Introduction

There is a huge volume of online news published each day, so it would be nice to know the characteristics that separate a popular online news article versus an unpopular one. This is the question we will attempt to answer for the single blog Mashable. Moreover, via machine learning methods, we will be able to predict whether an article will be popular prior to its publication.

Mashable was created by Pete Cashmore in July 2005, and has received numerous awards [3]. Mashable's slogan: Mashable is the leading media company for the Connected Generation and the voice of digital culture. The Mashable website can be found at http://mashable.com/.

Problem Statement and Hypothesis

Our goal is relatively straightforward - given the data set obtained from the UCI Machine Learning Repository [5], build a number of machine learning models to predict the popularity of Mashable articles. The work in this project has been inspired by the work in the paper: Predicting and Evaluating the Popularity of Online News by He Ren and Quan Yan [1]. We attempt to take up where Ren and Yan left off – that it achieve accuracy on the test set of 0.70.

Hypothesis: We propose that we can predict if a Mashable article is popular with 0.70 accuracy or greater.

Materials and Methods

Description of data set

The data set used in this project was obtained from the UCI Machine Learning Repository. The URL for the UCI Machine Learning Repository is https://archive.ics.uci.edu/ml/index.html and the URL for this data set is located at https://archive.ics.uci.edu/ml/datasets/Online+News+Popularity. The data is a heterogeneous set of features which describes articles published by Mashable of a two year period of time. The data set is generally characterized by the UCI Machine Learning Repository as being multivariate, in the area of business, the data values are either elements of the real numbers or elements of the integers, and has 61 columns and 39797 rows.

Columns in the data set:
0. url: URL of the article (non-predictive)
1. timedelta: Days between the article publication and the dataset acquisition (non-predictive)
2. n_tokens_title: Number of words in the title
3. n_tokens_content: Number of words in the content
4. n_unique_tokens: Rate of unique words in the content
5. n_non_stop_words: Rate of non-stop words in the content
6. n_non_stop_unique_tokens: Rate of unique non-stop words in the content
7. num_hrefs: Number of links
8. num_self_hrefs: Number of links to other articles published by Mashable
9. num_imgs: Number of images
10. num_videos: Number of videos
11. average_token_length: Average length of the words in the content
12. num_keywords: Number of keywords in the metadata
13. data_channel_is_lifestyle: Is data channel 'Lifestyle'?
14. data_channel_is_entertainment: Is data channel 'Entertainment'?
15. data_channel_is_bus: Is data channel 'Business'?

16. data_channel_is_socmed: Is data channel 'Social Media'?
17. data_channel_is_tech: Is data channel 'Tech'?
18. data_channel_is_world: Is data channel 'World'?
19. kw_min_min: Worst keyword (min. shares)
20. kw_max_min: Worst keyword (max. shares)
21. kw_avg_min: Worst keyword (avg. shares)
22. kw_min_max: Best keyword (min. shares)
23. kw_max_max: Best keyword (max. shares)
24. kw_avg_max: Best keyword (avg. shares)
25. kw_min_avg: Avg. keyword (min. shares)
26. kw_max_avg: Avg. keyword (max. shares)
27. kw_avg_avg: Avg. keyword (avg. shares)
28. self_reference_min_shares: Min. shares of referenced articles in Mashable
29. self_reference_max_shares: Max. shares of referenced articles in Mashable
30. self_reference_avg_sharess: Avg. shares of referenced articles in Mashable
31. weekday_is_monday: Was the article published on a Monday?
32. weekday_is_tuesday: Was the article published on a Tuesday?
33. weekday_is_wednesday: Was the article published on a Wednesday?
34. weekday_is_thursday: Was the article published on a Thursday?
35. weekday_is_friday: Was the article published on a Friday?
36. weekday_is_saturday: Was the article published on a Saturday?
37. weekday_is_sunday: Was the article published on a Sunday?
38. is_weekend: Was the article published on the weekend?
39. LDA_00: Closeness to LDA topic 0
40. LDA_01: Closeness to LDA topic 1
41. LDA_02: Closeness to LDA topic 2
42. LDA_03: Closeness to LDA topic 3
43. LDA_04: Closeness to LDA topic 4
44. global_subjectivity: Text subjectivity
45. global_sentiment_polarity: Text sentiment polarity
46. global_rate_positive_words: Rate of positive words in the content
47. global_rate_negative_words: Rate of negative words in the content
48. rate_positive_words: Rate of positive words among non-neutral tokens
49. rate_negative_words: Rate of negative words among non-neutral tokens
50. avg_positive_polarity: Avg. polarity of positive words
51. min_positive_polarity: Min. polarity of positive words
52. max_positive_polarity: Max. polarity of positive words
53. avg_negative_polarity: Avg. polarity of negative words
54. min_negative_polarity: Min. polarity of negative words
55. max_negative_polarity: Max. polarity of negative words
56. title_subjectivity: Title subjectivity
57. title_sentiment_polarity: Title polarity
58. abs_title_subjectivity: Absolute subjectivity level
59. abs_title_sentiment_polarity: Absolute polarity level
60. shares: Number of shares (target)

