571 Pattern Recognition Final Project: Fake News Challenge

Elliot Greenlee and Patricia Eckhart

May 6, 2017

Abstract

This report discusses the implementation and performance of numerous classification algorithms on a set of data comprised of 40,350 samples of headline and article pairs. 44 natural language processing features were extracted and normalized. Bayes Decision Rule was applied to the data under three different discriminant functions. The supervised learning algorithms k-nearest neighbors, backpropagation, decision tree, and support vector machine were implemented along with the unsupervised clustering algorithms k-means, winner takes all, and Kohonen maps. Classifier fusion was done using majority voting, naive Bayes, and behavior knowledge space algorithms. Our final results showed a highest accuracy of 88.0%, using behavior knowledge space fusion of our decision tree, backpropagation, support vector machine, and discriminant case two methods.

1 Introduction

For this project, we attempted a solution for Fake News Challenge Stage 1 (FNC-1): Stance Detection. The goal of this challenge is to obtain the highest score on a classification task: determine the relation between the headline and body text snippets of a news article. For each sample, 25% of the total score comes from determining if the headline and body text are related, while 75% comes from classifying the relationship as agreeing, disagreeing, or simply discussing.(Figure 1) This is just the first stage in the complex and cumbersome process of assessing the veracity of a news story [5].

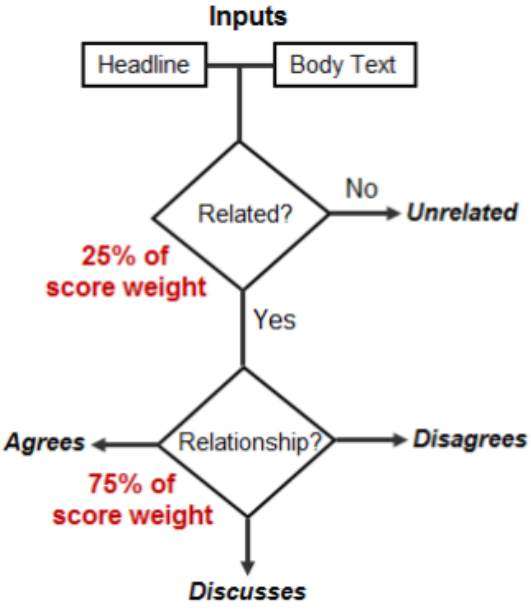

Figure 1: A description of the stance detection challenge.

Fake news has been much discussed in the last 12 months. It is a serious issue for the highly information driven world we live in, and has been of immense political bearing recently. 64% of U.S. adults in a Pew Research poll indicated that fakes news is causing confusion about the facts of current events[3]. As finger pointing about this issue begins, media giants like Facebook and Google have begun to respond. Educational initiatives from both companies are already underway, but more direct approaches like Google Fact Check, which tags search results using fact-checking groups, require the detection of stories which might be fake news. [12] Facebook uses humans to determine which stories might be fake, a task requiring untold hours. A first step in solving this challenging issue to perform stance detection on articles, giving human classifiers extra information on each suspected news story.

As part of this project we have been provided with around 40,000 classification samples with which to train our models.[11] These are combinations of headlines and body texts already hand classified. The challenge organizers have also provided a baseline feature extraction program. On June 1st, all registered teams will be provided with a set of testing data. After just 24 hours, each team must submit their results for evaluation. The baseline accuracy for this challenge is 79.53% using hand-coded features and gradient boosting with 10-fold cross validation.[13] Teams with a score greater than the baseline on the test data will compete for the top three places, which will receive prize money.

2 Background

Stance detection for natural language processing has been widely researched, although the specifics of its application to fake news classification is relatively new. Because of the complexity of working with discrete data such as words, and the challenge of turning the rich connotations and contexts of a language into simple classes, natural language processing tasks require many different techniques. The general pattern of preprocessing, feature extraction, and classification is followed, but with mixing between categories.

In order to begin the process of feature extraction, preprocessing must be performed on the language samples. Although it would seem natural to focus on the most frequent words in a sample, these are often common words which lend little to the meaning of a document. The first step in many natural language processing tasks is to remove these insignificant stop words in order to clarify the most meaningful data.[14] Other standard preprocessing steps include removal of punctuation and flattening to lowercase.

Because samples offer so much interrelated contextual data, the feature extraction process can be the most important aspect of achieving accuracy for a task. The N-gram model is the most basic. A gram represents a single word in the language, where an n-gram is multiple words in sequence. At the single gram level, word co-occurrence helps to find commonalities between two bodies of text. Additionally, targeted words from a certain category like political words or polarizing words can be counted.

Natural language processing often builds on previous work in the field in a way that uses classifications for future features. By analyzing various basic properties of the text such as n-grams, higher order data can be teased out of the original. At the most basic level, often these groups of words can provide increased accuracy when treated as features.[1] However, using n-grams for analysis to determine new features like tense, sentence voice, point of view, semantic orientation[10], and negated sentences can immensely increase the dimensions available for selection using methods like decision tree [9]. There is the danger of error carry-through from incorrect early classifications, but usually the accuracy of older techniques along with the reward of better semantic analysis is high enough to justify their use.

