L101 Practical Report

Do Pre-trained Word Vectors Help a Spam Classifier Generalise to new Email Data?

Zébulon Goriely, Queens', zg258

Word Count: 3592[1]

1 Introduction

Spam filtering is a text categorisation problem where a model must classify emails as either spam or genuine. As with many machine learning tasks, these models are typically trained on a corpus of relevant data. An issue with these supervised approaches is that of *adaptation*, the fact that the nature of spam and genuine email changes over time and between different users. Medlock (2006) addresses this issue by creating an interpolated model, designed to adapt to new, relevant instances of spam and genuine email data. Medlock also provides the *GenSpam* corpus for the purpose of evaluating systems that address this problem, consisting of two distinct distributions of spam/ham (spam and genuine) email data.

In this report, I study another approach to the adaptation problem by using *pre-trained word vectors*. These word vectors are trained on vast amounts of data to capture semantic and contextual information. I train and test a Support Vector Machines (SVM) classifier using both *sparse document vectors* (based on frequency of tokens in a document) and *dense document vectors* (taking the sum of word vectors for each token in a document) using the GenSpam corpus. I find that these dense document vectors increase accuracy when testing on a different distribution of spam/ham data and also generalise better than their sparse counterparts. Generalisation is determined by comparing accuracies achieved on the test set to a validation set, the latter consisting of emails from the distribution as the training set. Finally, I discuss the implications of this result concerning the use of SVMs for the spam filtering problem and for text classification in general.

[1] `texcount report/report.tex`

2 Background

2.1 GenSpam Dataset

The GenSpam corpus is a "representative, heterogeneous experimenal corpora" for spam filtering models assembled by Medlock. The corpus consists of 9072 genuine messages and 32332 spam messages and is divided into a *Training* set, an *Adaptation* set and a *Test* set. The emails in the *Training* set are sourced differently than the emails in the *Adaptation* and *Test* sets. The emails in the *Training* set come from a variety of sources, representing different users and topics, whereas the emails in the *Adaptation* and *Test* sets come from the contents of just two user inboxes, representing a real-world instance of the spam filtering problem. As both sets come from the same source, the *Adaptation* set represents highly relevant samples that an adaptive classifier, like the one described in Medlock's paper, can use to improve spam filtering. These two sets can be considered a divergent distribution of emails when compared to the *Training* set and so can also be used to indicate if a model generalises well to a new source of spam/ham data.

Medlock's paper also discusses the spam filtering problem as being set in the *semi-structured document classification* framework. Rather than regarding emails as having a flat structure, his classifier interpolates over the tree-structure of the email, combining estimates for both the **Subject** and **Body** of the email. GenSpam formats emails in XML to supports systems that make use of this structured information.

2.2 Sparse Document Vectors

Simple baseline systems for the text classification problem in NLP often make use of the *naive Bayes assumption*, that all features of a document are independent from each other given the context of the class (McCallum et al., 1998). This assumption leads to a common approach to document representation, the Bag-of-Words (BoW) model, which assigns a fixed-length vector to each document. If there are V words used by the model, then each document is assigned the *sparse document vector*

$$\mathbf{d} = (n_1, n_2, \ldots, n_V),$$

where n_j is the number of occurrences of word w_j in the document. The vocabulary used by the model does not necessarily coincide with the set of tokens present in the documents, as various pre-processing steps may stem the words, remove stopwords and/or remove low frequency words.

The BoW model is often criticised for its extreme sparsity (a document will likely contain only a fraction of the words used in the vocabulary) and that it does not effectively capture semantics (semantically similar documents may be mapped to very different vectors) (Zhao and Mao, 2017). Nonetheless, this model has proved very effective for a number of text classification tasks (Pang et al., 2002).

A slight variant is the bag-of-n-grams model (Harris, 1954) which relaxes the naive Bayes assumption to consider n-tuples of adjacent words as features. The inclusion of bigrams in particular increases performance in many classification tasks (Wang and Manning, 2012), while the use of higher-order n-grams often decreasing performance due to data sparsity and overfitting. In this report, I only consider unigrams as n-grams are not supported by pre-trained word vector sets.

