Ensemble Learning

Dept of Computer Science and Technology
Chen Liu, 刘晨, 2011010539

0.Problem&Data

The dataset used in our experiments is WEBSPAM-UK2006 1 , which is obtained from a crawler of the .uk domain. There are total 1803 spam hosts and 4411 normal hosts which have both content and transformed link features. Every host has 96 content features and 138 transformed link features. 138 transformed link-based features can be divided into five categories: Degree-related features, PageRank-related features, TrustRank-related features, Truncated PageRank-related features and Supporter-related features (The feature file is " ./exp2 webspamdata/ContentNewLinkAllSample.csv").In this experiment, we compare different ensemble learning algorithms with different classifiers.

1.Introduction

In this experiment, we need to implement serveal ensemble learning algorithms including bagging and adaboost.M1. In addition, we take advantage of serveal baseline learning algorithms including decision tree and support vector machine. Then we combine baseline learning algorithms and ensemble learning algorithms and achieve a better performance.

In this experiment, at least 4 combinations are implemented: D-tree+bagging, D-tree+adaboost.M1, SVM+bagging, SVM+adaboost.M1. What's more, some thicks, how to generate training set for example, will be used to polish up these algorithm's performance on a particular dataset, especially a asymmetric one.

2.Design and Implement

We implement this project with Java. We use the whole dataset into training set and testing set. We use cross validation to revaluate our classifiers.

The abstract class 'Classifier' offers interfaces of classifiers, including training, testing one instance and testing instances in a file. For each baseline learning algorithm, we implement a concrete classifier.

We use adaptor design pattern to transfer the source data file into the input file of a particular classifier. The abstract class 'FileAdaptor' offers interfaces of this adaptor. For each concrete classifier, we implement a corresponding adaptor.

For bagging algorithm, we have a class called 'GenSubset'. This class splits the whole training dataset into serveal training files. We simply pick out an instance by random from the dataset into a training file and repeat this process for many times. Due to this asymetric dataset, there will be more negative instances (normal E-mails) than positive instances (spam E-mails) in each training file. That's to say, asymetry in the dataset causes asymetry in the training files if we generating training files randomly.

In addition to 'GenSubset', there is also a class 'SymGenSubset' that generate symmetric training files. The generator pick out positive instances for half of the instances in the training file and negative instances for another half.

For adaboost.M1 algorithm, there is a class called 'GenTrainFile'. Given the dataset and the weights of each instances, the class will generate a training file by picking out instances with replacement. The possibility of an instance's being picked out is in proportion of their weights. A bigger weights means more chances to be picked out.

In this project, we have referred libsvm to implement SVM algorithm and c4 to implement decision tree algorithm. Because the decision tree only accept discrete values, we have to discretize the continuous property of each instance in the data set. We split each property to 4 level that the number of instances in each level is almost the same.

The packages and classes in this projects is shown as below:

```
-adaboost
    -Adaboost.java          //implement adaboost algorithm
    -GenTrainFile.java      //generate training files according to weights of each instance
-bagging
    -Bagging.java           //implement bagging algorithm
    -GenSubset.java         //generate training files by picking out instances randomly
    -SymGenSubset.java      //generate symmetric training files
-config                     //global configure parameters and abstract classes
    -Config.java
    -Classifier.java
    -FileAdaptor.java
    -SplitSrcFile.java      //split the data set into training set and testing set
-decisionTree_api                   //api of decision tree library(c4)
    ...
-decisionTree_exec
    -DecisionTreeClassifier.java    //implement decision tree interface for upper ensemble learning algorithm.
    -DecisionTreeFileAdaptor.java   //transfer source file to the input file of decision tree interfaces
-libsvm_api                         //api of SVM library(libSVM)
```

...
-libsvm_exec
  -LibsvmClassifier.java   //implement SVM interface for upper ensemble learning algorithm.
  -LibsvmFileAdaptor.java   //transfer source file to the input file of SVM interfaces

3.Experiments

We run each parameter configuration for 3 times to avoid the side effect of random picking. In the table is the average value of these three experiments

note: 'Asym' refers to 'Asymetric','Sym' refers to 'Symetric'. We use GenSubset class to generate asymetric training set and SymGenSubset class to generate symetric training set.

