Predicting Movie Review

Group Report – DSBA Fall 2022

Parag Saxena

Index

Project Summary 1
Data Set 1
Approach 1: 1
Approach 2: 1
Classical Machine Learning (Approach 1) 2
Pre-Processing I : One hot encoding of content rating feature 2
Pre-processing II : Ordinal Encoding of content rating feature 2

Pre-Processing III : Ordinal Encoding of target features 2
Pre-Processing IV : Check VIF for Multicollinearity........ 2
Pre-Processing V : Removed multicollinearity by applying PCA 2
Pre-Processing VI : Distribution of all features 2
Pre-Processing VII : applied if x<=0 then 1 and log transformation 3
Pre-Processing VIII : Checked distribution of target class 3
Pre-Processing IX : Split Dataset into Testing & Training Data 3
Models Building and Evaluation 4
Decision Tree........ 4
Random Forrest 5
Feature importance Analysis 6
Feature Exclusion 7
Fine tune Random Forest Model 7
RandomSearchCV on weighted Random Forrest 8
TEXT ANALYTICS OF REVIEWS DATA (Approach-2)........ 8
Data Preparation........ 8
Pre-Processing Data 8
Pre-Processing I : Ordinal Encoding of target Classes 8
Pre-Processing II: Case conversion of text & Stemming 8
Pre-Processing III: Filtering generic and customized stop words 8
Pre-Processing IV: Visualizing using Word Cloud........ 9
Pre-Processing V : Establish corpus from the list of documents 9
Pre-Processing VI Document term / frequency matrix........ 9
Pre-Processing VII : Sort Document Term Matrix on similarity. 9
Pre-Processing VIII : SVD with TF-IDF Model for classification 10
Pre-Processing IX : Split Dataset into Training & Testing Data 10
Model Building & Evaluation 11
Random Forrest 11
Weighted Random Forrest........ 12
Evaluation 13
Appendix 14

Python Code 14
Data Description 30

Project Summary

Data Set

- rotten_tomatoes_movies.csv - Contains basic information about each movie listed on Rotten Tomatoes; each row represents one movie.

- rotten_tomatoes_critic_reviews_50k.tsv - contains 50000 individual reviews by Rotten Tomatoes critics; each row represents one review corresponding to a movie.

There are two approaches, in first approach which we will predict whether a movie listed in rotten tomato is ‘Fresh’, ‘Rotten’ or ‘Certified Fresh’, using dataset-1. In second approach we will predict two classes ‘Fresh’ or ‘Rotten’ using dataset-2.

Approach 1:

In data-reprocessing we have converted categorical columns using one-hot encoding and ordinal encoding. Checked VIF and removed multi-collinearity by using PCA by adding the columns with very high VIF score and finally uses one PCA competent which explains about 76% variance. Furthermore, we performed log transformation to convert PCA component one and top critics count to normal distribution. And applied a test train split with test size as 20%

In model building we used few classifiers which include Logistic Regression, Multi-layered perceptron, Decision Tree Classifier, Random Forrest Classifier, Gradient Boosting and Extra tree Classifier. Based on the accuracies with default hyperparameters we found that **Decision Tree** and **Random Forrest** have higher accuracies including Gradient Boost and Extra Tree Classifier. However, we choose to proceed our analyses with RF and DT classifiers as they are covered in depth in our course.

In further analyses we found that Random Forrest Classifier with default hyper parameters have slight edge in term for accuracy, precision and recall as compared to Decision Tree. Hence, we optimized Random Forrest algorithm with feature ranking by removing lower ranked features, which improved models’ accuracy, precision and recall. Next, we implemented balanced class in hyperparameter further improving the evaluation metrics. Finally, incorporated RandomSearchCV to further optimize the results of Random Forrest.

Overall, first approach results have high accuracy, precision, recall and f1score.

Approach 2:

In this approach we have dataset with critic’s reviews and two target features ‘Fresh’ and ‘Rotten’. During feature pre-processing, first we joined the previous data with this critic dataset in order to get movie title information. For reviews we applied steps for text pre-processing. Converted reviews to lower case checked the dimensionality then incorporated stemming to convert different types to words to their root form which significantly reduced the dimensionality. In the next step of feature pre-processing, we removed stop-words, including customized stop-words for our further analysis. We finally visualized the most frequents words using WordCloud.

Created corpus from group of documents, used similarity based Ldamodel with four topics to create topic model. Applied TfIDF model to convert all the reviews in SVDs with number of SVDs set to 10. Applied train test split with test size of 20%.

During Model Building phase we again used Random Forrest for consistency. Our accuracy, precision and recall were around .55. we randomly selected the movies to predict the classes and got correct prediction in all the cases.

Classical Machine Learning (Approach 1)

Exploratory Data Analysis & Pre-Processing

Pre-Processing I : One hot encoding of content rating feature
Content Rating feature has 6 categories and we choose to use 'one hot encoding' to categorize the variables for our analysis. Content Rating categories in data set are : ['PG' 'R' 'NR' 'G' 'PG-13' 'NC17'].

Pre-processing II : Ordinal Encoding of content rating feature
Audience status feature has 2 categories and we chose ordinal encoding to encode the variables for us to use these variables in model building.

Audience status category: ['Spilled' 'Upright'] -> Ordinal Encoding (0,1)

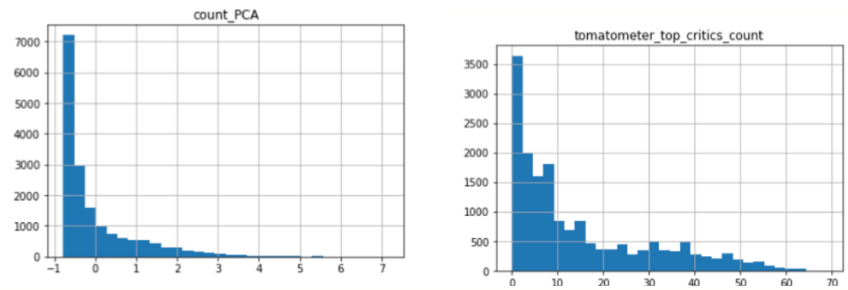
Pre-Processing III : Ordinal Encoding of target features
This is the target feature and has 3 categories:

['Rotten','Fresh','Certified-Fresh']-> Ordinal Encoding(0,1,2)

Pre-Processing IV : Check VIF for Multicollinearity
Three columns are highly multicollinear:
['tomatometer_count','tomatometer_fresh_critics_count','tomatometer_rotten_critics_count']

Pre-Processing V : Removed multicollinearity by applying PCA
For these three columns PCA is applied, and first component of PCA was able to explain 76% of variance. Out of three PCA components first is kept other 2 are removed.

Pre-Processing VI : Distribution of all features
Feature PCA_1 and tomatometer_top_critics_count was not in normal distribution:



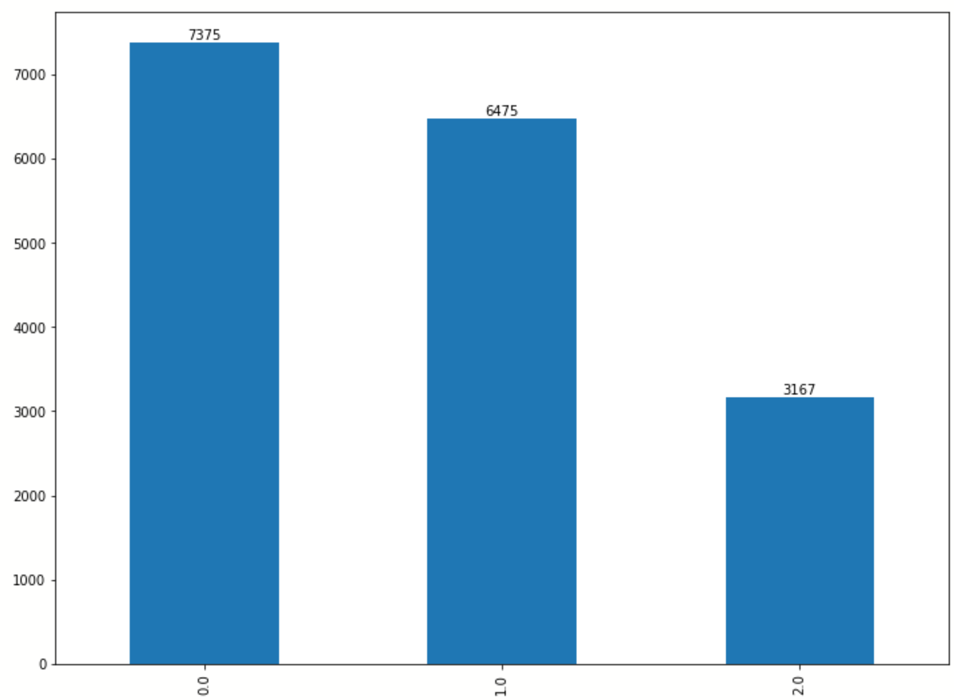
Distribution of PCA_1 & tomatometer_top_critics_count

