Linear and Non-Linear Classification of Text by Subject

Team: *Poco a Poco*

Charles C Onu
charles.onu@mail.mcgill.ca
260663256

Navin Mordani
navin.mordani@mail.mcgill.ca
260744902

Ramchalam Kinttinkara Ramakrishnan
ramchalam.kinattinkararamakrishn@mail.mcgill.ca
260711189

I. INTRODUCTION

In this report, we evaluate the performance of linear and non-linear classifiers in the categorization of abstracts from technical papers. We were provided with a dataset of 88639 abstracts, each classified as coming from one of 4 different subjects: mathematics, computer science, physics, and statistics. We put each abstract through a series of processes that included stemming, removal of stop words, tokenization, and vectorization, as well as optional feature selection and transformation. The feature set obtained thus was then fed into our learning algorithms to fit hyper parameters, validate models and test. We experimented with Naïve Bayes, k Nearest Neighbours (kNN), and Support Vector Machine (SVM) classifiers. Our results are consistent with the literature showing that Naïve Bayes a linear generative classifier is well suited to the text classification problem and that non-linear models like kNN and SVMs may perform better.

II. RELATED WORK

The prevalence of digitized documents today has made text classification a highly researched area in supervised machine learning. The ability to automatically classify a given text as coming from a specific subject area is vital to flexible search as well as efficient document processing. Amongst early efforts in text classification, the work Sebastiani [1] has been quite prominent, providing a synthesized review of best practices for "text categorization" – drawing from the then burgeoning field of information retrieval (IR). Since then, the bags of words representation [2] has been a generally accepted structure for text classification having been empirically validated widely [4, 5].

One common challenge with the bag-of-words approach is how quickly the dimension of the features scales. Several techniques (such as information gain, chi-square, mutual information, etc) which generally depend on ranking the features via some metrics have been proposed and used to success [2, 5]. Feature selection approaches have their limits beyond which performance may degrade. Methods to transform features from large to smaller spaces (eg: principal component analysis) have also been proposed [6].

The alternative methodology which attempts to take into account the semantic meaning of words in text classification tasks has also been explored. In this vein, Kehagias et al [7] studied the use of 'word – and sense-based categorisation'. It remains unclear whether such approaches have yielded considerable performance over the now conventional bag-of-words.

III. PROBLEM REPRESENTATION

There was a need to represent each abstract as a set of features, a format that can be used as valid input to the learning algorithms. In this section, we describe the different design decisions and processing steps that we took to achieve this.

A. *Stemming*

Stemming is the process of reducing each word to a root level such that several occurrences of the word in different grammatical forms are considered as same [2]. For example, the words 'reconstruct', 'reconstructed', 'reconstruction' will be converted to their root word 'reconstruct' by a stemming algorithm. This is important for holding on to as much semantic information as we move to other feature processing steps. Each abstract in our dataset was passed through the porter [1] stemmer from the Python Natural Language Toolkit [8].

B. *Removal of stop words and punctuations*

We removed stop words which occur regularly across documents and have no semantic implication. These include topic-neutral words like prepositions, conjunctions, and articles. These stop words were removed using the natural language toolkit.

C. *Vectorization*

Having performed stemming and stop word removal, we derived vector representations for each abstract. We applied a modification of the classic bag of words approach. In the bag of words approach, each unique word in the training set is used as a token/feature and the vector for an example could be either of 3 options:

- Binary values indicating whether or not a word occurred in the corresponding abstract;
- Integer values representing the occurrence (frequency) of each unique word.
- Term Frequency-Inverse Document Frequency(TF-IDF): are weight values that encode the relative importance of a word in an abstract. It is computed as specified in [2].

For all the above options, we grouped all numerical occurrences into one feature which records either the presence of any number in a given text (binary) or the number of occurrences of numbers (frequency). The intuition was that this grouping of numbers would reduce our number of features thereby cutting down on both space and time complexity in operations. And as seen in the experiments and results section this design decision reduced our features by over 10,000 at no cost to performance.

D. Feature selection

One challenge with the bag of words approach is that the number of features scales very quickly. We applied the chi-square [2] statistic in selecting a smaller number of features. Chi-square operates by ranking each feature according to how much it correlates to the output label.

