Generating Relevant Tags for Academic Papers

Jenny Kim
Stanford University
yeongjaek@stanford.edu

Jihyeon Janel Lee
Stanford University
jlee24@stanford.edu

James Li
Stanford University
dawwctor@stanford.edu

December 27, 2017

Abstract

Text classification is a recurring task, especially in fields with a heavy use of domain-specific terminology, such as academia. In our paper, we propose a variety of methods for generating identification tags that are tailored to the content of academic papers. We focus on increasing recall to recommend related categories to paper authors and encourage interdisciplinary research. We demonstrate the advantages and disadvantages of each method with the goal of enabling more precise classification. We find that our simpler models (e.g. SVMs) performed better overall than our more complex models (e.g. RNNs), although the majority of our models could correctly classify journals over half the time.

1 Introduction

1.1 Problem

Just this past month, over 11,500 papers were published to arXiv in over a hundred different fields and subfields. Along with the growing volume of papers, research has become increasingly interdisciplinary, making it more difficult to separate papers into specific academic departments.

1.2 Related Works

Text classification is a prevalent task in many areas of natural language processing (NLP), having applications in sentiment analysis, question answering, and short-text or document classification. For example, previous work in this area includes classifying different texts by genre [11] and summarizing the content of various blocks of text [5]. Several different approaches have been developed for both short-text and document classification. For short-text classification, models with rule-based features have been quite successful, including Support Vector Machines (SVMs) [12], a combination of SVMs and naive Bayes [13], and dependency trees with conditional random fields [14].

However, NLP has benefited with the rise of deep learning, and recent studies have shown promising results with neural networks, including hierarchical structures like convolutional neural networks (CNNs) [15] and sequential models like recurrent neural networks (RNNs) [16]. Hierarchical models have been considered to be better for sentiment classification since it usually depends on key phrases, while sequential models have been preferred for language modeling because of its flexibility in context dependencies [18]. However, there has been no clear conclusion drawn in NLP literature, and RNNs have performed well on document-level sentiment classification, while CNNs have outperformed LSTMs in language modeling tasks in some cases [20, 19, 18]. We thus experimented with both RNNs and CNNs.

Since both deep and relatively simpler models have proven to be robust in text classification tasks, we decided to experiment with methods from those different categories, establishing a baseline using Naive Bayes and testing SVM, Multi-Layer Perceptron Neural Network, Random Forests, and RNN. Because our task is specifically in the realm of generating labels for academic papers, we also differ from these past works in that we focus on improving recall.

2 Data and Feature Extraction

2.1 Data Collection

Our project's focus is on data that can be collected from scientific journals, specifically those related to STEM fields, such as Computer Science, Mathematics, Physics, etc. We decided to scrape data from the arXiv database, as we could easily leverage its search query API to obtain information about each paper, including metadata like the ID and the category tags, in addition to its abstract. While there are eight broad categories, we collected abstracts from each of the more specific 127 subcategories. We curated a small dataset of 30 abstracts per category because some of our more complex models could not handle more training data within our computing constraints, but we also had a large training set of 100 abstracts per category.

We structured our data into a stream of abstracts with their metadata, like the following example below:

```
  <id>
0704.1711v2
  <category>
stat.AP
  <abstract>
The paper deals with the study of labor market
dynamics, and aims to characterize its
equilibriums and possible trajectories.  The
theoretical background is the theory of the
segmented labor market.  The main idea is that this
theory is well adapted to interpret
the observed trajectories, due to the heterogeneity of
the work situations.
```

Some examples of the categories are stat.AP (Statistics - Applications), q-bio.BM (Quantitative Biology - Biomolecules), and cs.AI (Computer Science - Artificial Intelligence).

2.2 Feature Extraction

To model our data, we first preprocessed our inputs, which were strings containing the abstracts, by tokenizing and normalizing the text. The process of normalization included making all letters lowercase, removing interjections and determiners by leveraging part-of-speech (POS) tagging, and lemmatizing each word into their base forms. (e.g. "goes" to "go"). We then converted each abstract into a bag-of-words vector and extracted TF-IDF features.

TF-IDF stands for term frequency-inverse document frequency, in which each word is assigned a score and the score increases if it appears frequently but decreases if it appears commonly across documents (e.g. the word "the" appears frequently but would not have a high score because it occurs in all the documents).

