Topical and Sentimental Analysis of Weather-related Tweets.

Min Cheol Jeong, Jeff Tian, Rachel Yen, Johnny Zheng

1. INTRODUCTION

Our motivation for this project was the practical and powerful applications that topic and sentiment analysis can provide. In today's world, people are constantly sharing their thoughts and opinions via various social media networks. With the advent of the social media and mobile revolution, there is an unprecedented amount of invaluable user information that can be used for anything from quick opinion polls to refining advertisements. Such a large dataset is expensive to parse and analyze by humans, but the techniques we explore make it tractable. Topical analysis allows quick text classification according to topics of interest, and sentiment analysis indicates the popular opinion on the given topic.

This problem is interesting because there is inherently a lot of noise in social media posts, especially text-sparse tweets. It is in our interest to learn how to best deal with such informal and minimal bodies of texts. Often it is hard to tell whether a tweet is positive or not, even for human readers, given semantic ambiguity. In this project we not only try to apply different labels for a tweet, but also predict what percentage of humans readers would agree with that labeling. Ultimately we are asking, "How well can we model the topical/sentimental ambiguity (or lack thereof) that comes in a body of text? Can we answer such questions based on the results of different algorithms, and compare generative vs. discriminative algorithms at this task? What properties do we have to consider in order to interpret the intentions of its authors? Is the regression task more difficult when the body of text is not from a normal text document, but from short informal tweets? How do standard text preprocessing for classification tasks perform on tweets?"

Given the data set given by the competition, we wished to use an algorithm that would give non-deterministic outputs quickly in order to investigate the effects of basic natural language processing operations with the use of various python libraries. We were also interested in the possibilities of generating such unstructured texts by using a more generative algorithm such as Naïve Bayes.

In the end we found Ridge Regression to be the best algorithm to use, and improved our result using stacked prediction. Stop words and stemming were largely ineffective or detrimental.

2. PROBLEM DEFINITIONS, RESEARCH QUESTIONS, and METHODS

2.1 Problem Definition

As stated on the kaggle competition site for “Sunny with a Chance of Hashtags”: “In this competition you are provided a set of tweets related to the weather. The challenge is to analyze the tweet and determine whether it has a positive, negative, or neutral sentiment, whether the weather occurred in the past, present, or future, and what sort of weather the tweet references.”

More specifically, our prediction task is this: there are a total of 24 classes, each associated with some property about the current weather. There are 3 categories of classes: 's' which indicates whether the author has positive, negative or neutral feelings about the weather. 'w' which indicates whether the comment is about weather in the past, present, or a future forecast. 'k' indicates what type of weather it

talks about, ex. sunny, rainy, cloudy, etc. Our goal is to predict a confidence score for each category, making this a regression rather than a classification problem.

Even more interesting with this contest is that human raters, crowd-sourced by CrowdFlower Open Data Library, manually labeled the training data. Thus, there is an added difficulty in dealing with noise from user ratings, especially when, for example, different raters have different interpretations regarding how "happy" a certain tweet is.

Through this problem, we can explore a variety of interesting questions in an empirical way:

1. Is KNN, Ridge Regression, or Naïve Bayes better for classifying sentiment, tense, and topic in a tweet? How can we modify the parameters of each algorithm so that it performs optimally?

2. Can linguistic operations such as stopping and stemming help increase the precision of classification of tweets?

3. Is the problem more suited for a linear classifier given its high dimensionality?

2.2 METHODS

Note: We were provided with our training data by the kaggle competition.

Algorithms

1. **Random classifier –** As a baseline, we wrote code that generated random normalized values from 0 to 1 for each category.

2. **k-Nearest Neighbor –** kNN is a lazy-learning algorithm, where computation only occurs when the algorithm is fed data, hence it has a quick training time. Furthermore, kNN is a non-parametric method, so we do not care about the kind of distribution that generated our information, unlike Naïve Bayes which uses maximum likelihood for parameter estimation. Assuming tweets are more unpredictable than regular text documents, it is potentially useful to use such a non-parametric method since we do not try to make any assumptions about the generating distribution. kNN is also known to be quick at training and give reasonable results for text classification [7]. We use L2 distance because we suspect high locality between tweets with similar words and close distance. kNN is simple to implement and to run, allowing us to view our results quickly.

