Multi-Class Email Text Classification Model

WRITTEN BY: PEDRO JUNIOR VICENTE VALDEZ

Main Developers:
PEDRO JUNIOR VICENTE VALDEZ – RAJAN RANAJIT DUTTA

INFO 154/254 - Data Mining and Analytics

Introduction

When using email services, it is easy to have a disorder inbox, especially when it is used for everything including work, finance, and personal matters. A small fraction of the population organizes their inbox by collocating emails into specific folders which they themselves create. These folders might be named "bills", "work", "travel", etc. When going over the different public datasets avaliable online, we notice that the ENRON dataset was the only real email dataset publicly avaliable. Knowing this and the fact that inboxes might benefit from an automatic mode of ordering themselves, we decided to use this dataset to create a model that will classify the email based on information like day, time of the day, message content, length of the text, etc. Some of the features will have to be engineered and others extract. Some of our research questions include:

- What do we need to successfully build an email classification tool?
- What kind of preprocessing and feature engineering is useful and adequate for text classification problems?
- What type of models can we use?
- Are there some models that perform better in text classification problems?
- How do we improve and/or optimize said model?

More questions will arise as we explore and build solutions for this problem. All in all, our goal is to build a ***Multi-Class Email Text Classification Model***. If successful, this tool can be useful and important at first for employees of corporations and companies that want an automatic way of email classification so they can easily access the portion of their email that they want.

The reason we specify people in corporations/companies derives from the fact that the model will be train and tested in Enron's corporations' emails. Almost all the emails are related to business, energy and finance. With time, it could then be expanded to be used by everybody in order to bring order and efficiency to their email inbox. Imagine being able to receive a bill that goes directly into the bills folder, or event information that automatically gets stored in the Events folder, same with travelling, etc. Once built, this model can be hosted online and offered as an API or a widget that could be downloaded easily. The model could then be adjusted for the specific individual using their own personal information bringing then a better experience, just as micromarketing.

ENRON Email Dataset Description

The dataset used for training, testing, and validating was the Enron Email Dataset. This dataset contains 517401 rows corresponding to emails generated by employees at the Enron Corporation, mostly senior management. The raw dataset was originally published by the Federal Energy Regulatory Commission during the investigation of Enron's collapse in December of 2001. The dataset was process before the release due to privacy and integrity issues. Some of the preprocessing done prior to the release were:

- Deleted emails that would affect employees
- Invalid email corrections (changing invalid email addresses to *user*@enron.com or in case of no recipient no_address@enron.com)

This is one of the few real email datasets in existence therefore we wanted to take advantage to build an email classification tool for enterprises/companies. The dataset, as mentioned previously, is made of 517401 rows and two columns: file and message. The file column contains the label information. By label information we mean the folder were the email was saved/placed. The message column contains information like the sender's emails, receivers' email, date, subject, if it contains bcc or cc or both, and the message text/content. There is a public code avaliable for parsing and extracting information from the dataset. This code was used to have a working dataset from which we could further do preprocessing and feature engineering. The dataset was received from https://www.cs.cmu.edu/~./enron/ and the public tool for extracting information was obtained from https://www.kaggle.com/donaldtfung/e-mail-categorization/.

The folder we're the emails we're stored are taken as the label. Since there we're thousands of different labels, a clustering algorithm was implemented to group different labels into 11 labels which will be further described later in this document. The distribution of the labels can be seen in Figure 1.

