MOVIE SUCCESS PREDICTION USING TENSOR-FLOW

University of Houston Clear Lake

Data Mining Tools and Technique

Contributors:

Roshan Ghale

Mohammed Ifthikhar

Xiaodi Fu

Ankita Puranik

Preetham John

Table of Content

1. Purpose of the project

Movie Industry is one among the industries where millions of dollars are invested every year. Every movie is released with the aim of having a profit in mind but there is no certainty in the success of a movie. Predicting a success of the movie can be one solution to reduce the loss in term of money and effort undergone by the movie crew. Data mining play the important role in predicting the success of a movie. Accurate prediction can help to reduce the financial loss incurred by the producers. Lot of research has been going on for this purpose.

In our project, we will collect data about movies like revenue, budget, Facebook likes of movie, total, IMDB score, user votes for actor, actress, and some more, both before and after release date. All these data are collected from popular movie database websites IMDB, MovieLens.

Since the previous information about the success are known, classification can be the good choice for predicting the success of new movies based on the classification model trained from the previous movie data. There are many Classification algorithms like Naïve Bayes classification, Support Vector Machine, Decision tree, Neural Network. Though Naïve Bayes clarification is simple in terms of time and mathematical computational complexity, as the attributes used for prediction are considered as independent of each other, for this method, we cannot fit our movie data as there are close correlation between the number of user voted and vote count for each actor. This makes us to not use Bayes classifier. All these drawbacks from these model are overcome by implementing Deep Neural Network classifier using the Google's Tensor Flow Framework.

Identifying movie success is usually a binary classification problem because a movie is classified as a success or not a success (flop). But we will be using multi-classification model for predicting movie success in term of five classes (ranging from flop till Blockbuster). After prediction of movie success, we have evaluated the prediction rate using various quality metrics and result are presented in terms of percentage. This developed multi-classification model has the accuracy over 80 percentages over the test data used. For visualization of model metrics, TensorFlow visualization tools called TensorBoard is used.

2. **Expected Outcome or Result of Project**

After getting data from movie database sites, we have processed data and performed data cleaning. Finally we used data for over 800 movies. 500 movies are used for the training set and 300 movies are used for the test set. Model is created for predicting the success of the movie. For instance, when the attribute, value pair like the below is given to the DNN Classifier

num_critic_for_reviews': 184,
'revenue': =59600000,
'num_voted_users': 116642,
'cast_total_facebook_likes': 1421,
'num_user_for_reviews':389,
'budget': 950000,
'imdb_score': 7.3,
'movie_facebook_likes': 5000,
'vote_average': 6.9,
'vote_count': 953,
'cast_power': 1194,

The classifier need to predict the success as 'Super Hit' Movie. After model implantation and training with four Hidden Layer of 1024, 512, 256 and 128 nodes in each layer and using Rectified Linear unit and ProximalAdagradOptimizer, we are able to predict the movie as 'Super Hit' as expected with accuracy rate of over 95%. When all the training data tuples are considered for accuracy of the model, model shows the satisfactory result over 69%. Taking the multivalued class label prediction in considering, the accuracy metrics are high compared to other classification model, when the same datasets are used.

3 Background Research:

Research is still going on for the predicting success of a movie by using the IMDB datasets. Most of the researchers preferred the movie's box office gross revenue based on the datasets available from IMDB. And most of them used the binary classification model to predict a movies success scenario, either success or failure. In some previous researches, prediction of a movie success was made based on the available pre-release data like movie crew, budget, genre[1]. Some of the researchers implement Natural Language processing algorithm based on the comments provided by the users and critics of the movie to judge its sentiment. Creating the attribute and inserting to training data, after the sentimental analysis of both user and Critics comments can be give more accurate result but the preprocessing steps take more computation time.

IMDB and Box Office Mojo are the datasets repository for the movies used by many of the researchers but many of those used insufficient or irrelevant features to deal with this prediction. Some researchers used the idea of the social networking comments and influence of the hype factor to predict the success rate of the movie [2]. They preferred simple mathematical calculations instead of the Natural Language Processing to predict the results. While summing up the problem, most of the researchers classified this type of the prediction as the classification problem. Furthermore, many people did not consider the number of the screens and country's GDP as an attribute in their analysis. The number of the screens is also one of the reasons that it is not possible to get an appropriate result and when the movie released country has low GDP, obviously the number of people watching the movie can be low as compared to country good in terms of economic condition. Many of the researchers either analyze pre-released data like a director, writers, cast members, number of screens, budgets, etc. or post-released data available at IMDB. In some cases, researches used both data before and after release of movie but didn't make use of whole set of these attribute in their prediction model.

4 Methodology Taken:

Machine learning library are growing at the fast rate for rapid application development. Keras is widely used Library for neural network and other supervised and unsupervised learning problems.

For the last five years, Google's Brain team had also worked on building a Machine learning application program interface called TensorFlow. TensorFlow is an open source framework for machine learning and its first stable release was November 9, 2015. Since its release it is been used in various applications like Voice and image recognition, test based analysis. The most recent application is Google has implemented Tensorflow in Gmail application for smart reply in email. Due to TensorFlow ability to run the application in Central Processing Unit, Graphical Processing Unit and running the code in parallel across various node in cloud make it the ideal choice for our Movie Success prediction application. TensorFlow make use of Python for front end API for building application with framework and all internal complex mathematical processing is done in C++.

The First steps that is obvious in classification problem is the dataset collection and based on the dataset we will able to assess whether the model/Classifier is accurate or not. IMDB is considered as the largest Movie Database repository for getting information about the movie. Some of the information like Facebook like, director and actor rating are not provided directly in the IMDB Site so we had make use of MovieLens dataset which has these features. These features are combined with the IMBD dataset for further processing. After combining the features we had 15 different feature attributes. Using correlation analysis tools provide by Keras API, we had tried to remove some of the attribute which are not related to each other. Total cast power is considered as more important attribute when compared to Director and attribute name, so we had removed them from our dataset. Director and genres will be ideal choice for the movie recommendation system but not for the movie success prediction system. After omitting some feature from the original datasets, we had total of 4000 movies.