3. **Naïve Bayes –** The Naïve Bayes Classifier assumes a multinomial distribution of words in classification to capture word frequency occurrences. Given the naïve assumption of word independence of context and position, it will be interesting to see how this assumption performs on tweets.

We extended Naïve Bayes to our confidence-score prediction regression task by using the probability that Naïve Bayes gave for a certain label as the confidence score of that label, and then normalized scores over the *s* and *w* category labels so that they sum to 1. We did not normalize the *k* category labels because they do not need to sum to 1. We also used Laplace Smoothing so that no label receives probability of 0 just because we have not seen a particular word in that label before.

4. **Ridge Regression –** In problems with high dimensions, as we have in our task, ridge regressions performs well because high-dimensional data tend to be linearly separable. This helps with regression because we can confidently correlate most tweets with certain labels, and thus generate a high confidence score for that label, and low ones for others. Given the many different types of users we have in contributing tweets, our vocabulary size is quite large, leading to working with high feature vectors. Like SVM, ridge regression has a ridge or regularization parameter to account for overfitting of training data to the learning algorithm. Given our noisy twitter data, there are many instances where the tweets are not linearly separable, and this would be when the ridge parameter accounts for such noisy data.

Pre-processing:

Twitter tweets differ greatly from other documents, having never been edited for formality and grammaticality. Depending on the context and data, stop words removal and stemming have shown to either improve or worsen prediction. Consequently, we wish to explore the preprocessing of tweets.

Stemming:

By stemming, we hope to strip words down to their morphological roots (such as getting rid of tenses or plural forms of words) in order to reduce the differentiation between words that are the same semantically. We made use of the PorterStemmer from the nltk Python library.

Stop Words:

Our stop word list was provided by scikit-learn. We chose to remove stop words to filter out common words such as prepositions and determiners that would muddy the data from working with the most distinguishing words. The most common stop words are already downweighed by TFIDF, but there are other common words in English that may not appear as commonly among all tweets, but still provide little useful information. So we tried pruning the stop words to see if it gave a better generalization error.

Term Frequency Representation:

We used the TfidfVectorizer supplied by scikit-learn, which has parameters for a stop words list and tokenizer. By transformning the tweets into a tf-idf vector, we place more weight on words that are both applicable and rare. So we want to find words that are rare across all tweets but common in a specific subset of tweets, because these words really distinguish these tweets from the rest and thus have higher probability or revealing the sentiment and topic of that specific group of tweets. We believe these words will provide the best information regarding either the weather or sentiment or tense.

Evaluation:

Accuracy was determined by Root Mean Square Error

$$RMSE = \sqrt{\frac{\sum_{i=1}^{n}(p_i - a_i)^2}{n}}$$

Parameter Optimization and Statistical Tests

Grid Search:

For KNN and Ridge Regression, there were two features that we needed to optimize over: the maximum number of features (or words) considered by the TFIDF vectorizer, and k = number of neighbors. For this, we used the grid search algorithm, which tests pairs of these parameters at a time, and in the end we pick the best pair of number of features and number of neighbors to use for the algorithm. We do this because these two parameters may be interdependent, so we cannot optimize over one at a time. So we set up a list of feature and k values to test, and train the algorithm on every tuple in the cross product between the lists, and find the best tuple of parameters to use. We train on the training set and optimize over a held-out validation set.

K-Fold Cross Validation:

We cross-validate our classifiers over the training set to obtain sets of values that act as good estimators of the generalization error of each classifier. We also use the results of the cross validation in order to compare different algorithms with the paired-t test.

Paired-t test

To test whether any of the algorithms is better than another, we use a paired-t test on the results of the cross validation to test the hypothesis of whether the generalization root-mean square error of one learning algorithm is truly higher than that of another, by testing whether we can conclude with high probability that the errors of these algorithms come from different distributions when one algorithm's error rates are consistently higher than another's. We use 95% confidence as the benchmark and a null hypothesis that the errors indeed come from the same underlying distribution, so when we receive a p-value that is less than 0.05, we reject the null hypothesis.