We chose to use TF-IDF Bag of Words (BoW) vectors to model our data because TF-IDF features can be good indicators of what the important or significant words are for a given abstract; this can be especially informative for academic papers because an abstract in one field most likely uses domain-specific terminology that differentiates itself from an abstract in another. These TF-IDF features could serve as good indicators of a certain category and signal if they are present in a given abstract. We also normalized the vectors to unit length so that the original length of the text would not have an effect. Finally, we used `scikit` to split our total data in training and test data sets.

3 Baseline and Oracle

3.1 Baselines

We used two baselines: majority algorithm and Naive Bayes.

3.1.1 Majority Algorithm

The Majority algorithm reads in total amount of data present for each category. It then chooses a category with the maximum number of examples for every prediction. For every test example, regardless of the input, the majority algorithm returns the maximum category.

Unfortunately, as we have 100 data points for each of the 126 categories, there is no one category which contains the most data. Thus, this algorithm effectively chooses a random category. Using this method, we expect to accurately classify $\frac{1}{126}$ of the data but misclassify $\frac{125}{126}$ of the data.

In terms of pros and cons, this algorithm is certainly significantly faster than all of our other machine learning algorithms, but it yields a particularly bad recall, which is our main focus.

3.1.2 Naive Bayes

Naive Bayes is commonly used in text classification tasks (e.g. spam detection) and can be trained easily with relatively fast prediction. However, one of its major disadvantages is that it assumes the features are independent, which may not be the case because the order of the words can be significant.

To describe how a Naive Bayes classifier works, it calculates a prior for each category, e.g. 0.01 of the abstracts are stat.AP, 0.03 are math.AG, etc. Since our abstracts are labeled, we can also calculate the likelihood, or the probability that there are certain TF-IDF features present in an abstract given a category label. The classifier keeps track of a table of frequencies or a representative vector of the TF-IDF features for each category. Then, since we have both the likelihood and the prior probabilities, we can calculate the posterior probability, the probability that a given abstract should be in a certain category given the evidence, or its TF-IDF features. Thus, during testing, we extract the features of a test abstract and calculate this posterior probability for each category and return the most likely label.

Naive Bayes is relatively fast and easy to train compared to our other machine learning algorithms, but it assumes that all the features are conditionally independent. While this assumption is robust, and may suffice for more general texts, we deal with more field-specific knowledge and academic journals, so the presence of a particular word might heavily affect the usage of other words in related fields, explaining why this is merely a baseline.

3.2 Oracle

For our oracle, we used the categories and tags generated by human judgment. The humans will be given a set of predefined categories off of which they can label certain papers. Assuming that these humans have domain-specific knowledge and a fundamental understanding of the various domains, they will be able to classify the abstract accurately.

Because we assume that the humans would be able to understand the contents of the abstract, the oracle will give much better results than both the majority algorithm and Naive Bayes.

4 Algorithms and Approaches

We originally considered multi-label classification methods because a research paper can oftentimes refer to multiple different topics across several different departments. However, after implementing a variety of multi-label classification models, ensemble methods and multiple binary-label classification to name a few, we found a surprisingly drastic decrease in accuracy. This, combined with the difficulty of obtaining multi-labeled training data, persuaded us to restrict our abstract classification to just one of these categories.

4.1 Linear Support Vector Machine Classifier

Because Linear Support Vector Classifiers (Linear SVCs) are very good at incorporating a large number of features, they are a popular option for text classification. Scikit also mentions that linear SVC's perform well in the high-dimensional space, which could be useful as we have a large variety of words that could lead to higher-dimensional vectors.

As in Naive Bayes, Linear SVC used TF-IDF vector as an input X vector. These TF-IDF vectors will be constructed by parsing and transforming the abstract. Later, the vector was scaled using L_2 normalization. Our input y vector is a label (stat.AP, quantph, astroph, etc.), which is a string representing which label the abstract belongs to. In addition to this, we used a hinge loss model as our loss function for optimization.

Once the input is provided, Linear SVC will start plotting the features onto the hyperspace. Then, based on which category the data belongs to, the Linear SVC will draw hyperplane that (in an optimal situation) only contains single label. The test data set will be classified into which hyperspace it belongs to. The test data will also have labels and TF-IDF vectors. Because Linear Support Vector Classifiers (Linear SVC) are really good at incorporating a large number of features, Linear SVC's are really popular for performing text classification. Scikit also mentions that linear SVC's perform well in the high-dimensional space, which could be useful as we have a large variety of words that could lead to higher-dimensional vectors.