Outliers are the records which have attribute values not too distant from other dataset points. Noisy data also plays the significant part in reducing the accuracy of classifier. So we have removed the outlier, noise in our analysis. We used the following methodologies to analyze the data.

1. Q – Q Plot

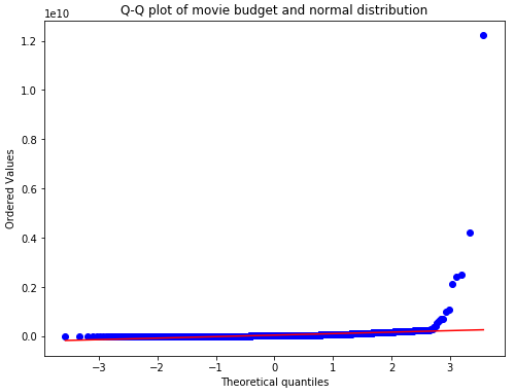

Fig 1

From the figure 1, we can know the distribution of movie budget is a skewed distribution. It is very different from normal distribution. The majority of movie budget focus relatively low level.

From the figure 2, we can get that the distribution of IMDB rating scores is similar with normal distribution, and a little right skewed.

From figure 3, we can conclude that there some one outlier in the plot, and the majority of movie revenue are very intensively near a level. Therefore, the distribution of movie revenue is a left skewed distribution.

From the figure 4, we can get at least three points.

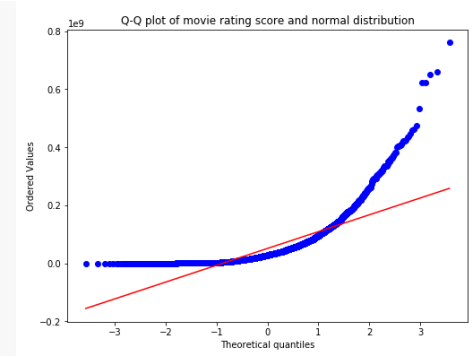

Fig 2

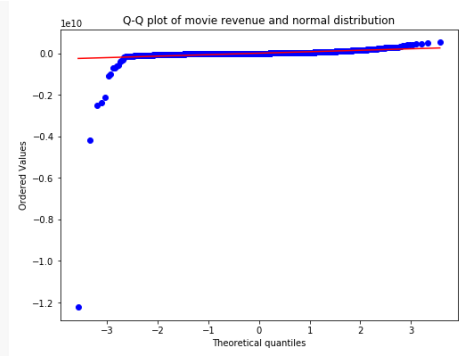

Fig 3

2. Matrix scatter of IMDB score, revenue, gross and budget

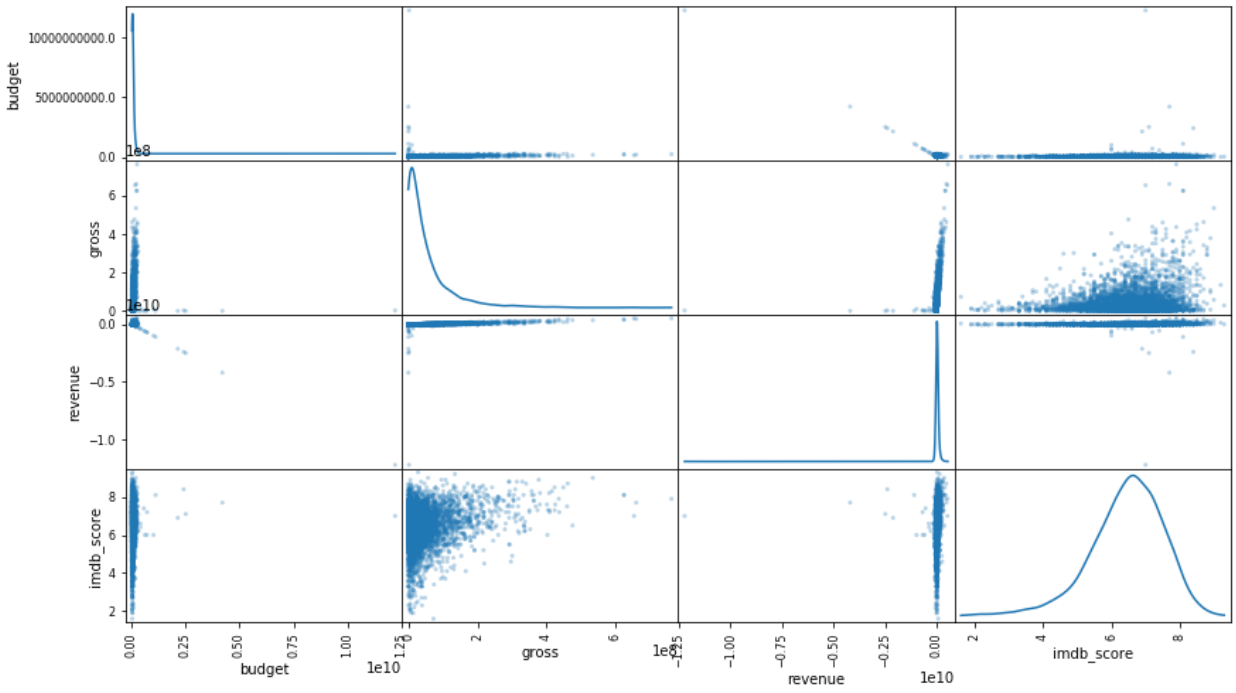

Fig 4

The first one is that the more budget of a movie,it has smaller probability to be low rating score movie. The second, grosser means that a movie has more probability to have higher rating score. The third, the distribution of revenue is a much skewed distribution.

3. Matrix scatter of IMDB score, critic reviews, user reviews, and revenue

From figure 5, we can conclude those following points:

- If a movie has more critic reviews, it has more opportunity to have higher rating score. However, relatively fewer review (the number is between 0 and 200) does not mean that they have lower rating score.
- The relation between the number of users' reviews and rating scores is similar with the relation between critic reviews and rating scores. If a movie has more users' reviews, it has more chance to be a higher rating score movie.

Relation between review and revenue is not obvious because there are some special points. Therefore, it may need sampling analysis.