RESULTS AND DISCUSSIONS

Finding Optimal Parameters for the Algorithms:

We employed the grid-search method for both kNN and ridge regression (Naïve Bayes has no parameters). For kNN, we tested every pair in a cross product of two vectors that together represent a wide range of possible values for n = number of features and k = number of neighbors. Here, number of features is the maximum of words that TFIDF considers. TFIDF chooses these words by picking those words with the top rankings, and it throws away the rest.

kNN:

n_arr = [500, 1000,1250, 1500,1750,2000,2500,5000,7500,10000, 20000, 30000, 50000]
k_arr = [10, 15,20,25,30,35,40,50,60,80,100,150,200,500,750,1000,2000,5000]
For kNN, we found optimal (number of features, value of k) pair for kNN to be (number of features = 1750, value of k = 30).

The graph above shows our 2-dimensional optimization problem, where we vary both number of features and k to find a globally low value for RMSE. The minima of this surface is at k = 30 and num features = 1750.

Ridge Regression:

We used the same grid search for ridge regression to optimize over the number of features and alpha:
n_arr = [500, 1000, 1250, 1500, 1750, 2000, 2500, 7500, 5000, 10000, 20000, 30000, 50000]
alphas = [.1, .2, .5, .7, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10]

We found that the optimal value for alpha = 2 and optimal number of features = 10000

Effects of Stopping and Stemming for KNN:

We decided to explore the effects of stemming and stopping primarily through the kNN algorithm. We trained 4 classifiers: KNN with neither stopping nor stemming, KNN with only stopping, KNN with only stemming, and KNN with both. We performed 5-fold cross validation with each of the four classifiers, and obtained the resulting estimates for generalization RMSE:

With only stemming: 0.1892, 0.1863, 0.1873, 0.1868, 0.1865 avg: 0.1872

With only stopping: 0.1893, 0.1896, 0.1893, 0.1894, 0.1898 avg: 0.1895

With both: 0.1880, 0.1882, 0.1890, 0.1884, 0.1882 avg: 0.1883

With neither: 0.1868, 0.1869, 0.1873, 0.1868, 0.1896 avg: 0.1875

Stemming and stopping both marginally hurt the generalization error of the resulting classifier. After doing paired t-tests with each of these modified kNN versions against the version without stopping or stemming, we found:

Only stemming vs. neither: p value = 0.737

Both stemming and stopping vs neither: p value = 0.157

Only stopping vs. neither: p value = 0.0064

In all cases, stopping and/or stemming marginally increased generalization error. We consider changes to be significant at or past the 95% confidence level. Stemming did not produce any statistically significant change to the classifier, and neither did both stopping and stemming, since both produced p-values lower than 0.05. Stopping is statistically proven to hurt our classifier.

We believe stemming does not help because in tweets, there are too many unique words including slang, accidental misspellings, and purposeful misspellings in an attempt to emphasize emotion. For example, someone might write 'happy' as "haaaaapy:D" which a stemmer cannot recognize as the same word. Another possible factor is the quality of the stemmer: PorterStemmer is known to make mistakes, and certain words are not stripped to the proper morphological stem and still remain a different word from its proper stem. Using kNN, two tweets that should have been more similar will not be classified as similar even though the two different words have the same meaning. This explains why stemming did not help, but to see why stemming made the error a bit worse, we have to consider overstemming. Overstemming occurs when two words with different meanings are stemmed to the same root when they have no relation whatsoever to each other semantically. We believe overstemming happened to a small degree with our data, resulting in marginally higher error.

For stopping we used a list of common English words from scikit-learn thought to provide little meaning. These stop words increased error because there might have been some non-intuitive correlation between some words and the labels. We also tried creating our own stop word list but got poor results because in such a small topic range and small bodies of text, helpful words like "sunny" appear almost the same amount as words like "the," so our self-generated stop word list threw away helpful words. We believe that stopping not only did not help but actually hurt because TFIDF already downweighs very frequent words anyways, and because there are some potentially useful emotion-related words that are thrown out by the stop word list as well.

Also, since tweets are so short and simple to begin with, removing stop words in some cases may remove the majority of the words in some tweets, leaving them with almost no words left. Whereas books and other documents are long enough and have rich enough vocabulary so that removal of stop words still

leaves the book with many useful words, tweets are simple and short, so removing stop words can potentially deprive a tweet of most of its content and structure. TFIDF is more benign, it just downweighs frequent words without removing them completely.

