9321 ASSESSMENT 3 REPORT

Part1: Regression

Data processing:

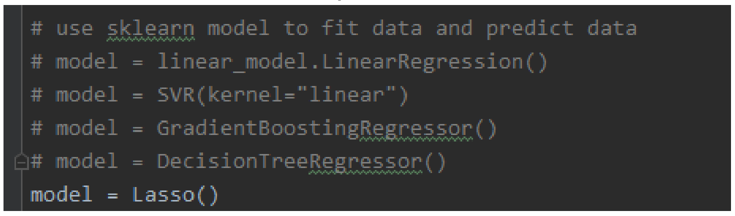
1) Json formatted columns
   In the data set, some columns are json formatted. Use json.loads() method to covert the json string to dictionaries. Now selected the most meaningful key-value pair in the dictionaries. For example, in the crew column, director is the most important for a movie. In the cast column, the actor who has the "0" order is the most popular star. Then find the top 30 values for every column, by using one-hot encoding method, "1" is used to present the director or actor appears in this movie and "0" means not appear. Same for the other json columns like "spoken_language", "key_words" and so on
2) Date formatted column
   Release date is a string which combined with year, month, day. Convert string to the date and add year and month to the dataset.
3) Scale
   Some columns' value (budget) is quite big. Use min max scaler to deal with the data.
4) Missing data
   Drop the rows if there are empty value or missing value in the rows. Use dropna method.

Feature selection

After processing data frame, I drop the text columns (overview, homepage, title), these columns can not be used by the regression model or they are quietly different for every movies. And drop columns which have similar information. For example, production countries, spoken languages and original languages is similar, just keep one of them. Use dataframe.corr() to compute each column's correlation value with the revenue. Remove the columns whose correlation is more than -0.3 and less than 0.3 (here 0.3 is produced by trying different threshold and comparing the p value and MSR). These columns' correlation values with revenue are close to zero, which means they have a weak correlation with revenue. So I do not use there columns to fit the model. However, in the test data set it might not have some columns in the training set. So I use the intersection of the test data columns and train data columns

Model selection

There are many regression models in the sklearn module. I choose LinearRegression, SVR, GradientBoostingRegressor and DecisionTreeRegressor to fit the data set. After running on these model one by one, choose the model which has highest pearson value and lowest mean squared error as the result.



```
# use sklearn model to fit data and predict data
# model = linear_model.LinearRegression()
# model = SVR(kernel="linear")
# model = GradientBoostingRegressor()
# model = DecisionTreeRegressor()
model = Lasso()
```

Part2: Classification

Data Processing

The data has been processed in the part 1. The only difference is that the y value in this part is “rating” column.

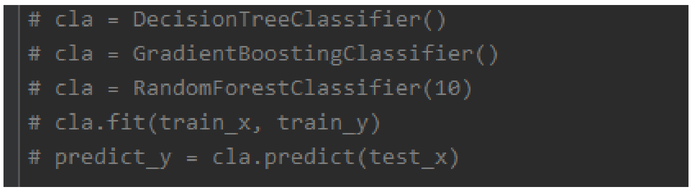
Feature Selection

Same as Part 1 Regression.

Model Selection

1) Select model

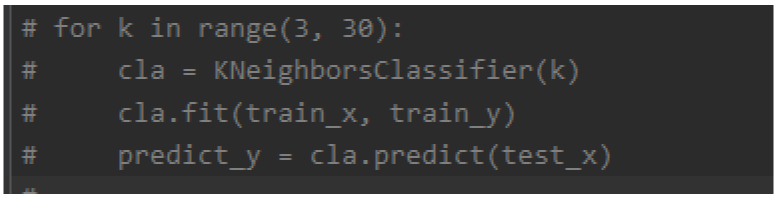
   I tried several different classifier (DecisionTreeClassifier, GradientBoostingClassifier, RandomForestClassifier, KNeighborsClassifier). After comparing the accuracy score of there models, I found the KNighborsClassifier has a better performance when I use a suitable K value.



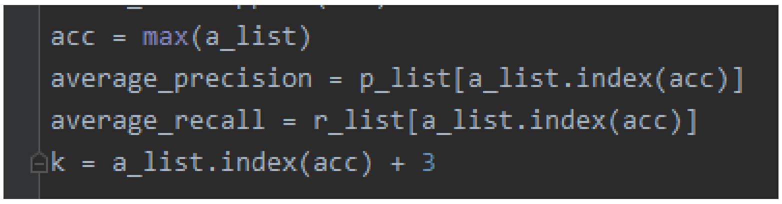
```
# cla = DecisionTreeClassifier()
# cla = GradientBoostingClassifier()
# cla = RandomForestClassifier(10)
# cla.fit(train_x, train_y)
# predict_y = cla.predict(test_x)
```

2) Get value k

   In order to get a k value which has a highest accuracy score, using a for loop to try k between 3 to 30. Keep the highest accuracy score and correspond recall and precision as result.



```
# for k in range(3, 30):
#     cla = KNeighborsClassifier(k)
#     cla.fit(train_x, train_y)
#     predict_y = cla.predict(test_x)
```

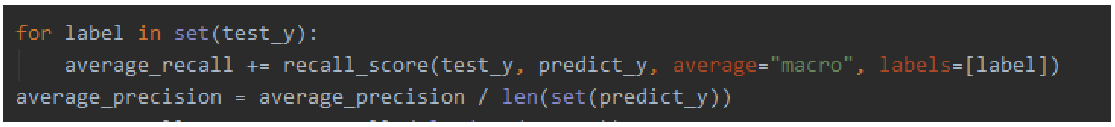


```
acc = max(a_list)
average_precision = p_list[a_list.index(acc)]
average_recall = r_list[a_list.index(acc)]
k = a_list.index(acc) + 3
```

Calculation

Average Precision and Average Recall

Calculate the precision/recall for every class in the predict y and divided by the number of classes. Because it is a multiclass classification, so we set average equals to “macro”.



```
for label in set(test_y):
    average_recall += recall_score(test_y, predict_y, average="macro", labels=[label])
average_precision = average_precision / len(set(predict_y))
```