CISC 5800 Machine Learning Report

Members: Serafim Petrov, Dylan Smith, Suzanne Zhen

Online News Popularity Classification using Neural Network for Base Data and Bayes, Decision Tree, and Logistic Classifiers for Principal Component Analysis Projected Data

1. Introduction

In the digital age, online news reports have gained significance as a channel to disseminate new information. Access to the latest news can be obtained with a click on the fingertip in a matter of seconds. With its widespread popularity, online news reports have attracted advertisers, who wish to grow their consumer base. As a result, it will be beneficial for advertisers to be able to predict whether an article will become popular or not prior to publication.

In our project, two supervised Machine Learning methods will be used to predict the popularity of an article. The first method will be an implementation of a three layer Neural Network and the second method will be an implementation of PCA alongside the Bayes classifier. The dataset 'Online NewsPopularity' published by University of California, Irvine was used to train and test our classification models. The dataset contains a total of 39,644 entries, published by Mashable (www.mashable .com). Each entry consists of 61 attributes: 1 label, 58 predictive attributes, and two non-predictive attributes. Please see below for a list of attributes and their relevant information:

```
7. Attribute Information:
     0. url:                           URL of the article
     1. timedelta:                     Days between the article publication and
                                       the dataset acquisition
     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)
```

As can be inferred from the feature labels, categorical features such as *data channel category*, and *weekday for publication* have already been processed by One-Hot encoding. In addition, there are no missing values within the dataset so imputation is not necessary.

```
NaN = np.isnan(labelData)
print(np.argwhere(NaN == True))
```

```
[]
```

During our data preprocessing, the two non-predictive attributes: ‘url’ and ‘timedelta’ were removed. The target label ‘shares’ was encoded to a value or either ‘1’or ‘0’ based on the share threshold of 1400 (popular if the number of shares exceeds 1400 and unpopular if the number of shares is below 1400). We chose 1400 as the threshold not only because it is recommended for binary classification, but also because it is the median of the target column ‘shares’ (see figure 1 - ‘50%’).

```
count     39644.000000
mean       3395.380184
std       11626.950749
min           1.000000
25%         946.000000
50%        1400.000000
75%        2800.000000
max      843300.000000
Name:  shares, dtype: float64
```

Figure 1. Statistical summary of target column ‘shares’

After labels assignment by the aforementioned threshold, the dataset has an almost even distribution of positive labels (53%) and negative labels (47%) (see figure 2).

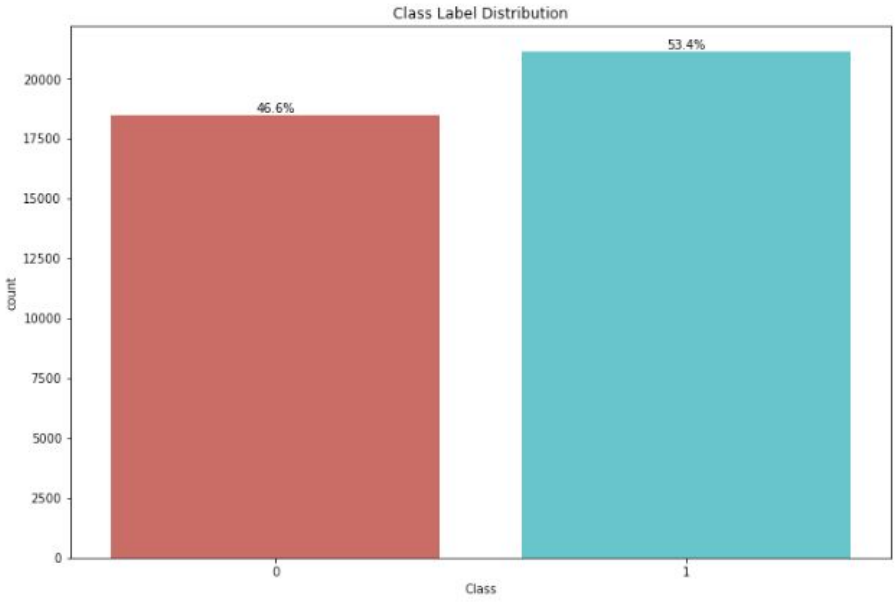

Figure 2. Class label distribution of dataset

From a simple observation of the feature labels, certain attributes such as “*weekdays for publication”* definitely have an impact for predicting whether the article is popular or not. For example, articles published on the weekends are more popular than the ones published during the week since people have more free time to read during the weekend (see figure 3).

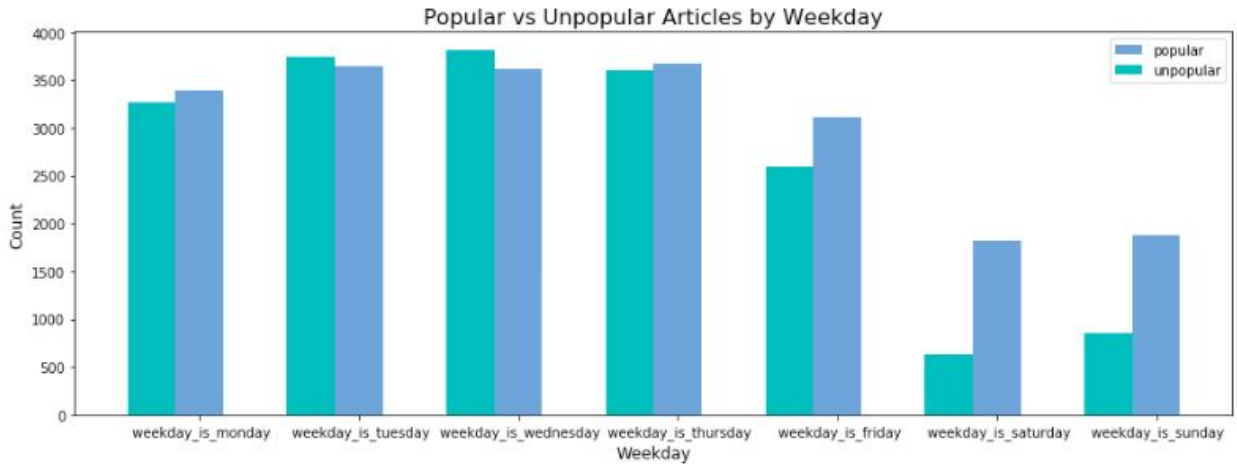

Figure 3. Count of Popular vs Unpopular Articles by Weekday

Another attribute that could potentially affect the popularity of the articles is the category the articles belong to. According to figure 4, there are more popular articles within the “Business”, “Tech”, and “Social Media” categories than unpopular articles. Articles within the “Entertainment” and “World” categories contain more

