them, we think that our Autoencoder worked well for that purpose. Since when we looked for the encoded user features of each user, we saw that they are different in each user with considerable amount of differences.

We obtained movie features by using Ridge Regression. We used average rating as label and looked for features which affect the likeliness of the movies. We put a threshold and eliminated the features such as isAdult, budget, subtitle. Then we made t-bag encoding to the remaining features and get 64 dimensions for each movie. We select lambda value by trying different lambda values with the comparing the error rate.

We trained our rating predictor Neural Network with the features that we obtained in Autoencoder and Ridge Regression stages. However, we cannot get error less than %49 in training. We tried different combinations of neuron counts, layer counts, weight initializations and learning rate. Best error rate that we could get is %49 error. We did not have time to fix that problem, we may try different labels such as one-hot encoded labels, we may try to shuffle the data instances in every epoch, we may try different techniques for weight initialization such as Xavier Initialization. In the further development process, we will try to fix that problem to recommend movies with better percentage.

Since we could not obtain a reasonable training error in Neural Network, we could not get proper recommendations from the system. Recommendation process is fairly simple; to recommend movies to the new users of our system, we made them to rate some movies they watched before and create their profiles according to that selections. Then we inputted new user’s feature and all movies features one by one and take the maximum rating obtained from unwatched movies. However, the recommendations are not effective due to the high error rate that we got in Neural Network

7. 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 – Autoencoder Code

```
import numpy as np
import matplotlib.pyplot as plt
import csv
import os
import random
from numpy import array
import math
from matplotlib.pyplot import scatter
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, learning_rate, shrink_rate):
        self.layer_count = len(neuron_list)
        self.neuron_list = neuron_list
        self.weights = [abs(np.random.randn(y, x)) / shrink_rate for x, y in zip(neuron_list[:-1],
neuron_list[1:])]
        self.activations = [np.zeros((x, 1)) for x in neuron_list]
        self.deltas = [np.zeros((x, 1)) for x in neuron_list]
        self.learning_rate = learning_rate

    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]))
```

```
        return self.activations[-1]

    def encode(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]))
        return self.activations[int(self.layer_count / 2)]

    def compute_deltas(self, output_labels):
        # Compute last layers' activations
        self.deltas[-1] = 2 * activation_function(np.dot(self.weights[-1], self.activations[-2]), True) *
(output_labels - activation_function(self.activations[-1]))
        # Compute all deltas in all layers
        for l in range(self.layer_count - 2, 0, -1):
            np.dot(np.dot(np.transpose(self.weights[l]), self.deltas[l + 1]),
np.transpose(activation_function(np.dot(self.weights[l - 1], self.activations[l - 1]), True)))

    def back_propogation(self, output_labels):
        # Compute deltas
        self.compute_deltas(output_labels)
        # Update weights
        for l in range(0, self.layer_count - 1):
            self.weights[l] += self.learning_rate * np.dot(self.deltas[l + 1],
np.transpose(self.activations[l]))

    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])
                pass_error += 100 * np.sum(((estimation - y[i])**2) / (y[i]**2)) / len(estimation)
                self.back_propogation(y[i])
            error.append(pass_error / len(x))
            print("Error: " + str(pass_error / len(x)))
        return error

    def test(self, x, y):
        error = []
        pass_error = 0
        print("Epoch: " + str(e))
        for i in range(len(x)):
            estimation = self.forward_propogation(x[i])
            pass_error += 100 * np.sum(((estimation - y[i]) * 2) / (y[i] * 2)) / len(estimation)
        error.append(pass_error / len(x))
        print("Error: " + str(pass_error / len(x)))
        return error

# Sample correlated data
```

```
xx = np.array([8, 20])
yy = np.array([3, 15])
means = [xx.mean(), yy.mean()]
stds = [xx.std() / 3, yy.std() / 3]
corr = 0.5         # correlation
covs = [[stds[0]**2, stds[0] * stds[1] * corr],
        [stds[0] * stds[1] * corr,          stds[1]**2]]

m = np.random.multivariate_normal(means, covs, 200).T

xx = np.array([10, 50])
yy = np.array([4, 30])
means = [xx.mean(), yy.mean()]
stds = [xx.std() / 3, yy.std() / 3]
corr = 0.7       # correlation
covs = [[stds[0]**2, stds[0] * stds[1] * corr],
        [stds[0] * stds[1] * corr,          stds[1]**2]]

m2 = np.random.multivariate_normal(means, covs, 200).T

inp = []
for i in range(200):
    a1 = [m2[0][i],  m[1][i]]
    a2 = [m[0][i],  m2[1][i]]
    a3 = a1 + a2
    a3 /= np.linalg.norm(a3)
    inp.append(np.transpose([a3]))

# Example training with correlated data
np.set_printoptions(suppress=True)
neuron_list = [4, 2, 4]
nn = Autoencoder(neuron_list, 0.01, 10)
err = nn.train(inp, inp, 30)
plt.plot(err)
plt.title('Autoencoder Training')
plt.xlabel('Epoch')
plt.ylabel('MSE percentage')
plt.show()
```

