Vector Representations for Multi-label Text Classification with Neural Networks

Trevor McInroe

Northwestern University

Vector Representations for Multi-label Text Classification with Neural Networks

Introduction and Problem Statement

In this work, we explore various methods for reference term vector creation for a given corpus. Specifically, we experiment with manually-selected term count vectors, TFIDF vectors, and vectors produced by doc2vec. We test their efficacy in a dense neural network trained on a multi-label classification problem. The goal of this study is to develop intuition around the creation of various vector representation types as well as their capabilities in a deep learning setting.

Background and Related Works

Text data has been proven to be useful across many classification tasks in various domains (Abikoye et al., 2018; Ahmed et al., 2010; Jeske & Liu, 2007). However, text data is inherently unstructured and, in its raw form, unfit to be run through any sort of machine learning or deep learning model. To apply mathematical techniques to text data, we must first transform the text into some numerical representation. It is not immediately obvious how to best perform this transformation, as the quality of the representation strongly depends on both the task, as well as the model. There are a variety of methods for representing text data in a numerical form. In this study, we have tested three different representations on one model for one task. The three representations are term count vectors, term frequency-inverse document frequency (TFIDF) vectors, and vectors produced by doc2vec. The model is a dense neural network and the task is multi-label classification.

The three vector representations differ in terms of their *representational capacity.* That is, the richness of information from the raw data that they can capture. Term count vectors simply convert a given document into a vector of counts using a given vocabulary, where the vocabulary is usually manually selected after inspection of the corpus. For consistency, each term in the vocabulary is given a specific index in the vector. This approach represents one of the simplest vector representations of text data. TFIDF was introduced by Jones (1972) and is an extension of the count vector. It assigns a normalized score to words on a per-term, per-document level, depending on both the frequency of the given term in the given document, as well as the inverse of the frequency of documents that contain the given term:

$$w_{t,d} = \mathcal{F}(t,d)\mathcal{Z}(t,\mathcal{D}) \tag{1}$$

$$\mathcal{F}(t,d) = \frac{\sum_{i=1}^{T_d} I(d_i = t)}{T_d} \tag{2}$$

$$\mathcal{Z}(t,\mathcal{D}) = log\frac{|\mathcal{D}|}{\sum_{i=1}^{\mathcal{D}} I(t \in \mathcal{D}_i)} \tag{3}$$

where d_i is the *ith* term in document d, t is the term of concern, T_d is the number of terms in document d, $|\mathcal{D}|$ is the total number of documents in the corpus $\mathcal{D}$, I is an indicator function that returns a one if the inner logic is true (otherwise a zero), and log is the natural logarithm. Equation 2 computes the count of term t in document d normalized by the total number of terms in d, or *term frequency.* Equation 3 is the natural log of the total number of documents in $\mathcal{D}$ divided by the total number of documents that t appears in, or *inverse document frequency.* Equation 1 is the computed weight for term t in document d. Intuitively, we can see why such representations create so-called *sparse* vectors, as a given term not being present in a given document would amount to its weight being zero.

The first two types of representations mentioned above are simple and, as such, have drawbacks. For one, they cause documents to lose the order of terms during the transformation. Perhaps more critically, the terms lose their context, or how they interact with nearby terms and the document as a whole. Such interactions between input variables cannot be captured in certain model types. For example, a model that is linear in its parameters, such as the popular Naïve Bayes model, cannot examine these interactions in their input data if the data itself does not capture the interaction. The mathematical structure of linear models assumes independence between the input variables and therefore does not attempt to estimate any form of dependence (Greene, 2018).

An additional limitation of simpler vector representations is the lack of a notion of distance between terms and phrases. For example, a native English speaker would know that words such as “truck”, “sedan”, and “bicycle” are related in that they are all vehicles. That is, the context of “vehicle” binds these three words together more closely than it would the two words “truck” and “cup”. This notion of contextual closeness simply cannot be represented in the previous two methods.

To alleviate the aforementioned issues, we can either turn to a model architecture designed to capture nonlinear and contextual effects (such as a neural network), use a more sophisticated representation, or both. The need for vector representations that are dense and capture the contextual information of words was the motivation behind word2vec from Mikolov et al. (2013) and, later, doc2vec from Le and Mikolov (2014). Put simply, doc2vec is an unsupervised learning algorithm that maps text data to a fixed-length vector representation with the goal of capturing semantic relationships within the text data.

