Machine Learning, Toxic Comment Classifier, NLP project

Mohammadmehdi Ezzatabadipour

February 21, 2018

1 Definition

1.1 Project Overview

A pillar of free democracy is the right to express your opinions, share your thoughts and have a constructive contribution to develope a safe place for every body to practice his/her rights in this regard. However, behind the shield of computers some people also think they can abuse and harass other people opinions and characters. Such online act of harrasments suppress so many of our fellow citizens to express their opinions. According to Huffington post, such misbehavior causes the constnat drop on the digital activity [1]. The [Conversation AI team][2], a research initiative founded by [Jigsaw][3], and Google (both a part of Alphabet) are working on tools to help improve online conversation. One of the aspects of their efforts is aiming to identify the toxic comments and lunch online toxicity monitoring system on the various of online social platforms. In the joint effort with Kaggle, they define the project as a contest [toxic comment classification challenge][4], with the prize of 35,000. Although the goal of challenge is developing a multi-label classifier, not only identify the toxic comments but also detect the type of toxicity such as **threats , **obscenity , **insults , and **identity-based hate, but I simplified the challenge for myself to a binray classification whether the comment is toxic or not. I wanted to start with the mono-label supervised classification NLP challenge.

The primary dataset is provided in [Kaggle website][5]. It is collected from a large datasets of Wikipedia'talk page edits containing 561809 number of comments which are rated by human-raters under the six classes of **toxic**, **severe toxic**, **obscene**, **threat**, **insult** and **identity hate**. Each input is basically is an online comment (which varies within from 0 to 200 words or more with average of 60 words in each), then it follows by 6 mentioned labels (each is either 0 or 1). Again, I simplified the project from multi-label classifier to mono-label classifier by removing the other labels but toxic. I use 80% of inputs as training data and then another 10% as validation and the last 10% as testing set. I briefly decribed the type of preprocessing (word vectorizations, bag of words/Tf-idf/embedding) as a way to have a mathematical description of data. The label is treated as a binary classification. I use 80% of 561809 number of comments as my training set and 10 percents as validation and 10 percents as testing set. I use test-train-split function from scikit-learn to split my data set to different classes. As I use multinaive base identifier in my benckmark which does not need validation set, I use the 80 percents training data and the then I use the 10 percents testing set. It makes it feasible to compare the results from becnhmark to the solution strategy.

[1] https://www.huffingtonpost.com/entry/online-harassment-impacts-majority-of-adults-finding_us_59653114e4b09be68c0055e2

[2] https://conversationai.github.io/

[3] https://jigsaw.google.com/

[4] https://www.kaggle.com/c/jigsaw-toxic-comment-classification-challenge

[5] https://www.kaggle.com/c/jigsaw-toxic-comment-classification-challenge/data

1.2 Motivation

My personal motivation is familiarizing myself with Natural language processing (a.k.a NLP) techniques. Vectorizing the words, exploring the semantic meaning and building up a mathematical inference of the contexts of comments has been my long interest. Meanwehile, I am curious how the recent developements in Machine Learning (a.k.a ML) such as RNN which has been proven to be quite sucessful in predicting time series can help NLP. As a physicisit, I see very similarities between a time series and a comment or review. The sequence of the words and the the occurrence of group of words at specific contexts are so similar to patterns observed in a time series. I like to test my perception using RNN in toxic comment classification and compare the performace with popular benchmarks in NLP.

1.3 Problem Statement

Regarding the first section, toxic comments and cyber harrasments is a major a growing concern non-only within the social media and other online platforms. With growing number of users and huge number of comments, we cenrtainly can not expect to have enough actual human raters to identify the toxicity of the comments. Although it is easy for human to classify them but it can not simply just cover the growing demands. An automated method specially using ML (Machine learnig algorithms) which can be trained over provided data set classified by human raters and further use to infer the toxic comments is one of the solutions. As deep neural netowrks (a.k.a DNN) has been proven to be quite powerful to come up with higher order paramters tabbing on grouping the original elements (pixels in pictures, tokenized words in this task) to infer the labels of comments, I use LSTM-RNN as one of the very recent advances in the field to learn the sequential relationship between choice of vocabulary in order to classify the toxic and non-toxic comments.

