Linear Classifiers

Yanze Wang
Department of Computer Science
University of Southern California
Los Angeles, CA 90089 USA
yanzewan@usc.edu

Abstract

This document is about classifying four different kinds of classification tasks using linear classifiers. Using language corpus, Naive Bayes model and Perceptron model are employed and related result analyses are included in this document. When optimizing for different tasks, several different features are chosen and tested to get a better output of accuracy. I also make some effort to deal with the out-of-vocabulary(OOV) problem and Byte Pair Encoding(BPE) will be introduced in the article.

1 Introduction

Nowadays, we are usually troubled by the very large amount of online documents. It becomes a big problem to classify these documents into correct categories. Many algorithms are developed by researchers in this area to deal with the problem and great results are achieved.

The four corpus datasets used are related to different categorization tasks. Some of them are topical, attempting to classify sentences based on their subject matter(e.g.,sports vs. business). Some of them are sentiment, attempting to classify sentences based on overall opinion (e.g., positive vs. negative). Also, the size, genre and language types of these datasets are different so this is interesting to use different features and models to deal with them.

In this paper, I examined the accuracy of Naive Bayes model and Perceptron model to deal with this corpus classification problem. A challenging aspect of this problem is the feature extractions(e.g., Bag of words, most frequent several words, n-grams). Different features make a difference to the results of these two models. I also used the Laplace smoothing and BPE to make the classifier be able to handle words that haven't been seen before in Naive Bayes model(both) and Perceptron (only BPE). From this task, I learn quite more about how to deal with models and get a better understanding of feature extractions and model algorithms.

2 Datasets

There are four datasets. 4dim dataset is a English reviews corpus which has four sentiment and topical combined labels (pos.tru, pos.dec, neg.tru, neg.dec). The size of it is quite small. odiya dataset is a Odiya language corpus which has three topical labels(business, sports, entertainment). The size of odiya is midium. products dataset is also a English reviews reviews corpus which has two sentiment labeels (pos, neg). Ths size of it is very big. questions dataset is a uncertain language corpus which has six labels(0,1,2,3,4,5).The size of it is midium.

3 Linear Classifiers

In this task, I used linear classifiers to do classification. Naive Bayes model and Perceptron model are tested on the datasets. The philosophies of these two models are different , but both effective in this classification problem. The specific principles of Naive Bayes and Perceptron are introduced below. The key difference between these two models is that the perceptron is a online learning algorithm whose weights changes every time an instance is chosen and the Naive Bayes model is a batch learning algorithm which relies on overall dataset. To explain the model ,we assume that the feature set here is using bag of words. Instead of using sentence or a vector of boolean values, BOW uses a vector of word frequencies and It doesn't keep the word order. So the document becomes a vector which represents the number of words in this document.

3.1 Naive Bayes

The principle of Naive Bayes classifier is quite simple. Given a document *d*, which may be represented as feature sets(BOW or something else), the

classifier should return the most likely class. We call the prediction of the collect class $\hat{y}$. And

$$\hat{y} = \arg\max_y P(y|d)$$

If a word is conditionally independent of the other words, then according to Baye's rule,which we can get

$$P(y|d) = \frac{P(y) \cdot P(d|y)}{P(d)}$$

Because *P(d)* is a constant number for all classes, we can just ignore it. We need to calculate *P(y)* and *P(d|y)* which we can call them **prior probability** of the class and the **likelihood** of the document. Since document *d* can be represented as a feature set so *P(d|y)* can be represented as $P(f_1, f_2, \ldots, f_n|y)$

Applying **naive Bayes assumption**, we can get

$$P(f_1, f_2, \ldots, f_n|c) = P(f_1|c) \cdot P(f_2|c) \cdot \ldots \cdot P(f_n|c)$$

Combined together with hat y. we can get

$$\hat{y} = \arg\max_y P(y) \cdot \prod_{i=1}^{n} P(f_i|y)$$

This is the final equation for the class chosen by a naive Bayes classifier. Also, to deal with the zero probabilities in the likelihood term, we add Laplace smoothing(usually add-one smoothing) to the likelihood term to avoid zero probabilities. It also helps to deal with out-of-vocabulary. It's also a good choice to do **log** function to *P(y)* and *P(d|y)* which can help to avoid underflow and increase calculate speed.