Bagging+SVM

1.Variable: the number of classifiers

Table 3.1:Each baseline classifiers has a training file of 2000 instances

Asym/Sym Training Set	Asym	Asym	Asym	Asym	Sym	Sym	Sym	Sym
Number of Classifiers	1	5	9	15	1	5	9	15
Precision	0.71451	0.71483	0.71451	0.71499	0.72111	0.71998	0.72111	0.72014
Recall of Positive Instance	0.01885	0.01885	0.01830	0.01996	0.04381	0.04104	0.04547	0.04215
Recall of Negative Instance	0.99886	0.99931	0.99909	0.99909	0.99795	0.99750	0.99727	0.99727
Overall Recall	0.50886	0.50908	0.50869	0.50952	0.52088	0.51927	0.52137	0.51971

2.Variable: the size of training files

Table 3.2:Each bagging system has 5 classifiers

Asym/Sym Training Set	Asym	Asym	Asym	Asym	Sym	Sym	Sym	Sym
Size of Training File	100	500	2000	5000	100	500	2000	5000
Precision	0.70984	0.70984	0.71483	0.71660	0.71178	0.71548	0.71998	0.71982
Recall of Positive Instance	0.00000	0.00110	0.01885	0.02662	0.00942	0.02440	0.04104	0.04159
Recall of Negative Instance	1.00000	0.99954	0.99931	0.99863	0.99886	0.99795	0.99750	0.99705
Overall Recall	0.50000	0.50032	0.50908	0.51263	0.50414	0.51118	0.51927	0.51932

Bagging+DecisionTree

1.Variable: the number of classifiers

Table 3.3:Each baseline classifiers has a training file of 2000 instances

Asym/Sym Training Set	Asym	Asym	Asym	Asym	Sym	Sym	Sym	Sym
Number of Classifiers	1	5	9	15	1	5	9	15
Precision	0.85533	0.91857	0.93579	0.94014	0.85082	0.91375	0.92839	0.93456
Recall of Positive Instance	0.73765	0.85357	0.88851	0.90293	0.82418	0.93677	0.94897	0.97966
Recall of Negative Instance	0.90342	0.94514	0.95512	0.95534	0.86171	0.90433	0.91998	0.91612
Overall Recall	0.82054	0.89936	0.92182	0.92914	0.84294	0.92055	0.93447	0.94789

2.Variable: the size of training files

Table 3.4:Each bagging system has 5 classifiers

Asym/Sym Training Set	Asym	Asym	Asym	Asym	Sym	Sym	Sym	Sym
Size of Training File	100	500	2000	5000	100	500	2000	5000
Precision	0.81348	0.86739	0.91857	0.94545	0.81879	0.85629	0.91375	0.93950
Recall of Positive Instance	0.57681	0.78424	0.85357	0.89351	0.82362	0.88685	0.93677	0.92346
Recall of Negative Instance	0.91022	0.90138	0.94514	0.96668	0.81682	0.84380	0.90433	0.94605
Overall Recall	0.74352	0.84281	0.89936	0.93009	0.82022	0.86532	0.92055	0.93475

AdaBoost.M1+SVM

Table 3.5: Each baseline classifiers has a training file of 2000 instances

Number of Classifiers	1	5	9	15
Precision	0.71403	0.72235	0.31654	0.31622
Recall of Positive Instance	0.01774	0.04825	0.99445	0.99389
Recall of Negative Instance	0.99864	0.99795	0.03944	0.03921
Overall Recall	0.50819	0.52310	0.51695	0.51655

Table 3.6 Each adaboost system has 5 classifiers

Size of Training File	100	500	2000	5000
Precision	0.71210	0.71676	0.72235	0.72207
Recall of Positive Instance	0.01053	0.02995	0.04825	0.04880
Recall of Negative Instance	0.99886	0.99750	0.99795	0.99727
Overall Recall	0.50470	0.51372	0.52310	0.52303

Adaboost.M1+DecisionTree

Table 3.7:Each baseline classifiers has a training file of 2000 instances

Number of Classifiers	1	5	9	15
Precision	0.86338	0.91761	0.95430	0.96090
Recall of Positive Instance	0.76039	0.86078	0.91569	0.93011
Recall of Negative Instance	0.90547	0.94083	0.97008	0.97348
Overall Recall	0.83293	0.90081	0.94289	0.95180

Table 3.8:Each adaboost system has 5 classifiers

Size of Training File	100	500	2000	5000
Precision	0.82394	0.87078	0.91761	0.94835
Recall of Positive Instance	0.65501	0.76650	0.86078	0.90460
Recall of Negative Instance	0.89299	0.91340	0.94083	0.96623
Overall Recall	0.77400	0.83995	0.90081	0.93541

4.Conclusion and Analysis

Why SVM Fails?