Because the features for natural language processing tasks can vary so widely, the classification techniques used must also cover a wide breadth, and rely on domain knowledge.[8] Stance detection is typically a supervised application of natural language processing; unsupervised classification is attraction for stance detection tasks where a corpus does not already exist, but since we are limited to a certain set of classes and data, supervised classification is the most likely place to start for our methodology.[2] Although typically parametric models converge faster and are more practical for classification tasks, non-parametric models are preferred for their weaker assumptions about the data. This is especially true in the world of natural language processing, where data is often organized discretely.[7]

Often, stance detection tasks require the use of multiple classification methods in tandem.[15] For example, the structure of our processing task lends itself to a multi-staged classification method. First, samples could be classified as either related or unrelated, and second, the related samples are put into the agree, disagree, or discuss category. One could further break this down into a "biased" versus neutral classification, followed by the final agree or disagree classification of the "biased" samples. Before using this cascading classifier, we must consider some assumptions about the data. The ambiguity of classification must be weighed against stage error. In the real world the majority of news articles and headlines match, meaning that a prior probability favoring this reality would lead to higher accuracy, and the first stage should have an easier decision to make. If the first stage method has a certain error, that error will carry through to the second stage. This error must be offset by removing ambiguity from the stage two samples, in order to increase the accuracy of that stage.[6] Another way to decrease this ambiguity is to use classifier fusion. A set of features that confuses one classifier may not confuse another, and vice versa, such that the combination of two classifiers produces the highest accuracy.

3 Technical Approach

3.1 Feature Extraction

The first steps in feature extraction from the text were preprocessing the original pairs. First, case was normalized and non-alphanumeric characters were removed. Next, a set list of stop words was applied to clarify important words. Word overlap features were extracted, as well as binary co-occurrence of char and n-grams. Refuting and polarizing word frequency, was counted and compared using words like 'doubt', 'despite', 'hoax', and 'fake'. The entire process created 44 different features to be used. Many of these were empty, such as the polarizing word features. We applied normalization to the features in order to standardize them for comparison.

3.2 Dimensionality Reduction

Although in theory a greater number of statistically independent features should reduce further and further the error, in practice additional features often lead to worse performance. This occurs either when the wrong model is used, or because the training sample is not infinite, so the distributions are not accurately estimated. This leads to problems with overfitting, where the model works well on the training data but poorly on the testing data, and complexity, where the model takes longer to run at $O(d^2n)$ where d is dimension and n is number of samples. The solution to these problems is dimensionality reduction. Linear methods are used because of their simplicity, projecting high dimensional data onto a lower dimensional space.

Principal Component Analysis attempts to reduce the data while minimizing information loss. Typically a maximum allowable loss is chosen, in our case 10%, 5%, and 1%. One might imagine the major and minor axis of a set of two dimensional elliptical data. Principal component analysis would choose the major axis as the project direction to preserve information. Because there is no consideration of class information, it is an unsupervised method. The projection vector is obtained using the Karhunen-Loeve theorem, by discarding the minimum orthogonal vectors. This is possible by obtaining the eigenvectors and values of the covariance matrix, and then discarding the smallest eigenvalues and corresponding vectors. This minimizes the squared error.

3.3 Supervised Methods

In Bayesian theory, one calculates the posterior probability of an input belonging to a particular class by taking the product of prior probability and likelihood of that event occurring and dividing it by the evidence. Methods based on this expression are a popular solution in pattern recognition.

$$P(w_j|x) = \frac{p(x|w_j)P(w_j)}{p(x)}$$

Classes are chosen based on the highest a-posteriori probability $P(w_j|x)$, calculated using the a-prior probability of a given class $P(w_j)$, the evidence $p(x)$, and the probability distribution function for the feature with respect to the given class $p(x|w_j)$. [4]

Maximum posterior probability assumes a Gaussian distribution. Minimum error rate classification can be achieved through use of the discriminant function

$$g_i(x) = \ln p(x|w_i) + \ln P(w_i)$$

Given two discriminant functions $g_0(x)$ and $g_1(x)$ the larger of the two posterior probabilities chooses the correct class. Each of the cases is a variation on this formula.

Case 1: The simplest case occurs when every feature is statistically independent and has the same variance σ^2. This correlates to the assumption that the data falls into equal-size hyperspherical clusters around the means. The equation simplifies to

$$g_i(x) = \frac{(x-\mu_i)^t(x-\mu_i)}{2\sigma^2} + \ln P(w_i)$$

This is plugged into the general discriminant function equation to determine the class. The variance was calculated by averaging every element of the two covariance matrices.

Case 2: Another simplified case occurs when the covariance matrices are equal for both classes. This correlates to the assumption that the samples fall into hyperellipsoid clusters of equal size and shape around the means. The equation now only simplifies to

$$g_i(x) = -\frac{1}{2}(x-\mu_i)^t\Sigma^{-1}(x-\mu_i) + \ln P(w_i)$$

This is plugged into the general discriminant function equation to determine the class. The covariance matrix was calculated by averaging the two covariance matrices.

Case 3: In the general multivariate normal case each covariance matrix is different. This correlates to any normal distribution of data. The full equation is

$$g_i(x) = -\frac{1}{2}x^t\Sigma_i^{-1}x + (\Sigma_i^{-1}\mu_i)^t x - \frac{1}{2}\mu_i^t\Sigma_i^{-1}\mu_i - \frac{1}{2}\ln|\Sigma_i| + \ln P(w_i)$$

3.3.1 k-Nearest Neighbor

The motivation for k-nearest neighbor is to estimate the density function without the assumption that the probability distribution has a particular form. Given a point x, we can grow a hypersphere of volume V that encloses k points. Counting the number of samples belonging to class m in that region as k_m, the estimated density is calculated as follows: $p(x|w_m) = \frac{k_m}{n_m V}$ is the probability distribution function, $P(w_m) = \frac{n_m}{n}$ is the prior probability, and $p(x) = \frac{k}{nV}$ is the evidence. The posterior probability is calculated as

$$P(w_j|x) = \frac{p(x|w_j)P(w_j)}{p(x)}$$

For the sample x, the class with the highest posterior probability is chosen as the decision. To calculate the enclosing volume, multiple different metrics can be used for distance. Variations on the Minkowski distance

$$L_k(a,b) = (\Sigma_{i=1}^d |a_i - b_i|^k)^{1/k}$$

such as the Euclidean distance at $k = 2$ are most commonly used. As the k value is increased, more and more samples are included in the calculation as the volume increases.

