Fast Johnson-Lindenstrauss Transform for Classification of High-Dimensional Data

Natasha Jaques
Department of Computer Science
University of British Columbia
Vancouver, BC V6T1Z4
jaquesn@cs.ubc.ca

Abstract

Classification of high-dimensional data presents many difficulties; not only do classifiers tend to overfit the data, especially when the number of sample points is low, but the computational complexity of many algorithms renders classification of such data prohibitive. Feature reduction techniques such as Principal Component Analysis can help to alleviate these problems, but can themselves be time consuming or ineffective. This paper presents the results of applying an efficient version of the Johnson-Lindenstrauss (JL) embedding known as the fast JL transform [1] in order to reduce the dimensionality of two datasets before classification. We show that this simple random projection technique can offer performance that is highly competitive with existing feature reduction methods, and enable the classification of high-dimensional data using computationally complex algorithms.

1 Introduction

In 1984, Johnson and Lindenstrauss [2] showed that an arbitrary dataset of dimensionality d can be projected into a feature space of dimension $k \ll d$, while preserving the lengths and pairwise distances of points in Euclidean space. Remarkably, this transformation can be achieved simply by multiplying the original data by a matrix composed of entries sampled from a standard Normal distribution. This result has since been the focus of much attention, with many researchers producing improvements to the technique (e.g. [1] [3] [4] [5] [6]). Because the JL transform preserves pairwise distances, it has often been used in nearest neighbor or clustering applications [7]. However, so far most work has focused on using JL embeddings in the context of nearest-neighbor techniques (e.g. [8]), rather than examining the usefulness of JL for classification as a whole. This is unfortunate, because an efficient and effective dimensionality reduction technique could be beneficial to many real world classification applications, which suffer from high dimensionality and low cardinality data [9]. This *curse of dimensionality* [10] leads classifiers to overfit the data (i.e. learn functions that do not have the ability to generalize well to new data), rendering efforts to use these classifiers in beneficial real-world applications ineffective.

This paper will explore the use of the JL transform for classification of two real-world datasets. The first contains a set of eye gaze features that are used to predict the emotional state of participants using an Intelligent Tutoring System (ITS) [11]. The second is a high-dimensionality text classification dataset known as 20 Newsgroups [12], which contains documents that must be classified as belonging to one of 20 different newsgroups about various topics. We will use an efficient variant of the JL transform known as the Fast Johnson-Lindenstrauss Transform (FJLT) [1] to project these high-dimensionality datasets to a lower dimension, and then perform classification with a variety of algorithms. Our hypothesis is that for classification algorithms which rely heavily on Euclidean distance (such as Support Vector Machines (SVM) and Logistic Regression (LR), the information contained in the original high dimensionality space will be closely approximated, and the classifiers

will achieve good performance. We will also compare the classification performance of FJLT against other common dimensionality-reduction methods such as Principle Component Analysis (PCA).

2 Related Work

The Johnson-Lindenstrauss (JL) lemma [2] states that given an arbitrary set P of n points in d dimensions, if k is a positive integer such that $k \geq O(\frac{logn}{\varepsilon^2})$, then there exists a mapping $f : R^d \rightarrow R^k$, such that $\forall x, y \in P, \varepsilon, \delta > 0$:

$$Pr[(1-\varepsilon) \parallel x-y \parallel_2 \leq \parallel f(x)-f(y) \parallel_2 \leq \parallel x-y \parallel_2 (1+\varepsilon)] \geq 1-\delta$$

This mapping is known as a random projection, and can computed by simple multiplication with a matrix composed of entries sample from a standard Gaussian. Many results related to the JL lemma have since been published (e.g. [13] [14] [15] [16]), including an optimal bound on k: $\Theta(\varepsilon^{-2}log(\frac{1}{\delta}))$ [17].

However, the JL transform itself can be computationally expensive, especially as n and d increase. Therefore significant effort has been invested to develop more efficient methods for computing JL embeddings. In 2003, Achlioptas [13] showed that the speed of the transform can be improved by making the JL matrix more sparse, specifically by demonstrating that the entries can be chosen uniformly from $\{-1, 0, +1\}$. The sparsity of the matrix was improved to $O(\frac{1}{\varepsilon^2})$ in 2010 [5]. In 2012, Kane and Nelson [3] showed that only a $O(\varepsilon)$-fraction of the entries in the columns of the transformation matrix need to be non-zero, once again improving the sparsity of the transformation matrix and thus the computation time. This $(1-\varepsilon)$ sparsity was later shown to be optimal [18].

For the purposes of this paper we will focus on an efficient JL embedding known as the Fast JL Transform (FJLT) [1]. The problem with methods which focus solely on improving the sparsity of the transformation matrix is that if the data itself is already highly sparse (for example, bag of words representations of text documents), then it will tend to be distorted by multiplication with an extremely sparse matrix. Therefore Ailon and Chazelle developed the FJLT technique based on the *Heisenberg Principle*, which states that a signal and its spectrum cannot both be sharply concentrated [19]. They use a randomized Fast Fourier Transform (FFT) to ensure the input data is not sharply concentrated (i.e. sparse), while still improving the computation time of the original JL embedding. We will provide further details on the FJLT method in section 3.1.