Preprocessing

The data was downloaded from https://archive.ics.uci.edu/ml/machine-learning-databases/00332/ as a single file. The file was read in as a Pandas data frame. The entire data set was subsetted by selecting the features which were used in Predicting and Evaluating the Popularity of Online News by He Ren and Quan Yan [1]. The features used in this study are:

1. kw_avg_avg,
2. LDA_02
3. data_channel_is_world
4. is_weekend
5. data_channel_is_socmed
6. weekday_is_saturday,
7. LDA_04
8. data_channel_is_entertainment
9. data_channel_is_tech
10. kw_max_avg
11. weekday_is_sunday
12. LDA_00
13. num_hrefs, global_
14. subjectivity
15. kw_avg_min
16. title_sentiment_polarity
17. rate_negative_words
18. kw_min_avg
19. title_subjectivity
20. LDA_01

The features data was normalized using the StandardScaler method from the preprocessing module of scikit-learn http://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.StandardScaler.html#sklearn.preprocessing.StandardScaler.

The column 'Shares' in the original data set was assigned as the independent or predicted variable, which was put in a Pandas series. The 'Shares' data is real valued, so it was necessary to map it to integer values because we are interested in classifying news articles into two classes: popular and unpopular. The threshold value of a Mashable shares score greater than or equal to 1400 was mapped to the popular class, while a score of less than 1400 was mapped to the unpopular class. The cutoff value of 1400 was selected because it was the same criterion that He Ren and Quan Yan used

Details of the Modeling Process

The Scikit-learn module was selected as the primary tool for this project. Five approaches were employed in the attempt to reach the goal of 0.70 accuracy on the test data set. The first was using a single model, the second was using an ensemble of models with the majority vote determining the predicted class, the third was an ensemble of models with logistic regression determine the predicted class, the fourth was bagging, and the fifth was using the grid search method from scikit-learn.

Single models:

Support Vector Machines: We used the SVMs with the liner kernel, polynomial, and rbf kernels. The models used are from scikit-learn http://scikit-learn.org/stable/modules/svm.html. For the linear kernel the following parameters were passed to the model object: C=1.0, cache_sixe=200, class_weight=None, coef0=0.0, decision_fucntion_shape=None, degree=3, gamma='auto', kernel='linear', max_iter=-1, probability=True, random_state=1, shrinking=True, tol=0.001, verbose=False. Five-fold cross validation was used in an attempt to improve results. Additional statistics such as precision and recall were extracted from the model objects and ROC curves were plotted. For the polynomial kernel, we used the following parameters: kernel='poly', degree=9, probablility=True, and random_state=1. For the radial kernel, the following parameters were used: kernel = 'rbf', degree=9, probablility=True, and random_state=1.

Random Forests: Random Forests from scikit-learn http://scikit-learn.org/stable/modules/ensemble.html#random-forests was used. The following papameters were passed to the RandomForestClassifier object: n_estimators=10, max_depth=None, min_samples_split=1, random_state=0. Additional statistics such as precision and recall were extracted from the model objects as, ROC curves were plotted.

K-nearest neighbors: K-nearest neighbors from scikit-learn http://scikit-learn.org/stable/modules/neighbors.html was used. For this portion of the project, the three nearest neighbors were used. Additional statistics such as precision and recall were extracted from the model objects as, ROC curves were plotted.

Logistic Regression: Logistic Regression were employed using default parameters from scikit-lear http://scikit-learn.org/stable/modules/linear_model.html#logistic-regression. Additional statistics such as precision and recall were extracted from the model objects as, ROC curves were plotted.

Naïve Bayes: Naïve Bayes were employed using default parameters from scikit-lear http://scikit-learn.org/stable/modules/naive_bayes.html. Additional statistics such as precision and recall were extracted from the model objects as, ROC curves were plotted.

Ensemble of models:

Two methods were used to take various subsets of the set of single models, and then combine them to obtain a single class prediction vector. The first method was a simple majority vote to determine the predicted class, and the second using logistic regression as outlined in Methods for Combining Heterogeneous Sets of Classifiers by Dennis Bahler andf Laura Navarro [2].

Majority Vote: A function, maj_vote, (see code-1) was written to perform the voting process for a set of models. The function takes a Pandas data frame as an argument and returns a list. The columns of the input data frame are the predicted class vectors of a set of models of interest. The output vector gives the class that 'won the vote' where like index values are compared. The maj_vote function was written because the voting classifier from scikit-learn http://scikit-learn.org/stable/modules/ensemble.html#votingclassifier has complex rules for dealing with ties. It was determined that for this project, we would omit such complexity altogether by writing a simple function for the voting process, and not allowing ties by using only odd numbers of models. The majority vote was taken for the complete set of all seven machine learning models, all twenty one permutations of five models at a time, and all thirty five permutations of three models at a time. The code in Code Snippet – 2 was used to generate all of the permutations.

```
def maj_vote(df):
    """Takes A Majority Vote"""
    num_of_models = df.shape[1]
    df_new = df.copy()

    maj_vote = df_new.sum(1)

    ensomble_result = [-1] * len(maj_vote)

    for i in range(len(ensomble_result)):
        if maj_vote[i] >= math.ceil(num_of_models/2):
            maj_vote[i] = 1
        else:
            maj_vote[i] = 0

    return maj_vote
```

