Meme Classification

Chirag Bhatt
Department of Mathematics
Indian Institute of Technology, Delhi
Delhi 110016, India
mt1180750@maths.iitd.ac.in

Subhalingam D
Department of Mathematics
Indian Institute of Technology, Delhi
Delhi 110016, India
mt1180770@maths.iitd.ac.in

1 Introduction

According to the Oxford English Dictionary, meme refers to an image, video, piece of text, typically humorous in nature, that is copied and spread rapidly by internet users, often with slight variations. In the last few years the popularity of memes have increased exponentially. Although it started with the intent of creating humour, they are also used by some people to spread their propaganda and manipulate people. Thus it becomes essential to identify the memes circulating in the social media that are derogatory towards someone and remove them.

1.1 Problem Statement

In this competition, our aim was to classify a given meme (image and the associated text) into one of the three categories:

- None (class `0`)
- Hilarious (class `1`)
- Troll (class `2`)

Troll memes are those that are derogatory or insulting or cause someone (individual or a group of individuals or organisation) to feel resentful, upset, or annoyed. Although classifying a meme as hilarious is subjective by nature, there are many memes which can be classified as hilarious by majority of people and hence can be termed as hilarious in general. Memes in the *None* category are those which are neither hilarious nor trolls. It may includes quotes, some random facts, plain images, etc.

2 Dataset

We used the dataset provided for *InClass Prediction Competition* which is available on Kaggle[1]. The train set consisted of 1991 data points and the prediction had to made for 600 data points for which the meme class is not known (*blinded* test set). A *data point* refers to image of a meme and the text present in it.

2.1 Description

The dataset consists of image of each meme, along with the text present in each of them in a separate `csv` file. The set of attributes in the `csv` file are:

{ID, image id, text, label, label_num}

where *label_num* belongs to one of {`0, 1, 2`} and the attribute *label* corresponds to the name of the category (string) which the *label_num* represents (i.e, `0` represents "*None*", `1` represents "*Hilarious*" and `2` represents "*Troll*") The *image id* contains the name of the image file in the dataset and *text* contains the text in the meme. Note that the test set does not have the label details.

Further, the dataset does not contain any missing values and the class distribution is almost balanced (Section 3.1).

3 Exploratory Data Analysis

3.1 Class Distribution

Class	Label	Count
0	None	604
1	Hilarious	689
2	Troll	698
	Total	1991

Table 1: Distribution of different classes in the training set

[1] https://www.kaggle.com/c/dataminingmtl782/data

The number of data points in each class is tabulated in Table 1.

3.2 Stop words

The need for stop-word removal depends on various factors including the nature of the dataset and models used. Though stop-words should be retained in models like BERT (which takes into account the *context*), it need be required in cases of simple models like SVM.

To decide if stop-words removal was necessary, we observed the most frequently occurring words in the texts of the meme. It was observed that stop words were present in all the classes and had similar frequency in each of them (Table 2) and the presence of a stop-word alone cannot distinguish which label a given meme would belong to. Hence, the stop-words were removed wherever required.

3.3 Word frequency

We were curious to see if any particular word is more popular among a particular class (e.g., political memes would mostly be associated with trolls; so memes with word "*election*" might probably be a *troll*). The top 10 most frequent words, after stop words removal, in each class is tabulated in Table 3.

3.4 Words count distribution

The distribution of number of words in the text of memes in train set, before any pre-processing, is shown in Figure 1.

3.5 Sentiment Scores

We calculated sentiment scores of the texts using TextBlob Library (Loria, 2018). The mean scores obtained for each class are tabulated in Table 4.

Class	Mean Sentiment Score
None	0.0891
Hilarious	0.0494
Troll	0.0319

Table 4: Mean sentiment score for different classes

4 Models

Model 1 VGG + BERT

Pre-processing: The text was converted to lower case, all URLs were removed, various non-alphanumeric characters were removed. Following this, the text was tokenized and lemmatized using WordNet Lemmatizer (Miller, 1995). The image is resized to 224 pixels and scaled to make the range of values $[0, 1]$.