Appendix B – Data Pre-Processing Code

```
import numpy as np
import matplotlib.pyplot as plt
import csv
import os
import pickle
import ast
import pandas
from numpy import array
import nbimporter
import auto_encoder as aec
np.set_printoptions(suppress=True)

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


# Linking Id's
'''
project_root = os.path.dirname(os.path.abspath(""))
data_path = os.path.join(project_root, "data\\ratings.csv")
rating = pandas.read_csv(data_path)

project_root = os.path.dirname(os.path.abspath(""))
data_path = os.path.join(project_root, "data\\links.csv")
links = pandas.read_csv(data_path)

project_root = os.path.dirname(os.path.abspath(""))
data_path = os.path.join(project_root, "data\\movies_metadata.csv")
movie_meta = pandas.read_csv(data_path)

joined = links.set_index('tmdbId').join(movie_meta.set_index('id'))

writer = pandas.ExcelWriter('linked_movieIds.xlsx')
joined.to_excel(writer,'Sheet1')
writer.save()
'''


# Removing less-voted movies & least frequent voters from ratings.csv
'''
project_root = os.path.dirname(os.path.abspath(""))
data_path = os.path.join(project_root, "data\\ratings.csv")
rating = pandas.read_csv(data_path, engine='python')

project_root = os.path.dirname(os.path.abspath(""))
data_path = os.path.join(project_root, "data\\movie_votecount.csv")
vote_count = pandas.read_csv(data_path)
```

```
votes = column(vote_count.values, 0)
ratings_updated = rating.loc[rating["movieId"].isin(votes)]

project_root = os.path.dirname(os.path.abspath(""))
data_path = os.path.join(project_root, "data\\user_votecount.csv")
user_vote = pandas.read_csv(data_path)

votes = column(user_vote.values, 0)
ratings_updated = ratings_updated.loc[ratings["userId"].isin(votes)]

file = open("ratings_updated", "wb")
pickle.dump(ratings_updated, file)
'''


file = open("ratings_updated", "rb")
rating = pickle.load(file)

# Removing less-voted movies & least frequent voters from ratings.csv
'''
project_root = os.path.dirname(os.path.abspath(""))
data_path = os.path.join(project_root, "data\\movie_votecount2.csv")
movie_vote = pandas.read_csv(data_path)
votes = column(movie_vote.values, 0)
ratings_updated = rating.loc[rating["movieId"].isin(votes)]

project_root = os.path.dirname(os.path.abspath(""))
data_path = os.path.join(project_root, "data\\user_votecount2.csv")
user_vote = pandas.read_csv(data_path)

votes = column(user_vote.values, 0)
ratings_updated = ratings_updated.loc[rating["userId"].isin(votes)]
file = open("ratings_updated_2", "wb")
pickle.dump(ratings_updated, file)
'''

file = open("ratings_updated_2", "rb")
ratings_updated = pickle.load(file)

# Movie matrix build
movies = ratings_updated["movieId"]
movies = set(movies)
movies = sorted(movies, key=lambda x: int(x), reverse=False)

# User matrix build
users = ratings_updated["userId"]
users = set(users)
users = sorted(users, key=lambda x: int(x), reverse=False)

# User-movie matrix build
```

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

project_root = os.path.dirname(os.path.abspath(""))
data_path = os.path.join(project_root, "data\\movies.csv")
movies_meta = pandas.read_csv(data_path, encoding = "ISO-8859-1")

# Fill non-rated movies as the average rating of that movie
for i in range(len(movies)):
   rating_of_i = (movies_meta.loc[movies_meta["movieId"] ==
movies[i]])["vote_average"].values[0]
   x = np.full((len(users),), rating_of_i / 2, dtype=float)
   user_movie[:,i] = x

for index, row in ratings_updated.iterrows():
   user_movie[users.index(row['userId'])][movies.index(row['movieId'])] = row['rating']

print(np.shape(user_movie))
file = open("user_movie", "wb")
pickle.dump(user_movie, file)

# Load user-movie matrix
file = open("user_movie", "rb")
user_movie = pickle.load(file)
np.shape(user_movie)

# Train the Autoencoder to get user features
neuron_list = [499,350,250,180,100,75,100,180,250,350,499]
nn = aec.Autoencoder(neuron_list, 0.01, 300)
inp = np.transpose([np.transpose(user_movie)])
print(np.shape(inp))
err = nn.train(inp,inp, 6)
plt.plot(err)
plt.title('Autoencoder Training')
plt.xlabel('Epoch')
plt.ylabel('MSE percentage')
plt.show()
print(err)

