Introduction to Machine Learning

CS-UY 4563

Final Project Write-up

Fake News and Disinformation Filter

Johnlouis Dahhan

December 1, 2020

**INTRODUCTION**

**The Problem:**

In the past few years, the issue of so called “fake news” has become a huge problem in the American political sphere. Increasing numbers of the population have been radicalized by abrasive opinion pieces, misleading articles, and increasingly, blatant lies. This has made its way to the highest offices in the land, with prominent politicians parroting falsehoods and turning scientific consensus into topics for debate. Moderation is difficult, and few seem to trust massive social media conglomerates with the task of defining the bounds of acceptable political discussion. The CEOs of Google, Facebook, and Twitter have all testified before congress, and much of the discussion surrounded their role in moderating misinformation. I am sure the methods that these companies use to analyze misinformation on their platforms are far beyond the scope of this class, but I still thought this problem would be an interesting one to analyze.

This seems to be a problem of classification rather than regression. While “fake news” is hardly a simple category to define, a social media platform is typically tasked with the choice of whether to block some article, and this is often a strict binary choice. I will attempt to use unsupervised analysis to see if we can create clusters that reveal some underlying trends in the articles, but I will be attempting to solve it as a binary classification problem when I apply the models. Another consideration could be the thresholds that we apply to the data. Some platforms may want to err on the side of caution when it comes to censorship, while others may want to avoid questionable articles even if it means getting some false positives. We will adjust the thresholds for every classification method we use and see how it affects the accuracy.

**Feature Engineering and Preprocessing:**

The features for this problem will exclusively be concerned with the text of the article, and not its source. At this point in time it is rather easy to set up websites incredibly quickly and with no accountability, and certain interest groups set up numerous outlets to disseminate disinformation. For this reason, I think it is not ideal to risk heavily weighing the source of an article, as this could result in the exploitation of the model by using unknown sources.

My own research has led me to believe that the best way to create features based on text would be to convert the training data into a dictionary of its most common words, and then for each of these words present in each article, its index in the dictionary will be incremented in the feature vector of the sample. Because of this, it will be difficult to graph the data as a function of its features, as each entry will have to have thousands of features. I am more interested in prediction than inference, but I will still analyze the feature weights to see which words seem to have the most influence on the classification of the article.

Sklearn makes it very easy to fit models to data, so the challenge for this problem is preprocessing the data and feature engineering. For every article in our training set we want to isolate each word and add it to a list. Once we have a list of every instance of every word, we want to create a dictionary that holds the word count, in order, of every word we have seen in the training set. This counter dictionary can be created easily enough using default libraries and most freshman computer science students could isolate and clean words from a sentence. The issue arises when we consider how we want to count our words. For one, should we count words like “if”, “the” and “an”? Probably not. These words are called stop words, and there are modules that can give us a list of stop words that we can use when filtering our data. Another question is that of different conjugations. Should “go” be a different feature than “going”? Again, probably not. The process of converting words to their most basic form, such as converting “going” to “go”, is called lemmatization. There are libraries that allow us to lemmatize words given their part of speech, and there are libraries that allow us to get a word’s part of speech. The final steps would be to populate and scale the features matrix. I started with each article’s counts of the 3000 most common words, but I tried the same experiment with fewer words to see if I was overfitting. Once we have a features matrix, we can just use an sklearn function to mean-center and unit scale every feature, and then we are ready to use the models.

I was able to create a function that created a dictionary of the most common words and populated the features matrix. This was harder than anticipated, mainly because of the time the imported functions seemed to take. The part of speech tagger was much slower than anticipated, so I had to rework my functions to optimize how words were tagged, but I eventually figured it out.

For every model I used, I tested varying regularization hyperparameters from 0.001 to 10 to see how they would change the accuracy. Please note that for the support vector machine and logistic regression models, this value represents C which is negatively correlated with the amount of regression, while for the neural network this value represents alpha, which is positively correlated with the amount of regression. I also adjusted the thresholds from 0.3 to 0.8, as I figured the variance across these values would reflect how confident the model is. If the accuracy barely changes from a threshold of 0.3 to 0.8, that must mean that it calculated most of its probabilities as very close to 0 or 100%. I tried to adjust the settings for each model to see what configuration would yield the most accurate results. When testing logistic regression, I attempted both LASSO and ridge regularization. For the support vector machines, I attempted to see how the models could use the polynomial, linear and radial basis function kernels to solve the problem. I measured different possible hidden layer sizes and configurations for the neural network models.