Pipeline: A multi-modal model was made with VGG (Simonyan and Zisserman, 2015) and BERT (Devlin et al., 2019) as base for images and texts respectively. Dense fully-connected layers were added with dropouts to each of VGG and BERT and concatenated. Further, fully connected layers were added and the model was trained for 3-classes categorical classification with softmax activation. The starter code was taken from `https://github.com/AmbiTyga/MemSem`. A schematic of the model can be seen in Figure 2.

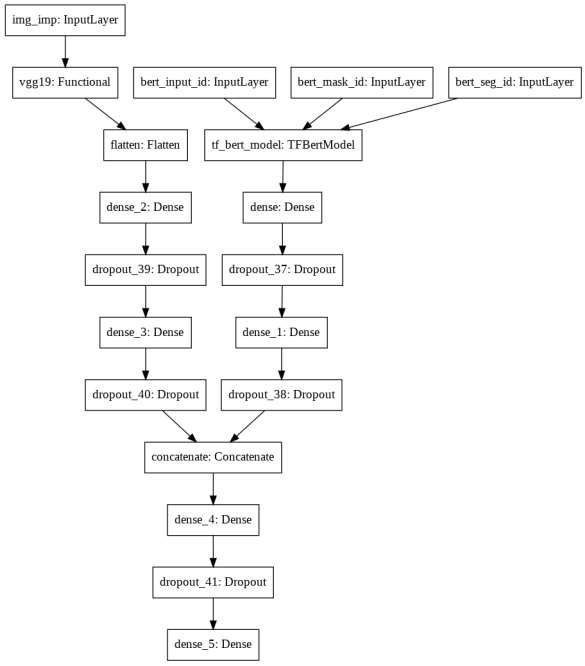

Figure 2: Pipeline for Model 1 and Model 2

Model 2 VGG + BERT

Pre-processing: Pre-processing steps were similar to the ones in Model 1. The only difference was that the text was not lemmatized.

Pipeline: The same model as in Model 1 was used– the only difference was in pre-processing step (and initialisation of weights).

Model 3 DenseNet = BERT

Pre-processing: Same as in Model 2.

Pipeline: The pipeline is almost similar to Model 1. However, the VGG base is replaced with DenseNet (Huang et al., 2017) base for images and a sequence output is obtained from BERT, which is fed into a Global Average Pooling layer before

Word	**Count**
the	321
you	223
to	171
i	170
a	168

(a) Class 0

Word	**Count**
the	303
you	261
i	200
to	200
a	185

(b) Class 1

Word	**Count**
the	326
you	236
a	217
i	206
to	195

(c) Class 2

Table 2: Frequency of words in different classes without stop-words removal

Word	**Count**
bill	108
meme	67
like	61
know	40
friend	32
get	31
time	31
say	27
make	24
look	23

(a) Class 0

Word	**Count**
meme	103
like	48
go	41
get	37
know	34
one	33
look	32
want	30
make	28
would	27

(b) Class 1

Word	**Count**
meme	57
like	41
say	36
go	36
one	36
make	33
know	32
get	32
kill	28
would	27

