Frank Antolino

Dr. Leeds

Machine Learning

Final Project

**I. Introduction**

A helpful indicator in determining how popular an online article is, is the number of times the given article was shared. Predicting the popularity of an article can be useful for authors of the content, advertisers, and site maintenance workers.

This project looks to use the “Online News Popularity” data set provided by the University of California, Irvine to make predictions regarding the number of times a web page will be shared. The data provided by UCI contains 60 features, (58 predictive), about each of the 39,797 instances of a web page. In addition to these features, each instance contains the number of times the web page was shared, which is what this project looks to investigate.

**II. Dataset and Feature Selection**

1. ***Data Collection, Processing, and Sampling***

This project uses data collected by *Mashable*, an online news site, and is csv formatted and provided by the University of California, Irvine. There is no data missing, that is, all instances of a news article have values for all of the given features.

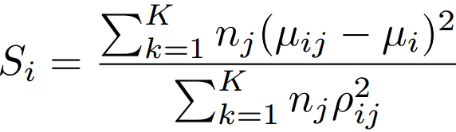
In order to separate the ‘number of shares’ attribute into two classes, a threshold of 1400 shares was used. An article was greater than or equal to 1400 shares is denoted as a 1, and anything less than 1400 is denoted as a 0. This binary representation of the class labels allows for use of both a logistic classifier and a SVM.

The data was then split into three categories; training data, testing data, and validation data. This project divides the data into groups of size 60%, 20%, and 20%, respectively. Because of the large number of data points, (39,797), using 60% of the data to train should be large enough to avoid overfitting and simultaneously allow for relatively large testing and validation sets.

***B. Feature Selection/Reduction***

In order to limit the number of features used in the model, Fisher Scoring was implemented. Fisher Scoring is a way to rank features in terms of how discriminatory they are likely to be. The Fisher Score calculation takes into account how far away the mean of the i’th feature in the j’th class is [µ\_ij] from the mean of the i’th feature across both classes [µ\_i], squared. This difference is weighted by the number of occurrences of the j’th class [n\_j]. As this squared and weighted difference increases, so too does the Fisher Score for a given feature [S\_i].

The score is penalized though, by the amount of squared variance in the i’th feature in the j’th class [(ρ^2)\_ij]. This value is also weighted by the number of occurrences of the j’th class [n\_j]. This formula can be more rigorously described below:



This project uses this idea to score each of the 59 features and in turn rank them in these descending order, based on these features. In theory, this list should represent the features that will be most informative to least informative. The results of these rankings are illustrated in the table below:

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| ***Rank*** | ***1*** | ***2*** | ***3*** | ***4*** | ***5*** | ***6*** | ***7*** | ***8*** | ***9*** | ***10*** | ***11*** | ***12*** | ***13*** | ***14*** | ***15*** |
| Feature Index | 25 | 39 | 16 | 36 | 12 | 14 | 34 | 15 | 41 | 5 | 23 | 35 | 38 | 43 | 24 |
| Score | .0264 | .0253 | .0223 | .0198 | .0148 | .0122 | .0120 | .0103 | .0088 | .0085 | .0084 | .0067 | .0065 | .0054 | .0052 |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| ***Rank*** | ***16*** | ***17*** | ***18*** | ***19*** | ***20*** | ***21*** | ***22*** | ***23*** | ***24*** | ***25*** | ***26*** | ***27*** | ***28*** | ***29*** | ***30*** |
| Feature Index | 37 | 28 | 7 | 10 | 47 | 42 | 26 | 44 | 27 | 4 | 40 | 19 | 17 | 2 | 55 |
| Score | .0051 | .0048 | .0047 | .0047 | .0047 | .0045 | .0036 | .0036 | .0035 | .0032 | .0030 | .0028 | .0026 | .0024 | .0024 |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| ***Rank*** | ***31*** | ***32*** | ***33*** | ***34*** | ***35*** | ***36*** | ***37*** | ***38*** | ***39*** | ***40*** | ***41*** | ***42*** | ***43*** | ***44*** | ***45*** |
| Feature Index | 31 | 50 | 1 | 46 | 6 | 57 | 18 | 30 | 21 | 11 | 54 | 48 | 49 | 32 | 9 |
| Score | .0023 | .0022 | .0021 | .0019 | .0016 | .0016 | .0015 | .0014 | .0012 | .0012 | .0009 | .0009 | .0008 | .0007 | .0006 |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| ***Rank*** | ***46*** | ***47*** | ***48*** | ***49*** | ***50*** | ***51*** | ***52*** | ***53*** | ***54*** | ***55*** | ***56*** | ***57*** |
| Feature Index | 45 | 29 | 22 | 3 | 20 | 33 | 52 | 8 | 53 | 51 | 13 | 56 |
| Score | .0005 | .0003 | .0003 | .0002 | .0001 | .0001 | .0000 | .0000 | .0000 | .0000 | .0000 | .0000 |

