Annals of Data Science (2023) 10(1):25–41
https://doi.org/10.1007/s40745-020-00277-x

Sparse Principal Component Analysis for Natural Language Processing

Reza Drikvandi[1] · **Olamide Lawal**[1]

Received: 11 March 2020 / Revised: 18 April 2020 / Accepted: 30 April 2020 / Published online: 18 May 2020

Abstract
High dimensional data are rapidly growing in many different disciplines, particularly in natural language processing. The analysis of natural language processing requires working with high dimensional matrices of word embeddings obtained from text data. Those matrices are often sparse in the sense that they contain many zero elements. Sparse principal component analysis is an advanced mathematical tool for the analysis of high dimensional data. In this paper, we study and apply the sparse principal component analysis for natural language processing, which can effectively handle large sparse matrices. We study several formulations for sparse principal component analysis, together with algorithms for implementing those formulations. Our work is motivated and illustrated by a real text dataset. We find that the sparse principal component analysis performs as good as the ordinary principal component analysis in terms of accuracy and precision, while it shows two major advantages: faster calculations and easier interpretation of the principal components. These advantages are very helpful especially in big data situations.

Keywords Classification · Dimensionality reduction · Ensemble learning · High dimensional data · Natural language processing · Sparse principal component analysis

1 Introduction

High dimensional data are rapidly growing in many different disciplines due to the development of technological advances [1]. High dimensional data are particularly common is natural language processing (NLP). The analysis of NLP requires working with high dimensional matrices of word embeddings created from text data. Those matrices are often sparse in the sense that they include many zero elements.

✉ Reza Drikvandi
r.drikvandi@mmu.ac.uk

[1] Department of Computing and Mathematics, Manchester Metropolitan University, Manchester, UK

Methods that can effectively handle such sparse matrices are therefore of great importance. In this paper, we study and apply sparse principal component analysis (SPCA) for natural language processing, which can effectively handle large sparse matrices. Unlike ordinary principal component (PCA) which has been extensively used for NLP, sparse PCA is not well investigated in this field.

Word embeddings resulting from NLP models are shown to be a great asset for a wide variety of NLP tasks [2]. However, such architecture is often time-consuming and difficult to train and implement when the word embeddings involve high dimensional matrices of size thousands or larger. It is therefore important to simplify and accelerate the computations of word embedding matrices to facilitate NLP analysis in high dimensions. A traditional approach is PCA, however a main limitation of the ordinary PCA is that the principal components are essentially linear combinations of all of the original variables (see Sect. 2 for details). In other words, all the weights in the linear combinations (known as loadings) are typically non-zero, hence resulting in a dense matrix. In this paper, we will show that the computations will be accelerated by applying sparse PCA which uses sparse principal components by setting the less important loadings to be zero. Furthermore, in many applications such as those explained in Sect. 2, the interpretation of principal components would be much easier if the principal components are sparse.

Dimensionality reduction techniques are frequently used for the analysis of high dimensional data from NLP. The curse of dimensionality reminds us the issues that emerge when working with data in higher dimensions which may not exist in lower dimensions (see, e.g., [1]). PCA and SPCA are two powerful tools for data analysis to carry out dimensionality reduction in large datasets. As mentioned above, the latter provides further dimensionality reduction by using sparse principal components, as detailed in Sect. 2.

As the number of features increases, mathematical models tend to become more difficult to work with. The more features we have, the larger sample we should have, in addition to all combinations of feature values to be representative for in the sample. It is well known that when the quantity of features expands, the models turn out to be progressively unpredictable (see Fig. 1, [3]). Also, a larger number of features could lead to higher chance of overfitting. A machine learning model that is trained on an enormous number of features would get progressively subject to the data it was trained on, and consequently overfitted, thereby defeating the purpose.

Feature selection is the process of recognising and choosing important features from the data. Feature engineering is manually producing new features from existing features, by making some changes or performing some manipulations on them. Feature selection is done either manually or automatically. For instance, suppose that we attempt to build a model that predicts individuals' weights and we have gathered a huge amount of data which depicts every individual completely. On the off chance that we had a column that depicted the colour of every individual's apparel, would that be much help in anticipating their weight? No one may think so. That is something we should remove right away. Should not something be said about a column that portrays their heights? This is a positive yes. We can make such straightforward manual feature selections and reduce the dimensionality when the significance or unimportance of specific features are