Code Snippet – 1

```
def permute_the_vote_2(list_models_results,list_names_models, size_of_sets):
    vote_itterator = itt.combinations(list_models_results,size_of_sets)
    name_itterator = itt.combinations(list_names_models,size_of_sets)

    #get list of combos
    list_of_mod_combos = []
    for i in vote_itterator:
        list_of_mod_combos.append(i)
        #print(i)

    #get list of names
    list_of_mod_combos_names = []
    for i in name_itterator:
        list_of_mod_combos_names.append(i)
        #print(i)

    #put convert tuples of combos to dataframes
    list_of_df =[]
    for i in range(len(list_of_mod_combos)):
        df_temp = pd.DataFrame(list(list_of_mod_combos[i])).T
        df_temp.columns = list_of_mod_combos_names[i]
        list_of_df.append(df_temp)

    #get results
    results_mod_combos =[]
    for i in range(len(list_of_df)):
        temp_maj_vote  = maj_vote(list_of_df[i])
        results_mod_combos.append([calc_con_4(target_ren_test,temp_maj_vote),list_of_mod_combos_names[i]])

    res_temp_df = pd.DataFrame(results_mod_combos, columns=['Results', 'Set of Models'])

    return results_mod_combos
```

Code Snipped - 2

Logistic Regression: We used logistic regression as a method for determining the overall classifier for an ensemble of distinct models. The process we employed, which was outlined in [2], and represented graphically in Figure – 1. The process is as follows. The set of models that make up the ensemble of models are trained using the set of training data. Each model has an output vector. The set of output vectors is then used to train a logistic regression model. Then the test data is run though the set of models that make up the ensemble, and the set of output vectors are used a test data set on the logistic regression model that we previously trained. The output of the logistic regression is our final classification vector.

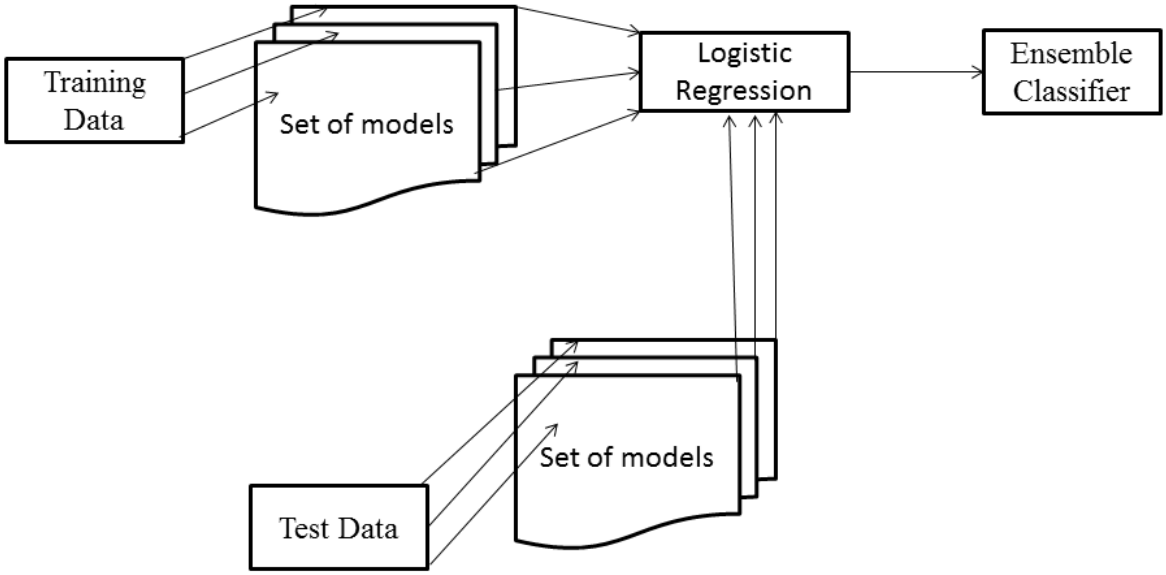

Figure – 1

Bagging: We used bagging on each of our models in an attempt to improve the test accuracy. The scikit-learn bagging classifier was used http://scikit-learn.org/stable/modules/generated/sklearn.ensemble.BaggingClassifier.html. The bagging classifier was employed on each of the following models: logistic regression, K-nearest neighbors, Support Vector Machines (kernel=linear), Support Vector Machines (kernel=polynomial), Support Vector Machines (kernel=rbf), Naïve Bayes, and Random Forests. The bagging classifier was used with default values except for max_samples were set to 0.5, and max_features was set to 0.5.

Gridsearch: The grid search method from scikit-learn http://scikit-learn.org/stable/modules/grid_search.html was used on the two best performing single models in an attempt to achieve better performance. The first model gridsearch was used with was Support Vector Machines (kernel=rbf), and we let the regularization parameter, C, take on the values 1,5, and 10, the gamma parameter was allowed to take on values 1,5, and 10. The second model we employed gridsearch on was Logistic Regression, and we let the regularization parameter take on the values 0.0001, .001, 0.01, 0.1, 1.0, and 10.0.