# Extract user-features after training
features = []
for i in range(len(user_movie)):
   encoded = nn.encode(inp[i])
   features.append(encoded)

user_features = []
for i in range(len(users)):
   user_features.append([users[i], features[i]])

file = open("user_features", "wb")
pickle.dump(user_features, file)
```

Appendix C– Movie Feature Selection Code

```
import pandas
import ast
import os
import pickle
import numpy as np
from sklearn.model_selection import train_test_split
import math
import matplotlib.pyplot as plt

project_root = os.path.dirname(os.path.abspath(""))
data_path = os.path.join(project_root, "data\\movies.csv")
df = pandas.read_csv(data_path, encoding="ISO-8859-1")

#_____________________________________________________________________
#_____________________________________________________________________
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((len(df), 20), dtype=int)
for y in range(len(df)):
    s = df['genres'][y]
    if not s == '[]' and not pandas.isnull(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(len(df)):
    s = df['spoken_languages'][y]
    if not s == '[]' and not pandas.isnull(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((len(df), 75), dtype=int)
for y in range(len(df)):
    s = df['spoken_languages'][y]
    if not s == '[]' and not pandas.isnull(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((len(df)), dtype=int)
for y in range(len(df)):
    s = df['adult'][y]
    if s == 'True':
        adult[y] = 1

#_____________________________________________________________________
#_____________________________________________________________________
inCollection = np.zeros((len(df)), dtype=int)
for y in range(len(df)):
    s = df['belongs_to_collection'][y]
    if not s == 'nan':
        inCollection[y] = 1


#_____________________________________________________________________
#_____________________________________________________________________
budget = np.zeros((len(df)), dtype=float)
for y in range(len(df)):
    s = df['budget'][y]
    if not s == '[]' and not pandas.isnull(s):
        budget[y] = float(s)
mean = np.mean(budget)
budget = (budget - mean)
budget = budget / np.max(budget)
#_____________________________________________________________________
#_____________________________________________________________________
popularity = np.zeros((len(df)), dtype=float)
for y in range(len(df)):
    s = df['popularity'][y]
    if not s == '[]' and not pandas.isnull(s):
        popularity[y] = float(s)
mean = np.mean(popularity)
popularity = (popularity - mean)
popularity = popularity / np.max(popularity)
#_____________________________________________________________________
#_____________________________________________________________________
runTime = np.zeros((len(df)), dtype=float)
for y in range(len(df)):
```

```
    s = df['runtime'][y]
    if not np.isnan(s):
        runTime[y] = float(s)
mean = np.mean(runTime)
runTime = (runTime - mean)
runTime = runTime / np.max(runTime)
#_____________________________________________________________________
#_____________________________________________________________________
voteCount = np.zeros((len(df)), dtype=float)
for y in range(len(df)):
    s = df['vote_count'][y]
    if not np.isnan(s):
        voteCount[y] = float(s)

mean = np.mean(voteCount)
voteCount = (voteCount - mean)
voteCount = voteCount / np.max(voteCount)

#_____________________________________________________________________
#_____________________________________________________________________
average = np.zeros((len(df)), dtype=float)
for y in range(len(df)):
    s = df['vote_average'][y]
    if not np.isnan(s):
        average[y] = float(s)

#_____________________________________________________________________
#_____________________________________________________________________

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

x = np.zeros((len(df), 102), dtype=float)

for Id in range(len(df)):
    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]


error = []
numOfDim = []
for i in range(1000):
    tX = np.transpose(x)
    xTx = np.dot(tX, x)
    left = np.add(xTx, (i * 0.1 + 0.01) * np.identity(102))
    b = np.dot(np.linalg.inv(left), tX)
    b = np.dot(b, average)
    error.append(np.sum(average - np.dot(x, b)))
    # Get movieId - movieFeatureVector array from data
    threshold = max(b) / 50
    c = np.where(abs(b) > threshold)[0]
    numOfDim.append(len(c))
    #x_reduced = x[:, c]

plt.plot(error, label='Error')
plt.plot(numOfDim, label='Number Of Dimension')
plt.xlabel('10*Lambda')
plt.legend(loc='upper left')
plt.show()

tX = np.transpose(x)
xTx = np.dot(tX, x)
left = np.add(xTx, (50 * 0.1 + 0.01) * np.identity(102))
b = np.dot(np.linalg.inv(left), tX)
b = np.dot(b, average)

error.append(np.sum(average - np.dot(x, b)))
# Get movieId - movieFeatureVector array from data
threshold = max(b) / 50
c = np.where(abs(b) > threshold)[0]
numOfDim.append(len(c))
x_reduced = x[:, c]

