Movie Recommender System Analysis

INFO7390 Final Project

Bo Cao, NUID: 001834167
Boyang Wei, NUID: 001883667

2018 Fall Semester

Abstract

The Movie Database (TMDb) is a community-built movie and TV database. Every piece of data has been added by our amazing community dating back to 2008. TMDb's strong international focus and breadth of data is unmatched and something we're incredibly proud of.

Our project will involve with a measurement on how much the things like budget, revenue, popularity and other factors determines the final vote score and if the movie is popular based on the score. Then deep learning model will be implemented by several steps, about 4,000 records of movies will be reshaped in order to fit in the convolution. CNN is a widely used model to image recognition. How the CNN could perform good in movie data as well will be explained in the following part.

In our project, Recommender System will be implemented to design a model used to recommend movies to users based on the estimate deep learning result. Also, we want to get know about the popular trend of the movie, based on the movie data.

Introduction

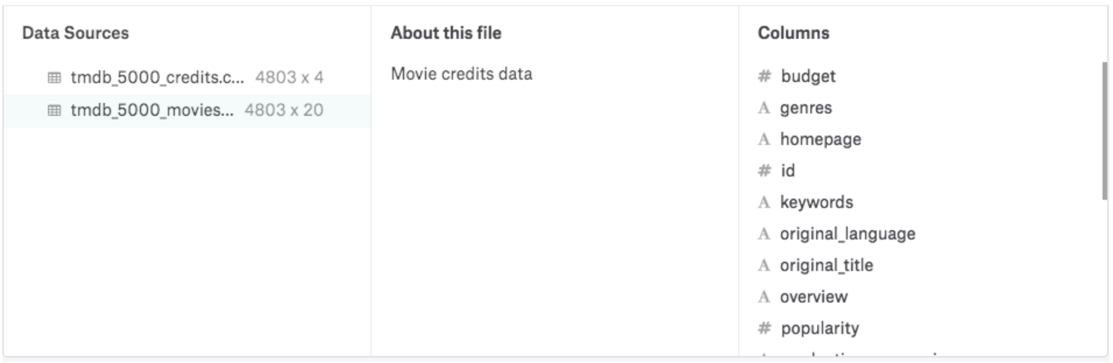
The Movie Database (TMDb) is a community-built movie and TV database. Every piece of data has been added by our amazing community dating back to 2008. TMDb's strong international focus and breadth of data is unmatched and something we're incredibly proud of.

What can we say about the success of a movie before it is released? Are there certain companies (Pixar?) that have found a consistent formula? Given that major films costing over $100 million to produce can still flop, this question is more important than ever to the industry. Film aficionados might have different interests. Can we predict which films will be highly rated, whether they are a commercial success?

This is a great topic to start digging in to those questions, with data on the plot, cast, crew, budget, and revenues of several thousand films.

I. Data Source

https://www.kaggle.com/tmdb/tmdb-movie-metadata



Data Sources

- tmdb_5000_credits.c... 4803 x 4
- tmdb_5000_movies... 4803 x 20

About this file

Movie credits data

Columns

- # budget
- A genres
- A homepage
- # id
- A keywords
- A original_language
- A original_title
- A overview
- # popularity

This collection includes approximately 5,000 of movie records.

Code with Documentation

Link to Source Code:

https://github.com/haipiao/INFO7390

https://github.com/BoyangWei/INFO_7390

Methods & Results

A. Multiple Linear Regression

I. Introduction

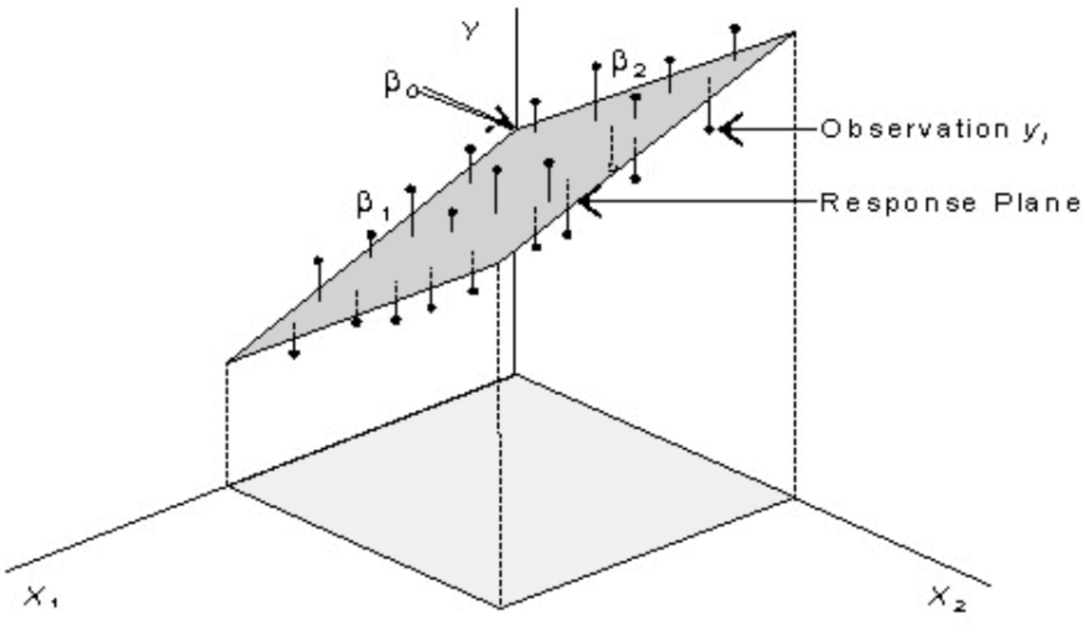
Multiple linear regression is the extension of a single predictor variable x to a set predictor variables, $\{x1, x2 \cdots xn,\}$, that is

$$Y = \beta_1 x_1 + \beta_2 x_2 + \beta_3 x_3 + \beta_n x_n + \varepsilon$$

these n equations are stacked together and written in vector form as

$$Y = \beta X + \varepsilon$$

Where Y, beta, and ε are vectors and X is a matrix (sometimes called the design matrix).

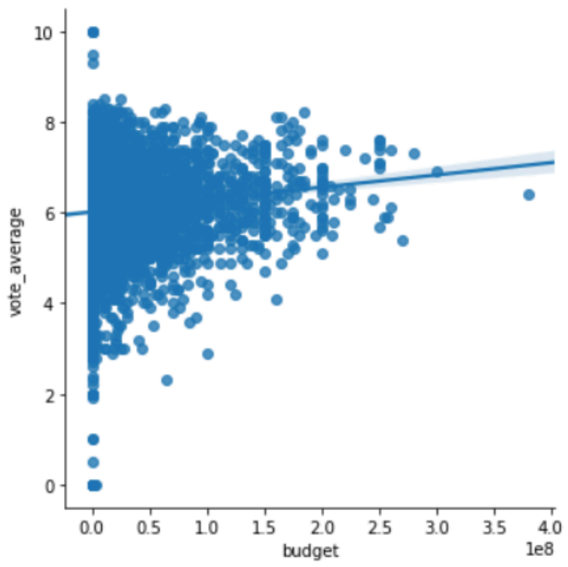


II. Result

(a) Linear Regression diagram for budget and vote_average.

```
sns.lmplot(x="budget", y="vote_average", data=movies)
```

```
<seaborn.axisgrid.FacetGrid at 0x1c0be1b7f0>
```



(2) Linear Regression for vote_average and budget, popularity, revenue.

```
# vote_average between budget, popularity and revenue
delay_model = ols("vote_average ~ budget + popularity + revenue  -1",data = movies).fit()
delay_model.summary()
```