After this ranking, the next step is choosing how many of these features to keep and how many to disregard. This project implements a logistic classifier [see section III] to help with this determination. The classifier’s weight vector was learned on the training data and its accuracy was tested on the testing data, starting with using only the highest ranked feature, and subsequently adding the next highest ranked feature. The function then selects the subset with the highest accuracy, to later test on the validation data while varying hyper-parameters.

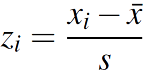
While all possible subsets could have been tested to find the literal best subset for the testing data, this process would have taken a significant amount of time. To avoid this large runtime, the feature selection function [featSelectFisher] assumes the Fisher Scores are at least somewhat indicative of level of discrimination. Therefore starting with the highest scored feature and adding one feature at a time, then selecting the highest accuracy from this list allows for a function that runs in O(n) | n= Number of features; and should in theory provide a classifier that uses the most informative features to predict class labels.

The results of these processes are shown below:

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| ***Rank*** | ***1*** | ***2*** | ***3*** | ***4*** | ***5*** | ***6*** | ***7*** | ***8*** | ***9*** | ***10*** | ***11*** | ***12*** | ***13*** | ***14*** | ***15*** | ***16*** | ***17*** | ***18*** | ***19*** | ***20*** |
| Feature Index | 25 | 39 | 16 | 36 | 12 | 14 | 34 | 15 | 41 | 5 | 23 | 35 | 38 | 43 | 24 | 37 | 28 | 7 | 10 | 47 |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| ***Rank*** | ***21*** | ***22*** | ***23*** | ***24*** | ***25*** | ***26*** | ***27*** | ***28*** | ***29*** |
| Feature Index | 42 | 26 | 44 | 27 | 4 | 40 | 19 | 17 | 2 |

*(i.e. including the 30th  best ranked feature reduced performance)*

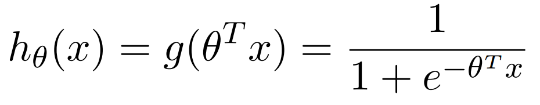
The data is then normalized using z-scores. This puts the data in terms of the number of standard deviations a given point is from the mean, for a given feature. To do this, the following formula is applied to each data point, where **X\_i -  x̄** gives the distance from each point X\_i, from the mean of the feature  **x̄**. This is then adjusted by dividing by the features standard deviation, S, to get the number of standard deviations each point is away from the mean. For example, a z-score of 0 means the point is equal to the mean. This is more formally illustrated below:



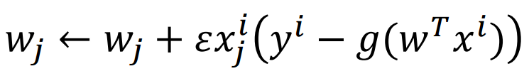
**III. Classifiers**

1. ***Logistic Classification***

The first classifier used in this project was a logistic classifier. This classifier uses the following sigmoid function, which constrains output to be between 0 and 1. The result can be thought of as a ‘pseudo’-probability of the data point belonging to class 1, although the numbers outputted do not reflect actual probabilities.



The classifier then continually updates the weight vector [θ] to more accurately allow features to influence the classification. This is done using gradient descent by implementing the following formula, where y is the actual class label, g(wTx) is the previously defined sigmoid function, (This time using w instead of θ to represent the weight vector), and x is the given data point:

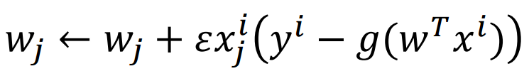


ε is a hyper-parameter used to adjust how drastically the weight vector is adjusted with each iteration. One of the things this project looks to manipulate is this value. A larger ε means that each pass through the data will correct the error (the y-g) more profoundly. This could mean less iterations are needed to reach a meaning weight vector, but simultaneously it could mean that because the adjustments are larger each step, the final weight vector is not as finely tuned and accurate as it could be with perhaps a smaller ε and more iterations. Therefore, several different values will be tested.

As alluded to earlier, another hyper-parameter to be manipulated in this project is the number of loops, or iterations through the data. If this number is too small, there is the risk of not converging to a minimum amount of error. However, if this number is too large, there is a potential for overfitting of the data and so the accuracy when the model is run on new data may be compromised. Therefore, several values for this hyper-parameter will be used as well.