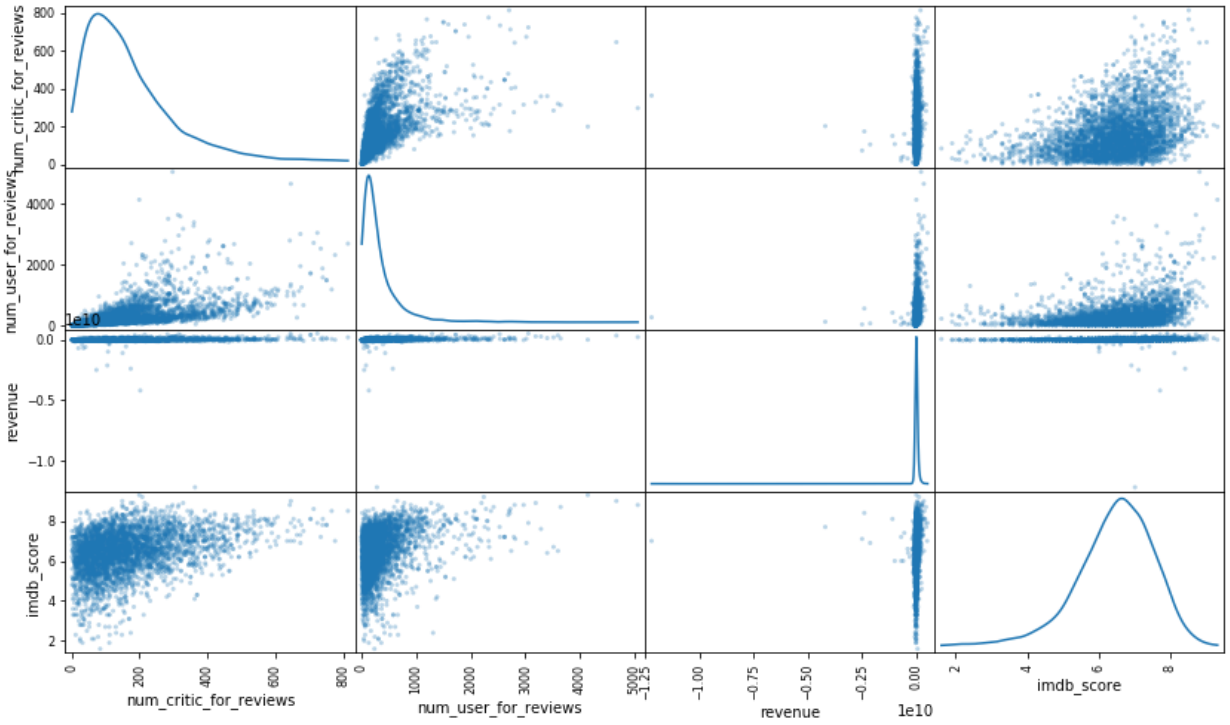

Fig 5

4. The correlation of budget, gross and revenue and IMDB rating score

From the analysis of correlation co-efficient shown Figure 6, we can see Gross has more relation with movie score than Budget and Revenue. So whether a movie is popular majorly depends on Gross. To customers, rating score can be a standard of selecting a movie. So from customers' perspective, higher rating score has more opportunities to become a success movie, even though it has low income. From the above matrix plots, we can see that Gross distribution is similar with score rating distribution. Generally speaking, a movie having more gross means that it's rating score is higher. We can also see that budget has inverse linear relation with income. From investors' perspective, more budget means more risk and relatively lower income.

5. Relation of revenue and critic review and the relation of revenue and user reviews

We can see the relation between revenue and the number of critic reviews is weak linear. The relation between revenue and the number of user reviews is also weakly linear.

6. Relation between director Facebook likes and rating scores

The phenomena is similar with the relation between director Facebook likes and rating scores as shown in Figure 9.

```
#analyze correlation coeffcient
#calculate the correlation coefficient for testing non-correlation
Budget_Score=stats.pearsonr(Money1['budget'], Money1['imdb_score'])[0]
Budget_Score
```

```
0.02913467819824174
```

```
#analyze correlation coeffcient
#calculate the correlation coefficient for testing non-correlation
Gross_Score=stats.pearsonr(Money1['gross'], Money1['imdb_score'])[0]
Gross_Score
```

```
0.21152530306948464
```

```
#analyze correlation coeffcient
#calculate the correlation coefficient for testing non-correlation
Income_Score=stats.pearsonr(Money1['revenue'], Money1['imdb_score'])[0]
Income_Score
```

```
0.036633512802823925
```

Fig 6

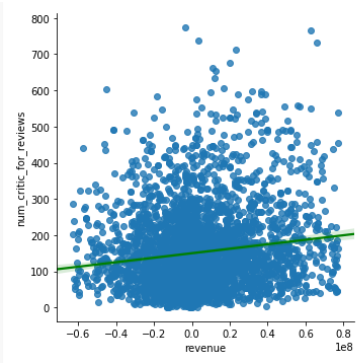

Fig 7

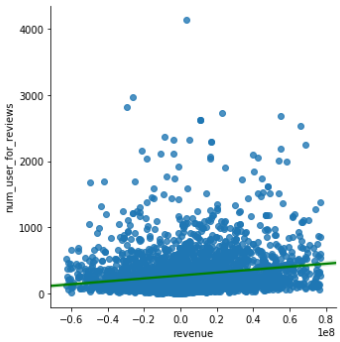

Fig 8

The majority of movies whose rating score are under 5 has smaller number of actor actress Facebook likes. It means that if the number of actor actress Facebook likes is smaller, a movie has more opportunity to be classified as a low rating score movie. According similarity (K-means clustering) between each point, we can classify those movies into two class. The cluster whose number of actor actress Facebook likes are about smaller than 10000 is yellow. And cluster whose number of actor actress Facebook likes are about larger than 10000 or equal to 10000 is blue. When the number of actor actress Facebook likes is about bigger than 10000, the relation between actor actress Facebook likes and ranting score is little.