The JL transform and its variants have proven to be valuable tools in a number of situations, particularly when the run-time of an algorithm depends heavily on the dimensionality of the data [13]. It has been used successfully in approximate nearest-neighbours [15], clustering [9] [20], and data streaming [21] applications. Papadimitriou [22] was able to significantly reduce the time needed to compute a low-rank approximation of a matrix A, without reducing its quality. The application of the JL transform to problems such as image hashing has also been explored [23]. For further discussion of various applications of JL embeddings, see [7].

There are few results that relate JL embeddings to classification. Theoretical results have established that the JL transform preserves margins within a constant factor [24]. There are also two studies that compare the classification performance obtained with random projection against that of PCA, another common dimensionality reduction technique. However, both studies we have found restrict their focus to the performance of a k-nearest-neighbour (kNN) classifier. The first [8] found that the performance of a random projection (RP) based on Achlioptas work [13] does not differ significantly from PCA, except that the random projection requires more dimensions to achieve the same classification accuracy. The second study [25] obtained essentially the same results; if the random projection is allowed to use a higher number of dimensions, the classification performance can exceed that of PCA, but with a smaller number of dimensions PCA outperforms RP. Although these findings are informative, the question of the effect of random projections like FJLT on the performance of other classification algorithms is still open.

3 Methods

In this section we will describe the techniques we use for dimensionality reduction, and the algorithms we use for classification. We compute our results using 10-fold cross validation, in which

the data is randomly partitioned into 10 folds, and each fold is held out to use as a test set once, while the remaining 9 are used for training. This procedure is repeated 10 times, and the results are computed as the average over the 10 testing folds. We report results in terms of both classification accuracy, as well as Cohen's κ, a measure of binary classification performance that accounts for correct predictions occurring by chance [26]. We use a similar measure, Fleiss' κ to measure classification performance when there are more than two class labels [27]. In order to compare results to the baseline and to each other, we use Tukey's Honestly Significant Difference (HSD) procedure, which allows us to make multiple pairwise comparisons while controlling for alpha inflation [28]. Our techniques are implemented using either the Weka data mining toolkit [29], or the scikit-learn Python library[1].

3.1 Dimensionality reduction techniques

The main focus of this paper is on the Fast JL Transform (FJLT) [1]. The transform is accomplished by multiplying the data with a matrix PHD. P is a $k \times d$ matrix, where with probability $q = min\{\Theta(\frac{\varepsilon log^2 n}{d}, 1\}$ each entry $x_{ij} \sim N(0, \frac{1}{q})$, and with probability $(1-q)$ each entry $x_{ij} = 0$. H is a $d \times d$ normalized Walsh-Hadamard matrix. The method for constructing H necessitates that d be a power of 2, so if it is not then the original matrix is padded with zeros to satisfy this requirement. D is a $d \times d$ diagonal matrix with diagonal entries drawn uniformly from $\{-1, 1\}$. Finally, we scale PHD by a factor of $\sqrt{\frac{k}{d}}$. The time complexity of FJLT is $O(\frac{dlogn}{\varepsilon^2})$. Our implementation can be found in Appendix A.

In addition to FJLT (described above), we also examine other techniques for dimensionality reduction. The first of these is Principal Component Analysis (PCA), which looks for existing structure in the feature space by seeking underlying components that explain many of the features [28]. It accomplishes this by looking for highly correlated subsets of features from which to create the new components. The time complexity of PCA is $O(d^3 + d^2 n)$ [30].

For our first experiment we also examine the accuracies obtained with a nested cross-validated Wrapper Feature Selection (WFS) procedure. Unlike a simple filter, WFS is classifier-specific, and assesses which subsets of features will be most useful in combination with each other by testing them with the classifier, treating the classifier as a black box [31]. In order to obtain more robust feature sets, we use nested 10-fold CV. Within each training fold, we use another 10-fold CV to perform wrapper feature selection 10 times, and select only those features that appear in more than 10% of the folds. This procedure is extremely computationally expensive, and can take several hours to select the features for use with one of the classifiers. For this reason we do not conduct WFS on the extremely high-dimensional dataset in experiment two.

Note that unlike PCA and WFS, FJLT is a totally naïve approach that uses no information about the data or the classification problem. PCA, on the other hand, looks for inherent structure within the dataset being tested, and WFS actually uses information about the individual classifiers as well as the dataset. For this reason, we would expect that PCA and WFS would have superior performance, given the information sources they exploit.

3.2 Classification algorithms

In this section we will give a brief description of the classification algorithms, and any theoretical results related to their performance, in the hopes of providing an explanation of how they will be affected by the feature reduction techniques.

3.2.1 Support vector machines

A support vector machine (SVM) is a binary classifier that often uses a kernel function to modify the data, and then constructs a linear decision boundary of the form $f(x) = sign(wx + b)$ on the kernelized data [32]. For a dataset that is linearly separable, SVM constructs a separating hyperplane that maximizes the *margin*; that is, the distance between the decision boundary and the point closest to it. SVM tends to be fairly robust to overfitting, scales well for large datasets, and is considered

[1] `http://scikit-learn.org/`

by some to be the state of the art in text classification [12]. We use a memory- and time-efficient implementation of SVM based on Platt's Sequential Minimal Optimization (SMO) algorithm [33], which is especially effective for sparse datasets.