Pre-Processing VII : applied if x<=0 then 1 and log transformation

For Features Feature PCA_1 and tomatometer_top_critics_count first a function is used to convert all the x<=0 to 1 and kept other number same as x to avoid infinity values before log transformation is applied. After that log transformation was applied of both the features

Pre-Processing VIII : Checked distribution of target class

This class distribution shows that data is not balanced.



Imbalanced distribution of data in target Classes

Pre-Processing IX : Split Dataset into Testing & Training Data

Data is portioned into Test Data to test the model accuracy and Training data to build the model, we have applied tain_test_split using test_size = .20

Models Building and Evaluation

All the classification models were checked to identify performance of best models

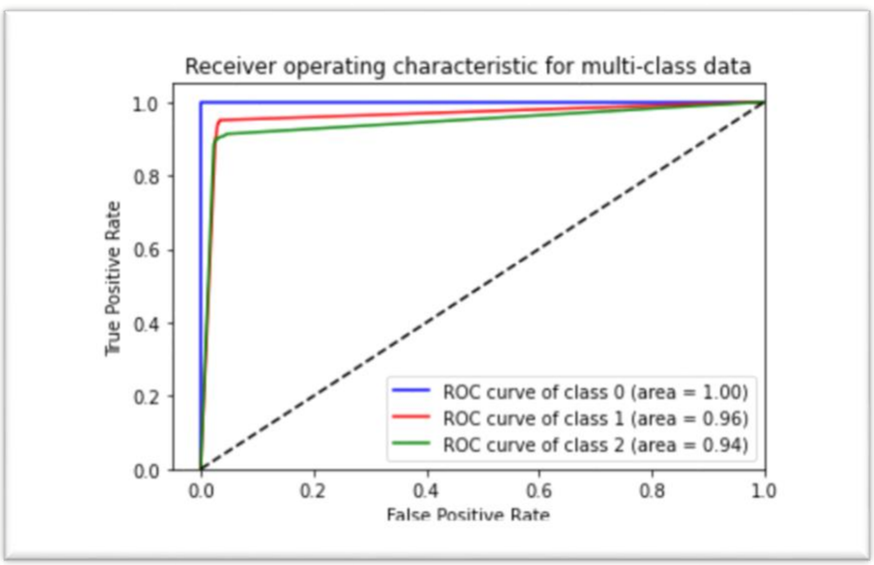
For this we Used for loop to run most of the classifiers to generate confusion-matrix and ROC_AUC curve for all the three target classes.

To understand which classifier may perform best all the classifiers were ran with minimum hyper-parameters tuning.

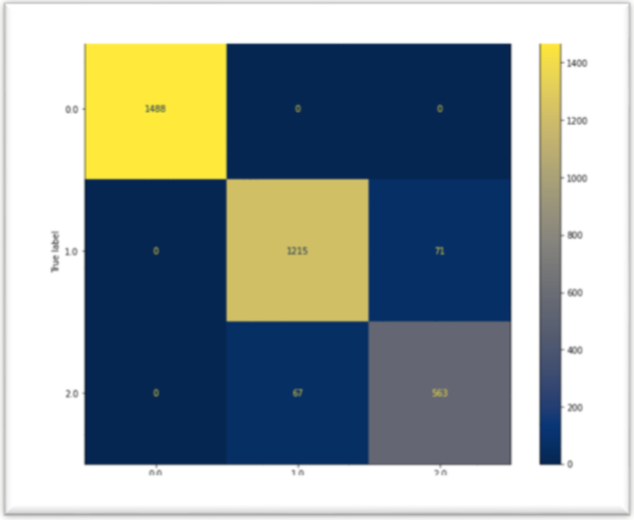
Classifier	Accuracy
Logistic Regression	.74
Neural Multilayer Perceptron	.43
Decision Tree Classifier	.95
Random Forrest Classifier	.96
Simple Vector Machine	.40
Gradient Boosting Classifier	.96
Extra Trees Classifier	.95

Based on the ROC curve and Accuracy Scores used Decision Tree Classifier and Random Forrest Classifier to move ahead with further analysis as they are covered in depth in course.