As in Naive Bayes, Linear SVC used TF-IDF vector as an input X vector. These TF-IDF vectors will be constructed by parsing and transforming the abstract. Later, the vector was scaled using L_2 normalization. Our input y vector is a label (stat.AP, quantph, astroph, etc.), which is a string representing which label the abstract belongs to. In addition to this, we used a hinge loss model as our loss function for optimization.

Once the input is provided, Linear SVC will start plotting the features onto the hyperspace. Then, based on which category the data belongs to, the Linear SVC will draw hyperplane that (in an optimal situation) only contains single label. The test data set will be classified into which hyperspace it belongs to. The test data will also have labels and TF-IDF vectors.

4.2 MLP Neural Network

We then chose to use a Multi-layer Perceptron (MLP) Neural Network because it can learn an approximate non-linear function for classification and potentially reveal more complex patterns that link abstract content to research paper category. However, some disadvantages are that it can easily get caught in local minima and that it requires further additional training in order to properly tune the hyperparameters of the neural network. The model learns a function $f(\cdot) : R^m \rightarrow R^o$ by training on our dataset of TF-IDF values, where m is the number of dimensions for input and o is the number of dimensions for output.

The input layer consists of a set of neurons $\{x_i|x_1, x_2, ..., x_m\}$ representing the input features. These input features are the TF-IDF vectors made from the words over all the abstracts. For example, one particular sentence might calculate to the input vector $[0.25, 0.1, 0.005]$.

Each neuron in the hidden layer transforms the values from the previous layer with a weighted linear summation $w_1x_1 + w_2x_2 + ... + w_mx_m$, followed by a non-linear activation function $g(\cdot) : R \rightarrow R$, which in our case is the rectified linear unit function $g(x) = \max(0, x)$. In our provided example, the weights might be $w_1 = -0.25, w_2 = 0.5, w_3 = 0.1$, which would make the input value to the hidden neuron -0.012, and the output of the hidden neuron $g(x) = \max(0, x) = 0$.

The output layer receives the values from the last hidden layer and linearly transforms them to produce an output value y, where y is the string of which category and label the abstract belongs to. The model then uses back-propagation on these two input and output arrays to further train the neural network. This back-propagation will use the loss and gradients to adjust the linear weights of the input variables to the hidden neurons and the resulting weights to the output result to eventually train a more accurate model.

By using the TF-IDF vectors as inputs and weighing them appropriately, we can combine and train a model that could more accurately recognize the latent structures that link abstracts to content, since it is unlikely to be based solely off of linear weighings of which words are present or not.

4.3 Random Forests

We also ran experiments using Random Forests because we believed this learning method could remedy our overfitting problem. However, Random Forests are really sensitive to the hyperparameters, so tuning the hyperparameters required considerable time. Random Forests are ensembles of multiple decision trees. This algorithm constructs multiple decision trees, with each of these trees taking in some fraction of the data. Individual trees tune their own parameters during the training period.

For the test data, the Random Forest samples a certain fraction of the decision trees and asks for their classification result. Using a majority vote, the algorithm decides where the test data should be classified into. Even though decision tree methods are notorious for overfitting, because random forests use multiple decision trees to produce a final decision, they are much less likely to overfit.

The Random Forest algorithm itself does not utilize cross-validation. Thus, in order to ensure that the results we got were not extremely biased, we manually cross-validated our output by randomly choosing data points to use as our test set while maintaining a strict 90% - 10% training-test split ratio.

4.4 RNN and CNN

For our final experiment, we decided to change our model design and use time-series ordered word vectors as fea-

tures and an RNN as our learning method. Our motivation was that bag-of-words features do not preserve the order of words, but the structure inherent to how abstracts are written could help us distinguish categories (e.g. starting with introductory information and ending with specific contributions of paper). We hypothesized that LSTMs would be helpful in capturing this order and the relationship between the words, potentially performing better than approaches that use the simpler BoW features.

To prepare the features, we preprocessed the data by converting all letters to lowercase and removing stopwords. Then we converted each abstract into a word embedding, where words are represented as vectors in a high dimensional space in which the closer two vectors are, the more similar the words are in meaning. We used GloVe's pretrained word vectors to create an embedding matrix, where each word is associated with a unique index that corresponds to its respective GloVe vector [17]. Initially, we were not familiar with Keras but learned to use it to create a Sequential model, add an embedding layer, an LSTM layer with 100 memory units, and a Dense output layer to make predictions. Finally, we used categorical cross entropy loss as our loss function and used RMSprop optimization while learning for 50 epochs with a batch size of 100.