OLS Regression Results

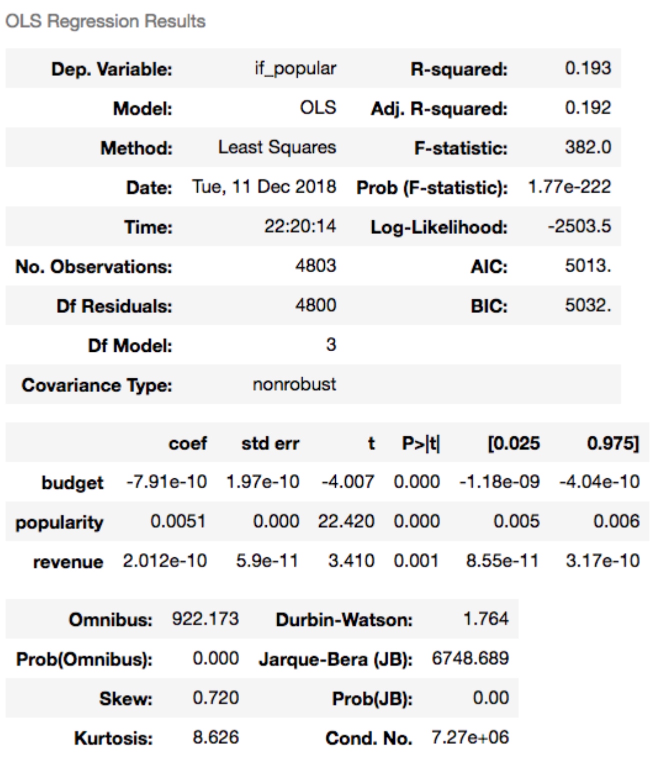
Dep. Variable:	vote_average	**R-squared:**	0.429
Model:	OLS	**Adj. R-squared:**	0.428
Method:	Least Squares	**F-statistic:**	1200.
Date:	Tue, 11 Dec 2018	**Prob (F-statistic):**	0.00
Time:	22:20:14	**Log-Likelihood:**	-14241.
No. Observations:	4803	**AIC:**	2.849e+04
Df Residuals:	4800	**BIC:**	2.851e+04
Df Model:	3		
Covariance Type:	nonrobust		

	coef	**std err**	**t**	**P>|t|**	**[0.025**	**0.975]**
budget	5.621e-08	2.27e-09	24.726	0.000	5.18e-08	6.07e-08
popularity	0.0711	0.003	26.959	0.000	0.066	0.076
revenue	-6.923e-09	6.79e-10	-10.189	0.000	-8.26e-09	-5.59e-09

Omnibus:	4487.236	**Durbin-Watson:**	0.338
Prob(Omnibus):	0.000	**Jarque-Bera (JB):**	446953.178
Skew:	-4.145	**Prob(JB):**	0.00
Kurtosis:	49.526	**Cond. No.**	7.27e+06

(3) Linear Regression for if_popular and budget, popularity, revenue.

```
# if_popular between budget, popularity and revenue

delay_model = ols("if_popular ~ budget + popularity + revenue  -1",data = movies).fit()

delay_model.summary()
```



OLS Regression Results

Dep. Variable:	if_popular	**R-squared:**	0.193
Model:	OLS	**Adj. R-squared:**	0.192
Method:	Least Squares	**F-statistic:**	382.0
Date:	Tue, 11 Dec 2018	**Prob (F-statistic):**	1.77e-222
Time:	22:20:14	**Log-Likelihood:**	-2503.5
No. Observations:	4803	**AIC:**	5013.
Df Residuals:	4800	**BIC:**	5032.
Df Model:	3		
Covariance Type:	nonrobust		

	coef	**std err**	**t**	**P>|t|**	**[0.025**	**0.975]**
budget	-7.91e-10	1.97e-10	-4.007	0.000	-1.18e-09	-4.04e-10
popularity	0.0051	0.000	22.420	0.000	0.005	0.006
revenue	2.012e-10	5.9e-11	3.410	0.001	8.55e-11	3.17e-10

Omnibus:	922.173	**Durbin-Watson:**	1.764
Prob(Omnibus):	0.000	**Jarque-Bera (JB):**	6748.689
Skew:	0.720	**Prob(JB):**	0.00
Kurtosis:	8.626	**Cond. No.**	7.27e+06

III. Conclusion

We can find there are linear relationship between budget and vote_average. As for multiple linear regression, there are linear relationship between vote_average and the combine of budget, popularity and revenue.

B. Logistic Linear Regression

I. Introduction

Logistic regression, or logit regression, or is a regression model where the outcome variable is categorical. Often this is used when the variable is binary. Logistic regression measures the relationship between the categorical response variable and one or more predictor variables by estimating probabilities.

Logistic regression: log-odds of a categorical response being "true" (1) is modeled as a linear combination of the features:

$$\ln\left(\frac{p}{1}-p\right)=\beta_0+\beta_1 x$$

This is called the **logit function**.
Probability is sometimes written as:

$$\ln\left(\frac{\pi}{1}-\pi\right)=\beta_0+\beta_1 x$$

The equation can be rearranged into the **logistic function**:

$$\pi=\frac{e^{\beta_0+\beta_1 x}}{1+e^{\beta_0+\beta_1 x}}$$

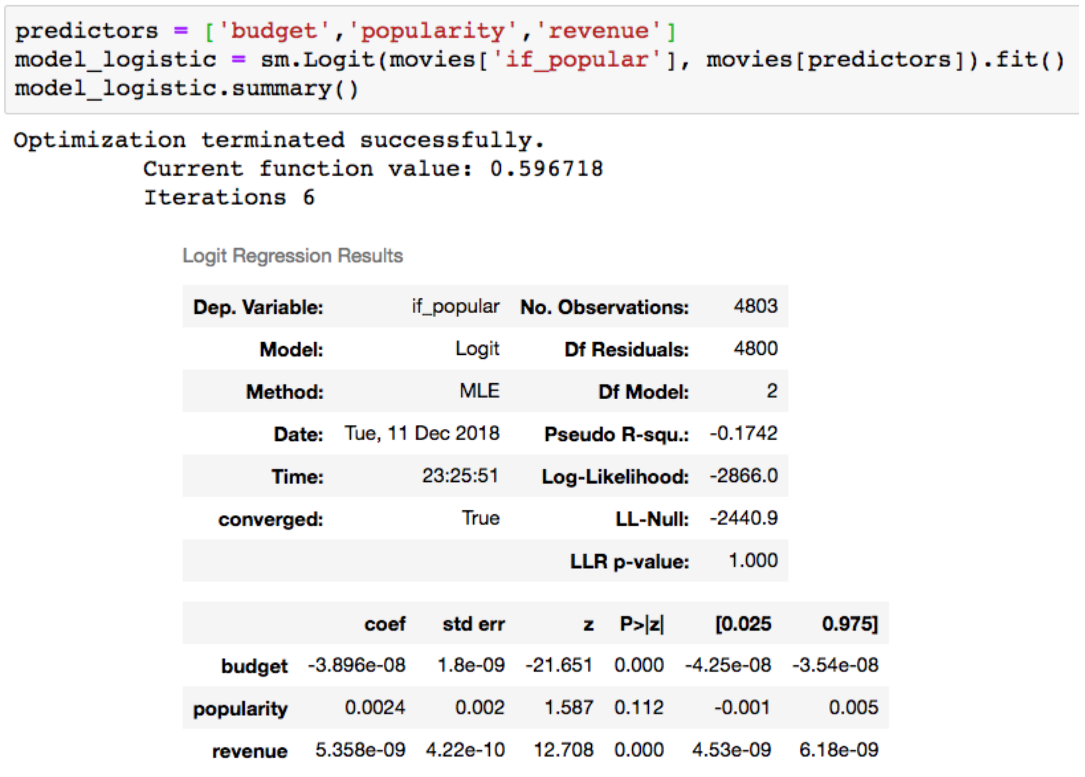
- Logistic regression outputs the **probabilities of a specific class**
- Those probabilities can be converted into **class predictions**

