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

The quantity of data available contitinues to grow, as does it's rate of production, with no sign of letting up. Most of the worlds information is in text form, which is both easily read by humans and, when used to relay an idea, is often more compact than other methods, such as images and sound.

Text classification has its origins in information retrieval (IR), as do many of the methods we use here. IR differs from text classification in the kind of problem it is trying to solve. In IR, documents are not seen as belonging to classes. Instead, the task is to determine, for a given query, which are the most relevant documents. Because the query is not fixed, one connot think of documents as belonging to a relevant and irrelevant set (at least until some query is given), so any pre-processing or representation methods that attempt to improve the retrieval of relevant documents must be carried out in an unsupervised fashion.

In the early 2000's, two methods emerged as front-runners in text classification task. These methods were not mutually exclusive, one being an approach to representation, that of LDA, and the other being an approach to classifier learning, SVM. SVM was found to be highly performant, beating k-Nearest Neighbour and multinomial Naive Bayes algorithms. In optimizing representation fed into SVM, it was found that applying a relevancy weighting to word counts improved the performance of the resulting classifier substancially—generally having a more important effect than the choice of kernal or tunable parameters. Two representations emereged that improved SVM accuracy. One was TFIDF, which was borowed directly from IR, and another was TFICF, which is based on a similar notion to TFIDF, but takes into account the example classes. These respectively improved SVM. But the application of SVM to LDA creates in many cases, an even more performant approach to induction. Both methods are space efficient, and once the LDA features are extracted, SVM is time-efficient during learning. learning

Here we review a broad set of representations and classifiers in a text classification task. We use a set of abstracts for

II. Related work

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

The LDA feature representation was introduced in 2003, building on the ideas of LSI. The purpose of the LDA representation is to identify a much lower dimentional subspace of the word-frequency feature space that still contains most of the variation between documents. This lower-dimensional manifold has 'topics' as its basis vertors, where each topic represents a probability distribution over words. Documents are seen as being made of mixtures of topics, and each word is seen as having arisen as the random draw from one of the topic word distributions.

Not only does the reduction in dimentsionality make subsequent inference algorithms more efficient, it actually improves the results in many cases. In much the same way that TFIDF identifies which words are relevant to the classification task, LDA forces the classifier to focus a small subset of features which, because they are capable of expressing a large amount of the variation, are likely to be relevant for classification.

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:

$$\text{TFIDF}(t_i, d_j) = \text{TF}(t_i, d_j) \cdot \text{IDF}(t_i) \tag{1}$$

$$= \text{TF}(t_i, d_j) \cdot \lg \frac{N}{n(t_i)} \tag{2}$$

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. The TFICF weight for term t_i and document d_j, which is denoted $\text{TFIDF}(t_i, d_j)$, is calculated as follows:

$$\text{TFIDF}(t_i, d_j) = \text{TF}(t_i, d_j) \cdot \text{ICF}(t_i) \tag{3}$$

$$= \text{ICF}(t_i, d_j) \cdot \lg \left(1 + \frac{C}{cf(t_i)}\right), \tag{4}$$

where $\text{TF}(t_i, d_j)$ is the frequency of term t_i in document d_j, $\text{ICF}(t_i)$ is the inverse *class* frequency, C is the total number of classes, and $cf(t_i)$ is the number of classes containing at least one document with an occurence of t_i.

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:

$$\text{MTFIDF}(t_i, d_j) = \text{TF}(t_i, d_j) \cdot \text{MICF}(t_i) \tag{5}$$

$$\text{MICF}(t_i) \equiv \lg \left(1 + C \sum_{j}^{C} \frac{1}{j} \Big(n_j(t_i) - n_{j+1}(t_i) \Big) \right) \tag{6}$$

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.