Decision Tree

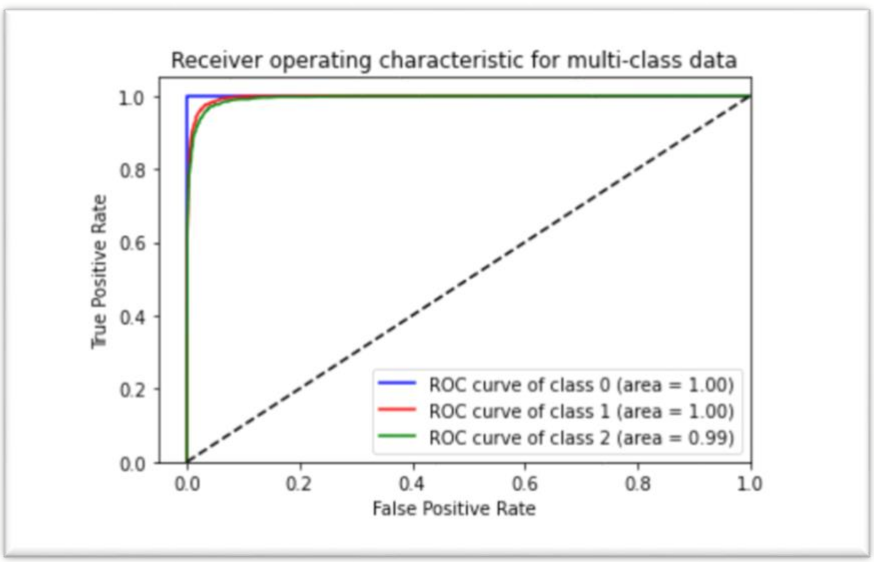


ROC Curve for Decessions Tree Model

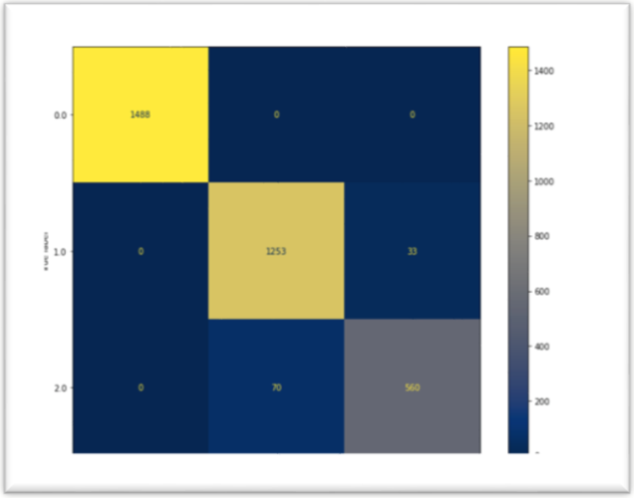


Confusion Matrix for Decessions Tree Model

Random Forrest



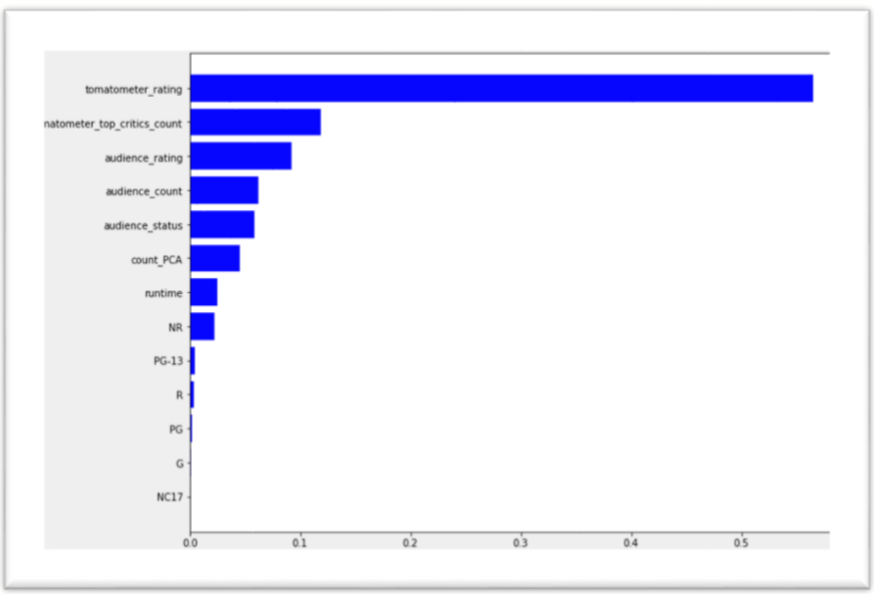
ROC Curve for Random Forest Model



Confusion Matrix for Random Forest Model

Feature importance Analysis

Using default hyperparameter settings for both the selected models we found that on default parameter settings Random Forrest has given higher accuracy than Decision Tree. Hence, we have used Random Forrest for measuring the feature importance.



Feature Importance

Feature Exclusion

Removed following features: runtime, NR, PG-13, R, PG, G and NC17 as they are least importance features. Hence, as a result we got better Precision and Recall and slightly better over-all accuracy.

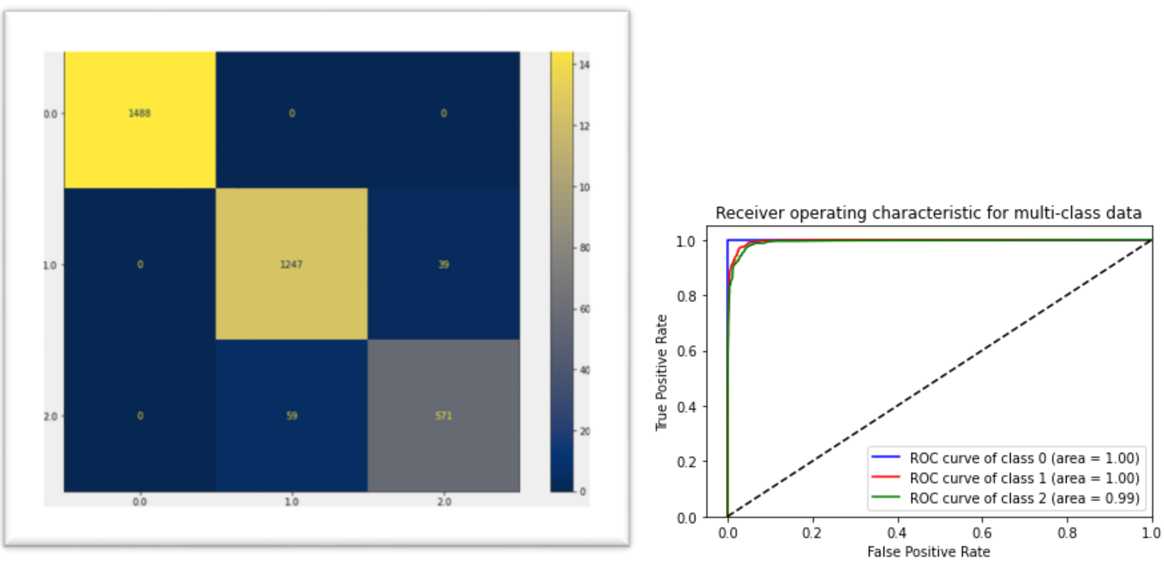
	precision	recall	f1-score	support
0.0	1.00	1.00	1.00	1488
1.0	0.95	0.97	0.96	1286
2.0	0.94	0.90	0.92	630
accuracy			0.97	3404
macro avg	0.96	0.96	0.96	3404
weighted avg	0.97	0.97	0.97	3404

Fine tune Random Forest Model

We implemented class balances to further tune Random Forest Model and visualized all classes and their contribution in target and Computed class balance for each class.

```
{0: 0.7691299435028248, 1: 0.8760360360360361, 2: 1.7910746237238186}
```

Using this class balance to again run weighted Random Forrest. Which further improved the accuracy, precision and recall.



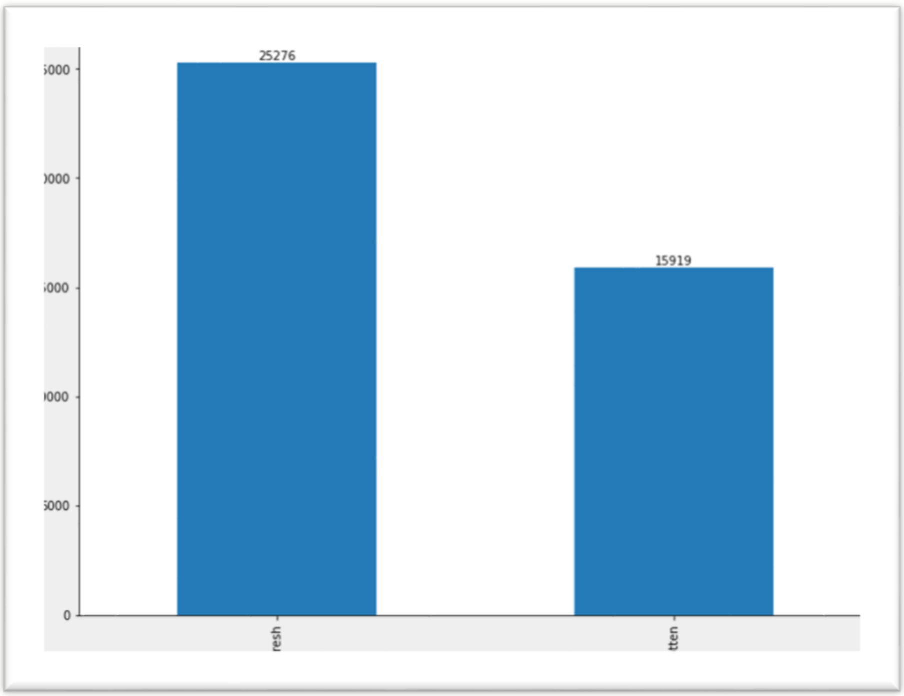
Confusion Matrix & ROC Curve for Random Forest

RandomSearchCV on weighted Random Forrest
As we used Random SearchCV on weighted Random Forest which generated best hyper parameters for weighted Random Forrest.

TEXT ANALYTICS OF REVIEWS DATA (Approach-2)

Data Preparation

Merged critics data frame with movie data frame to get information about movie name we merged the two data frames, text data frame has information about name of critic and reviews. This data frame will have only 2 categories which are '**Fresh'** and '**Rotten'.**



Merged Data set with two Categories

Pre-Processing Data

Pre-Processing I : Ordinal Encoding of target Classes
Both target classes ['Rotten','Fresh'] were encoded where Replaced Rotten by 0 and Fresh by 1. ['Rotten','Fresh'],[0,1].

Pre-Processing II: Case conversion of text & Stemming
After converting all the reviews in lowercase checked number of dimensions which resulted in 19766 potential dimensions. In the next step applied stemming which reduced the dimensionality to 16732.