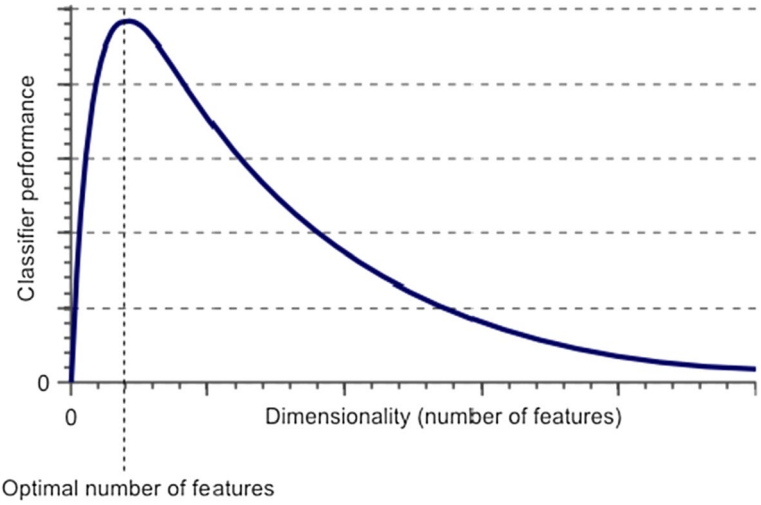

Fig. 1 The curse of dimensionality and classification performance [3]

evident or common knowledge. When that is not self-evident, there is a great deal of tools to carry out the feature selection (see, e.g., [4]).

In this paper, we focus on SPCA and consider an important application in natural language processing (NLP), which requires analysing high dimensional matrices, often of size thousands or larger. As mentioned above, PCA has already been used for text mining, however SPCA is not well understood for NLP. Our main contribution would be to study and apply SPCA for NLP, and understand and compare it with the ordinary PCA.

2 Formulations of PCA and Sparse PCA

In this section, we review the main formulations of PCA and sparse PCA. Ningmin and Jing [5] provide a comprehensive treatment of the formulations for PCA and its sparse variations. PCA basically aims to find direct mixes of the factors, which are linear combinations of the original variables and known as principal components, that lead to the directions of maximal discrepancy in the data. This process can be performed by applying the singular value decomposition (SVD) method to a data matrix A, or via an eigenvalue decomposition if A is a covariance matrix. The general idea is as follows.

Suppose that there are n iid data points $X_1, \ldots, X_n$ on p variables. Since p may be very large, we wish to use PCA for dimensionality reduction on p. Let $\Sigma = \mathbb{E}\left(XX^T\right)$ be the population covariance matrix with $\mathbb{E}(X) = 0$. The eigenvalue decomposition of Σ is as follows:

$$\Sigma = \lambda_1 v_1 v_1^T + \lambda_2 v_2 v_2^T + \cdots + \lambda_p v_p v_p^T,$$
$$\lambda_1 \geq \lambda_2 \geq \lambda_p \geq 0, \quad \text{(eigenvalues)},$$
$$v_i^T v_j = \delta_{ij}, \quad \text{(eigenvectors)},$$

with the "optimal" d-dimensional projection $X \to \Pi_d X$,

$$\Pi_d = V_d V_d^T, V_d = (v_1, v_2, \ldots, v_d).$$

The sample covariance matrix is defined as:

$$\hat{\Sigma} = n^{-1}\left(X_1 X_1^T + \cdots + X_n X_n^T\right).$$

We estimate $(\hat{\lambda}_j, \hat{v}_j)$ by the eigenvalue decomposition of $\hat{\Sigma}$ where

$$\hat{V}_d = \left(\hat{v}_1, \ldots, \hat{v}_d\right), \quad \hat{\Pi}_d = \hat{V}_d \hat{V}_d^T.$$

The importance of PCA is due to the use of a fewer number of components. Firstly, by catching bearings of most extreme difference in the information in the dataset, the important segments (i.e., main principal components) provide the opportunity to reduce and compress the data so minimum information is lost. Secondly, the important segments (main principal components) are uncorrelated, which can help understanding factual examination. Of course, PCA has a few well-reported disadvantages too. A specific limitation, which is our main motivation here, is that the principal components are essentially linear combinations of all of the original variables (see, e.g., [6]). In Other words, all the weights in the linear combination (known as loadings) are typically non-zero. However, in many applications, the coordinate axes have a physical interpretation; for example, in biology each axis may correspond to a specific gene. Also, in certain applications, for example in financial asset procedures based on principal components techniques, the sparsity of the loadings has significant outcomes, since the fewer non-zero loadings implies less fixed transaction costs. In such applications, understanding and interpreting the principal components would be much easier if the principal components are sparse and do not include many non-zero loadings. This can be achieved by SPCA as explained in the sequel.

