Knowledge Discovery and Data Mining - Assignment 2
Nicholas Hough - 312 075 510
Craig Sketchley - 312 127 790

Data Mining Assignment 2

Introduction

With increasingly larger amounts of data available across many disciplines, automatic processing and classification offer the chance to gain valuable insights with minimal human intervention. When applied to data-driven fields like computing and mobile applications, the results from such classifications can be used to increase usability, predict future trends, automatically monitor and verify user input, as well as countless other tasks critical to the evolution of a growing industry.

In this paper we will present the results of our method for classifying mobile applications into one of 30 application categories, using only the text description associated with an app. We use several Natural Language Processing techniques for preprocessing the data to extract useful features, before employing a bag-of-words model to compute a tf-idf matrix. These results are then input to a Support Vector Machine classifier, capable of classifying previously unseen apps and descriptions into their predicted category of best fit.

Method

Java was chosen as our implementation language due to the familiarity to both team members. It was decided that this assignment would use a machine library in Java and after a review of several options, Java-ML was chosen (Abeel et. al 2009). Java-ML offered a large toolkit of classification options including Naive Bayes (NB), Self-Organising Maps (SOMs), Support Vector Machines (SVMs), Random Forest (RF) and K-Nearest Neighbour (KNN). Caruana, R., & Niculescu-Mizil (2006) conducted an empirical study of classification techniques and Java-ML included some of the most successful methods.

Classifier Selection

Using Java-ML, we explored a wide range of classifiers, running tests on NB, SOMs, ZEROR, SVMs, RF and KNN with various parameters. To do this, we performed feature selection on the provided tf-idf data file, training_data.csv, to reduce the number of features on which to train our data, for both performance benefits and to avoid overfitting. We used Greedy Forward Selection with Pearson's Correlation Coefficient as the distance measure, aggressively reducing the number of features down to between 100 and 500 for each model. The main aim of this step was to generate a linearly independent subset of features that provided a good coverage of different category patterns.

During our testing, it was found that KNN and SOMs did not scale well on large datasets, and as they took too long to complete they were subsequently discarded. Below are the initial results of the various classifiers on 100 selected features with standard parameters for Java-ML.

Classifier	Precision	Recall	F-Measure
NB	0.3078	0.0802	0.1272
SVM	0.4026	0.2761	0.3276
ZEROR	0.0023	0.0322	0.0043
RF	0.3533	0.0728	0.1208

Table 1: Results from testing classifiers using 100 selected features and default parameters

It was clear from the results in Table 1 that SVM was the most successful on this dataset when using 100 selected features. It was decided that these classifiers should be tested on the full selection of 13,626 features, and we did not feature selection to see if there was any improvement. Running these classifiers on all 20,104 records would have taken a longer time than acceptable for preliminary testing, so a record subset was chosen for this test. Even with this reduced input set, the test failed with insufficient memory errors on our personal laptops, a Macbook Air with 1.8 GHz Intel Core i5 and 4GB RAM, and a Macbook Pro with 2.4GHz Intel Core i5 and 8GB RAM. The RAM requirement to load and classify the full 550Mb file was beyond our machines capabilities, and so an Amazon Web Services (AWS) EC2 instance was deployed with 8 cores and 64GB of RAM. Table 2 shows the results for NB, SVM and ZEROR from testing on the AWS instance. The RF runtime exceeded an acceptable time limit, considering it was run on a subset of the full data, and so was not able to be completed.

Classifier	Precision	Recall	F-Measure
NB	0.0006	0.0333	0.0011
SVM	0.6203	0.5755	0.5970
ZEROR	0.0027	0.0261	0.0050

Table 2: Results from testing classifiers using all 13,626 features and default parameters

SVM was a clear winner from the available classifiers, and it was therefore agreed that SVM would be chosen as the classification method.

With SVM decided, the configuration and parameter selection was explored. According to Hsu et. al (2003), (the author of the LibSVM library used as part of the Java-ML library) the recommended SVM kernel was Radial-Basis Function (RBF). It offered two hyperparameters, *C* and *gamma*, and could be configured to classify most data with reasonable accuracy. The plan was to apply the simple grid search hyperparameter optimisation to identify the best values for the two hyperparameters. The default kernel for LibSVM was a simple linear kernel, and it was found that moving from linear to RBF dramatically increased computation time. An attempt was made to optimise the hyperparameters for RBF but took too long to compute

even one iteration of grid search. Given more time, another option would have been to apply Bayesian Optimisation to hyperparameter selection, which has been shown to find optimal parameters with less experiments (Bergstra et. al 2011).