For the CNN approach, we also used GloVe to create word embeddings. The next few layers perform convolutions over the word vectors with a total of 128 filters of size 5 and max pooling of size 5 (three convolutional and pooling layer pairs total). There is a final fully connected layer with dropout and softmax activation to make predictions, and the architecture is based on Yoon Kim's paper on CNNs for sentence classification [21]. Here we also used categorical cross entropy loss and RMSprop optimization, learning for 20 epochs with a batch size of 50.

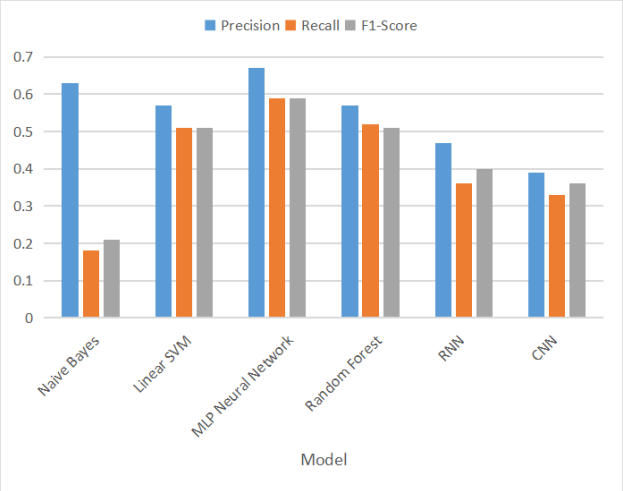
5 Results and Analysis

5.1 Training Performance

Models	Precision	Recall	f1-Score
Naive Bayes	0.91	0.40	0.47
Linear SVCs	1.00	1.00	1.00
MLP Neural Network	1.00	1.00	1.00
Random Forests	1.00	1.00	1.00
RNNs	0.97	0.97	0.97
CNNs	0.95	0.94	0.95

5.2 Test Performance

Models	Precision	Recall	f1-Score
Naive Bayes	0.63	0.18	0.21
Linear SVCs	0.57	0.51	0.51
MLP Neural Network	0.67	0.59	0.59
Random Forests	0.57	0.52	0.51
RNNs	0.47	0.36	0.40
CNNs	0.39	0.33	0.36



5.3 Analysis

For all of our models, we used a randomized 90%-10% training-test data split over all of our 12,800 abstracts total, except for Naive Bayes, where we used a 70%-30% split.

5.3.1 Naive Bayes

The Naive Bayes Classifier achieved roughly 63% precision and 18% recall, for a f1 score of 21% on the test

data. On the training data, it achieved roughly 91% precision and 40% recall, for a f1 score of 47% on the test data. This shows that despite the high precision on the training data, the recall was not very good, but as we want to ensure that the categories offered are relatively good suggestions for the abstract labels, we value recall more. Thus, our Naive Bayes baseline did not perform very well in the areas we are interested in. As we mentioned earlier, the texts we are classifying are much more domain-specific, so the assumption of conditional independence required by this algorithm is not very well founded, explaining its poor performance compared to our other, more targeted, methods. However, despite all this, the Naive Bayes algorithm is very cheap to train and use, so while it may be better suited to more general texts, this model isn't a bad choice if quick, cheap journal classifications need to be made.

5.3.2 Linear Support Vector Machine Classifier

The Linear Support Vector Machine Classifier achieved roughly 57% precision and 51% recall, for a f1 score of 51%. The training data's precision and recall yielded almost 1.0 for each. We were initially worried that this was a strong sign of overfitting. However, after additional testing, we realized that the test data's precision and recall decreased proportionally with the training data's precision and recall, so we were in fact increasing performance overall. The best test recall and precision that we could get was corresponded to having an almost perfect recall and precision during training. Moreover, Linear SVM classifiers are also relatively simple to train, so these models offer a good balance between performance and training time.

5.3.3 Multi-Layer Perceptron Neural Network

The Multi-Layer Perceptron Neural Network achieved roughly 67% precision and 59% recall, for a f1 score of 59%. On the training data, we also achieved nearly perfect precision and recall due to the complex nature of the model allowing for easier overfitting. In addition, while great accuracy might be achievable, we must also consider training time, as this model took much longer to train and use than both the Naive Bayes and Linear Support Vector Machine classifiers combined. If we reduced the number of hidden neurons used in the intermediate layers from 100, then we could reduce the total training time of this model, but it would also result in a decrease in performance.