The **logistic function** has some nice properties:

- Takes on an "s" shape
- Output is bounded by 0 and 1 (Probability is also bounded by 0 and 1)

II. Result

Logistic regression for if_popular and budget, popularity, revenue.



```
predictors = ['budget','popularity','revenue']
model_logistic = sm.Logit(movies['if_popular'], movies[predictors]).fit()
model_logistic.summary()
```

```
Optimization terminated successfully.
         Current function value: 0.596718
         Iterations 6
```

Logit Regression Results

Dep. Variable:	if_popular	**No. Observations:**	4803
Model:	Logit	**Df Residuals:**	4800
Method:	MLE	**Df Model:**	2
Date:	Tue, 11 Dec 2018	**Pseudo R-squ.:**	-0.1742
Time:	23:25:51	**Log-Likelihood:**	-2866.0
converged:	True	**LL-Null:**	-2440.9
		LLR p-value:	1.000

	coef	**std err**	**z**	**P>|z|**	**[0.025**	**0.975]**
budget	-3.896e-08	1.8e-09	-21.651	0.000	-4.25e-08	-3.54e-08
popularity	0.0024	0.002	1.587	0.112	-0.001	0.005
revenue	5.358e-09	4.22e-10	12.708	0.000	4.53e-09	6.18e-09

III. Conclusion

There is relationship between if_popular and budget and revenue, as |z|>2.58, which means the result is significant. However, as for popularity, it is not significant because its |z|<1.96.

C. Clustering

I. Introduction

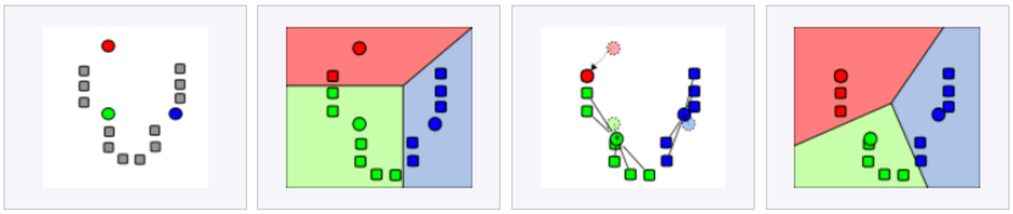
Clustering is grouping like with like such that:

- Similar objects are close to one another within the same cluster.
- Dissimilar to the objects in other clusters.

There are two primary approaches to measure the "closeness" of data, distance and similarity. Distance measures are based in the notion of a metric space. Similarity measures is a real-valued function that quantifies the similarity between two objects. In our project, we use K-means to do cluster analysis.

k-means clustering is a method of vector quantization, originally from signal processing, that is popular for cluster analysis in data mining. k-means clustering aims to partition n observations into k clusters in which each observation belongs to the cluster with the nearest mean, serving as a prototype of the cluster. This results in a partitioning of the data space into Voronoi cells.

Demonstration of the standard algorithm



1. *k* initial "means" (in this case *k*=3) are randomly generated within the data domain (shown in color).

2. *k* clusters are created by associating every observation with the nearest mean. The partitions here represent the Voronoi diagram generated by the means.

3. The centroid of each of the *k* clusters becomes the new mean.

4. Steps 2 and 3 are repeated until convergence has been reached.

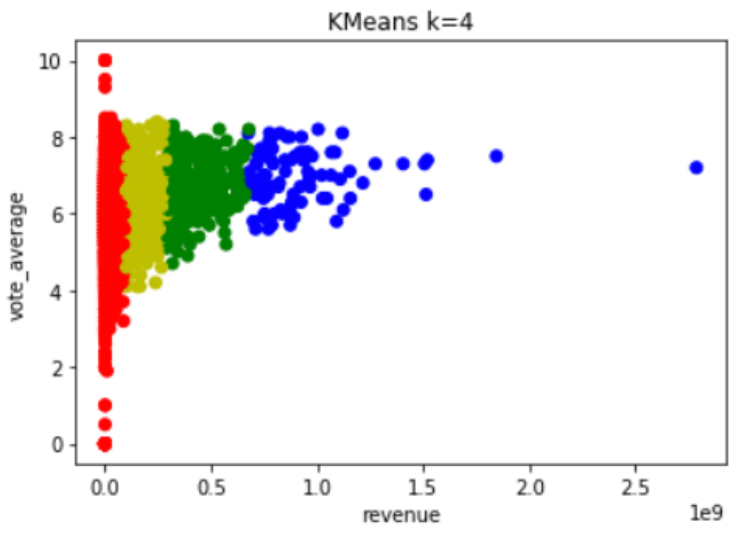
II. Result

Fit a K-means estimator (k=4):

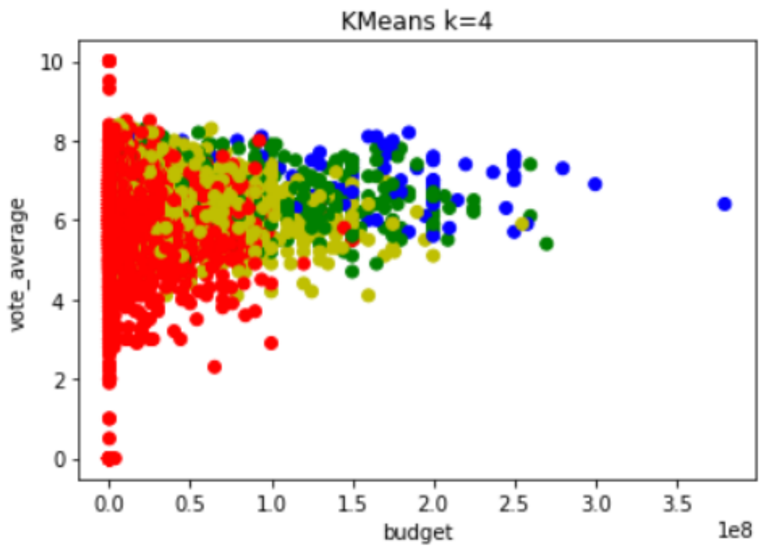
```
#### Fit a k-means estimator
X = np.array(movies)
estimator = KMeans(n_clusters = 4)
estimator.fit(X)

# Clusters are given in the labels_ attribute
labels = estimator.labels_
print (labels)
```

```
[3 3 3 ... 0 0 0]
```

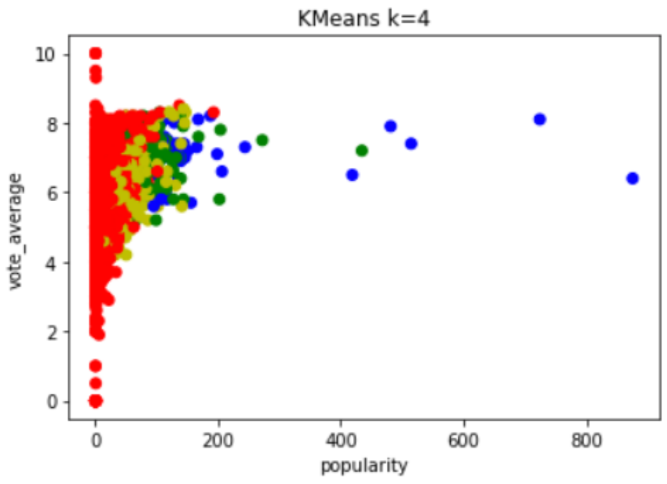
(1) K-means clustering for vote_average and revenue.



