

CS3244 Machine Learning

Assignment 2: Text Classification with WEKA

Shawn Tan (U096883L)

1 Introduction

Text classification is a common problem in the field of machine learning and Natural Language Processing (NLP). In this assignment, we were tasked to classify some posts on several newsgroups.

We were given stemmed texts from 5 newsgroups: `comp.graphics`, `comp.os.ms-windows.misc`, `comp.sys.ibm.pc.hardware`, `comp.sys.mac.hardware` and `comp.windows.x`. In our test set, we were given 1425 instances to classify, and 2935 training instances. Several scripts and programs were supplied to perform various tasks:

fs.php and fe.php These two PHP scripts help to extract the features from the texts using TF-IDF and χ^2 feature selection methods.

Formatting.exe This program converts the `.txt` files created by the PHP scripts into `.arff` files which can be read by WEKA.

The end result are two `.arff` files that consist of features that correspond to normalised word frequencies. These are the feature vectors which the various classifiers used will be working with. In this report, we experiment with using 3 different types of classification algorithms: k -Nearest Neighbour, Naive Bayes, and SVMs.

Our approach involves training different classifiers using each of the algorithms using the same dataset. Eventually, we take the best performing classifiers from each different algorithm,

and use these classifiers together to hopefully reduce any kind of overfitting caused by any of the individual algorithms. After evaluating this classifier, we then use this to classify our test set.

We make use of version 3.7.4 of WEKA for the tasks detailed in this report.

2 Selecting the Features

Setting an overly high value for feature selection may result in feature vectors that are too specific to the training set, and eventually cause overfitting. For our first experiment, we select only the top 50 keywords for each class for our feature vector. This resulted in 203 keywords in total.

Using the selected features, we extract the feature vectors from each of the newsgroup posts. Using this, we train three classifiers (k NN, Naive Bayes, SVM) using the default WEKA settings, and evaluate their performances before proceeding. We do this several times, with several different values of `fs_top_num`. Table 2 reports the different values we tried, and the weighted F-measure of the corresponding classifiers.

<code>fs_top_num</code>	Keywords/Features	Naive Bayes	SVM	IBk
50	203	0.738	0.77	0.732
100	428	0.74	0.801	0.758
150	641	0.743	0.814	0.767
200	857	0.74	0.822	0.774

Table 1: Experiments with the number of features used.

Increasing `fs_top_num` by 50 at each round of testing, we performed the experiment four times. We decided to use an `fs_top_num` value of 200 for our classification task, as larger feature vectors may cause classification to take long periods of time, making repeated testing difficult.

3 Tuning Performance of Individual Classifiers

In the following section we attempt to tune the performance of individual classifiers by adjusting the parameters for the different algorithms. For each algorithm, we evaluate the classifiers based on their performance on the training set and see how the classifiers can be improved.

For the following experiments with the classifiers, we use a 10-fold cross validation in all our evaluations.

3.1 Naive Bayes (NaiveBayes)

The Naive Bayes classifier does not allow for much tweaking of parameters. The default (using WEKA’s settings) classifier has an F-measure of 0.74 on the training set. It is worth noting that the F-measure on class 4 was 0.809 and has a true positive rate of 0.81 for class 3 , as this may be useful in our attempt to combine the classifiers later on.

Also, observing the F-measures for the Naive Bayesian classifier over different numbers of features, we observe that the performance of the classifier does not increase much after 0.74.

Label	TP Rate	FP Rate	Precision	Recall	F-Measure
NB	0.739	0.065	0.75	0.739	0.74

Table 2: Experiments with different exponent values in `PolyKernel`.

3.2 Support Vector Machines (SMO)

By default, WEKA’s SMO algorithm learns a classifier with a linear decision boundary. By setting the `-K` parameter, we can change this to higher exponents. This is effectively mapping the original representation of data points to a different feature space. This is expected to give us better results, since many real-world problems are unlikely to be linearly separated.

The original weighted F-measure for the SVM is 0.822. Again, we see that the F-measure for class 4 is the highest, but its true positive rate for class 0 documents is at 0.86.