From the experiments above, we can see that the SVM performs poorly in this classification problem no matter it combines bagging algorithm or adaboost.M1 algorithm. We notice that the SVM-based classifiers performs much better on the training set.For example, the SVM's performance in the bagging system is shown in the table below.

Table 4.1: Each baseline classifier trains 2000 instances

Asym/Sym Training Set	Asym	Asym	Asym	Asym	Sym	Sym	Sym	Sym
Size of Training File	100	500	2000	5000	100	500	2000	5000
Precision	0.70984	0.71258	0.75142	0.88327	0.71542	0.72787	0.82485	0.97001
Recall of Positive Instance	0.00000	0.01016	0.14438	0.59974	0.07117	0.06858	0.40118	0.90109
Recall of Negative Instance	1.00000	0.99954	0.99931	0.99863	0.99886	0.99795	0.99750	0.99705
Overall Recall	0.50000	0.50493	0.57196	0.79945	0.52497	0.53297	0.69960	0.94963

Table 4.2: Each ensemble classifier has 5 baseline classifiers

Asym/Sym Training Set	Asym	Asym	Asym	Asym	Sym	Sym	Sym	Sym
Number of Classifiers	1	5	9	15	1	5	9	15
Precision	0.79304	0.75142	0.73575	0.72079	0.84003	0.82485	0.81402	0.79251
Recall of Positive Instance	0.29099	0.14438	0.09077	0.03937	0.45461	0.40118	0.36457	0.29062
Recall of Negative Instance	0.99826	0.99954	0.99939	0.99931	0.99758	0.99803	0.99773	0.99765
Overall Recall	0.64462	0.57196	0.54508	0.51934	0.72609	0.69960	0.68115	0.64414

From the table above, we can find that the ensemble classifier performs better on the training set with fewer baseline classifiers, each of which has more training instances. However, SVM's performance is almost the same on the testing set. This phenomenon is called **over-fitting**. Bigger training size and fewer baseline classifiers will increase this tendency. This is partly because too many training instances per baseline classifier will cause this classifier's over–approximation to the training set, what's more, fewer baseline classifiers will make it more difficult to neutralize each classifier's over–approximation.

What is even worse is SVM's performance in adaboost.M1 algorithm. The first few SVMs' low recall of positive instances will significantly largen the weights of them and make the weights of negative instances close to 0, then much more positive instances than negative instances will be picked out as training instances for the next serveal classifiers. This causes the SVM classifiers' over–fitting to minor positive instances, which in turn leads to the low recall of negative instances. In all, there are more classifiers which is over–fitting to positive instances and the whole system will have low recall of the relatively majority, which results in even lower precision than bagging system that has the low recall of the minority.

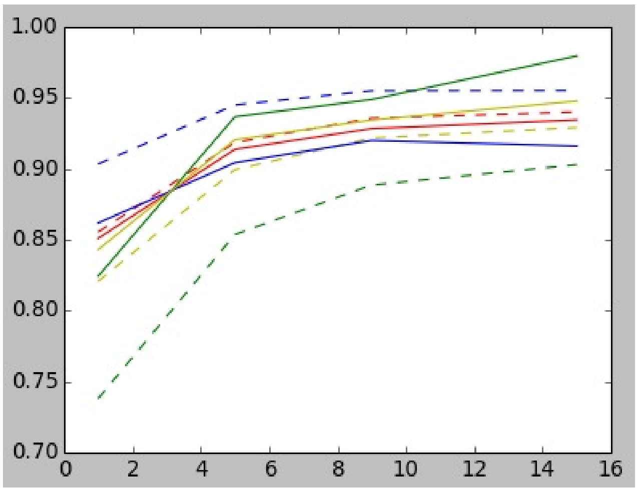
On the contrary, decision tree algorithm performance on the training set is almost the same as on the testing data. From this point of view, we can say that the SVM is a **'stronger' classifier** on this data set. However, bagging algorithm and adaboost.M1 algorithm require **'weaker' classifiers** as baseline classifier. Stronger baseline classifiers will eventually lead to over–fitting and poor performance.