5 Steps followed in reducing the noisy data:

1. Removed the data Tuples/Records whose values are missing for more than 3 feature attribute.

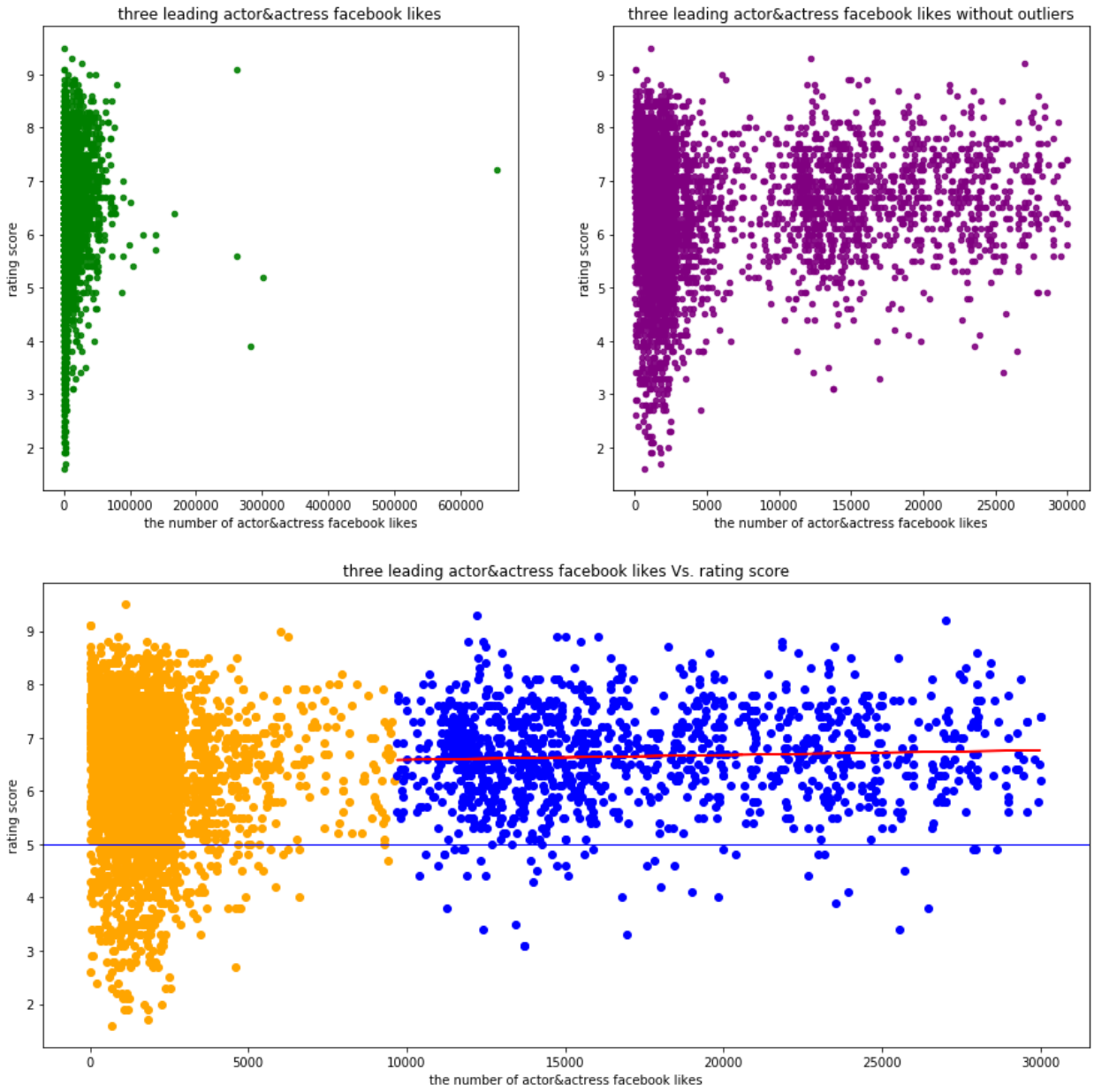

Fig 9

2. Used mean value when the some of the attribute rows like cast power, is Zero. For this kind of arithmetic, the data need be in numeric type.

We have included include the cast power and Rank attribute (class label) based on doing calculation from the critic vote, Facebook likes, actors vote and some more.

Table 1 represents the features and their respective datatype after data cleaning. We are using both pre and post released datasets for our analysis

Pre-release data Table

num_critic_for_reviews	Actors_1 popularity	Actors_2 popularity	Budget

Table1

Post-release Data Table

Average Rating	Revenue	Cast total facebook likes	Num user for reviews	Imdb score	movie facebook likes'	Vote average	Vote count	Cast power	Ranking(class Lable)

Table 2

Data Loading

After cleaning the data obtained from the IMDB site we stored the data in a file format called, comma separated values. Our dataset contains headers in the first line followed by the actual data called features. The last column is the label which is the value we want to predict. Python's pandas library is used for reading the .csv data. Dataset has been used instead of Data Frame for loading the data into the model as dataset creates the input pipeline for loading the data which helps in avoiding the data loading time into CPU.

6 Model Construction

The first set in model construction is creation of training and testing data set. Training dataset is used for training the model using the classifiers and testing dataset is used to evaluate the results predicted by the model. In this project training dataset is further split into Train_x (Feature Attribute) and Train_y (Class Attribute). Also training dataset is accompanied by label for each tuple. For calculating the labels we considered three features

- Profit
- Voting Average
- Number of Critics Review

We added the value of any two labels and took the average of them. Whichever combinations had the greatest value was considered for determining the final label.

Testing dataset into Test_x and Test_y (actual label which the classifier predicts). After the training data is loaded it is converted to feature columns. Feature columns are very rich, enabling you to transform a diverse range of raw data into formats that estimators can use, allowing easy experimentation. **Estimators** are high-level TensorFlow API that greatly simplifies machine learning programming. Estimators encapsulate the following actions:

- Training
- Evaluation
- Prediction
- Export to Serving

Having Time Constraints in mind we had make use of TensorFlow's premade estimators in the project for prediction and evaluation.