1.4 Metrics

- **Confusion Matrix** [6] It is very informative performance measures for classification tasks. $C_{i,j}$ an element of matrix tells how many of items with label i are classified as label j. Ideally we are looking for diagonal Confusion matrix where no item is miss-classified. The matrix below is a good representation for our binary classification. Positive (P) represents toxic label and n (negative) represents non-toxic label.

prediction outcome

		p	**n**	**total**
actual value	**p′**	TP = True positive	FN = False negative	P′
	n′	FP = False positive	TN = True negative	N′
	total	P	N	

Confusion Matrix

[6] http://scikit-learn.org/stable/modules/generated/sklearn.metrics.confusion_matrix.html

- **Accuracy:** This metric measures how many of the comments are labeled correctly. However in our data set where most of comments are not toxic, regardless of performance of model, we get high accuracy. According to confusion matrix:

$$Accuracy := \frac{TP + TN}{\mathrm{N}' + \mathrm{P}'} \tag{1}$$

- **Precision and Recall:** [7] Precision and recall in our case are designed to measure the performance of the model in classifying the toxic comments. *Precision* tells us what fraction of classified comments as toxic are truely toxic and *Recall* measures what frction of toxic comments are labeled correctly.

$$Precision := \frac{TP}{P} \tag{2}$$

$$Recall := \frac{TP}{\mathrm{P}'} \tag{3}$$

$$\tag{4}$$

- **f**β Score: [8] Both precision and recall matter to check the performanc eof the model. Having one metric in which combines them together is quite informative. By setting $\beta = 1$, it returns the harmonic mean of precision and recall.

$$F_\beta = (1 + \beta^2) \cdot \frac{\text{precision} \cdot \text{recall}}{(\beta^2 \cdot \text{precision}) + \text{recall}} \tag{5}$$

[7] http://scikit-learn.org/stable/auto_examples/model_selection/plot_precision_recall.html
[8] http://scikit-learn.org/stable/modules/generated/sklearn.metrics.fbeta_score.html

2 Analysis

2.1 Data Exploration and Visualization

In the training section, we are dealing with comments which have been already labeled as toxic and non-toxic. Fig 1. From the inference we learn in training data, a classifier will be built for non-labeled comments.

Figure 1: Comment sample of data

The methods will be applied through the course of this paper is an effort for semantic inference of the toxicity of online comments. This section is devoted to descriptive statical analysis before using the luxary of Machine learning. In short, we are looking for a statistical fingerprint of toxic comments in this section. In order to have a comparative statistical analysis, the toxic comments are seprated from non-toxic comments. The ntoicable fact is that the dataset is quite imbalanced. Out of **5172078** number of comments in the training set, there are **4774994** non-toxic comments and there are just about 7% of the comments are toxic **397084** (Fig. 2). We are certainly dealing with imbalanced data set.

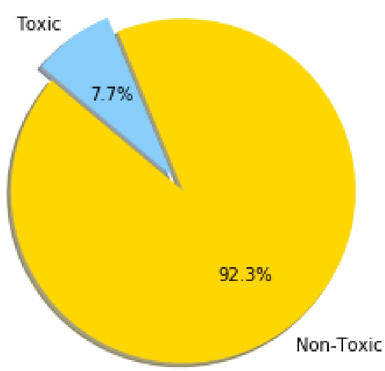


Figure 2: Imbalaced distribution of labels

In a well written article, **Learning from imbalanced classes**[9] some methods are suggested to handle

[9] https://www.svds.com/learning-imbalanced-classes/

imbalanced classes. After cleaning data [check preprocessing subsection], the length histogram (Fig. 3):