(2) K-means clustering for vote_average and budget.



(3) K-means clustering for vote_average and popularity.



III. Conclusion

We use K-means method to get the clusters. As for part (1), we can see that there are four clusters in revenue-vote_average axis plot. We know that the more revenue, the more chance of high vote score movie will be. As for budget-vote_average axis plot, the more budget input, the more chance a high score movie will be. As for popularity-vote_average axis plot, we cannot get a clear conclusion.

D. Decision Tree & Random Forest

I. Introduction

(1) Decision Tree

A decision tree is a supervised learning algorithm that uses a tree-like graph or model of decisions and their outcomes. The decision tree can be linearized into decision rules, where the outcome is the contents of the leaf node, and the conditions along the path form a conjunction in the if clause. In general, the rules have the form:

$$if \quad condition1 \quad and \quad condition2 \quad and \quad condition3 \quad then \quad outcome$$

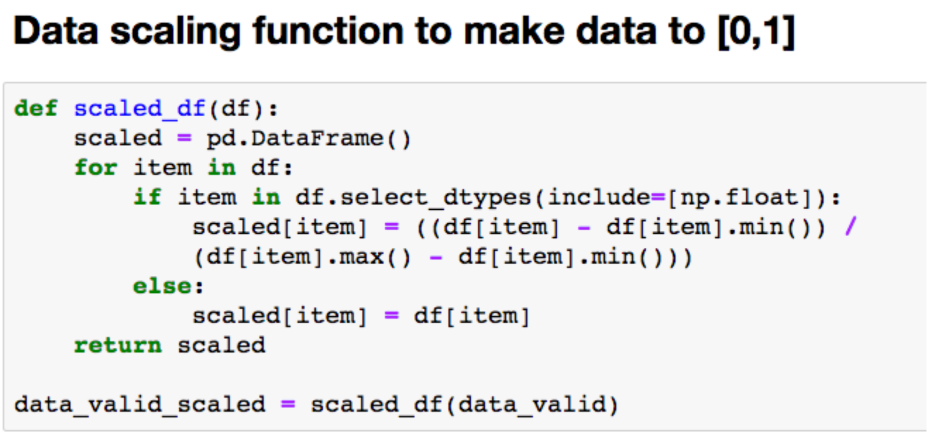
Each node in the tree is a decisions/tests. Each path from the tree root to a leaf corresponds to a conjunction of attribute decisions/tests. The tree itself corresponds to a disjunction of these conjunctions.

(2) Random Forest

Random forests are an ensemble learning method for classification, regression and other tasks, that operate by constructing a multitude of decision trees at training time and outputting the class that is the mode of the classes (classification) or mean prediction (regression) of the individual trees. Random decision forests correct for decision trees' habit of overfitting to their training set.

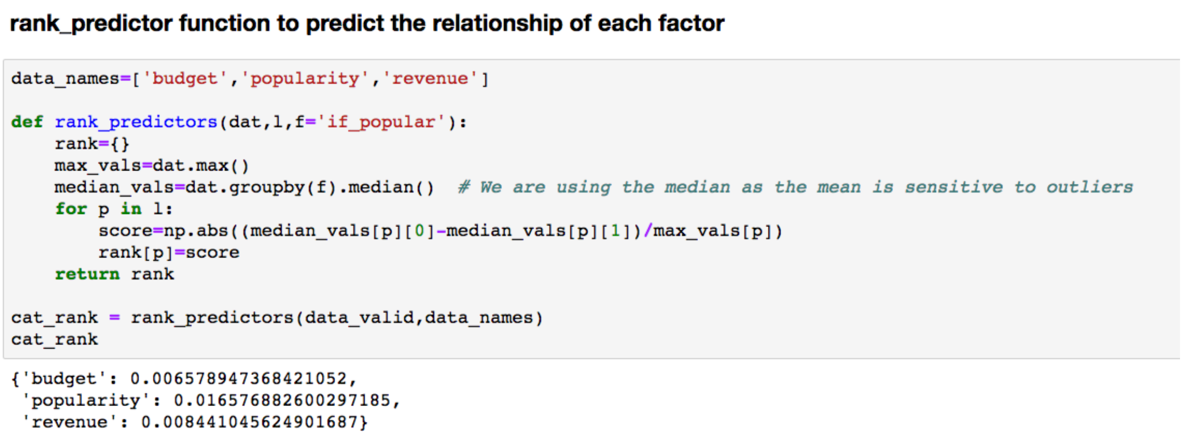
II. Result

(1) Decision Tree



Data scaling function to make data to [0,1]

```
def scaled_df(df):
    scaled = pd.DataFrame()
    for item in df:
        if item in df.select_dtypes(include=[np.float]):
            scaled[item] = ((df[item] - df[item].min()) /
            (df[item].max() - df[item].min()))
        else:
            scaled[item] = df[item]
    return scaled

data_valid_scaled = scaled_df(data_valid)
```



rank_predictor function to predict the relationship of each factor

```
data_names=['budget','popularity','revenue']

def rank_predictors(dat,l,f='if_popular'):
    rank={}
    max_vals=dat.max()
    median_vals=dat.groupby(f).median()  # We are using the median as the mean is sensitive to outliers
    for p in l:
        score=np.abs((median_vals[p][0]-median_vals[p][1])/max_vals[p])
        rank[p]=score
    return rank

cat_rank = rank_predictors(data_valid,data_names)
cat_rank
```

```
{'budget': 0.006578947368421052,
 'popularity': 0.016576882600297185,
 'revenue': 0.008441045624901687}
```

```
# Decision Tree classifier

DTm = DecisionTreeClassifier(max_depth=5,random_state=0,min_samples_split=2,min_samples_leaf=2)

# Decision Tree cross validation

print("KfoldCrossVal mean score using Decision Tree is %s" %cross_val_score(DTm,X,y,cv=10).mean())

# Decision Tree metrics
sm = DTm.fit(X_train, y_train)

y_pred = sm.predict(X_test)
print("Accuracy score using Decision Tree is %s" %metrics.accuracy_score(y_test, y_pred))
```

```
KfoldCrossVal mean score using Decision Tree is 0.7888778542875621
Accuracy score using Decision Tree is 0.8126951092611863
```

(2) Random Forest

```
# Random Forest classifier

RFm = RandomForestClassifier( random_state = 20,
                              criterion='gini',
                              n_estimators = 40,
                              min_samples_split = 2,
                              min_samples_leaf = 3)

# Random Forest cross validation

print("KfoldCrossVal mean score using Random Forest is %s" %cross_val_score(RFm,X,y,cv=10).mean())

# Random Forest metrics
sm = RFm.fit(X_train, y_train)

y_pred = sm.predict(X_test)
print("Accuracy score using Random Forest is %s" %metrics.accuracy_score(y_test, y_pred))
```

```
KfoldCrossVal mean score using Random Forest is 0.7736751389256609
Accuracy score using Random Forest is 0.8231009365244537
```

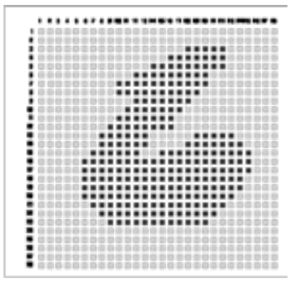
III. Conclusion

As for Decision Tree, its accuracy is 81.3%. As for Random Forest, its accuracy is 82.3%.