2.3 Dense Document Vectors

In the last twenty years, the use of vectors to capture semantic similarity (Lund and Burgess, 1996) has become increasingly popular in NLP and cognitive science. As computational power has become cheaper and more widely available, it has become more feasible to use these vectors in many NLP tasks. One example is the popular `word2vec` system popularised by Mikolov et al. (2013) that produces word vectors that retain multiple degrees of similarity. Unlike the sparse vectors described above, the dimensions of these vectors are not directly interpretable. They exist in a vector space where word vectors that are close to each other correspond to words with similar meaning.

There are now a variety of systems that produce such vectors, including `word2vec`, `GloVe` (Pennington et al., 2014) and `fastText` (Bojanowski et al., 2017). These systems have been trained on huge amounts of data (billions of tokens) to produce pre-trained vectors that can be downloaded and incorporated into machine learning systems. The hope is that these vectors are *universal*, that through exposure to vast amounts of data and a variety of contexts, each word has absorbed enough semantic information to be useful to these systems.

Using these pre-trained word vectors, it is possible to generate *dense document vectors* using a *compositional distributional semantics* (CDS) model. Mitchell and Lapata (2008) proposed a framework for representing the semantics of phrases and sentences in a vector space using vector composition operations. The simplest such operation is the *basic additive model* (BAM) whereby a distributional vector of a sequence of words is simply the weighted sum of the vectors for the component words, with the simplest parameterisation setting all weights to 1.

Using the basic additive model along with pre-trained word vectors, simple *dense document vectors* can be produced using the *sparse vectors* described in section 2.2. If we consider all n pre-trained word vectors of length p, we can produce a $p \times V$ word-vector matrix W. If we generate sparse document vectors of length n using the same vocabulary as the pre-trained word vectors (such that the i^{th} word in the sparse vector is also represented by the pre-trained vector in the i^{th} column of W) then we can produce dense doc-

ument vectors $\mathbf{d}'$ from $\mathbf{d}'$ as follows

$$\underbrace{\mathbf{d}'}_{p\times 1} = \underbrace{W}_{p\times V} \cdot \underbrace{\mathbf{d}}_{V\times 1} \tag{1}$$

The resulting dense vector $\mathbf{d}'$ has length p rather than length V. The value p for the length of the pre-trained vectors is arbitrary but often ranges from 50–300, much smaller than the size of the vocabulary.

Despite now incorporating some notion of semantic similarity, this process still makes use of the *naive Bayes assumption* since the resulting word vectors do not depend on the word order in the original document. There are many other ways of generating document vectors that account for word order, ranging from the other compositional operations described by Mitchell and Lapata (2010) to the unsupervised approach taken by the *Paragraph Vector* system (Le and Mikolov, 2014) for generating fixed-length vectors from variable-length pieces of text, making use a local context "window". In this report, I only consider the basic additive approach.

2.4 Support vector machines

SVMs are a supervised learning technique used for classification and regression, shown to be an effective and competitive baseline for text categorisation tasks such as topic classification and sentiment detection (Joachims, 1998; Wang and Manning, 2012). The training procedure consists of finding a hyperplane in the feature space that not only separates the labelled training vectors, but does so in such a way that the separating margin is as large as possible. Classification of vectors can then proceed by simply determining on which side of the hyperplane the test vectors fall.

It has been shown that SVM classifiers struggle to provide an optimal decision boundary when the training and test distributions diverge (Forman and Cohen, 2004). We would then expect an SVM-based spam classifier trained on emails from a particular source to perform worse when testing on emails from a different source. This makes SVMs a good candidate to test for whether dense document vectors can help the classifier generalise to a new source of data. We can do this by comparing the difference in classification performance on two testing sets, one which comes from the same distribution that the classifier was trained on and one which comes from a different distribution.