3.2 Perceptron

There exits some problems in Naive Bayes, we ignored the relationships between words, some words are obviously not independent so we can choose not to use probabilities and use weights for features to make predictions. we create a zero vector for weights and name it as θ. The length of the vector is the length of the features from the document. Each time we calculate the result to get the most likely class of θ multiple a instance chosen from the document. That is,

$$\hat{y} = \arg\max_y \theta \cdot f(x_i, y)$$

We then compare $\hat{y}$ with y , if $\hat{y}$ = y, we keep the θ same as before. If $\hat{y}$!= y, we increase the weights of instance features to the y vector of the θ and decrease the weights of instance features to the $\hat{y}$ vector of the θ. After several iterations, when the weights of the θ becomes stable, we can get the θ which can be used for prediction. Same with above, we multiple θ with the each sentence feature vectors and find the most likely class.

4 Evaluation

4.1 Experimental Set-up

To test accuracy in local environment, I split the four datasets into 90 percent training data and 10 percent testing data in local. The lines are split into two parts which are the document and the labels. At the beginning, I implemented the two key models(Naive Bayes and Perceptron) based on the pseudo code from the book (Speech and Language Processing 3rd edition - Jurafsky, Martin) and the book (Natural Language Processing - Eisenstein) accordingly. The pseudo code for Naive Bayes are more related to the words itself instead of feature vectors, so I made some changes to the pseudo code according to my own features(which is at beginning Bag of words). The ***fit*** function in naivebayes.py is the Implementation of Naive Bayes most related to words itself and the ***train*** function used feature vectors to realize Naive Bayes. They gave out the same result but actually realized from different method. I then saved logprior, loglikelihood,V and features into the model which are used for further classification. The Perceptron model only uses feature vectors to realize.

As for the feature part, at the beginning, I didn't convey the words into counts, I just use the split words as one-hot vector to test the accuracy of two models which is the **get_features_4** function in Features.py and then I used bag of words as features which count the frequencies of words to test the accuracy of two models(whole vocabulary) which is the **get_features** function in Features.py. And then I used the top X frequency words as features to test the accuracy(X equals to 500/1000/2000/3000/4000/5000/10000) which is the **get_features_2** function in Features.py. And at last I used n-grams to test the accuracy of the two models(unigrams and bigrams) which is the**get_features_3** function in Features.py. I tried to use 3-grams to test the accuracy without using an GPU but the whole feature becomes too big for my computer to calculate and the program was killed.

To deal with out-of-vocabulary problem. I tried to use smoothing and BPE method but it didn't make a big difference. I also tried to remove the

Datasets	Navie Bayes	Perceptron	# of Features	Feature method
4dim	0.875	0.725	11022	one-hot
odiya	**0.935**	**0.915**	22654	one-hot
products	0.824	0.81	87679	one-hot
questions	0.724	0.832	7098	one-hot
4dim	0.9	0.875	11022	Bag of words
odiya	0.935	0.906	22654	Bag of words
products	0.832	0.84	87679	Bag of words
questions	0.724	0.842	7098	Bag of words

Table 1: Classification accuracies resulting from different feature types and different models.(on hidden test txt on Vocareum)

punctuation to test the output accuracy but it becomes even worse than before. These are all done in Features.py Initialization part as a preprocessing part to deal with datasets.

4.2 Results

one-hot vector and bag of words vector for Naive Bayes and Perceptron testing This part is related to the accuracy performance of Navie Bayes model and Perceptron model based on two feature extraction method. One-hot vector and bag of words all use the whole vocabulary and the difference between them is the count method of words. One-hot vector stores one to each words and Bow stores the number of words it appears in the sentence. The classification accuracy of using one-hot vector and bag of words are shown in Figure 1. Obviously, the accuracies surpass the random-choice baselines of the datasets. It is also obviously that for most cases, Bow performs better than the one-hot vector. That may be the reason that one-hot vector only consider whether words exist or not exist in the sentence while Bow consider more about words frequency which will actually more appropriate for words. Words frequency should play an important role in many text applications. While there still exists some problems in Bow for example the feature are quite sparse. So I tried to pick the top X frequency words as features based on Bow.