E. Convolution Neural Network

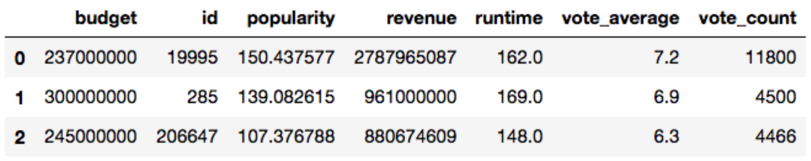
I. Introduction

The reason We chose CNN model to do deep learning in our data is that there do have similarity between an image(which is CNN usually applied to) and game data.

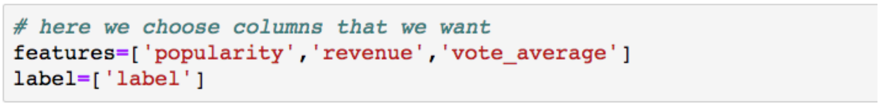


In MNIST data, a picture including a hand-written number is represent in 28*28 pixel, 1 and 0 represent the black and white, so there are totally 28*28*2 in one piece of data. If we apply traditional neural network on this, we got 784 input numbers and times of parameter in the first layer. Convolution and max pool oversee make the number of inputs decreased, but also guarantee the simplified input can keep the features of the original picture. As the game data always got a complicated dimension of data. CNN could be applied to movie data after the movie data is reshaped.

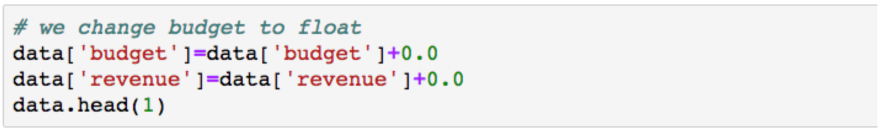
II. Result



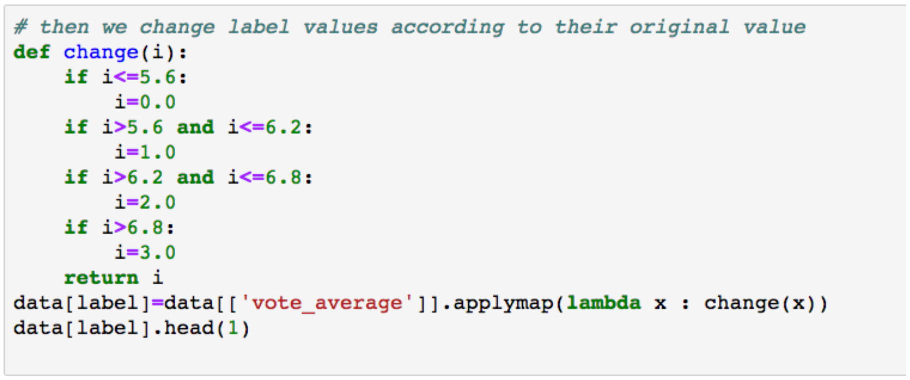
	budget	id	popularity	revenue	runtime	vote_average	vote_count
0	237000000	19995	150.437577	2787965087	162.0	7.2	11800
1	300000000	285	139.082615	961000000	169.0	6.9	4500
2	245000000	206647	107.376788	880674609	148.0	6.3	4466



```
# here we choose columns that we want
features=['popularity','revenue','vote_average']
label=['label']
```

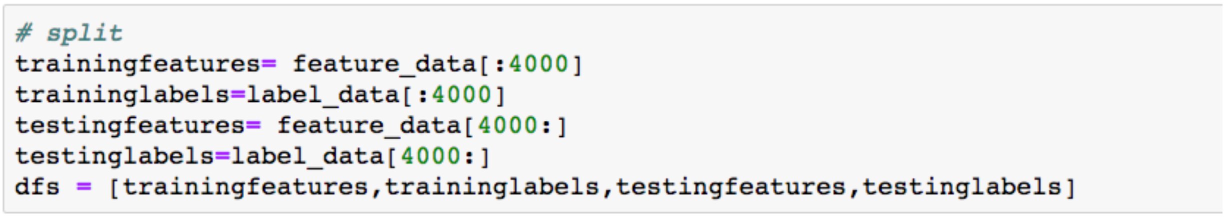


```
# we change budget to float
data['budget']=data['budget']+0.0
data['revenue']=data['revenue']+0.0
data.head(1)
```



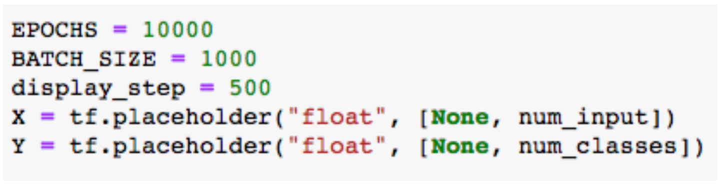
```
# then we change label values according to their original value
def change(i):
    if i<=5.6:
        i=0.0
    if i>5.6 and i<=6.2:
        i=1.0
    if i>6.2 and i<=6.8:
        i=2.0
    if i>6.8:
        i=3.0
    return i
data[label]=data[['vote_average']].applymap(lambda x : change(x))
data[label].head(1)
```

Split the data into training data and testing data.



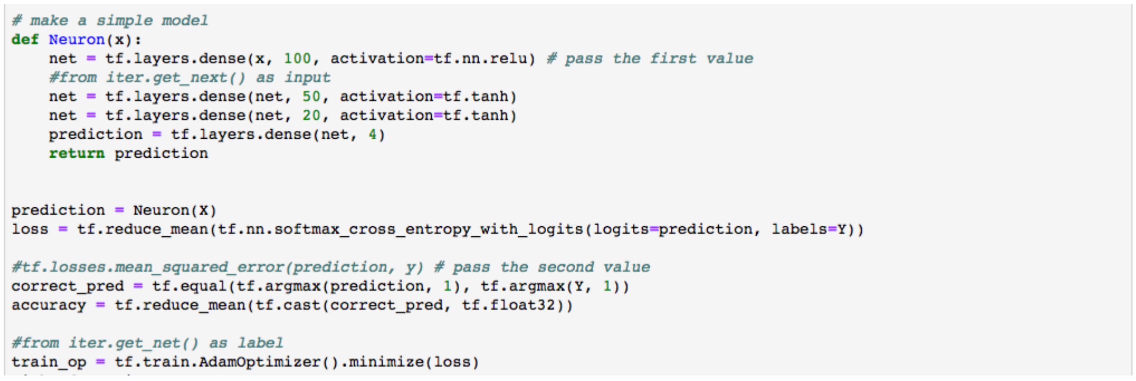
```
# split
trainingfeatures= feature_data[:4000]
traininglabels=label_data[:4000]
testingfeatures= feature_data[4000:]
testinglabels=label_data[4000:]
dfs = [trainingfeatures,traininglabels,testingfeatures,testinglabels]
```

Setting parameters.