Deep Neural Network(DNN) Classification model is used in the project for predicting the success of the movie. For optimization of learning rate of the classifier we have used ProximalAdagradOptimizer, which is one the type of gradient descent optimizer. Neural Network is composed of input layer, hidden layer and output layer. Input Layer contains node which hold the feature attribute values and output layer hold the value of class prediction. We have used three hidden layer with the tensor node of 1025, 512, 256,128 each. n_class is the attribute in DNN Classifiers which helps to give the number of possible values for the class label. When this n_class is set as two, the DNN Classifier becomes the binary classifier. Our model has value of 5 for this attribute. A Path which records the log is created for the loading the various event into system for further visualization and debugging purposes. Below is the training stage of the DNN Classifier

After the model definition step is performed, the model need to be tested with the data set. This kind of dataset is called as training dataset. Training is done using loss function and optimization function to improve the accuracy of the classifiers. Using the debugging facility and checkpoint created using the training steps, we can find the loss incurred by the model. This help to try using different number of hidden layer and the optimization functions available in the TensorFlow API.

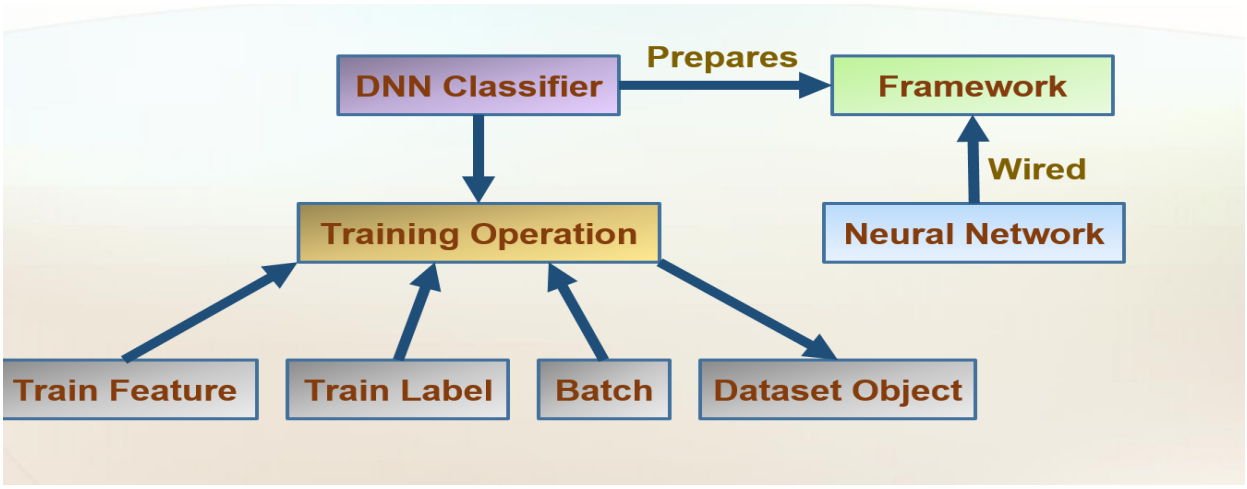

Fig 10

Next step after model training is model evaluation. Test data is used in this case instead of training data. At the end of this method overall accuracy for the testing data with respect to test data is found.

Now, the single tuple without the class label is sent for prediction into the DNN Classifier. The classifier return an integer value ranging from 1-5 based on the feature attribute values.
For the movie dataset, it's an integer value of 0, 1, 2, 3, 4 or 5 that corresponds to the class label values in the table 3.

TensorFlow has the in-built visualization tool called "TensorBoard". TensorBoard provides a tool to make easier for the developer to understand the model and optimization problem, debugging.

Target Class	**Range - Prediction Label**
1	Flop
2	Average
3	Hit
4	Super Hit
5	Block Buster

Table 3

Even it helps to visualize the TensorFlow graph and plot the quantitative metrics about the execution of graph and how the data pass through the layer in neural network.

7 Status of implementation

In terms of Implementation, the Movie Success Prediction have been completely implemented and tested with Tensorflow's predefined Estimator. We are comparing the test result with the other prediction models used in the Keras API. We have incorporated TensorBoard visualization tool for presenting the various metrics like Accuracy, Cross Entropy, Weights and bias flow between the hidden layers at certain training steps and presented the overall flow of project with the help of graph, showing important operations as the node and tensors as the dataflow.

8 Evaluation results

After the DNN Classifier is trained using the training data, the classifier is ready to predict the unknown class labels for the test inputs. After the prediction of test data, we need to finds its metrics in terms of accuracy and error rate. Below are some of the metrics which are used in the project for evaluating the results of the classifiers.

Evaluate metrics and plan

We should have some standards or metrics to evaluate models. We will use metrics such as the following:

1. Accuracy measures how often the classifier makes the correct prediction.
   Accuracy = The number of correct predictions / the total number of predictions

   Accuracy obtained from the DNN Classifier are **0.658** which can be further increased by adding more training set to the model for training. A screen shot from TensorBoard is shown in Figure 11
2. Error Rate measure the how often the classifier make the error in prediction.
   Error Rate = 1- Accuracy which is **.34**
3. Precision measures what proportion of movie we classified as success, actually were success.
   Precision = True success / (True success + False success)
4. Recall is what proportion of movie that actually were success were classified by us as success.
   Recall = True success / (True success + False not success)

The metrics are not limited just to the ones mentioned. We will consider some other factors in the project.

Screenshot for Loss and average loss is shown in Figure 12 and Figure 13.