As a feature selection method, chi-square has been shown [9] to be very reliable in dealing with the sparsity in text classification problems without losing information. In general, the approach is to compute the chi-square score for each feature, then apply cross-validation to find the top k scoring features which give comparable (or better) accuracy as using all the features.

E. Feature Transformation

Feature selection by the method of ranking can only take us so far before we experience diminishing returns from further reduction. We thus applied principal component analysis (PCA) to further transform the features into smaller space. PCA is a popular unsupervised learning method which works by the method of singular value decomposition [10], deriving a new set of features which are linear combinations of original ones.

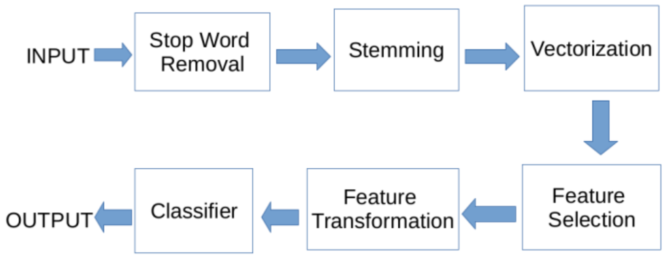

Figure 1: The feature processing pipeline

Figure 1 summarizes the workflow of our project. Including the different feature processing steps we applied to our data before passing it into the classifier. Some methods above (e.g. PCA) will not output valid values for some classifiers (eg Naïve Bayes). In such cases, the method was not applied for the classifier.

.

IV. ALGORITHM SELECTION AND IMPLEMENTATION

Dataset Split

We split our dataset into a training set and a test set in the ratio of 80:20. The training set was later sub-divided to perform k-fold cross-validation for the selection of model parameters. While the test set was left out and used to evaluate the performance of the best model from each classifier.

Evaluation Criteria

We used average accuracy on all classes as our evaluation metric. The classic performance metric for problems in information retrieval and text classification domain is precision and recall [2]. However, precision and recall only exist in 2-class classification problems. Some variants have been proposed for multi-class problems [11], however, intuition is lost very quickly and it still doesn't lend itself to a single-value metric for comparing models. Since the proportion of each class was approximately same, the average accuracy serves as a good evaluation metric on this task.

We experimented using one linear classifier Naïve Bayes and two (2) non-linear classifiers – k-nearest neighbors and support vector machines (SVM). In the following sub-section, we discuss the motivations behind these classifiers and the specific design choices made. All experiments were carried out in Python.

A. Naïve Bayes

Amongst the linear models experimented with in previous studies on text classification, Naïve Bayes has been very competitive [2]. We implemented Bernoulli Naïve Bayes by hand. For the multinomial Naïve Bayes experiment, we used Sci-kit learn's [12] implementation.

One benefit of Naïve Bayes is that it is very time efficient. Thus, in our experiments, due to time limitations, we applied Naïve Bayes in searching for optimum values for parameters used within its pipeline and that of other classifiers. For example during feature selection to select a fixed number of top ranking features based on chi-square statistic.

B. k-Nearest Neighbors

The k-Nearest Neighbors classifier is another popular choice for text classification problems [15]. It possesses a fairly huge time cost, due to its lazy approach to learning. We implemented the k-Nearest Neighbors algorithm for two (2) different distance metrics: The Hamming distance [13] and the Euclidean distance. The Hamming distance was used as distance metric when learning with the boolean bag of words representation of the dataset. While Euclidean distance [13] was used when learning with the TF-IDF representation. The number of nearest neighbors to be considered was decided empirically using k-fold cross validation.

C. Support Vector Machine (SVM)

Support Vector Machine was the final classifier we experimented with. We chose SVM because it is stable in very high dimensions and has been a huge success in handling sparse text classification datasets [15]. We used the sci-kit learn SVM implementation with the radial basis function (rbf) kernel for our experiments. In order to make, our SVM computations more tractable we first used the chi-square statistic to select a fixed subset of the vectorized features. In some of our experiments, we performed an additional step of feature transformation using principal component analysis. We show that using a number of principal components less than a hundredth of the dimension of the original features, we get comparative performance.