(c) Class 2

Table 3: Frequency of words in different classes after stop-words removal

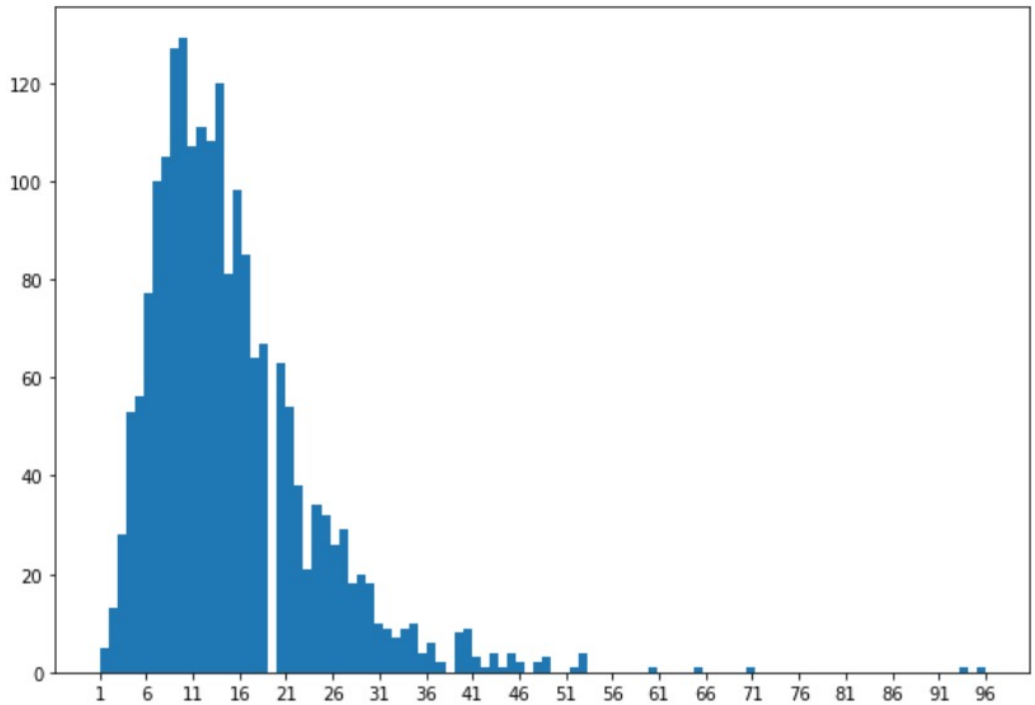

Figure 1: Words count distribution in text of memes in train set (before pre-processing)

using a fully-connected layer. A schematic of the model can be seen in Figure 3.

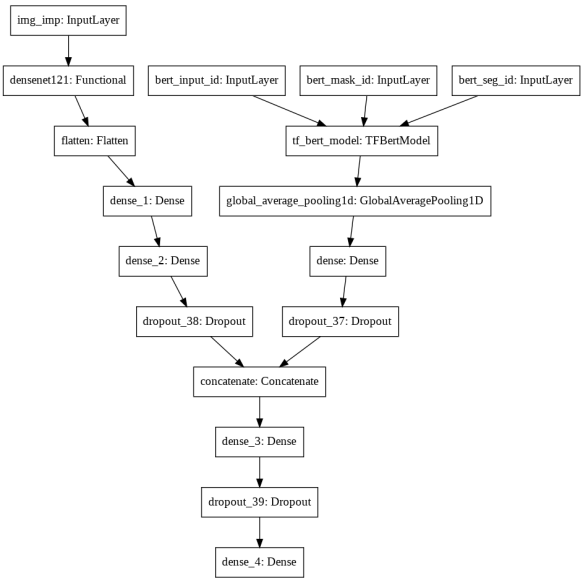

Figure 3: Pipeline for Model 3

Model 4 VGG + (GloVe+LSTM)

Pre-processing: Pre-processing steps were similar to the ones in Model 1. The only difference was that the stop words were also removed.

Pipeline: A multi-modal model was made with VGG and LSTM (Hochreiter and Schmidhuber, 1997) with GloVe word-embeddings (Pennington et al., 2014), as base for images and texts respectively. Dense fully-connected layers were added with dropouts to each of VGG and LSTM (from GloVe word-embeddings) and concatenated. Further, fully connected layers were added and the model was trained for 3-classes categorical classification with softmax activation. A schematic of the model can be seen in Figure 4.

Model 5 Radial Color Histogram+SVC

Radial Color Histogram measures the distribution of color in each segment (which can be radial) of the image.

Pre-processing: The data-points with index `858`, `1986` and `1990` in the training set were dropped to get rid of the erros arising from them.

Pipeline: The images were converted from RGB/B&W to HSV (as color degradation is less problematic when viewed in the HSV palette). Library available at `https://github.com/gmorinan/radialColorHistogram` was used for obtaining the radial color histogram with 3 bins.

