Text classification of Reddit corpus

Team **MaximumLikelihood**

Bogdan Mazoure *bogdan.mazoure@mail.mcgill.ca* 260636056,
Sorin Muchi *sorin.muchi@mail.mcgill.ca* 260575810,
Justin Smith *justin.smith3@mail.mcgill.ca* 260576439

I. Introduction

Natural Language Processing (i.e., NLP) techniques allow researchers to process huge amounts of linguistic data with little to no supervision (Aggarwal and Zhai 2012). Multiple different techniques have been and still are being developed in the field of speech processing and text classification to solve this task.

The goal of this project is to use the provided Reddit conversation corpus to train a classification algorithm, in order to correctly predict the topic (i.e., class label) of all the conversations from the test set. Unlike the first project's data, this dataset has balanced class sizes, which makes a zero-rule predictor useless to achieve state-of-the-art accuracy.

We propose three (not really) novel classifiers to solve the problem described above. The first approach uses a Naive Bayes classification method to model word dependencies, the second makes use of a non-linear k-nearest-neighbors (kNN) to split the text space in high dimensional space, and the last one uses bidirectional Long Short-Term Memory (LSTM) networks with dropout regularization on almost raw data. We will show that although the deep learning algorithm (LSTM network) achieves the highest accuracy on the test set, the more simplistic models like Naive Bayes are its heels in terms of both accuracy and running time.

II. Related work

Standard text preprocessing techniques as used in simpler algorithms have little impact on the performance of deep learning classifiers. Uysal and Gunal (2014) note that " ... the impacts of stop-word removal and stemming are small. However, it is suggested to apply stop-word removal and stemming in order to reduce the dimensionality of feature space and promote the efficiency of the text classification system". This is particularly important for Naive Bayes and non-linear classification techniques which rely on preprocessing steps in order to reduce random noise in the data set.

Kwon *et al.* (2003) had to deal with a somewhat similar problematic, where he used the TF·IDF representation inside the k-nearest-neighbors algorithm in order to classify Web sites. Although a little outdated, this approach can still provide a good insight about dealing with data taken from the Web using non-linear classification methods.

The current state-of-the-art deep learning text classification techniques are, as of February 2017, recurrent and convolutional neural networks (RNN and CNN, respectively). The Long Short-Term Memory (LSTM) networks, a variation of RNNs, are particularly known to yield good results in short text classification problems. A LSTM is a variation of a recurrent neural network, which can learn long-term dependencies lagged in time. It does so by passing modified "copies" of the input through iterations with its repeating module. Bidirectional LSTMs, on the other hand, split the hidden layer into forward and backward states, allowing the output neurons to combine information from past and future (Schuster and Paliwal 1997). Lee and Demoncourt (2016) compared the most widely known machine learning text classification algorithms on three datasets, and concluded that LSTM and CNN architectures, as suggested in their paper, yield the best accuracy. One of the most interesting deep learning text classification methods was proposed by Zhang *et al.* (2015), who suggested conducting character-level training using CNNs. This might be especially interesting in the context of social networks, where the text is often posted without proofreading and might contain typos.

III. Problem representation

Internally, all three models use slightly different representations of the provided Reddit corpus.

The non-linear k-nearest-neighbors classifier used the same "deep" cleaning preprocessing steps that was tried with the LSTM network, that is: removing stopwords, non-alphabetic characters, stemming using one of the best stemmers, Porter's Snowball stemmer (Jivani 2011) and finally splitting on whitespace.

For the optional model, all data is first cleaned from all typographical mistakes by using the top most common typos list on Wikipedia (`https://en.wikipedia.org/wiki/Wikipedia:Lists_of_common_misspellings/For_machines`). Then, the WordNet (Miller 1995) filtering is applied to the test data in order to remove as much unknown words as possible with minimal loss of information (see Appendix). This is of course an $ad\ hoc$ measure which attempts to minimize the amount of unseen tokens(i.e., words). Then, we apply a variation of text cleaning used by Kim (2014). That is, we expand all contractions, while keeping punctuation intact. We have experimented both with a deep cleaning procedure, where every non-alphabetical character was purged, as well as a "deep" cleaning version described previously.

The extended preprocessing was usually preferred, since it allowed the neural network to capture more subtle variations.