Pre-Processing III: Filtering generic and customized stop words
Removed all stop words as in English language using NLTK library. Additionally , added customized stop words to the list: ['thi','it\'','hi','ha','--','film','im','movi','wa'].

Pre-Processing IV: Visualizing using Word Cloud

Word Cloud

Pre-Processing V : Establish corpus from the list of documents
Engineered the feature ***review_content*** to create group of documents named as corpus.

Pre-Processing VI Document term / frequency matrix
Created the term dictionary of our corpus, where every unique term is assigned an index.

Filtered out extreme tokens:

- Less than no_below documents (absolute number)
- More than no_above documents (fraction of total corpus size, not absolute number)
- Took no_below=2, no_above=0.75

And, finally converted list of documents (corpus) into Document Term Matrix using dictionary prepared above.

Pre-Processing VII : Sort Document Term Matrix on similarity.
In this step, we sorted document term matrix on the basis of similarity and used genism to create Topic Modelling using n_topics =4 to create a **lda** model.

```
[(0, '0.008*"like" + 0.007*"one" + 0.006*"make" + 0.006*"plot" +
0.006*"stori" + 0.005*"run" + 0.005*"much" + 0.005*"good"'), (1,
'0.009*"one" + 0.009*"make" + 0.006*"much" + 0.006*"like" + 0.005*"time" +
0.005*"never" + 0.005*"stori" + 0.004*"christma"'), (2, '0.008*"run" +
```

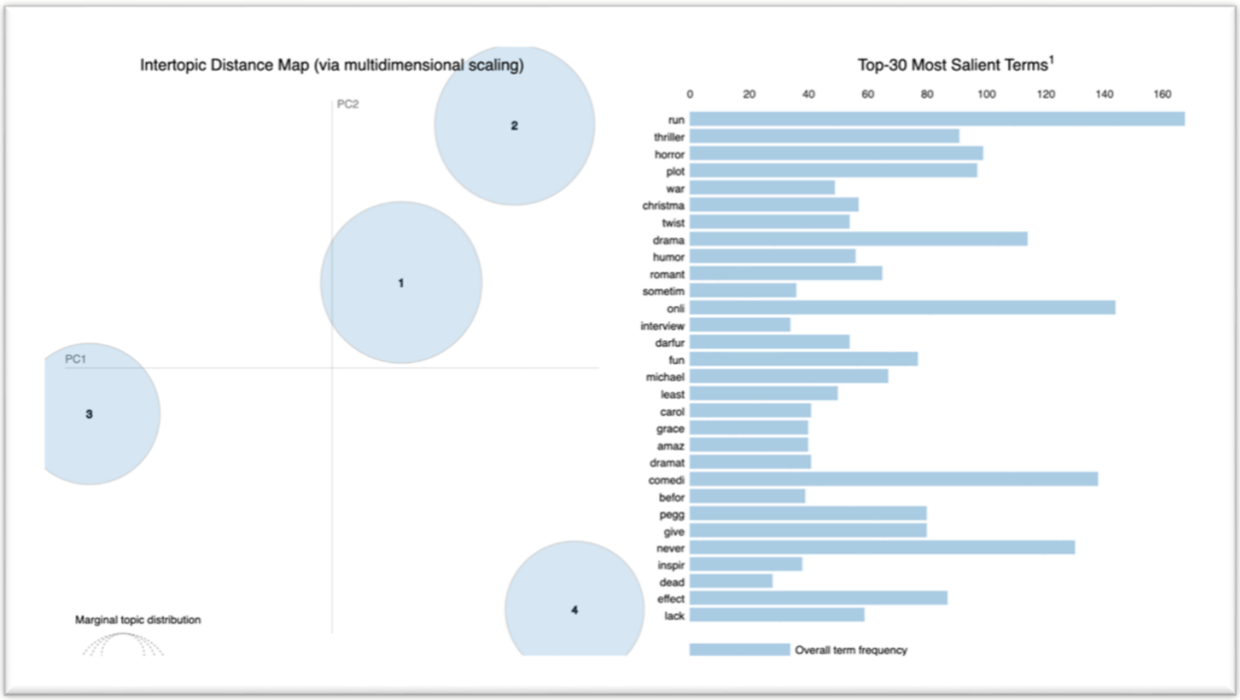
```
0.007*"thriller" + 0.007*"horror" + 0.006*"one" + 0.005*"drama" +
0.005*"good" + 0.005*"us" + 0.004*"war"'), (3, '0.009*"like" + 0.008*"one"
+ 0.005*"director" + 0.005*"charact" + 0.005*"feel" + 0.005*"onli" +
0.004*"michael" + 0.004*"way"')]
```

Topic 1: is related to reviews on plot and story that reviews like

Topic 2: is related to reviews on stories related to Christmas time

Topic 3: is related to reviews about thriller, drama and horror movies

Topic 4: is related to reviews about the director and characters in the movies



Topic Modelling

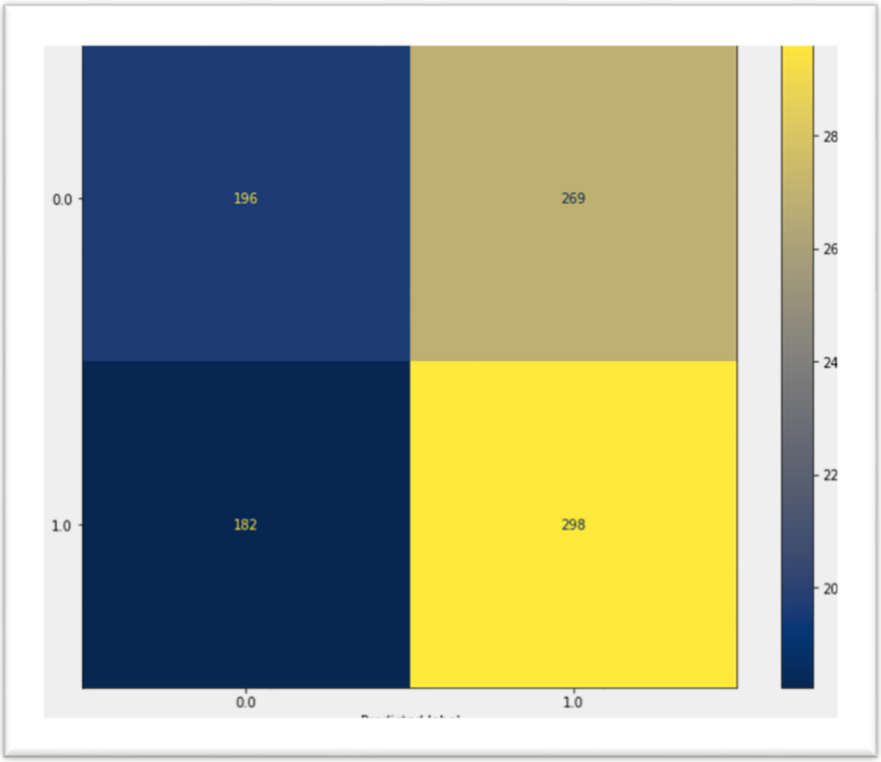
Pre-Processing VIII : SVD with TF-IDF Model for classification
Took n_SVD=10 and with TF_IDF model to create 10 features from review_content text data, which will then be used to for classification.

Pre-Processing IX : Split Dataset into Training & Testing Data
Applied test train split on 10 SVD features a X and target as classification 'Fresh' and 'Rotten' (1,0), with a test_size =0.2

Model Building & Evaluation

We choose to build models using Random Forrest and Weighted Random Forrest.

