Jacob Dineen

Text Mining

8/10/2018

Homework 4

**Task 1 - Doodlebook**

I’ve dropped my doodlebook drawing at the below link:

<https://doodlebook.org/#/articles/5b6a1f2d8c69125456bb6d53#5b6dfaea8c69125456bb6d54>

**Task 2 - Naïve Bayes for Classification**

Bayesian Inference, and Naïve Bayes in particular, draw on prior and conditional knowledge to estimate the probability of a hypothesis using Thomas Bayes’ theorem:

P(class|X) = P(X|Class) \* P(class) / P(X)

Or

Posterior = Conditional \* Prior

We are solving for the posterior given collected evidence, and ultimately determining the probability of a hypothesis [1] - i.e... given a collection of tokenized words, what is the probability that a document belongs in a class. As discussed, there are two main branches of the naïve bayes algorithm, benoulli and multinomial. Multinomial models are seen to work better, empirically, when dealing with word frequencies, and benoulli models are preferred for shorter documents that are vectorized in a Boolean (present/not present) manner.

**Data**

As we have not yet ventured into some of the intricacies of Sklearn for dealing with vectorization parameter tuning, I’ll be conducting these experiments/analyses using Weka, which offers a slightly more intuitive interface for data preprocessing.

The data itself contains two dependent variables; Lie and Sentiment. Each of these binarized classes will partitioned and used as isolates for two separate classification tasks.

1. Predict the sentiment of a review (Positive/Negative).
2. Predict whether a review is truthful, or a lie (Fake/True).

Intuition and prior experience working with this dataset, and other text data in general, tells me that a machine will have a much easier time formulating/representing a function that maps a vectorized representation of a document to a class signifying polarity, but will ultimately struggle identifying some of the idiosyncrasies of linguistics that signify whether something is deceitful or not.

The data is represented, initially, by a 3x92 matrix, meaning there are three classes and 92 instances of each class. In other words, we have 92 reviews, which would categorize this dataset as small and prone to overfitting.

**Task**

The presented task is to utilize a variety of different vectorization and preprocessing techniques to best judge which tuning positively impacts performance – All tests will utilize the multinomial naïve bayes algorithm, and all results will be documented and visually compared.

**Evaluating Performance on Sentiment Analysis**

To begin the task of training and testing a model that can predict sentiment based on a sparse representation of a document, we first remove the unnecessary, secondary class – Lie. This will be added back in later.

It was said that a filtered classifier is a better inference technique because we aren’t exposing any of the raw frequency counts to the model come training time – In other words, we aren’t vectorizing and then splitting, and risking the idea that a model will memorize the representation of our test set come deduction.

*Brief explanation of cross validation*

I think it is first important for a brief explanation of cross validation, which is the induction technique we will be using during training. Cross validation works by dividing the entire set into subsets of the original data. Because this assignment calls for 10fold cross validation, we will essentially be following the below steps:

* Randomly select 90% of training examples.
* Construct model using the aforementioned % split.
* Retain score/
* Reshuffle with replacement.
* Repeat steps 1-4 until all folds are exhausted.
* Take the mean of all testing scores (In my experience).

We are building ten separate models and evaluating test performance per the inferences of each. In a way, it seems to resemble an ensembling method using a voting classifier, although the key to ensembling is having several uncorrelated weak learners that work together to best fit a representation.

Back to the task of classifying sentiment- I’ve run the data through the 7 noted variations of hyperparameters, and have arrived at the following conclusions (Appendix 1.1):

* Of the four basic vectorization techniques with various hyperparameter tuning, 3 had the same exact accuracy score.
* Unigram term frequency and unigram normalized tf had the highest F1 scores. F1 score is a weighted computation of precision and recall, and basically tells us that these two models higher precision. Precision is calculated by taking the total number of true positives divided by the total number of true positives + the total number of false positives. Basically, we had fewer false positives, which could be characterized as this model identifying something as belonging to class ‘Positive’, when it really belongs to class negative.
* All four original models had a training time of .01 seconds.
* Stemming negatively impacted performance. I theorize that this happens because the total length of the vocabulary is relatively small, and there is a fear that words will lose their semantic meaning.
* Mutating the original best fitting model w/ bigrams and trigrams did not improve performance. This also took a bit longer to run (.06 seconds), so we could understand why this approach might exponentially consume resources at scale.

*Conclusion of Sentiment Analysis*

With a few our hyperparameters reaching the same levels of accuracy, it would seem to be that we must make a decision regarding which types of errors we want to reduce. If we want to reduce false positives, we should stick with unigram w/ term frequency, or normalized term frequency. If we want to optimize recall/false negatives, we should use a unigram vectorization technique with tfidf. All in all, we have a model that performs around 34% better than random guess. Exporting the conditional probabilities out to excel shows us how sparse our matrix really is, and why adding data might help us to generalize to a further extent.

**Evaluating Performance on Lie Detection**

Again, we need to go back and adjust our data to exclude the sentiment variable, and include the lie variable. I prefaced my hypothesis above, and remain skeptical on a machine learning to represent a function that models whether or not a representation falls into a Lie/Truth class. I’m going to say that we will perform slightly better than random guess.

This is essentially the same task, and results are documented in Appendix 1.2. My hypothesis turned out to be correct, but I attribute that to my familiarity with this assignment dating back to Data Mining. Our best model of the original 4 sets of tuning resulted in an accuracy of 52%, only slightly better than random guess. A mutation of that model with stemming resulted in a slight increase to 54%. Still, we are talking about 92 observations, so the difference between 54% and a random guess classifier would be about 4 correctly classified instances. It’s difficult to tell if our model is learning anything here. What was a bit interesting here was the fact that setting n >1 on an ngram tokenizer resulted in substantially worse performance (about a 10% decrease from the same model with unigram). This was curious, so I looked at the conditional probabilities of the bigram + trigram model, and every single feature had a score of 0 for both classes, so it’s not surprising to see that we underperformed random guess. Interestingly, the conditionals were all zero for my best performing model as well. I think ultimately this lends credence to the idea that we have constructed a model that couldn’t learn to represent/fit our data.

References:

[1]<https://brohrer.github.io/how_bayesian_inference_works.html>

[2]<https://nlp.stanford.edu/IR-book/html/htmledition/naive-bayes-text-classification-1.html#eqn:multinomial6>

Appendix:

* 1. **– Sentiment Analysis**



* 1. **– Lie Detection**