5.3.4 Random Forests

Random Forests generally perform well when there are more estimator trees. However, because of limited computational resources, the maximum number of estimator trees that we could use was about 1,000. With 1,000 estimator trees, we could achieve about 57% precision and 52% recall on the test data. Similar to Linear SVM classifiers, the best test recall and precision that we could get was by having almost perfect recall and precision during training.

5.3.5 RNN and CNN

We initially hypothesized that RNNs with time-series ordered word vector features would perform better than our baseline Naive Bayes with BoW features. Our experiment showed that this was correct, but there was a tradeoff between precision and recall. For example, we achieved a precision of 66.38% after 12 epochs but recall was only 2.24%, while we achieved a recall of 35.20% after 40 epochs but precision was 47.50%. This makes sense because precision is the percentage of classifications we make correctly across all categories, while recall is the average over all categories of the percentage of correct classifications made for a single category. We want to focus on increasing recall to make sure we're catching as many papers as possible for each category, so the RNN approach performed quite well compared to the Naive Bayes baseline in this case.

The final values for the RNN approach we report are after 50 epochs, when we achieved about 47% precision and 36% recall. As training accuracy increased, we were concerned about overfitting the data, and testing accuracy flatlined after about 30. For abstracts in particular, using features where order is meaningful can be effective in training. However, we also believe that we could have increased testing accuracy further with more resources, e.g. GPUs, time, and data, to tune hyperparameters more rigorously. We could also have experimented with different design choices, like adding a dropout layer to prevent

overfitting or an sentence-level attention layer. The final values for the CNN approach we report are after 50 epcohs, when we achieved 29% precision and 25% recall. We also think that we could have improved performance by testing out deeper models or incorporating a dropout layer, given more resources.

For CNNs, the training was much faster than for RNNs but the overall accuracy was not as high. We believe this is because CNNs extract the most informative ngrams for activation and thus certain key phrases signal that an abstract belongs to a certain category. Both recall and precision would benefit from this and thus climbed slowly higher together. On the other hand, RNNs being sequential give much more weight to the order of words than their presence. Thus, precision hit its peak when recall was still quite low since certain orderings of words could correspond to categories even though each category may not yet have a representative ordering. Later on, as recall increased, precision fell, which also makes sense since abstracts with similar structures would be classified correctly for an individual category, but it would be quite difficult to capture the different structures of all the abstracts in a given category and maintain a high precision across all 127 categories.

In the end, though, RNNs performed better than CNNs because key phrases may not be enough to differentiate between categories. In both cases, we performed better than our baseline at the very least. Since our project's goal is to increase recall, we conclude RNNs were a better fit for our task.

6 Conclusion and Future Work

6.1 Conclusion

We were able to achieve more than 50% recall and precision for most of our algorithms. We found that using a naive bag-of-words representation of our data worked well when paired with simple models, but time-series ordered vectors with RNNs could also achieve high precision and comparable recall. With greater resources to tune hyperparameters and test other designs like including a dropout layer, the deeper models could be improved to perform on par with the simpler models but definitely are more intensive training-wise. The MLP Neural Network had the best performance, but we would recommend RNNs to achieve either high precision or recall, since there is a tradeoff that makes it difficult to perform well for both. Even though our models as of now cannot outperform human classification, it will provide more than a rough guideline regarding classifying a journal and its abstract into academic categories. As we stated before, more than 11,500 papers were published in November 2017, and this number will only continue to grow. We hope our paper can eventually lead to additional help in classifying large volumes of research papers and encourage interdisciplinary work.

6.2 Future Work

6.2.1 Dataset Sampling

We have been scraping 100 abstracts for each label. However, this may produce an imbalanced data set, as we cannot accurately assume that every label category is equally likely to show up for a particular paper to have, as certain topics, like AI or Machine Learning, might be more popular and thus appear in a greater proportion of journals. In the future, we could resolve this issue by drawing the number of abstracts to be proportional to the number of journals present in the parent population for each label. This will improve our classification accuracy by providing a more realistic distribution of papers and the corresponding distribution of categories and subcategories, as well as making the majority algorithm baseline more useful. However, the current arXiv API makes it difficult to sample journals randomly, as we obtain them using a query off of a few specific search terms. We could randomly sample by the most recent papers submitted, but this could bias our training data towards popular subjects that produce disproportionately large amounts of papers, which would make it difficult to classify papers in less-popular, more niche fields.

