COMP-598 Miniproject2: Abstract classification team *Renegade Masters*

Ibrahim Abdelghany, Hardik Vala, and Edward Newell
School of Computer Science; *McGill University*
Montreal, Quebec, H3A 0G4, Canada
{*ibrahim.abdelghany, hardik.vala, edward.newell*}*@mail.mcgill.ca*

***Abstract*—These instructions give you the basic guidelines for preparing papers for icma2012/IEEE conference proceedings.To get more information, please visit our website: http://www.ieee-ICMA.org**

***Index Terms*—List key index terms here. No more than 5.**

I. Introduction

II. Related work

- stuff about information retrieval - stuff about text classification in general - stuff about abstract classification specifically - stuff about the progress in algorithms

III. Data pre-processing methods

The raw form of the data consisted of the full text of abstracts and the class of the abstract, which was either 'cs', 'stat', 'math', or 'physics'. The dataset contained some problematic entries. For example, sometimes we found that the abstract was actually only a statement that the paper had been removed. In other cases, we found what appeared to column headings (e.g. "abstract" and "class") within data rows. We removed all of these entries manually.

Before extracting features from the text, we did various things to improve results. First, all the text was lowercased, and all punctuation was removed. However, this step would tend to alter latex commands, numbers, and urls. We felt that the mere occurrence of these types of elements would be potentially descriminitive, so we included an optional pre-processing step that matched such types using regular expressions, and replaced them respectively with the tokens 'aNUMBER', 'aLATEX', 'aURL', so that counts of these types could be extracted later. We will refer to these types as "specials".

Coding of the "specials" constitutes a dimensionality reduction. We also performed other dimensionality reductions, including the removal of stop words and lemmatization. To do both of these, we used the wordnet corpus of the nltk package.

IV. Feature design and selection

We tried several alternative feature representations. The simplest of these was the binary representation, which simply recorded, for each possible word in the abstracts, which words were actually present. As a technical point, encoding this as a feature vector can take up a very large amount of space, because it would involve indicating a '0' for all words that *did not occur* in a given abstract. In general, for all our representations, we used an key-based encoding, such that the type (word) is used as a key, and the associated value, while tokens whose value is zero are simply omited. Another very similar feature set we used involved indicating the actual counts used.

We derived two refinements of the representation using word-counts, by applying a multiplicative weight to the counts. It has been found that, at least for one of the most successful induction algorithms—support vector machines (SVM)—applying multiplicative factors to frequency counts is far more important to the performance than the selection of the kernel and tuning of parameters within the algorithm. Therefore, we also prepared several representations in which the standard frequency count features were augmented by applying a multiplicative factor.

The first, TFIDF (term frequency inverse document frequency) is based on a relevance metric from information retrieval. It provides a greater weight to those words which occur in few documents, based on the notion that these words are more descriminative. TFIDF weights were calculated using the following formula:

XXXX

However, because TFIDF was developped for the information retrieval application it does not take into account the extent to which a token appears in each class. One might expect that a token which is quite common in one class but does not appear at all in another class, might be highly descriminative for the classification task; meanwhile a token which is quite rare overall, but equally represented in all classes, might not aid in the classification task. Thus tficf was introduced by X. The results of X showed that this weighting provided a performant representation, beating several information-theoretic weighting schemes. TFICF weights are calculated according to the following formula:

XXXX

Continuing the reasoning by which TFICF was developed, one might go further. Suppose one has a token which is very frequent in one class, but which still occurs only a few times in each of the other classes. One would expect this to be almost as descriminative as a feature which occured frequently in one class but not at all in another. Based on this reasoning, we developped *modified TFICF*, using the following formula:

XXXX

Our final representation used an unsupervised technique to detect the underlying 'topic' composition of abstracts, using latent dirichlet allocation. This technique works on the vector representation of words, and looks for a set of vectors which account for most of the variation seen between examples, in an unsupervised way. Each of these vectors, or 'topics' represents the presence of a mixture of word-frequencies that tend to co-occur (when their components are all positive) or be absent. The algorithm therefore reduces the dimensionality of the space of abstracts dramatically, by 'explaining' variations in frequency counts as variation in a much smaller number of topics. The topics constitute a new basis for a subspace of the original space of abstracts.

We considered applying normalization techniques to our features, but chose not to because we already had a very large number of possible pre-processing and feature representation combinations to explore, and because the abstracts tended to be all of a similar length, so we reasoned that normalizing vertor lengths would probably not alter the performance of the induction algorithms very much.

V. Algorithm selection

a) Baseline algorithm: We chose a Naive Bayes classifier as our baseline method, implemented to represent the set of word-frequencies of abstracts as a class-conditional multinomial distribution. This algorithm operates on the training data by keeping a tally of the frequency with which each token appears in a given class, and using the frequency of appearance to directly estimate the token probability.

Technically, multinomial distributions are supported over integer-valued frequency vectors (because a token cannot appear, e.g. 1.5 times in a document). However, there is no practical barrier to using the weighted representations, such as TFIDF and TFICF.