Both Naive Bayes and kNN classifiers made use of Term Frequency * Inverse Document Frequency (TF·IDF) to store a weighted count of some specific words from the processed Reddit corpus.

The LSTM network, in his turn, took as input two matrices: a training input matrix $T_X : \mathbb{N}^{n \times m}$ and a training label matrix $T_Y : \{0,1\}^{n \times 8}$ (T_Y is an expanded binary label matrix). Here, m depends on the maximum sentence length and n is given by the number of sentences in the training set. Such a representation embeds each sentence as a row vector of natural numbers, each replacing a word or punctuation token. The exact same process is applied to the test set, so that we do not have any discrepancy *a priori* our training process. All shorter sentences are zero-padded from the end. In the literature relating to natural language processing, it is suggested that taking the top 20,000 most frequent tokens might yield decently good results for a reduction in training time, but since we aim to achieve maximal accuracy, we discard this simplification.

IV. Algorithm selection and implementation

A. Naive Bayes

Text classification using a Naive Bayes model is done through originally training the model to get a conditional probability weighting for each class and word. This set of probability weights is then applied to the features in the test/validation documents to then pick the corresponding class with the highest probability. As a particularly important preprocessing step, the *nltk* Python package was used to yield a list of stop-words that needed to be removed.

It can be shown (and has been through the course), that during the training phase, conditional probabilities of the occurring words are to be viewed as a set of features. If we assume a multinomial underlying distribution of tokens, it is trivial to show using standard log-likelihood maximization techniques that the set of fitted probabilities which maximizes the previously mentioned likelihood function is simply $\hat{\theta}_{MLE} = \{\frac{n_i}{N}\}, \forall i$, where n_i is the number of instances with feature i, and N is the total number of instances. Essentially, using term frequencies or any function of them (since maximum likelihood estimators are invariant with respect to one-to-one mappings) will give us the best solution.

Laplace smoothing was used to handle words that had not been seen before. Laplace smoothing using an m-estimate assumes that each feature is given a prior probability, p, that is assumed to have been previously observed in a virtual sample of size m. This prevents us from predicting a zero probability for sentences with unknown words.

By transforming our text data with TF·IDF weighting before passing it to the classifier, more weight is given to words found in lesser documents of the training set. The classifier is thus able to handle training documents with words that were not previously seen (explained in more depth in k-nearest-neighbors section). With this, we must estimate the probability of a given class c_j from a test document D with words w_i as:

$$P(c_j|D) = P(c_j) \prod_{w_i \in D} \frac{f(w_i, c_j)}{\sum\limits_{w \in V} (f(w, c_j) + |V|} \log\left(\frac{nd}{nd_{w_i}}\right) \tag{1}$$

where V is our vocabulary (all of the words that are found in the training documents), $f()$ is the count function such that $f(w_i, c_j)$ gives a count of the all occurrences of word w_i in training documents of class c_j, nd is the total number of training documents, and nd_{w_i} is the number of training documents that actually contain word w_i. In other words, the Naive Bayes text classification model implies conditional independence of terms given their context (i.e., the document in which they are found). This is actually a practical problem, since words are in fact dependent on their context.

In order to estimate the performance of the Naive Bayes classifier on unseen data, cross-validation was conducted on partitions of the training dataset for hyper-parameter tuning. The hyper-parameter in question is the dimensionality of n-grams that should be used. Because we saw an immediate improvement after using 2-grams, we decided to investigate with larger values of n. By using leave-20,000-out cross-validation on 100,000 training instances, we determined that 3-grams yielded the highest average accuracy (μ_{acc}) across the hyper-parameter search space. Hence, our Naive Bayes model was based on TD-IDF weighting scheme with $n = 3$ n-grams.

B. k-nearest-neighbors

The k-nearest-neighbors algorithm is a lazy classification algorithm. kNN uses the class of the k nearest points (i.e., neighbors) to a query point as an approximation of that point's class. Our decision heuristic is a multi-class majority rule, where the input vector is labeled of class C if the class C has received most votes from all its neighbors; we break ties randomly and uniform polling is applied (each neighbor has equal vote weight). We used the Euclidean distance to measure the distance between any two points:

$$d(v_1, v_2) = \sqrt{\sum_{i=1}^{p} (v_{1i} - v_{2i})^2} \tag{2}$$

Multiple distance functions have previously been examined in the literature, such as Cosine Similarity (has a nice geometric interpretation), or any L_p norm defined over some L_p space (L_1, L_2 and L_∞ are the most used ones). However, a simple Euclidean distance measure is straight-forward to implement, has a relatively low computation cost, and performs sufficiently well in this context.