3.3.2 Backpropagation

Backpropagation seeks to find a set of weight values that will minimize the mean square error from a network by computing partial derivatives. It is a feedforward network composed of i inputs, n layers containing m neurons, and j outputs that back propagates an update to the set of weights associated with each neuron. The algorithm is a two phase process. The first phase consists of a forward pass across the network where each neuron's inputs are multiplied by their associated weights and then summed. The result from this summation is then fed into an activation function. The output from the activation function serves as the input to all other neurons it is connected to in the next layer. The difference between the output values from the last layer and the true

or desired output is found using the following equation. This difference propagates back to each neuron in each layer and begins the second phase of the algorithm.

$$E = \frac{1}{2}\Sigma_j(T_j - S(y_j))^2 \tag{1}$$

Since the network is feed forward it's final output is the result of composite functions. Thus to propagate the error backwards, the partial derivatives at each neuron need to be found. The update to each weight is found using the following equations.

$$\omega_{st}^{k+1} = \omega_{st}^{k} - c^k \frac{\delta E^k}{\delta \omega_{st}^k} \tag{2}$$

$$\frac{\delta E^k}{\delta \omega_{st}^k} = -(T_j - S_j)(S_j')(S_q(h_q)) \tag{3}$$

$$\frac{\delta E^k}{\delta \omega_{st}^k} = [\Sigma_j(T_j - S_j)(S_j')(\omega_{qj}](S_q')(x_i) \tag{4}$$

Equation 2 is used to calculate the update values on the weights associated to the neurons on the last layer and equation 3 is used to calculate the update value for the weights associated with the neurons found in the hidden layers.

3.3.3 Decision Tree

Decision trees, also known as classification and regression trees(CART), are a non-parametric learning technique used to classify data. A decision tree is composed of nodes and leaves. The algorithm recursively looks at nodes making a decision about a single variable until a leaf node is reached. When a leaf node is reached a classification or regression value is returned. A classification tree uses nominal true/false responses and regression trees use numeric values when making a decision at each node. A tree is trained by recursively splitting the training data into subsets based on the value of a single feature. All possible binary splits are examined and the optimal split is made using the Gini impurity value calculated using the following formula.

$$i(N) = 1 - \Sigma_j P^2(\omega_j) \tag{5}$$

The data stops splitting when the subset has become pure, has too few samples, or produces invalid splits. A subset is pure when all the values in the subset are equal to the parent value.

3.3.4 Support Vector Machine

A support vector machine is a supervised learning technique that seeks to find the best hyperplane to differentiate between classes and thus is excellent for linearly separable data. When the data is not linearly separable a kerneling trick can be employed allowing the data to be projected into a higher dimensional space making a bounding hyperplane evident. To implement support vector machines on multi-class data, a set of support vector machines are found to be used in conjunction.

3.4 Unsupervised Methods

In unsupervised classification, the classes of samples are not known, and the number of appropriate classes must be determined. In order to solve these problems, clustering is used based on distance metrics. Between the features of samples is a calculated measure of similarity, such as Euclidean distance. This metric can be used to compare samples, and to find clusters of similar samples. In figure 1 from researcher Kadir Peker, samples are comprised of income and debt features. If the features are plotted, it is possible to see three separate clusters of individuals. In this case, the discrimination is easy. With more data, more dimensions, and closer clusters, the problem becomes more challenging, and more appropriate for a computer to solve.

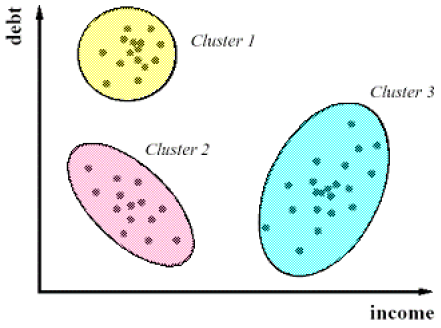

Figure 2: Income vs. debt cluster example

At the core of each clustering method is a measure of distance between two objects in the feature space. Generally, this is applied in three cases: between sample similarity, between cluster similarity, and sample-cluster similarity. In the simplest case of between sample similarity, typical Minkowski distances can be used without modification. When comparing to a cluster, a mean is calculated for the cluster and then that mean is treated as a point. Using this mean is called the centroid distance. Two other common possibilities for clusters are comparing the nearest and furthest neighbors. Typically, nearest neighbor leads to cluster structures reminiscent of minimum spanning tree, while furthest neighbor distance leads to highly interwoven clusters.