Results

Single models

Summary

In this project we used seven different machine learning models Support Vector Machines (kernel = linear), Support Vector Machines (kernel = polynomial degree 9), Support Vector Machines (kernel = rbf), Random Forests, K Nearest Neighbors, Logistic Regression, and Naïve Bayes**.** We then attempt to improve upon the single models by using various ensembles and bagging techniques.

The best performing single model was the Support Vector Machines (kernel = rbf) – it performed equal to the best performing models overall.

Support Vector Machines (kernel = linear): The accuracy on training set was 0.615, and the accuracy on test set was 0.611. See Table -1 for the confusion matrix associated with this model, and the ROC curve is shown in Figure -2.

	Actual Not Popular	Actual Popular
Predicted Not Popular	2334	2329
Predicted Popular	1530	3718

Table -1

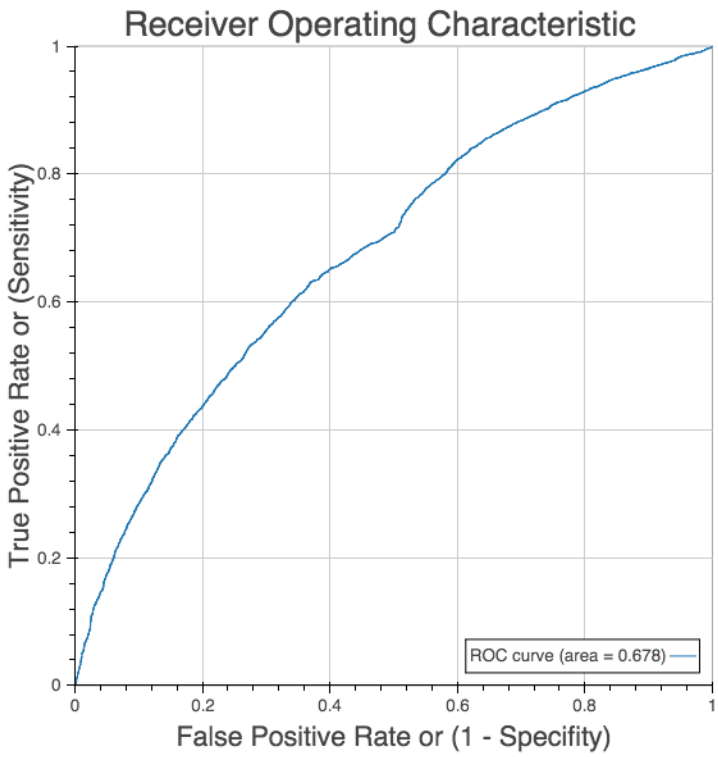

Figure - 2

Support Vector Machines (kernel = polynomial degree 9): The accuracy on training set was 0.696, and the accuracy on test set was 0.594. See Table -2 for the confusion matrix associated with this model, and the ROC curve is shown in Figure -2.

	Actual Not Popular	Actual Popular
Predicted Not Popular	3622	1041
Predicted Popular	2979	2269

Table -2

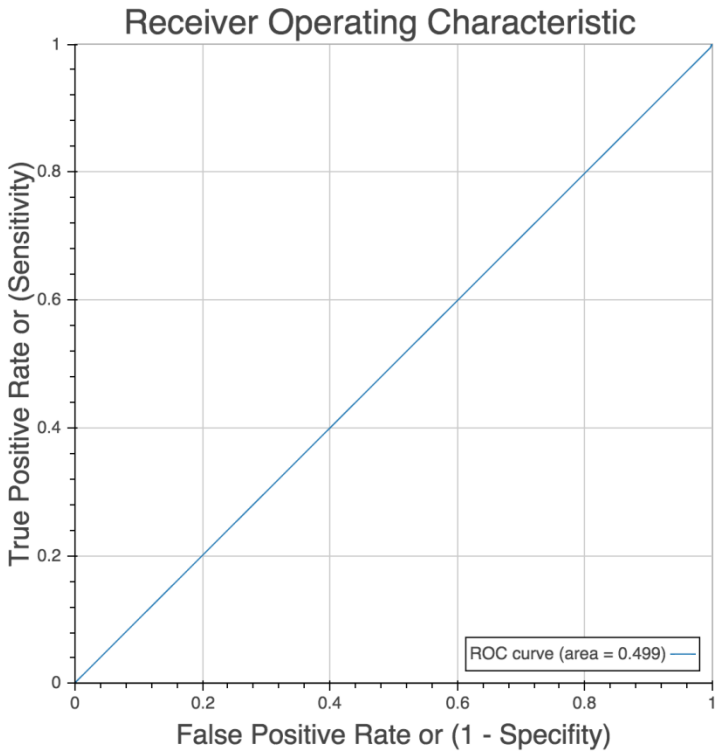

Figure – 3

Support Vector Machines (kernel = rbf): The accuracy on training set was 0.664, and the accuracy on test set was 0.647. See Table -3 for the confusion matrix associated with this model, and the ROC curve is shown in Figure -4.

