**z5135897**

**Mingkai Ma**

**Short Abstract**

This report records the steps and methodology that I use in this project. This project is about SVM, support vector machines. An algorithm is devised to fool a binary classifier, this classifier is based on svm and it is unknown for us before testing. Therefore, it is necessary to tuning parameters to get a local svm classifier, this is implemented based on sklearn in Python. Then the local svm classifier is used to devise the algorithm to modify the test data files. The success %-age I achieved is 67% up to now.

**Introduction**

Support vector machines is a binary classification model. It is widely used in text classification.

Before we build a text classifier based on svm, it is necessary to extract features from text. The Bernoulli model is used to represent a document. The document is represented by a binary vector. Let b be the feature vector for the document D, then t’th element of b, written bt, is either 0 and 1 representing the absence or presence of word wt in the document.

The target classifier is based on support vector machine. A support vector machine is a discriminative classifier formally defined by a separating hyperplane. SVM maximizes the margin around the separating hyperplane. Support vectors are the data points that lie closest to the hyperplane and the decision function is fully specified by these support vectors. Mathematically, this is a quadratic programming problem. Quadratic optimization algorithms can be used to identify which training points are support vectors with non-zero Lagrangian multipliers .

For linear SVMs, the dual problem is:

= 0

the decision function is:

For non-linear SVMs, the general idea is that: the original feature space can always be mapped to some higher-dimensional feature space where the training set is separable. Therefore, kernel trick is very important for non-linear SVMs. A kernel function is some function that corresponds to an inner product in some expanded feature space. For linear SVMs, just explained above, the kernel function is just the inner product between two vectors. For non-linear SVMs, every datapoint is mapped into high-dimensional space by some transformation , so the kernel function is:

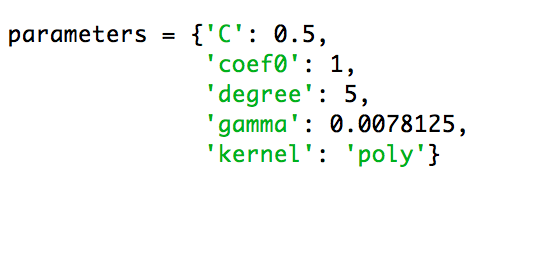
For non-linear SVMs, the dual problem is:

= 0

the decision function is:

**Methodology**

Firstly, grid search in sk-learn is used to find the best parameters for the local SVM. It is necessary to mention that grid search in sklearn is kind of slow when the parameters range is large. The parameters that is captured in the experiment is:



The kernel function is

The decision function is:

The SVM is implemented by sklearn.svm.SVC, therefore can be accessed through the members , holds support vector , and holds .

We are trying to fool the target classifier by modifying the test data according to the local SVM we have now. For every instance in test data, we are trying to minimize the value of:

For instance x, if , x will be classified as class 1, we want to fool the target classifier, so we need to minimize d(x) as much as possible, if we can minimize d(x) so that d(x) < 0, then we successfully fool the target classifier for test instance x. In order to minimize d(x) as much as possible for every test instance x, the following algorithm is devised:

The kernel function is

Firstly, put all , which is accessed from into a list, and sort according to the absolute value and reverse the list. Loop through the list, starting from the largest value of .

If > 0, we need to minimize inner product between . Because are all binary vector, we loop through simultaneously, if t-th element of are all 1, we need to change t-th element of as 0, which means t-th word is deleted from test instance.

If < 0, we need to maximize inner product between . Because are all binary vector, we loop through simultaneously, if t-th element of is 1 and t-th element of is 0, we need to change the t-th element of as 1, which means t-th word is added to test instance.

A counter is needed to make sure that only 20 modifications is made for each instance.

After modified the test features matrix, a new modified file is created according to the modified test features matrix.

**Results and Conclusions**

Success rate:



Before I achieved 67%, I got 65.5%. At that situation, for the local SVM I got, the kernel function is linear, and the only parameters that needed to be tuned is C. And the kernel function is inner product of two vectors, no transformation. After I changed the kernel function from linear to poly, the success rate is improved slightly, which means for this dataset, poly kernel function would be better than the linear.

At beginning, I also tried another algorithm. Find the nearest support vector for every test instance, and started from the nearest support vector, if there is a difference between support vector and test instance, modify the test instance to make it more like the support vector, the result is not good, then I changed my mind, trying to get solutions from the decision function:

and the results show that my final solution is better than the algorithm that I came up with at the very beginning.