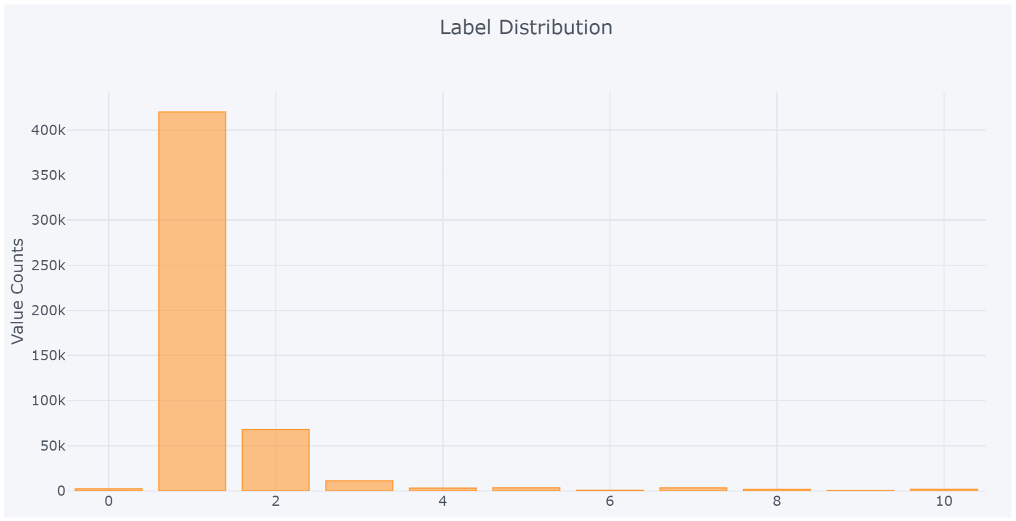


Figure 1 – Label Distribution after Clustering on entire ENRON email dataset

We can observe that the dataset is heavily distributed to label 1 which is no surprise since it belongs to the category “sent mail”. This folder is the default folder we’re the emails are stored. Not everybody stores their emails in folders. Rows with label 1 are dropped since prediction target is not known. Still the dataset is valuable and therefore it will be used to build an email classification tool for enterprises.

Approaches and Methodology

After extracting the working dataset, several tasks had to be overtaken in order to have a workable model. We decided to train four different models (one per group member) and at the end do an ensemble method using majority vote. We couldn't use the folder alone as the dependent variable because there were over 1,700 different folder names. Rajan built a clustering model that fit word embeddings to each folder name using Word2Vec (trained on the Google News corpus) and outputted 11 different clusters of folder types, and then created descriptions of each cluster (e.g. all folders with label 7 are related to a business arm or project). Rajan also made several manual edits to make to the classifications, mainly moving items from the "Random/NA" category to one of the more specific categories. The 11 folders are as follow:

```
0: **Weather/Natural Stuff**
1: **Administrative/Office General**
2: **Random/NA**
3: **Financial and Logistics**
4: **Related to other people**
5: **Places**
6: **Legal**
7: **Business arms/functions**
8: **2-letter/random stuff**
9: **Other Firms**
10: **HR, Recruiting, MBA programs**
```

Having the labels for our dataset we we're ready to go to start building our models. All team members contributed to the dataset preprocessing. I was in charge of building the LSTM model. This work will be described in detailed in the next section therefore I will only provide a summary of my teammates work in the rest of this section.

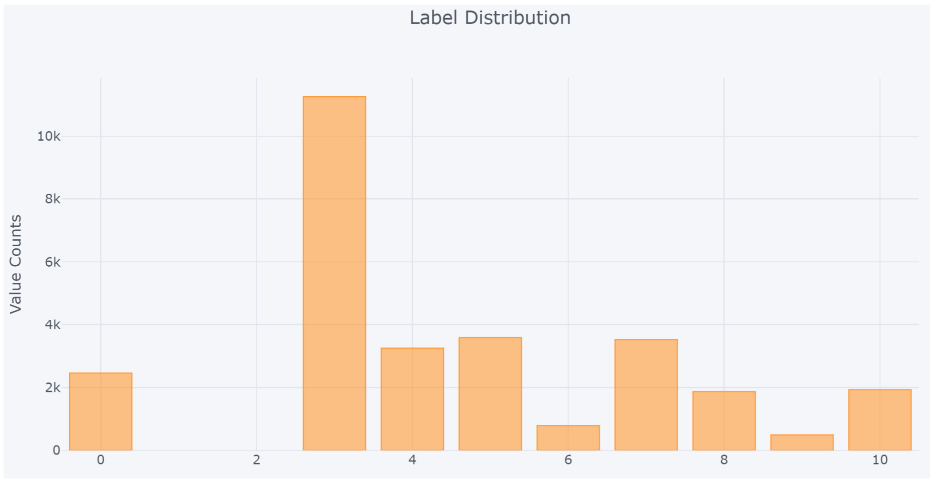
Rajan built a KNN model under the hypothesis that the labels should be clustered together, since we also used KMeans to create the label clustering. Therefore, we believed KNN should be an effective model. After training the KNN model with various values for n_components ranging from 3 to 11. The n_components value with the best result was n = 10, which achieved a training set accuracy of 45.8% and test accuracy of 38.4% (after performing a train-test split on the training dataset). It is good practice for a KNN to have an odd value for n (so there will always be a majority vote), hence Rajan used n = 11 in model tuning. As for optimization, Rajan tried to improve model performance using a bagging ensemble method. Unfortunately, this does did not help the model. Rajan also tried using PCA before fitting a KNN. This also did not yield much improvement, and so I save the vanilla KNN(n=11) classifier.