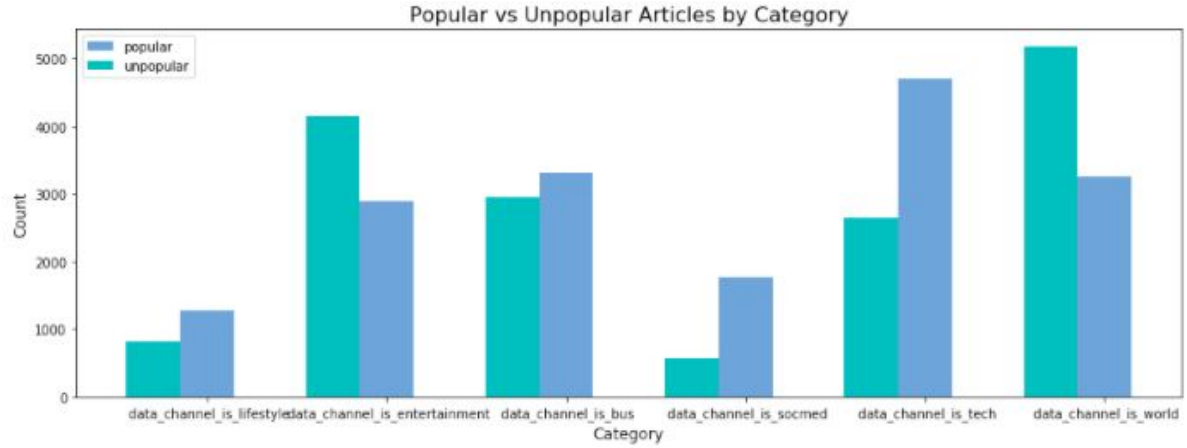

Figure 4. Count of Popular vs Unpopular Articles by Category

In the following sections, we will discuss in detail how we implemented the Neural Network and PCA Bayes Classifier.

2. Neural Network

2.1 Introduction

For our Neural Network Implementation, the discrete features were first normalized using the z-score normalization method to ensure all features had the same standard deviation and mean of 1 and 0 respectively. The dataset was divided with stratified sampling in order to preserve approx class membership ratio, using 75% (for training and 25% for testing. A network with three layers was trained with mini-batches to obtain the optimal weights and biases for our model. Later on, Improvements were made with different activation functions such as tanh and softplus in an attempt to improve the model accuracy.

2.2 Algorithm Explanation

Our algorithm was designed to run a three layer Neural Network on iterations (epochs). The number of nodes in each layer can be adjusted to your desired network model. On the first iteration, all training features get activated by taking the dot product with randomly initiated weights and adding the respective biases at each layer. The sizes of the weights and biases depend on the number of nodes on each layer in the network. For example, a network with a [5, 3, 1] architecture will have weights arrays with shape (58, 5) for the first layer, shape (5, 3) for the second layer, and shape (3,1) for the third layer. The same architecture will have biases arrays with shape (1,5) for the first layer, shape (1,3) for the second layer, and shape (1,1) for the third layer. All weights and biases are randomly initialized with a Gaussian distribution.

After activation, backward propagation was used to adjust the weights and biases on each layer in subsequent iterations. In our sigmoid model, the error term from the top layer is found by subtracting output from the last layer from the actual label. This error is then used to find the deltaW by multiplying the sigmoid derivative of the output (Sigmoid Gradient Descent) and the output. The deltaW from the top layer is propagated down to the second layer using the same calculation except the error term for the second layer is calculated by the dot product between the deltaW and weights of the top layer. DeltaW calculation for the first layer is the same as the second layer. Once the deltaW's for all layers are found,they are multiplied by the learning rate (step size) and added to the old weights to get the new weights. The updates to the biases (deltaB) are calculated by summing up the error terms for each layer and multiplying by the learning rate. The updated weights and biases are then used for the next iterations until we finish all the epochs. The final weights and biases are returned along with the error (loss) for each epoch.

In order to test the accuracy of our model, we used the 25% of test data we set aside earlier. The features for the test data are passed into a forward function, similar to the activation function we used for training except the weights and biases came from our trained model instead of random initiation. For each testing instance, the dot product between the weights and features is taken then add the biases at each layer to obtain the predicted output at the top layer. The predicted outputs are then compared with the actual labels to calculate the accuracy.

Different hyperparameters to experiment with include:

- ❏ Learning rate (experimented from 0.001 to 0.1)
- ❏ Activation function: Sigmoid, TanH, SoftPlus
- ❏ Batch sizes: Batch Gradient Ascent to compute and update the weights and biases on small batches of training data instead of the entire training set
- ❏ Weights and biases random initialization with Gaussian Distribution (mean 0 and standard deviation 0.01)
- ❏ Number of epochs
- ❏ Number of nodes in each layer (different architectures)

2.3 Experimentation Methods

Model 1:

Simple Neural Network Model. First, we constructed a simple model with standard hyperparameters (1 epoch, Sigmoid function, batch size: entire dataset, learning rate 0.1) . The architecture for the model is very simple: Hidden Layer 1: 5 neurons, Hidden Layer 2: 3 neurons, Output layer: 1 neuron. The model returned a very low accuracy 49%.

Model 2:

Improved Neural Network Model. After verifying that the traditional model is too simple to handle this classification task, we started to experiment with the hyperparameters: adding additional neurons to the layers, adjusting the Learning rate, and increasing the number of epochs. We monitored the accuracy at each epoch to find the optimal number of iterations. According to figure 5, the model converges at around 800 epochs. After observing the accuracy rate of many different parameters, we conclude that the following set returns the best combination of simplicity and accuracy (64%):

See Figure and Description below.