In this report, I use the SVC class from the `scikit-learn` library in Python (Pedregosa et al., 2011), which internally uses the `LIBSVM` library (Chang and Lin, 2011) to handle computations. SVC stands for Support Vector Classification and uses the C-SVC algorithm (Boser et al., 1992) for training the model.

3 Method

3.1 Data

I use the GenSpam corpus for training and testing my classifier. I do not use the *Adaptation* set, instead comparing results achieved on a held-back portion of the *Training* set to results achieved on the *Test* set. Although the corpus contains many email fields in XML format, I only use the **Normal Text** component of the **Body** field of each email. Some emails did not contain any text in the **Body** field, I chose to keep these as the classifier should be able to deal with this edge case. When extracting data from the *Training* set I encountered Unicode decoding errors on 1 line of the genuine emails and 23 lines of the spam emails. I chose to remove the emails that contained these errors.

From the *Training* and *Test* sets of the corpus I produce three sets of data I use for this experiment; one training set, one validation set and one test set. The test set is the entirety of the *Test* set provided by the corpus, representing a user-specific instance of the spam filtering problem, a different source of data than the *Training* set. The validation set and training set come from the *Training* set of the corpus with the validation set selected to contain the same number of genuine and spam emails as the testing set. This validation set is not used for hyper-parameter tuning but rather to be used alongside the test set to indicate whether the models generalise well to a different source of spam/ham data, compared to performance on the same source with which they

were trained. All three sets are shuffled. This resulted in the following number of emails for each set used in the experiment:

- Training set: 7191 genuine, 15729 spam
- Validation set: 749 genuine, 713 spam
- Testing set: 749 genuine, 713 spam

3.2 Pre-Processing

The GenSpam corpus is already tokenised. I do not remove stopwords, following Medlock's observation that stopword removal has a detrimental effect on classification, "presumably due to the fact that stopword usage differs between spam and genuine email". I also use a regular expression to remove words longer than 15 characters and shorter than 2. I do not stem any tokens because the words in the vocabulary of pre-trained word vectors are not stemmed.

The data preparation and pre-processing is carried out by `scripts/prepare_data.py` with the resulting data found in the `data/preprocessed` directory of the GitHub repository associated with this report[2].

3.3 Creating Vectors

I create sparse and dense document vectors for all emails in the training, validation and testing sets. To create sparse vectors, I use the `CountVectorizer` class of the `scikit-learn` library, which produces vectors of length V as described in section 2.2. V is the length of the vocabulary associated with the training set — if there are unseen words in the validation or testing data, they will not be incorporated into the sparse document vectors for those emails.

To create dense document vectors, I use the pre-trained word vector sets provided by `word2vec`[3], `GloVe`[4] and `fastText`[5]. These systems provide several different sets of vectors differing in training size, vector length and number of words provided, as seen in table 1. I chose a variety of sets provided by each system to use for this experiment, to examine if the length of these word vectors and the size of the training corpus have a significant effect on spam classification accuracy.

To create dense vectors I implemented eq. (1). One subtlety is that this computation requires the vocabulary used by the sparse vectors to be contained within the vocabulary of the pre-trained word vectors. In practice, the GenSpam dataset contains many rare words, typos or other tokens not present in the vocabularies of the pre-trained word vectors (mostly from the spam emails) and so do not have associated word vectors. To account for the differences in vocabulary, I also produce sparse word vectors that only count words that also appear in the vocabulary of the pre-trained word vector set, and use them to produce the dense word vectors. For the `fastText` vectors, this reduced the length of the vocabulary (and the lengths of the sparse vectors) from 53554 to 30957. In order to ensure that classification results using the dense vectors are not a product of this reduced vocabulary size, I also report results achieved using these shorter sparse vectors.

A description of all vectors produced by this step can be seen in table 2. The code for this step can be found in `scripts/prepare_vectors.py` and the training, validation and testing vectors generated by this script can be found in the `data/vectors` directory of the GitHub repository.