2.1 Formulation of PCA

Suppose that the input data matrix is denoted by $X = [x_1, x_2, \ldots, x_n]^T \in R^{n \times d}$, where n is the size of the observed data, and further d represents the dimensionality of the data. Assume all the variables are centred, that is, $\Sigma_i x_i = 0$ and $\Sigma = \frac{1}{n} X^T X \in R^{d \times d}$ be the data covariance matrix. PCA seeks to find a number of $p \ll d$ linear combinations of the n variables in the projected linear space as $\tilde{z}_k = X^T u_k = \Sigma_{i=1}^d u_{k,i} x_i$, where $\tilde{z}_k$ is the k-th principal component (PC) and u_k is the corresponding unit-length loadings vector. As mentioned above, PCA can be performed by either an eigenvalue decomposition of the covariance matrix or by SVD. The formulation of PCA can be derived from the data-variance-maximisation viewpoint. The goal is to find u where

the input data variance is maximised. This leads to the following optimisation problem: (see also [5])

$$\max_{u} u^T \Sigma u \quad s.t. \quad ||u|| = 1. \tag{1}$$

2.2 Formulation of Sparse PCA

The main objective of sparse PCA is to force a number of less important loadings to be zero, resulting in sparse eigenvectors. In order to achieve such sparsity on the extracted components, most of the available methods find the PC's of the covariance matrix through adding a constraint or penalty term from the PCA formulation (1). A constrained l_0-norm minimisation problem is usually considered as the basic sparse PCA problem as follows: (see also [5])

$$u = \underset{u}{\operatorname{argmax}}\, u^T \Sigma u \quad s.t. \quad ||u||_2 = 1, \ ||u||_0 \leq k, \tag{2}$$

where k is the number of non-zero loadings. The SPCA problem in (2) is non-convex and NP-hard. All the formulations and algorithms can be categorised into three main classes: the data-variance-maximisation, minimal-reconstruction-error and probabilistic modelling viewpoints [5]. When moving from PCA to SPCA, there are two interrelated issues that should be examined: (1) the presence of relationship amongst scores and/or loadings in various components, and (2) the variation of components from the row-space characterised by the first data.

The primary point makes the customary method of computing scores, residuals and caught variance utilised in PCA not relevant in SPCA. The caught variance is regularly considered as a standard criterion for model quality and examination of model variations. In this manner, its exact calculation is essential. The subsequent point influences the interpretability, notably for multi-part models.

Most SPCA calculations would change the old style PCA by including sparse-inducing constraints or penalties with the L0 or L1 standards. Jolliffe et al. [7] built up the simplified component technique-LASSO known as the SCoTLASS algorithm, which uses the least total shrinkage and selection operator [8], where the L1 standard (absolute value) of the loadings is penalised. The SCoTLASS criterion is as follows:

$$\hat{P}^{SL} = arg \max_{p} ||Xp||_F^2 \quad s.t. \quad ||p||_1 \leq c, ||p||_2^2 = 1, \tag{3}$$

where $\hat{P}^{SL}$ is the resulting sparse loading with superscript SL alluding the subsequent sparse loading. To get progressive components, the SCoTLASS optimisation forces the second and further sparse loadings to be orthogonal to the rest. The SCoTLASS criterion (3) could be computationally demanding [9], which makes its utilisation eccentric for information investigation. Likewise, the number of non-zero components in $\hat{P}^{SL}$ is upper-limited by the quantity of observation in the data, which is a basic impediment of the lasso constraint.

3 Sparse PCA for NLP

Natural language processing is a field concerned with the ability of computer to understand, analyse, manipulate, and potentially generate human language. This could be any type of communication with and between humans. In this context, one has to convert each word to a numerical value that the computer language can understand. However, vectorising text would lead to high dimensional matrices, which are often highly sparse. Sparse PCA can thus be applied to the vectorised text in order to effectively handle those sparse metrics for machine learning classifier models by which one can analyse text data. From a business point of view, this is important for dealing with high dimensional text with labels than can predict what class they belong to.

Our paper is motivated by a text dataset on the spam filter problem, which is an important application of NLP in real life. Emails are filtered into inbox and spam, and when collecting data we can class the labels as spam or ham. Based on the heading and text of the email, we can find patterns of what would attribute to an inbox or spam email. The dataset considered here contains a class of ham and spam labels, which are associated with emails received by a person from different sources (see Table 1 for a small picture of the data structure).

Recent work in high dimensional statistics has focused on sparse PCA since ordinary PCA estimates can become inconsistent in very high dimensional situations. In SPCA, the principal components are restricted to be sparse in the sense that there are only a few non-zero entries in the original basis. This has the advantage, amongst others, that the components are more interpretable [10], while components may no longer be uncorrelated.