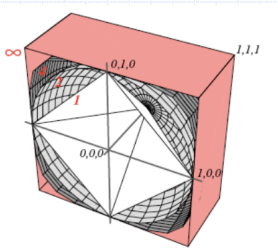

Figure 3: Various Minkowski distances

3.4.1 K-Means

K-means is a centroid-based clustering technique, where clusters centers are represented by a structure identical to the individual samples. Because K-means is a parametric method, the number of clusters must be given at the start of calculation, a significant drawback. However, the simplicity of implementation and relatability to the k-nearest-neighbor algorithm makes this a commonly chosen method. The algorithm looks for the optimal cluster placement such that the squared distances to each sample in the cluster is minimized. The algorithm begins by creating arbitrary cluster centroids, and then assigning samples to the nearest centroid. The centroid is then recalculated as the mean of the assigned samples. Samples are reassigned, and if any sample classification has changed, then centroid is recalculated and the process begins again.

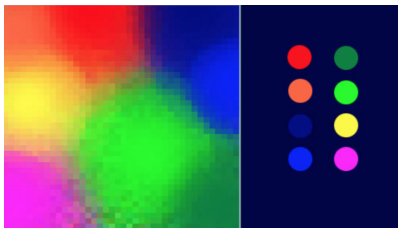

Figure 4: Representative clusters for an example image

3.4.2 Winner-Take-All

Winner-take-all is another centroid-based clustering technique similar to the K-means algorithm. The number of clusters must still be given at the start of calculation. In this method, a learning

rate parameter is specified to control the movement of the centroid towards each sample. Each sample exerts a pull on the closest centroid during each iteration. The algorithm begins by creating arbitrary cluster centroids. Then for each sample, the closest centroid is chosen to be updated by adding the distance from the sample x to this winner ω_r using the formula

$$\omega_r^{k+1} = \omega_r^k + \epsilon(x - \omega_r^k)$$

where ϵ is the learning rate, and is typically chosen to be on the order of 0.01.

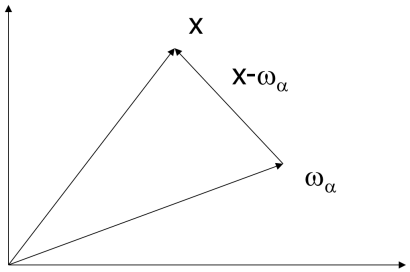

Figure 5: Winning cluster update for a sample in winner-takes-all

3.4.3 Kohonen Maps

Kohonen maps, or self-organizing maps, are similar to artificial neural networks in structure and are used for a type of dimensionality reduction. For each problem, a topological space is chosen on which to apply the cluster centers. The goal of this space is to cause different parts of the feature space to interact and improve performance. For each problem, a topological structure is assumed to exist between each pair of the cluster centers. This is not dependent on the clustering data in any way, and is purposefully assigned arbitrarily. Choices for this structure are single point, linear, and grid topologies. In the same way that the structure of an artificial neural network is arbitrary, but can affect the performance of training for a specific problem, so does the topology of the self-organizing map extend the winner-takes-all algorithm. As the winning cluster is updated, its neighbors in the topological space are also updated.

The algorithm for Kohonen maps runs in the same way as winner-takes-all. However, when a winner is chosen for a specific sample, that sample contributes to the movement of the winner and other clusters using the function

$$\omega_r^{k+1} = \omega_r^k + \epsilon * \Phi(k) * (x - \omega_r^k)$$

In this function, a learning rate ϵ is multiplied by a measure of topological distance $\Phi(k)$ from the winner and a feature distance $(x - \omega_r^k)$. The topological distance can be calculated using the function

$$\Phi = e^{\frac{\|g_r - g_{winner}\|^2}{2\sigma^2}}$$

In our solution, clusters were randomly assigned to a linear value from one to the number of clusters, leading to a linear distance calculation for this topology. Other topologies are possible such as the grid topology shown below.

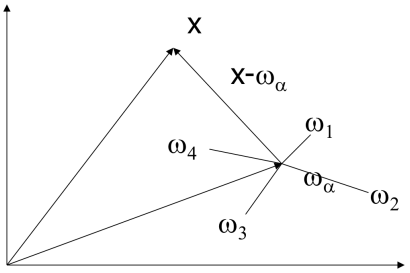

Figure 6: Winning cluster update for a sample in Kohonen mapping

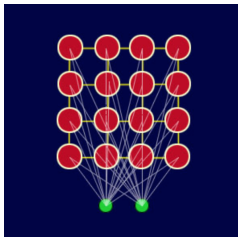

Figure 7: An example grid topology

3.5 Classifier Fusion

The fundamental idea behind classifier fusion is the saying, "Three heads are better than one." By combining classifiers in the same way we might combine expert opinions, higher accuracy can be achieved. Three common methods include voting, naive Bayes fusion, and behavior knowledge space fusion.

Majority voting is the most simple approach to fusion. There are as many ways to approach this as there are voting methods, but for our approach we chose the class with a plurality of votes. In the case of any ties, the decision was made in favor of the class with the higher natural training prior probability.

Naive Bayes classifier fusion operates under the assumption that the classifiers being fused are mutually independent. A look up table is created by finding the highest joint probability of the likelihood of classifying a sample as class m given it belongs to class n. A matrix is created to hold the joint probabilities. The rows of the joint probability matrix represent the actual class labels and the columns represent the combinations of possible classifier class labels. For example, the first column of the joint probability matrix holds the likelihood a sample was classified as class zero by both classifiers. The second column held the likelihood that the first classifier classified as class zero and the second classifier as class one. The look up table is generated from this joint probability matrix by finding the row that contains the highest likelihood for a each combination. The row that contains the highest probability represents the class label the classifier combination produces. For example, if a classifier combination 1,3 has the highest probability in row 0, then a sample that is classified by classifier one as class one and classifier two as class 3 will be classified as class 0 by the fusion of the two classifiers.

