Dimensionality reduction for bag-of-words models: PCA vs LSA

Benjamin Fayyazuddin Ljungberg
benfl@stanford.edu

Abstract

We study a collection of texts stored as "bags of words" and implement two methods for reducing the dimension of the data. We compare how easy it is to perform authorship identification on the dimensionally-reduced data.

1 Introduction

In Natural Language Processing (NLP), texts are often represented as a "bag-of-words." That is, they are reduced to the data of which words occur in the text and with what frequency they occur. When represented this way, a text will typically be a very sparsely populated vector living in a very high-dimensional space. In order to make the data amenable to standard machine learning techniques, it is often desirable to reduce the dimension of the feature space while retaining as much information as possible. A standard method for feature space reduction is Principal Component Analysis, which roughly speaking, projects a set of points onto a smaller dimensional affine subspace of "best fit." In NLP, a similar technique known as Latent Semantic Analysis (LSA) or Latent Semantic Indexing, is frequently used. From a geometric perspective, this amounts to projecting onto a linear subspace instead of an affine subspace. From a computational perspective, both PCA and LSA are about computing singular value decompositions (SVDs) of large matrices. It is natural to want to compare the utility of these two methods.

The application in mind here is authorship classification. Given a set of texts one would like to classify who wrote which text. Ideally, the end goal would be *unsupervised* author classification: given unlabeled texts, one would like to cluster the texts by author. In order to achieve this long-term goal, it would be necessary to not only reduce the feature space, but also pick out features that are indicative of authorship (as opposed to topic, say).

For this paper, I ran an experiment on a collection of texts that was easily classifiable using the bag-of-words model. I performed dimension reduction using both LSA and PCA, and compared the performance of a basic one-vs-rest support vector machine (SVM) using different numbers of features from the two methods. The result was that PCA performed much better than LSA for this problem.

1.1 Previous work

The use of singular value decompositions to simplify the feature space in NLP began with the work Deerwester et al. (1990). In Li et al. (2006), genetic algorithms are used to reduce the number of features in texts for the purposes of unsupervised learning. As an alternative route than ours, Diederich et al. (2003) discuss many features that are not tied to the specific words used in the text, and how to select those for authorship attribution.

As far as algorithms go, the most important work that we will be building on in this paper are the methods in Baglama and Reichel (2005) and Martinsson et al. (2011) for computing singular value decompositions. These are essential in quickly performing latent semantic analysis and principal component analysis.

2 Theoretical background

2.1 Matrix decompositions and dimensionality reduction

Given an $m \times n$ matrix A with n large, it is often desirable to project the rows onto a smaller-dimensional space, giving a matrix of shape $m \times k$ with $k \ll n$. For now, take $k = 1$, so that we project the rows onto a line. If the rows of A

represent samples from some data, we would like this projection to keep the variance of the samples as large as possible, because this corresponds to losing as little information as possible. A standard calculation shows that we should take the line to be spanned by a right eigenvector of A^TA with maximal eigenvalue (A^TA is positive semidefinite, so its eigenvalues are all positive). In general, if we want k features, it is optimal to take the k eigenvectors of A^TA with maximal eigenvalues. These eigenvectors are known as the right singular vectors of A. Finding the singular vectors along with their eigenvalues is essentially the process known as *singular value decomposition* (SVD).

In LSA, we naively apply SVD to reduce the dimension of the feature space (Manning et al. (2008) ch. 18). In PCA, we first perform some normalizations: we scale the columns to have variance 1 and translate them to have mean 0 before applying SVD. Geometrically, this amounts to normalizing the features to the same scale, and projecting the data not onto a linear subspace but onto an affine subspace (the "origin" of this affine subspace is the point in $\mathbb{R}^n$ given by the means of the columns of A). We clearly expect PCA to perform better, since the collection of linear subspaces is contained in the collection of affine subspaces, but it is not clear if the difference will be noticeable. For example, if the features nearly have mean 0 and variance 1 to begin with, the normalization shouldn't affect much.

For the purposes of natural language processing, the matrix A is often sparse, but after subtracting column means it will no longer have this property. For this reason, it may appear that LSA will be much more computationally tractable. In practice however, many algorithms for calculating SVDs do not perform entrywise calculations on matrices; they only need to be able to compute matrix-vector products $(A^TA)v$ quickly (see e.g. Baglama and Reichel, 2005 and Martinsson et al., 2011). This can be done efficiently if we remember the original sparse matrix as well as the vector of column means.

2.2 F1 scores

The data involved in this project was quite unbalanced. To account for this, it can be useful to consider other error metrics than just accuracy. For a binary classification problem, we can use the F1 score. Let the precision p be the number of labels correctly predicted to be positive by the model divided by the number of labels predicted to be positive in total (whether correct or not). Let the recall r be the number of labels correctly predicted to be positive divided by the total number of labels that actually are positive. Then the F1 score is given by

$$F_1 = \frac{2pr}{p+r}.$$

So the $F1$ score is zero precisely when *no* labels are correctly predicted to be positive. It is 1 precisely when both p and r are equal to 1, so that everything is predicted correctly.