The sparse PCA model we use here is as follows:

$$\Sigma = \underbrace{\begin{pmatrix} \overbrace{UDU^T}^{s} & \overbrace{0}^{p-s} \\ 0 & 0 \end{pmatrix}}_{signal} + \underbrace{\begin{pmatrix} \Gamma_1 & \Gamma_{12} \\ \Gamma_{21} & \Gamma_2 \end{pmatrix}}_{noise}, \qquad \Pi_d = \begin{pmatrix} UU^T & 0 \\ 0 & 0 \end{pmatrix}, \tag{4}$$

Table 1 The structure of the ham-spam dataset

	Label	body_text	body_len	Punct %
0	spam	Free entry in 2 a wkly comp to win FA Cup fina…	128	4.7
1	ham	Nah I don't think he goes to usf, he lives aro…	49	4.1
2	ham	Even my brother is not like to speak with me …	62	3.2
3	ham	I HAVE A DATE ON SUNDAY WITH WILL!!	28	7.1
4	ham	As per your request 'Melle Melle (Oru Minnamin …	135	4.4

in which $U \in \mathbb{R}^{s\times d}$ is the non-zero block of V_d, $D = \text{diag}(\lambda_1, \ldots, \lambda_d)$, "signal" $= \lambda_1 v_1 v_1^T + \cdots + \lambda_d v_d v_d^T$, "noise" $= \lambda_{d+1} v_{d+1} v_{d+1}^T + \cdots + \lambda_p v_p v_p^T$.

The decomposition is unique when $\lambda_d > \lambda_{d+1}$. Sparsity assumptions play an important role in a variety of other problems in high dimensional statistics, particularly linear regression analysis. Regression analysis is ill-posed in high dimensional situations, so by imposing sparsity on the regression parameters vector one can recover tractability [8].

When moving from a traditional PCA model to any of its sparse variations there are several ramifications that are crucial to assess, particularly regarding the understanding and interpretation of data. We discuss them in the following.

3.1 Moving Outside the Row-Space

Sparse loadings can be outside the data row-space because of the constraints/penalties applied. In actuality, there is no sparse arrangement inside the row-space in the existence of noise. On the off chance that the penalties are sensible, we can improve the model quality with it. A prominent case of this are the non-negative constraints, which are helpful to model non-negative information in the data. The withdrawal of a sparse component can be helpful.

3.2 Correlation of Loadings

Correlation of loadings is an important issue, specifically when scatter plots of scores are used for translation. The representation in scatter plots expects orthogonal axes in the first variable space. This holds for the standard PCA components, however it does not fundamentally hold for sparse components. Again, care ought to be taken when calculating the caught variance with associated or correlated loadings. Note that assessing and testing a significant variance in correlated models is a non-standard testing problem [11–14].

3.3 Correlation of Scores

The scores obtained by ordinary PCA are uncorrelated as an outcome of expanding variance in each component, but the scores of the SPCA model are correlated. This correlation convolutes appropriate visualisation yet in addition takes into account a progressively flexible modelling. Witten et al. [15] provide an answer for this correlation issue, however it is not evident whether it should be used. More critically, care ought to be taken when computing the caught variance with correlated scores.

3.4 Loss of Captured Variance per Principal Component

The use of main principal components is ideal when the caught variance is maximised. This is a property that makes PCA helpful for data clarification, because a huge segment of variance/information is captured by a much smaller set of components.

Table 2 The dataset head

	Label	body_text
0	ham	I ‘ve been searching for the right words to thank you for this breather. I promise I won’t take yo…
1	spam	Free entry in 2 a wkly comp to win FA Cup final tkts 21st May 2005. Text FA to 87121 to…
2	ham	Nah I don’t think he goes to usf, he lives around here though…
3	ham	Even my brother is not like to speak with me. They treat me like aids patent…
4	ham	I HAVE A DATE ON SUNDAY WITH WILL!!

Table 3 The dataset tail

	Label	body_text
5663	spam	This is the 2nd time we have tried 2 contact u. U have won the £750 Pound prize. 2 claim is easy…
5664	ham	Will u be going to esplanade fr home?
5665	ham	Pity, * was in mood for that. So… any other suggestions?
5666	ham	The guy did some bitching but I acted like I’d be interested in buying something else next week…
5667	ham	Rofl. Its true to its name…

This in turn rearranges the visualisation of the data, since a decreased number of PCs need to be visualised to represent a specific level of variance. However, some portion of the caught variance probably would not be of intrigue and could even entangle clarification. As an example, all PCs incorporate a specific measure of installed noise [16]. When we move to the sparse settings, we rearrange the understanding of loadings, yet in addition require more components to catch a similar measure of variance.

4 Analysis

4.1 The Data Description

The dataset we use contains texts, with two columns for “label” and “body_text” respectively. The data has 5568 rows and 2 columns and, out of 5568 rows, 746 are spam whilst 4822 are ham. The head and tail of the data are reported in Tables 2 and 3, respectively. There are no missing observations in the dataset.

4.2 Pre-processing the Text Data

Cleaning up the text data is usually a necessary step to prepare and highlight attributes that are required to be processed by the machine learning system. Cleaning or

pre-processing text data typically consists of a number of steps, such as removing punctuation, tokenising, removing stop words and stem. We needed to apply these steps to our text dataset.

4.3 Vectorising Text

After cleaning up the text dataset, we now need to understand how to get the text into a form that a machine learning model and software (we use Python) can understand and use to comprehend and train a model. The procedure which is used to convert the text into a form that Python and machine learning models can understand is called vectorising. This is characterised as the process of encoding text as integers to create feature vectors. A feature vector is an n-dimensional vector of numerical features that represent some object. So in this situation, that implies we take a singular text message and convert it to a numeric vector that speaks to that text message.