BKS fusion is derived from the same logic as the naive Bayes method, without the assumption of independence. In this method, we examine all possible combinations of classifier predictions. For each, the total number in each true class is counted, and the maximum is assigned as the new predicted class, in the same way that naive Bayes chooses a new prediction. This scheme is then applied to the training data.

3.6 Validation

In order to test the performance of the classification methods, the data must be divided into a training and testing set. The training set is used to create the model, and the testing set is used to evaluate the learned performance. Unfortunately, many models suffer because the training sample is not infinite, so the distributions are not accurately estimated. This leads to problems with overfitting, where the model works well on the training data but poorly on the testing data. Cross validation attempts a solution to this problem by using the data to create multiple, randomly interselected sets of training and testing data. With m-fold cross validation, the data is broken into m samples of equal size for testing, with the rest of the data being used for training. This method provides more realistic accuracy numbers for the data, and lessens the effects of overfitting.

4 Experiments and Results

4.1 Data

The data for this project were 40,350 sample combinations of headlines and body text snippets from many different news articles. These data were hand classified according to a stance, with 73% being unrelated, 18% discussing, 1% agreeing, and 2% disagreeing. Each pair was sanitized,

and then had 44 natural language processing features extracted. Across our experiments we used both the normalized and the dimensionality reduced versions of these data.

4.2 Dimensionality Reduction

Across our 44 features the data is mostly sparse, filled with many 0s. In order to improve runtimes and explore the necessity of these features, we applied principal component analysis to the normalized set of training features. We tried three different error rates: at 10% error only four components remained, at 5% error six remained, and at 1% error nine remained. The most prominent features were the word intersections of the headline and body text, and the number of headline n-grams that appear in the body text. We tested each of these reductions against the MPP, decision tree, and SVM methods. The results of these tests are below. There appears to be an accuracy limit based on the features around 83%. The methods with high accuracy seem to need all features to make discriminations, while the methods with lower accuracy are improved by removing these obscuring details.

Method	Normalized	PCA 4 (10%)	PCA 6 (5%)	PCA 9 (1%)
MPP Case 1	76.7%	80.7%	80.0%	78.7%
MPP Case 2	83.7%	83.2%	83.4%	83.6%
MPP Case 3	79.2%	83.4%	83.6%	83.6%
Decision Tree	84.2%	81.1%	81.2%	81.4%
SVM	86.9%	85.6%	85.6%	86.4%

Table 1: Effects of various principal component analysis reductions

4.3 Prior Probability

In order to explore the effects of various prior probabilities across the four possible classes, we chose to vary the intensity of one class while keeping the rest equal. These results were compared against the standard of prior probability calculated from our testing data. For this experiment, we ran the normalized feature data through all three cases of MPP, and then averaged the results into our final metric. We found that even though the true prior probabilities are not equal, with class four making up around 75% of the data, our best accuracy came from roughly equal prior probabilities. The chart below summarizes our results.

Changing Class

Changing Intensity

Class 1	Class 2	Class 3	Class 4
[0.07, 0.02, 0.18, 0.73]	*	*	*
78.5%	78.5%	78.5%	78.5%
[0.1, 0.3, 0.3, 0.3]	[0.3, 0.1, 0.3, 0.3]	[0.3, 0.3, 0.1, 0.3]	[0.3, 0.3, 0.3, 0.1]
78.3%	79.6%	75.9%	77.3%
[0.25, 0.25, 0.25, 0.25]	*	*	*
78.6%	78.6%	78.6%	78.6%
[0.4, 0.2, 0.2, 0.2]	[0.2, 0.4, 0.2, 0.2]	[0.2, 0.2, 0.4, 0.2]	[0.2, 0.2, 0.2, 0.4]
77.4%	77.0%	79.4%	78.6%
[0.7, 0.1, 0.1, 0.1]	[0.1, 0.7, 0.1, 0.1]	[0.1, 0.1, 0.7, 0.1]	[0.1, 0.1, 0.1, 0.7]
74.6%	71.9%	77.9%	77.4%

Figure 8: Average MPP results for varying prior probabilities

4.4 KNN

We experimented with both the neighborhood size and distance metric for k-nearest neighbors using normalized data. We found that for this problem, the distance metric does not have a

profound effect on the accuracy. By varying the order of the Minkowski distance, we saw only a slow downward trend as the order increased, which is shown in the figure below. Varying the neighborhood size saw more interesting results. At k values lower than 6, KNN achieved almost 100% accuracy on our data set, while at higher values the accuracy declined quickly. We do not believe this means we have found a new incredible method for this problem, but instead believe the nature of the data contributes to this accuracy. Although there are 40,000 data samples, these are not 40,000 separate articles, but instead combinations of a much smaller number of articles. Because of this, the neighborhood for a testing sample will find identical body texts and headlines, as article pieces create clusters. We believe that this is an overfitting problem, and that KNN would perform poorly on new articles.