V. TESTING AND VALIDATION

80% of our data (training set) was used for hyper parameter fitting via k-fold cross-validation. Each abstract went through 4 feature processing steps of stop word (and symbol) removal, stemming, vectorization and feature selection. Our training set contained 70,911 examples and 66,886 features. For each of the 3 classifiers we experimented with, we trained a model using our best set of parameters, then we report the accuracy on the left out 20% (17,728 examples) of the data (test set). In the following subsections, we describe in sequence, the experiments we carried out across classifiers.

A. Naïve Bayes (NB)

In this sub-section, we show results of 10-fold cross validation, and testing using Naïve Bayes. The objective of the experiments was to find the best k, the smallest number of top ranked features that could be selected without any impact on performance. The experiments were performed using the 3 different vectorization techniques outlined in section III.

Experiment 1: NB with binary features

We ran Bernoulli Naïve Bayes using binary vectorized features. Figure 2 shows the cross-validation and training accuracy plotted against k, the number of top ranking features. It is seen that k at 30000 has the same accuracy of 89.4% as using all features.

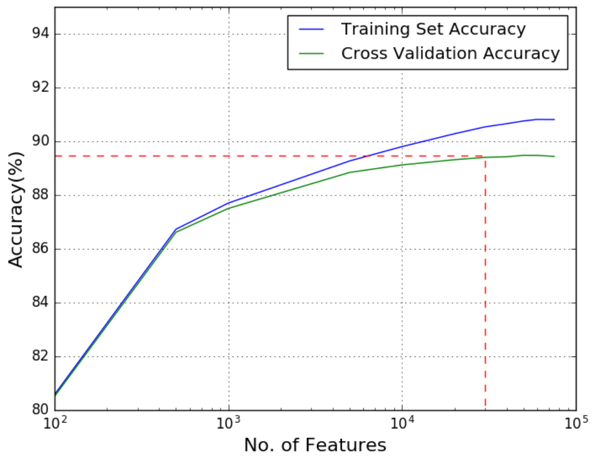

Figure 2: plot of training and cross-validation accuracy against number of **binary** features ranked by chi-square statistic (Note: x-axis is plotted on a log scale for increased resolution)

Experiment 2: NB with frequency features

We executed Multinomial Naïve Bayes using the features vectorized as frequencies. Considering that Naïve Bayes doesn't support continuous values, the implication of this was that we were considering each frequency value (0,1,2,3,…) as discrete categories. Figure 3 shows the plot of cross-validation and training accuracy versus k. Again accuracy peaked at k=30000 at a value of 89.6%.

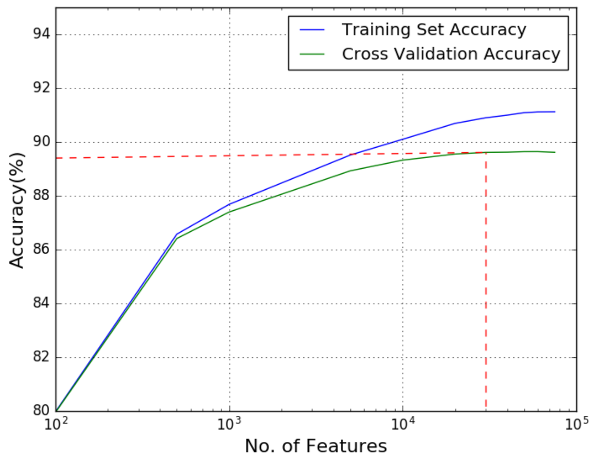

Figure 3: plot of training and cross-validation accuracy against number of **frequency** features ranked by chi-square statistic(Note: x-axis is plotted on a log scale for increased resolution)

Experiment 3: NB with TF-IDF features

We performed one final experiment based on features vectorized as TF-IDF weights. Note that the TF-IDF weights take on decimal values between 0 and 1 (inclusive). In order formulate this for Naïve Bayes, we applied a threshold, t to the weight. For values >= t, feature value is 1, otherwise 0. In this case, we show, in Figure 4, the variation of accuracies with the threshold for Bernoulli Naïve Bayes. Best cross validation accuracy of 88.82% was obtained at t = 0.1.