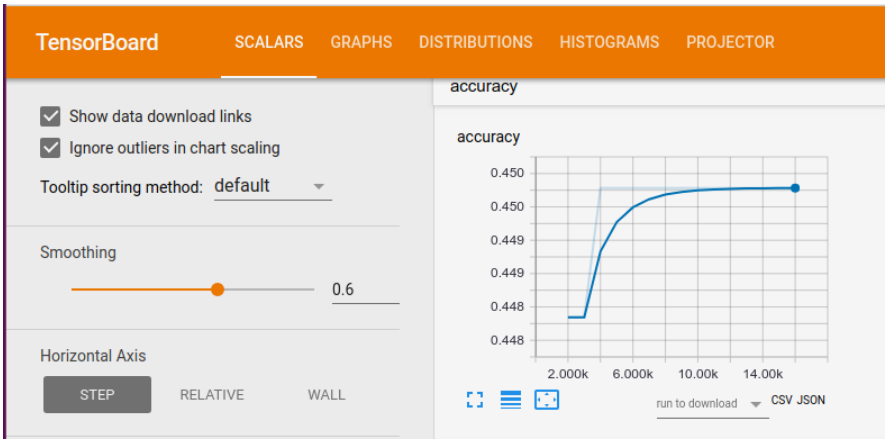

Fig 11

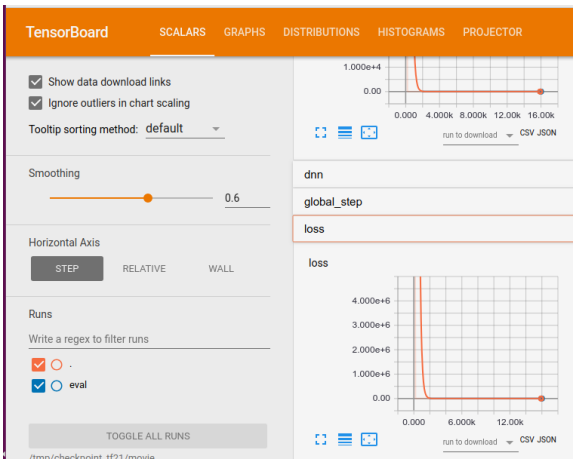
TensorBoard
SCALARS
GRAPHS
DISTRIBUTIONS
HISTOGRAMS
PROJECTOR
Show data download links
Ignore outliers in chart scaling
Tooltip sorting method: default
Smoothing
0.6
Horizontal Axis
STEP
RELATIVE
WALL
Runs
Write a regex to filter runs
eval
TOGGLE ALL RUNS
1.000e+4
0.00
0.000 4.000k 8.000k 12.00k 16.00k
run to download
CSV JSON
dnn
global_step
loss
loss
4.000e+6
3.000e+6
2.000e+6
1.000e+6
0.00
0.000 6.000k 12.00k
run to download
CSV JSON

Fig 12

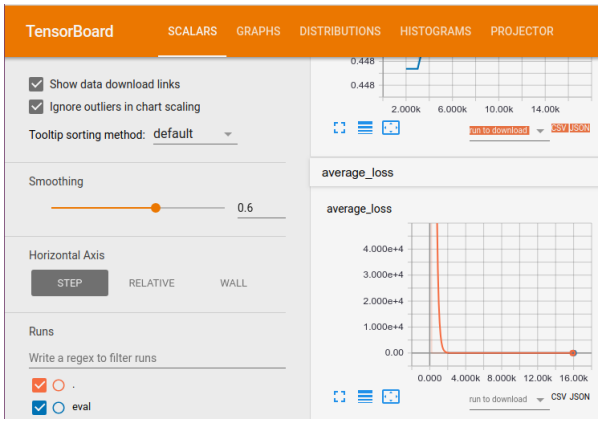
TensorBoard
SCALARS
GRAPHS
DISTRIBUTIONS
HISTOGRAMS
PROJECTOR
Show data download links
Ignore outliers in chart scaling
Tooltip sorting method: default
Smoothing
0.6
Horizontal Axis
STEP
RELATIVE
WALL
Runs
Write a regex to filter runs
eval
0.448
0.448
2.000k 6.000k 10.00k 14.00k
run to download
CSV JSON
average_loss
average_loss
4.000e+4
3.000e+4
2.000e+4
1.000e+4
0.00
0.000 4.000k 8.000k 12.00k 16.00k
run to download
CSV JSON

Fig 13

```
INFO:tensorflow:Loss for final step: 86582240.0.
INFO:tensorflow:Calling model_fn.
INFO:tensorflow:Done calling model_fn.
INFO:tensorflow:Starting evaluation at 2018-07-04-18:12:36
INFO:tensorflow:Graph was finalized.
INFO:tensorflow:Restoring parameters from C:/Users/ifthi/D
tion/graphs\model.ckpt-1000
INFO:tensorflow:Running local_init_op.
INFO:tensorflow:Done running local_init_op.
INFO:tensorflow:Finished evaluation at 2018-07-04-18:12:36
INFO:tensorflow:Saving dict for global step 1000: accuracy
= 222799.89, global_step = 1000, loss = 11474194.0

Test set accuracy: 0.650

INFO:tensorflow:Calling model_fn.
INFO:tensorflow:Done calling model_fn.
INFO:tensorflow:Graph was finalized.
INFO:tensorflow:Restoring parameters from C:/Users/ifthi/D
tion/graphs\model.ckpt-1000
INFO:tensorflow:Running local_init_op.
INFO:tensorflow:Done running local_init_op.

Label Predicted: 3

Prediction is "SuperHit" (100.0%), expected "SuperHit"
```

From the above figure, we can see when the expected test class label is “SuperHit”, the model which is developed predicts the same “SuperHit”. Training a large Deep Neural Network is very complex and confusing. To make it easier to understand, debug, and optimize, TensorFlow programs, we've included a visualization tools called TensorBoard. You can use TensorBoard to visualize your TensorFlow graph, plot quantitative metrics about the execution of your graph, and show additional data like images that pass through it.

TensorBoard runs by reading the even file generated during training state of the model. These even file summaries of data like loss, accuracy, histogram representation of weights and bias of node in each layer of neural network. In Figure14 you can see the 3D representation of tensor node in hidden layer 3 of the deep neural network. These metrics will further try to understand about the model and make DNN Classifiers predict more accurate results