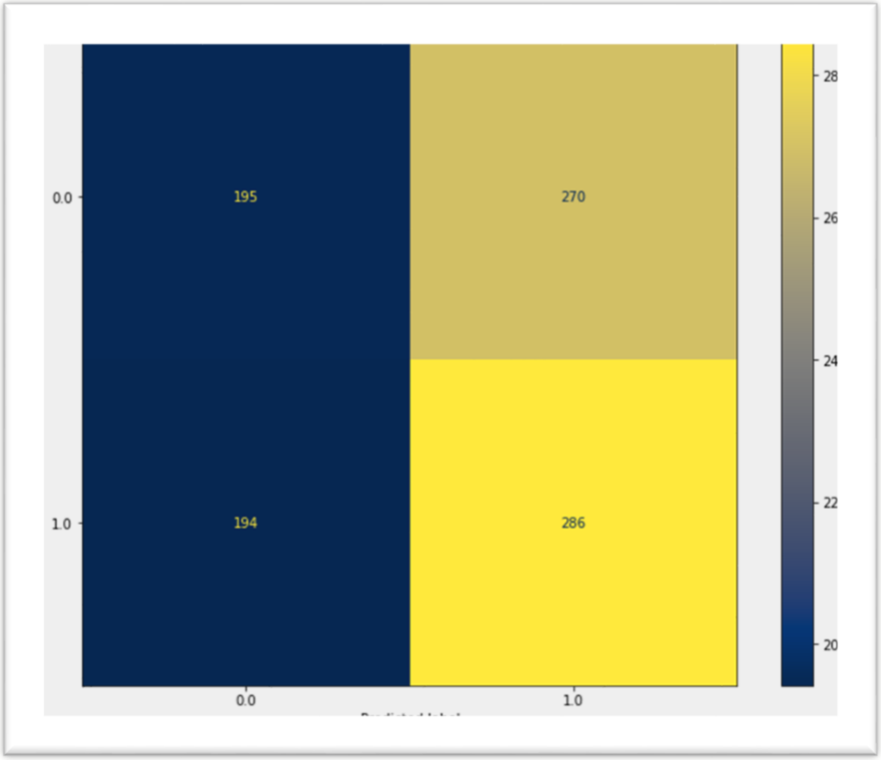
Random Forrest



Confusion Matrix

	precision	recall	f1-score	support
0.0	0.54	0.45	0.49	465
1.0	0.54	0.62	0.58	480
accuracy			0.54	945
macro avg	0.54	0.54	0.53	945
weighted avg	0.54	0.54	0.53	945

Weighted Random Forrest



Confusion Matrix

	precision	recall	f1-score	support
0.0	0.52	0.42	0.47	465
1.0	0.53	0.62	0.57	480
accuracy			0.52	945
macro avg	0.52	0.52	0.52	945
weighted avg	0.52	0.52	0.52	945

Evaluation

To test our models, we have used random movie names to correctly predict classes and compared it with actual classes. Filtered data frame for only the randomly selected movie_titles, predicted the target using svd vectorized features.

Applied Random Forrest model to correctly predict randomly selected movie titles.

Deep Blue

```
# Get the prediction
y_predicted_blue = rf_weighted.predict(X_blue)
predict_movie_status(y_predicted_blue)

Positive review:68.97%
Movie status: Fresh

05] # Get the true label
df_merged['tomatometer_status'].loc[df_merged['movie_title'] == 'Deep Blue'].unique()

array(['Fresh'], dtype=object)
```

Criminal

```
# Get the prediction
y_predicted_Cri = rf_weighted.predict(X_cri)
predict_movie_status(y_predicted_Cri)

Positive review:68.97%
Movie status: Fresh

.01] # Get the true label
df_merged['tomatometer_status'].loc[df_merged['movie_title'] == 'Criminal'].unique()

array(['Fresh'], dtype=object)
```

King Corn

```
[96] # Get the prediction
y_predicted_bol = rf_weighted.predict(X_bol)
predict_movie_status(y_predicted_bol)

Positive review:91.30%
Movie status: Fresh

# Get the true label
df_merged['tomatometer_status'].loc[df_merged['movie_title'] == 'King Corn'].unique()

array(['Fresh'], dtype=object)
```

Appendix

Python Code

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

from sklearn.utils.class_weight import compute_class_weight
from sklearn.tree import DecisionTreeClassifier, plot_tree
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import plot_confusion_matrix, classification_report,
confusion_matrix, accuracy_score
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.model_selection import train_test_split


 ! pip install matplotlib --upgrade
```

EXPLORATORY DATA ANALYSIS AND FEATURE ENGINEERING

```
# Read movie data
df_movie = pd.read_csv('rotten_tomatoes_movies.csv')
df_movie.head()
# Check data distribution
df_movie.describe()
# Data preprocessing I: content_rating feature
print(f'Content Rating category: {df_movie.content_rating.unique()}')

# Visualize the distribution of each category in content_rating feature
ax = df_movie.content_rating.value_counts().plot(kind='bar', figsize=(12,9
))
ax.bar_label(ax.containers[0])
# One hot encoding content_rating feature
content_rating = pd.get_dummies(df_movie.content_rating)
content_rating.head()
# Data preprocessing II: audience_status feature
print(f'Audience status category: {df_movie.audience_status.unique()}')

# Visualize the distribution of each category
ax = df_movie.audience_status.value_counts().plot(kind='bar', figsize=(12,
9))
ax.bar_label(ax.containers[0])
# Encode audience status feature with ordinal encoding
```

```
audience_status = pd.DataFrame(df_movie.audience_status.replace(['Spilled'
,'Upright'],[0,1]))
audience_status.head()

# Data preprocessing III: tomatometer_status feature
# Encode tomatometer status feature with ordinal encoding
tomatometer_status = pd.DataFrame(df_movie.tomatometer_status.replace(['Ro
tten','Fresh','Certified-Fresh'],[0,1,2]))
tomatometer_status
# Combine all of the features together into one dataframe
df_feature = pd.concat([df_movie[['runtime', 'tomatometer_rating', 'tomato
meter_count', 'audience_rating', 'audience_count', 'tomatometer_top_critic
s_count', 'tomatometer_fresh_critics_count', 'tomatometer_rotten_critics_c
ount']]
                            , content_rating, audience_status, tomatometer_sta
tus], axis=1).dropna()
df_feature.head()
# Check the distribution of feature dataframe
df_feature.describe()
# Applying VIF on data to check multi-colinearity

from statsmodels.stats.outliers_influence import variance_inflation_factor
vif_data = pd.DataFrame()
vif_data['features'] = df_feature.columns
vif_data['VIF'] = [variance_inflation_factor(df_feature.values,i)
for i in range(len(df_feature.columns))]
vif_data
# combined all the highly multicolinear features
df_PCA  = df_feature[['tomatometer_count','tomatometer_fresh_critics_count
','tomatometer_rotten_critics_count']]
#PCA to remove multicolinearity

# Do feature scaling
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
Tomato_ALLcounts = sc.fit_transform(df_PCA)

# Apply PCA
from sklearn.decomposition import PCA
pca = PCA(n_components=None)
pca.fit(Tomato_ALLcounts)

# Get the eigenvalues
print("Eigenvalues:")
print(pca.explained_variance_)
```

```
print()

# Get explained variances
print("Variances (Percentage):")
print(pca.explained_variance_ratio_ * 100)
print()

# Make the scree plot
plt.plot(np.cumsum(pca.explained_variance_ratio_ * 100))
plt.xlabel("Number of components (Dimensions)")
plt.ylabel("Explained variance (%)")
counts = Tomato_ALLcounts[:,0]
df_feature.drop(columns=df_PCA.columns,inplace=True)
df_feature.columns
df_feature['count_PCA'] = counts
df_feature.hist(bins =30 , figsize=(30,20))
def convert_zero_to_one(x):
  if x<=0:
    return 1
  else:
    return x

df_feature['tomatometer_top_critics_count'] = df_feature['tomatometer_top_
critics_count'].apply(convert_zero_to_one)
df_feature['count_PCA'] = df_feature['count_PCA'].apply(convert_zero_to_on
e)

df_feature['count_PCA'] = np.log(df_feature['count_PCA'])
df_feature['tomatometer_top_critics_count'] = np.log(df_feature.tomatomete
r_top_critics_count)
df_feature
# Check class distribution of our target variable:tomatometer_status
ax = df_feature.tomatometer_status.value_counts().plot(kind='bar', figsize
=(12,9))
ax.bar_label(ax.containers[0])
y= df_feature.tomatometer_status
# Split the data into training and test data
X_train, X_test, y_train, y_test = train_test_split(df_feature.drop(['toma
tometer_status'], axis=1), df_feature.tomatometer_status, test_size= 0.2,
random_state=42)
print(f'Size of training data is {len(X_train)} and the size of test data
is {len(X_test)}')
from nltk.classify.weka import ClassifierI
import numpy as np
import pandas as pd
```