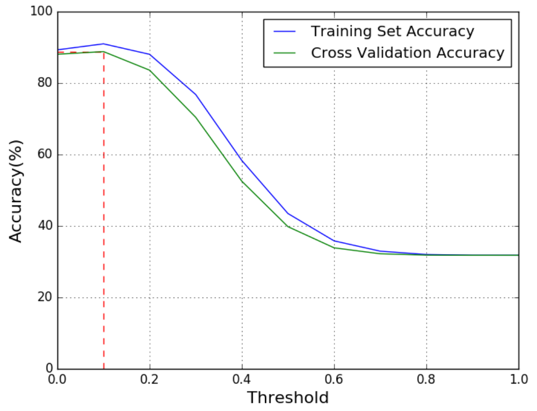

Figure 4: plot of training and cross-validation accuracy against Bernoulli threshold for TF-IDF features

In our Naïve Bayes experiments, we obtained our highest cross-validation accuracy of 89.61% using top 30,000 **Frequency** features (experiment 2). We thus re-trained a Multinomial Naïve Bayes model using the specified parameters on our entire training set. The model was evaluated on the test set to give an accuracy of 89.49%.

B. *k-Nearest Neighbours (kNN)*

In this sub-section, we show results of 5-fold cross validation, and testing using kNN. Our cross-validation objective was to find the best value of n, the number of neighbors to be considered. Given that the time complexity of kNN is in the order O(dm) (where d is feature dimension and m number of examples), it would take nearly quadratic time to run each iteration of kNN in our case where d is almost equal to m. Thus, we used only the top 500 features as ranked by chi-square scoring. (Note that this comes at only a small cost to performance. See Experiment 1 and Figure 2.) We performed experiments using separate pairings of features and distance metrics.

Experiment 4: kNN - binary features and Hamming distance
Figure 5 shows the variation of accuracy with n – number of neighbors. It is seen that number of neighbors at 79 that maximizes cross-validation accuracy is 76.1%.

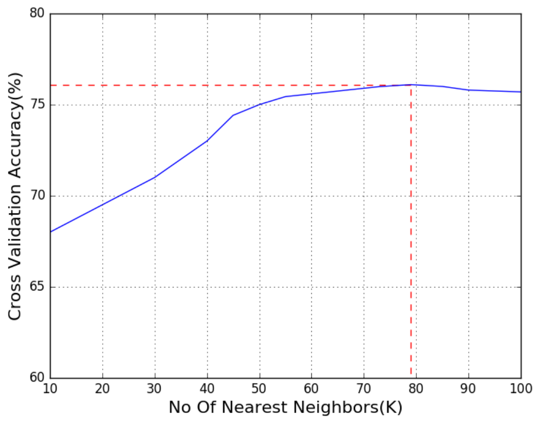

Figure 6: plot of cross-validation accuracy against no of nearest neighbors for binary features

Experiment 5: kNN - TF-IDF features and Euclidean distance
Figure 6 shows the variation of accuracy with n – number of neighbors. It is seen that a number of neighbors at 58 maximizes cross-validation accuracy is 81.3%.

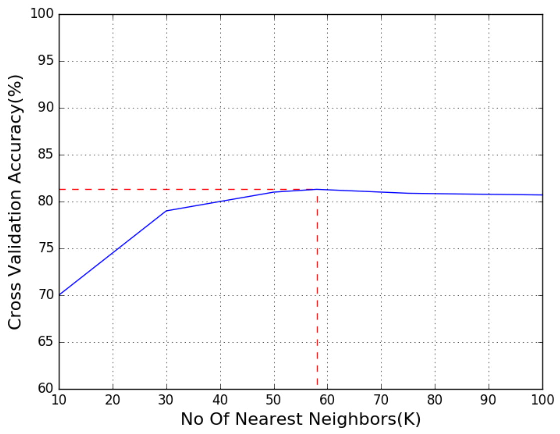

Figure 6: plot of cross-validation accuracy against no of nearest neighbors for TF-IDF features

We then applied kNN with TF-IDF and Euclidean distance metric using the 58 nearest neighbors to classify our test set based on the entire training set. Our test accuracy was 80.8%.

C. *Support Vector Machine (SVM)*

In this sub-section, we show results of 5-fold cross validation and testing using SVM with the radial basis function (rbf) kernel. The objective of the experiments was to search for the best values of C and gamma, the box constraint and width of the rbf kernel, respectively. The value of k, the best number of (chi-square-ranked) features was kept at 30,000 as obtained from the Naïve Bayes experiments. We ran experiments with binary features and TF-IDF features.