My results will be compiled in data sheets that can be found after my discussion of the results. I rounded to 5 significant figures, and if there are data points with seemingly fewer significant figures, it is because Excel automatically removes the zeroes at the end of a decimal, so assume that the remaining digits are zeroes. I will also be uploading the full excel spreadsheet this data is taken from, as well as the original Jupyter notebook so that this experiment can be easily verified.

**UNSUPERVISED ANALYSIS**

I was unsure of how to approach unsupervised analysis for a problem like this. I fully expected for the k-means algorithm to not find anything interesting. In a problem like this, it seems like the feature weights are the most important part. The main thing separating a spam filter or genre detecting algorithm from a misinformation filter is which words are weighed in which ways. This meant that I did not have high hopes for what k-means would find. Despite this, I decided to see if k-means could surprise me by separating the data into the classifications I was looking for. I also decided to see if the results would change depending on whether I scaled the features. I figured that the unscaled features might have some potentially predictive weight that biases the most popular words. So, for the scaled and unscaled data I performed k-means with 2 means and checked the accuracy of the clusters against the actual training labels. I also decided to see whether I could observe anything visually interesting when running k means in higher dimensions by graphing the result in 3d with each axis representing one of the 3 most common words. I ran k means for k values 2-10.

As expected, I did not find any interesting patterns. I was somewhat vindicated in my prediction that scaling the features would result in the algorithm being more likely to pick out trends. However, both were still very weak correlations. The unscaled features were clustered with a 58.04% accuracy compared to the original labels, as opposed to the scaled features which had an accuracy of 54.21%. The data graphed in 2 dimensions was not linearly separable, but there was almost a bimodal quality when observing the original labels of the sample I was using. The k-means algorithm never seemed to consistently pick up on this pattern, but it was also accounting for thousands of features that would be impossible to graph, so I am not surprised. Interestingly, the scaling did not seem to skew or distort the first three features. The shape of the graph itself before and after scaling was visually very similar, the only notable difference being the actual scale on the axes. Notably, despite running this algorithm with up to 10 means and always seeing the first 2 clusters graphed, I very rarely saw more than 4 clusters in the graphs of the first 3 features. I can imagine that this is because the data was simply not easily separable into so many clusters, so the algorithm ended up creating multiple tiny clusters. Sadly, I am incapable of performing the high dimensional analysis necessary to make a definitive statement about how the data was clustered.

**SUPERVISED ANALYSIS**

**Logistic Regression:**

I tried logistic regression with LASSO and ridge regularization, using C values of 0.001, 0.01, 0.1, 1, and 10. I also tried to see how the accuracy would change at different threshold values. The threshold values were 0.3, 0.4, 0.5, 0.6, 0.7 and 0.8. I also measured how the 5 most weighted features changed throughout the different regularization methods and values. The best model in this category, and in the entire experiment was the model with LASSO regularization and a C value of 1. This model consistently received a test accuracy above 96% across every threshold except for 0.8, implying a large degree of confidence in the classification. This was also the only model in the entire experiment that ever achieved a test accuracy above 97%, doing so across 4 different thresholds and peaking at a threshold of 0.5 where it reached 97.416%.

It was interesting watching how the regularization changed the accuracy and confidence of the models. As less regularization was applied, the models typically became more accurate and confident, no matter the regression method used. The ridge regression model with the most regularization (C = 0.001) had a training accuracy of 83% when the threshold was set to 0.2, but a training accuracy of 99% when the threshold was 0.5. This implies that many of the probabilities calculated were not close to 1 or 0, which means the model is less certain of where to classify even the training data. However, nowhere was the negative influence of underfitting clearer than in the high regularization tests with LASSO regularization. The highest level of regularization for this model was so aggressive that the feature weights were whittled down to 0 and test and training accuracies hovered around 50%, which would be what one would expect from guessing the same classification for every sample.

