Classifying Websites using Word Vectors and Other Techniques

Alejandro Robles[1] Johanna Thiemich[2]

***Abstract*— In this paper, we explore the effectiveness of different classification techniques in classifying websites into four categories: Sports, Games & Toys, Travel, and Food & Drink. The classifiers we tried were Support Vector Machine (SVM) and Convolutional Neural Network (CNN) with two different feature extraction methods term frequency-inverse document frequency (TF-IDF) and Glove word embeddings. The SVM with TF-IDF as input produced very good results with an accuracy of 91% . The more complex CNN with Glove embeddings as input produces 92% accuracy. The worst performing was the SVM with Glove embeddings with accuracy of 89%.**

All the code can be found on github.com[1].

I. INTRODUCTION

While working as a data scientist for a marketing & advertising company, one of the first requested tasks is finding new websites to display appropriate advertisements. With millions of websites online, it is not humanly possible to visit each website, read the content, and classify websites. This allows companies to display their advertisements on websites that are visited by people most likely to be interested in their products. Classifying websites based on their content is not a trivial task. Natural language processing must be used to process website data, which is often unstructured data, to be able to build models from it. Techniques such as words vectors provides an intuitive way to structure data.

The objective of our project is to not only increase the accuracy of classifying websites but also generate meaningful embeddings for these websites. In short, we want to extend the results for word vectors generated by models such as skip-gram to websites. Just like similar words are mapped close together, we are hoping that our url embeddings will also map similar sites close in distance. If this is true, then downstream machine learning applications such as clustering or classifying sites will be greatly facilitated.

To measure success we will classify sites into four categories using a common classifier such as linear support vector machines. We will test how different feature extraction methods affect accuracy. As a base, we will use TF-IDF to generate features for the sites. Then we will use GloVe word embeddings to see if this increases the accuracy and check how useful the embeddings turn out.

II. WORD VECTORS

A. *Skip-gram*

This uses the idea that the context of the words defines the word. This uses neural networks to predict a word given a sequence of words. This surprisingly has linear relationships.

[1]https://github.com/johannathiemich/WebsiteClassification

For example, the operation vec(”Madrid”) - vec(”Spain”) + vec(”France”) is closest to the vec(”Paris”) than any other word vector representation. It tries to maximize the average log probability $\frac{1}{T}\sum\sum logp(w_{t+j}|w_t)$ where $p(w_0|w_I) = \frac{exp(v_{w0}^T v_{wI})}{\sum exp(v_w^T v_{wI})}$ where v are the input and output vector representations of w.

B. *Hierarchical Softmax*

The issue of the Skip-gram is that it computationally expensive so Hierarchical Softmax is good alternative for calculating conditional probabilities. Here, they use a tree structure called Huffman tree to efficiently find common words. This tries to approximately maximize the average log probability.

$$p(w|w_1) = \Pi\sigma([\![n(w,j+1) = ch(n(w,j))]\!] * v_n^T v_{wI}) \quad (1)$$

C. *Noise Contrastive Estimation (NCE)*

The intuition is that a good model should be able to differentiate real words from noise using a classifier such as logistic regression. This also tends to approximately maximize the average log probability but is mainly concerned with learning quality vector representations. Here $P_n(w)$ is a noise distribution where they found that a uniform distribution raised to the 0.75 power outperforms usual uniform distribution.

$$log\sigma(v_{w0}^T v_{wI}) + \sum \mathbb{E}_{wi} \sim P_n(w)[log\sigma(-v_{wi}^T v_{wI})] \quad (2)$$

III. PARAGRAPH OR DOCUMENT VECTORS

A. *Bag of Words*

Bag of words is simple approach for representing documents in a fixed-length vector. It has the following structure: $\{word_1 : frequency_1, word_2 : frequency_2, ..., word_n : frequency_n\}$. While it is easy to train, it has multiple disadvantages. This does not capture any semantics and quite sparse.

B. *TF-IDF*

Term frequency-inverse document frequency is a transformation method of text data to a numeric matrix that reflects the importance of a term to a website in the corpus.

Let t = a term (word), d = a document (website), and D = the corpus (the collection of websites). Let TF(t, f) be a function that counts the number of times a term appears in a website (Term Frequency) and DF(t, D) be a function that counts the number of websites that contains the term t. Then TF-IDF is the product of TF and IDF, such that

$$IDF(t, D) = log \frac{|D|}{DF(t, d) + 1}$$

$$TFIDF(t, d, D) = TF(t, d) \cdot DF(t, D)$$

where $|D|$ is the total number of documents in the corpus. Notice that a smoothing term is applied to avoid dividing by zero. TF-IDF provides a way to emphasize important words and give less weight to terms that appear in many websites which probably will not have much importance like common words such as "the".