Blum [24] gives theoretical results which show that because the dot products of vectors are approximately preserved in JL embeddings, the margin is preserved within a constant factor. Specifically, if the original data is linearly separable with margin γ, and the embedding dimension $d = O(\frac{1}{\gamma^2} log(\frac{1}{\varepsilon^\delta}))$ then under a JL transformation the data will still be lineary separable with margin $\frac{\gamma}{2}$. This gives evidence to suggest that the SVM classifier will perform well under FJLT.

3.2.2 Random forests

Random Forests (RF) are an ensemble method, in which many weak decision tree classifiers are trained, and vote for the most popular classification label. The implementation of RF used for this paper is based on a 2001 paper by Leo Breiman [34], which introduces randomness through the use of bagging (constructing each tree using a set of examples that are randomly sampled from the data set without replacement), and random feature selection (the features used to split each node are randomly selected). Breiman's method yields forests with error rates that improve upon Adaboost, and are more robust to noise.

Breiman also gives bounds on the generalization error of random forests [34]. Firstly, he uses the Law of Large Numbers to show that the generalization error converges as the number of trees in the forest becomes large, and therefore that random forests do not overfit as more trees are added. Secondly, he shows that the upper bound on the generalization error depends on the strength of the individual trees and the correlation between them. This suggests that the best RF will be constructed from a large set of highly discriminative features; however weak, interrelated features will lead to poor performance. Since we have made no effort to manually remove correlated features from either dataset, simply reducing the number of features without removing those that are related may worsen performance.

3.2.3 Naïve Bayes

Naïve Bayes is a probabilistic algorithm for supervised induction based on Bayes' rule [35]. It makes important simplifying assumptions that lead to efficient computation of the probabilities of each class label given the training data: 1) that the features are conditionally independent given the class, and 2) that no hidden attributes influence the prediction process. While these assumptions might seem restrictive, empirically Naïve Bayes achieves impressive performance on real-world data [35]. It has also been used extensively in text classification [12]. Unlike many Naïve Bayes classifiers, the Weka implementation based on [35] does not assume that each continuous feature comes from a Gaussian distribution. Rather, it builds the distribution using a series of Gaussian kernels, one for each training example, allowing it to model complex or multi-modal distributions. This method is known as kernel density estimation.

In the case where there are many weak features, none of which can distinguish between the classes, but the underlying probability of each class is higher for certain features, Naïve Bayes should show optimal performance [34]. However, this rests on the assumption that the features are independent of each other, which may not be the case with our data. We might expect that feature reduction techniques which remove any correlated features (such as PCA and WFS) will improve the performance of the Naïve Bayes classifier.

3.2.4 Logistic regression

Logistic regression (LR) is a method for modelling binary data in which a weight is learned for each input feature [36]. The equation for logistic regression is calculated using the *sigmoid* or *logit* function:

$$p(X_i) = \frac{exp(X_i\beta)}{exp(1 + X_i\beta)}$$

where $p(X_i)$ gives the probability that the sample X_i is classified as 1, and β is the vector of weights applied to the input data. Maximizing the log-likelihood of this equation gives the maximum-likelihood estimate (MLE) for the weights. The implementation of LR used in this paper is a slight

modification of ridge regression [37], which introduces a penalty term into the MLE, forcing the coefficients of some parameters to 0 and thereby achieving regularization.

Because the LR equation is calculated using the dot products of vectors, and we know that dot products are preserved within a constant factor under JL embeddings [24], we have reason to expect that LR should show good performance on data transformed using FJLT. Note however that since the time complexity of LR can be as high as $O(nd^3)$ where d is the number of features, LR is an impractical choice for classification of extremely high-dimensional data.

3.2.5 Multilayer perceptron

The multilayer perceptron algorithm is frequently used for classification of high-dimensional data [38]. It consists of a feedforward neural network made up of nodes connected in layers [39]. Each layer is fully connected to the next by a series of weighted edges. Each node or *neuron* receives as input the dot product of the values of all nodes in the previous layer and the weights of the corresponding edges [40]. It then performs an *activation function* on this input, which is often sigmoidal in nature, as in logistic regression. Once again the computation depends upon dot-products, so a JL embedding may preserve the information needed to train the MLP within a constant error factor.

4 Experiment 1 - Emotion Classification

4.1 Data

The first dataset was obtained from a study of 67 undergraduate students who used an Intelligent Tutoring System (ITS) called MetaTutor for 90 minutes [41]. The students' eye gaze patterns were collected with a Tobii T60 eye tracker. Eye gaze data takes the form of *fixations* on a single point on the screen, and *saccades*, which are movements between two consecutive fixation points. We used an open source software package called EMDAT[2] to extract gaze features that consist of statistics related to the number of fixations, fixation rate, saccade length, and saccade angles. EMDAT also includes the ability to define Areas of Interest (AOIs) on the screen that relate to components of the MetaTutor interface. We can then obtain features related to the number, proportion, and duration of fixations on each AOI, as well as the number and proportion of gaze transitions between each pair of AOIs. In total, we have 157 gaze features.

During the study, participants also reported their emotional states on a 5-point Likert scale every 14 minutes. Therefore, the classification problem for this experiment is to predict a student's emotional state given their eye tracking data. We focus on predicting the two most strongly reported learning-centred emotions: boredom and curiosity [11]. We treat this as two independent, binary classification problems. The gaze features collected before each self-report are labelled as *true* (indicating the presence of the emotion) if the Likert rating is 3 or higher, and *false* otherwise. After our data validation process (see [11] for more details), we are left with 51 participants, who each contribute 4 emotion self-reports to the dataset. Therefore we have 204 data points.