	Actual Not Popular	Actual Popular
Predicted Not Popular	2673	1990
Predicted Popular	1509	3739

Table -3

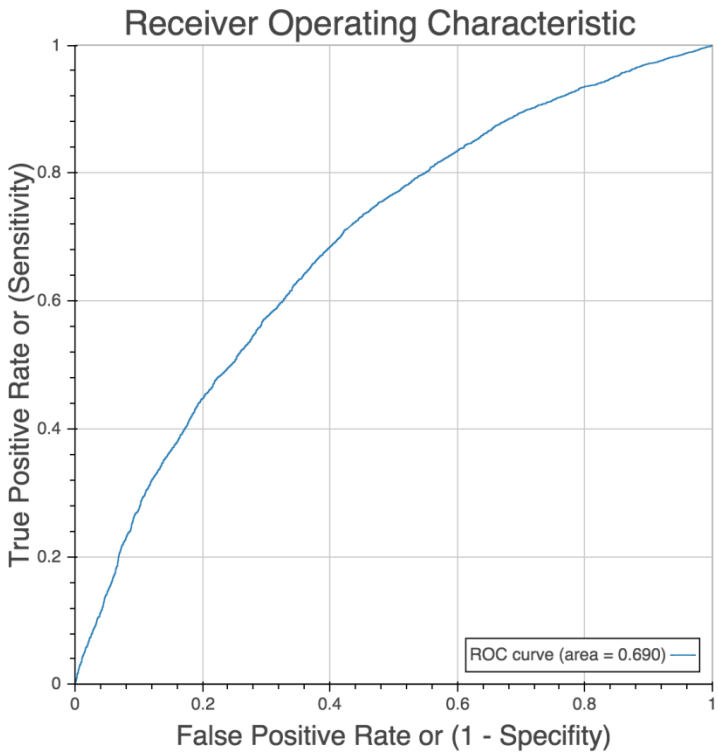

Figure – 4

Random Forests: The accuracy on training set was 0.985, and the accuracy on test set was 0.617. See Table -4 for the confusion matrix associated with this model, and the ROC curve is shown in Figure -5.

	Actual Not Popular	Actual Popular
Predicted Not Popular	2673	1990
Predicted Popular	1509	3739

Table - 4

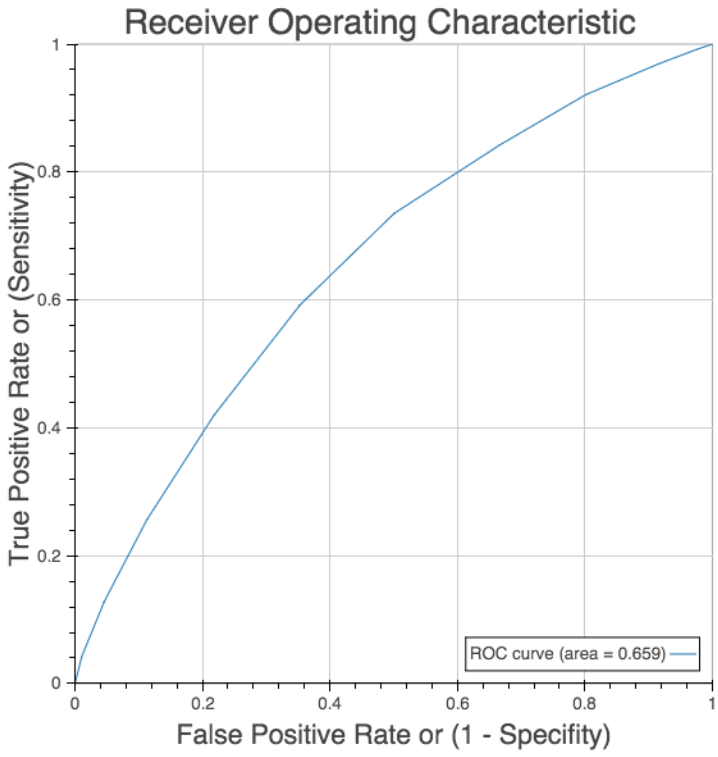

Figure – 5

K Nearest Neighbors: The accuracy on training set was 0.788, and the accuracy on test set was 0.593. See Table -5 for the confusion matrix associated with this model, and the ROC curve is shown in Figure - 6.

	Actual Not Popular	Actual Popular
Predicted Not Popular	2643	2020
Predicted Popular	2017	3231