C. Paragraph Vectors

Very similar to the skip-gram approach for word vectors. The intuition is that this model provided quality vector representations for words given a supervised task of predicting the next word in the sequence. For paragraph vectors they attempt to do the same but just adding a paragraph id represented as a matrix while jointly trying to predict words in that paragraph.

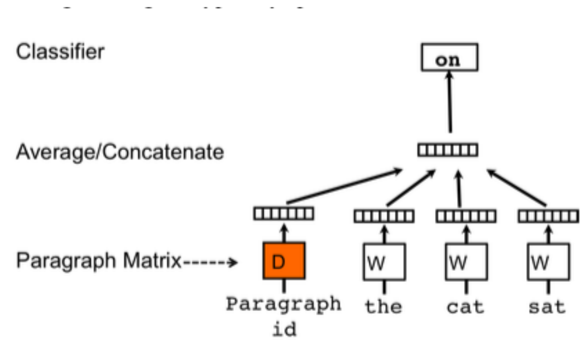


Fig. 1: Paragraph Vector

D. Concatenated Power Mean Word Embeddings

The idea is to generalize averaging word vectors to represent documents. The problem with earlier attempts is that naively averaging word vectors causes information loss about the ordering of the words. This new method attempts to concatenate different averaged word vectors to capture both the ordering a different signals such a semantic information. The power means generalize this function.

Power Means:

$$(\frac{x_1^p + ... + x_n^p}{n})^{1/p} \tag{3}$$

Sentence Vector:

$$s^{(i)} = H_{p1}(W^{(i)}) \oplus ... \oplus H_{pk}(W^{(i)}) \tag{4}$$

Where $H_p(W)$ is a vector whose d components are the power means of the sequences $(w_{1i}, ..., w_{ni})$ and $\oplus$ stands for concatenation of $p_1, ..., p_k$, which are K different power mean values.

E. GloVe - Global Vectors for Word Representation

GloVe vectors are similar to Word2Vec vectors but generally perform better. While Word2Vec mainly takes into account the local neighborhood of a word, GloVe vectors consider global context. That means they look at the whole document when generating a word vector instead of just the surrounding words.

Just as Word2Vec, GloVe is an unsupervised learning algorithm that maps words into vector space where distances between words are related to their similarity. It does so by counting how often two words co-occur and then finding the global optimum of some probability function.

The objective is to choose the word vectors in a way so that their dot product equals the logarithm of the words' probability of co-occurrence.

To get an idea of how GloVe vectors perform on our dataset one can look at figure 2. We mapped the GloVe vectors generated by a pretrained model into 2D space using T-SNE and PCA. It can be seen that the embbedings for the websites have a lot of potential for discriminating between categories. Some of the clusters are kind of spread out but that is probably owed to the dimensionality reduction applied before.

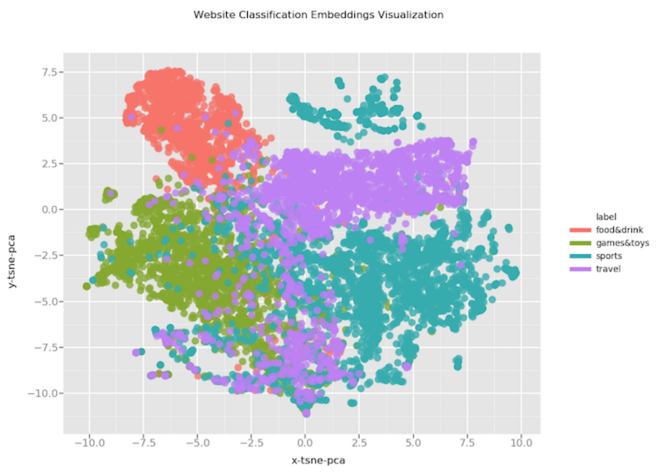


Fig. 2: Website Embeddings

F. Support Vector Machines (SVM)

A Support Vector Machine (or SVM) is a supervised machine learning algorithm that can be used for both regression and classification, although it is more frequently used for classification.