We ran the training set against the SMO algorithm with the `PolyKernel` at different exponents and evaluated the results.

Label	Exponent	TP Rate	FP Rate	Precision	Recall	F-Measure
SVM	1	0.821	0.045	0.823	0.821	0.822
SVM0.5	0.5	0.748	0.063	0.75	0.748	0.748
SVM1.5	1.5	0.844	0.039	0.845	0.844	0.844
SVM2.0	2.0	0.838	0.041	0.839	0.838	0.838
SVM3.0	3.0	0.831	0.042	0.833	0.831	0.832
SVMRBF	—	0.751	0.063	0.787	0.751	0.749

Table 3: Experiments with different exponent values in `PolyKernel`.

Evaluating kernels of exponent 0.5, 1, 1.5, 2, and 3, we saw that the best performing SVM was when `-E` was set at 1.5, which performed slightly better than the default settings. This setting brings up the F-measure over all classes to over 0.8, which hopefully, gives us better performance. Beyond 1.5, the values of the weighted F-measures seem to decrease. The best reported F-measure was for class 3 at 0.869. Higher orders of exponents seem to result in poorer results, suggesting that the data does not fit well to functions of orders 2 and above.

We attempt the same classification task with the RBF kernel, but obtained results poorer than the default settings for the SVM. Again, this suggests that the RBF kernel is not a good fit for the data.

There was also a `-L` option for the kernel in order to allow the SVM to model the function using smaller orders of the variables. We again ran the algorithm on the same set of data, except now with the `-L` option turned on.

Label	Exponent	TP Rate	FP Rate	Precision	Recall	F-Measure
SVM0.5-L	0.5	0.828	0.043	0.831	0.828	0.829
SVM1.5-L	1.5	0.83	0.043	0.831	0.83	0.83
SVM2.0-L	2.0	0.836	0.041	0.836	0.836	0.836
SVM3.0-L	3.0	0.843	0.039	0.843	0.843	0.843
SVM4.0-L	4.0	0.847	0.038	0.848	0.847	0.847

Table 4: Experiments with different exponent values in `PolyKernel` using `-L` option.

Observing that there was a gradual upward trend in this case, we tried another iteration of the experiment using `-L -E 4.0`. This resulted in a classifier that yielded an average F-measure of 0.847

Eventually, we decided to use `SVM1.5` and `SVM4.0-L` for our SVM classifiers. Our experimental results are shown in Table 3. These are listed as the weighted averages of the different shown values.

3.3 k -Nearest Neighbours (IBk)

The default setting for IBk on WEKA has the $k = 1$. This results in every new instance being classified the same as the first nearest neighbour it sees. We experiment with different values of k to find a good classifier.

The classifier trained with the default settings gave an F-measure of 0.774, and an F-measure of 0.805 on class 0 and 0.807 on class 4. From the performance of the other classifiers on this classification, this suggests that instances in classes 0 and 4 are more easily distinguishable from the rest of the dataset.

We experiment with different values of k , evaluating the classifier with different k values.

Label	k	TP Rate	FP Rate	Precision	Recall	F-Measure
IB1	1	0.774	0.056	0.775	0.774	0.774
IB5	5	0.762	0.06	0.77	0.762	0.763
IB10	10	0.768	0.058	0.775	0.768	0.768
IB20	20	0.769	0.058	0.776	0.769	0.77
IB30	30	0.76	0.06	0.767	0.76	0.761

Table 5: Experiments with the number of features used.

Using the `-X` function to choose k did not seem to contribute much to the results of the classifier. One noticeable characteristic of this classifier was that it was much faster than the training times of the SVM. However, the results of the classifiers tend to be poorer than that of the SVM as well.

Weighing the distances by its inverse and similarities using the `-I` and `-F` options improve

the classifier’s performance slightly. We turn on the options using the $k = 10$ and 20.

Label	k	TP Rate	FP Rate	Precision	Recall	F-Measure
IB10-F	10	0.783	0.054	0.786	0.783	0.783
IB20-F	20	0.775	0.056	0.78	0.775	0.776
IB10-I	10	0.788	0.053	0.791	0.788	0.789
IB20-I	20	0.787	0.053	0.792	0.787	0.788

Table 6: Experiments with the number of features used.