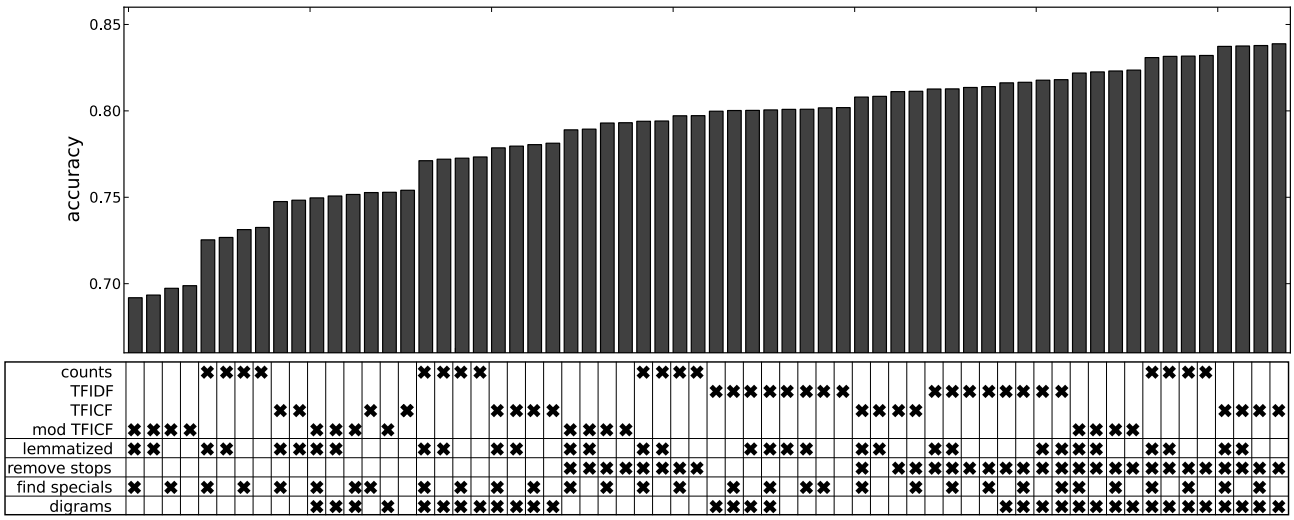

Fig. 1. Caption...

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. Additionally, we optimized parameter selections for specific induction methods, which we describe in the next section.

Optimizing preprocessing for Naive Bayes: Since the NB algorithm was not computationally intensive, we used it to systematically investigate all of the various options for feature representations. For each of the basic representations (TF, TFIDF, TFICF, and MTFICF), we tried all combinations of the pre-processing options, that is, with and without lemmatization, stop word removal, finding "special tokens", and using digrams.

The performance of the most basic representation, the term frequency (TF), provides a good entry point to understanding the effects of all of these options (see fig X). First observe that, when ranked by performance, the TF-based representations occupy a wide range of ranks. However, the TF representations are always clustered together into groups of four. Looking within these clusters, it is apparent that that including digrams, removing stop words, or doing both, successively improve the result of TF approaches, while lemmatization, and finding "special tokens" always had a slightly adverse effect.

The TFIDF representation was, in general, less affected by the pre-processing options, which can be seen (in fig X) because all the TFIDF approaches are clustered together within an upper middle range of ranks. So, TFIDF seems to make the classifier more robust against making the wrong choices. However when the optimal pre-procesning choices are made for each basic representation, TFIDF performs the worst. Interestingly, lemmatization and finding special tokens helped TFIDF, unlike for all of the other representations. Overall, TFIDF moderates the performance, and improves it overall, but the best performance was not achievable with it.

The TFICF representation performed very well. For a given set of options, it always performed better than IF, and performed best overall when the optimal options were chosen. It responded to options in the same way as IF, namely, including digrams, and especially removing stop words was helpful, but lemmatization and finding special tokens was not.

The last basic representation was our experimental MTFICF. Unfortunately it tended to perform slightly worse than plain TF for a given selection of options. However, for the optimal selection of options, it did outperform TFIDF. Admittedly, scoring the features based on their contributions to the l_1-distance between the class-conditioned distributions is very similar to what the Naive Bayes algorithm already accomplishes by estimating the conditional probabilities. Therefore, using MTFICF in connection with NB might merely overemphasizes the basis for descrimination that is already used.

kNN distance metric: We next describe optimizations for the kNN algoritm. One of the most important choices to be made in implementing a nearest-neighbour approach is to decide on to measure distance. The Euclidean distance is in many ways the most intuitive, however, it is difficult to justify why it would be meaningful in the case of text classification. One reason to doubt its usefulness is that, for very high dimensional spaces (such as we have), the euclidean distance suffers from "sphere hardening", in which all points begin to seem far from one another.

Cosine similarity provides an alternative with a good track-record as a similarity measure for text documents. Techni-

cally cosine similarity is not a true distance metric, because it does not satisfy the triangle inequality, but this leads to no technical problems in using it in the kNN algorithm. Intuitively, because cosine similarity looks only at the angle between the feature representations of documents, it is not affected by sphere hardening.

For reference, the formal definitions for the euclidian distance, L_2 and cosine distance, $L_{\cos}$, calculated between two documents having feature vectors $\mathbf{x}_1$ and $\mathbf{x}_2$ are as follows:

$$L_2(\mathbf{x}_1, \mathbf{x}_2) = \sqrt{\sum_{m=1}^{M} (x_2^{(m)} - x_1^{(m)})^2}, \tag{7}$$

$$L_{\cos}(\mathbf{x}_1, \mathbf{x}_2) = 1 - \frac{\mathbf{x}_1 \cdot \mathbf{x}_2}{||\mathbf{x}_1|| ||\mathbf{x}_2||}, \tag{8}$$

where $x_1^{(m)}$ designates the mth component of the vector $\mathbf{x}_1$, and where '$\cdot$' signifies the dot-product (inner product) of two vectors, and $||x_1||$ signifies the L_2-norm, which is the euclidean distance from the of the vector relative to the zero vector: $L_2(\mathbf{x}, \mathbf{0})$.