The general idea of SVMs is to find a number of hyperplanes to divide the data into a (given) number of classes. SVMs makes use of Support vectors which are the data points nearest to those hyperplanes. If those points were removed from the dataset, it would influence the positions of the hyperplanes. SVMs use the so called hinge loss as loss function. It is defined as:

$$l(y) = max(0, 1 - t \cdot y)s$$

For linear SVMs (which we are using here) $y = \mathbf{w} \cdot \mathbf{x} + b$ with w and b being the parameters of the hyperplane and $\mathbf{x}$ the instance to classify.

In order to find the hyperplanes that minimize this loss, SVMs make use of the Stochastic Gradient Descent method.

SVMs are very commonly used for text classification. Their ability to reach a high accuracy even on small datasets makes them the perfect model for an approach to classify the websites.

IV. Dataset

This dataset was scraped from 12,874 websites in February of 2019. The dataset is all the text found on each of those websites. After cleaning the data 10,626 distinct websites remained. Cleaning the data included finding non-english text and removing those websites. Furthermore some websites did not contain any text so we also removed these instances from our dataset.

After that we cleaned the text of the remaining websites. This included removing stopwords, transforming all the text to lowercase and removing non-ASCII-characters.

Each of the 10,626 websites belongs to one of the categories "Sports", "Food & Drink", "Games & Toys", "Travel". We chose these four categories since they contain the biggest number of instances and are not similar to each other, i.e. they cannot be confused with each other. The distribution of instances across these four classes can be seen in Figure 8. Even though the data is imbalanced, a classifier always predicting "Sports" will only reach an accuracy of 46 %.

The website with the greatest amount of words contains 44,801 words. On average, each website contains 678 words. The total number of words across all websites sums up to 7,200,956 words.

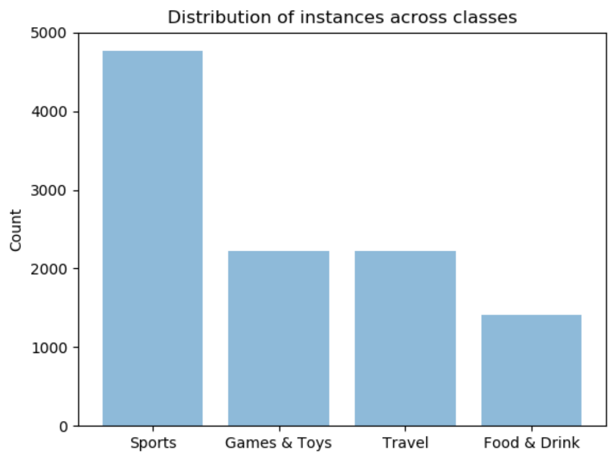


Fig. 3: Distribution of instances among classes

V. Our approaches

For this project we tried 3 different approaches: using TF-IDF and an SVM, GloVe vectors and an SVM, and CNN.

model	accuracy
TF-IDF & SVM	0.91
GloVe & SVM	0.89
GloVe & CNN	0.92

TABLE I: Accuracy results for all of our approaches

The overall accuracy for each approach can be found in table I.

A. TF-IDF and SVM

As a first step we trained a linear Support Vector Machine classifier on our data since linear models usually perform well due to their simplicity. Also SVMs usually still work well even when the amount of data available is not huge. For the feature extraction we first used the TF-IDF representation of the website texts. Our goal was to compare different feature extraction methods in order to find out which one performs the best.

The python library we used to extract the features and derive the model was the `sklearn` library.

The pipeline to obtain the trained model included the following steps. It is also visualized in figure 4

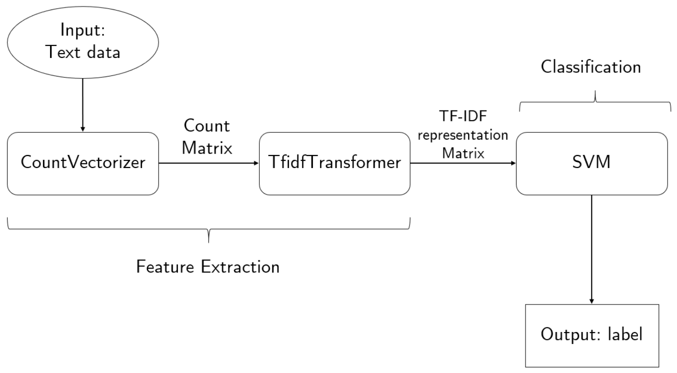


Fig. 4: Workflow of using TF-IDF vectors with an SVM

We split the cleaned dataset into 70 % training and 30 % test. After that we used the `sklearn Pipeline` class to apply the three sequential transformations to the training data. The first step is the `CountVectorizer` which converts our website texts to a matrix of token counts. This count matrix will be transformed to a normalized TF-IDF representation using the `sklearn TfidfTransformer`. The theory behind the general TF-IDF transformation is described in section III-B. And finally the model is derived using Stochastic Gradient Descent (via python's `SGDClassifier` class). Some of the hyperparameters for this model include:

- `loss` function: hinge loss since we want to retrieve an SVM model.
- `max_iter` : the maximum number of passes over the training data is set to 20.
- `tol`: the stopping criterion. The iterations will stop when $loss > previous\_loss - 0.001$.
- `alpha`: Constant that multiplies the regularization term in the calculated gradient; is set to 0.001.

class name	precision
Food & Drink	0.95
Games & Toys	0.88
Sports	0.90
Travel	0.93

TABLE II: Metrics for the TF-IDF and SVM model for each class

class name	precision
Food & Drink	0.88
Games & Toys	0.93
Sports	0.95
Travel	0.78

TABLE III: Metrics for the GloVe vectors and SVM model for each class

- `learning_rate`: the learning rate in the calculated gradient; is set to its default value which is defined as
$$learning\_rate = 1.0/(alpha * (t + t_0))$$
where t_0 is chosen by a heuristic proposed by Leon Bottou.

As it turns out, this very simple linear model actually yields very good results regarding accuracy. On the test data (which we split from the original dataset and has the size of 30 % of the whole dataset) it reaches an accuracy of 90.97% which is about twice as good as a classifier that would always guess "Sports". Since in our dataset the classes are very imbalanced, the precision gives a better insight into the performance of our classifier. It is displayed in table IV for every single class. One can see that we reach high values for precision.

B. *GloVe vectors and SVM*

For our next approach we wanted to test how much better or worse a more complex feature extraction method performs compared to TF-IDF. The method we decided to use was GloVe vectors. They were created using a pretrained model from the Python NLP library Spacy.

This model was trained on the dataset Common Crawl which is a large corpora that provides text scraped from websites. Since we are working with website text classification this model seemed to fit our purpose very well.

The worfklow was the following: After downloading the pretrained model, we load it into the Python script. We only use it for inference by inputting all of our text data and saving the resulting document vectors. These document vectors are then used to train a simple linear SVM which performs the classification based on those vectors. The output of the SVM is the label of the input text. This process can also be seen in figure 5.

Surprisingly, this more complex model performed a little worse than the TF-IDF approach. The accuracy across all classes was 89.11%. The precision for each class label can be seen in table III.

The hyperparameters were the same as the ones we used for the SVM that was trained on the TF-IDF vectors.