3.4 Classification

To classify the emails, I trained an SVM using the SVC class from `scikit-learn`. I use a linear kernel and use `scikit-learn`'s StandardScaler class to scale features to unit variance (otherwise some features may dominate the SVM's objective function). I use default parameters for the SVM otherwise. For each vector type described

[2] `https://www.github.com/codebyzeb/l101`
[3] `https://code.google.com/archive/p/word2vec`
[4] `https://nlp.stanford.edu/projects/glove`
[5] `https://fasttext.cc/docs/en/english-vectors.html`

Training System	Vector Length	Vocabulary Size	Training Data
fastText	300	1M	Wikipedia + statmt.org news (16B tokens)
fastText	300	2M	Common Crawl (600B tokens)
GloVe	**50**, 100, 200 & 300	400K	Wikipedia + Gigaword 5 (6B tokens)
GloVe	**300**	1.9M	Common Crawl (42B tokens)
GloVe	300	2.2M	Common Crawl (840B tokens)
GloVe	25, 50, 100 & 200	1.2M	Twitter (2B tweets)
word2vec	300	3M	Google News (100B tokens)

Table 1: A selection of popular pre-trained word vector sets for English with those used in this experiment marked in **bold**.

Name	Vocabulary Size	Vector Length
sparse-full	53781	53781
sparse-fasttext	31382	31382
dense-fasttext	31382	300
sparse-glove50	29727	29727
dense-glove50	29727	50
sparse-glove300	36156	36156
dense-glove300	36156	300
sparse-word2vec	28679	28679
dense-word2vec	28679	300

Table 2: Description of sparse and dense vectors used in this experiment.

in table 2, the classifier is trained on the training vectors and tested on the validation and testing vectors. The code for this step can be found in `scripts/classify_emails.py` in the GitHub repository.

4 Results

For each vector type, I report the *recall* for both classes and the overall *accuracy* achieved by the classifier, seen in table 3. The table also displays the drop in accuracy between the tests on the validation and testing set for each vector type.

4.1 Generalisation of Dense Vectors

As expected, the performance of the classifier tested on the testing set is lower than when testing on the validation set, for every vector type. As discussed in section 2.1, this is due to the fact that emails in the validation set come from the same source as the training set whereas the testing set represents a completely different source of spam/ham data.

What is clear from the results is that this drop in performance between the two test sets is consistently smaller when using dense vectors than when using sparse vectors, suggesting that the dense vectors do indeed help the classifier generalise to a new source of spam/ham data. For example, the accuracy when using the `dense-fasttext` vectors drop from 92.8 to 92.7 whereas the accuracy when using the `sparse-fasttext` vectors drops from 94.0 to 87.3. In fact, the smallest drop in accuracy when using sparse vectors (6.6 points for `sparse-word2vec`) is larger than the largest drop when using dense vectors (1.8 points for `dense-glove300`).

The dense vectors also lead to better performance on the testing set than the corresponding sparse vectors for each vocabulary set (with the exception of `dense-glove50`), with `dense-fasttext` giving the best accuracy out of all vector types. This is achieved despite the sparse vectors per-

Vector Type	Validation Set			Testing Set		
	Gen Recall	Spam Recall	Accuracy	Gen Recall	Spam Recall	Accuracy
sparse-full	90.8	96.8	93.7	80.5	90.0	85.2 (-8.5)
sparse-fasttext	90.7	97.5	**94.0**	83.8	91.0	87.3 (-6.7)
dense-fasttext	86.9	99.0	92.8	88.8	96.8	**92.7** (-0.1)
sparse-glove50	90.3	97.2	93.6	83.0	90.9	86.9 (-6.7)
dense-glove50	64.8	97.8	80.8	64.8	96.5	80.2 (-0.6)
sparse-glove300	90.1	97.5	93.7	82.1	91.3	86.6 (-7.1)
dense-glove300	87.7	98.2	92.8	84.9	97.5	91.0 (-1.8)
sparse-word2vec	90.4	97.6	93.9	83.2	91.7	87.3 (-6.6)
dense-word2vec	84.8	98.6	91.5	83.3	97.8	90.4 (-1.1)