Architecture: 25, 15, 1
Learning rate: 0.01
Batch size: 2000
Number of epochs: 850
Activation functions: sigmoid

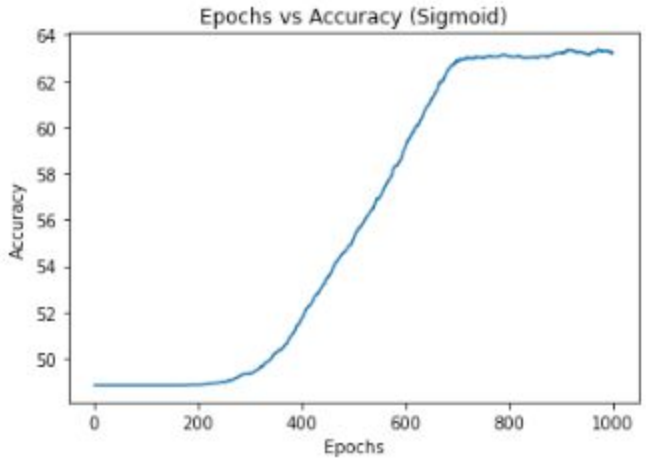

Figure 5 Epochs vs Accuracy for Improved NN Model

Model 3:

Advanced NN models. We proceeded with our experimentation and tried different combinations of activation functions. We used hyperbolic tangent function for the first hidden layer, sigmoid function for the second hidden layer, and softplus function for the output layer. We kept all the other hyperparameters the same as for the Improved Neural Network Model. As a result, we observe lowered accuracy of 51%.

Model 4:

Pure models. In the end, we used pure activation function models: hyperbolic tangent or softplus for all layers. We notice that the accuracy and computational time are lower than the Improved Neural Network Model, which uses sigmoid function for all the layers.

2.4 Results

After running multiple experiments, we came to the conclusion that there are 4 distinctive models for our project. We observed similar accuracies and computation times among the majority of the models.

Please see table 1 for a summary of our findings:

#	Model	Architecture	Activation	Time (sec)	MSE	Accuracy
1	Simple NN Model	5,3,1	Sigmoid	11	453	62.8%
2	Improved NN Model	25,15,1	Sigmoid	24	437	64.8%
3	Advanced NN Model	25,15,1	Tanh, Sigmoid, SoftPlus	21	1097	51.1%
4	Pure Models	25,15,1	Tanh/SoftPlus	20	5228	51.1%

For our highest performing model, improved NN Model, we have included the following confusion matrix:

	precision	recall	f1-score	support
0.0	0.6766	0.6011	0.6366	5071
1.0	0.6258	0.6990	0.6604	4840
accuracy			0.6489	9911
macro avg	0.6512	0.6500	0.6485	9911
weighted avg	0.6518	0.6489	0.6482	9911

In conclusion, although the final accuracy is far from desirable, we had expected the low accuracies due to simplistic versions of the models as well as the experimental nature of the project. A model with simple architecture could provide sufficient performance to correctly identify the majority of the data. We had successfully implemented several models of the algorithm, and adjusted different hyperparameters to optimize it. We came to the conclusion that the more complex functions do not always guarantee higher accuracy.

3. Principal Component Analysis

3.1 Dataset Structuring

Previously, the target variable “class” was derived by using an application of a threshold value of 1400 on the “shares” feature, wherein values at or above would be set to 1 and values below would be set to 0. Index was set to the article title, and the first two features, which are non-predictive, were then dropped.

The dataset was divided with stratified sampling into Training and Testing sets in order to preserve approx class membership ratio, using 80% for training and 20% for testing. Additionally, the predictive features were normalized using the z-score normalization method to ensure all Training features had the same standard deviation and mean of 1 and 0 respectively. In order to keep the Testing data on the same scale, it was normalized with the variance and mean of the Training data. This is also important because in a professional or academic setting the model must be built entirely without the knowledge of the Testing data. The post stratified division distributions are as such:

Train		
class	number	ratio
1	16923	53.4%
0	14792	46.6%
total	31715	0.8000

Test		
class	number	ratio
1	4231	53.4%
0	3698	46.6%
total	7929	0.2000

Total		
class	number	ratio
1	21154	53.4%
0	18490	46.6%
total	39644	

3.2 PCA Introduction

Principal Component Analysis is a method by which a dataset can be mapped linearly onto a new set of orthogonal coordinates. Specifically, its goal is to transform a feature space from its original, into a version that preserves most of the original information, while removing redundant information in features that are potentially highly correlated. This method has been used successfully to remove noise in complicated datasets, enable easier clustering, and allow simplified visualization of high-dimensional data.

Here, our goal is primarily the first of the three, where we hope to remove noise in the data and streamline the original featurespace into a smaller one. We will then run this transformed dataset through a classifier and compare its results to other cutoffs for number of components, and to the original. This will be discussed in detail in the following sections.

Ideally, our results would show the same or better measures with a smaller feature number. These results can be somewhat subjective based on a researcher’s priority in one statistical measure over another. We will evaluate feature importance by assessing the impact each feature has on a principal component.

3.3 PCA Derivation

As mentioned in **3.1**, the dataset was first normalized with Z-Score normalization. This was done for two reasons. First, the features were normalized with respect to their standard deviation and mean so that they can be compared on the same scale. With differing mean or standard deviation, the eigenvector decomposition would prioritize variation in features with greater mean and greater standard deviation. Additionally, the features being set to zero mean makes the covariance calculation far more direct, faster, and requiring less processing power. This point is extremely important for larger datasets.