Comparison of the Algorithms:

To compare these algorithms, first found optimal parameters for each, as described previously. Then, using these optimal parameters, we first looked at the generalization RMSE values they produced on a held-out prediction set, as well as the results of 5-fold cross validation. To compare whether one was truly better than another, we used the paired-t test on the 5-fold values between pairs of algorithms, and tested against the null hypothesis that the two algorithms have the same generalization RMSE rates.

Here are the results for each algorithm (all errors given are RMSE):

Random:
Five-fold result: 0.4989, 0.4991, 0.4993, 0.4956, 0.4988
Error on test set: 0.4990

kNN:
Five-fold result: 0.1868, 0.1869, 0.1873, 0.1868, 0.1896
Error on test set: 0.1875
p-value for paired t-test against baseline:

Ridge Regression:
Five-fold result: 0.1574, 0.1586, 0.1581, 0.1574, 0.1576
Error on test set: 0.1579
p-value for paired t-test against baseline:

Naïve Bayes:
Five-fold result: 0.3932, 0.3924, 0.3925, 0.3930, 0.3927
Error on test set: 0.3929
p-value for paired t-test against baseline:

In addition to the 3 base algorithms, we also present corresponding numbers from the random classifier and the p-value indicating the probability that each algorithm is better than the baseline, in order to demonstrate each algorithm's value in comparison to a non-learning baseline and anchor the performance numbers we get. As can be seen from the p-values comparing them to the baseline, all algorithms do indeed learn a target concept to a good degree, as the chance that they perform as well as they did by chance is almost 0, so they are definitely much better than random guessers.

Now, we test to see whether it is true that kNN is better than Naïve Bayes, and whether Ridge Regression is better than kNN:

kNN vs. Naïve Bayes: p-value =
Ridge Regression vs. kNN: p-value =

Both of these p-values are significantly smaller than the 0.05 threshold needed, so we conclude by transitivity with very high confidence that Naïve Bayes is the worst classifier at this prediction task, kNN is intermediate, and ridge regression is the best at this task. Ridge regression's performance is good

enough that it would rank #68 on the current Kaggle boards. Now, we provide insight in order to explain the reason behind our findings:

1) KNN and ridge regression are much better at tweet classification than multinomial Naïve Bayes. We believe that this is an inherent limitation of the generative assumptions of Naïve Bayes. We believe that this is because the bag-of-words generative model grounding Naïve Bayes does not work well with our dataset and learning task. The regular Naïve Bayes classifier assumes a bag of words for each discrete output label from which words are drawn. In our situation, we attempted to extend Naïve Bayes to the confidence score prediction regression task, which distorts this previously clean generative model. Instead of first picking a prior class label and then drawing words out of the class-conditional distribution for that bag, there is no more "true class" that a tweet is generated from anymore, instead we might have a tweet whose true label indicates 70% confidence that it is sunny, and 30% chance that it is rainy. The generative model would have to draw 70% of words from the sunny bag and 30% from the rainy one. The maximum likelihood estimation for regression shifts from being an argmax over discrete labels to a difficult real-valued maximization problem that our algorithm may not estimate well.

2) KNN is generally a strong algorithm to use, but has quite a few weaknesses in this specific learning task. KNN's strengths come from the fact that tweets demonstrate good locality. If two tweets use relatively the same words (for example related to summer weather), then both are likely to talk about the same weather trend and also use many of the same words to display the similar emotions about the weather. KNN's weakness comes from the curse of dimensionality, from the fact that text classification necessitates high-dimensional vectors. In these high dimensions, distance between many instances approach 0, because there are just so many dimensions and most tweets lie at 0 with respect to most dimensions (these are words in the vocabulary that do not appear in the tweet). So the usefulness of the distance function diminishes. Another serious problem of KNN is that its prediction for a new instance is seriously limited by what it has seen before. For example, some of the weather labels we needed to predict are presumably very rare, like 'tornado' or 'hurricane'. So among the k-nearest neighbors of a hurricane-related tweet, only to top few will actually be related to hurricanes, and the rest will be unrelated, and heavily outweigh the true nearest neighbors and distort the prediction for the new instance.