Picking the top X frequency words as features Let's consider to choose the top X frequency words as features to see whether we can get a better accuracies. The classification accuracy of the top X frequency words as features based on Bow are shown in Figure 2. When in the local environment, I tested X = 500,1000,2000,3000,4000,5000 and 10000 top words and find that as the number increases, the overall trend in accuracy is upward. So I chose to pick X = 5000, 10000 and 20000 to be the features size to test on hidden test file on Vocareum. As we can see, the accuracy is better for 4dim dataset compared with Bag of words and 5000-top feature method performs best. There are not obviously improvement for other three datasets even worse than before. Btw, when writing this report, I think that is a good choice not using number and using percent to pick the feature size since the size of different dataset has different feature size. That will be more accurate. But since I have included the 25% and 50% of feature size for all datasets, I think I don't need to retrain my model again since it takes quite long time to train the model especially for Perceptron model.

n-grams feature vector I also tested N-grams feature vector selection.Using bigrams to test on vocareum, for datasets 4dim and odiya, the accuracy became worse from 0.9 to 0.825 and 0.93 to 0.87. But It shows a great improvement for dataset questions which increase from 0.74 to 0.8. I think the main reason is that the number of features are too small in questions dataset and 2-grams increase the number of features from 7098 to 22614 which makes a great influence on this dataset.

OOV handling I used BPE and smoothing to deal with OOV handling problem as described in the Experimental Set-up part above. For datasets 4dim and odiya, there was very little improvement to the accuracy and dataset products were killed by my computer for too long calculating. (Actually, I think it should have a better performance for BPE but it didn't change a lot maybe 0.01 and 0.02 for the accuracy so I didn't record that. Maybe more tries should be done to deal with that). But for dataset questions, it shows a improvement in the accuracy. Using Bow as feature set and the accuracy improves from 0.724 to 0.75 for Naive

Datasets	Navie Bayes	Perceptron	# of Features	Feature method
4dim	**0.95**	**0.925**	5000	5000-top
odiya	0.92	0.887	5000	5000-top
products	0.82	0.814	5000	5000-top
questions	**0.75**	0.824	5000	5000-top
4dim	0.9	0.9	10000	10000-top
odiya	0.931	0.899	10000	10000-top
products	0.829	0.813	10000	10000-top
questions	0.724	**0.848**	7098	10000-top
4dim	0.9	0.9	11022	20000-top
odiya	0.933	0.913	20000	20000-top
products	0.834	0.828	20000	20000-top
questions	0.724	0.808	7098	20000-top
products	**0.835**	**0.837**	40000	40000-top

Table 2: Classification accuracies resulting from X top frequency features and different models.(on hidden test txt on Vocareum)

Bayes Model and 0.842 to 0.848 for Perceptron Model.

5 Discussion

For Naive Bayes classification model, it is a probabilistic machine learning algorithm based on the Bayes Theorem. I think it's hard for Naive Bayes model to overfit since it is biased and less flexible. But it is still possible for it to overfit, but for these four datasets, I think it is implemented correctly. It has some negative log-likelihood loss and it may suffer from zero probability which can be improved by Laplace smoothing.

For Perceptron classification model, learning rate helps to scale weights. We want to get a stable theta in Perceptron classification and learning rate helps to do so. But it is also acceptable to ignore learning rate since Perceptron will always give us a suitable solution as iteration continues. A too large learning rate may lead us to fail to get a solution and a too small learning rate can lead us too slow to get a solution. Learning rate is not necessary in Perceptron but it may offer some help if we choose it correctly. For overfitting, I tested some sample sizes in my local environment to see how the performance of the Perceptron model changes, I didn't see a obvious decrease to the performance of the Perceptron model so I think it's not overfitted. Loss function for Perceptron is introduced in book Natural language processing(Eisenstein,2018).

When testing for the accuracy of the two models, we should find a trade off between time and accuracy. The size of the data influence the prediction accuracy greatly and doing a topical classification seems to be more accurate than a sentiment classification(odiya vs. 4dim and products). Also, more class leads to a more bad classification result(questions vs. odiya). The words vocabulary of the datasets also has a big impact on the results, dataset questions only have 7098 words totally in the train file and with a n-gram feature selection,it becomes quite better than before.

At last , I tried to do some tries to see how to get a trade off between the iteration times in perceptron and the accuracy output. It seems the accuracy increase quickly when itertaion increase from 1 to 50 and becomes stable between 50 to 200 times.

References

James H. Martin Daniel Jurafsky. Speech and language processing.

Jacob Eisenstein. 2018. Natural language processing. *Jacob Eisenstein*.

Bo Pang, Lillian Lee, and Shivakumar Vaithyanathan. 2002. Thumbs up? sentiment classification using machine learning techniques. In *Proceedings of the 2002 Conference on Empirical Methods in Natural Language Processing (EMNLP 2002)*, pages 79–86. Association for Computational Linguistics.

F. ROSENBLATT. 1958. The perceptron: A probabilistic model for information storage and organization in the brain.