We used the standard text preprocessing algorithms described in the text classification tokenization literature: typographical error correction, word stemming, and stop-word removal. The last preprocessing step consisted in creating a 148,500 by 4,228 sparse matrix containing the TF·IDF data for our

training set, and a 16,500 by 4,228 sparse matrix for our validation set. In fact, we used the top 4,228 most frequent words as features (due to the huge time complexity of the prediction process).TF·IDF was used to quantify the importance of a given word through a given corpus, by computing the term frequency of a word in one document, multiplied by the inverse document frequency of the given word, across all documents. It is formally defined as follows:

$$\mathcal{T}_{w,D} = tf_{w,D} \cdot idf_{w,D} = f(w,D)\left(\log\left(\frac{nd+1}{1+nd_w}\right)+1\right) \tag{3}$$

, where w is some word in the vocabulary, D is some text document, $f()$, nd and nd_w are defined as in Equation 1. We can see that Equation(3) uses Laplace smoothing to eliminate zero probability prediction for missing data (equivalent to a uniform prior).

We used TF*IDF with a frequency threshold proportional to the Frequency-Exclusion (F-E) defined in the Naive Bayes section, of 0.0009, meaning that all features with a TF*IDF value less than 0,0009 were pruned out, as they would not bring significant information gain. L_2 normalization (division by L_2 norm) was also used when generating the the weights.

C. LSTM neural network

The core of this classifier is (thankfully) already encoded into the Tensorflow/Theano libraries and handled at a very high level by Keras. The network architecture relies on a mixture between ideas of Lee and Dernoncourt (2016) and on examples from Keras documentation. Figure 1 shows a schematic view of the model used. The embedding layer takes as input T_X and T_Y, and encodes each token (in this case a number) in d dimensions. Here, d was empirically selected from $\{64, 100, 110, 128, 300\}$ to be $d = 300$ using leave-16,500-out cross-validation, in order to maximize validation accuracy. Because the resulting layer had space complexity $m \times n \times d$, we needed to run the training process on Google Cloud with 32 Gb RAM machines. The resulting tensor from the embedding layer was fed to a bidirectional Long Short-Term Memory (LSTM) (Hochreiter and Schmidhuber 1997), with 300 hidden neurons. Dropout regularization, a technique proposed by Srivastava *et al.* (2014) which consists in disabling a part of neurons in the LSTM layer with probability $p = 0.5$ (chosen empirically from $\{0.2, .05\}$ using leave-16,500-out cross-validation), was used. Dropout has been suggested to avoid over-fitting, since neighbors of a disabled neuron will have to work without its input, which can result in slightly modified instances of data. Finally, an ordinary (dense) fully-connected neural network layer of size 8 sits on top of the LSTM. A softmax activation function, which is a general version of the logistic function, is then applied on the output of the dense layer to select the most likely class label. Other parameters, such as the LSTM activation function and LSTM weight initialization technique were left to default values, due to time constraints. The categorical cross-entropy loss (also known as multiclass logloss), defined as

$$\mathcal{L} = \sum_{j}^{m}\left(-\frac{1}{n}\sum_{i}^{n} x_{ij}\log(p_{ij}))\right) \tag{4}$$

was the target function selected to be minimized. Here, m is the number of different output features, n is the number of training/validation/test examples, y_i is the predicted label vector ($\{1,0,0,0,0,0,0,0\}$ for predicting instance i to be of class 1) and $p_i = \{p_1, ..., p_8\}$ is the probability vector returned by the classifier. The stochastic gradient descent ADAM optimizer (Kingma and Ba 2014) was used for training, in order to minimize the loss function defined in (1). Because of the huge memory complexity, training had to be conducted in batches of 64 rows (i.e., sentences). Once again, batch size was selected from $\{32, 64\}$ using the same cross-validation as before. All cross-validation procedures consisted in varying the target parameterize while holding the others constant. The hyper-parameter search space was taken from Lee and Dernoncourt (2016). Training was conducted for 4 epochs, after which the model started to overfit. Early stopping was used if the validation loss did not improve for 2 epochs. The learning rate was reduced by a factor of 0.7 once the validation loss did not improve sufficiently for 2 epochs.