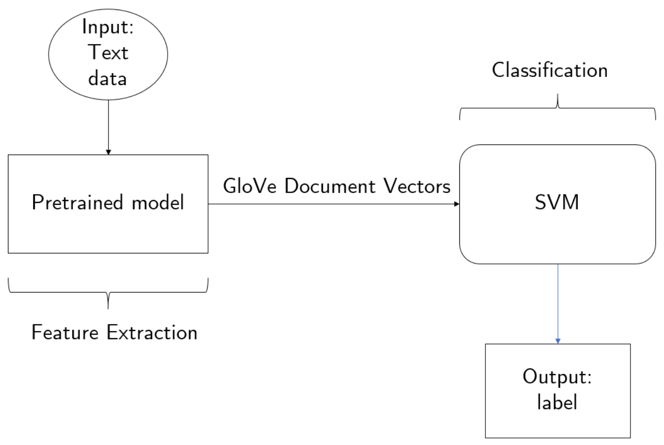


Fig. 5: Workflow for training an SVM with GloVe vectors

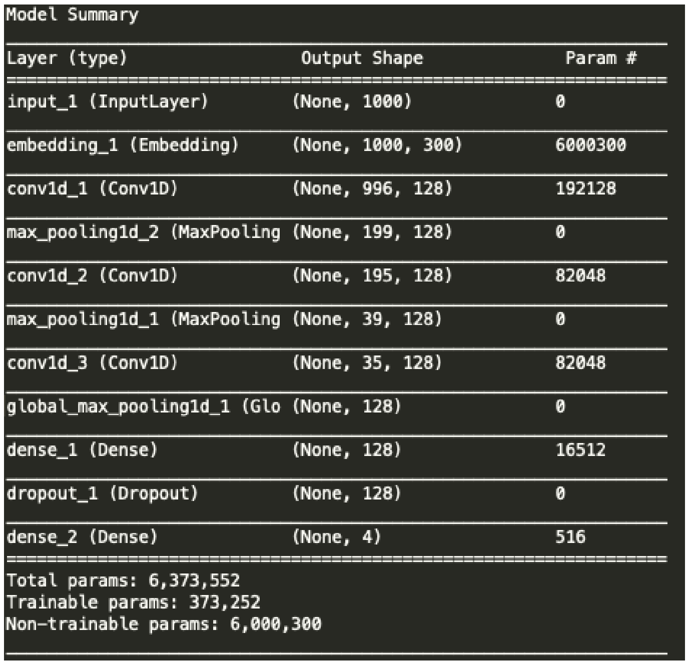
VI. GLOVE VECTORS AND CNN

The intuition was to use a pre-trained model such as Glove word vectors and use them for transfer learning for a classification task as part of a one dimensional convolutional neural network. For text classification, researches have primarily used 1D Convolutional Neural Network, Recurrent Neural Networks, or Hierarchical Attention Networks. We chose a more simplistic 1D convolutional network because there are studies showing that it tends to consistently perform up to par with the more complex RNN architectures. Because text data is sequential data, 1D convolutions are a great fit for this type of data.

In general, CNN are used to extract simple patterns within the first layers but sequentially learn more complex features within deeper layers. Here a 1D convolution architecture should be able to derive interesting features from shorter segments of the text data.

A. *Architecture*

The architecture is summarized in Figure 6. It is composed of an embedding layer that has pretrained glove model for the word vectors. We make sure to freeze them for training because we do not want to erase the learned weights. It is then followed by 1D convolution layers each followed by either a max pooling or a global max pooling layer. We then finish with two dense layers with a dropout layer between them. The final dense layer has dimension (None, 4) with a softmax activation function which tries to predict the probability of belonging in one of the four categories.



```
Model Summary
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_1 (InputLayer)         (None, 1000)              0
_________________________________________________________________
embedding_1 (Embedding)      (None, 1000, 300)         6000300
_________________________________________________________________
conv1d_1 (Conv1D)            (None, 996, 128)          192128
_________________________________________________________________
max_pooling1d_2 (MaxPooling (None, 199, 128)          0
_________________________________________________________________
conv1d_2 (Conv1D)            (None, 195, 128)          82048
_________________________________________________________________
max_pooling1d_1 (MaxPooling (None, 39, 128)           0
_________________________________________________________________
conv1d_3 (Conv1D)            (None, 35, 128)           82048
_________________________________________________________________
global_max_pooling1d_1 (Glo (None, 128)               0
_________________________________________________________________
dense_1 (Dense)              (None, 128)               16512
_________________________________________________________________
dropout_1 (Dropout)          (None, 128)               0
_________________________________________________________________
dense_2 (Dense)              (None, 4)                 516
=================================================================
Total params: 6,373,552
Trainable params: 373,252
Non-trainable params: 6,000,300
_________________________________________________________________
```

Fig. 6: Model Architecture

B. Parameters

- `loss`: Categorical Crossentropy
- `epochs` : 20
- `batch_size`: 512
- `optimizer`: rmsprop
- `learning_rate`: 0.001

C. Layers

1) The Embedding Layer: The Embedding layer can be seen as a dictionary that maps integers words (or indices) to dense vectors.

2) 1D Convolution Layer: This implements a convolution kernel which convolves the input layer over a one dimensional spatial dimension producing a tensor

3) Dropout Layer: Applies dropout to the input layer, which randomly sets some neuron weights to zero. This helps with overfitting.

4) Dense Layer: This is a regular densely-connected neural network layer.

5) MaxPooling1D: Max pooling operation for temporal data.

6) GlobalMaxPooling1D: Global max pooling is the same as max pooling layer but with pool size that equals to the size of the input. For example, if the input of the max pooling layer is 0,1,2,2,5,1,2 then the global max pooling outputs 5.

D. Overfitting

Initially, our model had an overfitting issue. We can see here that the training (**blue line**) diverges from the test (**orange line**) as the epoch steps increase. For this attempt we did not do any hyperparameter tuning so it was not too suprising to see overfitting. This issue was also present for the loss metric.