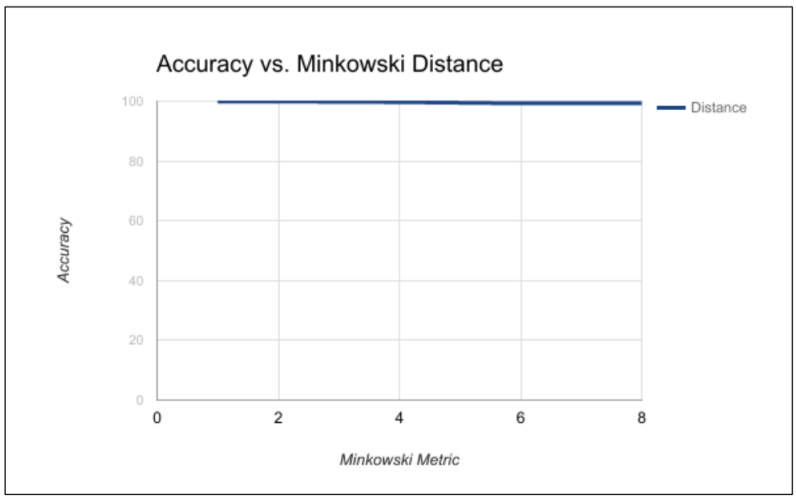

Figure 9: Varying the Minkowski distance has little effect on KNN

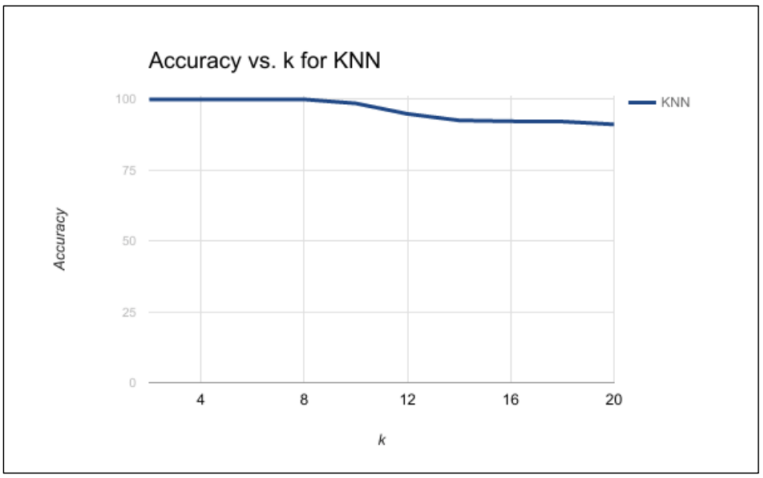

Figure 10: As the neighborhood size k increases, the accuracy decreases

4.5 Fusion

We attempted fusion using a combination of multiple of our classifiers. We chose to leave out KNN due to its problems with overfitting. We found that across our suite of tests, using the best four singular methods led to the best combined accuracy. The table below gives some of our results. SVM was most influential on our methods, adding the most to our overall accuracy.

Fusion	Methods	Accuracy
Majority Voting Fusion		
	SVM x BPNN	85.2%
	Decision Tree x BPNN	84.9%
	SVM x MPP 2	86.6%
	BPNN x MPP 2 x Decision Tree	86.6%
	SVM x BPNN x MPP 2 x Decision Tree	86.9%
Naive Bayes Fusion		
	SVM x BPNN	73.3%
BKS Fusion		
	SVM x BPNN	87.5%
	Decision Tree x BPNN	85.8%
	SVM x MPP 2	87.9%
	BPNN x MPP 2 x Decision Tree	87.2%
	SVM x BPNN x MPP 2 x Decision Tree	88.0%

Table 2: Fusing different methods led to different accuracies

4.6 Overall Results

Although different setups and data were used across all of our methods, every method was evaluated using the same 10-fold cross validation setup, using the same sets. All accuracy metrics are reported as an average of these ten folds. The best results are shown in the table below. We found that across all of our methods, KNN best predicted the classes; however, we believe this is more related to the clustering of articles in the dataset as explained above, and is an overfitting of the data that would not be representative on new articles. The real champion of our singular methods was the support vector machine, with an accuracy of 86.9%. Behavior-knowledge space fusion produced our highest accuracy at 88.0%. Multiple of our methods passed the competition benchmark of gradient boosting.

Method	Accuracy
Gradient Boosting (Baseline)	79.5%
MPP Case 1	80.7%
MPP Case 2	86.8%
MPP Case 3	83.6%
KNN	99.9%
BPNN	85.1%
Decision Tree	84.2%
SVM	86.9%
K-Means	56.7%
Winner-Takes-All	50.7%
Kohonen Maps	56.0%
Majority Voting Fusion	86.9%
Naive Bayes Fusion	73.3%
BKS Fusion	88.0%

Table 3: Best results from each method

The best setups for all of our methods are found below. For each method, the confusion matrix was generated as a sum of all ten folds of cross validation, representing the whole data set. The baseline competition accuracy from gradient boosting ran on the non-normalized feature set extracted. The details of this setup are unknown.