Other architectures have been tried: unidirectional (forward) LSTM, CNNs with various filter sizes (in this context, consecutive words) and numbers of filters, as well as a combination of CNN and LSTM architectures. All of them yielded results similar to the ones we obtained using our proposed pipeline, which is why we omit their metric plots. Moreover, the following data preprocessing techniques have been tried in various combinations: stopword removal, lemmatization, stemming, punctuation removal, typo correction using either pre-built lists or WordNet similarity (see Appendix), contraction expansion. CNNs were much faster than LSTM, but needed more iterations to achieve the same accuracy (more than 5 epochs for CNN vs. 2 epochs for LSTM for over 95% training accuracy).

V. Testing and validation

The accuracy for various models is mostly being reported in this section, due to the fact that it is used to assess the performance on the Kaggle leaderboard. However, additional metrics (like loss) can also be found.

A. Naive Bayes

The Naive Bayes classifier performed surprisingly well on both the training set with cross-validation, and on the test set. It is interesting that, for example, "politics", "news", and "worldnews" instances were most often misclassified. Since they are very closely related topics, this has a nice intuitive interpretation. On the other hand, the classifier confused many sports classes ("nba","nfl","hockey"), since they tend to have many common words and idioms. Table I compares the Naive Bayes' model for text classification when using 3-Grams data encoding. The average training accuracy, μ_{acc}, as well as the batch's standard deviation, σ_{acc}, is included.

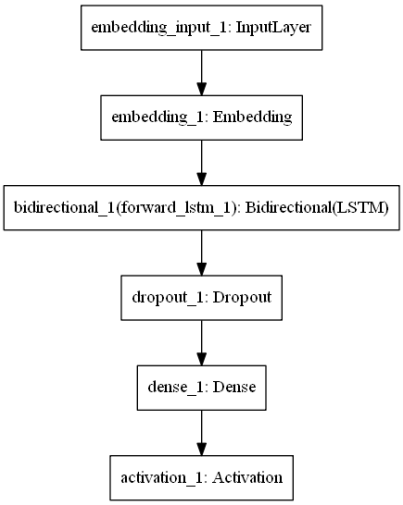

Fig. 1: Neural Network architecture used to classify Reddit documents (natively generated via Keras)

F-E	μ_{acc}	σ_{acc}
0	0.945	0.00163
1	0.941	0.00127
2	0.935	0.00152
3	0.930	0.00147
4	0.926	0.00133

TABLE I: Comparison of Naive Bayes' model performance (in mean accuracy across folds) when varying the Frequency-Exclusion parameter (F-E), while holding other parameters constant.

These quantities were obtained using leave-20,000-out cross-validation. The Frequency-Exclusion parameter regulates the cutoff frequency value required for a word to be considered "frequent", and is directly linked to the frequency threshold defined in the kNN model.

B. k-nearest-neighbors

The k-nearest-neighbors model is very particular, because it technically does not have neither training nor validation phases. In other words, fitting the model just means querying k closest points to some row or column vector of features. Using leave-16,500-out cross-validation, we achieved a 0.839 validation accuracy, and 0.743 test accuracy.

k	*Valid. accuracy*
2	0.821
3	0.839
4	0.836

TABLE II: Overview of validation accuracy

Table II shows how the number of neighbors, k was varied to obtain the best accuracy with $k = 3$. The fact that k has little impact on classification accuracy suggests that the preprocessing steps cleaned out much of the useless words. Additional visual resources (confusion matrices) are available for various values of the hyperparameter k in *Appendix.C*.

C. LSTM neural network

Almost state-of-the-art performance was achieved with the LSTM network. Figure 2 shows a plot the loss as defined in equation (4) versus batches processed. Due to huge amounts of data, the training had to be split into 4 epochs (iterations), and each of the 150,000 training samples had to be loaded into memory in batches of 64. We see, however, that the loss indeed decreased as the training progressed. On the other hand, Figure 3 shows that accuracy increases as a function of complexity (i.e., of number of training batches and epochs).