```python
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import mean_squared_error
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.ensemble import ExtraTreesClassifier
from sklearn.ensemble import VotingClassifier
from sklearn import linear_model
from sklearn.neural_network import MLPClassifier
from sklearn.svm import SVC

from sklearn.pipeline import make_pipeline
from sklearn.preprocessing import StandardScaler, MaxAbsScaler

import matplotlib.pyplot as plt
from sklearn import tree
import seaborn as sns



Classifiers = {
    'Logistic' : LogisticRegression(),
    'MLP': MLPClassifier(random_state=42, max_iter=500, learning_rate="con
stant", learning_rate_init=0.6),
    'DecisionTree': DecisionTreeClassifier(max_depth=15, random_state=42),
    'RandomForest': RandomForestClassifier(random_state=42),
    'SVM': SVC(class_weight='balanced',kernel='rbf',probability=True),
    'GradientBoosting': GradientBoostingClassifier(random_state=42,learnin
g_rate=0.6, warm_start=True),
    'ExtraTrees': ExtraTreesClassifier(n_estimators=100, random_state=42),
}



for model in Classifiers:
  # Train the model on the training data

  Classifiers[model].fit(X_train, y_train)


# Predict the test data with the trained model
  y_predict = Classifiers[model].predict(X_test)

#Print accuracy score and classification report
```

```
  print(accuracy_score(y_test, y_predict))
  print(classification_report(y_test, y_predict))

  fig, ax = plt.subplots(figsize=(12, 9))
  plot_confusion_matrix(Classifiers[model], X_test, y_test, cmap ='cividis
', ax=ax)
import matplotlib.pyplot as plt
from sklearn import datasets
from itertools import cycle
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import label_binarize
from sklearn.metrics import roc_curve, auc
from sklearn.multiclass import OneVsRestClassifier
from sklearn.tree import DecisionTreeClassifier

for model in Classifiers:

 y_score = Classifiers[model].fit(X_train, y_train).predict_proba(X_test)

 y_test_bin = label_binarize(y_test, classes=[0, 1, 2])
 n_classes = y_test_bin.shape[1]



 fpr = dict()
 tpr = dict()
 roc_auc = dict()
 for i in range(n_classes):
    fpr[i], tpr[i], _ = roc_curve(y_test_bin[:, i], y_score[:, i])
    roc_auc[i] = auc(fpr[i], tpr[i])
 colors = cycle(['blue', 'red', 'green'])
 for i, color in zip(range(n_classes), colors):
    plt.plot(fpr[i], tpr[i], color=color,
             label='ROC curve of class {0} (area = {1:0.2f})'
             ''.format(i, roc_auc[i]))

 plt.plot([0, 1], [0, 1], 'k--')
 plt.xlim([-0.05, 1.0])
 plt.ylim([0.0, 1.05])
 plt.xlabel('False Positive Rate')
 plt.ylabel('True Positive Rate')
 plt.title('Receiver operating characteristic for multi-class data')
 plt.legend(loc="lower right")
 plt.show()
# Instantiate Decision Tree Classifier with max leaf nodes = 3
```

```
tree_3_leaf = DecisionTreeClassifier(max_leaf_nodes= 3, random_state=2)

# Train the classifier on the training data
tree_3_leaf.fit(X_train, y_train)

# Predict the test data with trained tree classifier
y_predict = tree_3_leaf.predict(X_test)

# Print accuracy and classification report on test data
print(accuracy_score(y_test, y_predict))
print(classification_report(y_test, y_predict))

# Plot confusion matrix on test data
fig, ax = plt.subplots(figsize=(12, 9))
plot_confusion_matrix(tree_3_leaf, X_test, y_test, cmap ='cividis', ax=ax)
# Visualize decision logic of decision tree model
fig, ax = plt.subplots(figsize=(12, 9))
plot_tree(tree_3_leaf, ax= ax)
plt.show()
# Instantiate Decision Tree Classifier with max leaf nodes = 3
tree_3_leaf = DecisionTreeClassifier(max_leaf_nodes= 3, random_state=2)

# Train the classifier on the training data
tree_3_leaf.fit(X_train, y_train)

# Predict the test data with trained tree classifier
y_predict = tree_3_leaf.predict(X_test)

# Print accuracy and classification report on test data
print(accuracy_score(y_test, y_predict))
print(classification_report(y_test, y_predict))

# Plot confusion matrix on test data
fig, ax = plt.subplots(figsize=(12, 9))
plot_confusion_matrix(tree_3_leaf, X_test, y_test, cmap ='cividis', ax=ax)
# Instantiate Decision Tree Classifier with default hyperparameter setting
s
tree = DecisionTreeClassifier(random_state=2)

# Train the classifier on the training data
tree.fit(X_train, y_train)

# Predict the test data with trained tree classifier
y_predict = tree.predict(X_test)
```

```
# Print accuracy and classification report on test data
print(accuracy_score(y_test, y_predict))
print(classification_report(y_test, y_predict))

# Plot confusion matrix on test data
fig, ax = plt.subplots(figsize=(12, 9))
plot_confusion_matrix(tree, X_test, y_test, cmap ='cividis', ax=ax)
# Instantiate Random Forest Classifier
rf = RandomForestClassifier(random_state=2)

# Train Random Forest Classifier on training data
rf.fit(X_train, y_train)

# Predict test data with trained model
y_predict = rf.predict(X_test)

# Print accuracy score and classification report
print(accuracy_score(y_test, y_predict))
print(classification_report(y_test, y_predict))

# Plot confusion matrix
fig, ax = plt.subplots(figsize=(12, 9))
plot_confusion_matrix(rf, X_test, y_test, cmap ='cividis', ax=ax)
# Get the fature importance
feature_importance = rf.feature_importances_

# Print feature importance
for i, feature in enumerate(X_train.columns):
    print(f'{feature} = {feature_importance[i]}')

# Visualize feature from the most important to the least important
indices = np.argsort(feature_importance)

plt.figure(figsize=(12,9))
plt.title('Feature Importances')
plt.barh(range(len(indices)), feature_importance[indices], color='b', alig
n='center')
plt.yticks(range(len(indices)), [X_train.columns[i] for i in indices])
plt.xlabel('Relative Importance')
plt.show()
# Split data into train and test after feature selection
X_train, X_test, y_train, y_test = train_test_split(df_feature.drop(['toma
tometer_status', 'NR', 'runtime', 'PG-
13', 'R', 'PG','G', 'NC17'], axis=1),df_feature.tomatometer_status, test_s
ize= 0.2, random_state=42)
```

```
print(f'Size of training data is {len(X_train)} and the size of test data
is {len(X_test)}')
# Initialize Random Forest class
rf = RandomForestClassifier(random_state=2)

# Train Random Forest on the training data after feature selection
rf.fit(X_train, y_train)

# Predict the traind model on the test data after feature selection
y_predict = rf.predict(X_test)

# Print the accuracy score and the classification report
print(accuracy_score(y_test, y_predict))
print(classification_report(y_test, y_predict))

# Plot the confusion matrix
fig, ax = plt.subplots(figsize=(12, 9))
plot_confusion_matrix(rf, X_test, y_test, cmap ='cividis', ax=ax)
# Check class distribution of target variable once more
ax = df_feature.tomatometer_status.value_counts().plot(kind='bar', figsize
=(12,9))
ax.bar_label(ax.containers[0])
# Compute class weight
class_weight = compute_class_weight(class_weight= 'balanced', classes= np.
unique(df_feature.tomatometer_status),
                      y = df_feature.tomatometer_status.values)

class_weight_dict = dict(zip(range(len(class_weight.tolist())), class_weig
ht.tolist()))
class_weight_dict
# Initialize Random Forest model with weight information
rf_weighted = RandomForestClassifier(random_state=2, class_weight=class_we
ight_dict)