The actual derivation of PCA is finding the Principal Components and associated Explained Variances. We then order the eigenvectors in declining order according to the associated size of their explained variance. From there, we limit the Principal Components according to desired summed explained variance as a percentage of the complete explained variance. Then, these pruned components can be used to directly transform the original data into the projection space. Here, we use the same projection derived from Training to transform both Training and Test data. The exact steps are enumerated below:

Training Set:

1. Find Covariance Matrix of Featureset (Grid of covariances between original factors, here, 58x58)
2. Compute Eigenvectors and Eigenvalues from this matrix
3. Arrange Eigenvectors in declining order according to associated Eigenvalue (Vertical Ordered Arrangement of Eigenvectors is PC Matrix)
4. Compute ordered arrangement of Eigenvalues as a percentage of sum of Eigenvalues (Explained Variance Ratios)
5. Based on input cutoff C%, e.g., 70%, Find minimum number of eigenvalues such that the sum is >=C% (result is N)
6. Prune PC Matrix to only take the first N vectors (Here, 58 x N grid)
7. Transform original featureset by taking the dot product of [Featureset, Pruned PC Matrix], with output M samples by N components
8. Append associated y values to matrix to form New DataSet: M x N+1

Testing Set:

1. Take Pruned PC Matrix from Training Set
2. Transform original featureset by taking the dot product of [Featureset, Pruned PC Matrix], with output M’ samples by N components
3. Append associated y values to matrix to form New DataSet: M’ x N+1

The above steps 1-4 was implemented once and used to derive the PC matrix and associated New Datasets (steps 5-8 and B.1-3) for each explained variance ratio cutoff studied. These seven pairs of transformed datasets were compared among themselves and with the original normalized Training and Testing datasets.

3.4 Experimentation Methods

Different structures and parameters to experiment with include:

- ❏ Total Explained Variance Ratio Cutoff (.7, .75, .8, .85, .9, .95, .999)
- ❏ Threshold Values for Prediction Rounding (.4, .45, .475, .5, .525, .55, .6)
- ❏ Grid Search for Classifier Parameters:
- ❏ Bernoulli Naïve Bayes : Alpha: {0, .5, 1, 2}
- ❏ Decision Tree : Min Samples to Split: {20, 80}, Min Samples for Leaf: {20, 80}, Max Depth of Tree: {4, 8}
- ❏ Logistic : Regularization parameter C: {.1, 1, 10}

Architecture: PCA EVR Cutoff : We varied the cutoff for pruned explained variance percentage from 70 to near 100% in 5% increments to study the effect of removing additional components, and specifically to test whether there was useful information encoded in these low variance features.

Architecture: Threshold Values for Predictions : We varied the threshold values for prediction of class 1 (and therefore class 0 implicitly) in order to demonstrate the range of the algorithm in predicting the class of a data point within different set of priorities, e.g. priority to class 1 accuracy over class 0.

Models: Grid Search : To test which hyper parameters are best, we use GridSearchCV, which simply runs the model at each combination of hyper parameters, and compares the average validation ROC Area Under the Curve Score under cross validation at each parameter combination. This is done independently for each model and dataset.

Model 1: Bernoulli Naïve Bayes Classifier : We chose Bernoulli Naïve Bayes' Classifier as a model because it generally does a good job of classifying binary data, and it runs extremely quickly with only one parameter to tune, alpha.

Model 2: Decision Tree Classifier : We chose Decision Tree Classifier as a model because it is a tree-based and potentially intricate model that can extract hidden information from the training sets that might not be picked up by Bernoulli or Logistic. Additionally, we hypothesize that it might work well with a transformed dataset like PCA. We vary Min Samples needed to create a leaf and to split a branch to test different implicit regularizations. We vary Max Depth to test overfitting of the data explicitly.

Model 3: Logistic Classifier : We chose Logistic Classifier as a model because similarly to Bayes' Classifier, it generally does a great job for binary data, it has a simple model with only one hyper parameter to test. We vary C values at .1, 1, and 10 to test different regularization amounts.

3.5 Results

After running PCA at different explained variance cutoff levels, we found the number of Principal Components (PC's) explained by each of the cutoffs below. Four of the components do not explain any of our data. Fifty-five of the components can fully explain our dataset (down to a grain size of 11 decimals). We chose our target cutoffs to be at 85% to 90% since the feature space can be reduced by half with a disproportionately small reduction in explained variance. A very small explained variance per component implies that it is mostly noise. We verify this assumption in our comparison of different cutoff values and their associated classification measures.

feature num	cutoff
59	
55	1
51	0.999
36	0.95
30	0.9
26	0.85
23	0.8
20	0.75
18	0.7

Figure 6. Number of Features Explained by Cutoff Level

Based on the explained variance, we found the below basic features to have the highest significance in predicting our target label. This is computed for each basic feature by taking the coefficient for each PC in a weighted sum with the explained variance ratio for that component, across all principal components.

Description	feat	importance	% of total
8. num_self_hrefs: Number of links to other articles published by Mashable	num_self_hrefs	0.109	2.7%
22. kw_min_max: Best keyword (min. shares)	kw_min_max	0.107	2.7%
25. kw_min_avg: Avg. keyword (min. shares)	kw_min_avg	0.107	2.7%
26. kw_max_avg: Avg. keyword (max. shares)	kw_max_avg	0.106	2.6%
39. LDA_00: Closeness to LDA topic 0	LDA_00	0.104	2.6%
14. data_channel_is_entertainment: Is data channel 'Entertainment'?	data_channel_is_entertainment	0.104	2.6%
37. weekday_is_sunday: Was the article published on a Sunday?	weekday_is_sunday	0.101	2.5%
9. num_imgs: Number of images	num_imgs	0.097	2.4%
5. n_non_stop_words: Rate of non-stop words in the content	n_non_stop_words	0.097	2.4%
31. weekday_is_monday: Was the article published on a Monday?	weekday_is_monday	0.094	2.4%
6. n_non_stop_unique_tokens: Rate of unique non-stop words in the content	n_non_stop_unique_tokens	0.094	2.3%
27. kw_avg_avg: Avg. keyword (avg. shares)	kw_avg_avg	0.093	2.3%
28. self_reference_min_shares: Min. shares of referenced articles in Mashable	self_reference_min_shares	0.093	2.3%
32. weekday_is_tuesday: Was the article published on a Tuesday?	weekday_is_tuesday	0.092	2.3%
33. weekday_is_wednesday: Was the article published on a Wednesday?	weekday_is_wednesday	0.091	2.3%
12. num_keywords: Number of keywords in the metadata	num_keywords	0.091	2.3%
56. title_subjectivity: Title subjectivity	title_subjectivity	0.087	2.2%
34. weekday_is_thursday: Was the article published on a Thursday?	weekday_is_thursday	0.086	2.2%
11. average_token_length: Average length of the words in the content	average_token_length	0.084	2.1%
23. kw_max_max: Best keyword (max. shares)	kw_max_max	0.083	2.1%

Figure 7. Top 20 Features with Highest Significance

After finding our Principal Components, we created a new dataset using features that explain more than 70% of our data. We then trained this dataset on Bernoulli Bayes, Decision Tree, and Logistic Classifier using Sklearn and tested with the test data set aside earlier. We obtained weighted total accuracies around 60-63%, but up to 65.5% for Logistic Classifier. However, if we examine the F1 Score and Accuracy by target label, we observe a much higher rate for class label 1 than class label 0 (10-15% higher, see figure 8, 9 , and 10). This indicates that the Principle Components we selected are strong predictive features for the positive class: popular articles. Using a variance cutoff anywhere from 70% to 99.9% (increasing number of features) returned similar overall accuracies, with class 0 accuracies skewing downward with fewer components, implying variance contained in later components is more predictive of class 0 accuracies.