The various architectures that were tried (unidirectional

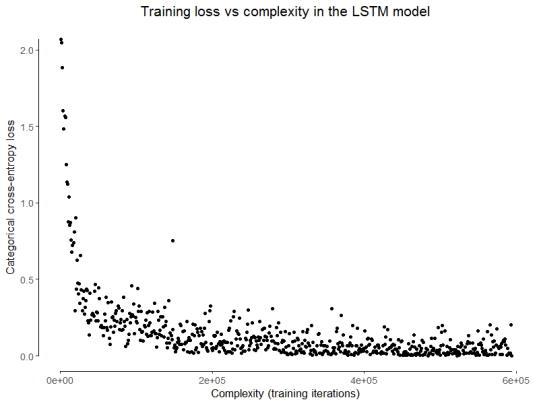

Fig. 2: Training categorical cross-entropy loss in the LSTM network as function of batches processed over 4 epochs

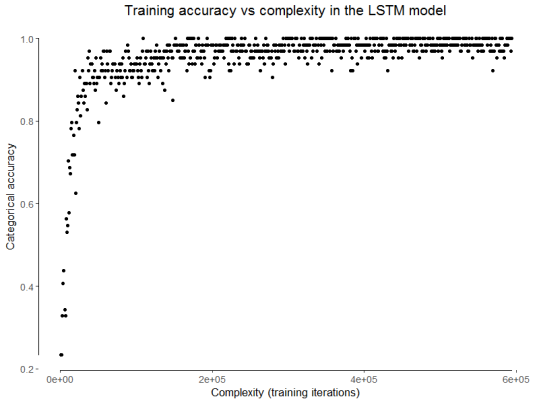

Fig. 3: Training categorical accuracy in the LSTM network as function of batches processed over 4 epochs

LSTM, CNN and CNN combined with LSTM) resulted in roughly the same test set accuracy. The major criterion which helped us choose the best model was hence the time required to run a training epoch, and also how fast (in terms of epochs number) could a given network converge to around 95% validation accuracy. Table III summarizes the three models' metrics and performance:

The above table suggests that, in terms of public test set accuracy, the bidirectional LSTM network should be preferred to the two other simpler models. However, note that the categorical accuracy on the private test set remained unknown until Thursday, February 16^{th} 7:00pm. On one

Model	*Train. accuracy*	*Valid. accuracy*	*Test accuracy*
Naive Bayes	0.9300	0.9300	0.95580
k-nearest-neighbors	-	0.8390	0.74276
LSTM	0.9877	0.9609	0.96358

TABLE III: Overview of accuracy achieved for the Naive Bayes, kNN and LSTM models on the training, validation and test (public) sets. Note how no training accuracy is presented for kNN, since no proper training phase occurs.

hand, if we assumed that the split of the test set into public and private parts was uniform, by basic properties of expectation we would get roughly the same accuracy on the private section of the set. If, however, the private test set was purposely selected as to contain all the ambiguous instances, then the private test accuracy will be lower. However, it turned out, after seeing the results of the categorical accuracy on the private test set, that the test set was indeed split uniformly into two parts.

Finally, Table IV presents an overview of the best model (bidirectional LSTM with dropout regularization)'s training and validation metrics:

Metric	*Value*	*Training/Validation*
F_1	0.9878	Train.
Precision	0.9890	Train.
Recall	0.9866	Train.
Loss	0.1526	Valid.
F_1	0.9612	Valid.
Precision	0.9634	Valid.
Recall	0.9590	Valid.

TABLE IV: Metrics from the LSTM network, on the final epoch. The loss function here is the multiclass logloss

As expected, performance on the validation set is slightly worse than on the training set, due to unseen data (e.g. new words).

VI. Discussion

The best model (bidirectional LSTM with dropout) achieved a public test categorical accuracy score of 96.358 % and 96.388 % private test set accuracy, which is high, but lower than expected, given that simpler models like Naive Bayes scored nearly as well. However, one major advantage of the LSTM network over kNN and Naive Bayes was that, for the LSTM network, preprocessing had very little impact. For instance, stemming, lemmatization, stopword/punctuation removal did not contribute to improve the accuracy. Even casting every letter to lower case did not significantly impact accuracy. The only advantage of doing such steps would be to reduce the number of words passed as input to the Embedding layer of the network; if the feature space reduction is significant enough, preprocessing may improve the running time of the algorithm.