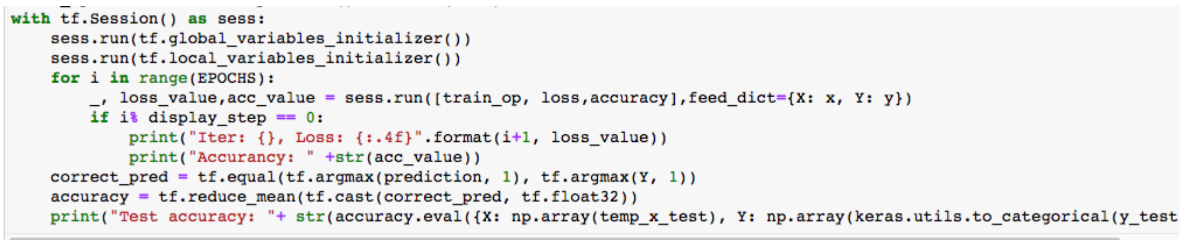
```
EPOCHS = 10000
BATCH_SIZE = 1000
display_step = 500
X = tf.placeholder("float", [None, num_input])
Y = tf.placeholder("float", [None, num_classes])
```

Creating model, loss function, Gradient estimation

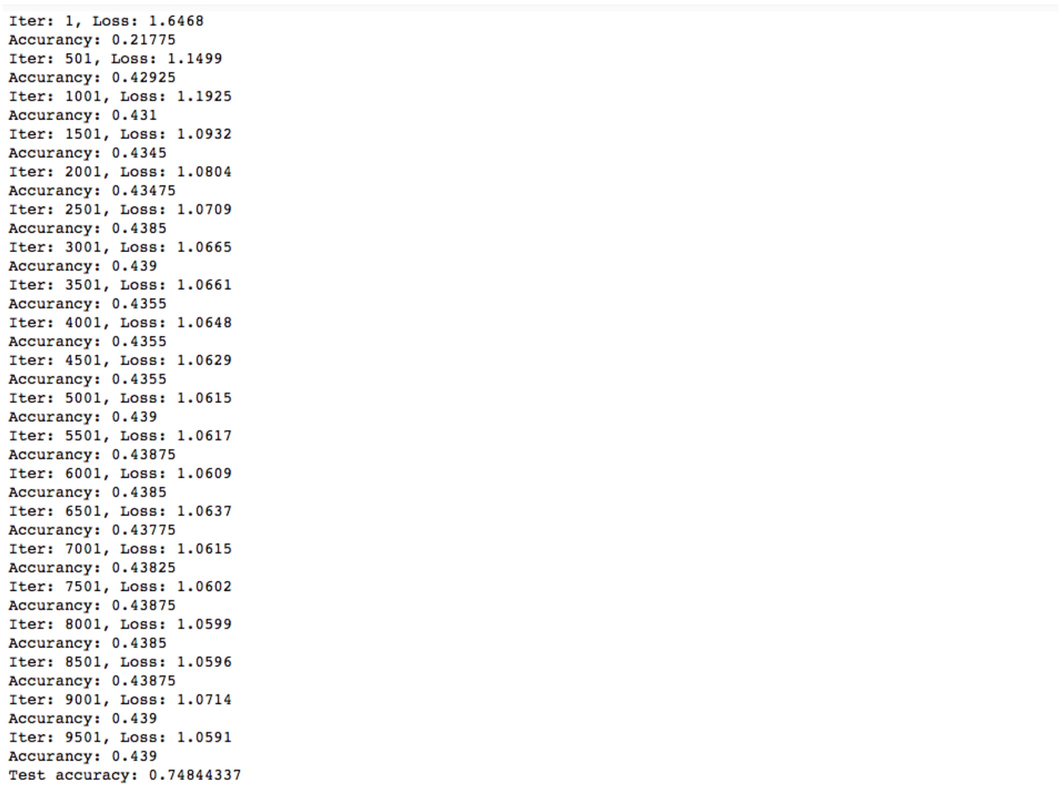


```
# make a simple model
def Neuron(x):
    net = tf.layers.dense(x, 100, activation=tf.nn.relu) # pass the first value
    #from iter.get_next() as input
    net = tf.layers.dense(net, 50, activation=tf.tanh)
    net = tf.layers.dense(net, 20, activation=tf.tanh)
    prediction = tf.layers.dense(net, 4)
    return prediction


prediction = Neuron(X)
loss = tf.reduce_mean(tf.nn.softmax_cross_entropy_with_logits(logits=prediction, labels=Y))

#tf.losses.mean_squared_error(prediction, y) # pass the second value
correct_pred = tf.equal(tf.argmax(prediction, 1), tf.argmax(Y, 1))
accuracy = tf.reduce_mean(tf.cast(correct_pred, tf.float32))

#from iter.get_net() as label
train_op = tf.train.AdamOptimizer().minimize(loss)
```

Then start training, and get result:



```
with tf.Session() as sess:
    sess.run(tf.global_variables_initializer())
    sess.run(tf.local_variables_initializer())
    for i in range(EPOCHS):
        _, loss_value,acc_value = sess.run([train_op, loss,accuracy],feed_dict={X: x, Y: y})
        if i% display_step == 0:
            print("Iter: {}, Loss: {:.4f}".format(i+1, loss_value))
            print("Accurancy: " +str(acc_value))
    correct_pred = tf.equal(tf.argmax(prediction, 1), tf.argmax(Y, 1))
    accuracy = tf.reduce_mean(tf.cast(correct_pred, tf.float32))
    print("Test accuracy: "+ str(accuracy.eval({X: np.array(temp_x_test), Y: np.array(keras.utils.to_categorical(y_test
```



```
Iter: 1, Loss: 1.6468
Accurancy: 0.21775
Iter: 501, Loss: 1.1499
Accurancy: 0.42925
Iter: 1001, Loss: 1.1925
Accurancy: 0.431
Iter: 1501, Loss: 1.0932
Accurancy: 0.4345
Iter: 2001, Loss: 1.0804
Accurancy: 0.43475
Iter: 2501, Loss: 1.0709
Accurancy: 0.4385
Iter: 3001, Loss: 1.0665
Accurancy: 0.439
Iter: 3501, Loss: 1.0661
Accurancy: 0.4355
Iter: 4001, Loss: 1.0648
Accurancy: 0.4355
Iter: 4501, Loss: 1.0629
Accurancy: 0.4355
Iter: 5001, Loss: 1.0615
Accurancy: 0.439
Iter: 5501, Loss: 1.0617
Accurancy: 0.43875
Iter: 6001, Loss: 1.0609
Accurancy: 0.4385
Iter: 6501, Loss: 1.0637
Accurancy: 0.43775
Iter: 7001, Loss: 1.0615
Accurancy: 0.43825
Iter: 7501, Loss: 1.0602
Accurancy: 0.43875
Iter: 8001, Loss: 1.0599
Accurancy: 0.4385
Iter: 8501, Loss: 1.0596
Accurancy: 0.43875
Iter: 9001, Loss: 1.0714
Accurancy: 0.439
Iter: 9501, Loss: 1.0591
Accurancy: 0.439
Test accuracy: 0.74844337
```

III. Conclusion

The accuracy is 43.9% for the model, 74.8% for the train data. We can also clearly see the loss is getting decreased during iteration.

F. Recommender System

I. Introduction

A recommender system or a recommendation system is an information filtering system that seeks to predict the "rating" or "preference" a user would give to an item. Recommender systems do two things: 1) Predict the "rating" a user would give to an item. 2) Predict a list of items a user would like. In our project, Content Based Recommenders are used.

Content Based Recommenders

- Measure "how close" items are
- Use content attributes as features
- There are many ways to measure the similarity
- The same items can produce different recommendations

Content-based recommenders measure "how close" items are and suggest similar items based on content features. This can be used to find items similar to an item. The same items can produce different recommendations depending on which attributes are chosen and how similarity is measured.