Table – 5

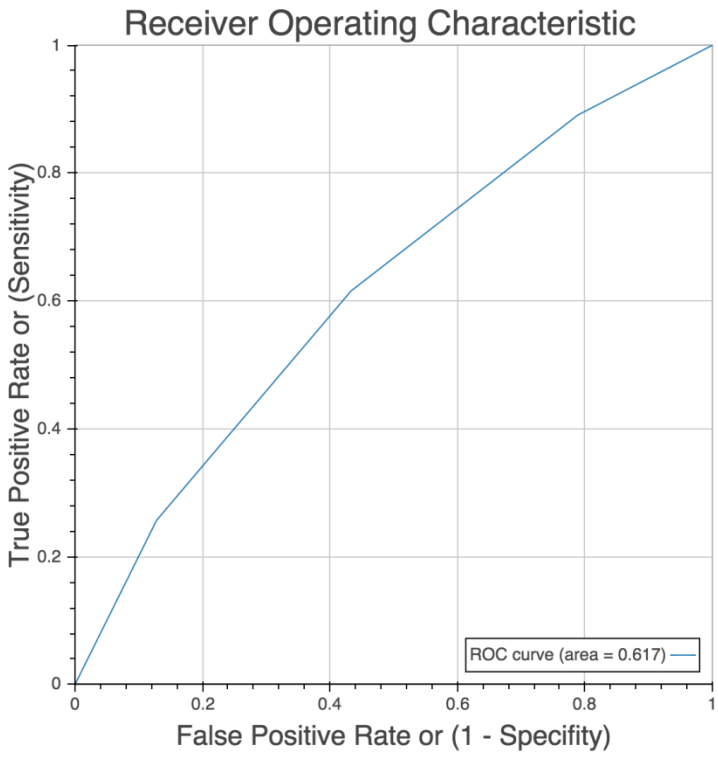

Figure - 6

Logistic Regression: The accuracy on training set was 0.647, and the accuracy on test set was 0.642. See Table -6 for the confusion matrix associated with this model, and the ROC curve is shown in Figure - 7.

	Actual Not Popular	Actual Popular
Predicted Not Popular	2700	1963
Predicted Popular	1581	3667

Table – 6

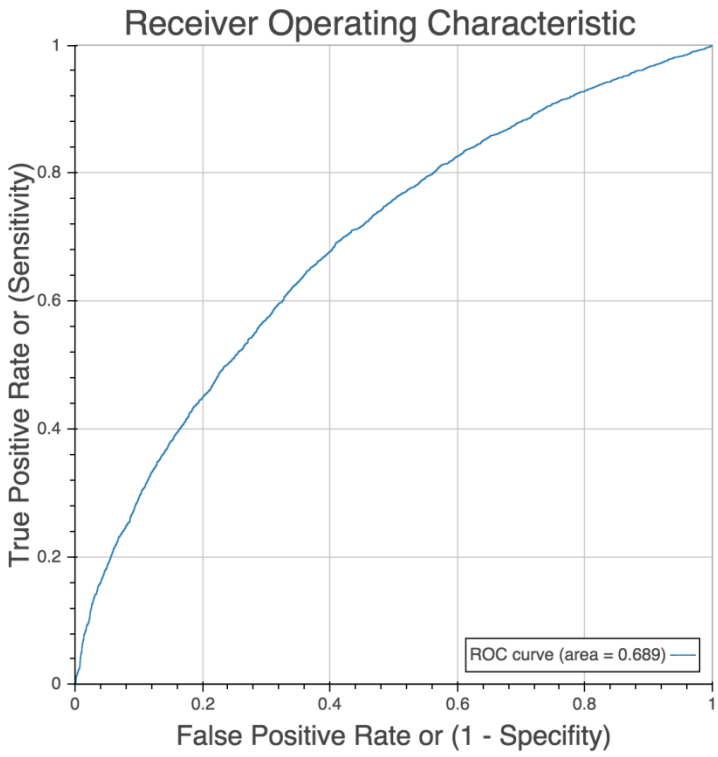

Figure - 7

Naïve Bayes: The accuracy on training set was 0.614, and the accuracy on test set was 0.609. See Table -7 for the confusion matrix associated with this model, and the ROC curve is shown in Figure - 8.

	Actual Not Popular	Actual Popular
Predicted Not Popular	3443	1220
Predicted Popular	2657	2591

Table – 7

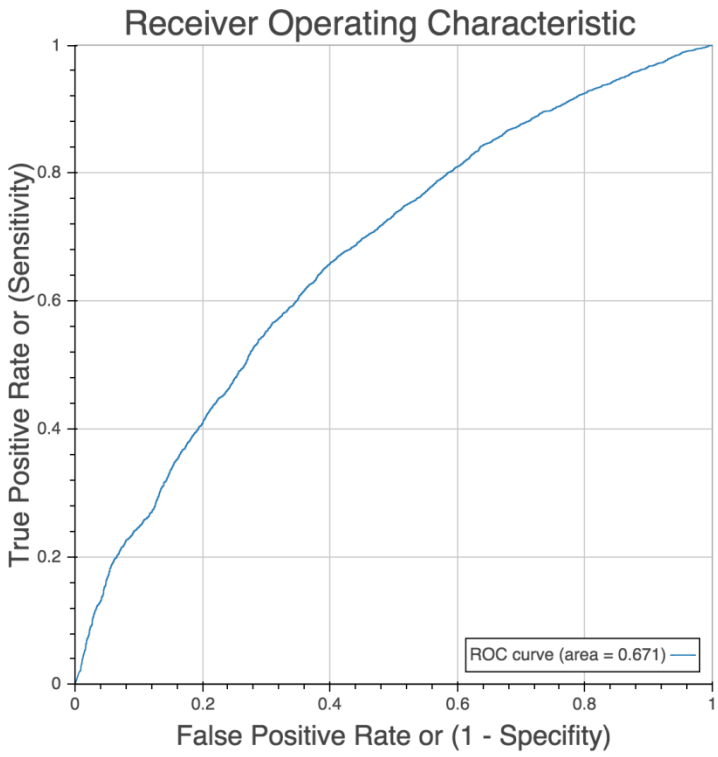

Figure – 8

Majority Vote:

Summary

We employ two distinct voting strategies for ensembles of our single models: majority vote and logistic regression.