Table 3: Results of the SVM classifier tested on the validation and testing data sets for each vector type, with best results in **bold**. The bracketed values represent the drop in accuracy between the validation and testing sets.

forming better than all corresponding dense vectors on the validation set. It may be that sparse vectors are a better option for when the training and testing data come from the same distribution, as with the training and validation sets here. This result is of less interest, however, since in a real-world implementation of the spam filtering task, it is unlikely that the emails being filtered come from the same distribution as those used to train the filter. Instead, we care more about the results achieved on the testing set and about the ability of the classifier to generalise and we conclude that dense vectors are better than sparse vectors for this task.

4.2 Choice of Pre-trained Vectors

Despite the fact that the pre-trained word vectors used to produce the dense vectors were trained using different algorithms on different datasets of varying lengths, the classifiers that use the `dense-fastText`, `dense-glove300` and `dense-word2vec` vectors all achieve similar results, getting accuracies of 92.7, 91.0 and 90.4 respectively on the testing set. It is interesting that the best result is achieved when using the `fastText` pre-trained word-vectors, as these provide the smallest vocabulary and were trained with the smallest corpus of the three (only 16B tokens, compared to the 42B and 100B for the `GloVe` and `word2vec` word vectors). The classifier using the `dense-glove50` vectors performs much worse than these three, giving an accuracy of 80.2 on the testing set. The pre-trained vectors used for this system have the smallest vocabulary size (400M), shortest length (50) and were trained on the fewest tokens (6B). It is unclear which of these factors contributes to this lower performance. However, it is clear that in order to achieve good performance using dense vectors, certain properties are required of the pre-trained vectors used.

4.3 Feature Selection for Sparse Vectors

The only difference between all of the `sparse` vectors is the vocabulary used to produce them (hence, also their length). The accuracies achieved by these vectors are similar, with the `sparse-fasttext` achieving the highest score out of all vector types when testing on the validation set (94.0) and also achieving the highest score out of the sparse vectors when testing on the testing set (87.3). The `sparse-full` vectors consider 22399 more features than the `sparse-fasttext` vectors, none of which exist in the one million word vocabulary of the pre-trained `fastText` vectors. Despite these additional features, the classifier that uses this vector type performs worse than the other sparse vectors on the testing set and also has the largest drop in performance between the two test sets out of all classifiers. This result suggests that including these rare words not only decreases classification accuracy but also hinders the ability of the clas-

sifier to generalise to new data, likely because the rare words considered do not appear again in the testing set, causing the classifier to overfit.

A common technique for constructing sparse vectors is to only consider features whose count surpasses a "minimum frequency" threshold. If this is applied to the `sparse-full` vectors, the vector length decrease from 53781 to 25645, with accuracy on the testing set increasing to 85.8. This accuracy is still lower than the other sparse vectors but nonetheless demonstrates the importance of feature selection.

5 Discussion

My results show that using dense vectors produced by composing pre-trained word vectors associated with the words in a document helps an SVM model generalise to new distributions of email data. Further work could investigate if such methods could be applied to other classification problems that use SVMs to find if this holds true for other datasets.

The limitation of this study is that I cannot claim that dense vectors will help *all* spam classifiers generalise to new sources of spam/ham data. For instance, a simple Naive Bayes classifier using the *sparse-full* vectors, trained in the same way as the SVM classifier used here, achieves accuracies of 95.1 and 95.3 on the validation and testing sets respectively. There seems to be no issue of degrading performance when testing on a different distribution: generalisation may not even be a problem for this model. Further work would be required to examine this question for Naive Bayes and other baseline classification models.