4.2 Results

The FJLT can project a set of data points into k dimensions, where $k \geq O(\frac{logn}{\varepsilon^2})$. The ε^2 term in the denominator demonstrates that the error rate depends heavily on the number of dimensions to which the data will be reduced. For our dataset of 204 points, if we wished to achieve $\varepsilon = 0.1$ we would have to project to a space of dimensionality $k = 532$ (which is far greater than our original 157 features). In order to cut the number of dimensions approximately in half, we chose $\varepsilon = 0.25$. At first glance it would appear that the FJLT technique is only appropriate for datasets of extremely high dimensionality, while for datasets of moderate dimensionality like the one described here, it could introduce a high degree of error in the Euclidean norms in the projected space.

However, despite this theoretically poor error bound, we found that FJLT achieved good performance, even as compared to PCA and WFS. Figure 1 and 2 show the classification accuracy of each technique and each classifier in predicting boredom and curiosity, respectively. In order to compare the results, we conducted a 5 (classifier) by 4 (feature reduction technique) General Linear Model

[2] `http://www.cs.ubc.ca/~skardan/EMDAT/index.html`

(GLM), with boredom accuracy and kappa, and curiosity accuracy and kappa as the dependent variables. We apply a Bonferroni correction to adjust for familywise error.

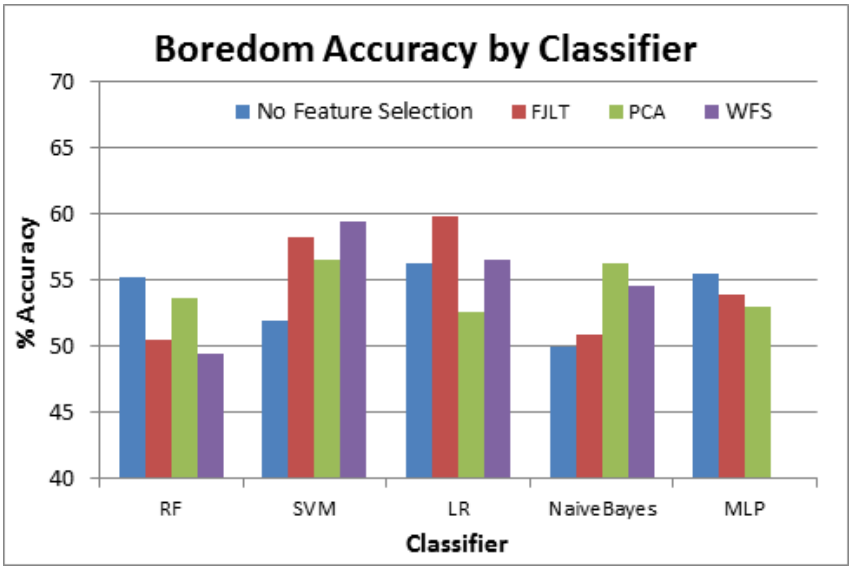

Figure 1: The accuracy in predicting boredom for each technique and each classifier

For boredom, we found no significant effects of classifier or of technique. There was no significant difference between WFS, the technique with the highest accuracy (M = 54.99%, SD = 11.75, κ = .09) and no feature selection, which had the lowest accuracy (M = 53.77%, SD = 10.53, κ = .07). We use Tukey's HSD test to compare the accuracies obtained by each method and for each classifier to the majority-class baseline, and show those scores that are significantly better than the baseline as bold entries in Table 1. Note that all feature selection methods can offer performance exceeding the baseline. It is also worthwhile to note that the highest accuracy (M = 59.84%, κ = .20) was actually produced by FJLT with Logistic Regression. This is extremely interesting given the fact that FJLT is a much more lightweight method than either PCA or WFS, and FJLT uses no knowledge of the dataset to construct its random projection.

Table 1: Boredom Accuracy

Feature Selection	RF	SVM	LR	NaiveBayes	MLP
None	55.26	51.90	**56.24**	50.00	**55.48**
FJLT	50.53	**58.29**	**59.84**	50.93	53.93
PCA	53.71	**56.51**	52.61	**56.29**	53.06
WFS	49.42	**59.42**	**56.56**	54.53	

For curiosity, however, there was a significant effect of both technique, $F(3, 171) = 7.314$, $\eta^2 = .114$, $p < .001$, and classifier, $F(4, 171) = 3.574$, $\eta^2 = .077$, $p < .05$. Tukey post-hoc analysis revealed that WFS (M = 63.26%, SD = 10.52, κ = .18) was significantly better than every other technique *except* FJLT (M = 58.17%, SD = 11.54, κ = .17). However, FJLT did not significantly outperform PCA or no feature selection. It is no surprise that WFS displayed the best performance, given the additional information it makes use of to select the best features. It is surprising, however, that FJLT can offer comparable performance with no such information, in a fraction of the time. Incidentally, the only significant difference between the classifiers was between SVM (M = 60.95%, SD = 10.20, κ = .15) and Naïve Bayes (M = 51.93% SD = 13.22, κ = .08). Note that because the dataset for curiosity is sharply skewed, the majority class baseline is 59.8%. Only two tests significantly exceeded this baseline, WFS with LR (M = 65.85%, κ = .25), and FJLT with SVM (M = 64.68%, κ = .22).