Some regularization was necessary to optimize the models and prevent overfitting. For the ridge regularization, the optimal C value was 0.01, at which point the model peaked with an accuracy of 96.748%. However, even though this performed the best of all the ridge models, there is a notably wide variance across threshold values when C was 0.01 compared to models with even less regularization. These models performed only slightly worse, within a margin of 0.2%, but they were much more confident, with the accuracies at a threshold of 0.2 being very close to the accuracy at 0.5. If I were using ridge regularization, I would probably use less regularization than 0.01. The mild overfitting seems like a low cost for the much higher degree of model confidence.

The overfitting was less notable with the LASSO regularization, as the best model was the one with the second highest C value, which was 1, and while overfitting past this resulted in a decrease in test accuracy, this was only a decrease of 0.7%. However, I would not use the least or no regularization in this case, as the variance across thresholds for the highest level of regularization did not seem notably lower than that of the better performing model.

I graphed the top 5 most heavily weighed features (taken getting the maximum 5 elements of the absolute values of the coefficients) over different C values to see how they would behave as regularization decreased. The most interesting result was that when C was 0.001, the LASSO regularization model had all its weights set to 0. I expected the LASSO regularization to be less powerful, but I suppose I was wrong. Overall, the results were rather unsurprising, as expected the decrease regularization resulted in starker differences in the weights of the top 5 features. The feature that seemed to clue me in the most on how the model worked was the word “video” which was consistently in the top 3 or top 1 most heavily weighed feature. I can only assume this is because the presence of video evidence is heavily correlated with credible news.

**Support Vector Machines**

I tried support vector machines using the same regularization hyperparameters and threshold values as the logistic regression. I used 3 different kernels, namely the polynomial, radial basis function and linear kernels. For the polynomial kernel I tried degrees 3, 5, 7 and 9. I measured how the weights of the top 5 features changed throughout the different regularization values for the linear kernel, but did not for the other transformations, as sklearn does not allow for an easy extraction of the features for other kernels.

Between the varying levels of success using support vector machines and the high level of success when using logistic regression, it seems like this problem is rather linearly separable. The best performing kernel was the linear kernel with the most regularization. The C value was 0.001 and the accuracy at the standard threshold of 0.5 was 96.347%. Notably, this was worse than the logistic regression, which peaked at above 97%. Another interesting factor here was the influence of overfitting. Even with the most regularization, the training accuracy never reached below 99.5% across any threshold or any regularization value. In fact, for every regularization value besides the one with the most regularization, the training accuracy was 100% with no variance across thresholds, as the test accuracy continued to fall below 94%.

The worst performing kernel by far was the polynomial kernel. I was blown away by how terrible its performance was. The optimal regularization value seemed to be C = 10, and even then, the model seemed to overfit, with a training accuracy of 99.021% and a test accuracy of 81.246%. Any less regularization and the model struggled getting 50% accuracy on the training set, and any more regularization meant that the model could not even reach 90% accuracy for the training data. This was all conducted using a default degree of 3, so I decided to try other degrees and fixed the regularization parameter to 10. This was the most surprising result by far, as none of the models even came close to the degree 3’s mediocre accuracy of 81%. I would assume that applying more regularization would make the model perform better, as it would flatten out and potentially resemble the linear hyperplanes that the linear kernel and logistic models had great success using.

The radial basis function kernel did not seem to have anything interesting to observe. It did suffer from the same problem as the other kernels, where the intensity of regularization seemed to skew its results. Underfitting especially seemed to be a big problem for this kernel. When a C value of 0.001 to 0.1 was used, the model could not even reach a training accuracy of 93% or test accuracy of 87%. This is a significant overperformance compared to the abysmal polynomial kernel, but not so impressive when compared to the linear kernel. The C value 1 provided the best model with an accuracy of 95.634%, but any less regularization seemed to allow for overfitting, as the training accuracy increased to 100% while the test accuracy slightly decreased.