movie_features = []
for index, row in df.iterrows():
    movie_features.append([row['movieId'], x_reduced[index]])

file = open("movie_features", "wb")
pickle.dump(movie_features, file)
```

Appendix D– Neural Network Code

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

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

    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]))
        return self.activations[-1]

    def compute_deltas(self, output_labels):
        # Compute last layers' activations
        self.deltas[-1] = 2 * activation_function(np.dot(self.weights[-1], self.activations[-2]), True) *
(output_labels - activation_function(self.activations[-1]))
        # Compute all deltas in all layers
        for l in range(self.layer_count - 2, 0, -1):
            np.dot(np.dot(np.transpose(self.weights[l]), self.deltas[l + 1]),
np.transpose(activation_function(np.dot(self.weights[l - 1], self.activations[l - 1]), True)))

    def back_propogation(self, output_labels):
        # Compute deltas
        self.compute_deltas(output_labels)
        # Update weights
        for l in range(0, self.layer_count - 1):
            self.weights[l] += self.learning_rate * np.dot(self.deltas[l + 1],
np.transpose(self.activations[l]))

    def shuffle(self, x, y):
        total = np.hstack((x, y))
        np.random.shuffle(total)
        b = total[:, 0:-np.shape(x)[1]]
        a = total[:, -np.shape(x)[1]:]
        return a, b

    def train(self, x, y, epoch):
        error = []
        for e in range(epoch):
            instance_error = 0
            print("Epoch: " + str(e))
            for i in range(len(x)):
                estimation = self.forward_propogation(x[i])
                instance_error += 100 * np.sum(((estimation - y[i])**2) / (y[i]**2)) / len(estimation)
                self.back_propogation(y[i])
            print("Error: " + str(instance_error / len(x)))
            error.append(instance_error / len(x))
        return error

    def test(self, x, y):
        error = []
        instance_error = 0
        for i in range(len(x)):
            estimation = self.forward_propogation(x[i])
            instance_error += 100 * np.sum(((estimation - y[i]) * 2) / (y[i] * 2)) / len(estimation)
        print("Test Error: " + str(instance_error / len(x)))
        error.append(instance_error / len(x))
        return error
```

Appendix E– Rating Prediction with Neural Network

```
import numpy as np
import matplotlib.pyplot as plt
import csv
import os
import pickle
import ast
import pandas
from numpy import array
import nbimporter
import neural_network as nn
np.set_printoptions(suppress=True)


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

# Read user and movie features
file = open("movie_features", "rb")
movie_features = pickle.load(file)

file = open("user_features", "rb")
user_features = pickle.load(file)


'''
# Read ratigns.csv to build labeled data
file = open("ratings_updated_2", "rb")
ratings = pickle.load(file)

input_data = []
input_label = []
for index, row in ratings.iterrows():
    uu = user_features[column(user_features,0).index(row['userId'])][1]
    mm = movie_features[column(movie_features,0).index(row['movieId'])][1]
    input_data.append(np.concatenate((uu, mm), axis=None))
    input_label.append(row['rating'])

file = open("input_data", "wb")
pickle.dump(input_data, file)

file = open("input_label", "wb")
pickle.dump(input_label, file)
'''

# Load input data and input label
file = open("input_data", "rb")
input_data = pickle.load(file)
```

```
# Add 1 to the input to use bias
left = np.full((len(input_data), 1), 1, dtype=float)
input_data = np.hstack((left, input_data))

file = open("input_label", "rb")
input_label = pickle.load(file)
input_label = [x / 5 for x in input_label]

# Divide it for train and validation
inp = np.asarray(input_data[:-4])
inp = np.split(inp, 5)

label = np.asarray(input_label[:-4])
label = np.split(label, 5)

# K-fold validation
i = 1
print("Fold " + str(i + 1))

neural = nn.NeuralNetwork([140, 200, 160, 120, 80, 40, 1], 0.04, 100)

inpFold = np.vstack((inp[i % 5], inp[(i + 1) % 5], inp[(i + 2) % 5], inp[(i + 3) % 5]))
inpFold = np.transpose([np.transpose(inpFold)])

labelFold = np.hstack((label[i % 5], label[(i + 1) % 5], label[(i + 2) % 5], label[(i + 3) % 5]))
labelFold = np.transpose([[labelFold]])

testInp = np.transpose([np.transpose(inp[(i + 4) % 5])])

print("Training start")
errTrain = neural.train(inpFold, labelFold, 5)
print("Training end")
print("Test error for fold " + str(i + 1))
errTest = neural.test(testInp, testLabel)
plt.plot(errTrain)
plt.plot(errTest)
plt.show()
```