This project will also look to implement Ridge Regularization (also known as L2 Regularization) to its logistic classifier. This form of regularization in particular is useful in limiting a given feature from growing too large in weight. This, in turn, is useful in preventing overfitting, which can be a result of having too many features. The original data contains 59 predictive features, so even with the inclusion of feature selection via Fisher Scores, there will potentially be a relatively large number of features used in the model. Therefore, the inclusion of L2 Regularization will help to mitigate the potentially damaging effects of having too many parameters, namely, overfitting.

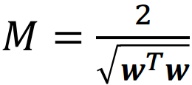
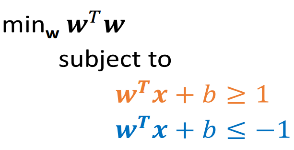
In addition to the two previously mentioned hyper-parameters the lambda value in L2 Regularization will also be manipulated. L2 Regularization uses the following variation of the gradient descent model previously defined:

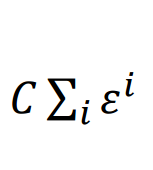


As lambda increases to infinity in the equation, the regularization term reaches 0. This is one value the project looks to investigate; essentially, how the model performs with no regularization. Smaller values of lambda will increase the regularization term and therefore increase how limited the values of the weight vector are in growing in magnitude. Because varying this value will result in varying accuracies of performance, the project will test different values for this hyper-parameter.

1. *Support Vector Machine [SVM]*

The second classification method used in this project was a Support Vector Machine [SVM]. This classifier tries to maximize a separating margin in the data between the classes, using “support vectors”. Support vectors are the data points that are used to help find this margin. One benefit to using only these support vectors as opposed to all of the data points is the fact that doing so prevents outlier data from skewing the margin. To do this, the following function is maximized (left), or minimized (right):

 Alternatively:

This function can work well, however, it is possible that in fitting the margin with the hopes of correctly classifying all data, a very narrow margin is created, and perhaps one that does not optimally separate the classes and can result in overfitting. Therefore, the inclusion of slack variables, or allowed misclassifications, is used to develop a larger boundary, at the expense of some improperly classified data points. This is implemented in the above equation through including the term .

The C value in this expression is one of the hyper-parameters of SVM that this project will look to explore. As the C value increases, the classifier will look for a tighter margin, and will not tolerate as many misclassifications. This tighter margin though, can lead to overfitting of the training data and consequently less accurate results on new data. As the C value decreases though, a larger margin that allows for some misclassifications will be produced. This can help prevent overfitting but can potentially result in a margin that does not separate the data as well. Because of this, this project will look at varying levels of C and compare the accuracies produced.

A second hyper-parameter to be altered during this project’s use of SVMs is the kernel function. A kernel function is a way to estimate a higher dimensional feature space in a more computationally efficient manner than actually computing the data coordinates in the higher dimension. Different kernel functions work better for different data sets and so this project will look to explore five different functions and compare the results of each.

The third hyper-parameter this project will look to explore is the iteration limit of the SVM. This will put a limit of the amount of iterations the SVM will use and stop building the model if this limit is reached, even if it has not yet converged. Different values for this hyper-parameter will be explored and their effects on accuracies compared.

**IV. Results**

*(All accuracies are based on validation data, which is a randomly selected subset of the total data and was not used in the making of the models)*

1. ***Logistic Classification***

Varying epsilon in the logistic classifier, while keeping the number of iterations equal to 10 and not including L2 Regularization (i.e. lambda = inf), produced the following accuracies:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Epsilon Value** | .01 | .02 | .03 | .04 | .05 |
| **Accuracy %** | 62.1642 | 62.1768 | 62.202 | 62.202 | 62.202 |

Increasing the value of epsilon seemed to have very little effect on the classifier’s accuracies. Perhaps in the future, a larger range of values can be tested to see whether or not this could improve future accuracy.

Now, varying the lambda in L2 regularization, but keeping the number of iterations equal to 10 and epsilon equal to .01, the classifier produced the following accuracies:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Lambda Value** | 100 | 50 | 30 | 10 | 5 |
| **Accuracy %** | 62.3282 | 61.3949 | 60.7391 | 58.2167 | 56.5519 |

As lambda decreased, (more regularization), the classifier performance decreased. The best accuracy, 62.3282, is higher than in the previous table where lambda was infinity. Therefore, this may suggest that some regularization is beneficial, but perhaps the lambda values chosen here were too high and values greater than 100 could be used in the future.