Substituting the cosine distance in place of the Euclidean distance when training on 10000 examples dramatically increased accuracy, which reached 0.71, up from 0.482. These tests used feature reduction based on TFICF scores, which we explain next.

kNN feature reduction: The kNN algorithm tends to be more susceptible to irrelevant features than the NB and LDA algorithms, which limit the impact of non-descriminative features. This susceptibility comes from the fact that kNN necessarily attributes equal importance to all features, up to scaling of the feature axes (and assuming isotropic distance metrics are used). Thus, eliminating the least descriminative features outright can improve accuracy with kNN.

Another reason to limit the number of features was that the kNN algorithm was very computationally intensive. Thus, our feature reduction decisions reflect both an attempt to improve accuracy as well as to make executing the algorithm on many training examples feasible. To limit the number of features in a principled way, we employed lematization, stop-word removal, and binning special tokens. We also chose not to include digrams, even though it might improve the accuracy, because it would also more than double the number of features.

In addition we tried rejecting any terms that had TFICF ≤ 1. Applying a cutoff for TFIDF is a common way of diagnosing words that are universally too prevalent, and is a way to produce a stop-word list. However, based on our results from the NB algorithm, it appeared that TFICF would be a good choice for this. When trained on 5000 examples using the cosine distance metric, eliminating the features for which TFICF ≤ 1 improved the accuracy from 0.65 to 0.69. This suggests that TFICF is a good metric for determining which terms have descriminative value in the classification task.

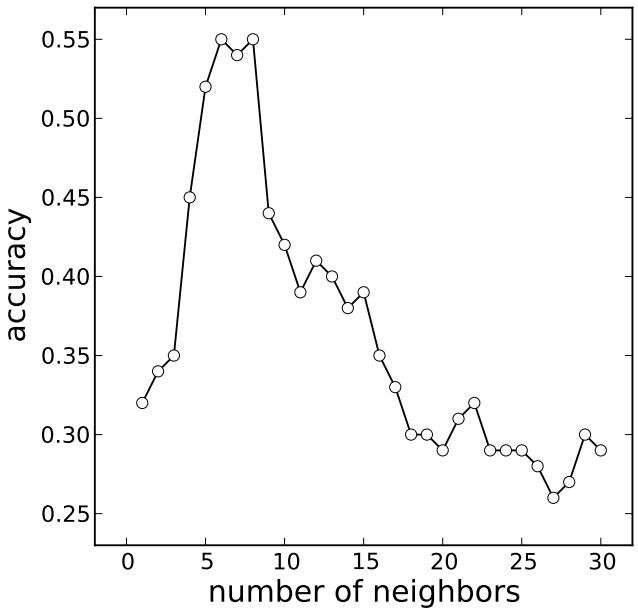

Fig. 2. Caption...

kNN optimizing number of neighbours: In the k-Nearest Neighbours algorithm, we were fundamentally limited by the computatinal intensity of the algorithm. We are aware of solutions for this challenge, such as the use of k-d trees, which reduce the burden of finding the nearest neighbours. However our initial findings were not promising, and so we made a strategic decision to focus a greater amount of effort on optimizing the most promising approaches. Nevertheless, by limiting the algorithm to work on a smaller subset of the training dataset, we investigated the effects of changing the neighborhood size k, and of using alternative distance metrics.

To investigate the effects of k, we randomly sampled 250 examples from the training data, and estimated the classification accuracy using cross-validation (our approach to validation is described in a dedicated section below).

The results show a distinct peak for a neighborhood size of 6 to 8 examples. Based on how kNN works, it is expected that neighborhood sizes that are too small yield predictions with very high variance, whereas neighborhood sizes that are too low lead to predictions that reflect the class priors rather than being dependant on the examples that are very close to the query example. Our observations are in-line with these tendencies. We do remark, however, that including all of the training data would likely shift peak accuracy to the right (i.e. giving optimality for larger neighborhoods), since the density of examples in all areas would increase, and so larger values of k would still represent neighborhoods that are quite close to the query point.