In comparison to the 90% Principal component model measures, measures from the full set of features will only give us a slight increase in accuracy (1-2%), with there being only a small difference in most models x threshold between 99.9% PC model and the base model. Unfortunately, we did not observe a noted increase in overall predictive capacity for any of the Principal Component Models in comparison to the associated Base model, however in many cases we achieved higher class 1 performance. A comparison of results at Prediction Threshold of 50% can be found below, with associated tables in Appendix and complete results to be found in attached “./PCA Files/excel_results.xlsx”.

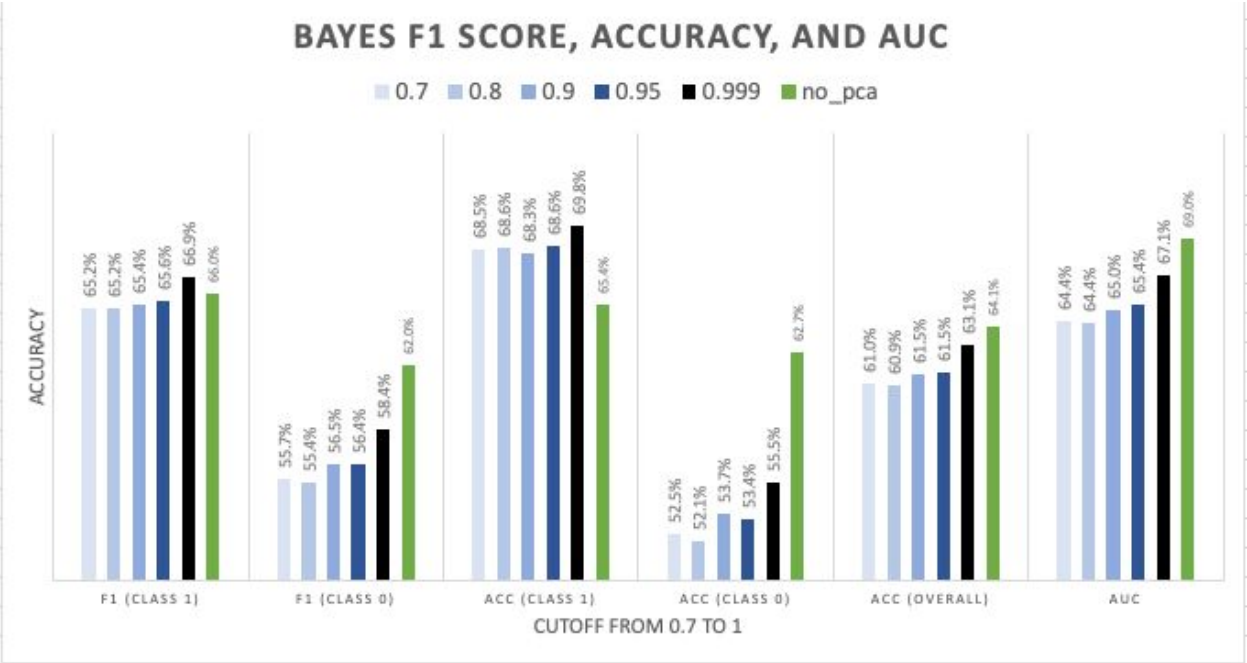

Figure 8. F Score, Accuracy, and AUC for Bayes Classifier by Variance Cutoff

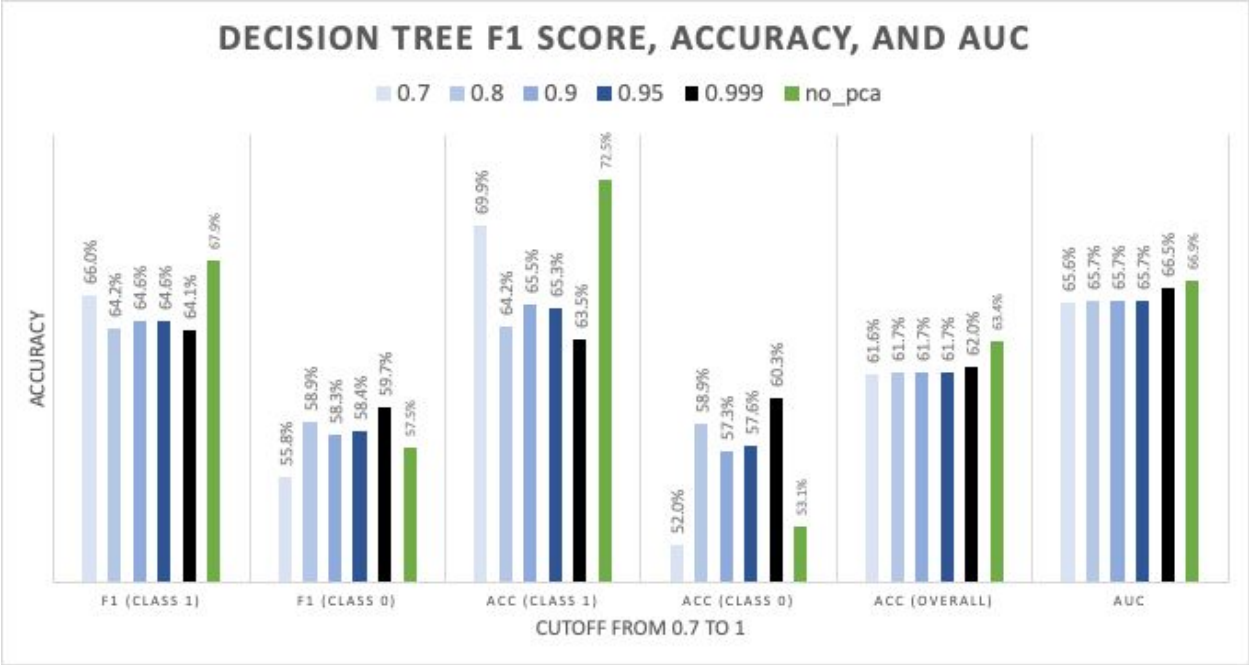

Figure 9. F Score, Accuracy, and AUC for Decision Tree by Variance Cutoff

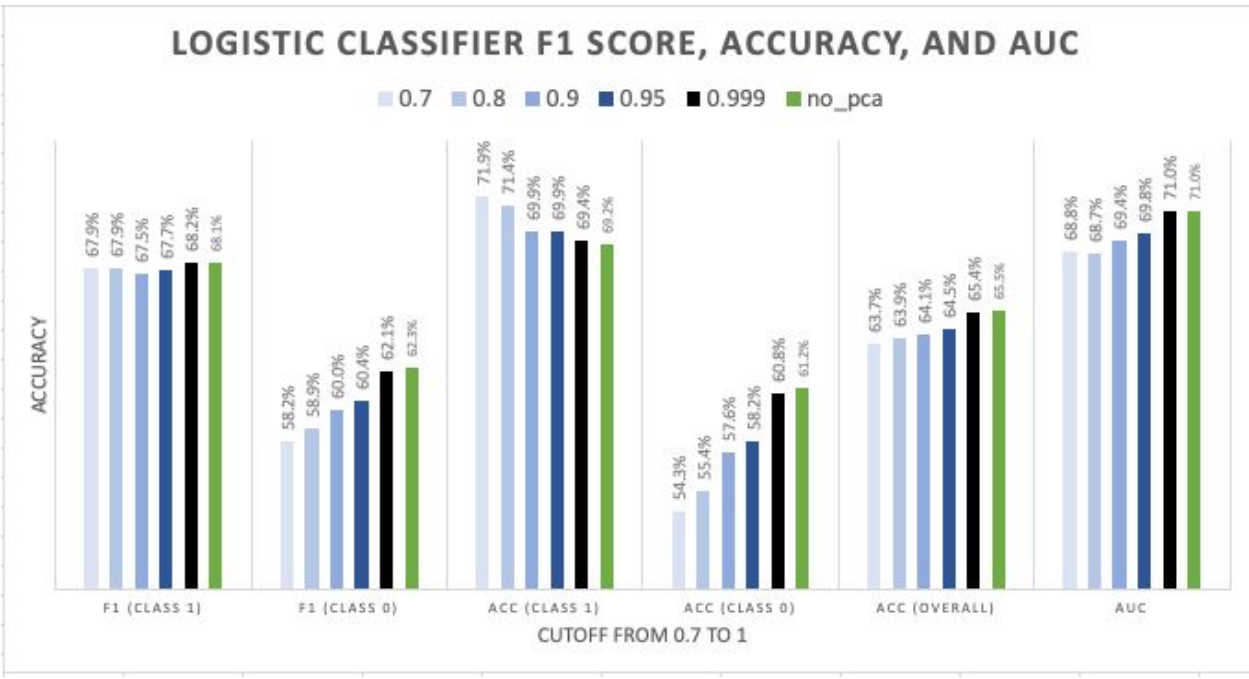

Figure 10. F Score, Accuracy, and AUC for Logistic Classifier by Variance Cutoff

We can see from the above results that despite overall accuracy and auc scores not generally increasing in any PC cutoff in comparison to the base data, scores are nearly identical when compared between 51 component cutoff and base data with 58 components. This tells us that there is a good amount of redundant data, and gives us a

starting point of 7 features for base data feature pruning. The feature importance calculated with PCA can be an additional guide to this process.

We gain additional information about the dataset from the PCA process in the form of the preference for class 1 in earlier components and implicit preference for class 0 in later components, and we learn there is more variance in class 1 prediction compared to class 0 prediction.

4. Final Thoughts and Further Exploration

This project was successful in that we were able to verify the results of implemented models in PCA and Neural Network by using our own understanding to directly build and implement these strategies. We were able to optimize these results by using different thresholds and parameters.

Despite all of this, our ending results cannot be defined as extremely good, with average accuracies around 65%, however our implemented Neural Net had similar results to SKLearn implemented Bayes, Decision Tree, and Logistic classifiers. Similarly, our PCA implementation had identical results to the SKLearn one.

Additionally, we can see areas to improve upon our accuracy measures, specifically by doing feature selection in the original feature set, or by further refining the parameters of the Neural Net or other Classifiers. Other means of improvement in classification could be to make the output multi-class and deciding boundaries more methodically, for instance with clustering.

5. Appendix

A. Table for Bayes Classifier Results, pictured in Section 3.5, Figure 8 (thresholds .45, .5, .55 at cutoffs .7, .75, .8, .85, .9, .95, .999 and non-pca)