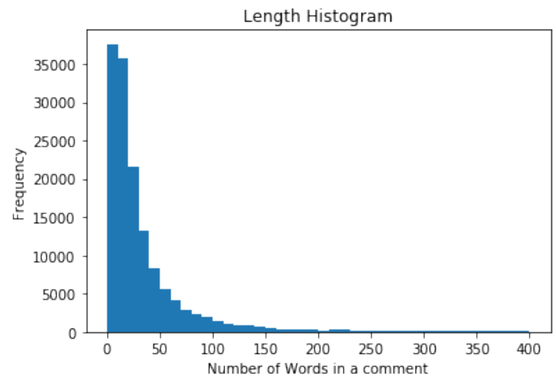
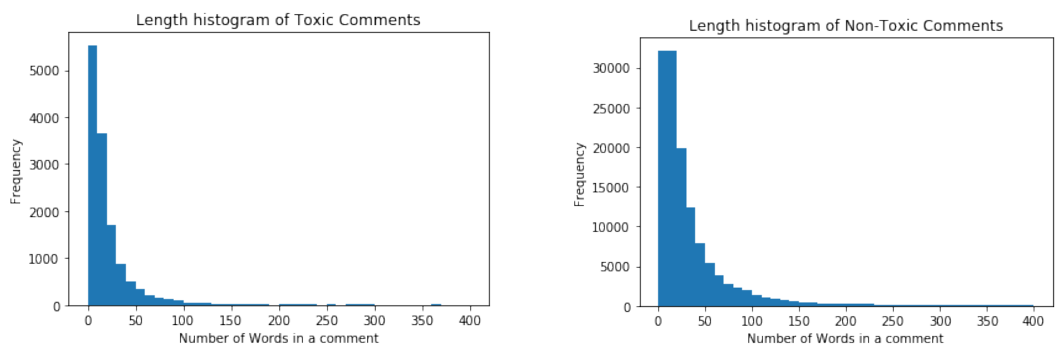


Figure 3: Unlabeled comment length histogram



(a) Toxic

(b) Non-Toxic

Figure 4: Comment length histogram versus labels

As you see in Fig 4, the comment length distribution is not label specific, however, the higher length is more frequent for **Non-Toxic comments**, (tail is thicker).

2.2 Pre-analysis of Term Frequency

The other desciptive analysis is finding the most frequenct words which are used across the labels. If the frequently used vocabulary is different, then **term frequency (a.k.a tf)** and more advanced technique **tf-idf (term frequency inverse document frequency)**[10] are very good choices as a classifier.

```
                Term Frequency
Non—Toxic Comments              Toxic Comments
('article', 65271)          ('fuck', 10357)
('page', 49256)             ('suck', 4433)
('wikipedia', 40244)        ('go', 3462)
('talk', 34863)             ('wikipedia', 3411)
('use', 28817)              ('shit', 3399)
('one', 26354)              ('like', 3358)
('please', 26157)           ('u', 3170)
('make', 25582)             ('nigger', 3035)
('would', 25505)            ('get', 2773)
('edit', 24758)             ('page', 2314)
('like', 22521)             ('know', 2248)
('see', 21981)              ('gay', 2175)
('say', 21278)              ('bitch', 2156)
('think', 20789)            ('hate', 2151)
('source', 20577)           ('die', 2107)
('know', 19346)             ('faggot', 2005)
('also', 17918)             ('moron', 1896)
('add', 17097)              ('make', 1801)
('get', 16974)              ('fucking', 1764)
```

2.3 Algorithms and Techniques

Long short term memory recurrent neural network (a.k.a [LSTM-RNN] [11]) which is a recent branch of deep neural network (DNN) is applied to identify the toxic comments. DNN and its applications in NLP are found to be quite clever for constructing higher order paramters which are very vital for a specific classification. In this application, the higher order paramter can be a sequence of words which are label specific. LSTM-RNN treats the comment as set the vectorized words like a time series and trying to learn how the words are aligned in a time series attributed to a specific label. The performance of the model will be tested against the benchmark model. The detailed discussions about the benchmark classifier and prepossessing steps are described in next sessions. I opened a new subsection to discuss more about [LSTM-RNN] in details Subsection 2.4.

Briefly, I descibe the various steps has been taken until the final classfication. The output of each section is the input of next section. It is a sequential modeling. The inspiration for this architecture is coming from a kaggle kernel for this project [12].

[10] http://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.text.TfidfVectorizer.html
[11] http://www.wildml.com/2015/09/recurrent-neural-networks-tutorial-part-1-introduction-to-rnns/
[12] https://www.kaggle.com/sbongo/for-beginners-tackling-toxic-using-keras

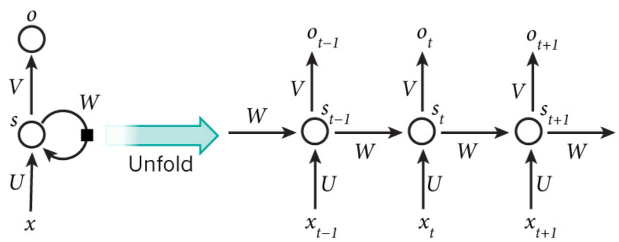


Figure 5: Simple Illustration of RNN Architecture

1. **Preprocessing**:
   In this section, NLTK [13] (natural language toolkit library) is used. The input of this workflow is a comment. This process breaks each comment into sentences, then break each sentence into words. Further it removes the stop words from our corpus (commonly used vocabulary). The token words are finally lammatized to their meaningfull roots. This section is in common between both benchmark model and our solution model.