Custom Corpus Selection

Whilst we obtained reasonable results using the provided tf-idf matrix data file, we decided to explore our own preprocessing steps to generate a new tf-idf matrix. Taira et. al. (1999) demonstrated that the accuracy of SVM's in text categorization could be improved by filtering the features by their relevant parts-of-speech (POS), retaining only those that act as good discriminators between classes. They showed that this method significantly outperformed mutual information (MI) filtering in most cases, especially when the quantity of non-discriminating words was high, as is the case in our data set.

Using this premise, we discarded our previous method of feature selection using Pearson's Correlation Coefficient, and instead performed POS filtering on the description of every app, extracting only the relevant parts-of-speech. We accomplished this using the Apache OpenNLP library. We began with tokenisation, splitting the descriptions into words and removing irrelevant punctuation. Each token was then tagged with it's computed Penn Treebank POS tag. The tokens were all converted to their lowercase version, and only tokens containing purely alphabetical characters were retained, in an attempt to reduce noisy data.

In addition, only singular, plural and mass nouns were kept, and the rest discarded. By doing this, we filtered out irrelevant parts-of-speech (including adverbs, conjunctions, pronouns and prepositions) that would not greatly assist the categorisation process (as they are generally not domain specific) and reduced the number of features to assist in performance on large data. As a final step, we converted plural nouns to their singular counterpart, once more reducing the number of irrelevant features. The remaining tokens were then output to a text file, and these filtered descriptions were used as input for the creation of our own tf-idf values.

TF-IDF Selection & Normalisation

Upon selecting the word features, a new *tf-idf* matrix was computed. Two methods of normalisation were examined during trials of the classifier: the cosine normalisation and scaling attribute vector to [0,1]. The best results were observed when using cosine normalisation, and so this was used to generate the final version of our *tf-idf* matrix. It is worth noting that our matrix had significantly more features than the matrix supplied with the assignment.

Results

We tested our Support Vector Machine classifier on 3 different input data sets, randomly selecting 500, 2000 and 5000 apps for training respectively. After training, 10-fold cross validation was performed on each of the models to determine the accuracy of our approach on each data set. The results are displayed in Table 3 and Figure 1:

No. of Apps in Training Data	Precision	Recall	F-Measure
500	0.4635	0.3216	0.3797
2000	0.5489	0.4944	0.5202
5000	0.6128	0.5693	0.5902