Majority vote for set of seven models: Table – 8 summarizes the accuracy on the test set for the entire set of seven models using the ensemble majority voting classifier. See Code Snippet – 1 for code regarding our implementation of the ensemble classifier.

Models	Accuracy on Test Set
SVM(linear),SVM(ploy),SVM(RBF),Random Forest,Logistic Rgression, KNN, Naïve Bayes	0.643

Table – 8

Majority vote for seven models taken five at a time: Table - 9 summarizes the accuracy of the ensemble on the test set for each of the twenty one permutations of the total set of seven models.

Model	Accuracy on Test Set
SVM(linear),SVM(ploy),SVM(RBF),Random Forest, Logistic Regression	0.646
SVM(linear),SVM(rbf),Random Forest, Logistic Regression, Naive Bayes	0.645
SVM(linear),SVM(ploy),SVM(RBF),KNN, Logistic Regression	0.644
SVM(linear),SVM(rbf),Random Forest, KNN, Logistic Regression	0.644
SVM(linear),SVM(rbf),KNN, Logistic Regression ,Naive Bayes	0.644
SVM(poly),SVM(rbf),Random Forest, KNN, Logistic Regression	0.644
SVM(rbf),Random Forest, KNN, Logistic Regression, Naïve Bayes	0.643
SVM(linear),SVM(ploy),SVM(RBF),Logistic Regression, Naive Bayes	0.641
SVM(linear),SVM(rbf),Random Forest, KNN, Naive Bayes	0.641
SVM(linear),SVM(ploy),Random Forest, KNN, Logistic Regression	0.64
SVM(linear),Random Forest, KNN, Logistic Regression, Naive Bayes	0.64
SVM(linear),SVM(ploy),SVM(RBF),Random Forest, KNN	0.639
SVM(linear),SVM(ploy),SVM(RBF),Random Forest, Naive Bayes	0.639
SVM(poly),SVM(rbf),Random Forest, Logistic Regression, Naïve Bayes	0.639
SVM(poly),SVM(rbf),KNN,Logistic Regression, Naïve Bayes	0.637
SVM(linear),SVM(ploy),Random Forest, Logistic Regression, Naive Bayes	0.636
SVM(linear),SVM(ploy),SVM(RBF),KNN, Naive Bayes	0.634
SVM(linear),SVM(ploy)KNN, Logistic Regression, Naive Bayes	0.634
SVM(poly),Random Forest,KNN, Logistic Regression, Naïve Bayes	0.634
SVM(poly),SVM(rbf),Random Forest, KNN, Naïve Bayes	0.633
SVM(linear),SVM(ploy)Random Forest, KNN, Naive Bayes	0.627

Table – 9

Majority vote for seven models taken three at a time: Table - 10 summarizes the accuracy ensemble on the test set for each of the twenty one permutations of the total set of seven models.

Model	Accuracy on Test Set
SVM(rbf), Random Forest, Logistic Regression	0.647
SVM(rbf),KNN,Logistic Regression	0.647
SVM(poly), SVM(rbf), logistic Regression	0.646
SVM(linear), SVM(rbf), Logistic Regression	0.645
SVM(linear), SVM(rbf), Naïve Bays	0.643
SVM(rbf), Logistic Regression, Naïve Bays	0.643
SVM(linear),SVM(ploy),SVM(rbf)	0.642
SVM(linear), SVM(rbf), Random Forest	0.642
SVM(linear), Random Forest, Logistic Regression	0.639
SVM(linear), Logistic Regression, Naïve Bayes	0.639
SVM(rbf), Random Forest, KNN	0.639
Random Forest, KNN, Logistic Regression	0.639
SVM(linear),SVM(poly),Logistic Regression,	0.638
SVM(poly), SVM(rbf), Random Forest	0.638
SVM(poly), Random Forest, Logistic Regression	0.637
SVM(rbf), Random Forest, Naïve Bays	0.637
SVM(linear), SVM(rbf),KNN	0.636
SVM(rbf), KNN,Naïve Bays	0.636
Random Forest, Logistic Regression, Naïve Bays	0.636
KNN,Logistic Regression, Naïve Bays	0.635
SVM(linear), KNN, Logistic Regression	0.634
SVM(linear), SVM(poly), Random Forest	0.632
SMV(linear),Random Forest, KNN	0.63
SVM(linear), Random Forest, Naïve Bayes	0.63
SVM(poly), KNN, Logistic Regression	0.629
SVM(ploy), SVM(rbf),KNN	0.628
Random Forest, KNN, Naïve Bays	0.625
SVM(linear),KNN, Naïve Bayes	0.622
SMV(poly), SVM(rbf), Naive Bays	0.622
SVM(poly), Logistic Regression, Naïve Bays	0.62
SVM(poly), Random Forest, KNN	0.619
SVM(linear),SVM(ploy), KNN	0.616
SVM(poly), Random Forest, Naïve Bays	0.615
SVM(linear), SVM(poly), Naïve Bays	0.614
SVM(poly), KNN, Naïve Bays	0.611

Table - 10

Ensemble using logistic regression

Ensemble using logistic regression**:** The accuracy on training set was 0.985, and the accuracy on test set was 0.618. See Table -11 for the confusion matrix associated with this model.

