Lessons from the CIS 520 Competition

Obinna Asinugo ASINUGO@SEAS.UPENN.EDU
Rhea Chitalia CHITALIA@SEAS.UPENN.EDU
Nicolette Driscoll NDRIS@SEAS.UPENN.EDU

Abstract

This paper summarizes our overall strategies used to compete in the CIS 520 Machine Learning Competition at the University of Pennsylania. We present the general models (Naive Bayes, Logistic Regression, and K-Nearest Neighbours) used to conduct multi-class classification of tweets. More specifically, we detail our successes and failures during the implementation of each model. Our final results showed that ensembling by a weighted probability average led to the best performance on the validation set.

Keywords: Naive Bayes, Logistic Regression, k-nearest neighbours, ensemble

1. Introduction

Multi-class classification of tweets represents a real-world challenge as many institutions seek to analyze emotional responses from people. For instance, a company may be interested in gauging consumer reaction to a new product in order to improve marketing strategies; or a mental-health institution may screen a patients tweet history as a check for depression. The applications of tweet classification are endless. However, given the richness and variation of language often found in tweets, factors such as slang words, abbreviations, misspellings, sarcasm, and the use of emojis complicate tweet classification.

The objective of this project was to overcome such factors by creating robust learning algorithms that could classify tweets as one of the following emotions: joy, sadness, surprise, anger, or fear. Instead of evaluating simple precision of our algorithms, we considered the relationship between labels to define the cost when predicting a wrong label. Thus, our algorithms were devised to yield the lowest possible cost according to the cost matrix defined in the project description document. In addition, adding another layer of complexity to this project, one of our models was expected to overcome one of the two cost baselines: Baseline 1 (0.9500) and Baseline 2 (0.9400).

To perform multiclass tweet classification, we created three different types of models: Naive Bayes (a generative model), logistic regression (a discriminative model), and k-Nearest Neighbors (an instance-based model). We then analyzed and compared the cost performance amongst each of the models. Finally, using ensemble techniques, we combined different models to produce our best performing classification algorithm.

2. Models

2.1 Generative Model: Naive Bayes

Naive Bayes classifiers are a family of simple probabilistic classifiers based on applying Bayes theorem with naive, or strong, independence assumptions between the features. Since Naive Bayes specifies a joint probability distribution over observations and label values, we can conclude that it is a generative model. Furthermore, by using Bayes rule, we can form a conditional distribution.

2.1.1 What succeeded?

For this particular project, we used a binary multinomial Naive Bayes (NB) classifier to classify tweets. With the bag of words assumptions, we assumed that position between the words in the tweets did not matter. Additionally, instead of training on the frequency of a word occurring in a tweet, we simply trained on whether a word occurred or not. Thus, we clipped the word counts to 1 in order to improve the classifiers performance.

When testing this model against the training set without a class probability estimation (CPE) model, the resulting cost performance was 0.6410. When we tested this model against the validation set without a CPE model, it generated a cost of 0.9553. We realized that without incorporating a CPE model, our classification results were not sensitive to the weighted contribution of each cost in our cost-sensitive loss matrix. By devising a CPE model (where for each new tweet, instead of predicting the class with the maximal estimated probability, we predict the class the class with the smallest expected loss), we were able to boost the end performance our binary multinomial NB model. After implementing a CPE model, the training cost decreased to 0.6330. The final validation cost of this algorithm achieved 0.9472, which passed Baseline 1. In comparison to the other single classifier models, binary multinomial NB achieved the best validation cost performance.

2.1.2 What failed?

In getting to our final Naive Bayes model, we spent a significant amount of time trying to improve our results by using feature selection. First, we tried removing all words with a frequency of lower than 10 and higher than 750. We hypothesized that removing words that occurred too often and words that occurred less often would help the model learn the most general words associated with a particular emotion. To our surprise, we found that the resulting training cost increased to 0.6490; the testing cost also exhibited an increase to 0.9722. Binary multinomial NB is sensitive to the amount of features it receives. When you feed it more unique features, it tends to increase the models performance. However, if you remove too many features from the data set, particularly those of higher frequency that influence certain classifications or those of lower frequency which also may heavily influence specific classifications, then the models performance will deteriorate. Hence why we experienced such conflicting results.

In another effort to decrease the overall cost of the model, we attempted to append new features to our training data set. More specifically, we wanted to check for whether a unique emoji existed in a particular tweet. To do this, we identified at least 10 unique emojis in our vocabulary set and appended the same number of feature columns to our training

set. Then, we cycled through each training tweet and checked to see if any of the emojis occurred, handling cases for which it did accordingly. We hypothesized that checking to see if a particular emoji exists in a tweet should help the model learn which emojis contribute most to a specific emotion. Once again, our results conflicted with our intuition. The resulting training cost was 0.6411, and the resulting testing cost was 0.9701. We attributed this deteriorating performance to feature repetition. When appending features that already exist in the data, the performance in a binary multinomial NB model will decrease. In our case, our binary training data set already had a check for whether a particular emoji existed before appending such features. Hence why again we experienced results that conflicted with our intuition.

2.2 Discriminative Model

Broadly, discriminative models model the dependence of an unobserved label on instances of observed variables. This model assumes a form of $P(Y|X)$, which is then estimated from the training data. Logistic regression classifiers use a regression model where labels are categorical variables.