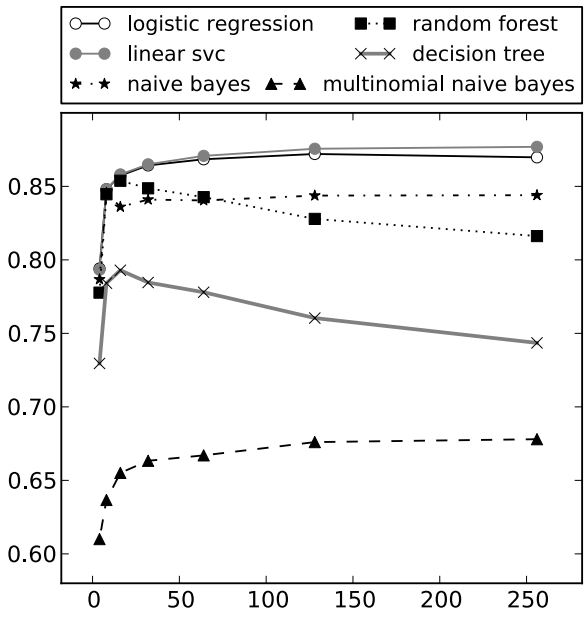

Fig. 3. Caption…

LDA: optimizing number of features: LDA is an unsupervised algorithm, however it is necessary to specify the number of features, or topics, that the algorithm should find. Topics should not be confused with classes. Topics represent, in a geometric sense, the vectors of greatest variation, in the space of word counts. Taken together, these topics, span a subspace of the original feature space; this subspace is chosen in such a way that when the documents are projected onto it, most of the variation between documents is preserved. The vectors are called "topics" because, often, the vectors major components (terms) that are all related to a particular identifiable topic. By identifying a restricted set of topics that explain most of the variation of the documents, LDA produces a set of features that are likely to be more informative to downstream classifier algorithms than the original word frequencies.

To use LDA, it is necessary to specify the number of features (topics) that are to be extracted. We repeatedly ran LDA, while specifying different numbers of features, to see the effect on a suite of downstream classifiers (see fig X). We investigated using between 4 and 256 topics; beyond 256 features the the feature extraction became too computationally intensive.

In general, the dowstream classifiers responded in one of two ways. Either the classifier continually improved as more features were added (including multinomial naive bayes, naive bayes, linear SVC, and logistic regression), or the classifier had peak performance near 16 features (including the decision tree, and random forest classifiers). The naive bayes classifier is a bit more difficult to place. It technically has a maximum at 8 features, but this is unusual for Gaussian naive bayes; normally adding more features continues to improve the accuracy but with diminishing returns. It seems possible that the very high performance at 8 features is due to variance in the error estimate.

It makes sense that the decision tree and random forest method plateau in terms of accuracy. These algorithms are very susceptible to overfitting when the depth of the tree is not controlled. Since we were focussing an assessing LDA accross many algoritms, which was already computationally intensive, we did not attempt to detect and mitigate overfitting when building these trees for given LDA features.

Since we used Naive Bayes both with representations based on (weighted) word counts and LDA features, we can make a comparison accross these representations. When using word count-based representations, with the TFICF weighting, stop-word removal, and including bigrams, we achieved an accuracy of 0.839. Using the LDA features, we achieved an optimal result of 0.844. These are remarkably close, with LDA being better. In constructing the word-count based representations, we looked at many combinations of options to achieve the optimal result. This shows that LDA does yield a compact representations of highly relavent features for text classification.

VII. Testing and validation

10-fold cross validation: With one exception, the optimizations discussed above were done with 10-fold cross validation to estimate the error for each combination of representation, classifier algorithm, and parameter settings we quoted above.

Validation for kNN: The kNN algorithm used a slight variation. During each of the 10 folds of cross validation, an optimization was performed on the training portion of the data for the given fold. Using the 9/10 training data for a given fold a nested cross-validation sub-routine was executed to determine the optimal number of neighbors to use, by trying all values between 5 and 30. The nested cross-validation was the same as the outer routine, but used only the training data for the given fold of the outer routine. The optimization of k was done independantly for each fold of the outer cross-validation routine, so no information from the test set was leaked into the training data. Thus, this nested cross-validation procedure still provides an estimate of the true error.

Validation for ensemble classifiers: Above, we described the optimization of a variety of classifiers. Taking the predictions of each of these "base classifiers" as features, we then constructed an ensemble classifier, with the hope of improving performance still further.

This raised an interesting difficulty in validation: because the base-level algorithms had been optimized using the entire

training set, there could be no guarantee that cross-validation of the ensemble classifier would give an unbiased estimate of the true error. Indeed, we found that the performance of the ensemble classifier in cross-validation was in great excess of the score that would actually be achieved when posting results to the leaderboard. This demonstrates how overfitting can arise quite subtly. For this reason, we do not report further on the validation of the ensemble classifier. For the purposes of the contest, we used leaderboard scores to test different configurations for the ensemble classifier.

VIII. 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.