9 Code

```
# package import
import os
import pandas as pd
import numpy as np
import tensorflow as tf
import tensorflow.contrib.learn
from __future__ import absolute_import
from __future__ import division
from __future__ import print_function
import argparse
import tensorflow as tf
import movie_data
PATH = "C:/Users/ifthi/Desktop/DataMining/moviePredition/graphs"
tf.logging.set_verbosity(tf.logging.INFO)
from tensorflow.contrib.learn.python.learn import monitors as monitor_lib
CSV_COLUMN_NAMES = ['num_critic_for_reviews','revenue','num_voted_users','cast_total_facebook_likes','num_user_for_reviews','budget','imdb_score','movie_facebook_likes','vote_average','vote_count','cast_power','ranking']
CLASSIFICATION = ['Flop','Average','Hit','SuperHit','BlockBuster']

CSV_TYPES = [[0], [0], [0], [0],[0],[0],[0.0],[0],[0],[0],[0],[0]]
# function to load the data from the csv and split into test and training set
def load_csv_data (y_name='ranking'):

    train = pd.read_csv("movie_training.csv", names=CSV_COLUMN_NAMES, header=0)
    train_x, train_y = train, train.pop(y_name)

    test = pd.read_csv("movie_test.csv", names=CSV_COLUMN_NAMES, header=0)
    test_x, test_y = test, test.pop(y_name)
    print("> 1. Loading the data in memory ...")

    return (train_x, train_y), (test_x, test_y)
# function which make the loaded csv data into Dataset optimized for TensorFlow operation
def train_input(features, labels, batch_size):
    """An input function for training"""
    # Convert the inputs to a Dataset.
    dataset = tf.data.Dataset.from_tensor_slices((dict(features), labels))

    # Shuffle, repeat, and batch the examples.
    dataset = dataset.shuffle(1000).repeat().batch(batch_size)

    # Return the dataset.
    return dataset
```

```
# fuction called during the classification evaluation based on test data
def evaluate_input(features, labels, batch_size):
    """An input function for evaluation or prediction"""
    features=dict(features)
    if labels is None:
        # No labels, use only features.
        inputs = features
    else:
        inputs = (features, labels)

    # Convert the inputs to a Dataset.
    dataset = tf.data.Dataset.from_tensor_slices(inputs)

    # Batch the examples
    assert batch_size is not None, "batch_size must not be None"
    dataset = dataset.batch(batch_size)

    # Return the dataset.
    return dataset
parser = argparse.ArgumentParser()
parser.add_argument('--batch_size', default=100, type=int, help='batch size')
parser.add_argument('--train_steps', default=1000, type=int, help='number of training steps')
import os
os.environ['TF_CPP_MIN_LOG_LEVEL'] = '2'
def main(argv):
    #args = parser.parse_args(argv[1:])
    # Fetch the data
    with tf.name_scope('input'):
        (train_x, train_y), (test_x, test_y) = movie_data.load_csv_data ()
    # Feature columns describe how to use the input.
    my_feature_columns = []
    for key in train_x.keys():
        my_feature_columns.append(tf.feature_column.numeric_column(key=key))
    # Launch the graph in a session.
    def variable_summaries(var):
        with tf.name_scope('summaries'):
            mean = tf.reduce_mean(var)
            tf.summary.scalar('mean', mean)
        with tf.name_scope('stddev'):
            stddev = tf.sqrt(tf.reduce_mean(tf.square(var - mean)))
            tf.summary.scalar('stddev', stddev)
            tf.summary.scalar('max', tf.reduce_max(var))
            tf.summary.scalar('min', tf.reduce_min(var))
            tf.summary.histogram('histogram', var)
    merged = tf.summary.merge_all()
    validation_metrics = {
```

```
    "accuracy":
        tf.contrib.learn.MetricSpec(
            metric_fn=tf.contrib.metrics.streaming_accuracy,
            prediction_key=tf.contrib.learn.PredictionKey.
            CLASSES),
    "precision":
        tf.contrib.learn.MetricSpec(
            metric_fn=tf.contrib.metrics.streaming_precision,
            prediction_key=tf.contrib.learn.PredictionKey.
            CLASSES),
    "recall":
        tf.contrib.learn.MetricSpec(
            metric_fn=tf.contrib.metrics.streaming_recall,
            prediction_key=tf.contrib.learn.PredictionKey.
            CLASSES)
    }
    list_of_monitors_and_hooks =
[tf.contrib.learn.monitors.ValidationMonitor(validation_metrics)]


    validation_monitor =
tf.contrib.learn.monitors.ValidationMonitor(test_x,test_y,every_n_steps=50,metrics=validation_
metrics,early_stopping_metric="loss",
    early_stopping_metric_minimize=True,
    early_stopping_rounds=200)
    classifier = tf.estimator.DNNClassifier(
        feature_columns=my_feature_columns,
        # Three hidden layers of 50,100,50,150 nodes each.
        hidden_units=[50,100,50,50],
        activation_fn=tf.nn.relu,
        optimizer=tf.train.ProximalAdagradOptimizer(
        learning_rate=0.001,
        l1_regularization_strength=0.01),
        config=tf.estimator.RunConfig().replace(save_summary_steps=10,
        # The model must choose between 5 classes.
        n_classes=5,
        dropout=0.1,
        model_dir=PATH)
    hooks = monitor_lib.replace_monitors_with_hooks(list_of_monitors_and_hooks, classifier)
    #classifier.fit(x=train_x,y=train_y,steps=2000,monitors=[validation_monitor])
    print("--> 3. Training the model using the test data ---> ")
    # Train the Model.
    with tf.name_scope('train'):
        classifier.train(
        input_fn=lambda:movie_data.train_input(train_x, train_y,
                                    100),steps=1000)
```

```
    # Evaluate the model.
    eval_result = classifier.evaluate(
        input_fn=lambda:movie_data.evaluate_input (test_x, test_y,
                                        100))
    print('\nTest set accuracy: {accuracy:0.3f}\n'.format(**eval_result))
    # Generate predictions from the model
    expected = ['SuperHit']
    predict_x={
    'num_critic_for_reviews': [184],
    'revenue': [59600000],
    'num_voted_users': [116642],
    'cast_total_facebook_likes': [1421],
    'num_user_for_reviews':[389],
    'budget': [950000],
    'imdb_score': [7.3],
    'movie_facebook_likes': [5000],
    'vote_average': [6.9],
    'vote_count': [953],
    'cast_power': [1194]
    }

    predictions = classifier.predict(
        input_fn=lambda:movie_data.eval_input_fn(predict_x,
                                        labels=None,
                                        batch_size=100))

    for pred_dict, expectation in zip(predictions, expected):
        template = ('\nPrediction is "{}" ({:.1f}%), expected "{}"')

        class_id = pred_dict['class_ids'][0]
        print ("\nLabel Predicted:",class_id)
        probability = pred_dict['probabilities'][class_id]

        print(template.format(movie_data.CLASSIFICATION[class_id],
                              100 * probability, expectation))
```