# Train the model on the training data
rf_weighted.fit(X_train, y_train)

# Predict the test data with the trained model
y_predict = rf_weighted.predict(X_test)

#Print accuracy score and classification report
print(accuracy_score(y_test, y_predict))
print(classification_report(y_test, y_predict))

#Plot confusion matrix
```

```
fig, ax = plt.subplots(figsize=(12, 9))
plot_confusion_matrix(rf_weighted, X_test, y_test, cmap ='cividis', ax=ax)
# Initialize Random Forest model with weight information
y_score = rf_weighted.fit(X_train, y_train).predict_proba(X_test)

y_test_bin = label_binarize(y_test, classes=[0, 1, 2])
n_classes = y_test_bin.shape[1]



fpr = dict()
tpr = dict()
roc_auc = dict()
for i in range(n_classes):
    fpr[i], tpr[i], _ = roc_curve(y_test_bin[:, i], y_score[:, i])
    roc_auc[i] = auc(fpr[i], tpr[i])
colors = cycle(['blue', 'red', 'green'])
for i, color in zip(range(n_classes), colors):
    plt.plot(fpr[i], tpr[i], color=color,
             label='ROC curve of class {0} (area = {1:0.2f})'
             ''.format(i, roc_auc[i]))

plt.plot([0, 1], [0, 1], 'k--')
plt.xlim([-0.05, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver operating characteristic for multi-class data')
plt.legend(loc="lower right")
plt.show()
# Initialize Random Forest model with weight information
rf_weighted = RandomForestClassifier(random_state=2, class_weight=class_we
ight_dict)

# Train the model on the training data
rf_weighted.fit(X_train, y_train)

# Predict the test data with the trained model
y_predict = rf_weighted.predict(X_test)

#Print accuracy score and classification report
print(accuracy_score(y_test, y_predict))
print(classification_report(y_test, y_predict))

#Plot confusion matrix
```

```
fig, ax = plt.subplots(figsize=(12, 9))
plot_confusion_matrix(rf_weighted, X_test, y_test, cmap ='cividis', ax=ax)
from sklearn.model_selection import RandomizedSearchCV
# Number of trees in random forest
n_estimators = [int(x) for x in np.linspace(start = 200, stop = 2000, num
= 10)]
# Number of features to consider at every split
max_features = ['auto', 'sqrt']
# Maximum number of levels in tree
max_depth = [int(x) for x in np.linspace(10, 110, num = 11)]
max_depth.append(None)
# Minimum number of samples required to split a node
min_samples_split = [2, 5, 10]
# Minimum number of samples required at each leaf node
min_samples_leaf = [1, 2, 4]
# Method of selecting samples for training each tree
bootstrap = [True, False]
# Create the random grid
random_grid = {'n_estimators': n_estimators,
               'max_features': max_features,
               'max_depth': max_depth,
               'min_samples_split': min_samples_split,
               'min_samples_leaf': min_samples_leaf,
               'bootstrap': bootstrap}
print(random_grid)
# Use the random grid to search for best hyperparameters
# Random search of parameters, using 3 fold cross validation,
# search across 100 different combinations, and use all available cores
rf_random = RandomizedSearchCV(estimator = rf_weighted, param_distribution
s = random_grid, n_iter = 100, cv = 3, verbose=2, random_state=42, n_jobs
= -1)
# Fit the random search model
rf_random.fit(X_train, y_train)
rf_random.best_params_
model = RandomForestClassifier()
model.set_params(**rf_random.best_params_)
```

second approach

```
# Read critics dataframe
df_critics = pd.read_csv('rotten_tomatoes_critic_reviews_50k.csv')
df_critics.head()
# Merge critics dataframe with movie dataframe
```

```
df_merged = df_critics.merge(df_movie, how='inner', on=['rotten_tomatoes_l
ink'])
df_merged = df_merged[['rotten_tomatoes_link', 'movie_title', 'review_cont
ent', 'review_type', 'tomatometer_status']]

# Drop entries with missing reviews
df_merged = df_merged.dropna(subset=['review_content'])
# Plot distribution of the review
ax = df_merged.review_type.value_counts().plot(kind='bar', figsize=(12,9))
ax.bar_label(ax.containers[0])
# Pick only 5000 entries from the original dataset
df_sub = df_merged[0:5000]

# Encode the label
review_type = pd.DataFrame(df_sub.review_type.replace(['Rotten','Fresh'],[
0,1]))

# Build final dataframe
df_feature_critics = pd.concat([df_sub[['review_content']]
                        ,review_type], axis=1).dropna()
df_feature_critics.head()
freq = pd.Series(' '.join(df_feature_critics['review_content']).split()).v
alue_counts()
# Convert all words into lower cases
df_feature_critics['review_content'] = df_feature_critics['review_content'
].str.lower()
df_feature_critics['review_content'].head()
def dim():
  dimensions = len(set(df_feature_critics['review_content'].str.split().ex
plode().values))
  print(f'{dimensions} dimensions in the potential DFM.')

dim()
freq = pd.Series(' '.join(df_feature_critics['review_content']).split()).v
alue_counts()
freq = pd.DataFrame(freq).reset_index()
from nltk.stem import PorterStemmer
st = PorterStemmer()
df_feature_critics['review_content'] = df_feature_critics['review_content'
].apply(lambda x: " ".join([st.stem(word) for word in x.split()]))
dim()
import nltk
nltk.download('stopwords')
from nltk.corpus import stopwords
stop = stopwords.words('english')
```

```
custom_stop = ['thi','it\'','hi','ha','--','film','im','movi','wa']
stop.extend(custom_stop)
df_feature_critics['review_content'] = df_feature_critics['review_content'
].apply(lambda x: " ".join(x for x in x.split() if x not in stop))
freq = pd.Series(' '.join(df_feature_critics['review_content']).split()).v
alue_counts()[:20]
freq
df_feature_critics['review_content'] = df_feature_critics['review_content'
].str.replace(r'[^\w\s]+', '')
freq = pd.Series(' '.join(df_feature_critics['review_content']).split()).v
alue_counts()
freq
df_feature_critics['review_content'] = df_feature_critics['review_content'
].apply(lambda x: " ".join(x for x in x.split() if x not in stop))
from wordcloud import WordCloud
from matplotlib import pyplot as plt
comment_words = str(' '.join(df_feature_critics['review_content']).split()
)

import string
comment_words = comment_words.translate(str.maketrans('','',string.punctua
tion))

wordcloud = WordCloud(background_color='white',
                          max_words=200,
                          width=1000,height=1000,
                         ).generate(comment_words)
plt.figure(figsize=(8,8))
plt.clf()
plt.imshow(wordcloud)
plt.axis('off')
plt.show()
print(wordcloud)
doc_complete = df_feature_critics['review_content']


doc_complete
corpus = [doc.split() for doc in doc_complete]

corpus
import gensim
from gensim import corpora, models

# Creating the term dictionary of our courpus, where every unique term is
assigned an index.
```

```
dictionary = corpora.Dictionary(corpus)

# Filter out extreme tokens
# Less than no_below documents (absolute number)
# More than no_above documents (fraction of total corpus size, not absolut
e number)
dictionary.filter_extremes(no_below=2, no_above=0.75)