All three cases of maximum posterior probability were implemented in Python. For case 1, the pca reduced four feature data with prior probabilities of 10%, 10%, 70%, and 10% for classes one to four, respectively, found the highest accuracy of 80.7%. For case 2, the normalized data with prior probabilities of 0.07%, 0.02%, 0.18%, and 73.0% for classes one to four, respectively, found the highest accuracy of 86.8%. For case 3, the pca reduced nine feature data with prior probabilities of 10%, 70%, 10%, and 10% for classes one to four, respectively, found the highest

accuracy of 83.6%. The confusion matrices for these setups are given below. The assumptions for case two proved to be most successful for this problem.

class	0	1	2	3
0	141	278	2058	439
1	24	90	443	121
2	306	596	5194	1013
3	59	756	1658	27174

Table 4: MPP Case 1 Confusion Matrix

class	0	1	2	3
0	550	41	1678	647
1	54	45	390	189
2	277	24	5222	1586
3	29	8	385	29225

Table 5: MPP Case 2 Confusion Matrix

class	0	1	2	3
0	841	41	1760	274
1	115	47	416	100
2	1134	69	5187	719
3	491	195	1295	27666

Table 6: MPP Case 3 Confusion Matrix

The k-nearest neighbor algorithm was implemented in Python. The best run used the normalized data, with a a k of 2 and a Minkowski distance order of 1. Equal prior probability was assumed for all cases. This setup obtained an accuracy of 99.9%, overfitting the data for the reasons above. The confusion matrix for this setup is shown below.

class	0	1	2	3
0	2914	0	0	2
1	0	678	0	0
2	1	0	7104	4
3	28	0	3	29616

Table 7: KNN Confusion Matrix

A neural network composed of 10 hidden layers with 4 neurons and a learning rate of 0.01 was trained to obtain $\epsilon = .001$. The network trained for 100,000 epochs using 70% of the data as training data, 15% as validation data, and 15% as test data under 6 fold cross validation. After running for 100,000 epochs, $\epsilon = .001$ was not obtained. Instead the network converged to $\epsilon = .055$ after 10,000 epochs. The trained network obtained a classification accuracy of 85.1%. Figure 11 depicts the network structure and figure 12 shows the performance of the network across all epochs. The confusion matrix shown in table 8 expresses the exact classification results of the network.

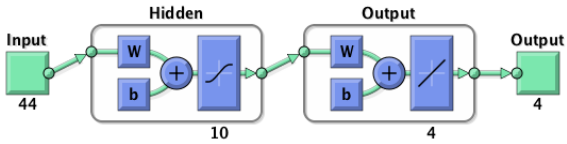

Figure 11: Backpropagation Neural Network

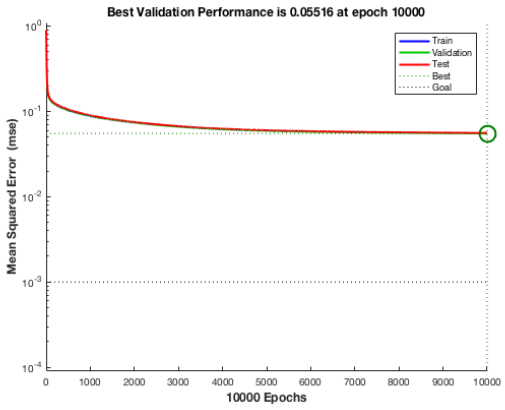

Figure 12: BP performance results

class	0	1	2	3
0	20	4	4	1
1	2	0	1	0
2	1948	0	1	0
3	946	253	2127	29353

Table 8: BPNN Confusion Matrix

A classification tree with binary splits was implemented using each training fold from ten fold cross validation. The binary splits in the decision tree were found using the exact search algorithm. The corresponding test fold was classified using the decision tree. A classification accuracy for each fold was found. A confusion matrix containing the classification breakdown from each fold was generated. A summation of the confusion matrices from each fold was done resulting in one confusion matrix representing the performance of decision tree on the entire data set and is shown in table9. The statistical overall performance of the decision tree algorithm was found by obtaining the mean of all the fold accuracies. The overall performance of the decision tree algorithm was 84.2%.

class	0	1	2	3
0	923	151	1433	409
1	169	54	329	126
2	1483	272	4403	951
3	353	88	864	28342

Table 9: Decision Tree Confusion Matrix

A multi-class support vector machine was implemented in Matlab using the training data from each fold of ten fold cross validation. Six support vector machine learners were created internally by Matlab to classify the data into the four classes. The classification accuracy from each fold was determined. The mean classification accuracy of 86.9% was found and represents the overall performance of the multi-class svm model used in this report. A confusion matrix presenting each folds performance was obtained. The summation of all the confusion matrices is shown in table 10.

class	0	1	2	3
0	334	0	2172	410
1	51	0	506	121
2	161	0	5994	954
3	38	0	618	28991

Table 10: Support Vector Machine Confusion Matrix

Three clustering algorithms were implemented in Python, with the best results obtained using the normalized data and a Minkowski distance order of two across all methods. K-means obtained an accuracy of 56.7%, winner-takes all achieved an accuracy of 50.7%, and Kohonen Maps, using a linear organization, obtained an accuracy of 56.0%. These methods all suffered from the article clustering issue that affected KNN. The confusion matrices for these methods can be found below.

class	0	1	2	3
0	886	124	1626	280
1	154	48	407	69
2	2009	369	4053	678
3	256	8992	2522	17877

Table 11: K-means Confusion Matrix

class	0	1	2	3
0	626	165	1949	176
1	181	40	412	45
2	1605	388	4685	431
3	4464	9071	1014	15098

Table 12: Winner-takes all Confusion Matrix

class	0	1	2	3
0	619	90	1806	401
1	112	39	435	92
2	1373	269	4531	936
3	158	9765	2320	17404

Table 13: Kohonen Maps Confusion Matrix

Two nxm matrices whose elements hold the likelihood of classifying a sample as class m given it belongs to class n were generated. One likelihood matrix was created from the support vector machine confusion matrix and the other from the backpropagation neural network confusion matrix. These two likelihood matrices are shown in tables 14 and 15, respectively. A $4x16$ matrix, not shown in this report, was generated to hold the joint probabilities that result from joining these two probability spaces. The resulting look up table from the joint probability matrix generated from the svm and backpropagation confusion matrices is shown in table 16 and obtained a classification accuracy 0f 73.3%.