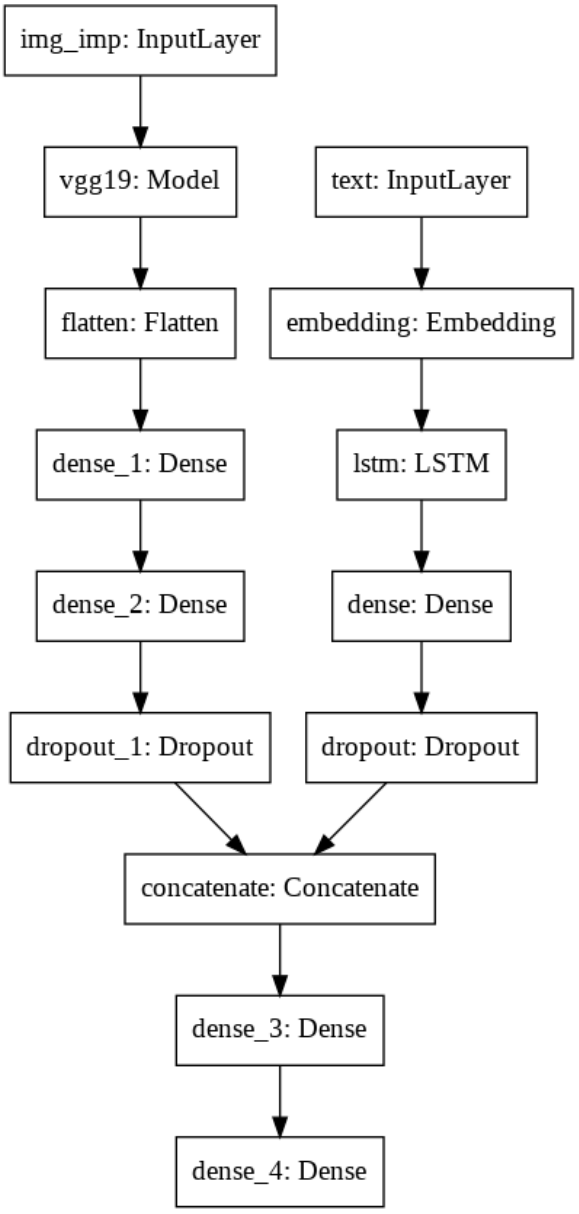

Figure 4: Pipeline for Model 4

The output features were fed into a Support Vector machine Classifier (SVC) (Cortes and Vapnik, 1995) with linear kernel. *This was the only setup where we did not make a validation split and used the entire train set for training.*

Model 6 Radial Color Histogram+SVC

The model setup was same as in Model 5. In this case, the only difference was that the training data was split into train and validation set (with split 0.15).

Model 7 Tf-Idf+SVC

Pre-processing: The text was converted to lower case, all URLs were removed, contractions were

expanded. At this point of time, we thought that symbols @ (used for tagging people) and # (used in hashtags) would help in classification and hence these symbols were retained specifically. Following these, various non-alphanumeric characters were removed. Finally, the text was tokenized, stop words were removed and tokens were lemmatized using WordNet Lemmatizer.

Pipeline: The texts were fed into a Term frequency - Inverse document frequency (Tf-Idf) Vectoriser which retained those words (unigrams and bigrams) whose document frequency (df) is at least 3, followed by a Support Vector machine Classifier (SVC).

Model 8 Radial Color Histogram + Tf-Idf+NB

Pre-processing: The images and texts were pre-processed as mentioned in pre-processing of Model 5 and Model 7 respectively.

Pipeline: The images were fed into a Radial Color Histogram model as specified in Model 5. The texts were fed into a Term frequency - Inverse document frequency (Tf-Idf) Vectoriser which retained those words (only unigrams) whose document frequency (df) is at least 3, followed by a Multinomial Naive Bayes (NB) Classifier. The outputs from images model and texts model were merged using SVC to create an ensemble. The idea was that making ensemble would improve the performance of the overall model, especially if certain models perfom well for one class and others for the class.

The texts are fed into a Radial Color Histogram model as specified in Model 5. The outputs from the text and images are merged

Model 9 BoW+Tf-Idf+SVC

Pre-processing: Texts with upper case characters were embedded with <CAPS> tag and multi-line texts were embedded with <MULTILINE> tag. *We thought that text in hilarious and troll memes would be split into multiple lines and have upper case characters (however, the <CAPS> tag was removed in the later models as we observed that the assumption was not correct all the time).* The text was then converted to lower case, all URLs (including those without http:// and https:// prefix) were removed, contractions were expanded. At this point of time, we thought that symbols @ (used for tagging people) and # (used in hashtags) would help in classification and hence these symbols were retained specifically. Following these, various non-alphanumeric characters were removed. Finally, the text was tokenized, stop words were removed and tokens were stemmed using SnowBall Stemmer (Porter, 2001).