One big advantage of kNN over other classification algorithms is that kNN can have an arbitrary decision boundary that closely fits the data. With enough data and a large enough hyper-parameter k, kNN performs reasonably well in terms of accuracy.

However, kNN's problem was that it took a very long time to query (i.e., predict) each of the 53,000 test instances. The search space is actually of exponential complexity, since for each of the test instances, we will have to to query each point of the training set, compute the distance function, sort it, pick the k closest neighbors and finally apply a pooling function (in our case, multiclass majority vote) to retrieve the label. Hence, we suggest to parallelize the prediction process, because all predictions are independent and can be done separately.

Other disadvantages of kNN as opposed to Naive Bayes are that for kNN the hyper-parameter k must be optimized, either with training-validation split or through cross-validation, which increases the computational cost Moreover, kNN does not assign different weights to different data points. We managed to compensate for the latter by implicitly using the TF*IDF weights in our distance metric. Note that this step would not be necessarily straight-forward for other types of data sets, like images.

Naive Bayes may seem like an overly simplistic model, but it did perform surprisingly well even if its conditional independence of features assumptions were most likely violated in natural language corpora.

In conclusion, we have proposed three text classification algorithms, Naive Bayes, k-nearest-neighbors and bidirectional LSTM with dropout, which yielded better than expected performances. We suggest to look into such preprocessing steps as $word2vec$ binning (which was not examined due to time constraints, combined with typo correction. Empirical evidence suggests typo correction has a tractable impact on text classification, and used together with other approaches it might yield a decently complete model of social network corpora.

VII. Statement of Contributions

1) **Bogdan Mazoure**: Implemented the LSTM neural network model, wrote the report except details specific to Naive Bayes classification and k-nearest-neighbors.
2) **Sorin Muchi**: Implemented the k-nearest-neighbors algorithm, performed data preprocessing using TF*IDF, classified data using kNN, and wrote kNN-specific parts of the report.
3) **Justin Smith**: Implemented the Naive Bayes method for text classification and Laplace Smoothing, and wrote the parts of the report relating to Naive Bayes classification.

We hereby state that all the work presented in this report is that of the authors.

References

[1] Aggarwal, Charu C., and ChengXiang Zhai. "A survey of text classification algorithms." Mining text data. Springer US, 2012, 163-222.

[2] Lee, Ji Young, and Franck Dernoncourt. "Sequential short-text classification with recurrent and convolutional neural networks." arXiv preprint arXiv:1603.03827, 2016.

[3] Zhang, Xiang, Junbo Zhao, and Yann LeCun. "Character-level convolutional networks for text classification." Advances in neural information processing systems, 2015.

[4] Gutman, Jacqueline and Richard Nam. "Text classification of reddit posts." *https://jgutman.github.io/assets/SNLP_writeup_gutman_nam.pdf*
[5] Sriram, Bharath. "Short Text Classification in Twitter to Improve Information Filtering." *https://etd.ohiolink.edu/rws_etd/document/get/osu1275406094/inline*
[6] Uysal, Alper Kursat, and Serkan Gunal. "The impact of preprocessing on text classification." Information Processing & Management 50.1, 2014, 104-112.
[7] Kim, Yoon. "Convolutional neural networks for sentence classification." arXiv preprint arXiv:1408.5882, 2014.
[8] Lewis, David D. "Feature selection and feature extraction for text categorization." Proceedings of the workshop on Speech and Natural Language. Association for Computational Linguistics, 1992.
[9] Hochreiter, Sepp, and Jrgen Schmidhuber. "Long short-term memory." Neural computation 9.8, 1997, 1735-1780.
[10] Schuster, Mike, and Kuldip K. Paliwal. "Bidirectional recurrent neural networks." IEEE Transactions on Signal Processing 45.11, 1997, 2673-2681.
[11] Kingma, Diederik, and Jimmy Ba. "Adam: A method for stochastic optimization." arXiv preprint arXiv:1412.6980, 2014.
[12] Srivastava, Nitish, et al. "Dropout: a simple way to prevent neural networks from overfitting." Journal of Machine Learning Research 15.1, 2014, 1929-1958.
[13] Kwon, Oh-Woog, and Jong-Hyeok Lee. "Text categorization based on k-nearest neighbor approach for web site classification." Information Processing & Management 39.1, 2003, 25-44.
[14] Jivani, Anjali Ganesh. "A comparative study of stemming algorithms." Int. J. Comp. Tech. Appl 2.6, 2011, 1930-1938.
[15] Miller, George A. "WordNet: a lexical database for English." Communications of the ACM 38.11, 1995, 39-41.