It should be noted, however, that the WFS process usually selects a much smaller number of features (between 5 and 30), while PCA tended to consistently extract 66 components, and the number of

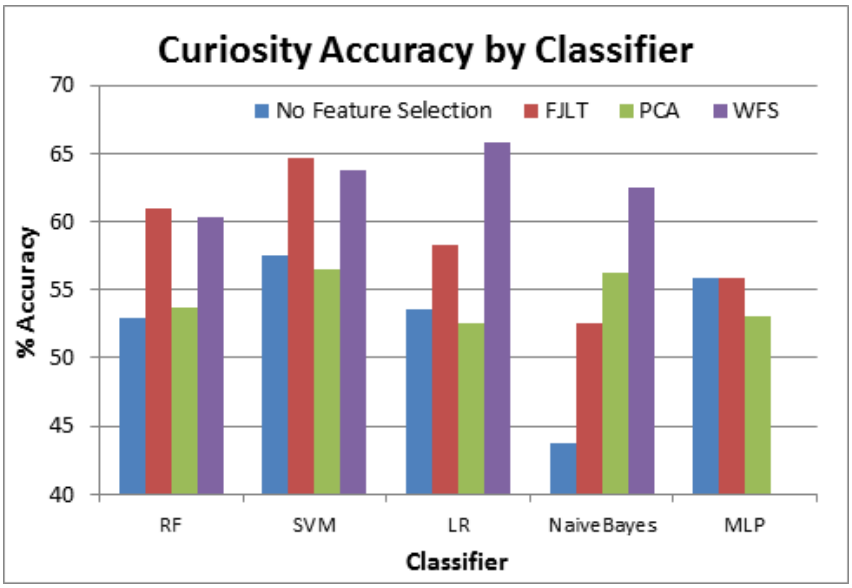

Figure 2: The accuracy in predicting curiosity for each technique and each classifier

dimensions the FJLT produced was fixed at 85. Therefore, as with previous research on this topic [8] [23], although the classification performance of PCA and FJLT did not differ significantly, FJLT required more features to achieve this performance.

5 Experiment 2 - High Dimensionality Text Classification

5.1 Data

Because the FJLT appears to be most effective for reducing the dimensionality of datasets with thousands of features, we decided to investigate performing text classification on extremely high-dimensional data. We obtained a dataset consisting of the text contents of 7528 articles, labelled with one of 20 newsgroups to which the articles belong [12]. The articles are typical postings and include headers, subject lines, and quoted portions of other articles. The classification labels include a range of 20 topics, from "alt.atheism" to "sci.space". As part of preprocessing the data was stemmed, meaning that various conjugations and tenses of the same word are reduced to the root representation (e.g."running" would be converted to "run"). Stop words from the SMART stop list [42] were removed, as well as any additional words shorter than three characters. We used a python library of topic modelling tools[3] to compile a dictionary of 43,720 unique words contained within the corpus, filter words that occurred in either a very small or very large proportion of the documents, and keep the remaining 10,000 most popular words. Then, we computed the *term frequency-inverse document frequency* (TF-IDF) [43] representation of each word, which increases with the frequency of the word in the document, and the rarity of the word in the corpus. Our final experiments were conducted on a 7528×10000 dataset of TF-IDF statistics.

We set the ε parameter of FJLT to 0.1, which in effect projects the original 10,000 dimensions down to 893. Because it is difficult to directly compare the performance of PCA and FJLT when they result in a different number of dimensions in the reduced space, we constrain PCA to always produce 893 components. In this way we can make a more direct comparison of the classification performance of the two techniques.

[3] http://radimrehurek.com/gensim/

5.2 Results

Figure 3 gives the results of performing classification on the 20 newsgroups text dataset. We do not include the results of using LR or MLP on the raw 10,000 features, because these algorithms are extremely computationally expensive in the number of features, and so such a test would not be practical. The memory requirements of LR also make it an unreasonable choice for this task. Only by performing a feature reduction technique such as FJLT do we render these classifiers viable.

This is fortunate, because MLP offers performance that actually exceeds the best results obtained by Ana Margarita, who published the version of the dataset we are using [12]. Margarita's best results were 82.8% with SVM, and 81.0% with Naïve Bayes. Our MLP classifier was able to surpass this, achieving 84.0% (κ = .831) with PCA. Our implementation of SVM actually achieved even better results, reaching a peak of 88.4% (κ = .878) with no feature selection. This could be the result of implementation details, or filtering the original 43,720 dimensions to 10,000 by removing excessively common or rare words. Our Naïve Bayes classifier did not outperform the Naïve Bayes results in [12], although it did reach a height of 75.6% (κ = .746) under the JL embedding.