Another aspect of the spam filtering problem to consider is that the correct classification of genuine mail is far more important than the misclassification of spam. This corresponds to achieving a high recall score for classifying genuine mail. In table 3, we see that that this score is, in all cases, lower than both the spam recall score and the overall accuracy score. The particular difference between the genuine mail recall score and the spam mail recall score may be explained by the fact that the training data is unbalanced. It has been identified that the hyperplane produced by training an SVM with an unbalanced dataset can be skewed towards the minority class and can degrade the performance of the model with respect to that class (Veropoulos et al., 1999). In this case, fewer samples of genuine mail may have resulted in the misclassification of genuine mail. Along with the statement in section 2.4 that SVMs produce sub-optimal decision boundaries in the presence of divergent distributions, this may indicate that the characteristics of SVM classifiers do not make them an appropriate choice for spam filtering systems.

Conclusion

I have shown that using dense vectors, consisting of a composition of pre-trained word vectors, can help an SVM spam classifier generalise well to a different distribution of spam/ham data. I have compared different sources of pre-trained word vectors to find that vectors with a sufficiently large vocabulary size and of sufficient length all lead to better performance than when using sparse vectors constructed using the same vocabulary. I have discussed the issue of feature selection and in particular the "minimum frequency" threshold with regards to the spam filtering problem. Finally, I concluded that this study could have implications for the problem of generalisation of SVMs, but further work would be required to show that these dense vectors help with the spam filtering problem, and with the wider field of text classification in general.

References

Bojanowski, P., Grave, E., Joulin, A., and Mikolov, T. (2017). Enriching word vectors with subword information. *Transactions of the Association for Computational Linguistics*, 5:135–146.

Boser, B. E., Guyon, I. M., and Vapnik, V. N. (1992). A training algorithm for optimal margin classifiers. In *Proceedings of the fifth annual workshop on Computational learning theory*, pages 144–152.

Chang, C.-C. and Lin, C.-J. (2011). Libsvm: A library for support vector machines. *ACM transactions on intelligent systems and technology (TIST)*, 2(3):1–27.

Forman, G. and Cohen, I. (2004). Learning from little: Comparison of classifiers given little training. In *European Conference on Principles of Data Mining and Knowledge Discovery*, pages 161–172. Springer.

Harris, Z. S. (1954). Distributional structure. *Word*, 10(2-3):146–162.

Joachims, T. (1998). Text categorization with support vector machines: Learning with many relevant features. In *European conference on machine learning*, pages 137–142. Springer.

Le, Q. and Mikolov, T. (2014). Distributed representations of sentences and documents. In *International conference on machine learning*, pages 1188–1196.

Lund, K. and Burgess, C. (1996). Producing high-dimensional semantic spaces from lexical co-occurrence. *Behavior research methods, instruments, & computers*, 28(2):203–208.

McCallum, A., Nigam, K., et al. (1998). A comparison of event models for naive bayes text classification. In *AAAI-98 workshop on learning for text categorization*, volume 752, pages 41–48. Citeseer.

Medlock, B. (2006). An adaptive, semi-structured language model approach to spam filtering on a new corpus. In *CEAS*.

Mikolov, T., Chen, K., Corrado, G., and Dean, J. (2013). Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*.

Mitchell, J. and Lapata, M. (2008). Vector-based models of semantic composition. In *proceedings of ACL-08: HLT*, pages 236–244.

Mitchell, J. and Lapata, M. (2010). Composition in distributional models of semantics. *Cognitive science*, 34(8):1388–1429.

Pang, B., Lee, L., and Vaithyanathan, S. (2002). Thumbs up? sentiment classification using machine learning techniques. *arXiv preprint cs/0205070*.

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

Pennington, J., Socher, R., and Manning, C. D. (2014). Glove: Global vectors for word representation. In *Empirical Methods in Natural Language Processing (EMNLP)*, pages 1532–1543.

Veropoulos, K., Campbell, C., Cristianini, N., et al. (1999). Controlling the sensitivity of support vector machines. In *Proceedings of the international joint conference on AI*, volume 55, page 60.

Wang, S. I. and Manning, C. D. (2012). Baselines and bigrams: Simple, good sentiment and topic classification. In *Proceedings of the 50th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 90–94.

Zhao, R. and Mao, K. (2017). Fuzzy bag-of-words model for document representation. *IEEE transactions on fuzzy systems*, 26(2):794–804.