Looking at the results in Table 6, we find that the performance of classifiers using the -I option generally do better. This may suggest that small differences in the distance between data points make a big difference for this classification problem. The fact that we see about a 2% improvement over the previous values also suggest that there are numerous cases in which there are equal numbers of the different classes in the k nearest data points.

It should be noted that the IBk classifiers generally have nearly 0% errors when run on their own training set. This is because the points to be classified fall directly on themselves, giving the same results. As such, testing the IBk classifier on the same set of data as the training set is unproductive.

3.4 Conclusion

In general, all our selected classifiers perform at over 0.75 for their F-measure, and above 75% for their average true positive rates. In comparison, a random classifier would average at a 20% rate for accuracy. In the case for our best performing classifier, **SVM4.0-L**, our accuracy is at 84%. This is also comparable with the textbook’s example from Joachims , which has an accuracy of 89%.

4 Combining the Classifiers

From each type of classifier, we pick the two best performing. Since theres only one instance of the Naive Bayes classifier, we only use one. We then combine the 5 resulting classifiers into one using the `weka.classifiers.meta.Vote` classifier. We choose to combine them by

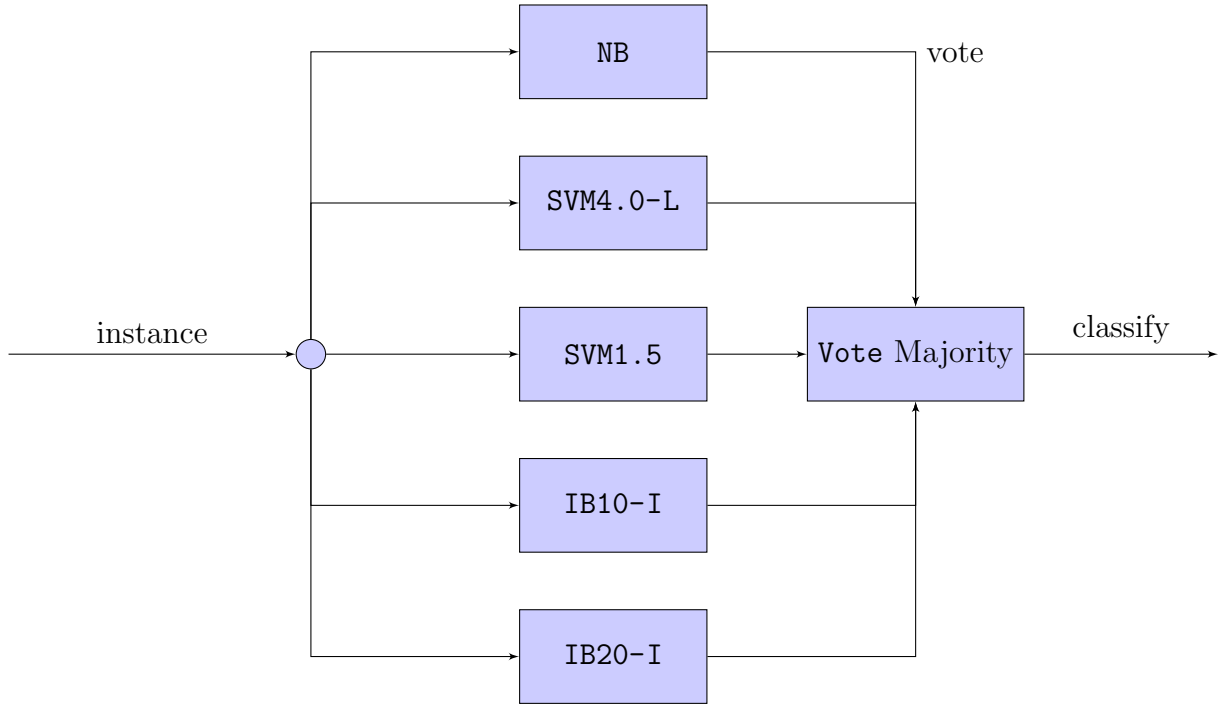


Figure 1: Combining the classifiers.

majority vote, which means that the classification will be determined by the most common classification among the 5 classifiers.