Measuring the quality of the vector representations of text data is a well-studied problem. Researchers may turn to various semantic or syntactic metrics, such as rank-based similarity, cosine similarity, or others (Mikolov et al. 2013, Santus et al. 2018). In this study, we are applying, and not inventing, vector representations. As such, our concern is with the performance of a given model with a given representation. Our task is to take, as input, a vectorized representation of the corpus and to output a classification for each document. This task is complicated by the fact that the classes for our documents, for our task, are not mutually exclusive, and a single document can be described with a set of classes. This makes our problem a multi-label classification endeavor.

Data Overview

The data used in this exercise is the class corpus procured for MSDS-453 Summer 2020 quarter. It contains 64 open-text movie review utterances from both professional and non-professional movie critics. Each document is around 500 characters in length. To create our ground truth labels, we have used genre classification from the Internet

Movie Database (IMDB). The full set of classes is: {Action, Adventure, Comedy, Sci-Fi, Drama, Thriller, Family, Animation}[1]. This ultimately means that our ground truth label for each document is a vector of length 8 of binary variables, $y = (y_1, ..., y_8) \in \{0, 1\}^8$.

Document Cleaning

Before the creation of any vectorized forms, all documents have been tokenized, have stop words and punctuation removed, have been converted to lower-case, and lemmatized using NLTK's WordNetLemmatizer[2].

Vocabulary/Vector Creation

The term count vectors have been created by manually selecting five words from each document. To select these words, we have used the following subjective rules:

- The term must appear more than once in the document.
- The term cannot be a pronoun.
- The term should not be overly-present across every document as to introduce noise (e.g., "movie" or "film").

This process of selecting terms has amounted to a vocabulary of 234 unique terms and a vector of length 234. The TFIDF vectors were created by using every unique term in the corpus as a whole. This has resulted in vectors of length 6,852. The doc2vec vectors were specified to be of length 500.

Research Design and Methods

Our task for this exercise is multi-label classification. Specifically, given a vectorized representation of our movie review corpus, how accurately can we classify the genres of the movies discussed in the reviews? To test this question, we have taken the

[1] For a full list of document-label pairs, see https://github.com/trevormcinroe/msds453/week3/data

[2] http://www.nltk.org/_modules/nltk/stem/wordnet.html#WordNetLemmatizer

three representations reviewed in the previous sections and trained a classification model for each. The following sections outline the model of choice, the metrics for comparison, and additional implementation details specific to the doc2vec vectors and the neural networks.

Model

To overcome some of the issues mentioned in the preceding sections, we have implemented a model capable of capturing both nonlinear relationships and interaction effects among the input data. Specifically, we have used a dense neural network with a single hidden layer. For the nonlinearities, we have chosen the ReLU activation function between the hidden layer and the output layer, and the sigmoid activation function as the final nonlinearity directly after the network output. Contrary to single-label classification problems, a multi-label problem cannot use the popular softmax activation function. This is because the softmax function "compresses" the vector of outputs for each example in the input data into a probability distribution (i.e., sums to one). This makes it mathematically incongruent with any multi-label classification task with more than two classes. In contrast, the sigmoid function computes its transformation for each entry in the output independently, which produces a probability of the example belonging to each possible class.

Performance Metrics

Hamming loss, as specified by Sorower (2010), is a popular metric for measuring the performance of a multi-label classification model. For this exercise, we have used the unweighted version of the hamming loss, which assumes that each class is equally important for our classification task:

$$HL = \frac{1}{n|C|}\sum_{i=1}^{n}\sum_{j=1}^{|C|}[I(C_j \in \hat{y}_i \wedge C_j \notin y_i) + I(C_j \notin \hat{y}_i \wedge C_j \in y_i)]$$

where n is the number of examples, C_j is the jth class and $|C|$ is the number of classes, $\hat{y}$ is the binary prediction vector, y is the binary ground-truth vector, $\wedge$ is logic-notation for *and*, and I is an indicator function that returns a one if the inner-logic is true (otherwise a zero). The leftmost indicator function produces a one when the jth class is wrongfully present in the prediction, and the rightmost indicator function produces a one if the jth class is wrongfully not in the prediction. It is clear, then, that the hamming loss penalizes both types of prediction error for every class for every example. When comparing models, a lower hamming loss indicates superior performance and a hamming loss of zero indicates perfect performance.