class	0	1	2	3
0	.57	.00	.23	.01
1	.08	.00	.05	.00
2	.28	.00	.64	.034
3	.07	.00	.07	.95

Table 14: Support Vector Machine Probabilities

class	0	1	2	3
0	.01	.02	.02	.003
1	.00	.00	.004	.004
2	.67	.00	.004	.001
3	.32	.98	.99	.997

Table 15: Backpropagation Probabilities

$label_1$,$label_2$	Fused Class
0, 0	2
0, 1	3
0, 2	3
0, 3	3
1, 0	3
1, 1	2
1, 2	3
1, 3	3
2, 0	3
2, 1	3
2, 2	3
2, 3	3
3, 0	3
3, 1	3
3, 2	3
3, 3	3

Table 16: Naive Bayes Classifier Fusion Look up table

For both majority voting and behavior-knowledge space fusion, a combination of the top four best singular methods proved to be most accurate in combination. These were SVM, BPNN, MPP 2, and Decision Tree. The accuracy for majority voting was 86.9%, while the accuracy for BKS fusion was 88.0%, the highest overall accuracy. The confusion matrices for these two methods can be found below.

class	0	1	2	3
0	279	2	1988	652
1	34	0	456	188
2	61	1	5493	1554
3	15	1	322	29309

Table 17: Majority Voting Fusion Confusion Matrix

class	0	1	2	3
0	586	22	1923	385
1	62	42	450	124
2	279	22	5912	896
3	32	2	649	28964

Table 18: Behavior-Knowledge Space Fusion Confusion Matrix

5 Conclusion

Bayes Decision Rule was applied to the data under three different discriminant functions. The supervised learning algorithms k-nearest neighbors, backpropagation, decision tree, and support vector machine were implemented along with the unsupervised clustering algorithms k-means, winner takes all, and Kohonen maps. Classifier fusion was done using majority voting, naive Bayes, and behavior knowledge space algorithms. Each was applied to the problem of stance detection between pairs of news article headlines and body snippets to determine if they are related. Experimentation was done to determine the effects of dimensionality reduction, varying prior probability, assuming different probability distribution functions, changing the distance metric, and applying different fusion methods. Overall the best result of 88% accuracy came from behavior-knowledge space fusion.

The objective of this project was to apply the comprehensive suite of machine learning methods from this class to the problem of fake news detection. From this, we first and foremost gained experience using all the methods from the classroom. We also learned more about multi-class

classification problems, and the challenges of real data sources. Along the way, we struggled with the challenge of keeping our methods and data organized. Of special note was difficulty with getting the right results to represent trends in and other features of the data. Our future work is centered around the fake news challenge competition. Before the testing data is released, our generalized and scattered methods need to be tailored to fit this problem for higher accuracy, and moved into a comprehensive suite that can run from start to finish. New features like point of view, sentence voice, and tense can all be extracted from the raw text. We would also like to try newer methods that previous research has shown to be effective on this type of problem, such as long short term memory networks. Once this work is done, we will be ready for the competition.

6 Appendix

All code available at https://github.com/LambentLight/571final-fnc, or by request from egreenle@vols.utk.edu.

References

[1] Micropinion generation: An unsupervised approach to generating ultra-concise summaries of opinions. April 2012.

[2] Isabelle Augenstein and Tim Rocktaschel. Stance detection with bidirectional conditional encoding. September 2016.

[3] Michael Barthel, Amy Mitchell, and Jesse Holcomb. Many americans believe fake news is sowing confustion, December 2016.

[4] Peter E Hart David G Stork Duda, Richard O. *Pattern Classification*. Wiley, New York, 2nd edition, 2001.

[5] Dhruv Ghulati. Introducing factmata - artificial intelligence for automated fact-checking, 2016.

[6] Geremy Heitz, Stephen Gould, Ashutosh Saxena, and Daphne Koller. Cascaded classification models: Combining models for holistic scene understanding. *Advances in Neural Information Processing Systems*, November 2008.

[7] Brigitte Krenn and Samuelsson Christer. The linguist's guide to statistics. December 1997.

[8] Mirko Lai, Delia Irazu Hernandez Farias, Viviana Patti, and Paolo Rosso. Friends and enemies of clinton and trump: Using context for detecting stance in political tweets. February 2017.

[9] Marilena Lazar and Diana Militaru. The role of decision trees in natural language processing. *Sisom and Acoustics*, 2015.

[10] B. Liu. Sentiment analysis and opinion mining. *Synthesis Lectures on Human Language Technologies*, 5:1—-167, 2012.

[11] Edward Misback and Craig Pfeifer. Fake news challenge/fnc-1, February 2017.

[12] Alyssa Newcomb. Google offers fact checked results amid 'fake news' crisis, April 2017.

[13] Dean Pomeraleau and Delip Rao. Fakenewschallenge: Exploring how artificial intelligence tachnologies could be leveraged to combat fake news, 2016.

[14] Jeffrey Ullman. *Mining of Massive Datasets*. Cambridge University Press, 2012.

[15] Prashanth Vijayaraghavan, Ivan Sysoev, Soroush Vosoughi, and Deb Roy. Deepstance at semeval-2016 task 6: Detecting stance in tweets using character and word-level cnns. *Proceedings of SemEval-2016*, June 2016.