3) Ridge regression works very well and is particularly well-suited for this learning task. Ridge regression works well because linear classifiers work better in higher-dimensional settings, where previously inseparable data becomes linearly separable due to the extra dimensions. This is why the kernel trick works, and it is also why linear classifiers do well in text classification. In particular, any d+1 points can be shattered by a linear classifier in d dimensions, so more dimensions allows more degrees of freedom and lets our algorithm have more freedom in choosing how to classify the points. So previously inseparable data becomes separable with high margin. At the same time, the alpha parameter allows us to tune the classifier over the validation set and avoid overfitting, so we simultaneously do not fit the training data overly well and make a good tradeoff between margin and training error. Also, unlike KNN, ridge regression does not depend on having previously seen many instances of a specific class in order to classify it correctly. So in a sense, it is one of the best possible algorithms for our learning task.

Results of Layered Prediction For KNN:

In addition to the base algorithms, we thought of 'layering' the predictions of ridge regression, our best algorithm, in order to lower the test error even further. Layering is our name for the two-step learning process where we train a ridge regression classifier on the training data, and then run the classifier on same examples we just trained on to generate the predicted confidence score vectors for those examples. Then, we concatenate each predicted confidence score vector with the actual confidence score vector for

each training example to generate a new set of vectors (one for each vector in the training set), and use these vectors as the training examples to train a second ridge regression classifier.

Then, to predict the confidence scores on a test instance, we run the first classifier on it, and then run the second classifier on the output of the first classifier, and use the output of the second classifier as our final output. Doing 5-fold cross validation with this layered method, we obtain the following RMSE values:

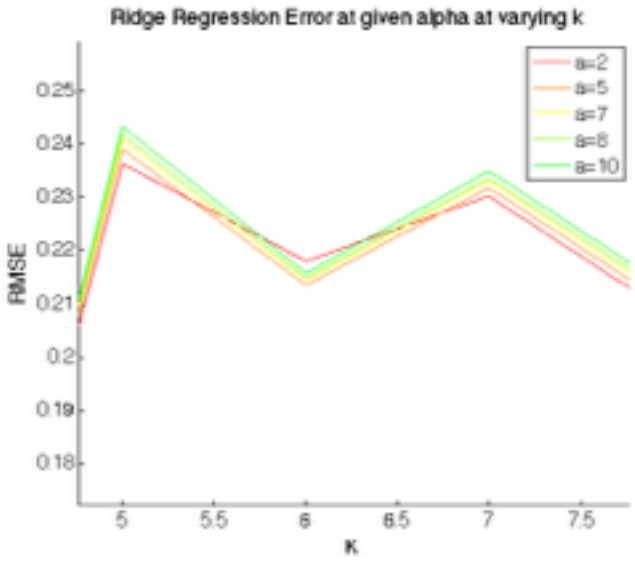
0.1541, 0.1553, 0.1552, 0.1544, 0.1563

It seems that this is an improvement over the regular unlayered ridge regression classifier, so we do a paired t-test with the regular classifier. We obtain p-value = 0.0002813 < 0.05, so we can conclude that this layered predictor is indeed better than our normal predictor.

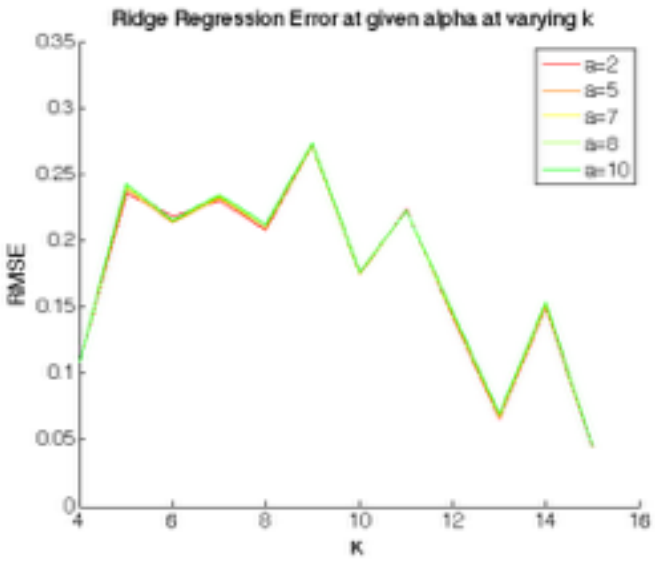
We believe that there are good theoretic foundations for this approach that made it work, as layering the prediction helps us 'learn away' the bias. In particular, consider the situation in which we always over or underpredict the confidence score for a certain label over all tweets in which a certain word appears. For example, consider the case in which the appearance of the word 'happy' makes our classifier overly biased towards predicting this tweet as 'sunny'. How can we allow our classifier to 'learn' about its bias to make it less biased? The solution is to make a second ridge regression classifier that trains on the predicted outputs of the first, so that by seeing both the biased predictions and the true labels, it can hopefully find a pattern in overprediction or underprediction among a certain class of tweets, and use this pattern to make a less biased prediction the next time it sees an instance in this class.