Our chosen classifiers are:

1. NB
2. SVM1.0
3. SVM4.0-L
4. IB10-I
5. IB20-I

In our selection of the classifiers we have to take into consideration the performance of each of the included classifier. Having multiple classifiers prone to errors would result in a correspondingly error prone combined classifier. The benefit of combining them, however, is due to the fact that the different algorithms have different strengths and weaknesses, and

giving only the output which majority of the classifiers agree on is likely to improve the overall performance. The error made by one classifier could then be corrected by the other classifiers.

Combining the classifiers, we run the training set against the newly formed classifier. The obtained results are in Table 7.

Class	TP Rate	FP Rate	Precision	Recall	F-Measure
0	0.998	0	0.998	0.998	0.998
1	0.998	0	0.998	0.998	0.998
2	1	0	1	1	1
3	1	0	1	1	1
4	1	0	1	1	1
Weighted Avg.	0.999	0	0.999	0.999	0.999

Table 7: Class breakdown of the performance of the combined classifier.

Recall that our best individual classifier had an accuracy of 84% while this combined classifier has an accuracy close to 100%. This goes to show that there are certain instances for which some of the classifiers fail, and, conveniently, the majority of the other classifiers correctly classify the instance. This is in agreement with our original hypothesis that the different classifiers have different strengths in classifying the various different categories, and combining them with a majority vote gives an exceptionally accurate classification.

We must keep in mind that there is a possibility that the classifier may be over fitted for the training set. However, the chances of this might be lower, due to the fact that there is cross-validation going on in between the different classifiers, giving us a much more accurate result. As such, we attempt to use the classifier to make a prediction on the training set.

Using the output and manually (and randomly) checking the classified instances, we found that the predicted values were fairly accurate. WEKA reported 17 instances for which classifier were not confident, and we found that the posts were generally short, or had few distinguishing features to determine its class. It was difficult to determine its category from the content, even if done by hand.

5 Conclusion

Through our experiments, we see that each of the different strengths when it comes to the task of text classification. In general, SVMs do well at this task, while the Naive Bayes classifier was the worse of the three. This may have been due to the type of data presented were not discrete boolean values of whether the word was present, but frequency distributions. Analysing the data in WEKA, we see that it is difficult to have any form of classification of the data done using one attribute alone. This may have also contributed to the poor performance of the Naive Bayes classifier.

The k -Nearest Neighbour instance-based learning algorithm was also not very effective at predicting the classes, although using inverse distance voting, we managed to get an F-measure of around 0.79. This suggests some of the data points are close together, and penalising points just slightly further away from the newly introduced instance improves the performance.

The SVM classifier, even with its default setting to learn a linear function to model the data, does relatively well compared to the other two algorithms. Through our experiments, we achieved the best performance when using the an exponent of 1.5, and 4.0 and lower. This exhibits some of the power of using SVMs, as simply by switching the kernel, huge gains in performance can be seen. This comes at the price of the time taken to compute the classifier. Also, our experimental data seems to suggest that the data is best approximated with a polynomial function of order 4, with perhaps larger coefficients for terms with order 1.5.

Combining the learnt classifiers, we attain a near perfect classification of the training set. This suggests that despite a maximum accuracy of 84%, the combined power of these classifiers can correct their own errors via cross-validation amongst the classifiers. It is unfortunate that WEKA toolkit does not provide analysis tools or more detailed output for its `meta.Vote` classifier, as it would be interesting to look at which instances a minority of the classifiers classify wrongly, but is out-voted by the rest to give a right answer. We suspect that this occurs relatively frequently with a good spread of different classifiers making mistakes, given the huge performance gain we are seeing.

We are aware, however, that there is a danger of the individual classifiers overfitting the training set, but we believe that the exceedingly accurate performance by the meta-classifier is due to the cross-validation taking place between classifiers. Taking into account the meta-classifier’s performance on the training set, together with the number of instances for which the meta-classifier is uncertain, we are confident with the predictions that have been made.