	Actual Not Popular	Actual Popular
Predicted Not Popular	3019	1644
Predicted Popular	2143	3105

Table – 11

Bagging

Bagging on logistic regression: The accuracy on the training set was 0.635, and the accuracy on the test set was 0.627. See Table – 12 for the confusion matrix associated with this model.

	Actual Not Popular	Actual Popular
Predicted Not Popular	2295	2368
Predicted Popular	1331	3917

Table – 12

Bagging on K Nearest Neighbors: The accuracy on the training set was 0.789, and the accuracy on the test set was 0.623. See Table – 13 for the confusion matrix associated with this model.

	Actual Not Popular	Actual Popular
Predicted Not Popular	2752	1911
Predicted Popular	1821	3427

Table – 13

Bagging on Support Vector Machine (kernel=rbf): The accuracy on the training set was 0.667, and the accuracy on the test set was 0.646. See Table – 14 for the confusion matrix associated with this model.

	Actual Not Popular	Actual Popular
Predicted Not Popular	2752	1911
Predicted Popular	1821	3427

Table – 14

Bagging on Support Vector Machine (kernel=linear): The accuracy on the training set was 0.615, and the accuracy on the test set was 0.611. See Table – 15 for the confusion matrix associated with this model.

	Actual Not Popular	Actual Popular
Predicted Not Popular	2334	2329
Predicted Popular	1530	3718

Table – 15

Bagging on Support Vector Machine (kernel=polynomial, degree = 8): The accuracy on the training set was 0.535, and the accuracy on the test set was 0.630. See Table – 16 for the confusion matrix associated with this model.

	Actual Not Popular	Actual Popular
Predicted Not Popular	2	4661
Predicted Popular	1	5247

Table – 16

Bagging on Naïve Bayes: The accuracy on the training set was 0.615, and the accuracy on the test set was 0.609. See Table – 17 for the confusion matrix associated with this model.

	Actual Not Popular	Actual Popular
Predicted Not Popular	3426	1237
Predicted Popular	2634	2614

Table – 17

Bagging on Random Forests: The accuracy on the training set was 0.972, and the accuracy on the test set was 0.647 See Table – 18 for the confusion matrix associated with this model.

	Actual Not Popular	Actual Popular
Predicted Not Popular	2761	1902
Predicted Popular	1595	3653

Table – 18

Gridsearch

Gridsearch with Support Vector Machine (kernel=rbf): The accuracy the test set was 0.646, where the best parameters were found to be C = 1, and gamma = 0.100.

Gridsearch Logistic Regression: The accuracy the test set was 0.642, where the best parameters was found to be C = 1.

Conclusions

The goal of reaching an accuracy of 0.70 on the test data set was not reached, but five strategies we used resulted in a value of 0.647. It is true we set a rather difficult goal to reach – given that Ren and Yang et al. where not able to achieve the goal either. Additionally, we used mostly "out of the box" machine learning methods for this project. We did try some interesting ensemble methods, most notably using logistic regression as an ensemble voting method.

What I Learned

My personal goal that I set for myself before the first day of the Data Science course at General Academy was to gain a basic understanding and facility with machine learning using the Python programming language. I believe this goal has been satisfied by lecture and coursework, but this project has been central in clarifying my understanding. Not only did I learn how to employ seven specific machine learning models from scikit-learn, I learned to use the methods in ensembles as well as employ bagging on them.

I also learned that employing a model may not be that difficult, but building a quality model may not be so easy. Although I did use a few techniques that Ren and Yang did not, I still was not able to achieve the desired accuracy.

Challenges and Successes

The primary challenge was coming up with methods to attack the problem. Initially, I employed each machine learning model individually in similar manner to Ren and Yang. Once I realized that I could combine individual machine learning models to achieve better performance, and then a whole new strategy was opened up. Additionally, I found the paper: Methods for Combining Heterogeneous Sets of Classifiers by Bahler and Navarro [2] which was extremely helpful.

From a theoretical perspective, the greatest success was learning about and employing logistic regression as an ensemble voting method.

Future Work

In order to reach the goal of 0.70 accuracy on the test set, there are three strategies that could be followed. The first would be to try more sophisticated techniques such as elastic net, or implement some of the other methods to combine an ensemble of models described in [2]. The second strategy would be to look more closely at the features and try to obtain an optimal subset of the total set of features for each specific model. The third would be to augment the data set. It may be that this data set is too small to reach the stated goal.

Possible Extensions or Business Applications

If it was possible to build a machine learning model that can accurately predict the popularity of an online news article, then it would be a useful tool for the myriad of online news sources and aggregators. Clearly, many improvements would need to be made to the set of models used in this project.

Some obvious improvements are the following. First and foremost – it is necessary to significantly improving the accuracy on test data. It would also be desirable to make sure that the model was general enough to be used for sites other than Mashable. The model would also be more useful if it was not limited to being a binary predictor. It could either be extended such that it was a continuous predictor, or just extend to multiple predicted categories.

Bibliography

1. http://cs229.stanford.edu/proj2015/328_report.pdf
2. http://www4.ncsu.edu/~bahler/aaai2000/aaai2000.pdf
3. https://en.wikipedia.org/wiki/Mashable