In terms of implementation, we found that validation and testing of the Naive Bayes algorithm could be made highly performant by implementing it in a lazy way: the algorithm keeps a tally of frequencies, but only calculates the necessary conditional probabilities for features when called upon to classify a given new example. Because the raw frequencies are stored, the classifier can be re-trained on a different subset of the training data by simply adding or subtracting from the tallies based on which examples are newly present (or absent) in the new subset. This makes cross validation much faster. As a result, we were able to systematically investigate all combinations of pre-treatment and feature representation, including LDA. For the LDA representation, we also used an implementation of Naive Bayes which assumes that word frequencies have a Gaussian distribution. Again this distribution embeds an impossible assumption that words can occur a fractional number of times, but there is in fact no barrier to using the such an implementation in practice.

b) Standard algorithm: For our standard algorithm, we implemented k-nearest neighbor classification. This method works by assuming a distance metric between examples within the space defined by word frequencies. The logic of using such an algorithm is that words which are close, and hence have similar word-frequencies, are more likely to be from the same class. An important optimization in this algorithm is the selection of a distance metric. While the euclidean distance metric is the most intuitive, it is not necessarily the most suitable for a given domain. In text classification, prior results have shown that cosine similarity (a pseudo-distance metric because it does not obeay the triangel inequality) is a better choice. We investigated both techniques.

Unlike the Naive Bayes classifier, kNN becomes computationally very intensive as one increases the number of examples. This is because, for each query example, the algorithm must search through the entire training data set to find the k nearest neighbors. This complexity increases linearly with the size of the training set data, the dimensionality of the data, and the number of neighbors k. As a result, we were forced to tradeoff these values for the algorithm to be feasible given our computational resources. There exist known data structures that reduce the number of distance calculations and comparisons that need to be made within the kNN algorithm, however, based on our early results from this algorithm, which were not very promising, we decided to spend more time optimizing the most performant algorithms rather than pursuing an computationally efficient kNN algorithm.

c) Advanced algorithms: We chose to investigate several advanced algorithms, given the existance of high quality and easy-to-use packages available for text classification. We used the mallet Java package to produce the LDA feature representation, and then used a variety of classifiers based on the LDA features.

Using the LDA representation, we performed the multinomial Naive Bayes, Gaussian Naive Bayes, logistic regression, random forest, decision tree, and SVM algorithms. In this case, because of the wide variety of implementations we decided to try as many as possible. We will come back to this point when we discuss the ensemble classifier.

We also tried SVM on the feature counts, weighted by TFICF. We chose this weighting outright according to the results of X, who showed that this was the most efficient representation, having also considered X,Y, and Z. and weighted feature representations. For variety, we chose Scikit-learn's implementation, rather than re-using Mallet's (which we used on LDA topics).

The major reason for focussing on testing a variety of classifiers is that we will combine the various configurations of classifiers using an ensemble classifier. We will do this by producing predictions from all of the classifiers individually, and treat these as features. Since each feature learns a different 'concept' for the classes, there biases will tend to be different. On top of these features we use a decision tree. The decision tree will rely on the output of different classifiers, based on their track-records during cross-validation.

To build the training set for the decision tree-based ensemble classifier, we record the predictions that each classifier makes during cross-validation. Thus, we get predictions by thos classifiers for all of the training data, but the prediction always made about an example that the classifier has not seen. After cross validation, we trained a each classifier on the entire training set, using the optimal settings, and then classified the test data. This produces our base-classifier feature set. We then trained the decision tree using the entire set of predictions.

At this point the reader may have expected us to cross-validate the ensemble classifier. Theoretically we could do this, but the computational complexity is prohibitive. In fact, we had to make some trade-offs in constructing the base classifiers to negotiate their complexity. Since we intended to submit the ensemble classifier, we chose not to leave aside any data in our training set since this would sacrifice our accuracy in the competition. In our case, retraining the decision tree-based ensemble classifier, after an initial validation on partial data, requires retraining all of the base-classifiers as well. Instead, *and only for the ensemble classifier*, we relied or our leaderboard score as an estimate of our true error.

VI. Optimization

Our optimization spanned multiple levels. First, by trying as many kinds of representations and classification algorithms, we attempted to optimize the representation and inductive methods. In some cases we were able to try all the representations with a given classifier, such as with Naive Bayes for example.

We also optimized specific choices within the classifiers. Some of these decisions were based on considerations for computational efficiency

- Naive Bayes - optimization of the representation: binary, counts, tfidf, tdicf modified tficf, LDA

- distance metric for nearest neighbor - Nearest neighbor – tradeoff in time complexity and number of dimensions - use of distance metric: euclidean, cosine similarity

VII. Parameter selection method

- LDA – selection of number of features - SVM what parameter values used

VIII. Testing and validation

- the cross val method used - technically, the method used cannot produce an unbiased estimate of true error for our best results, because of the repeated optimization of features on the testing data

IX. Discussion

- visualization of performance in validation

- Naive Bayes results remarkable increase in performance by removing stop words and using the tficf representation. Adding digrams also improved the result.

- Nearest Neighbor results - cosine much better than euclidean - performance was not great overall - comment on the difficulty of running with large numbers of data

- LDA - quite good results for Decision Trees, Naive Bayes, and SVM

- SVM + tficf results

- ensemble technique

We hereby state that all the work presented in this report is that of the authors.