We start with crude text and tokenise the text. Then we carry out the vectorisation process. Fundamentally, what we do when we vectorise text is we take the dataset that has one line for each document with the cell entry as the genuine text message and then we convert it to a matrix that still has one line per document. Note that each word is utilised over all documents as the columns of our matrix, and then inside every cell is counting the number of times that a certain word showed up in the document. This is called document term matrix (see Table 4 for a sample matrix).

Once we have this numeric representation for every text message, we can continue down the pipeline and fit and train a desired model. Thus, we vectorise the text to create a matrix that just has numeric entries. We check how often each particular word appears in every text message. The machine learning algorithm comprehends these counts.

So in the event that it sees a one or a two or a three of a cell, then that model can start to correlate with whatever we are attempting to predict. In our case study, that would be spam. In this specific circumstance, it can utilise how frequently those certain words seem to decide whether the singular instant message is spam or not.

To better understand the spam-ham data as a text dataset, let us look at a small part of the data where we just have 10 text messages and we need to order them each as either spam or ham. Let us simply state there are 400 special words across those 10 text messages. That would mean our matrices, after vectorising, would still have 10 rows, one per text message. It would have 400 columns, one for every unique word. Every cell entry would check the number of times that word was utilised in

Table 4 A sample document term matrix

body_text	Call	Claim	Free	Text	Label
Free entry in …	0	0	1	1	spam
Nah I don't tl …	0	0	0	0	spam
HAVE A DA …	1	0	0	0	ham
As per your r …	0	2	1	0	ham
I'm gonna be …	1	0	0	0	spam

every text message (word embedding). Note that the outcome of this part, which is reported in Table 5, is just a small subset of the entire archive term matrix.

We simply concentrate on two words utilised here in content messages, offer and lol, alongside the name of either spam or ham. But once more, as a general rule, this "…" here would speak to the next 398 words that were utilised over these content messages. So, that is what this resembles after vectorising.

4.4 Applying TF-IDF Vector Technique

The "term frequency-inverse document frequency" method, known as the TF-IDF method, is a popular technique for text vectorising (see, e.g., [17]). The TF-IDF vector technique creates a document-term matrix where the columns represent single unique terms (unigrams) while each cell represents a weighting which indicates how important a word is to a document, as defined below [17]:

$$w_{i,j} = tf_{i,j} \times \log \frac{N}{df_i} \tag{5}$$

where $tf_{i,j} = number\ of\ times\ i\ occurs\ in\ j\ divided\ by\ total\ number\ of\ terms\ in\ j$, $df_i = number\ of\ documents\ containing\ i$, $N = total\ number\ of\ documents$.

We start with the TF expression in (5), which is the number of times that term i occurs in text message j, divided by the quantity of terms in text message j. It is simply the percentage of terms in this given text message that are this particular word. For example, on the off chance that we use "I like NLP", and the word we are centred around is NLP, then this term would be 1 separated by 3, or 0.33. Then, the second piece of this equation measures how as often as possible this word occurs across all other content messages. It ascertains the quantity of text messages in the dataset divided by the quantity of text messages that this word shows up in. That takes the log of the majority of that. Let us simply state that we have 20 text messages, so that is going to represent N in this case, and just one of those contains NLP. That will be df. The second piece of this equation would then be log of 20 isolated by 1. As this

Table 5 A sample document matrix

id	lets	lol	…	Label
1	0	2	…	ham
2	1	0	…	ham
3	0	0	…	spam
4	2	1	…	spam
5	0	4	…	ham
6	0	1	…	spam
7	1	1	…	ham
8	0	0	…	ham
9	0	0	…	spam
10	4	0	…	ham

division inside the log gets larger, the log of that part likewise gets larger. Now suppose that we have 40 text messages instead of 20, however NLP still just happens in one of them, so the denominator here will again be 1. Now, this division is 40 over 1. The term NLP is less continuous, and this term collectively will be larger.

Basically, the rarer the word is, the higher that its worth is going to be (i.e., higher $w_{i,j}$). If a word happens most of the time inside a particular text message, yet very infrequently elsewhere, that will be the second term. Then an exceptionally enormous number will be assigned, and it will be thought to be very important to separating that instant message from the others. In rundown, this strategy encourages us to pull out the important words.