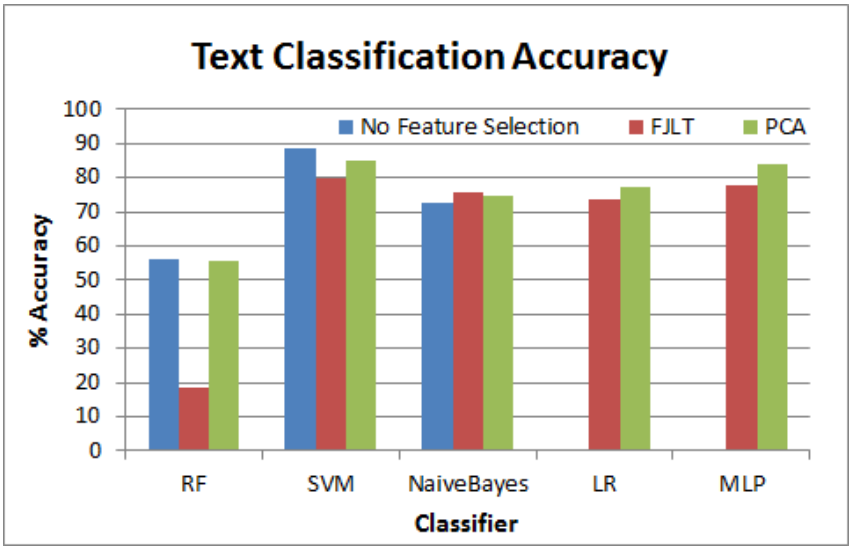

Figure 3: The text classification accuracy obtained by each classifier using the different feature reduction methods

Because there are 20 classification labels for this task that are approximately equally distributed, the baseline is approximately 5%, and we can see that all classifiers significantly exceed it. For this test we are more interested in how the performance of the different feature selection methods differs within each classifier. Using Tukey's HSD procedure, we find that, unsurprisingly, the performance of RF under FJLT is significantly worse than with either no feature selection or PCA $q(6,3) = 4.97, p < .05$. The same trend occurs for the SVM classifier; FJLT is significantly worse than the other two methods, $q(6,3) = 3.97, p < .05$, which do not differ significantly from each other. However it is important to note that even under FJLT, the SVM classifier achieved an accuracy of 80.0% (κ = .789), which is competitive with the best previously published results using SVM [12]. For Naïve Bayes, we find that FJLT significantly outperforms no feature selection, $q(6,3) = 7.06, p < .05$, although it is not significantly different from PCA. Finally, we find no significant differences between PCA and FJLT for LR or MLP.

6 Discussion and Limitations

We can see that the JL embedding affects the performance of the classifiers differently, and that these effects may not be consistent across datasets. For example, in two out of the three tests, FJLT significantly impaired the classification performance of Random Forests, but actually improved performance in classifying curiosity. In general it is easy to see that FJLT provides no good guarantees

for the RF classifier, which does not build classification rules based on Euclidean distance. Rather, it benefits when the trees it builds using random subsets of features are both strongly discriminative and unrelated (they do not share many features). When the data are transformed using FJLT, there is no guarantee that the features produced will be discriminative, only that there will be fewer of them. Since there are simply fewer features, it is more likely that each tree will be related, reducing the strength of the forest as a whole.

For the eye gaze dataset, FJLT improved the performance of the SVM and LR classifiers, while for the text dataset it was slightly lower for SVM. Because the eye gaze dataset has so few data points, it is extremely prone to overfitting [11]. Therefore the generalization error of classifiers trained on this dataset is more likely to benefit from simply reducing the number of features. Since the SVM and LR classifiers calculate discriminative functions using dot products, which are approximately preserved under the JL embedding, we should see that the performance of these classifiers remains high, within a small margin of error. This appears to be the case for the text classification results.

The results of the Naïve Bayes classifier consistently improve in response to reducing the number of features. This may be due to the assumption that gives Naïve Bayes its name: that the features are conditionally independent given the classification label. For the eye gaze data, it is quite likely that this assumption is violated by correlations which exist between the features, so feature reduction methods that remove these correlations result in steadily increasing performance. It is also possible that the dictionary words involved in text classification are correlated, so removing these may have led to the slight increase in performance in Naïve Bayes for both PCA and FJLT.

The performance of the MLP classifier did not differ significantly under PCA or FJLT for any of the tests, but in the eye gaze test we can see that both methods led to slightly worse results than including all of the original features. Although the complex nature of MLP makes proving theoretical results about it difficult [32], in practice it is often used to deal with high-dimensional data [38], and so it may not benefit from feature reduction in the same way as other classifiers.

Note that there are some inconsistencies in the way FJLT affects the performance of the classifiers between tests. This is to be expected; the generalizability of the results is limited by the number of datasets we can test. Further, in the original paper that describes FJLT [1], the authors repeatedly compute the FJLT projection, run the nearest neighbours classification algorithm for each projection, and take the nearest of all possible outputs. Whereas for this paper, we compute the FJLT only once for each dataset, so the quality of the random projection may not be consistent.

7 Conclusions

Although the FJLT did not improve the results of every classifier on every dataset, for most tests it was statistically no worse than PCA. This is quite remarkable, given the fact that FJLT needs no prior knowledge of the data to compute the embedding, while PCA looks for existing structure within the data. Further, the time complexity of PCA is $O(nd^2)$, while FJLT is $O(\frac{dlogn}{\varepsilon^2})$. Compared to PCA, and other excessively computationally expensive methods like WFS, FJLT offers good classification performance in a fraction of the time. In every test, FJLT was able to provide performance that significantly exceeded the majority-class baseline. In fact, in some cases FJLT actually led to the best performance out of any method for a single classifier, or even for the boredom test as a whole. Taken together, these results suggest that FJLT is a viable and efficient feature reduction technique, worth investigating for classification problems.