In the multi-class setting, if we want to take into account imbalances in the data, we can take the mean of the F1 scores for each class. Another possibility is to take the minimum of the F1 scores over all classes.

3 Data

I collected 8500 text fragments from novels by 13 different authors, labeled by author, and split them into a training set and test set. The split was done at random by author, so that for each author, a random selection of 10% of the texts was chosen to be in the test set. The texts were downloaded from Project Gutenberg (`http://gutenberg.org/`) using the Gutenberg package for Python (`https://pypi.python.org/pypi/Gutenberg`). The data was quite unbalanced: some authors only had 100 texts representing them, while one author has 1200 texts. This was done on purpose, because with unsupervised author classification it would not be possible to control the balancedness of the data.

3.1 Preprocessing

I used the Snowball stemming algorithm built into Python's Natural Language Toolkit (Bird et al., 2009) to reduce each word to its stem. A representative vocabulary of 4731 words was selected from these stems, consisting of those stems occurring at least 100 times and at most 1000 times in all the texts. The training and test sets were then represented as term-frequency matrices of size 7650×4731 and 850×4731 respectively (so the i, j entry contains the number of times vocabulary entry j occurs in text i).

3.2 Feature space reduction

I used the IRLBA package for R (Baglama et al., 2017) to extract the first 90 principal components of the train set and test set taken together and to find the projection of the data onto the lower-dimensional space. This was convenient because the package allows the user to automatically subtract a vector from each row of the matrix, optimizing for the fact that the

original matrix was sparse. For LSA, I used the `decomposition.TruncatedSVD` class from the scikit-learn package for Python (Pedregosa et al., 2011) to project the data onto a 200-dimensional space. The discrepancy of only finding 90 features for PCA while we find 200 for LSA stems from the fact that `TruncatedSVD` uses a faster and more stable algorithm (but it does not have a built-in way to normalize the colums).

4 Experiment

4.1 Methods

4.1.1 Naive Bayes

I trained a Naive Bayes classifier using the multinomial event model, and with Laplace smoothing on the training set and made predictions on the test set. This means that we make the hypothesis that texts are chosen as follows. First the author a is picked with some probability ϕ_a (so that $\sum_a \phi_a = 1$). Then (with the author fixed) the text is generated one word at a time: each word is generated independently, and it has some probability $\phi_{a,v}$ of being vocabulary entry v. This forces, for a fixed, $\sum_{v \in V} \phi_{a,v} = 1$. We estimate the probabilities as

$$\phi_a = \frac{\#\text{texts by author } a}{\text{total number of texts}} \qquad \phi_{a,v} = \frac{1 + \#\text{times } v \text{ occurs in a text by } a}{|V| + \#\text{words in all texts by } a}$$

(essentially, for each author, we "pretend" that we've seen each vocabulary entry one more time than we actually have in order to make the probabilities nonzero; this is Laplace smoothing).

Given a text t in which each vocabulary entry v occurs n_v times, we can compute the probability that a given author was selected given that this text was produced (using Bayes's rule):

$$P(\text{author } a \mid \text{text } t) = \frac{\phi_a P(\text{text } t \mid \text{author } a)}{P(v_i \text{ occurs } n_i \text{ times})} = \frac{\phi_a \prod_{v \in V} \phi_{a,v}^{n_v}}{P(\text{text } t)}.$$

The author with maximal probability is the prediction of the model. The denominator is the same regardless of author, so we simply find the author that maximizes the numerator.

I used the `MultinomialNB` class built in to scikit-learn (Pedregosa et al., 2011). The accuracy was 99.53%; on a test set of 850 items, this means that only four were misclassified. This indicates that our initial selection of vocabulary (as well as our choice of the bag-of-words model) is good enough to mostly distinguish the texts. This high performance was the reason why I didn't delve further into the task of making accurate predictions, and instead decided to see how much information I could lose while still making good predictions.

4.1.2 Support vector machines

Support vector machines are usually used for binary classification problems, so some modification is needed for this multi-class classification problem. I used a one-vs-rest strategy: for each author a, an SVM was trained to distinguish classes labeled a from classes not labeled a. Given a test point, for each author, we compute the signed distance from the hyperplane given by that author's SVM (with the positive direction being the direction that positively identifies the author). The predicted label is the author giving the maximum distance.