We apply the TF-IDF vectoriser, and then we use this analyser parameter and pass it for the sake of our cleaning function. Then, we can store this all as tfidf_vect. Now that we have that put away, we can call the tfidf_vect and utilise the fit_transform capacity, and run that on data ['body_text'] column. Next, we store that vectorised data in X_tfidf. Just as we did previously, we can print out the X_tfidf shape, and we also print out the highlight names from tfidf_vect. We go ahead and run that. We observe that the shape is actually the same as it was for the tally vectoriser, 5567 pushes by 8104 columns. Here is the Python code for the TF-IDF vectoriser:

```
from sklearn.feature_extraction.text import TfidfVectorizer

tfidf_vect = TfidfVectorizer(analyzer=clean_text)
X_tfidf = tfidf_vect.fit_transform(data['body_text'])
print(X_tfidf.shape)
print(tfidf_vect.get_feature_names())
X_tfidf_df = pd.DataFrame(X_tfidf.toarray())
X_tfidf_df.columns = tfidf_vect.get_feature_names()
X_tfidf_df
```

The vectorisers output a "sparse matrix" whose most entries are 0 (see Table 6 for a small part of this large sparse matrix). In the interest of storage efficiency, a sparse matrix will then be used by storing only the locations of the non-zero elements.

4.5 Cross Validation

Cross validation involves model training based on a training dataset and testing the trained model on a holdout test dataset which has not been touched during the model training process. One can use k-fold cross validation to assess the performance of a machine learning model for classification (see, e.g., [18]). In this general procedure, the full informational index is k-subsets and the holdout strategy is repeated k times. That is, in every cycle one of the k-subsets is treated as the holdout test set and the other k-1 subsets are assembled to train the model. We here use fivefold cross validation to comprehend the potential execution results. In our case study, we start with a full informational index of 10,000 models and then run 5 overlay cross approvals. The initial step is to split the 10,000 model informational indexes into 5 subsets. There would be 5 subsets of information and everyone has 2000 models. So this is

Table 6 A small part of the sparse matrix of the vectorised text

	08002986030	08452810075over18	09061701461	1	100	100000	11	12	150pday	…	Wet	Win	Winner	w
0	0.0000	0.1989	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	…	0.0000	0.1749	0.0000	0.1989
1	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	…	0.0000	0.0000	0.0000	0.0000
2	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	…	0.0000	0.0000	0.0000	0.0000
3	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	…	0.0000	0.0000	0.0000	0.0000
4	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	…	0.0000	0.0000	0.0000	0.0000
5	0.0000	0.0000	0.2316	0.0000	0.0000	0.0000	0.0000	0.2316	0.0000	…	0.0000	0.0000	0.2316	0.0000
6	0.1976	0.0000	0.0000	0.0000	0.0000	0.0000	0.1976	0.0000	0.0000	…	0.0000	0.0000	0.0000	0.0000
7	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	…	0.0000	0.0000	0.0000	0.0000
8	0.0000	0.0000	0.0000	0.0000	0.2249	0.0000	0.0000	0.0000	0.2249	…	0.0000	0.1976	0.0000	0.0000
9	0.0000	0.0000	0.0000	0.2529	0.0000	0.2529	0.0000	0.0000	0.0000	…	0.0000	0.0000	0.0000	0.0000
10	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	…	0.0000	0.0000	0.0000	0.0000
11	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	…	0.0000	0.0000	0.0000	0.0000
12	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	…	0.2911	0.0000	0.0000	0.0000
13	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	…	0.0000	0.0000	0.0000	0.0000
14	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	…	0.0000	0.0000	0.0000	0.0000
15	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	…	0.0000	0.0000	0.0000	0.0000
16	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	…	0.0000	0.0000	0.0000	0.0000
17	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	…	0.0000	0.0000	0.0000	0.0000
18	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	…	0.0000	0.0000	0.0000	0.0000
19	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	…	0.0000	0.0000	0.0000	0.0000

inspecting without substitution, each of the 10,000 models are still represented and it is significant that these subsets would continue as before throughout the whole procedure. Therefore, a model in subset one will stay in subset one entirely through the part of the bargain approval.

4.5.1 Evaluation Metrics

Since the spam-ham informational collection is a classification problem, we use three primary evaluation metrics defined below:

$$\text{Accuracy} = \frac{\#\ \text{predicted}\ \ \text{correctly}}{\text{total}\ \ \#\ \text{of}\ \ \text{observations}}$$

$$\text{Precision} = \frac{\#\ \text{predicted}\ \ \text{as}\ \ \text{spam that are actually}\ \ \text{spam}}{\text{total}\ \#\ \text{predicted}\ \ \text{as}\ \ \text{spam}}$$

$$\text{Recall} = \frac{\#\ \text{predicted}\ \ \text{as}\ \ \text{spam that are}\ \ \text{actually spam}}{\text{total}\ \ \#\ \text{that are}\ \ \ \text{actally spam}}.$$

The first metric is accuracy which is the number that is anticipated accurately over the absolute number of perceptions. For example, in the event that we have 10,000 perceptions and we have 800 of them named effectively then the precision would be 80%. The second metric is precision which is the number that the model anticipated as spam accurately partitioned by the absolute number that the model anticipated as spam. The last evaluation metric is recall which is the number anticipated by the model to be spam that are really spam, so again that is a similar numerator as precision however now it is simply separated by the complete number that are really spam rather than the absolute number that are anticipated as spam.