Individual Work

Pedro Junior Vicente Valdez

During the project, I performed several tasks and implemented several concepts learned during class and while completing the labs. During the preprocessing step I first worked in optimizing the dataset. This included getting rid of unnecessary data, features and deleting data instances that were not applicable for training. This was important since several models like doc2vec required large memory requirements. The bigger the file, the bigger those requirements. Reducing and optimizing the dataset also results in faster training times. After cleaning the data, I added some features including the message length and one hot encoded several features like the day of the week. I decided not to give the day of the week integers within the same column since it would bias the model to give more weight to Sunday since it has a higher integer. After Rajan implemented the clustering algorithm for label reduction, I also implemented PCA and TSNE to reduce the dimensionality of the vector representation of the labels to create visualization for exploratory data analysis. Figure A1 in the appendix is an example of only 200 labels represented in 2-Dimensional space.

Afterwards, most of the work rely on building different models for testing with the dataset. It is worth mentioning that the dataset is strongly skewed. We can see that most of the labels belong to legal/financial/logistical (3). We will now describe the final model.

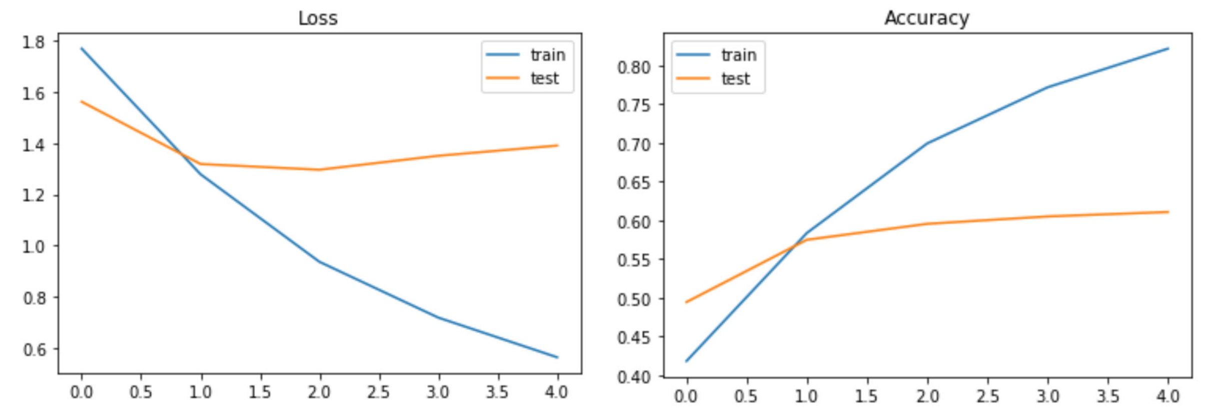


Long Short-Term Memory Model

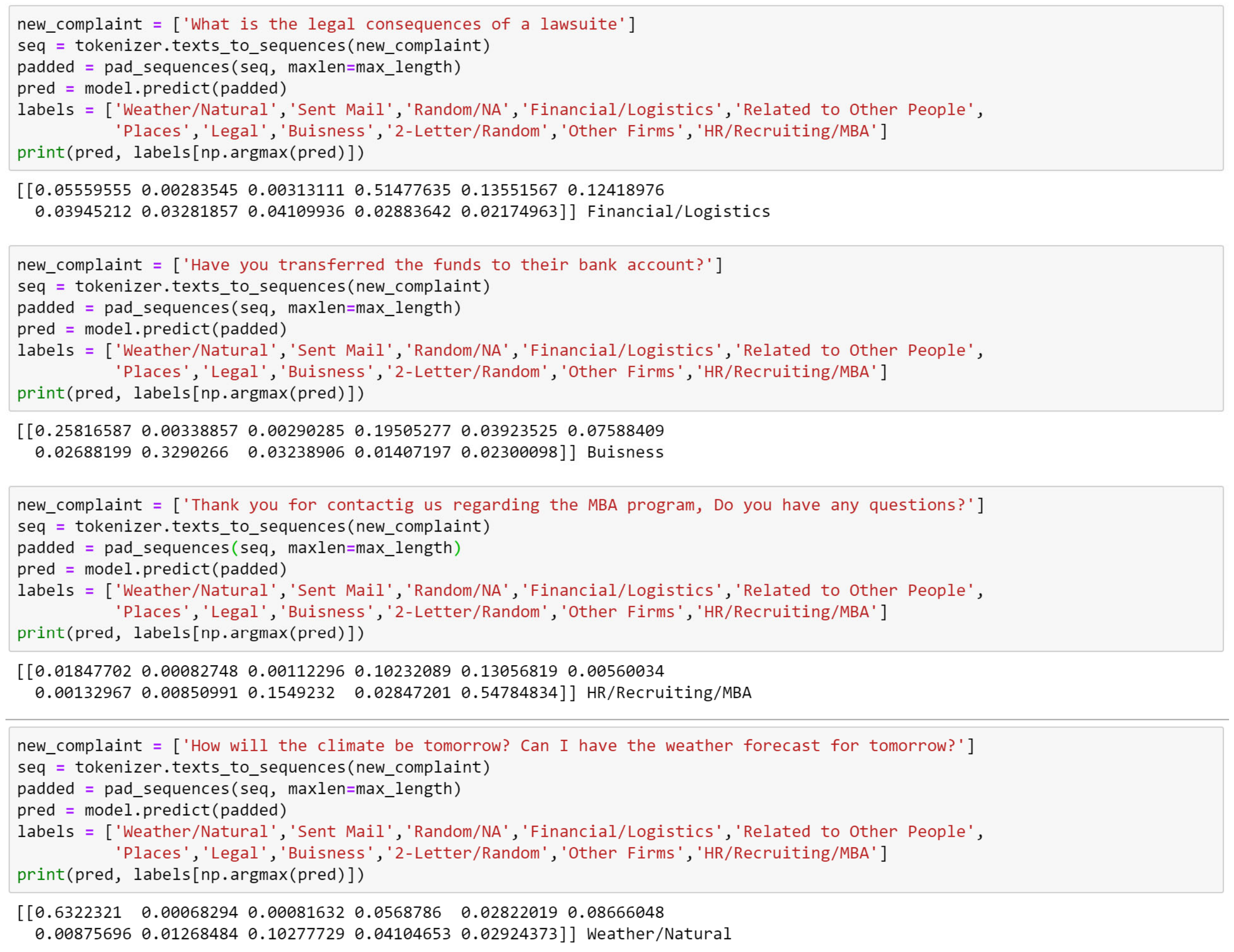
We fixed the number of words to take to generate the embedding as well as the input for prediction to 250 characters. We could do longer but for the purposes of this project we stuck with 250. The embedding was taken arbitrarily to 100 dimensions. With these values set, we use the tokenizer class from Keras to vectorize the message feature of our dataset (as we did in one of the labs in the class). Under the hood, this basically turns text into a specific sequence of integers. This fitted tokenizer was saved using pickle since it will be used later for inference.

For LSTM's, it is good practice to one hot encode the target if we are dealing with a multi-class classifier. Hence, the label was one hot encoded resulting in 11 more features. Also, we implemented a Min-Max scaler for numerical features since neural networks work better and faster in normalized data. It is also good practice to create a normalizer model in the test data and apply it to all the features during inference. This methodology was also implemented. The final dataset consisted in the features message length, the 250-vector representation, and the day of the week (one hot encoded). Afterwards we used a standard stratified test_train_split() using scikit-learn. Using stratified splitting is important to ensure both the training and testing are representative of the dataset. Like traffic predictions, some emails tend to be urgent if sent in non-working hours during the week and especially in the weekend. One way to capture this trend and differentiating from 9 PM on a Monday to 9 pm on a Saturday is to implement feature crosses. This was implemented using TensorFlow.