Pipeline: We used a Bag of Words (BoW) representation to assign unique IDs to each word and Tf-Idf Transformer to take into account the term frequency of words rather than just their occurrences (as longer documents would have higher average count values than shorter documents, even though they might talk about the same topics). The transformed vector representations were fed into Support Vector machine Classifier (SVC) and the whole model was trained.

Model 10 BoW+Tf-Idf+SVC with text augmented with synonyms

Pre-processing: The initial pre-processing was same as Model 9, except that the <CAPS> tag was not embedded to the text, for the same reason stated earlier. Following all the previous pre-processing steps, a copy of train set was augmented with synonyms (i.e., words were replaced with their synonyms). The list of synonyms was obtained from https://raw.githubusercontent.com/Opla/SmallData-Augmentation-MachineLearning/master/data_augmentation/ppdb-xl.txt. The train set, hence, had double the number of data points. *Note that the validation set was not augmented.* Words with single characters were also removed as part of pre-processing.

Pipeline: The model setup was exactly same as Model 9. Here, the model was trained with double the number of data points (original text + text augmented with synonyms).

Model 11 BoW+Tf-Idf+SVC with text augmented with synonyms and WordNet synonymous paraphrasing

Pre-processing: The pre-processing steps were exactly same as in Model 10. Along with text augmentation with synonyms, a copy of train set was also paraphrased with synonyms from WordNet. Hence, the train set size was three times the actual one (after validation split). *Note that the validation set was not augmented.*

Pipeline: The model setup was exactly same as Model 9. Here, the model was trained with triple the number of data points (original text + text augmented with synonyms + text paraphrased using

WordNet).

Model 12 **Radial Color Histogram + LSTM + BoW+Tf-Idf+SVC with text augmented with synonyms and WordNet synonymous paraphrasing**

This setup had three sub-models which were trained independently and were merged for the final prediction.

1. **Radial Color Histogram:** The pre-processing and model setup were same as in Model 5.

2. **LSTM:** The pre-processing was similar to way texts were pre-processed in Model 4. The model, however, had only GloVe embedding layer, LSTM, fully connected layers and final output with softmax activation. The train set was also augmented with synonyms and paraphrased with WordNet (similar to Model 11).

3. The pre-processing and model setup were exactly same as in Model 11.

An ensemble was made using the three sub-models and the final decision was made using soft-voting (probabilities of a given data point belonging to each class from each sub-model were added and the class with maximum value was chosen as the final prediction).

Model 13 **Transfer learning from text tone classifier**

We thought that the tone of the text in meme might determine the class of the meme. Hence, we tried to use NLP model trained on a large dataset for classifying the tone of the text and transfer the knowledge for our meme classification task. One such trained model was obtained from `https://github.com/lukasgarbas/nlp-text-emotion` which can classify the text tone into *joy*, *sad*, *anger*, *fear* or *neutral* using CNN and Word2Vec (Mikolov et al., 2013). The probability predictions of this model for the five classes was used as the input to train a Decision Tree classifier for our meme classification task. Note that this model takes only the text in the meme into account.

Model 14 **Ensemble of transfer learning model from text tone classifier and BoW+Tf-Idf+SVC trained with text augmented with synonyms and WordNet synonymous paraphrasing**

This setup had two sub-models which were merged for the final prediction.

1. **Transfer Learning model from text tone classifier:** The pre-processing and model setup were similar to Model 13.

2. **BoW+Tf-Idf+SVC:** The pre-processing and model setup were similar to Model 11.

The predictions from the sub-models were fed into a Decision Tree classifer whose prediction was considered final.