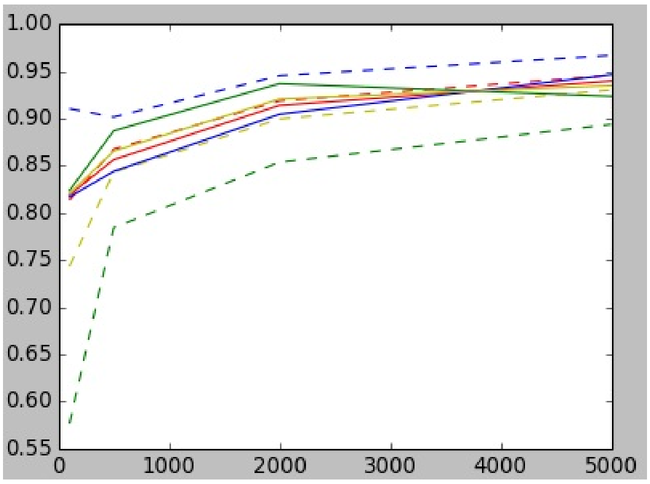
Decision Tree's Performance

Take bagging algorithm for example.

Here are 2 line charts, the upper one shows the performance with more and more baseline classifiersm, the bottom one shows the performance with bigger and bigger training set each classifier.

legeng:
dash line: asymetric training data
solid line: symetric training data
red line: precision
green line: recall of positive instance
blue line: recall of negative instance
yellow line: overall recall





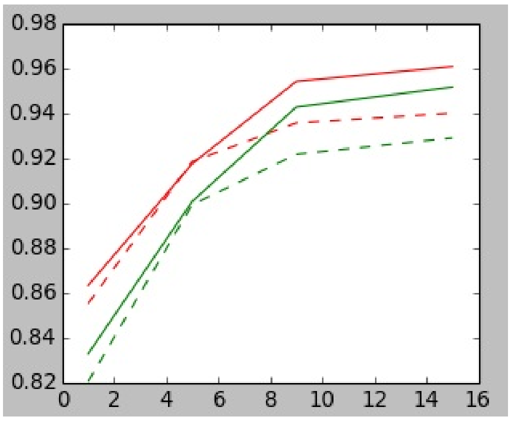
From both charts, we can discover that the decision tree performs fairly well both on the asymetric and symetric training set in the aspect of precision. However, it performs better on the symetric training set than the asymetric one in the aspect of recall, with much higher recall on the minor positive instance and little lower recall on the major negative instance.

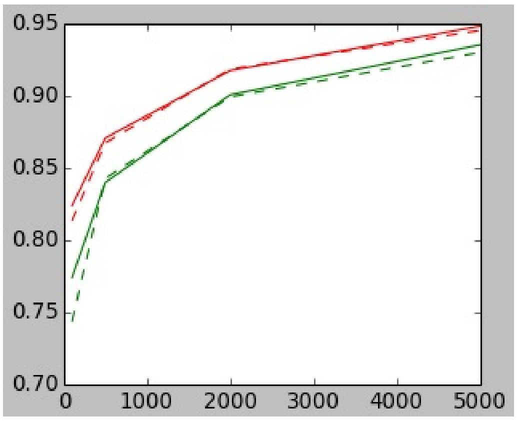
The upper chart shows the system will perform better in the aspect of both precision and recall with more baseline classifiers, while the bottom chart shows better performance can be achieved by generating bigger training set for each baseline classifiers. This is partly because more classifiers and bigger training set means more priori and adequate training, which usually leads to better performance.

Compare Bagging with Adaboost.M1

Here are 2 line charts comparing bagging with adaboost, the bagging algorithm are based on asymetric training set because the adaboost algorithm have no tricks to avoid asymetry. The upper one shows the performance with more and more baseline classifiersm, the bottom one shows the performance with bigger and bigger training set each classifier.

legeng:
dash line: Bagging algorithm with asymetric training set
solid line: Adaboost algorithm
red line: precision
green line: recall





From both charts, we can discover that the adaboost algorithm plays better than bagging algorithm in the aspect of both precision and recall. That's partly because of the adaboost's self–adjustment on the training set, which can relieve adverse impact of the asymetry of the training set. On the other hand, in the bagging system, the baseline classifiers are independent of each other and have equal weight to vote, which can suffer from the asymetry of the training set.

Reference

1. libsvm http://www.csie.ntu.edu.tw/~cjlin/libsvm/
2. c4 decision tree