Machine learning models (and especially deep learning models) generally take a considerable amount of compute to train due, in part, to their large number of parameters and iterative learning methods. As such, we argue that model complexity is an important metric. As an implementer of machine learning methods, we should strive for *parsimony*, which means simultaneously attempting to minimize model complexity while maximizing the performance of the model on the task at hand. While various constructs from theoretical computer science such as *Big-O Notation* from Landau (1974) and the *RAM Model of Computation* from Skiena (2008) outline frameworks for estimating the complexity of an algorithm, it is not so straightforward to estimate complexity for machine learning models. Much of the compute performance during the training of said models depends on the hardware architecture of the computer being used. Instead, for this study, we have chosen to copy Mikolov et al. (2013) and use the number of learnable parameters as an analogy for complexity.

Implementation Details

We note that we do not make use of any pretrained models for this exercise. Everything is trained from randomized initial weights, including the doc2vec embeddings themselves. Based on empirically driven advice from work by Lau and Baldwin (2016), we have set our doc2vec hyperparameters as follows:

- Window = 3
- Epochs = 500
- MinCount = 1

In addition, our embeddings are learned in a standalone method. That is, the learning of the embeddings is independent of the learning of the genre-classification model that uses the embeddings as input.

As mentioned in the preceding sections, we have used a dense neural network with a single hidden layer that contains the ReLU and sigmoid nonlinearities. The network itself has 128 nodes in the hidden layer. To optimize the network, we have used the popular Adam optimizer introduced by Kingma and Ba (2014). We have implemented the default hyperparameters suggested in the paper, as follows:

- $\alpha = 0.01$
- $\beta_1 = 0.99$
- $\beta_2 = 0.999$
- $\epsilon = 0.00001$

Experiments

Due to our relatively-small corpus, we have implemented five-fold cross-validation during model training. Our folds have been pre-computed before the experiments to ensure consistency across runs. In addition, we note that the training trajectory of neural networks is sensitive to the random initialization of their weights. To overcome this, we have run each network, for each fold, for each representation type, five times, resetting the network's weights each time. Within each of the five runs, we have summarized performance as the average hamming loss across the five folds. More explicitly:

1. Project text data into a given vector representation type.
2. Train the neural network for 50 epochs. For each epoch:
   (a) Record hamming loss on the training data.
   (b) Record hamming loss on the testing data.

3. Repeat step 2 (while resetting the network) for the given vector representation type, resetting training/testing data determined by the folds, once for each fold.

4. Record the average performance for the given vector representation type across the five folds.

5. Repeat steps 2 through 4, five times for the given vector representation type, setting a new random seed for the network weights each time.

6. Repeat steps 1 through 5, for each vector representation.

In our graphs, we have reported on this averaged performance for each of the five randomly initialized networks for each representation type. Specifically, we have reported the average hamming loss, as well as the minimum and maximum performance for each epoch, to show the stability of the vector representation type for both training and testing datasets.

Result Analysis and Interpretation

Observing Table 1, we note the stark difference in vector sparsity, as measured by the proportion of weights in the representations that are zero. The doc2vec vectors are indeed dense, having only a sparsity of 1.6%, while the term count and TFIDF vectors have a sparsity of 88.3% and 96.7%, respectively. In addition, we note that the TFIDF network contains over ten times more learnable parameters than the next largest network (doc2vec), due to the relatively large size of the TFIDF vectors.

Observing the learning performance in Figure 1 on the training datasets for all vector representation types shows the strong representational capacity of neural networks. Despite both the term count vectors and the TFIDF vectors being extremely sparse, the networks were able to achieve excellent performance on the training set. However, this performance did not generalize well onto the test dataset for all representation types. We note that the term count vector network quickly overfit on the training set, shown by the worsening performance on the test set after epoch 20. In addition, we note that both the TFIDF and doc2vec networks continued to show improving performance on the test dataset, indicating that they could have been underfit at 50 epochs. This suggests that performance could have been improved further with more training. Ultimately, the doc2vec representations showed the best performance both in terms of the hamming loss on the test dataset, as well as model complexity. The network trained on the doc2vec vectors showed better performance than the next best-performing representation type while containing 92.6% less learnable parameters than the next best-performing network.

Conclusion

In this study, we examine the creation process of three different vector representations of text data: term counts, TFIDF, and doc2vec. We observe how term counts and TFIDF vectors create sparse representations while doc2vec vectors are dense. Finally, we prove that, for the given task of multi-label classification, doc2vec vectors are the superior choice in terms of both model performance and model complexity.

Future Work

Several open questions remain. First, at what point do the networks trained on the TFIDF and doc2vec vectors begin to overfit? Second, is the small size of the corpus stopping the doc2vec representations from performing as well as they possibly could? Third, are there any automated methods for creating the term count vectors that would allow them to perform better than they did in this experiment?