6.2.2 Overfitting

Deeper, more complex models are sensitive to hyperparameter choices, like hidden neuron or decision tree counts, but tuning them for robustness and performance takes extensive cross-validation. Even though our training accuracy for all the models was decently high, ob-

taining better test-time performance requires even more cross-validation. However, this requires extensive training time and additional computational resources, which we currently do not have available for us to use.

7 Acknowledgement

We would like to express special thanks to our mentor Hansohl for giving helpful advice for this research and TA Amani for helping us learn how to use Tensorflow and Keras. We also want to thank all the teaching staffs; this project would not have been possible without all of their insights and help.

References

[1] 1.17. Neural Network Models (Supervised) - Scikit-Learn 0.19.1 Documentation, scikit-learn.org/stable/modules/neural_networks_supervised.html.

[2] 4.2. Feature extraction - scikit-Learn 0.19.1 documentation, scikit-learn.org/stable/modules/feature_extraction.html.

[3] Bookstein, A. & Swanson, Don R. (1974). Probabilistic models for automatic indexing. Journal of the American Society of Information Science, 25, 312-318.

[4] Cao, J. and Chen, J. (2013). An Improved Web Text Classification Algorithm Based on SVM-KNN. Applied Mechanics and Materials, 278-280, pp.1305-1308.

[5] Dohare, Shibhansh, et al. Text Summarization Using Abstract Meaning Representation. [1706.01678] Text Summarization Using Abstract Meaning Representation, 17 July 2017, arxiv.org/abs/1706.01678.

[6] Gao, M., Li, F., Ding, Z. and Xiao, W. (2014). Coupling Sentiment Dictionary and SVM Classification for Text Orientation Analysis. Advanced Materials Research, 989-994, pp.2444-2449.

[7] Moorthy, K. and Mohamad, M. (2011). Random forest for gene selection and microarray data classification. Bioinformation, 7(3), pp.142-146.

[8] Pranckevicius, T. and Marcinkevicius, V. (2017). Comparison of Naive Bayes, Random Forest, Decision Tree, Support Vector Machines, and Logistic Regression Classifiers for Text Reviews Classification. Baltic Journal of Modern Computing, 5(2).

[9] Salimi, F. and Zarei, A. (2015). Presentation a Neural Network with GradualClustering Performance for Text Classification. International Journal of Computer Applications, 129(17), pp.1-4.

[10] Salmon Run. Sentence Genre Classification Using Scikit-Learn Linear SVC, sujitpal.blogspot.com/2013/08/-sentence-genre-classification-using.html.

[11] Text Genre Classification with Genre-Revealing and Subject-Revealing Features. ACM Digital Library, ACM, dl.acm.org/citation.cfm?id=564403.

[12] Silva, J., Coheur, L., Mendes, A.C., and Wichert, A. (2011). "From symbolic to sub-symbolic information in question classification." Artificial Intelligence Review, 35(2), pp.137-154.

[13] Wang, S. and Manning, C. (2012). Baselines and bigrams: Simple, good sentiment and topic classification. Proceedings of the 50th Annual Meeting of the Association for Computational Linguistics: Short Papers, Volume 2, pp. 90-94.

[14] Nakagawa, T., Inui, K., and Kurohashi, S. (2010). Dependency tree-based sentiment classification using CRFs with hidden variables. Human Language Technologies: The 2010 Annual Conference of the North American Chapter of the Associationfor Computational Linguistics, pp. 786-794.

[15] LeCun, Y., Bottou, L., Bengio, Y., and Haffner, P. (1998). Gradient-based learning applied to document recognition. Proceedings of the IEEE, 86(11), pp. 2278-2324.

[16] Elman, J. (1990). Finding structure in time. Cognitive Science, 14(2), pp. 179-211.

[17] Pennington, J., Socher, R., and Manning, C. (2014). GloVe: Global Vectors for Word representation, https://nlp.stanford.edu/projects/glove/.

[18] Yin, W., Kann, K., Yu, M., and Zchutze, H. (2017). Comparative Study of CNN and RNN for Natural Language Processing, arXiv:1702.01923.

[19] Dauphin, Y., Fan, A., Auli, M., and Grangier, D. (2016). Language modeling with gated convolutional networks, arXiv:1612.08083.

[20] Tang, D., Qin, B., and Liu, T. (2015). Document modeling with gated recurrent neural network for sentiment classification. In Proceedings of EMNLP, pp. 1422-1432.

[21] Kim, Yoon. (2014). Convolutional Neural Networks for Sentence Classification, arXiv:1408.5882.