For the SVM for author a, the training points are labeled x_i and their labels are y_i with $y_i = +1$ if a wrote that text and $y_i = -1$ if a did not write that text. The SVM uses squared hinge loss. That is, we fit a hyperplane $\{x \mid w^T x = b\}$ that minimizes the quantity

$$\frac{1}{2}\|w\|^2 + \sum_i \max(0, 1 - y_i(w \cdot x_i - b))^2.$$

All of this was built-in to the `LinearSVC` class in scikit-learn (Pedregosa et al., 2011).

4.2 Baselines

In contrast to many machine learning tasks, here we need baselines to determine what optimal performance would look like. It would be very surprising (although not inconceivable) if classification accuracies improved after reducing dimensionality. I performed two standard classification algorithms on the original sparse matrices before feature reduction: the naive Bayes classifier and support vector machines.

4.3 Tests

For the dimensionally reduced data, it no longer makes sense to use a naive Bayes classifier (for one thing, the features are no longer valued in positive integers). However, we can still use support vector machines for classification.

To test PCA, I trained and tested the support vector machine model using the first n principal components for $n = 10, 11, \ldots, 90$. For LSA, I did the same thing except for $n = 10, 11, \ldots, 200$. I recorded the accuracy each time.

5 Results

For each of the error metrics {accuracy, mean F1 score, minimum F1 score}, I plotted the performance for SVMs using the two methods of dimension reduction. The red horizontal lines are the baseline accuracies. The green vertical lines are always at $x = 50$ while the black vertical lines are at $x = 90$ (which is where the x-axis ends for the models using PCA). These are only there to facilitate comparisons between the graphs. The results can be found in Figure 1. Some representative values from the graphs can be found in Table 1.

Table 1: Some of the results from the experiment

	Accuracy	Mean F1 score	Minimum F1 score
Baseline	0.9847	0.9832	0.9524
PCA 50 features	0.9788	0.9512	0.7429
LSA 50 features	0.8494	0.6855	0.0000
PCA 90 features	0.9800	0.9675	0.8571
LSA 90 features	0.9188	0.8009	0.3226
LSA 200 features	0.9741	0.9524	0.8000

We see that the performance of LSA using 200 features is comparable to PCA using just 50 features, and generally that PCA requires far fewer dimensions than LSA to achieve similar results.

5.1 Analysis

5.1.1 Discussion of results

It is vital not to overinterpret these results. It is clear that for this particular classification problem, and with this particular model, the features produced by PCA work much better than those produced by LSA. This is a good indication that PCA tends to be a much more efficient way to reduce dimensionality, but it is far from conclusive.

Experimentally, it would make sense to try the same methods on different sets of texts, and not just the particular texts used here. For whatever reason, it may be the case that these particular texts are particularly ill-suited to latent semantic analysis. Theoretically, it would be very interesting to find some quantitative prediction of just *how much* better PCA should work than LSA.

