COMP9318 Final Project Report

(SVM)

Member1: Bin Zhu (z5137034)

Member2: Bihe Zhu (z5149413)

1. Abstract

Our project requires us to develop a technique to fool a binary classifier, which is able to classify a paragraph of words to 2 categories (class-0, class-1). With partial access to the given training data, we are required to train SVM to make it be capable of classifying data into two categories with high accuracy. Based on this, we will change some words in the test data(belongs to class 0) to mislead the classifier.
To be more specific, the more test samples (after our changing) can be classified to class 1, the better is our algorithm.
We will discuss the specific process of training SVM, and the difficulty we met and the future improvement we may apply.

2. Methodology

2.1 Training of SVM

Support Vector Machine is widely used in supervised machine learning and used for classification and regression analysis. The most interesting part of it is the usage of kernel, which can efficiently help us to find hyper-planes for classification.
Many available kernels can be used in SVM, but we choose the most simple one -- linear kernel for this project. At first we used a very some iteration number and which means we have much mistake in calculating weight for each word. Then we add the iteration up and make the margin wider to get better performance.
Besides the difficulty in selecting kernels, we also had trouble in training data. SVM takes 5 or -5 for output of y, and a list of sub-list as x input that is a bag of words from 540 paragraphs(including 180 paragraphs of class-1 and others of class-0), then we use these distinct word elements to create dictionary to get weights for each one of them.
For instance, we set initial weight for all words is 0, and the learning speed Seta= 0.001
For each class-1 paragraph,e.g., if we have [a,b,c,d] in it.
So we add Seta to the weight of a,b,c,d to make the y of this paragraph to be more than 5(we set it to 5 to make the difference of two categories bigger)
Just like we did above, for a class-0 paragraph, e.g., if we have [a,c,e,f,g].
Then we decrease (number of class-1 paragraphs/number of class-0 paragraphs) times of Seta on weights of a, c, e, f, g to make the output y smaller than -5.

And we repeat this process 2000 of times to offset any small over-fitting and to avoid under-fitting.

2.2 Modification

To fool the classifier, we develop an algorithm to modify some exact words in test_data.txt. More specifically, we replace the most “class-1 like” words by most “class-0 like” words and in some cases, we add some “class-0 like” words to the original test paragraphs.

Since the number of steps we can make is limited to 20, so we need to consider “adding ” and “replacing” which one is more appropriate for any specific situation. And after several experiments we find out that we need to do the replacement first and after we have changed all the “class-1 like” words to “class-0 like” words, the 1-step cost “adding” is more efficient than 2-step cost “replacing new words into ‘class-0 like’ words”.

For example, in test paragraph we have [h, i, j, k] and the weights of them are [2,6,0,-7]. At first we need to find out the biggest weight word and that would be i, then according to the weight-dictionary we learned in the training process, we find the smallest weight word (for example, here is also k). Because we already have word k in the paragraph, so we find the second smallest weight word (for example it’s d). So we replace i with d, so on so forth. After we replaced all the positive weight words (“class-1 like” words), we start to add negative words (“class-0 like” words) into the test paragraph.

So just like this, we make all the modifying steps by very simple codes and have the accuracy around 72%.

3. Conclusion and Self-Estimation

In conclusion, we have a better understanding of SVM and kernels after working on the project. We also had a great chance to practice our problem solving skills.

But because of the lack of time, we haven’t tried the polynomial kernel to solve this problem, maybe it will have higher accuracy on classifying data.

Anyway, we successfully developed an algorithm to fool the classifier at more than 71% accuracy.