 # function which is used to run the main function with the logging level as INFO

```
 if __name__ == '__main__':
    tf.logging.set_verbosity(tf.logging.INFO)
    tf.app.run(
    main=main,
    argv=None
)
```

TENSORBOARD COMMANDS:

tensorboard –logdir= ”path to the log directory”

10 Conclusion

Prediction movie success is always an area of improvement when using the machine learning algorithms. As various classification algorithms are developed and published often, choosing the correct model for our need makes more sense in terms of more accuracy rate and less error. Using supervised learning method like DNN Classifiers in his project helps to successfully predict the whether the movie is Hit, Flop, Blockbuster, Super Hit, Hit based on test attributes like number of vote, Facebook likes, Actors like, IMDB Score and some more. Basically, predicting actual Super Hit movie as Hit Movie can be considered as not a bad model as it is close to be a correct prediction.

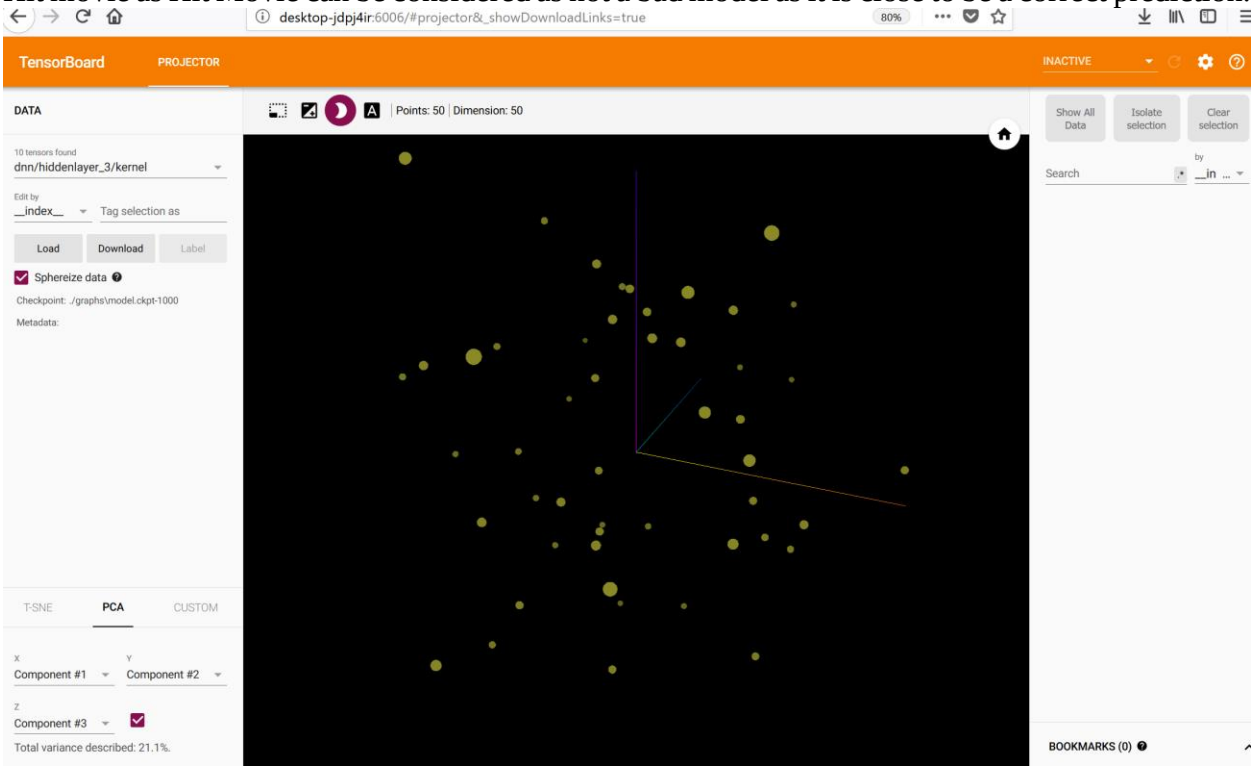

Fig 14

But our implemented DNN classifier for movie prediction is able to make the most exact prediction after training with good dataset.

11 Future work

Some attribute can be further added to the training data set using sentimental analysis in getting position or negative comment from the users. and recommend the movies which are similar in genre, budget that made a huge Box Office success in the past. We will be working on to resolve one classifier error which will further make the movie success prediction classifier more accurate and try to implement our TensorFlow backed application to run under Graphical Processing Unit.

12 Responsibility of Team Members

Everyone in the team are discussing and working on the development of this project.
Collecting data – every member
Preprocessing data – Xiaodi Fu, Preetham Wilfred John, Mohammed Ifthikar M (Roshan, Ankita helped in cleaning the data).
Selecting algorithm – Preetham Wilfred John
Selecting Model – Ankita Puranik
Executing Model – Roshan Ghale
Evaluating Model – Mohammed Ifthikar M

13 Reference:

[
1] Quader, N., Gani, M.O., and Chaki, D.: 'Performance evaluation of seven machine learning classification techniques for movie box office success prediction', in Editor (Ed.)^(Eds.): 'Book Performance evaluation of seven machine learning classification techniques for movie box office success prediction' (2017, edn.), pp. 1-6
[2] A. Sivasantoshreddy, P.K., and A. Jain: 'Box-Office Opening Prediction of Movies based on Hype Analysis through Data Mining', International Journal of Computer Applications, 2012, vol 56, no. 1, pp. 1-5, 201
[3] Martín Abadi, P.B., Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, Manjunath Kudlur, Josh Levenberg, Rajat Monga, Sherry Moore, Derek G. Murray, Benoit Steiner, Paul Tucker, Vijay Vasudevan, Pete arden, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng 'TensorFlow: A System for Large-Scale Machine Learning'. Proc. 12th USENIX Symposium on Operating Systems Design and Implementation, Savannah, GA, USA, November 2–4, 2016

14 GitHub Link

https://github.com/preethamwilfredjohn/MoviePrediction.git