VIII. Appendix

A. Dealing with unseen data in the LSTM model

After tokenization, we are left with two sets: tokens in the training set and tokens in the test set. Another partition could be to assign tokens from both sets above to group 1 if they appear in the training set and in group 2 otherwise. When we do so, we are left with around 12,000 unseen symbols in the test set. This means that, if we use standard techniques, they will either: (1) be categorized as the "UNKNOWN" token, or (2) be removed completely. We introduced a third approach, which might yield a better result.

From the 12,000 tokens, we correct possible typos in each of them (given that the data is not proofread) via PyEnchant. Note that if no error is found or if the error is too pronounced, the token is left as is. We then cluster words based on WordNet distance (also known as synset distance), as defined in NLTK's wup_similarity(). For each word $w_1 \in corrected\ Test\ set$ and for each $w_2 \in Training\ set$, we compute the WordNet pairwise distance $d(\cdot,\cdot)$ or equivalently the similarity measure $s(\cdot,\cdot)$ between w_1, w_2. We then take $arg\ \max_{w2} d(w_1, w_2) = arg\ \min_{w2} s(w_1, w_2)$ as being the closest word in the test set to the training set. Here is an example of WordNet mappings typo-corrected Test set $\Rightarrow$ Train set:

Raw Test set	*Typo corrected*	*Train set*
inthebreeze	interbreed	crossbred
transcending	transcending	maybe
buryfc	bury	burys
interested	interested	interesting
crated	crated	incase
presiden	preside	preside
unstaged	upstaged	upstaged
pitchforking	pitchforking	forked

TABLE V: Wordnet Similarity results

Typos are also corrected using the Wikipedia 5,000 most common typing mistakes, after the application of WordNet similarity filtering. This approach reduces the number of unseen words in the test corpus.

B. Links to datasets

1) **Test set as NumPy arrays**: `https://www.dropbox.com/s/rraxmprkj3ewg9l/test_CaseSensitive.pickle?dl=0`
2) **Training set as NumPy arrays**: `https://www.dropbox.com/s/ue5as1m1dgtlbgn/train_CaseSensitive.pickle?dl=0`

C. k-nearest-neighbors performance

Performance was assessed mainly via confusion matrices, since no ROC curve can be drawn in 8 dimensions (yet). Figures 4,5 show the confusion matrices obtained by varying the hyper-parameter k in the kNN algorithm. The visualizations are presented below:

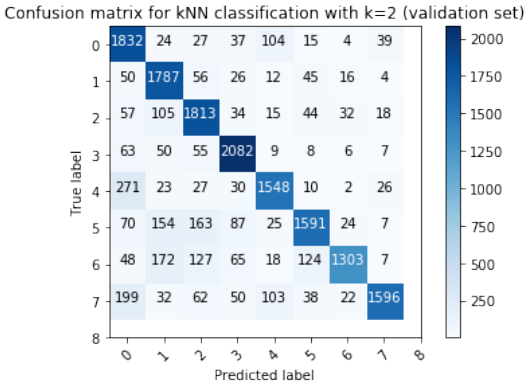

Fig. 4: kNN confusion matrix with $k = 2$

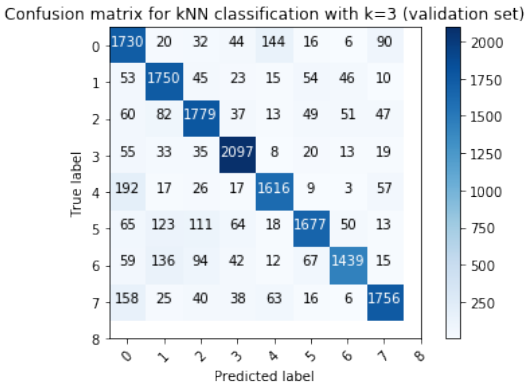

Fig. 5: kNN confusion matrix with $k = 3$