After having our final dataset, we used Keras deep learning library running in Tensorflow. Our first approach was to use all features but the strategy for feeding it into the model was different from having all inputs go to the same layer. We separated the dataset into the embedded dataset (the vector representation of the message) and a numerical/categorial dataset (message length, day of the week, hour, etc). The embedded dataset was feed into an LSTM layer with 100 units, a dropout of 0.2, and a recurrent dropout of 0.2. After the LSTM layer a Dense layer with 11 units was inserted with a SoftMax function since we are modelling the problem as probabilities (What is the probability that this message belongs to 1, 2, 3, 4, …, 11?). For the numerical dataset, I passed the input into a 64 (ReLu Activated), 32 (ReLU Activated), 11 (SoftMax activated) neural network. We concatenate both neural networks by passing both SoftMax outputs to another 11-unit SoftMax activated layer. As a loss we use categorical cross entropy, and as our metric accuracy. A picture of our compiled model can be found in Figure 2A. The model did not perform very well but by implementing L1 regularization we found out that extra features we're reducing our accuracy. Based in this information I decided to train the LSTM neural network with the message embedding only. This boosted our accuracy to 0.76 for the train set and 0.60 for the test dataset.



In the following image you can see how the model correctly and incorrectly classifies text. Clearly some optimization has to be done but for the purposes of this class, the model seems to be working great by itself. It manages to capture the escence of the message and classify it correctly.



```
new_complaint = ['What is the legal consequences of a lawsuite']
seq = tokenizer.texts_to_sequences(new_complaint)
padded = pad_sequences(seq, maxlen=max_length)
pred = model.predict(padded)
labels = ['Weather/Natural','Sent Mail','Random/NA','Financial/Logistics','Related to Other People',
          'Places','Legal','Buisness','2-Letter/Random','Other Firms','HR/Recruiting/MBA']
print(pred, labels[np.argmax(pred)])
```

```
[[0.05559555 0.00283545 0.00313111 0.51477635 0.13551567 0.12418976
  0.03945212 0.03281857 0.04109936 0.02883642 0.02174963]] Financial/Logistics
```

```
new_complaint = ['Have you transferred the funds to their bank account?']
seq = tokenizer.texts_to_sequences(new_complaint)
padded = pad_sequences(seq, maxlen=max_length)
pred = model.predict(padded)
labels = ['Weather/Natural','Sent Mail','Random/NA','Financial/Logistics','Related to Other People',
          'Places','Legal','Buisness','2-Letter/Random','Other Firms','HR/Recruiting/MBA']
print(pred, labels[np.argmax(pred)])
```

```
[[0.25816587 0.00338857 0.00290285 0.19505277 0.03923525 0.07588409
  0.02688199 0.3290266  0.03238906 0.01407197 0.02300098]] Buisness
```

```
new_complaint = ['Thank you for contactig us regarding the MBA program, Do you have any questions?']
seq = tokenizer.texts_to_sequences(new_complaint)
padded = pad_sequences(seq, maxlen=max_length)
pred = model.predict(padded)
labels = ['Weather/Natural','Sent Mail','Random/NA','Financial/Logistics','Related to Other People',
          'Places','Legal','Buisness','2-Letter/Random','Other Firms','HR/Recruiting/MBA']
print(pred, labels[np.argmax(pred)])
```

```
[[0.01847702 0.00082748 0.00112296 0.10232089 0.13056819 0.00560034
  0.00132967 0.00850991 0.1549232  0.02847201 0.54784834]] HR/Recruiting/MBA
```

```
new_complaint = ['How will the climate be tomorrow? Can I have the weather forecast for tomorrow?']
seq = tokenizer.texts_to_sequences(new_complaint)
padded = pad_sequences(seq, maxlen=max_length)
pred = model.predict(padded)
labels = ['Weather/Natural','Sent Mail','Random/NA','Financial/Logistics','Related to Other People',
          'Places','Legal','Buisness','2-Letter/Random','Other Firms','HR/Recruiting/MBA']
print(pred, labels[np.argmax(pred)])
```

```
[[0.6322321  0.00068294 0.00081632 0.0568786  0.02822019 0.08666048
  0.00875696 0.01268484 0.10277729 0.04104653 0.02924373]] Weather/Natural
```

I also tried to perform Doc2Vec followed by logistic regression since when doing literature research I found out that it boosts accuracy in LSTM models but it was difficult to implement since the dataset big size meant big memory requirements when fitting the embedding. The computational cost was enough reason to abandon this approach but the notebook is attached in the documentaiton as proof.