4.6 Machine Learning Classifier Ensemble Methods

Ensemble learning methods refer to the techniques that create multiple models and then combine them to produce better results than any of the single models individually. We here utilise two ensemble learning methods for our text classification problem.

4.6.1 Random Forest

Random forest is an ensemble learning method that constructs a collection of decision trees and then aggregates the predictions of each tree to determine the final prediction. So in this case, the weak models are the individual decision trees, and then those are combined into a powerful model that is the aggregated random forest model. This can be used for both classification and regression. It can handle outliers, missing values, skewed data, and the data do not have to be on the same scale. It can also accept various types of inputs (e.g., ordinal or continuous data) as well.

Random forest is less likely to overfit compared to some of the other machine learning models.

4.6.2 Gradient Boosting

Gradient boosting is another ensemble learning method that takes an iterative approach to combining weak learners to create a powerful learner by focusing on the mistakes of prior iterations. It has some similarities with random forest, however there are some key differences. Gradient boosting uses decision trees as well, but they are incredibly basic, like a decision stump. Gradient boosting evaluates what it gets right and what it gets wrong on that first tree, and then with the next iteration it places a heavier weight on those observations that it got wrong focusing on the examples it does not quite understand yet until it has minimised the error as much as possible. Gradient boosting is one of the most powerful machine learning classifiers. It also works with various types of inputs like random forest, making it very flexible. A main drawback however is that it takes longer to train because it cannot be parallelised, so it is more likely to overfit because it obsesses over those ones that it got wrong, and it can get lost pursuing those outliers that do not really represent the overall population.

5 Evaluation of the Empirical Results

We apply both sparse PCA and ordinary PCA to our text dataset before implementing the two ensemble classifiers random forest (abbreviated here by rf) and gradient boosting (abbreviated here by gb). In addition to the evaluation metrics defined in Sect. 4.5, we also consider another criterion that is the run time. This enables us to track and understand how long it does take for each model to train and predict.

Table 7 presents the empirical results on accuracy, precision and recall, as well as the fit time and prediction time for the SPCA and PCA dimensionality reduction and machine learning methods. The results suggest that SPCA performs as good as PCA in terms of accuracy and precision, while it tends to be generally faster to train and predict compared to PCA. In addition to faster computations, SPCA also results in sparse principal components so it would be easier to interpret those sparse principal components compared to PCA, and the interpretability is indeed desirable as discussed in Sect. 2. It can also be seen that the number of components used is important too, as previously discussed in Sect. 1. The accuracy of both SPCA and PCA improves when the number of principal components is increased, probably because more information is used with more components.

We note that unlike the accuracy and precision performances, SPCA does not perform well in terms of the recall performance, compared to PCA. One possible reason for this could be due to the use of zero weights for some of the loadings, however this problem requires further research.

Table 7 The empirical performance of SPCA and PCA. Note that dr-ml is the abbreviation of dimensionality reduction and machine learning

dr-ml method	Number of components	Accuracy	Precision	Recall	Fit time (min)	Prediction time (min)
PCA rf	2	0.899	0.628	0.449	6.92	0.53
SPCA rf	2	0.893	0.610	0.214	6.15	0.50
PCA rf	4	0.919	0.745	0.650	7.97	0.89
SPCA rf	4	0.898	0.713	0.312	5.72	0.74
PCA rf	6	0.956	0.802	0.704	10.55	1.22
SPCA rf	6	0.930	0.785	0.371	7.46	0.96
PCA gb	2	0.885	0.536	0.385	11.28	0.90
SPCA gb	2	0.880	0.521	0.114	9.52	0.82
PCA gb	4	0.914	0.729	0.619	20.23	1.24
SPCA gb	4	0.892	0.697	0.301	17.44	1.02
PCA gb	6	0.950	0.786	0.680	32.05	1.51
SPCA gb	6	0.927	0.769	0.292	27.61	1.33

Also, random forest and gradient boosting are abbreviated by rf and gb, respectively

6 Conclusions

In this paper, we studied several formulations for sparse PCA, together with some algorithms for implementing those formulations. We applied sparse PCA to an application from natural language processing. The analysis required working with high dimensional sparse matrices obtained from such text data, and we used sparse PCA to handle and analyse those sparse matrices. Our empirical results showed that sparse PCA did perform as good as PCA in terms of accuracy and precision, whilst it showed two main advantages over PCA. First, sparse PCA results in sparse principal components so it is much easier to interpret those sparse principal components compared to PCA, and indeed interpretability is desirable. Second, the computational time is generally faster for sparse PCA because it uses sparse principal components. Faster calculation is important especially for big data situations.