# Converting list of documents (corpus) into Document Term Matrix using di
ctionary prepared above.
DFM = [dictionary.doc2bow(doc) for doc in corpus]
print(DFM)
len(DFM)
print(dictionary.token2id)
len(dictionary.token2id)
from gensim.similarities import MatrixSimilarity
simil = MatrixSimilarity(DFM, num_features=len(dictionary))
distance = 1 - simil[DFM]
text_sim = pd.DataFrame(simil[DFM])
text_sim[76].sort_values(ascending = False)
from gensim.models import Word2Vec
model = Word2Vec(corpus, min_count=20,size= 40,workers=3, window =3, sg =
1)
n_topics = 4
ldamodel = models.LdaModel(DFM, num_topics=n_topics,id2word = dictionary,
passes=40)
print(ldamodel.print_topics(num_topics=n_topics, num_words=8))
pip install pyLDAvis

import pyLDAvis
pyLDAvis.enable_notebook()
import pyLDAvis.gensim_models
vis = pyLDAvis.gensim_models.prepare(ldamodel, DFM, dictionary)
vis
tfidf = models.TfidfModel(DFM)
DFM_tfidf = tfidf[DFM]

n_SVD = 10
SVD_model = models.LsiModel(DFM_tfidf,
                            id2word=dictionary,
                            num_topics=n_SVD)

SVD=SVD_model[DFM_tfidf]
svd_array = gensim.matutils.corpus2csc(SVD).T.toarray()
```

```
# Convert the results into data frame
svd_df = pd.DataFrame(svd_array)

# Show SVD results: reduced vector representation of the text documents
svd_df

# Pick only 5000 entries from the original dataset
df_sub = df_merged[0:5000]

# Encode the label
review_type = pd.DataFrame(df_sub.review_type.replace(['Rotten','Fresh'],[
0,1]))

# Build final dataframe
df_feature_c = pd.concat([df_sub[['review_content','movie_title']]
                        ,review_type], axis=1).dropna()

df_feature_critics1 = pd.concat([svd_df,df_feature_c],axis=1).dropna()
df_sub
df_feature_critics1

# Split data into training and test data
X_train, X_test, y_train, y_test = train_test_split( df_feature_critics1.d
rop(columns=['review_type','review_content','movie_title']), df_feature_cr
itics1['review_type'], test_size=0.2, random_state=42)
X_train
# Instantiate vectorizer class
#vectorizer = CountVectorizer(min_df=1)

# Transform our text data into vector
#X_train_vec = vectorizer.fit_transform(X_train).toarray()

# Initialize random forest and train it
rf = RandomForestClassifier(random_state=2)
rf.fit(X_train, y_train)

# Predict and output classification report
y_predicted = rf.predict(X_test)

print(classification_report(y_test, y_predicted))

fig, ax = plt.subplots(figsize=(12, 9))
plot_confusion_matrix(rf, X_test, y_test, cmap ='cividis', ax=ax)
# Calculate class weight
```

```
class_weight = compute_class_weight(class_weight= 'balanced', classes= np.
unique(df_feature_critics1.review_type),
                        y = df_feature_critics1.review_type.values)

class_weight_dict = dict(zip(range(len(class_weight.tolist())), class_weig
ht.tolist()))
class_weight_dict


# Instantiate vectorizer class
#vectorizer = CountVectorizer(min_df=1)

# Transform our text data into vector
#X_train_vec = vectorizer.fit_transform(X_train)

# Initialize random forest and train it
rf_weighted = RandomForestClassifier(random_state=2, class_weight=class_we
ight_dict)
rf_weighted.fit(X_train, y_train)

# Predict and output classification report
y_predicted = rf_weighted.predict(X_test)

print(classification_report(y_test, y_predicted))

fig, ax = plt.subplots(figsize=(12, 9))
plot_confusion_matrix(rf_weighted, X_test, y_test, cmap ='cividis', ax=ax)


# Define a function to predict movie status based on the overall sentiment
def predict_movie_status(prediction):
    """Assign label (Fresh/Rotten) based on prediction"""
    positive_percentage = (prediction == 1).sum()/len(prediction)*100

    prediction = 'Fresh' if positive_percentage >= 60 else 'Rotten'

    print(f'Positive review:{positive_percentage:.2f}%')
    print(f'Movie status: {prediction}')
df_feature_critics1['movie_title'][df_feature_critics1['movie_title'].asty
pe('str').str.len()<=10][:200]

df_bol = df_feature_critics1.loc[df_feature_critics1['movie_title'] == 'Ki
ng Corn']
df_bol.head()
# Get the prediction
```

```
X_bol=df_feature_critics1.drop(columns=['review_type','review_content','mo
vie_title'])
X_bol=X_bol.loc[df_merged['movie_title'] == 'King Corn']
X_bol


# Get the prediction
y_predicted_bol = rf_weighted.predict(X_bol)
predict_movie_status(y_predicted_bol)
# Get the true label
df_merged['tomatometer_status'].loc[df_merged['movie_title'] == 'King Corn
'].unique()
# Gather all of the reviews of Angel Heart movie


df_cri = df_feature_critics1.loc[df_feature_critics1['movie_title'] == 'Cr
iminal']
df_cri.head()

X_cri=df_feature_critics1.drop(columns=['review_type','review_content','mo
vie_title'])
X_cri=X_cri.loc[df_merged['movie_title'] == 'Criminal']
X_cri
# Get the prediction
y_predicted_Cri = rf_weighted.predict(X_cri)
predict_movie_status(y_predicted_Cri)
# Get the true label
df_merged['tomatometer_status'].loc[df_merged['movie_title'] == 'Criminal'
].unique()
# Gather all of the reviews of The Duchess movie
df_blue = df_feature_critics1.loc[df_feature_critics1['movie_title'] == 'D
eep Blue']
df_blue.head()


X_blue=df_feature_critics1.drop(columns=['review_type','review_content','m
ovie_title'])
X_blue=X_blue.loc[df_merged['movie_title'] == 'Criminal']
X_blue
# Get the prediction
y_predicted_blue = rf_weighted.predict(X_blue)
predict_movie_status(y_predicted_blue)

# Get the true label
```

```
df_merged['tomatometer_status'].loc[df_merged['movie_title'] == 'Deep Blue
'].unique()
```

Data Description

There are 2 datasets

1. rotten_tomatoes_movies.csv - contains basic information about each movie listed on Rotten Tomatoes; each row represents one movie;
2. rotten_tomatoes_critic_reviews_50k.tsv - contains 50.000 individual reviews by Rotten Tomatoes critics; each row represents one review corresponding to a movie;

rotten_tomatoes_movies dataset contains the following columns:

- rotten_tomatoes_link - movie ID
- movie_title - title of the movie as displayed on the Rotten Tomatoes website
- movie_info - brief description of the movie
- critics_consensus - comment from Rotten Tomatoes
- content_rating - category based on the movie suitability for audience
- genres - movie genres separated by commes, if multiple
- directors - name of director(s)
- authors - name of author(s)
- actors - name of actors
- original_release_date - date in which the movie has been released in theatres, in YYY-MM-DD format
- streaming_release_date - date in which the movie has been released on streaming platforms, in YYY-MM-DD format
- runtime - duration of the movie in minutes
- production_company - name of a studio/company that produced the movie
- tomatometer_status - a label assgined by Rotten Tomatoes: "Fresh", "Certified-Fresh" or "Rotten"; **this is the target variables in this challenge**
- tomatometer_rating - percentage of positive critic ratings
- tomatometer_count - critic ratings counted for the calculation of the tomatomer status
- audience_status - a label assgined based on user ratings: "Spilled" or "Upright"
- audience_rating - percentage of positive user ratings
- audience_count - user ratings counted for the calculation of the audience status
- tomatometer_top_critics_count - number of ratings by top critics
- tomatometer_fresh_critics_count - number of critic ratings labeled "Fresh"
- tomatometer_rotten_critics_count - - number of critic ratings labeled "Rotten"

rotten_tomatoes_critic_reviews_50k dataset contains the following columns:

- rotten_tomatoes_link - movie ID
- critic_name - name of critic who rated the movie
- top_critic - boolean value that clarifies whether the critic is a top critic or not
- publisher_name - name of the publisher for which the critic works

- review_type - was the review labeled "Fresh" or "Rotten"?
- review_score - review score provided by the critic
- review_date - date of the review in YYYY-MM-DD format

review_content - text of the review.