	Test:	f1	f0	acc1	acc0	acc	auc	Train:	f1	f0	acc1	acc0	acc	auc
0.45_0.7	18	67.9%	49.2%	78.1%	40.8%	60.7%	64.4%		68.2%	48.7%	79.0%	40.0%	60.8%	64.0%
0.45_0.75	20	68.1%	49.5%	78.3%	41.0%	60.9%	64.4%		68.2%	48.7%	78.8%	40.0%	60.7%	64.1%
0.45_0.8	23	68.1%	49.2%	78.3%	40.8%	60.8%	64.4%		68.2%	48.8%	78.8%	40.1%	60.7%	64.1%
0.45_0.85	26	68.4%	50.2%	78.4%	41.8%	61.3%	64.7%		68.3%	49.3%	78.7%	40.7%	61.0%	64.4%
0.45_0.9	30	68.4%	50.6%	78.0%	42.4%	61.4%	65.0%		68.1%	50.0%	78.0%	41.7%	61.1%	64.9%
0.45_0.95	36	68.6%	51.5%	77.9%	43.4%	61.8%	65.4%		68.3%	50.3%	78.1%	42.0%	61.3%	65.2%
0.45_0.999	51	69.1%	53.1%	78.1%	45.2%	62.8%	67.1%		68.7%	52.3%	77.7%	44.5%	62.2%	66.3%
0.45_0	no_pca	67.5%	60.5%	69.3%	58.7%	64.3%	69.0%		67.3%	59.4%	69.8%	56.9%	63.8%	68.4%
0.5_0.7	18	65.2%	55.7%	68.5%	52.5%	61.0%	64.4%		65.6%	55.1%	69.6%	51.3%	61.0%	64.0%
0.5_0.75	20	65.3%	55.6%	68.7%	52.3%	61.0%	64.4%		65.6%	55.2%	69.6%	51.4%	61.1%	64.1%
0.5_0.8	23	65.2%	55.4%	68.6%	52.1%	60.9%	64.4%		65.6%	55.2%	69.5%	51.4%	61.0%	64.1%
0.5_0.85	26	65.4%	55.4%	68.9%	51.9%	61.0%	64.7%		65.8%	55.6%	69.6%	51.9%	61.4%	64.4%
0.5_0.9	30	65.4%	56.5%	68.3%	53.7%	61.5%	65.0%		65.6%	56.2%	68.8%	53.0%	61.4%	64.9%
0.5_0.95	36	65.6%	56.4%	68.6%	53.4%	61.5%	65.4%		65.8%	56.5%	69.0%	53.3%	61.7%	65.2%
0.5_0.999	51	66.9%	58.4%	69.8%	55.5%	63.1%	67.1%		66.0%	57.3%	68.8%	54.4%	62.1%	66.3%
0.5_0	no_pca	66.0%	62.0%	65.4%	62.7%	64.1%	69.0%		65.9%	61.0%	66.0%	60.9%	63.6%	68.4%
0.55_0.7	18	60.6%	59.7%	57.4%	63.3%	60.1%	64.4%		60.7%	59.4%	57.9%	62.7%	60.1%	64.0%
0.55_0.75	20	60.4%	59.5%	57.2%	63.2%	60.0%	64.4%		60.5%	59.2%	57.6%	62.5%	59.9%	64.1%
0.55_0.8	23	60.4%	59.3%	57.4%	62.8%	59.9%	64.4%		60.6%	59.2%	57.8%	62.4%	59.9%	64.1%
0.55_0.85	26	60.4%	59.3%	57.4%	62.7%	59.9%	64.7%		60.8%	59.5%	58.0%	62.7%	60.2%	64.4%
0.55_0.9	30	60.6%	60.3%	57.1%	64.3%	60.4%	65.0%		61.0%	60.0%	57.8%	63.6%	60.5%	64.9%
0.55_0.95	36	61.3%	60.5%	58.0%	64.1%	60.9%	65.4%		61.4%	60.3%	58.4%	63.7%	60.8%	65.2%
0.55_0.999	51	62.4%	61.4%	59.3%	65.0%	61.9%	67.1%		62.0%	61.0%	58.9%	64.5%	61.5%	66.3%
0.55_0	no_pca	64.5%	63.2%	61.5%	66.5%	63.9%	69.0%		64.2%	62.3%	61.8%	65.0%	63.3%	68.4%

B. Table for Decision Tree Classifier Results, pictured in Section 3.5, Figure 9 (thresholds .45, .5, .55 at cutoffs .7, .75, .8, .85, .9, .95, .999 and non-pca)

	Test:	f1	f0	acc1	acc0	acc	auc	Train:	f1	f0	acc1	acc0	acc	auc
0.45_0.7	18	68.1%	51.6%	76.9%	43.9%	61.5%	65.6%		70.4%	54.8%	79.8%	46.4%	64.2%	69.9%
0.45_0.75	20	68.3%	50.9%	77.8%	42.9%	61.5%	65.1%		70.5%	54.5%	80.3%	45.9%	64.2%	70.0%
0.45_0.8	23	68.4%	49.5%	79.0%	40.8%	61.1%	65.7%		70.7%	52.6%	81.8%	43.2%	63.8%	70.3%
0.45_0.85	26	68.3%	49.1%	78.8%	40.5%	60.9%	65.6%		70.7%	52.7%	81.8%	43.3%	63.8%	70.3%
0.45_0.9	30	68.0%	50.4%	77.6%	42.3%	61.1%	65.7%		70.5%	53.9%	80.5%	45.1%	64.0%	70.5%
0.45_0.95	36	67.9%	50.0%	77.5%	41.9%	60.9%	65.7%		70.6%	53.7%	80.9%	44.7%	64.0%	70.5%
0.45_0.999	51	67.1%	55.5%	72.3%	50.6%	62.2%	66.5%		69.6%	58.8%	75.0%	53.5%	65.0%	70.9%
0.45_0	no_pca	69.0%	55.8%	76.1%	49.2%	63.6%	66.9%		69.4%	55.9%	76.8%	49.1%	63.9%	67.6%
0.5_0.7	18	66.0%	55.8%	69.9%	52.0%	61.6%	65.6%		68.6%	59.0%	72.7%	55.0%	64.4%	69.9%
0.5_0.75	20	65.1%	56.2%	67.9%	53.5%	61.1%	65.1%		68.2%	60.1%	71.1%	57.1%	64.6%	70.0%
0.5_0.8	23	64.2%	58.9%	64.2%	58.9%	61.7%	65.7%		67.0%	62.4%	66.9%	62.5%	64.9%	70.3%
0.5_0.85	26	64.2%	58.5%	64.7%	58.0%	61.6%	65.6%		67.2%	62.2%	67.4%	61.9%	64.9%	70.3%
0.5_0.9	30	64.6%	58.3%	65.5%	57.3%	61.7%	65.7%		67.6%	62.0%	68.5%	61.1%	65.0%	70.5%
0.5_0.95	36	64.6%	58.4%	65.3%	57.6%	61.7%	65.7%		67.6%	62.1%	68.4%	61.3%	65.1%	70.5%
0.5_0.999	51	64.1%	59.7%	63.5%	60.3%	62.0%	66.5%		67.1%	63.0%	66.5%	63.6%	65.1%	70.9%
0.5_0	no_pca	67.9%	57.5%	72.5%	53.1%	63.4%	66.9%		68.4%	58.1%	73.1%	53.6%	64.0%	67.6%
0.55_0.7	18	61.1%	61.9%	56.7%	67.0%	61.5%	65.6%		63.5%	64.3%	58.8%	69.6%	63.9%	69.9%
0.55_0.75	20	61.5%	60.7%	58.2%	64.4%	61.1%	65.1%		64.4%	63.9%	60.7%	68.0%	64.1%	70.0%
0.55_0.8	23	63.3%	59.7%	62.1%	61.0%	61.6%	65.7%		66.2%	63.2%	64.6%	65.0%	64.8%	70.3%
0.55_0.85	26	62.5%	60.1%	60.4%	62.4%	61.3%	65.6%		65.5%	63.7%	63.1%	66.4%	64.6%	70.3%
0.55_0.9	30	62.9%	59.8%	61.4%	61.5%	61.4%	65.7%		66.1%	63.5%	64.1%	65.6%	64.8%	70.5%
0.55_0.95	36	62.9%	60.1%	61.1%	62.0%	61.5%	65.7%		66.1%	63.6%	64.2%	65.7%	64.9%	70.5%
0.55_0.999	51	60.8%	62.6%	55.7%	68.7%	61.7%	66.5%		63.8%	65.3%	58.6%	71.4%	64.5%	70.9%
0.55_0	no_pca	64.4%	60.6%	63.3%	61.7%	62.6%	66.9%		65.6%	61.6%	64.8%	62.4%	63.7%	67.6%