II. Approach & Result

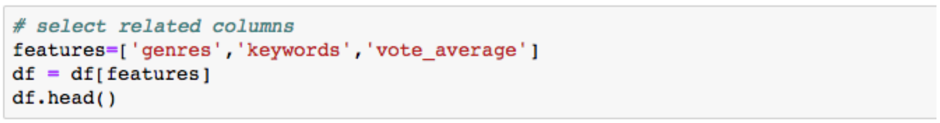
General approach

- First thing we need to do is defining some useful class since we are going deal objects. For this reason, we defined movie, person and recommend system three classes and their behavior.
- Secondly,the training mode definition. At beginning, system knows nothing about target people, thus it recommend high vote movies. Every recommendation results in a feedback dictionary which contains target reactions. Then system learn from such feedback by

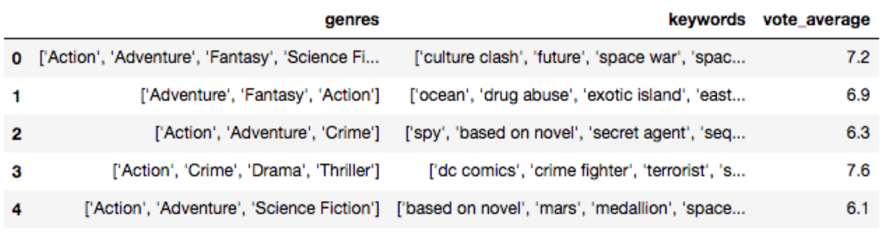
updating target's preference dictionary. Last step is so called personal recommendation. System pick most target's most preferred genres and recommend accordingly.

- Last but same important, how shall targets react on recommendations. In other word, simulation of targets' behavior. It is really complicated in the real world. Here, we simplified it as following: targets react on recommendations based on his own preference. We calculated an average preference for each recommended movie. And if it's higher than 0.6, target gives a thumb up.

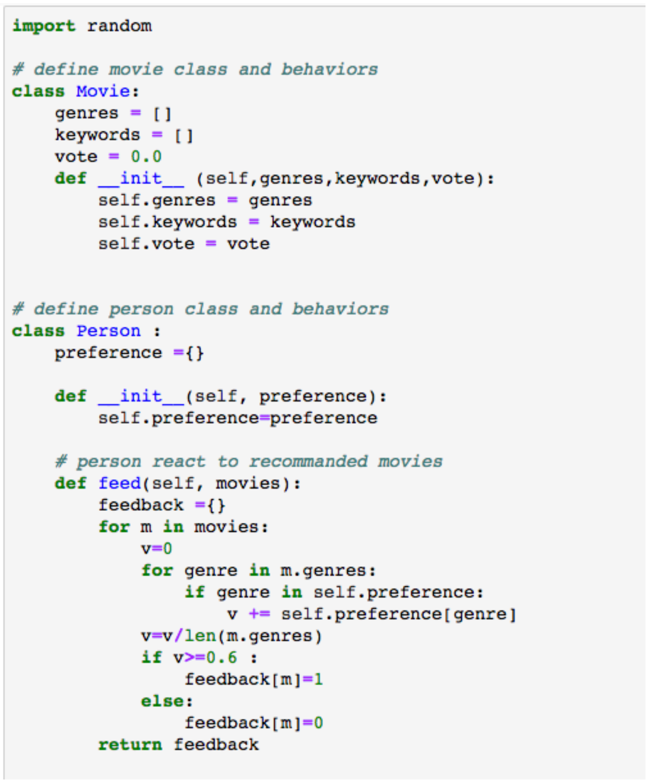
Select related columns.



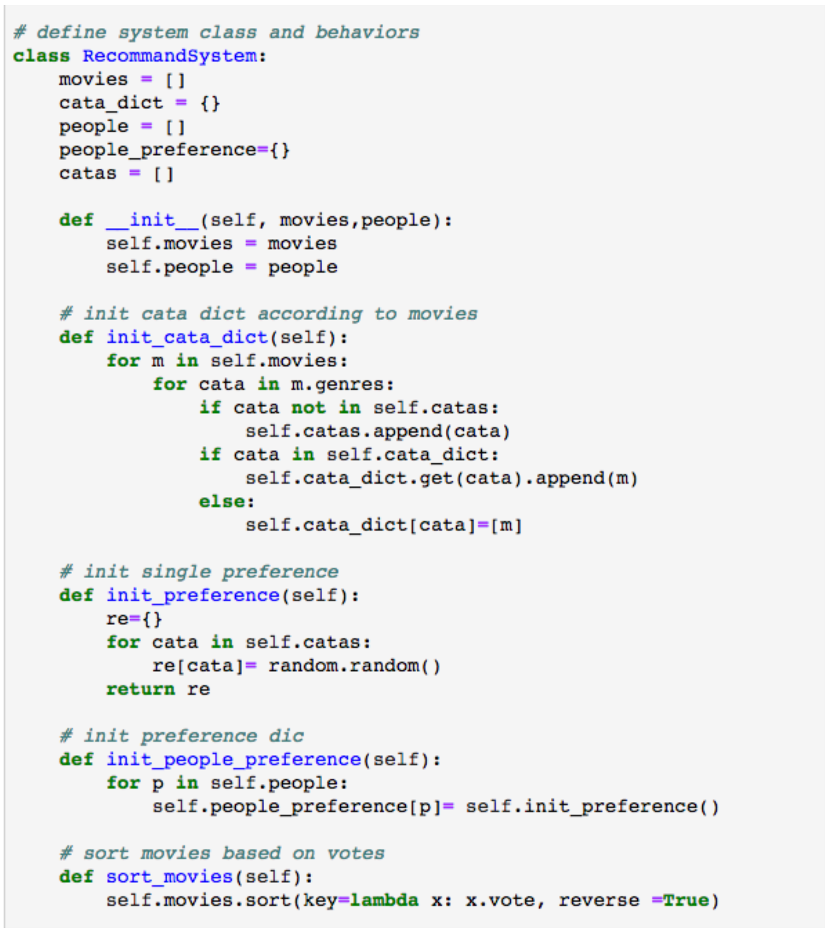
```python
# select related columns
features=['genres','keywords','vote_average']
df = df[features]
df.head()
```



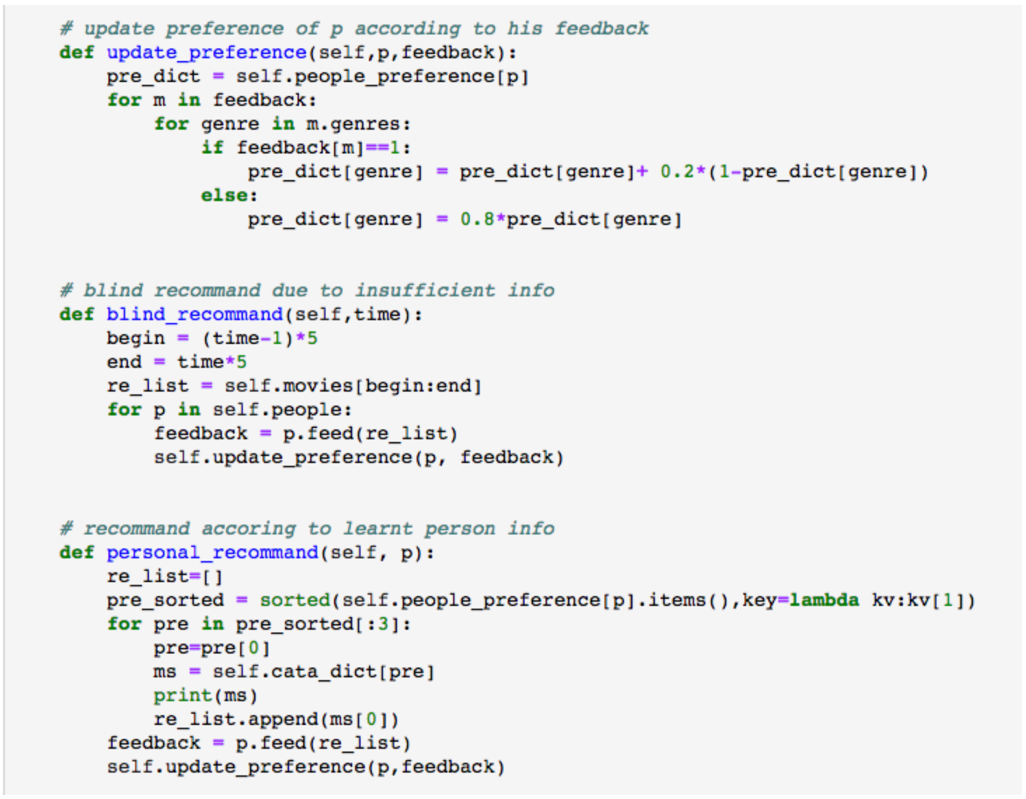
	genres	keywords	vote_average
0	['Action', 'Adventure', 'Fantasy', 'Science Fi...	['culture clash', 'future', 'space war', 'spac...	7.2
1	['Adventure', 'Fantasy', 'Action']	['ocean', 'drug abuse', 'exotic island', 'east...	6.9
2	['Action', 'Adventure', 'Crime']	['spy', 'based on novel', 'secret agent', 'seq...	6.3
3	['Action', 'Crime', 'Drama', 'Thriller']	['dc comics', 'crime fighter', 'terrorist', 's...	7.6
4	['Action', 'Adventure', 'Science Fiction']	['based on novel', 'mars', 'medallion', 'space...	6.1