References

Abikoye, O. C., Omokanye, S. O. & Aro, T. O. (2018). Text classification using data mining techniques: A review. *Computing and Information Systems*, *22*(2), 1. https://link-gale-com.turing.library.northwestern.edu/apps/doc/A541103802/AONE?u=northwestern&sid=AONE&xid=632f6965

Ahmed, M. S., Khan, L. & Rajeswari, M. (2010). Using correlation based subspace clustering for multi-label text data classification, In *2010 22nd IEEE International Conference on Tools with Artificial Intelligence. Vol. 2*, IEEE. https://doi.org/10.1109/ICTAI.2010.115

Greene, W. H. (2018). *Econometric analysis* (8th ed.). New York, NY, Pearson.

Jeske, D. R. & Liu, R. Y. (2007). Mining and tracking massive text data: Classification, construction of tracking statistics, and inference under misclassification. *Technometrics*, *49*(2), 116–128. http://www.tandfonline.com/doi/abs/10.1198/004017006000000471

Jones, K. (1972). A statistical interpretation of term specificity and its application in retrieval. *Journal of Documentation.* http://search.proquest.com/docview/57585760/

Kingma, D. P. & Ba, J. (2014). Adam: A method for stochastic optimization, arXiv:1412.6980 [cs.LG].

Landau, E. (1974). *Handbuch der lehre von der verteilung der primzahlen.* (3d ed.). New York, Chelsea Pub. Co.

Lau, J. H. & Baldwin, T. (2016). An empirical evaluation of doc2vec with practical insights into document embedding generation, arXiv:1607.05368 [cs.CL].

Le, Q. V. & Mikolov, T. (2014). Distributed representations of sentences and documents, arXiv:1405.4053 [cs.CL].

Mikolov, T., Chen, K., Corrado, G. & Dean, J. (2013). Efficient estimation of word representations in vector space, arXiv:1301.3781 [cs.CL].

Santus, E., Wang, H., Chersoni, E. & Zhang, Y. (2018). A rank-based similarity metric for word embeddings, arXiv:1805.01923 [cs.CL].

Skiena, S. S. (2008). *The algorithm design manual* (2nd ed.). London, Springer London.

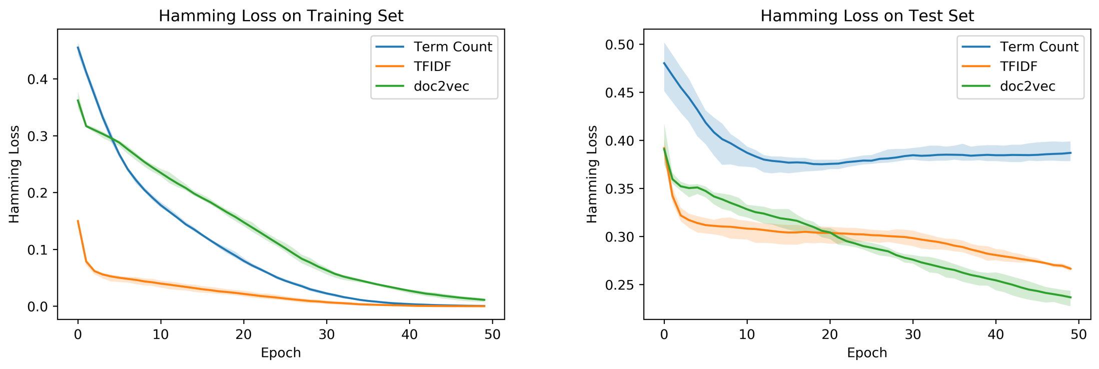
Sorower, M. S. (2010). *A literature survey on algorithms for multi-label learning* (tech. rep. No. 18). Oregon State University. Corvallis. https://pdfs.semanticscholar.org/6b56/91db1e3a79af5e3c136d2dd322016a687a0b.pdf

Table 1

Number of trainable parameters and sparsity related to the vector representation types

Vector Type	Trainable Parameters	Sparsity
Term count	31,112	88.3%
TFIDF	878,216	96.7%
doc2vec	65,160	1.6%

Figure 1. Hamming loss of networks on training data (*left*) and testing data (*right*)


Hamming Loss on Training Set
Term Count
TFIDF
doc2vec
Hamming Loss
Epoch
Hamming Loss on Test Set
Term Count
TFIDF
doc2vec
Hamming Loss
Epoch