2. **Vectorization, Embedding**:
   [14]: In this section each word is transformed to a vector[15] by using a technique called **Embedding**. It leaves a paramter for tunning, the size of vector which represents each word.

3. **Padding**: This size of each comment is fixed across all the comments. It leaves a parameter for tunning **fixed comment size**.

4. **LSTM-RNN architecture**:
   After padding, each comment which is a list of vectors are fed to LSTM-RNN architecture. The output is a list of vectors with reduced size. The reduced size of is matter of tunning for classification.

5. **Global Maxpooling**:
   By now each comment is a list of vectors with reduced size, however, using max pooling in each dimension of the vector, the maximum value is picked up. So the entire comment is represented as a single vector.

6. **Hidden Layer of Neural Network**:
   The elements of the vector are fed into hidden neural layer with the same size.

7. **Classifier Perceptron**: The outscome of previous hidden are all fed to to final perceptron with **sigmoid** activation function.

[13] http://www.nltk.org/
[14] https://www.tensorflow.org/tutorials/word2vec
[15] https://www.tensorflow.org/tutorials/word2vec

2.4 More discussions on LSTM-RNN and its application in NLP

If we treat a sentence or comment like a time-series, the natural choice for us is using RNN (Recurrent neural network). However, for long time series and more its applications in NLP, LSTM is highly preferred above RNN. The achilies heel of so many ML technqiues is a problematic error called **Valinshing Gradient** [16]. It shows up when there exists series of neural layers and backpropagation is used to train the weights. Most of the time, due to chain rule and the fact that the inverse of activiation functions are most of the time is less than one, the early layers are pruned to be treated with very small portion of gradient decents. As a result, those early layers or deeper layers from the solution, are not trained. RNN (NLP tasks) treats the first, second ,... words as earlier nodes, as a result of Vanishing Gradient, those begining words dont contribute as much as later words for NLP tasks.In order to solve that problem, LSTM is designed in a way to have a more capacity for remembering long term memories. The idea behind LSTM is in fact simple. Rather than each hidden node being simply a node with a single activation function, each node is a memory cell that can store other information. Specifically, it maintains its own cell state. Normal RNNs take in their previous hidden state and the current input, and output a new hidden state. An LSTM does the same, except it also takes in its old cell state and outputs its new cell state [17]. LSTM has a way to control the flow information through its hidden nodes. Each LSTM memory cell has three subdevisions: Fig 6

1. **Forgot Gate** It decides what portion of information in needed from previous node.
2. **Input Gate** It update the memory cell based on new input and using the previous information.
3. **Output Gate** It decides what information is needed to pass to the next node.

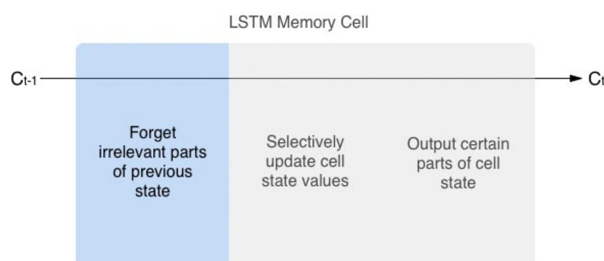


Figure 6: LSTM Memory Cell

I also got some of my inspirations from the discussions about the promissing techniques in NLP made in this article. [18].

2.5 Benchmark

- **Naive Classifier**: According to Fig. 1, more than 90% of the comments are **non-toxic**. As a very naive classifier, we can assume an arbitrary comment will be **non-toxic** as well. Based on that, we can measure our metrics and use it as a reference for the other classifiers.

  Based on the values of confusion matrix, according to the section (metric), the other metrics can be measured.