References

[1] Nir Ailon and Bernard Chazelle. The fast johnson-lindenstrauss transform and approximate nearest neighbors. *SIAM Journal on Computing*, 39(1):302–322, 2009.

[2] William B Johnson and Joram Lindenstrauss. Extensions of lipschitz mappings into a hilbert space. *Contemporary mathematics*, 26(189-206):1, 1984.

[3] Daniel M Kane and Jelani Nelson. Sparser johnson-lindenstrauss transforms. In *Proceedings of the Twenty-Third Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 1195–1206. SIAM, 2012.

[4] Nir Ailon and Edo Liberty. An almost optimal unrestricted fast johnson-lindenstrauss transform. *ACM Transactions on Algorithms (TALG)*, 9(3):21, 2013.

[5] Anirban Dasgupta, Ravi Kumar, and Tamás Sarlós. A sparse johnson: Lindenstrauss transform. In *Proceedings of the 42nd ACM Symposium on Theory of Computing*, pages 341–350. ACM, 2010.

[6] Jiří Matoušek. On variants of the johnson–lindenstrauss lemma. *Random Structures & Algorithms*, 33(2):142–156, 2008.

[7] Piotr Indyk. Algorithmic applications of low-distortion geometric embeddings. In *focs*, volume 1, pages 10–33, 2001.

[8] Sampath Deegalla and Henrik Bostrom. Reducing high-dimensional data by principal component analysis vs. random projection for nearest neighbor classification. In *Machine Learning and Applications, 2006. ICMLA'06. 5th International Conference on*, pages 245–250. IEEE, 2006.

[9] Alberto Bertoni and Giorgio Valentini. Random projections for assessing gene expression cluster stability. In *Neural Networks, 2005. IJCNN'05. Proceedings. 2005 IEEE International Joint Conference on*, volume 1, pages 149–154. IEEE, 2005.

[10] David L Donoho et al. High-dimensional data analysis: The curses and blessings of dimensionality. *AMS Math Challenges Lecture*, pages 1–32, 2000.

[11] Natasha Jaques, Cristina Conati, Jason Harley, and Roger Azevedo. Predicting affect from gaze data during interaction with an intelligent tutoring system. In *Intelligent Tutoring Systems*. (to appear), June 2014.

[12] Ana Margarida de Jesus Cardoso Cachopo. *Improving Methods for Single-label Text Categorization*. PhD thesis, Universidade Técnica de Lisboa, 2007.

[13] Dimitris Achlioptas. Database-friendly random projections: Johnson-lindenstrauss with binary coins. *Journal of computer and System Sciences*, 66(4):671–687, 2003.

[14] Rosa I Arriaga and Santosh Vempala. An algorithmic theory of learning: Robust concepts and random projection. In *Foundations of Computer Science, 1999. 40th Annual Symposium on*, pages 616–623. IEEE, 1999.

[15] Piotr Indyk and Rajeev Motwani. Approximate nearest neighbors: towards removing the curse of dimensionality. In *Proceedings of the thirtieth annual ACM symposium on Theory of computing*, pages 604–613. ACM, 1998.

[16] Vladimir Braverman, Rafail Ostrovsky, and Yuval Rabani. Rademacher chaos, random eulerian graphs and the sparse johnson-lindenstrauss transform. *arXiv preprint arXiv:1011.2590*, 2010.

[17] TS Jayram and David P Woodruff. Optimal bounds for johnson-lindenstrauss transforms and streaming problems with subconstant error. *ACM Transactions on Algorithms (TALG)*, 9(3):26, 2013.

[18] Jelani Nelson and Huy L NguyÅn. Sparsity lower bounds for dimensionality reducing maps. In *Proceedings of the 45th annual ACM symposium on Symposium on theory of computing*, pages 101–110. ACM, 2013.

[19] Nir Ailon and Bernard Chazelle. Faster dimension reduction. *Communications of the ACM*, 53(2):97–104, 2010.

[20] Leonard J Schulman. Clustering for edge-cost minimization. In *Proceedings of the thirty-second annual ACM symposium on Theory of computing*, pages 547–555. ACM, 2000.

[21] Piotr Indyk. Stable distributions, pseudorandom generators, embeddings and data stream computation. In *Foundations of Computer Science, 2000. Proceedings. 41st Annual Symposium on*, pages 189–197. IEEE, 2000.

[22] Christos H Papadimitriou, Hisao Tamaki, Prabhakar Raghavan, and Santosh Vempala. Latent semantic indexing: A probabilistic analysis. In *Proceedings of the seventeenth ACM SIGACT-SIGMOD-SIGART symposium on Principles of database systems*, pages 159–168. ACM, 1998.

[23] Xudong Lv and J Wang. Fast johnson-lindenstrauss transform for robust and secure image hashing. In *Multimedia Signal Processing, 2008 IEEE 10th Workshop on*, pages 725–729. IEEE, 2008.

[24] Avrim Blum. Random projection, margins, kernels, and feature-selection. In *Subspace, Latent Structure and Feature Selection*, pages 52–68. Springer, 2006.

[25] Sampath Deegalla and Henrik Boström. Classification of microarrays with knn: Comparison of dimensionality reduction methods. In *Intelligent Data Engineering and Automated Learning-IDEAL 2007*, pages 800–809. Springer, 2007.