Next, the project kept epsilon at .01 with no regularization (lambda equal to infinity), and altered the number of iterations through the data. With this, the classifier yielded the following the accuracies:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Number of Iterations** | 5 | 10 | 25 | 50 | 100 |
| **Accuracy %** | 62.1642 | 62.1642 | 62.1894 | 62.202 | 62.2147 |

The accuracy of the classifier did not change significantly as the number of iterations through the data was altered. Perhaps some more significant changes could have been observed had values less than 5 been chosen, as values greater than 5 did not seem to change much. Having less than 5 iterations could have potentially resulted in better results, if by the 5th iteration there was an increase in overfitting. However, at the same time, less than 5 iterations may have resulted in worse results if less than 5 iterations was not enough to properly learn the weight vector. Therefore, in the future a wider range, perhaps starting at 1 iteration, can be used to gather some more meaningful insight.

1. ***Support Vector Machine [SVM]***

The first hyper-parameter this project varied in SVM’s was the kernel function. For these models, no iteration limit was placed and a C-value of .001 was kept constant throughout. This produced the following accuracies:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Kernel Function** | Linear | RBF | Quadratic | Cubic | Quartic |
| **Accuracy %** | 62.177 | 63.224 | 64.031 | 46.652 | 46.324 |

Interestingly, but not entirely unexpected, is the fact that varying the kernel function varied the accuracies greatly. Using a cubic and quartic kernel function performed extremely poorly (less than 50% accuracy with two classes). However, using linear, RBF, and quadratic functions all performed relatively well, around 63% accuracy, which is right around the best accuracies throughout the project. The rest of the SVM models in this project keep the kernel function set at RBF, however, since quadratic performed slightly better here, it’d be interesting to vary the rest of the hyper-parameters using a quadratic kernel function and see if this improves the accuracies produced.

The second hyper-parameter varied in this project was the iteration limit. As alluded to earlier, the RBF kernel was used throughput these models, and a C-value of .001 was kept constant. This yielded the following accuracies:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Iteration Limit** | 1e4 | 1e5 | 1e6 | 1e7 | 1e8 |
| **Accuracy %** | 63.224 | 53.676 | 53.676 | 53.676 | 53.676 |

Having the iteration limit set to 1e4 produced the highest accuracy, with all other (larger) values producing the same, lower accuracies. This can potentially be due to the fact that including a high number of iterations through the data can lead to overfitting, and so once the iteration limit was set to 1e5 and larger, the model was overfit on the training data and accuracies on the validation data were hindered. It would be interesting in the future to look at iteration limits less than 1e4 to observe how the model behaves with a lower number of iterations, and find out at what point does lowering the number of iterations become detrimental to the model.

The third and final hyper-parameter this project varied was the C-value in addressing slack variables. While varying this, the kernel function, RBF, was kept constant, and no iteration limit was placed on building the model. The SVM yielded the following accuracies:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **C-Value** | .0001 | .1 | 1 | 10 | 100 |
| **Accuracy %** | 53.676 | 55.341 | 63.438 | 61.546 | 61.471 |

Varying the value of C produced a range of results. Interestingly, when a small C was chosen, (.0001 and .1), and therefore a larger margin produced, the classifier performed relatively poorly. This is likely because the model allowed for too many misclassifications and therefore had a difficult time properly classifying the new (validation) data. The opposite seems to be true when C was too large though, (10 and 100), as the classifier did not allow for many misclassifications and chose a tighter margin. This may have resulted in overfitting the training data and in turn a poorer classification of the validation data. A C value of 1 performed the best here, as it seems to have balanced the influence of slack variables well. In the future, exploring C values more closely around 1, say from .5 to 5, can give insight into optimizing the selection of C for this data and model.

**V. Conclusion**

The best performing model on this data set was the project’s SVM with a quadratic kernel function, a C-value of .001, and no iteration limit. This yielded an accuracy of 64.031%. Seeing as setting an iteration limit of 1e4 yielded similar results (~63%), it would an interesting experiment to try combining these hyper-parameter selections, and having, for example, a quadratic kernel function with an iteration limit of 1e4, to see if this improves the model’s accuracy.

Overall, an accuracy of ~64% is less than ideal given a two class classification, however, it is indicative of the fact that some information was in fact useful in predicting the number of shares of the news articles. In the future, perhaps a different form of feature selection/dimensionality reduction could result in improved performance. PCA could be a beneficial method to use as it could help to reduce some of the correlation between features that Fisher Scoring does not deal with. This could be substantial, especially given the fact that 59 features were used, since it is likely that among these features, several of them were correlated. Repeating this project and using PCA instead of Fisher Scoring would be an interesting area for future research and could help gain insight into the importance of a given feature.

There is much more than can be done to improve the models outlined in this project and certainly room for improvement. However, as a preliminary experiment this project is useful in pointing future research in the right direction.