[16] https://en.wikipedia.org/wiki/Vanishing_gradient_problem
[17] http://harinisuresh.com/2016/10/09/lstms/
[18] http://ruder.io/deep-learning-nlp-best-practices/index.html

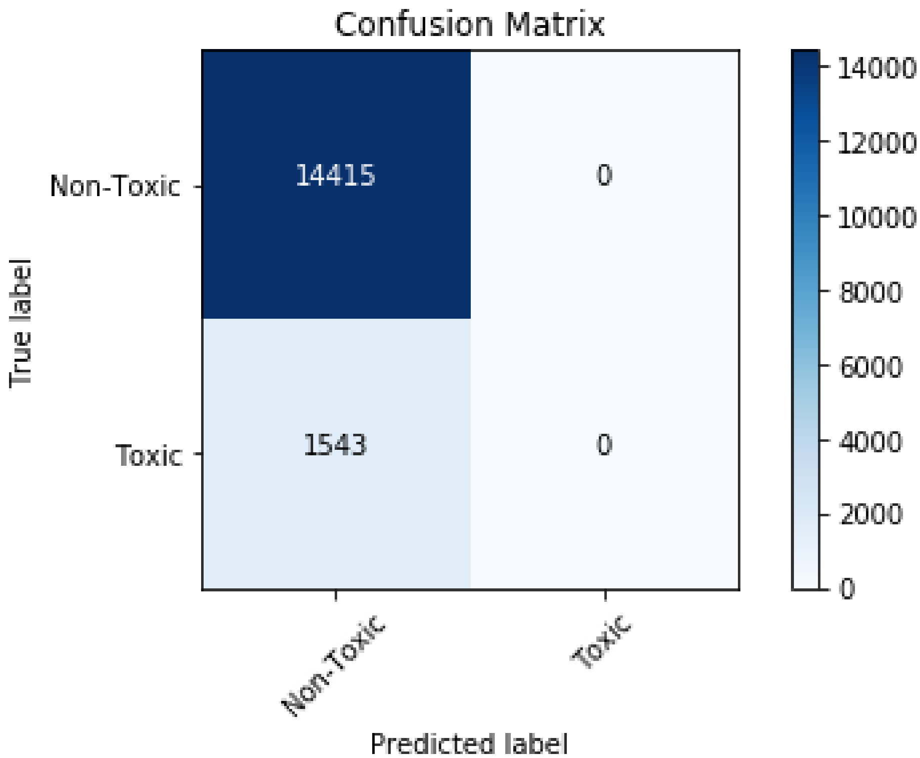


Figure 7: Confusion Matrix for **Naive Classifier**

Metrics	Value
Accuracy	0.903
Precision	0.0
Recall	0.0
F_1	0.0

Table 1: Metrics for **Naive Classifier**

- **tf-idf+MultinomialNB**: The distinctive results from term frequency shows that the classifier which is utilizing the term frequency and specially those using **tf-idf** can provide a very promising results. After preprocessing and cleaning data, scikit-learn library "TfidfVectroizer" [19] is used to tranform the texts into **tf-idf** features. As a classifier, I choose Multnomial naive bayes (MultinomialNB) classifier from scikit-learn libraries [20].

  Based on the values of confusion matrix, according to the section (metric), the other metrics can be measured.

[19] `http://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.text.TfidfVectorizer.html`
[20] `http://scikit-learn.org/stable/modules/generated/sklearn.naive_bayes.MultinomialNB.html`