Results of Optimizing alpha for each Label for Ridge:

In another attempt to creatively improve the accuracy of the ridge regression algorithm, we decided to make a separate ridge regression classifier for each of the output labels (24 in total) in order to find an optimal alpha value for each individual label. Our rationale was that a single ridge regression classifier with a globally optimal alpha value may not have an alpha value that works best for a large minority of the output labels, so training a classifier for each allows us to treat each output with more individual attention.



The above graphs show the result of varying alpha across different k, where k = index in output vector of the individual output. As can be seen, some alpha values result in higher error for some of the output



labels but lower errors for others. Our classifier attempts to rectify this problem by picking and using the optimal alpha for each output label.

We then made our whole classifier by running each test instance through all 24 classifiers and stitch together each predicted entry in the confidence score vector. To test whether this method gives improvement over the normal ridge classifier, we performed 5-fold cross validation with these results:

RMSE: 0.1576, 0.1587, 0.1584, 0.1576, 0.1577

Doing a paired t-test against the normal ridge classifier, we found p-value = 0.5978, so there is really no statistical evidence that this approach of making 24 classifiers and optimizing each alpha is better than the single-classifier approach. As can be seen from the graph, this is probably because the optimal alpha values are so close to each other over the different output labels that the benefit achieved over the single classifier is so minimal that it can easily be attributed to random variance.

RELATED WORK:

In Yu, Ho, Aunachalam, Somaiya, and Lin's (2010) paper, they also explored the effects of stopping and stemming during pre-processing in text classification. However, what they dealt with was product titles rather than tweets in our situation. They actually find stopping and stemming to have a statistically significant error, suggesting stopping and stemming in very short pieces of text with many domain-specific names is incredibly harmful for classification. While our results did not yield a statistically significant error difference with our pre-processing steps, it is important to note the common harmful effects of stopping and stemming prior to text classification. Their paper uses a Naïve Bayes algorithm and a linear SVM. We decided to try Naïve Bayes as well and use ridge regression instead.

Kiremis and Toker delineate text classification with the k-Nearest Neighbor algorithm and also transformed documents into tf-idf vectors. Based on its simple weighting approach (Kiremis and Toker), k-Nearest Neighbors allows us to quickly regress/classify several categories and learn quickly as we feed more tweets even though actual classification is quite slow while observing the effects of pre-processing.

FUTURE WORK:

We would like to try other preprocessing methods to better model user behavior on the Internet, especially a medium as informal as Twitter. Given the ambiguity of computer-mediated communication, we could improve our tokenizer to account for such occurrences. On the Internet, in order to avoid offense or misinterpretation given the lack of usual nonverbal cues such as facial expressions and body language, users often use "smilies" or "emoticons" after a sarcastic statement to reveal their true intents. By mapping popular smilies to certain sentiment words, would such preprocessing improve machine learning algorithms? According to González-Ibáñez et al. (2011), machine learning algorithms and human judges perform poorly in detecting sarcasm on Twitter. In Walther and D'Addrio(2001), emoticon variations plus verbal message effects help communicate sarcasm. Both papers show emoticons are important in determining sarcasm for human judges. Therefore, further work in improving these algorithms based on our increasing knowledge of user behavior on social media would be an interesting preprocessing step to better classify tweets and their content.

Furthermore, most spelling correction algorithms correct based on unintentional errors. Text on the Internet is riddled with intentional errors due to slang, abbreviations, and acronyms such as idk, btw, brb, lol, wth, and omg. Such terms do carry indications of sentiment and intention. Also of interest to us would be using a spelling correction algorithm specialized for the social media vernacular, which tends to have common misspellings such as "prolly, rly, and r. It would be of interest to explore further how Internet informalities reduce or perhaps not affect the accuracy of machine learning text classification algorithms by fixing such "errors."