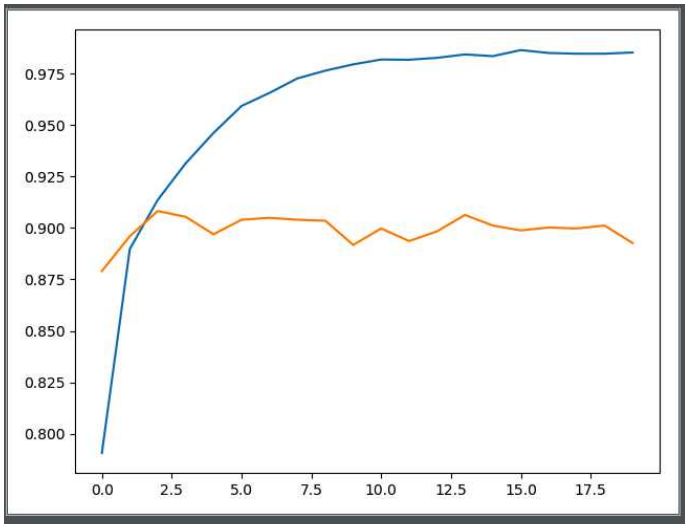


Fig. 7: Accuracy

class name	accuracy
Food & Drink	0.89
Games & Toys	0.90
Sports	**0.95**
Travel	0.88

TABLE IV: Accuracy for the Glove Vectors and CNN model for each class

We tried different approaches such as using batch normalization and adding a dropout layer to reduce overfitting. Ultimately, the dropout layer gave the best result. After some hyperparameter tuning, we settled for a droput rate of 30%. Below we can visualize the improvement.

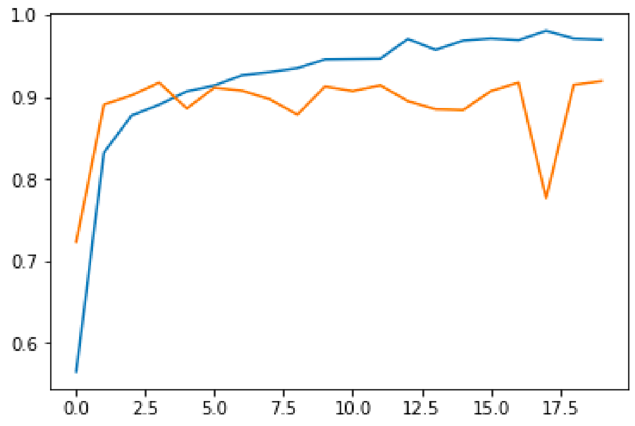


Fig. 8: Accuracy

VII. Conclusion

Overall one can see that the most complex model - the CNN - performed the best. Because the CNN was only marginally better than the simpler model, this might be the model of choice since it is much easier to train and is less likely to overfit.

VIII. Future Work

There are a lot ways how our work can be continued and improved. Having more data can not only help to improve the overfitting problem for the CNN but can also lead to a higher accuracy for the SVM classifiers. The next step would be to clean the data more thoroughly. This includes removing the most and least frequent words in the text and stemming words.

Also the hyperparameters for the CNN model should be fine tuned more in order to improve its performance. Other network architectures such as Recurrent Neural Networks or Hierarchical Attention Networks could lead to better results as well.

Furthermore, it might be a good approach to train our own word embeddings instead of using a pretrained one. By training from scratch on our dataset the model will be more fitted to our specific tasks and potentially produce better results.

Finally, concatenating different word embeddings may help improve model accuracy. This approach can inject complementary information from diverse word embeddings, which helps generalize the representations. It can be viewed as a type of ensemble learning.

References

[1] Quoc V. Le and Tomas Mikolov. Distributed representations of sentences and documents. *CoRR*, abs/1405.4053, 2014.

[2] Andreas Rücklé, Steffen Eger, Maxime Peyrard, and Iryna Gurevych. Concatenated p-mean word embeddings as universal cross-lingual sentence representations. *CoRR*, abs/1803.01400, 2018.

[3] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg Corrado, and Jeffrey Dean. Distributed representations of words and phrases and their compositionality. *CoRR*, abs/1310.4546, 2013.

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

[5] N. Bambrick. Support vector machines: A simple explanation. 2019. https://www.kdnuggets.com/2016/07/support-vector-machines-simple-explanation.html.

[6] 2019. https://en.wikipedia.org/wiki/Hinge_loss.

[7] Jeffrey Pennington, Richard Socher, and Christopher D. Manning. Glove: Global vectors for word representation. In *Empirical Methods in Natural Language Processing (EMNLP)*, pages 1532–1543, 2014.

[8] François Chollet et al. Keras. https://keras.io, 2015.

[9] Franois Chollet. *Deep Learning with Python*. Manning, November 2017.