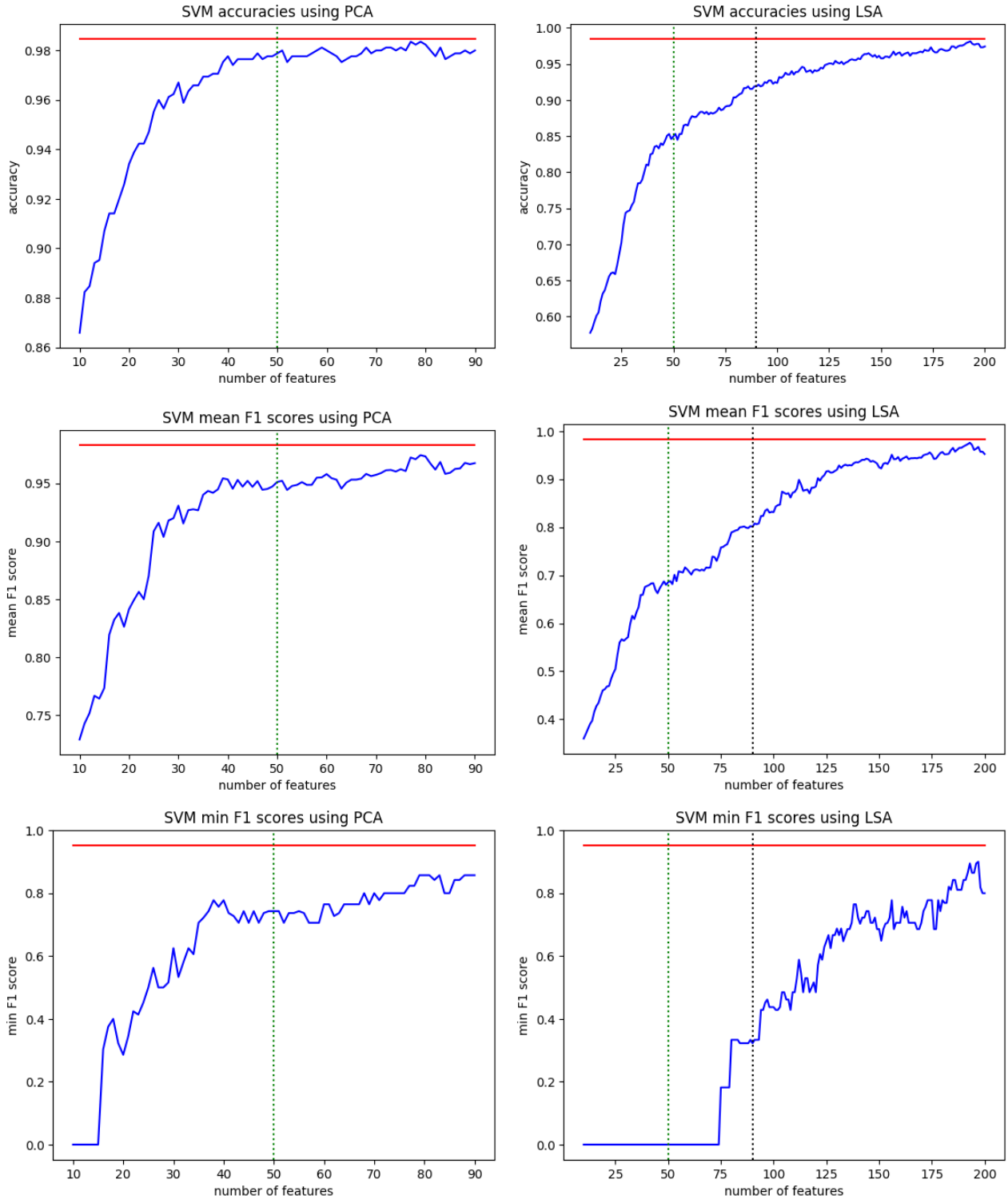
5.1.2 Future directions

As mentioned in the introduction, the original motivation for this dimension-reduction process was to perform unsupervised author clustering. After a few very basic tests, it seems that it is still hopeless to try to cluster these texts by author. This is unsurprising: the dimensionally-reduced features we have found do capture authorship very well, but they presumably also capture other aspects of the texts, such as the setting of the plot. Ideally, one would like to be able to separate the features into ones that are indicative of authorship and ones that are indicative of text topic. Gatys et al. (2015) are able to do something analogous for images, separating content from style, so it might be possible to use neural networks for this NLP task.

One obvious practical thing to do would be to implement the randomized SVD algorithm of Martinsson et al. (2011) so that it performs well after normalizing sparse matrices so that the column means are 0. We saw that this algorithm was able to produce many more singular vectors than the one of Baglama and Reichel (2005).

Finally, one might ask if there are theoretical reasons why we might expect the data to lie on an affine subspace of a large vector space. If not, there might be better approximations than PCA: one might be able to use some submanifold of the high-dimensional feature space that is easy to parametrize. It is not clear what kind of shape one would expect for this manifold.

Figure 1: Results vs numbers of features



Acknowledgments

I appreciate the work of the teaching staff for the Fall 2017 version of CS 229. All of the machine learning I know comes from them.

References

James Baglama and Lothar Reichel. Augmented implicitly restarted Lanczos bidiagonalization methods. *SIAM Journal on Scientific Computing*, 27(1):19–42, 2005.

Jim Baglama, Lothar Reichel, and B. W. Lewis. *irlba: Fast Truncated Singular Value Decomposition and Principal Components Analysis for Large Dense and Sparse Matrices*, 2017. URL `https://CRAN.R-project.org/package=irlba`. R package version 2.3.1.

Steven Bird, Ewan Klein, and Edward Loper. *Natural Language Processing with Python*. O'Reilly Media, 2009.

Scott Deerwester, Susan T Dumais, George W Furnas, Thomas K Landauer, and Richard Harshman. Indexing by latent semantic analysis. *Journal of the American society for information science*, 41(6):391, 1990.

Joachim Diederich, Jörg Kindermann, Edda Leopold, and Gerhard Paass. Authorship attribution with support vector machines. *Applied intelligence*, 19(1):109–123, 2003.

Leon A Gatys, Alexander S Ecker, and Matthias Bethge. A neural algorithm of artistic style. *arXiv preprint arXiv:1508.06576*, 2015.

Jiexun Li, Rong Zheng, and Hsinchun Chen. From fingerprint to writeprint. *Communications of the ACM*, 49(4):76–82, 2006.

Christopher D. Manning, Prabhakar Raghavan, and Hinrich Schütze. *Introduction to Information Retrieval*. Cambridge University Press, 2008.

Per-Gunnar Martinsson, Vladimir Rokhlin, and Mark Tygert. A randomized algorithm for the decomposition of matrices. *Applied and Computational Harmonic Analysis*, 30(1):47–68, 2011.

F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.