Model 15 **Ensemble of transfer learning model from text tone classifier and BoW+Tf-Idf+SVC trained with text augmented with synonyms and WordNet synonymous paraphrasing**

The pre-processing and model setup were exactly same as in Model 14. In this case, the only difference was in values of some of the model parameters that were tweaked while tuning.

5 Experiment

5.1 Settings

The models were implemented in Python using Scikit-learn (Pedregosa et al., 2011) and Keras (Chollet et al., 2015) with TensorFlow (Abadi et al., 2015) backend. Matplotlib (Hunter, 2007) was used to plot graphs and Pandas (Reback et al., 2020; Wes McKinney, 2010) was used for data analysis. Other python libraries used include NLTK (Loper and Bird, 2002), Transformers (Wolf et al., 2020), OpenCV (Bradski, 2000), textaugment (Marivate and Sefara, 2020).

The `random_state` was set to 42 in most of setups. The dataset was split into training and validation (train-val) sets with splits varying from 0.1 to 0.2 depending on the setups (0.2 for Model 1 to Model 4; 0.15 for Model 6; 0.1 for Model 7 to Model 15).

Model	Macro F1-score			
	Train	Val	10% Test	90% Test
Model 1	0.7699	0.3399	0.25000	0.38333
Model 2	0.7597	0.3893	0.30000	0.37592
Model 3	0.9573	0.4245	0.28333	0.33333
Model 4	0.9989	0.3745	0.30000	0.36666
Model 5	0.48	—	0.30000	0.33148
Model 6	0.47	0.40	0.40000	0.31666
Model 7	0.78	0.38	0.31666	0.32777
Model 8	0.70	0.39	0.36666	0.32222
Model 9	0.89	0.43	0.33333	0.30000
Model 10	0.99	0.41	0.43333	0.37592
Model 11	0.99	0.42	0.36666	0.29259
Model 12	0.89	0.44	0.35000	0.34259
Model 13	1.00	0.41	0.46666	0.32592
Model 14	0.99	0.41	0.41666	0.34074
Model 15	1.00	0.41	0.40000	0.34074

Table 5: Performance of different Models

5.2 Results

To judge the performance of the models, we define the following metrics:

$$\text{Precision } \textit{(Prec)} = \frac{TP}{TP+FP}$$

$$\text{Recall } \textit{(Rec)} = \frac{TP}{TP+FN}$$

$$\text{F1-Score } \textit{(F1)} = \frac{2\times\text{Precision}\times\text{Recall}}{\text{Precision}+\text{Recall}}$$

where:

- **TP** (True Positive): Model predicts Positive and it is actually Positive.
- **FP** (False Positive): Model predicts Positive but it is actually Negative.
- **FN** (False Negative): Model predicts Negative but it is actually Positive.
- **TN** (True Negative): Model predicts Negative and it is actually Negative.

We use **macro average F1-score** (unweighted average of the F1-scores of each class) as the only performance metric. The performance of each model with this metric is tabulated in Table 5.

5.3 Observations

From the results in Table 5 and Table 5, the following observations can be drawn:

- Due a limited dataset size, most of the models were overfitting.
- For basic ML models, which have no knowledge about the language, usage of text augmentation techniques, like replacing words with synonyms and synonymous paraphrasing using WordNet, seemed to improve the performance significantly.
- Embedding special tags like `<CAPS>` and `<MULTILINE>` helped in classification when Bag of Words (BoW) technique was used.
- We observed that models pre-processed with simple word tokenizer performed better than tweet tokenizer in most of the cases. *However, we were not able to find reasons for the same.*

6 More Approaches

We document the models that we tried but did not give good results in this section.

1. **Using VGG/DenseNet and BERT (as in Model 1) independently.** The F1 score on validation set was not good.
2. **Augment images in multimodal setup.** As the dataset was very small, we thought that image augmentation would reduce overfitting. However, the performance did not improve significantly,

3. **Identify the person face in the image using a pre-trained model and use it for making predictions.** For example, we observed that many political memes were trolls–so we thought that images with political figures would most probably be a troll (practically). However, we did not get any model that would be useful for us nor were we able to find any dataset using which we could train such models.