**Neural Network**

For neural networks, I again tried different regularization values (regularization was fixed at L2) and threshold values. This time, the regularization parameter represented the alpha parameter, which positively correlates to how much regression is applied. I also changed whether I used one or two hidden layers, and whether each layer had 100 or 1000 nodes. I guessed that more layers and nodes and less regularization would result in overfitting.

The results for this model were quite interesting. Firstly, each different network structure was able to reach 96% test accuracy, which surprised me as I expected there to be more variance in the results. Every model also fit the training data remarkably well, with all of them having their training accuracy at 100% when the regularization parameter was less than 10. There was remarkably little variance across threshold values for any of the models. None of the models had a change in either accuracy greater than 1% across any thresholds. This shows a remarkable level of confidence in the networks’ predictions, as almost all the probabilities calculated were below 0.2 or above 0.8. The best performing models were the network with 2 hidden layers with 1000 nodes each and a regularization of 0.01 and the network with 1 hidden layer with 100 nodes and a regularization of 0.1. This was incredibly surprising, as I expected to at least see at least a small degree of overfitting or underfitting depending on the number of nodes or layers, but I was pleasantly surprised to see how wrong I was. Both models were able to achieve a test accuracy of 96.703%, but this was less than 0.5% greater than the best performance of the worst scoring network structure. Overall, while it was unable to achieve an incredibly high accuracy, the neural networks were by far the most consistent models, and this could result in someone preferring it to the more finnicky logistic regression model.

**Different Feature Matrix Sizes**

It was very possible that I used too many features and was overfitting to my training data. Because of this, I decided to retry the best performing models with fewer features. I tried with 3000, 2000, 1000 and 500 features and measured how the best models’ performances changed. I fixed the threshold to 0.5, as it was consistently the best threshold. I expected that the original 3000 features would be too much, especially considering that we only have around 4500 samples in the dataset. I assumed we might be overfitting, and that if we decrease the number of features, we might get a higher test accuracy. I used the LASSO regularized logistic regression model with C = 1, the SVM linear kernel with C = 0.001, and the neural network with 1 hidden layer and 100 nodes with an alpha of 0.1. I chose this neural network over the one with 2 layers and 1000 nodes and the same accuracy because that one took a very long time to calculate.

The results were surprising. Only the support vector machine model ever increased in test accuracy as the number of features decreased. The support vector machine peaked in test accuracy at 1000 features with an accuracy of 96.080%, and seemingly started underfitting the data when there were only 500 features. The logistic regression and neural network models all decreased in accuracy, and while the logistic regression was still the most accurate with 3000 features, it performed the worst with 500 features. From this data it seems like 3000 features is generally a decent number of features. The support vector machine would probably be the best model if someone knows they only want 1000 features, while the neural network would probably be best if one is unsure about how many features they need, as it decreased the least overall. Despite all this, the logistic regression model was still the best model at 3000 features, although it had the steepest decline.

**TABULATED RESULTS**

Due to the difficulty of formatting Excel into Word, I will also be uploading the original spreadsheet with a few figures that are left out of this document.

**Unsupervised Analysis**







(Higher dimensional analysis can be seen in the Jupyter notebook, but I’m leaving it out here because it wasn’t particularly insightful and would take a lot of space)

**Logistic Regression**









Top 5 feature weights for Ridge Regression over values of C:







Top 5 feature weights for LASSO Regression over values of C:



(Note that the regularization was so aggressive that all weights were removed)





**Support Vector Machines**



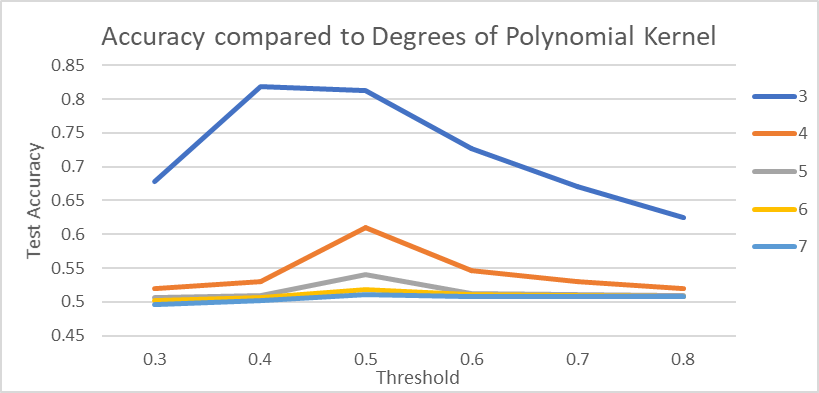






(C = 100 not pictured, but would be the lowest accuracy for both training and test)