Conclusions

Going into the project, I did not know how we would solve this problem. There are several methods and models out there that have been tested for everything but at the end, every problem is different and it will depend on the goal. Our goal was to classify emails (multi-class classification) according to their content. While researching, I learned that several binary classifiers can actually be used for multi-class classification. For example, in logistic regression we can use the sigmod function to predict probabilities of the email belonging to class 1 or others. Then we implement the same modeling but this time calculating the probability it belongs to class 2 or others and so on. Basically for this problem we had 11 classes therefore 11 models and we could just picked the model that produces the highest probability and choose as the overal predicted class. However, we found out that there are serveral issues. Scaling is different for each model therefore introducing bias.

We initially decided to use an ensemble method using majority vote. This meant combining all our models to create a stronger one. Our original objective changed when we couldn't boost the models accuracy for decision trees and KNN clustering beyond 40%. Ensembling this models with a clearly stronger model (the LSTM) could actually damage the better model.

The KNN model built by Rajan, was sensitive to different embedding techniques and also not enough cluster separation was observed resulting in a week model. The differences in embedding technique may also had an impact. For the message, the class was Tokenizer (since it was the most recommended class for LSTM models), while we used Word2Vec for embedding the folders before clustering into labels. Also, with 1,700+ different folders and a significant amount of overlap, the label clusters were not as disjoint as we had expected. We therefore opt to use neural networks only and specifically the LSTM. As for the activation funciton, we also learned that a softmax function needed to be implemented since the output can know be interpreted as probabilities.

For the LSTM, we got good results with accuracies on the training set of approximatly 0.90. One clear issue that arise during training was overfitting. The LSTM got a 0.68 accuracy score in the test dataset. While it peforms well in classifiying "obvious" text, it has difficulties labeling properly emails with more complex contexts. This peformance could probably be enhance if we provide the model with cleaner, more consice, email messages.

If optimize further, this model could have a real impact world by making peoples inboxes more ordered and therefore more efficient. An optimized model could be built into an app, widget or extension for people around the world to use to have their emails sorted by topic (within the limits of the model).

Future work includes expanding the model to a multi-class multi-label classification problem since many of the emails can belong to not only one, but two or three classes. For this, we need to use something like sigmoid cross entropy with logits but it will probably result in a computationally expensive approach. After doing more research, and accroding to Google, these type of problems could also employ Candidate Sampling which according to them "calculates for all the positive labels but only for a random sample of negatives" or noise_contrastive which "approximates the denomiatro of softmax by modeling the distribuion of outputs".