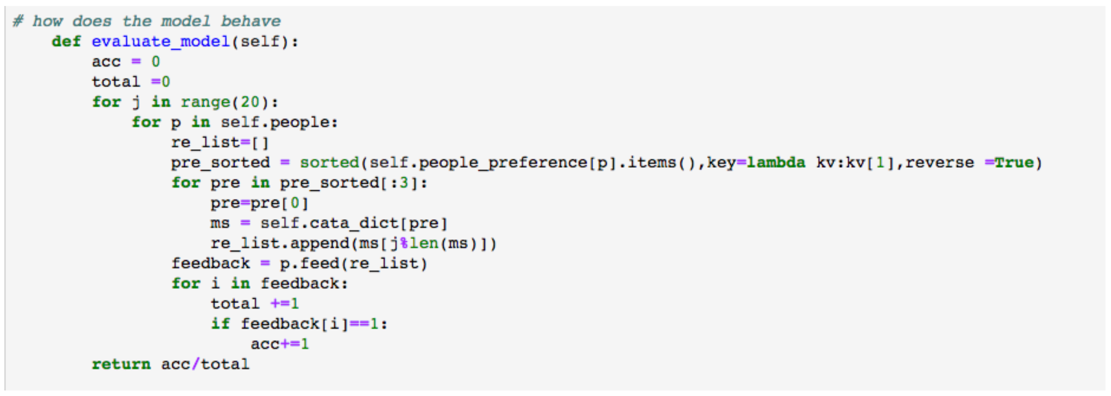
```python
import random

# define movie class and behaviors
class Movie:
    genres = []
    keywords = []
    vote = 0.0
    def __init__ (self,genres,keywords,vote):
        self.genres = genres
        self.keywords = keywords
        self.vote = vote


# define person class and behaviors
class Person :
    preference ={}

    def __init__(self, preference):
        self.preference=preference

    # person react to recommanded movies
    def feed(self, movies):
        feedback ={}
        for m in movies:
            v=0
            for genre in m.genres:
                if genre in self.preference:
                    v += self.preference[genre]
            v=v/len(m.genres)
            if v>=0.6 :
                feedback[m]=1
            else:
                feedback[m]=0
        return feedback
```



```python
# define system class and behaviors
class RecommandSystem:
    movies = []
    cata_dict = {}
    people = []
    people_preference={}
    catas = []

    def __init__(self, movies,people):
        self.movies = movies
        self.people = people

    # init cata dict according to movies
    def init_cata_dict(self):
        for m in self.movies:
            for cata in m.genres:
                if cata not in self.catas:
                    self.catas.append(cata)
                if cata in self.cata_dict:
                    self.cata_dict.get(cata).append(m)
                else:
                    self.cata_dict[cata]=[m]

    # init single preference
    def init_preference(self):
        re={}
        for cata in self.catas:
            re[cata]= random.random()
        return re

    # init preference dic
    def init_people_preference(self):
        for p in self.people:
            self.people_preference[p]= self.init_preference()

    # sort movies based on votes
    def sort_movies(self):
        self.movies.sort(key=lambda x: x.vote, reverse =True)
```

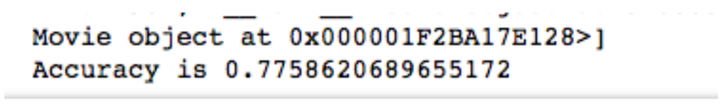


```python
    # update preference of p according to his feedback
    def update_preference(self,p,feedback):
        pre_dict = self.people_preference[p]
        for m in feedback:
            for genre in m.genres:
                if feedback[m]==1:
                    pre_dict[genre] = pre_dict[genre]+ 0.2*(1-pre_dict[genre])
                else:
                    pre_dict[genre] = 0.8*pre_dict[genre]


    # blind recommand due to insufficient info
    def blind_recommand(self,time):
        begin = (time-1)*5
        end = time*5
        re_list = self.movies[begin:end]
        for p in self.people:
            feedback = p.feed(re_list)
            self.update_preference(p, feedback)


    # recommand accoring to learnt person info
    def personal_recommand(self, p):
        re_list=[]
        pre_sorted = sorted(self.people_preference[p].items(),key=lambda kv:kv[1])
        for pre in pre_sorted[:3]:
            pre=pre[0]
            ms = self.cata_dict[pre]
            print(ms)
            re_list.append(ms[0])
        feedback = p.feed(re_list)
        self.update_preference(p,feedback)
```



```
# how does the model behave
    def evaluate_model(self):
        acc = 0
        total =0
        for j in range(20):
            for p in self.people:
                re_list=[]
                pre_sorted = sorted(self.people_preference[p].items(),key=lambda kv:kv[1],reverse =True)
                for pre in pre_sorted[:3]:
                    pre=pre[0]
                    ms = self.cata_dict[pre]
                    re_list.append(ms[j%len(ms)])
                feedback = p.feed(re_list)
                for i in feedback:
                    total +=1
                    if feedback[i]==1:
                        acc+=1
        return acc/total
```

Get result:



```
Movie object at 0x000001F2BA17E128>]
Accuracy is 0.7758620689655172
```

III. Conclusion

Basically, due to random initial preference dictionary and machine resource boundaries, the accuracy printed out is not stable. However, for a certain initial value, the more we train the model, the better its performance will be. The low accuracy indicates that the prediction relationship between the feature and the label is not obvious, which is consistent with the results of our previous regression analysis.

Discussion

Conclusion

Based on results above, we find there are some relationship between vote_average and budget, revenue and to determine if the movie is popular or not. With the Recommender System, we can recommend to the user based on their own preference and update continuously. For the future step, we plan to implement the algorithm model into practice software or App to get further test result based on users' feedback in our reality world.

References

[1] https://en.wikipedia.org/wiki/Recommender_system
Recommender System, From Wikipedia, the free encyclopedia

[2] https://en.wikipedia.org/wiki/K-means_clustering
k-means clustering, From Wikipedia, the free encyclopedia

[3] https://en.wikipedia.org/wiki/Cluster_analysis
Cluster analysis, From Wikipedia, the free encyclopedia

[4] https://www.tensorflow.org/api_docs/python/tf/layers/dense
tf.layers.dense, From TensorFlow