**Neural Networks**











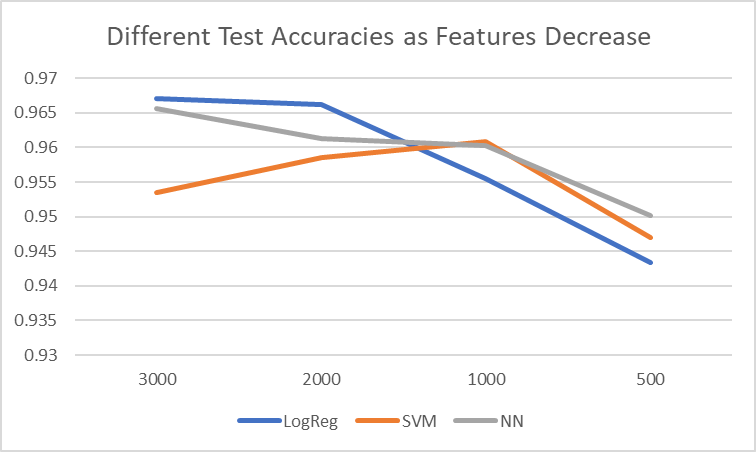






**Feature Matrix Resizing**



****

(Y = test accuracy, X = number of features)

**CONCLUSION**

This experiment was very illuminating. The biggest surprise was how accurate the filter was. I came into this project thinking that I would be lucky to get an accuracy of 80%, but this task was not as difficult as expected. The hardest part by far was parsing the dataset for features. It was difficult to discern what I could do alone and what I needed to import libraries for. Eventually, although my parsing was not perfect, I believe I exceeded what I expected of myself. I also was very surprised at the importance of maximizing efficiency. I originally created rather inefficient functions to handle modeling and feature preprocessing, but I quickly realized that I would never finish the experiment in time. I had to revisit and reoptimize my models several times before I arrived at something that could even run on my computer.   
 I was also surprised at how effective logistic regression was. When learning about these tools, I typically thought in 2 dimensions and wondered when we would ever find data that was clean and linearly separable. I always assumed problems like this required incredibly complex models to solve. However, the best performing model was by far the simplest, and when I tried to overcomplicate it by introducing kernel transformations, the accuracy only got worse. This was a relief, as I was questioning if I knew enough about machine learning to tackle a problem like this. I think this experiment has made me more confident in my own abilities and more confident in the effectiveness of these models that seem to work like magic.

I cannot help but be disappointed with the limited bounds that I had to constrain my experiment to. The original dataset had almost 45,000 articles, but I was forced to restrict it to 10% of its original size to get anything done in a reasonable amount of time. The models would have done much better had they been trained on ten times more data, and it is disappointed that I cannot see how good these models could get. The feature preprocessing also could have been improved. There was not nearly enough time or manpower invested in this project to experiment on which words or characters to filter to get the best models, and this must have hurt the accuracy in the long run. This project could also be further improved by implementing different classification methods. I have a very limited understanding of even the types of neural networks out there, let alone all the infinitely complicated classification methods available. Someone with a stronger understanding of the field could probably find an even stronger model than the logistic regression model that worked best in this experiment.

Overall, I am very pleased with this experiment. The work of moderating misinformation needs to be done, and I am glad that I was so successful in attempting it myself. I firmly believe that if I could create models as accurate as they were in the time that I had, social media conglomerates have a duty to the public to attempt to implement something similar. I now have more of an understanding of the kind of fine tuning necessary before deploying a model, but with the vast resources available to them, large tech companies should be able to figure out a consistent solution. I believe that machine learning is an incredible tool in the looming disinformation age, and I believe it needs to be utilized to its fullest potential.