Appendix

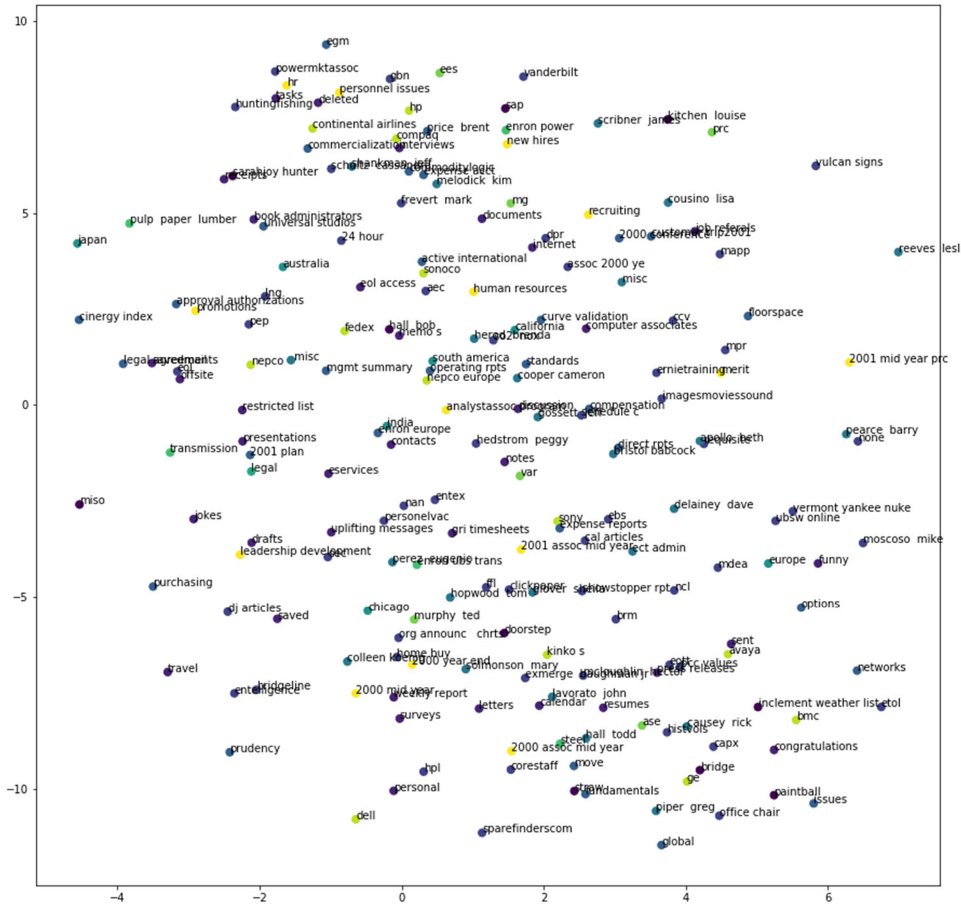
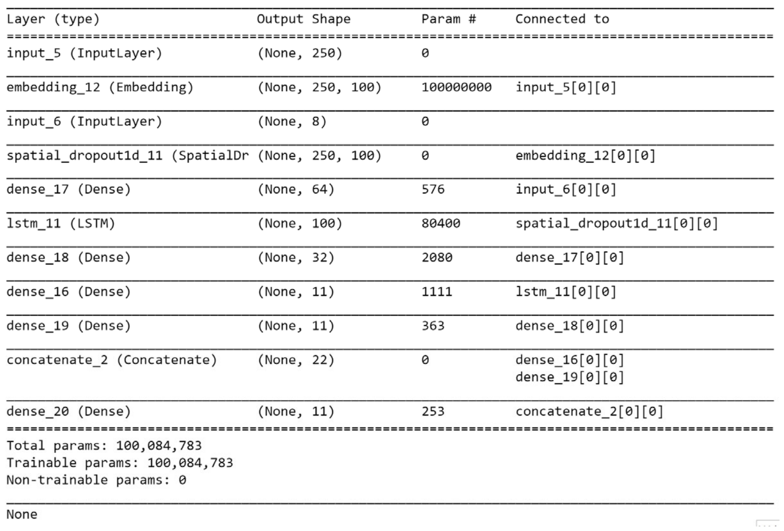


Figure A1: Label Clustering for further Reduction



```
__________________________________________________________________________________________________
Layer (type)                    Output Shape         Param #     Connected to
==================================================================================================
input_5 (InputLayer)            (None, 250)          0
__________________________________________________________________________________________________
embedding_12 (Embedding)        (None, 250, 100)     100000000   input_5[0][0]
__________________________________________________________________________________________________
input_6 (InputLayer)            (None, 8)            0
__________________________________________________________________________________________________
spatial_dropout1d_11 (SpatialDr (None, 250, 100)     0           embedding_12[0][0]
__________________________________________________________________________________________________
dense_17 (Dense)                (None, 64)           576         input_6[0][0]
__________________________________________________________________________________________________
lstm_11 (LSTM)                  (None, 100)          80400       spatial_dropout1d_11[0][0]
__________________________________________________________________________________________________
dense_18 (Dense)                (None, 32)           2080        dense_17[0][0]
__________________________________________________________________________________________________
dense_16 (Dense)                (None, 11)           1111        lstm_11[0][0]
__________________________________________________________________________________________________
dense_19 (Dense)                (None, 11)           363         dense_18[0][0]
__________________________________________________________________________________________________
concatenate_2 (Concatenate)     (None, 22)           0           dense_16[0][0]
                                                                 dense_19[0][0]
__________________________________________________________________________________________________
dense_20 (Dense)                (None, 11)           253         concatenate_2[0][0]
==================================================================================================
Total params: 100,084,783
Trainable params: 100,084,783
Non-trainable params: 0
__________________________________________________________________________________________________
None
```

Figure A2: LSTM Model Summary