2.2.1 What succeeded?

While feature selection would have allowed for us to select the most meaningful words within our model, we chose to use the bag of words method, and implemented a set of binary input features encoding whether or not a given word, emoticon, or punctuation in our total vocabulary occurred in a given tweet. For the sake of consistency, the feature set that was tested for NB was used for generating this model. In order to expedite computational time, we used the *liblinear* package, which is an open source set of functions used to train and test logistic regression models.

For our main logistic model, we decided to implement a L2-regularized logistic regression model. When predicting labels for a new set of data using our trained model, we implemented a cost sensitive prediction where the estimated label for an input tweet would be determined by calculating the lowest estimated cost, given the probabilities associated with each prediction label. This model gave us a training cost of 0.4512, and a validation cost of 0.9737.

2.3 Instance-based Model

k-Nearest Neighbors is a non-parametric classification method used for classification and regression. In this model, a tweet is classified according to the k closest training examples in the feature space. As for our other models, the feature matrix we chose to train our final k-NN classifier algorithm on consisted of binary variables encoding word presence or absence in a tweet.

2.3.1 What succeeded?

Very similarly to our generative and discrimiantive model, we trained and tested our k-NN classifier with an identical feature set. To generate the model, we used *fitcknn* in

MATLAB. By default, *fitcknn* will find the single nearest neighbors using the Euclidean distance metric for a particular sample.

When testing this algorithm, we our training and validation cost were 0.0082 and 1.8092. Of course, our k-NN model was significantly overfitting the training data, leading to less than optimal performance on the validation set. We can attribute this low performance to the fact that our 1-Nearest Neighbour learner is consistent. In other words, the bias of this model approaches zero as the number of examples grows to infinity. As a result, the model tends to overfit that data too much, and since it no longer is generalized, the model will perform poorly on the validation set.

Due to time constraints, we were not able to perform cross-validation to seek the best k-NN model. We believe that incorporating cross-validation of parameter k, or even changing the distance function for determining a nearest neighbor, may have led to improved results.

2.4 Ensemble Methods

In the end, our best performing model came out of using ensemble methods to combine several classifiers. Ensemble methods are learning algorithms that construct a set of classifiers and then classify new data points by taking a weighted probability average or majority vote of their predictions.

2.4.1 What failed?

Our first attempt for an ensemble method combined both discriminative and generative models to allow for a majority vote based on predicted output label values. Specifically, we implemented a Naive Bayes, Logistic Regression, Support vector machine (SVM), and Decision Tree model. Each model utilized the binary features that represented whether or not a world, emoticon, or punctuation that existed within our overall training set vocabulary was present in a specific tweet to be classified. For the Naive Bayes and Logistic Regression models, we implemented CPE models, where our classification results were sensitive to the weighted contribution of each cost in our cost-sensitive loss matrix. This gave an output label from 1-5 for each instance tweet. For the SVM model, we simply predicted the output label without taking into account the cost for a misclassified tweet. Our Decision Tree bagging model was constructed by the treebagger set of functions in MATLAB. Specifically, we generated an ensemble of decision trees, where every tree was grown from an independently drawn bootstrap selection of the input data, with a predetermined total of 100 trees. This number was chosen to ensure adequate bootstrapping and model generation, yet still within the mandatory computational time requirement.

Upon prediction of the output label from each model, we then implemented a majority vote method, where the final label for a given tweet was determined by the mode label from each model. In the case of a tie, the label derived from the Naive Bayes model was chosen as the final label. We chose the Naive Bayes generated label to be our tie-breaking label, as it independently gave us the lowest training error.

This ensemble method attempt gave us training cost of 0.3958 which, while highly overfitting, was still a lower training cost than what our stand-alone models of Naive Bayes and Logistic Regression had given us, confirming our intuition that an ensemble method would likely perform better than a single, stand-alone model. When using this model to predict

on the test set, we got validation cost of 0.9485, which passed Baseline 1, but not Baseline 2.

2.4.2 What succeeded?

As our majority-vote based ensemble algorithm did not give an adequate validation cost, we then focused combining models with a probabilistic output from which we could implement CPE. More specifically, we found a weighted average probability for each class and classified the tweet as the class with the highest resulting probability.

Since we were relying on probabilistic models, our final ensemble algorithm combined logistic regression and Naive Bayes classifiers. Each trained classifier predicted a label for a given tweet and output a posterior probability for that respective label. Using weighted average of the posterior probabilities from the two classifiers, we assigned the final predicted label as the class with the highest average probability. We then used our CPE model to boost the end performance of this ensembling method. Although its training cost was 0.5631, our weighted average ensembling model achieved the best performance of all models on the validation set, surpassing Baseline 2 with a cost of 0.9257.

Due to time constraints, we were unable to include more probabilistic models, such as SVM or Convolutional Neural Networks (with a softmax output layer). We believe that the addition of several uncorrelated probabilistic models would have boosted results. Certain models may be more confident in predicting or classifying specific tweets. Thus, a greater diversity of models in our weighted average ensembling technique would have lead to more confident classifications for each sample.

3. Conclusion

In conclusion, we found that selecting specific features from our bag of words did not improve our classification model performance. As a result, we simply used a binary set of features as described in Naive Bayes section, to determine the presence or absence of a word, emoticon, or punctuation. For the sake of consistency, we trained all of the models with the same feature set. Out of our three general models, the generative model with CPE resulted in the lowest validation cost. Overall, our ensemble method of combining a Naive Bayes and Logistic Regression model, and performing CPE on the posterior probabilities from the two classifiers in order to assign the final predicted label was our highest performing model.