Experiment 6: SVM with binary features
Running SVM with features vectorized as binary, Figure 7 shows a plot of cross-validation and training accuracy against gamma, at C = 5 (rigorous cross-validation was done to estimate C). It is seen that the value of gamma that minimizes cross validation error is 0.00464 at the accuracy of 90.82%.

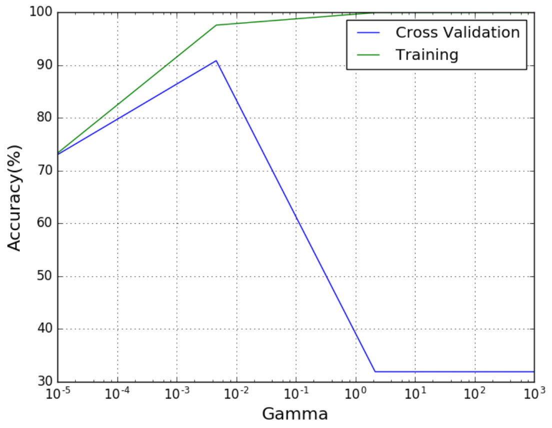

Figure 7: plot of cross-validation accuracy against gamma, the width of the Gaussian kernel, using **binary** features(Note: x-axis is plotted on a log scale for increased resolution)

Experiment 7: SVM with TF-IDF features
Figure 8 shows a plot of cross-validation and training accuracy against gamma, at C = 5, when features are vectorised as TF-IDF weights. The best value of gamma is 2 at a cross validation accuracy of 90.63%.

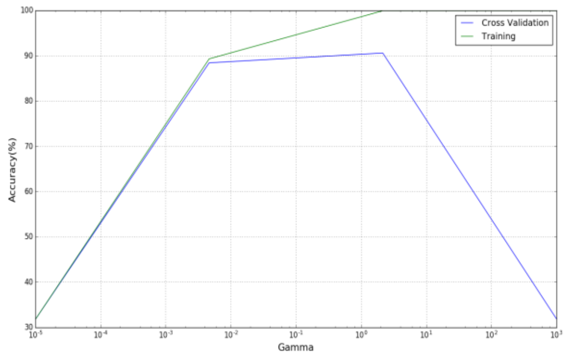

Figure 8: plot of cross-validation accuracy against gamma, the width of the Gaussian kernel using **TF-IDF** features(Note: x-axis is plotted on a log scale for increased resolution)

Taking our best set of hyper-parameters as k=30,000, C=5, and gamma=0.00464, we trained an SVM model on the entire training set. The model gave an accuracy of 91.5% on our left out test set.

Table 1 summarizes the best training, cross-validation and test accuracies obtained from each classifier.

	Naïve Bayes	kNN	SVM
Train	90.9%	86.5%	97.55
Crossvalidation	89.61%	81.3%	90.82%
Test	89.49%	80.8%	91.5%

Table 1: Summary of best models from each classifier.

VI. DISCUSSION

The biggest challenge faced in executing the project was the dimensionality of the feature space. This is inherent to text classification using the bag-of-words approach. It, nevertheless proved to be a reliable approach as is the case in the literature.

Expectedly, as classifier complexity increased from Naïve Bayes to Support Vector Machines, we saw a steady rise in generalization accuracy (with the exception of kNN. As discussed in the experiments section, kNN was trained on only 500 features due to space complexity issues.) Regardless, Naïve Bayes obviously punches above its height as a linear classifier. And coupled with its very fast training and test time, it is no wonder that it is an algorithm of choice for text classification in practice.

We acknowledge a few additional steps that could have been taken removing the constraints of time. It would have been ideal to empirically search for optimal k – number of chi-square – for each classifier as it is very likely that the non-linear classifiers would find a smaller optimum than the linear. However, as shown we only performed this search using Naïve Bayes.

In the SVM experiments, performing a finer grid search, that is, on a smaller range of C and gamma would be important to truly find the best solution [16]. Also, when learning with SVMs, while the Gaussian (or RBF) kernel is usually a good start choice due to the central limit theorem [17], it could also be helpful to explore other kernels like the polynomial kernel.