[26] James M Wood. Understanding and computing cohen's kappa: A tutorial. *WebPsychEmpiricist. Web Journal at http://wpe. info/*, 2007.

[27] Joseph L Fleiss. Measuring nominal scale agreement among many raters. *Psychological bulletin*, 76(5):378, 1971.

[28] Andy Field. *Discovering statistics using SPSS*. Sage publications, 2009.

[29] Mark Hall, Eibe Frank, Geoffrey Holmes, Bernhard Pfahringer, Peter Reutemann, and Ian H Witten. The weka data mining software: an update. *ACM SIGKDD explorations newsletter*, 11(1):10–18, 2009.

[30] Alok Sharma and Kuldip K Paliwal. Fast principal component analysis using fixed-point algorithm. *Pattern Recognition Letters*, 28(10):1151–1155, 2007.

[31] Isabelle Guyon and André Elisseeff. An introduction to variable and feature selection. *The Journal of Machine Learning Research*, 3:1157–1182, 2003.

[32] Marti A. Hearst, ST Dumais, E Osman, John Platt, and Bernhard Scholkopf. Support vector machines. *Intelligent Systems and their Applications, IEEE*, 13(4):18–28, 1998.

[33] John C Platt. Fast training of support vector machines using sequential minimal optimization. 1999.

[34] Leo Breiman. Random forests. *Machine learning*, 45(1):5–32, 2001.

[35] George H John and Pat Langley. Estimating continuous distributions in bayesian classifiers. In *Proceedings of the Eleventh conference on Uncertainty in artificial intelligence*, pages 338–345. Morgan Kaufmann Publishers Inc., 1995.

[36] John T Pohlman and Dennis W Leitner. A comparison of ordinary least squares and logistic regression. 2003.

[37] Saskia Le Cessie and JC Van Houwelingen. Ridge estimators in logistic regression. *Applied statistics*, pages 191–201, 1992.

[38] Geoffrey E Hinton, Nitish Srivastava, Alex Krizhevsky, Ilya Sutskever, and Ruslan R Salakhutdinov. Improving neural networks by preventing co-adaptation of feature detectors. *arXiv preprint arXiv:1207.0580*, 2012.

[39] Ian H Witten and Eibe Frank. *Data Mining: Practical machine learning tools and techniques*. Morgan Kaufmann, 2005.

[40] G Sahoo et al. Analysis of parametric & non parametric classifiers for classification technique using weka. *International Journal of Information Technology and Computer Science (IJITCS)*, 4(7):43, 2012.

[41] Jason M Harley, François Bouchet, and Roger Azevedo. Aligning and comparing data on emotions experienced during learning with metatutor. In *Artificial Intelligence in Education*, pages 61–70. Springer, 2013.

[42] Gerard Salton. The smart retrieval system—experiments in automatic document processing. 1971.

[43] Gerard Salton and Michael J McGill. Introduction to modern information retrieval. 1983.

A FJLT code

```
import numpy as np
import math
import random

C = 1 #constant used in computing k
```

```
def constructP(epsilon, p, k, d, n):
#k x d
  term = epsilon**(p-2)*(math.log(n))**p / d
  q = min(term,1)

  P = np.zeros(shape=(k,d))
  for i in range(k):
    for j in range(d):
      r = random.uniform(0.0,1.0)
      if r <= q:
        P[i,j] = np.random.normal(0, 1.0/q)

  return P

def constructH(d):
#d x d
  H = np.matrix([[1,1],[1,-1]]) # base Hadamard matrix

  #compute Hadamard matrix of size dxd
  for i in range(int(math.log(d,2))-1):
    H1 = np.concatenate((H,H))
    H2 = np.concatenate((H, H * -1))
    H = np.concatenate((H1,H2), axis=1)

  H = H * d**(-0.5)
  return H

def constructD(d):
#d x d
  D = np.zeros(shape=(d,d))

  for i in range(d):
    r = random.uniform(0.0,1.0)
    if r <= 0.5:
      D[i,i] = 1.0
    else:
      D[i,i] = -1.0
  return D

def findClosestPowerOf2(d):
  i = 1
  while(1):
    if 2**i >= d:
      return 2**i
    i = i + 1

def FJLT(X, epsilon):
  # ref: http://www.cs.princeton.edu/~chazelle/pubs/fasterdim-ac10.pdf
  # X = input data - set of vectors in Euclidean space
  # n = size of set X
  # d = dimension of Euclidean space
  # k = dimension we will reduce to
  # epsilon = error tolerance parameter
  # p = the norm we are using (if Euclidean, p = 2)

  random.seed()

  p = 2

  n = X.shape[0]

  #pad data to a power of 2
  d_orig = X.shape[1]
  d = findClosestPowerOf2(d_orig)
  X_F = np.zeros(shape=(n,d))
```

```
    X_F[:,0:d_orig] = X[:,:]

    k = int(C * math.log(n) / epsilon**2)

    print n, "x", d_orig
    print "k=", k
    print "d=", d

    P = constructP(epsilon, p, k, d, n)
    H = constructH(d)
    D = constructD(d)

    F = P * H * D

    result = X_F * F.transpose()
    result = result / math.sqrt(float(k)/d)

    return result
```