Table 3: Results from our SVM classifier on part-of-speech extracted features

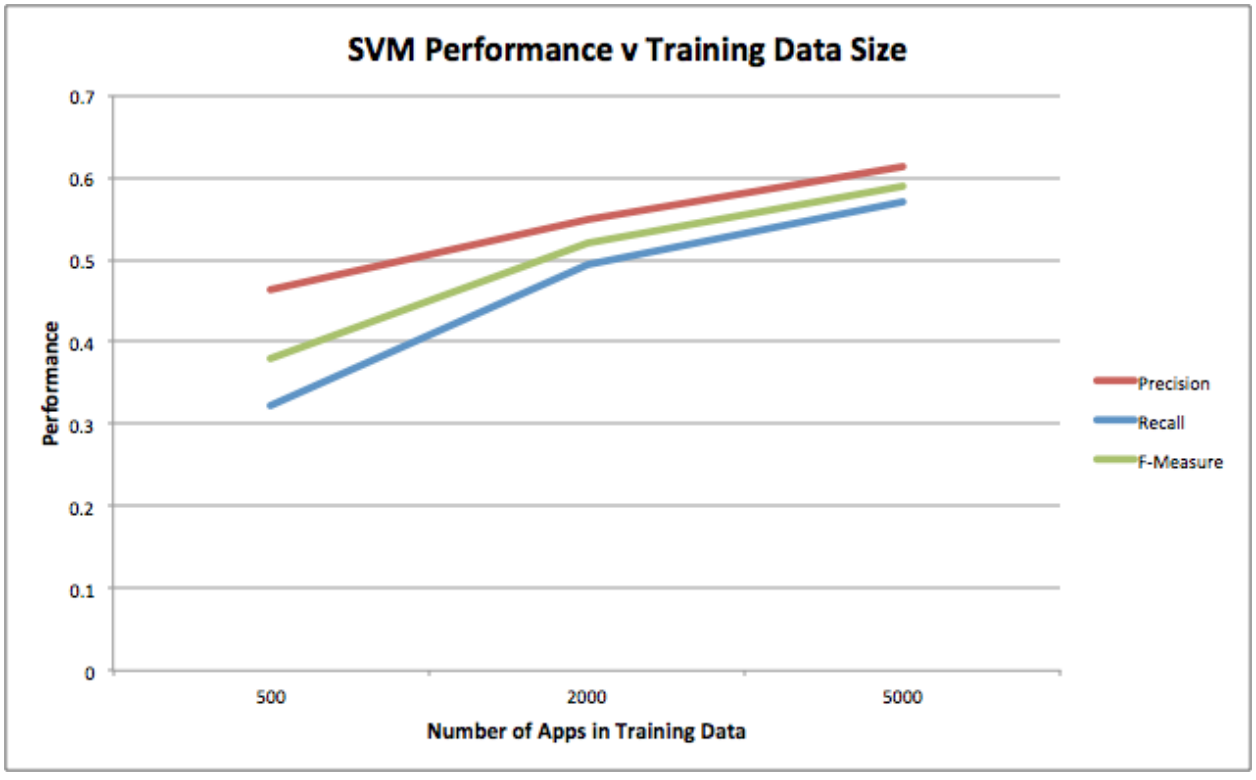

Figure 1: Performance of our POS filtered SVM v Training Data Size

Nicholas Hough - 312 075 510
Craig Sketchley - 312 127 790

Discussion

The precision and recall results for 5000 features demonstrates a good classification success rate, for what is a noisy data set with a high number of similar classes. The results outperformed all other forms of classifier from our original feature selection tests, re-enforcing the applicability of Support Vector Machines to the task of text classification. In addition, significant improvement was shown when using a high number of features on Support Vector Machines, as opposed to the early SVM tests which used aggressive feature selection.

Our part-of-speech tagging was successful in selecting a range of good quality features for our tf-idf matrix, with our 5000 app cross-validation test producing very similar results to the values obtained when using the tf-idf values provided with the assignment. An interesting observation is that our preprocessing method produced more features than the provided tf-idf matrix. Whilst large numbers of relevant features usually provide better classification properties for Support Vector Machines, the disadvantage of using such a large number of features is the high computational power required to run the classifier on large input training sets.

Given more time and resources, the full dataset of 20,104 apps could have been trained with this classifier. Considering the improvement observed in raising the training data from 500 to 5000 apps, it is projected that this upward trend would have continued, improving the precision and recall rates as more training data was added. In addition, further filtering could have been applied during the part-of-speech filtering process. A list of stop-words may have been an interesting addition, potentially reducing the number of irrelevant features, allowing for less computational time and improved feature discrimination.

Other interesting modifications could include retaining different part-of-speech tags. Whilst common nouns tend to be quite domain specific, Taira et. al. (1999) showed that proper nouns, verbal nouns and verbs also share this property. They showed that selection of the most effective part-of-speech for classification was highly domain dependant, and a single tag or a combination of part-of-speech tags may be of benefit in some cases. They did not specifically study the domain of mobile applications, and so it would be interesting to further develop our model to determine the most appropriate measure for the very specific domain.

Finally, a avenue for further optimisation of classification accuracy could be to examine the effects of alternate kernels in the Support Vector Machine model. Hyperparameters could be optimised for these kernels using grid search or Bayesian Optimisation, improving on the linear kernel used in our SVM. Once again, computational and time constraints limited this avenue for us, and significant feature reduction and/or more powerful resources would be required to explore this solution effectively on such a large set of data.

Conclusion

In summary, our Support Vector Machine classification model demonstrated adequate classification ability on a noisy data set with a large number of classes. Our use of Natural Language Processing techniques allowed us to effectively reduce the number of features in the model, and we demonstrated a significant increase in precision and recall with larger training data. Although we were limited by the computational time required to train on larger data sets, we predict that both the accuracy and precision would be increased when using our model larger input sets, given sufficient time and resources.

Code Description

For a detailed description of the code, please review the commented source and readme file included.

References

Abeel, T., Van de Peer, Y., & Saeys, Y. (2009). Java-ML: A machine learning library. The Journal of Machine Learning Research, 10, 931-934.

Caruana, R., & Niculescu-Mizil, A. (2006, June). An empirical comparison of supervised learning algorithms. In Proceedings of the 23rd international conference on Machine learning (pp. 161-168). ACM.

Hsu, C. W., Chang, C. C., & Lin, C. J. (2003). A practical guide to support vector classification.

Bergstra, J. S., Bardenet, R., Bengio, Y., & Kégl, B. (2011). Algorithms for hyper-parameter optimization. In Advances in Neural Information Processing Systems (pp. 2546-2554).

Taira, H., & Haruno, M. (1999, July). Feature selection in SVM text categorization. In *AAAI/IAAI* (pp. 480-486).