In terms of feature transformation, we were unable to use principal component analysis (PCA) in our pipeline for grid search because of the very expensive memory and time cost of computing eigenvalues for such high-dimensional data. We thus had to go with the less ideal approach of applying intuition to pick a number of principal components to compute.

It is valuable to note also that the approach we have taken completely ignores the relative position of the words in the document. Some other works have explored such as using n-grams of words [14]. Empirical results are not consistent as to the impact that has on performance.

In conclusion, our experiments validate the efficacy of the bag-of-words approach to text classification. Given good feature selection procedures, simple linear classifiers like Naïve Bayes can offer impressive performance.

VII. STATEMENT OF CONTRIBUTION

All team members were involved in the design process and strategy, after which tasks were split.

Kinttinkara implemented some of the preprocessing functions. He was responsible for implementing the Bernoulli Naïve Bayes algorithm, performing the experiments and reporting the results. He also reviewed other sections of the report.

Mordani was responsible for implementing the k Nearest Neighbour algorithm, performing the experiments and reporting the results. He also reviewed other sections of the report.

Onu implemented the feature extraction procedures and the structure for the learning pipeline used in all classifiers. He was responsible for implementing support vector machines, performing the experiments and reporting the results. He wrote the first draft of this report.

REFERENCES

[1] F. Sebastiani, "Machine Learning in Automated Text Categorization", ACM Computing Surveys, vol. 34 (1), 2002, pp. 1-47.
[2] M. Ikonomakis, S. Kotsiantis, V. Tampakas, "Text Classification Using Machine Learning Techniques", WSEAS Transactions on Computers, Issue 8, vol 4, 2005, pp. 966-974.
[3] Y. Ko, "A study of term weighting schemes using class information for text classification", 35th international ACM SIGIR conference on Research and development in information retrieval, 2012 1029-1030.
[4] Y. Bao and N. Ishii, "Combining Multiple kNN Classifiers for Text Categorization by Reducts", LNCS 2534, 2002, pp. 340-347
[5] Forman, G., An Experimental Study of Feature Selection Metrics for Text Categorization. Journal of Machine Learning Research, 3 2003, pp. 1289-1305
[6] X. Han, G. Zu, W. Ohyama, T. Wakabayashi, F. Kimura, "Accuracy Improvement of Automatic Text Classification Based on Feature Transformation and Multi-classifier Combination," LNCS, Volume 3309, Jan 2004, pp. 463-468

[7] A. Kehagias, V. Petridis, V. Kaburlasos, P. Fragkou, “A Comparison of Word- and Sense-Based Text Categorization Using Several Classification Algorithms”, JIIS, Volume 21, Issue 3, 2003, pp. 227-247.

[8] S. Bird, E. Klein, and E. Loper (2009). Natural Language Processing with Python. O’Reilly Media Inc. http://nltk.org/book

[9] M. Oakes, R. Gaizauskas, H. Fowkes, “A Method Based on the Chi-Square Test for Document Classification”, 24th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval, September 9-13, 2001, New Orleans, Louisiana, USA

[10] I.T. Jolliffe, “Principal Component Analysis, Series: Springer Series in Statistics,” 2nd ed., Springer, NY, 2002, XXIX, 487 p. 28

[11] M. Sokolova, & G. Lapalme, “A systematic analysis of performance measures for classification tasks. Information Processing and Management,” 45, 2009. pp. 427-437.

[12] Pedregosa et al., “Scikit-learn: Machine Learning in Python,” JMLR 12, pp. 2825-2830, 2011.

[13] L. Yang, “Distance Metric Learning: A Comprehensive Survey,” Department of Computer Science and Engineering, Michigan State University, 2006.

[14] W. B. Cavnar, J. M. Trenkle, “N-Gram-Based Text Categorization,” Environmental Research Institute of Michigan.

[15] G. Guo, H. Wang, D. Bell, Y. Bi , and K. Greer (2006), Using kNN Model-based Approach for Automatic Text Categorization, Soft Computing , March 2006, Volume 10, Issue 5, pp 423-430.

[16] C. Hsu, C. Chang, C. Lin, “A Practical Guide to Support Vector Classification,” Department of Computer Science, National Taiwan University, 2016.

[17] https://en.wikipedia.org/wiki/Central_limit_theorem