Unlike the accuracy and precision performances, sparse PCA did not perform well in terms of the recall performance, compared to PCA. This is a topic to further research for natural language processing using sparse PCA.

The move from PCA to sparse PCA is interesting and advantageous, as discussed above. The text data that was vectorised into sparse matrices indeed showed the need for dimensionality reduction as a feature selection process for selecting the components used in the machine learning classifiers.

Finally, there are other problems to be studied when using sparse PCA. For example, outside of the locally merged calculation in [19], not many techniques can handle the issue of finding a few driving scanty head segments. Furthermore, most techniques even some basic ones perform all around ok on simple "normal" informational collections while just the costliest semidefinite relaxations appear to deliver great limits on the arbitrary lattices utilised in packed detecting

applications, or when just a couple of tests are accessible for instance. Portraying what makes "normal" informational indexes simpler than arbitrary ones remains an open problem now. It is likewise not clear yet how to expand the measurable optimality explanations of Amini and Wainwright [20] to more extensive, for example deterministic, classes of frameworks. It is also desirable to apply the sparse PCA to other areas of text analysis and data mining [21–24] and particularly big data [25].

References

1. Sirimongkolkasem T, Drikvandi R (2019) On regularisation methods for analysis of high dimensional data. Ann Data Sci 6(4):737–763
2. Collobert R (2014) Word embeddings through hellinger PCA. In: Proceedings of the 14th conference of the European chapter of the association for computational linguistics
3. Spruyt V (2014) The curse of dimensionality in classification. https://www.visiondummy.com/2014/04/curse-dimensionality-affect-classification. Accessed 16 Apr 2014
4. Aggarwal CC, Zhai C (2012) Mining text data. Springer, New York
5. Ning-min S, Jing L (2015) A literature survey on high-dimensional sparse principal component analysis. Int J Datab Theory Appl 8(6):57–74
6. Jolliffe IT (2002) Principal component analysis. EEUU: Springer, New York
7. Jolliffe IT, Trendafilov NT, Uddin M (2003) A modified principal component technique based on the LASSO. J Comput Graph Stat 12(3):531–547
8. Robert T (1996) Regression Selection and Shrinkage via the Lasso. J R Stat Soc B 58:267–288
9. Trevor H, Robert T (2015) Statistical learning with sparsity: the lasso and generalizations. Chapman & Hall/CR, London
10. Hui Z, Trevor H, Robert T (2006) Sparse principal component analysis. J Comput Graph Stat 15:265–286
11. Drikvandi R, Khodadadi A, Verbeke G (2012) Testing variance components in balanced linear growth curve models. J Appl Stat 39(3):563–572
12. Drikvandi R, Noorian S (2019) Testing random effects in linear mixed-effects models with serially correlated errors. Biom J 61(4):802–812
13. Drikvandi R (2017) Nonlinear mixed-effects models for pharmacokinetic data analysis: assessment of the random-effects distribution. J Pharmacokinet Pharmacodyn 44(3):223–232
14. Rao K, Drikvandi R, Saville B (2019) Permutation and Bayesian tests for testing random effects in linear mixed-effects models. Stat Med 38(25):5034–5047
15. Witten DM, Robert T, Trevor H (2009) A penalized matrix decomposition, with applications to sparse principal components and canonical correlation analysis. Biostatistics 10:515–534
16. Camacho J, Smilde AK, Saccenti E, Westerhuis JA (2020) All sparse PCA models are wrong, but some are useful. Part I: computation of scores, residuals and explained variance. Chemom Intell Lab Syst 196:103907
17. Zhang W, Yoshida T, Tang X (2011) A comparative study of TF*IDF, LSI and multi-words for text classification. Expert Syst Appl 38(3):2758–2765

18. Japkowicz N, Shah M (2011) Evaluating learning algorithms: a classification perspective. Cambridge University Press, Cambridge
19. Journee M, Nesterov Y, Richtarik P, Sepulchre R (2008) Generalized power method for sparse principal component analysis. arXiv:0811.4724
20. Amini, A.A. and Wainwright, M.J. (2008). High-dimensional analysis of semidefinite relaxations for sparse principal components. In: 2008 IEEE international symposium on information theory, pp 2454–2458
21. Olson D, Shi Y (2007) Introduction to business data mining. McGraw-Hill/Irwin, New York
22. Shi Y, Tian YJ, Kou G, Peng Y, Li JP (2011) Optimization based data mining: theory and applications. Springer, New York
23. Haddi E, Liu X, Shi Y (2013) The role of text pre-processing in sentiment analysis. Procedia Comput Sci ITQM 2013(17):26–32
24. Shi Y, Tang Y, Cui L et al (2018) A text mining based study of investor sentiment and its influence on stock returns. Econ Comput Econ Cybern Stud Res 52(1):183–199
25. Shi Y (2014) Big data: history, current status, and challenges going forward. The bridge. US Natl Acad Eng 44(4):6–11