4. **Identifying named entities in the text of the meme.** Generally some person (or a group of people) or a place is/are mentioned in the memes for one of the two purposes: First is to troll that person; and other is to write something good about that. So one approach could be that we find whether some named entity is present in the text and using the sentiment score of the text to predict which class the meme belongs to. For example, if the sentiment score is negative and the text contains name of a person, then most probably it would be the case where the person mentioned is getting trolled. On the other hand, if the sentiment score is highly positive, then it might indicate that some good fact is written about the mentioned person, and hence it might belong to class "*None*".

7 Conclusion

We tried several machine learning techniques, starting from a basic Naive Bayes classifier, to complex deep learning models, like BERT/LSTM and VGG/DenseNet, for the given meme classification task. The given dataset was small and we used some augmentation techniques, along with other ML techniques like regularization and dropouts, to combat this issue. We made 15 submissions on Kaggle as part of the competition, and obtained the best private score (i.e., mean F1-score on 90% test set) as **0.3833**.

Acknowledgments

The authors thank Google Colaboratory facility for computational resources.

References

Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dan Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. 2015. TensorFlow: Large-scale machine learning on heterogeneous systems. Software available from tensorflow.org.

G. Bradski. 2000. The OpenCV Library. *Dr. Dobb's Journal of Software Tools*.

François Chollet et al. 2015. Keras. `https://keras.io`.

Corinna Cortes and Vladimir Vapnik. 1995. Support-vector networks. *Mach. Learn.*, 20(3):273–297.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. Bert: Pre-training of deep bidirectional transformers for language understanding.

Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long short-term memory. *Neural Comput.*, 9(8):1735–1780.

Gao Huang, Zhuang Liu, Laurens Van Der Maaten, and Kilian Q. Weinberger. 2017. Densely connected convolutional networks. In *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2261–2269.

J. D. Hunter. 2007. Matplotlib: A 2d graphics environment. *Computing in Science & Engineering*, 9(3):90–95.

Edward Loper and Steven Bird. 2002. Nltk: The natural language toolkit. *CoRR*, cs.CL/0205028.

Steven Loria. 2018. textblob documentation. *Release 0.15*, 2.

Vukosi Marivate and Tshephisho Sefara. 2020. Improving short text classification through global augmentation methods. In *International Cross-Domain Conference for Machine Learning and Knowledge Extraction*, pages 385–399. Springer.

Wes McKinney. 2010. Data Structures for Statistical Computing in Python. In *Proceedings of the 9th Python in Science Conference*, pages 56 – 61.

Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. 2013. Efficient estimation of word representations in vector space.

George A. Miller. 1995. Wordnet: A lexical database for english. *Commun. ACM*, 38(11):39–41.

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. 2011. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830.

Jeffrey Pennington, Richard Socher, and Christopher Manning. 2014. GloVe: Global vectors for word representation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1532–1543, Doha, Qatar. Association for Computational Linguistics.

Martin F Porter. 2001. Snowball: A language for stemming algorithms.

Jeff Reback, Wes McKinney, jbrockmendel, Joris Van den Bossche, Tom Augspurger, Phillip Cloud, gfyoung, Sinhrks, Adam Klein, Matthew Roeschke, Simon Hawkins, Jeff Tratner, Chang She, William Ayd, Terji Petersen, Marc Garcia, Jeremy Schendel, Andy Hayden, MomIsBestFriend, Vytautas Jancauskas, Pietro Battiston, Skipper Seabold, chris b1, h vetinari, Stephan Hoyer, Wouter Overmeire, alimcmaster1, Kaiqi Dong, Christopher Whelan, and Mortada Mehyar. 2020. pandas-dev/pandas: Pandas 1.0.3.

Karen Simonyan and Andrew Zisserman. 2015. Very deep convolutional networks for large-scale image recognition.

Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Rémi Louf, Morgan Funtowicz, Joe Davison, Sam Shleifer, Patrick von Platen, Clara Ma, Yacine Jernite, Julien Plu, Canwen Xu, Teven Le Scao, Sylvain Gugger, Mariama Drame, Quentin Lhoest, and Alexander M. Rush. 2020. Transformers: State-of-the-art natural language processing. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pages 38–45, Online. Association for Computational Linguistics.