C. Table for Logistic Classifier Results, pictured in Section 3.5, Figure 10 (thresholds .45, .5, .55 at cutoffs .7, .75, .8, .85, .9, .95, .999 and non-pca)

	Test:	f1	f0	acc1	acc0	acc	auc	Train:	f1	f0	acc1	acc0	acc	auc
0.45_0.7	18	69.8%	52.1%	80.2%	43.2%	62.9%	68.8%		70.0%	51.5%	81.0%	42.2%	62.9%	68.0%
0.45_0.75	20	69.7%	52.3%	79.9%	43.6%	63.0%	68.7%		70.0%	52.0%	80.8%	42.8%	63.1%	68.1%
0.45_0.8	23	69.4%	52.5%	79.0%	44.2%	62.8%	68.7%		69.8%	52.3%	80.2%	43.5%	63.1%	68.4%
0.45_0.85	26	69.6%	53.0%	79.2%	44.6%	63.1%	68.9%		69.9%	52.6%	80.2%	43.7%	63.2%	68.7%
0.45_0.9	30	70.1%	55.3%	78.7%	47.5%	64.2%	69.4%		69.9%	54.3%	78.9%	46.3%	63.7%	69.3%
0.45_0.95	36	70.1%	55.9%	78.4%	48.4%	64.4%	69.8%		70.0%	55.2%	78.6%	47.5%	64.1%	69.7%
0.45_0.999	51	70.4%	58.0%	77.5%	51.3%	65.3%	71.0%		70.2%	57.2%	77.5%	50.4%	64.8%	70.8%
0.45_0	no_pca	70.1%	57.9%	76.9%	51.5%	65.1%	71.0%		70.2%	57.4%	77.5%	50.6%	65.0%	70.8%
0.5_0.7	18	67.9%	58.2%	71.9%	54.3%	63.7%	68.8%		68.1%	57.6%	72.8%	53.0%	63.6%	68.0%
0.5_0.75	20	67.8%	58.6%	71.5%	54.9%	63.8%	68.7%		68.1%	58.1%	72.5%	53.9%	63.8%	68.1%
0.5_0.8	23	67.9%	58.9%	71.4%	55.4%	63.9%	68.7%		67.9%	58.4%	71.8%	54.5%	63.7%	68.4%
0.5_0.85	26	68.0%	58.8%	71.8%	55.1%	64.0%	68.9%		68.1%	58.8%	72.1%	54.9%	64.1%	68.7%
0.5_0.9	30	67.5%	60.0%	69.9%	57.6%	64.1%	69.4%		67.9%	60.2%	70.4%	57.6%	64.4%	69.3%
0.5_0.95	36	67.7%	60.4%	69.9%	58.2%	64.5%	69.8%		67.9%	60.3%	70.3%	57.8%	64.5%	69.7%
0.5_0.999	51	68.2%	62.1%	69.4%	60.8%	65.4%	71.0%		68.4%	61.9%	70.0%	60.2%	65.4%	70.8%
0.5_0	no_pca	68.1%	62.3%	69.2%	61.2%	65.5%	71.0%		68.4%	62.0%	70.0%	60.4%	65.5%	70.8%
0.55_0.7	18	63.7%	63.6%	59.8%	68.1%	63.7%	68.8%		63.3%	62.6%	59.8%	66.5%	62.9%	68.0%
0.55_0.75	20	63.5%	63.6%	59.4%	68.3%	63.5%	68.7%		63.3%	62.9%	59.6%	67.1%	63.1%	68.1%
0.55_0.8	23	63.9%	63.5%	60.2%	67.7%	63.7%	68.7%		63.4%	63.1%	59.8%	67.2%	63.3%	68.4%
0.55_0.85	26	64.0%	63.8%	60.1%	68.3%	63.9%	68.9%		63.8%	63.2%	60.3%	67.1%	63.5%	68.7%
0.55_0.9	30	63.4%	63.9%	59.1%	68.9%	63.7%	69.4%		63.6%	64.0%	59.4%	68.9%	63.8%	69.3%
0.55_0.95	36	63.5%	63.8%	59.2%	68.8%	63.7%	69.8%		63.8%	64.1%	59.6%	69.0%	64.0%	69.7%
0.55_0.999	51	64.9%	65.0%	60.8%	69.7%	64.9%	71.0%		65.1%	65.0%	61.1%	69.6%	65.0%	70.8%
0.55_0	no_pca	64.8%	64.9%	60.6%	69.7%	64.9%	71.0%		65.1%	65.1%	61.0%	69.7%	65.1%	70.8%