Given the success of the ridge regression algorithm, we would like to do future work that deals with the multiple outputs because the outputs are correlated with each other. In particular, it would make sense that "weather" and "sentiment" have some correlation because of the connotations associated with many words. For example, words such as "hot" and "humid" would indicate negative emotions whereas "warm" and "cozy" would be more positive. Finding an intermediate step in feeding this correlation information between the outputs and then re-training again with the ridge regression could produce interesting results and a statistically-significant improvement.

CONCLUSION:

The first research question asked which algorithm was the best for text classification involving tweets. We tried out Naive Bayes, KNN, and Ridge Regression, and a random baseline as candidates. To compare algorithms we did five-fold cross validation on each of them and used a paired t-test to check for a significant difference in RMS error. A difference was counted as significant if $p < 0.5$ for the null hypothesis.

We found that Ridge Regression performed the best overall with its lowest RMS on the cross validation being 0.1574. We attributed this result to the robustness and appropriateness of Ridge Regression to this particular problem, as opposed to the other algorithms. Naive Bayes suffered because it is not suited for continuous labels. We tried to overcome this problem by treating each probability that NB gave for a label as its confidence score. However, NB performed very poorly because with so many unique words, the probability of any given set of words being related to a label is very low. Because of this, we are never able to predict that a tweet is highly correlated to a label. KNN performed much better than NB, but was still worse than Ridge Regression due to lack of robustness. If KNN receives a tweet that is nothing like anything it has seen before, it is still forced to find the weighted average between the top k most related tweets, which are in this case not very good estimates. RR solves both these problems because it is meant for predicting continuous data, and it tries to find a mapping between tweets and

confidence scores that works for any given tweet. RR also avoids overfitting very well because it penalizes large weight vectors.

After finding Ridge Regression as the best overall algorithm, we used stacked prediction as a way to try and reduce bias. The idea behind stacked prediction is to predict using RR as normal, and then re-predict using the first prediction as a set of features. By finding a relationship between our first predictions and the true labels, we can effectively learn away some of the biases we make when predicting the first round. After experimentation we found that this method of bias reduction does help. There is a significant improvement between unstacked and stacked ridge regression.

Lastly, we explored the effects of stopping and stemming by using KNN. Only when we did stemming was the error significantly worse. Given such short pieces of text, a bad stemming job easily changes sentence meanings. As noted in Khoo et al. (2006), without large bodies of text providing more contextual information, stemming potentially takes away too much meaning from short pieces of text. Stemming can also change sentence form, changing a possibly imperative sentence to a declarative one, changing the meaning. We have concluded stemming was not a necessary pre-processing step in our task.

References:

Roberto González-Ibáñez, Smaranda Muresan, Nina Wacholder, Identifying sarcasm in Twitter: A closer look. *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies: short papers* - Vol. 2, The Association for Computer Linguistics, 2011, pp. 581–586. (8)

Anthony Khoo, Yuval Marom, and David Albrecht. 2006. Experiments with sentence classification. *Proceedings of the 2006 Australasian Language Technology Workshop*, Sydney, Australia. (2)

O. Kirmemis and G. Toker. Text categorization using k Nearest Neighbor Classification. Survey paper, Middle East Technical University

Andrew McCallum and Kamal Nigam. A comparison of event models for Naive Bayes text classification. 1998. *AAAI-98 Workshop on Learning for Text Categorization*.

Alexander Pak, Patrick Paroubek. 2010.Twitter as a corpus for sentiment analysis and opinion mining. *Proceedings of the Seventh conference on International Language Resources and Evaluation (LREC'10)*, European Language Resources Association (ELRA), Valletta, Malta.

Joseph B. Walther and Kyle P. D'Addario. 2001. The impacts of emoticons on message interpretation in computer-mediated communication. *Social Science Computer Review, 19, 323-345*.

Hsiang-Fu Yu, Chia-Hua Ho, Prakash Arunachalam, Manas Somaiya, and Chih-Jen Lin. Product title classification versus text classification. 2010. Technical report. URL http://www.csie.ntu.edu.tw/~cjlin/papers/title.pdf (1)