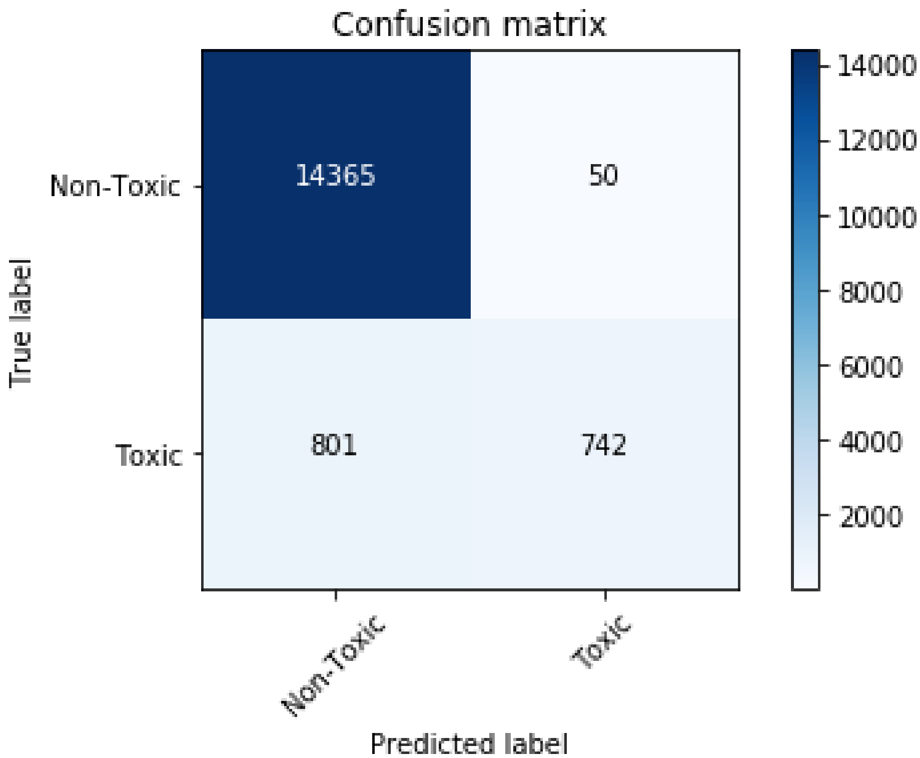


Figure 8: Confusion Matrix for **tf-idf+MultinomialNB**

Metrics	Value
Accuracy	0.947
Precision	0.937
Recall	0.481
F_1	0.636

Table 2: Metrics for **tf-idf+MultinomialNB**

3 Methodology

3.1 Programming Language and Libraries

- **Python 2.**
- **scikit-learn**. Open source machine learning library for Python.
- **Keras.** Open source neural network library written in Python. It is capable of running on top of either Tensorflow or Theano.
- **TensorFlow.** Open source software libraries for deep learning.

3.2 Preprocessing

In this section, I mainly use NLTK[21] (natural language toolkit library). The input of this workflow is a comment. This process breaks each comment into sentences, then break each sentence into words. Further it removes the stop words from our corpus. In order to simplify the rest of process, I lemmatize the words.

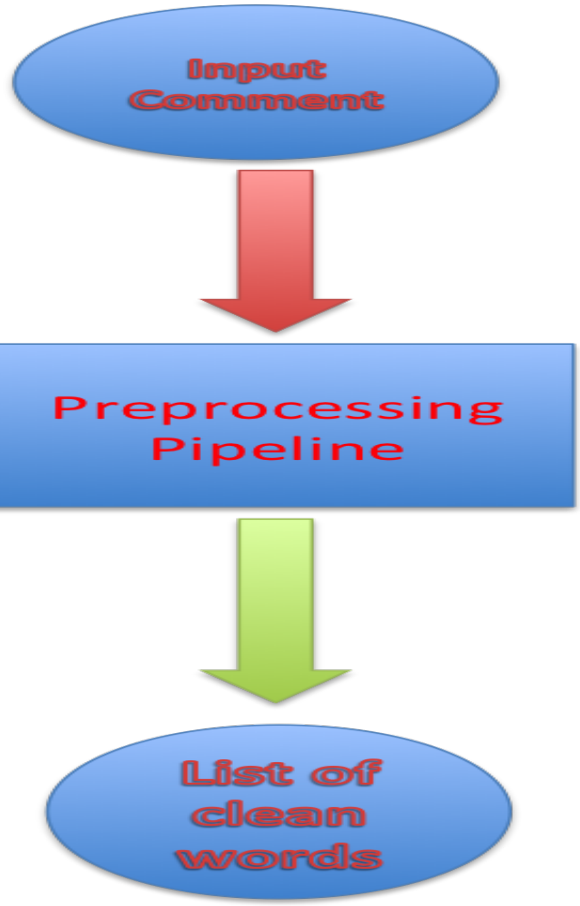


Figure 9: Preprocessing Workflow

```
class NLTKPreprocesor(BaseEstimator,TransformerMixin):
    def __init__(self,stopwords = None,punct = None,lower = True,strip=True):
        self.lower = lower
        self.strip = strip
        self.stopwords = stopwords or set(sw.words('english'))
        self.punct = punct or set(string.punctuation)
        self.lemmatizer = WordNetLemmatizer()
#          self.contractions = load(open('contractions.pickle','rb'))

    def fit(self,X,y=None):
```

[21] http://www.nltk.org/

```
        return self

    def inverse_transform(self,X):
        pass

    def transform(self,X):
        return [list(self.tokenize(doc)) for doc in X]

    def tokenize(self,sDocument):
        document=sDocument.decode('utf-8')
#        doc.strip(" ")
        for sent in sent_tokenize(document):
            for token,tag in pos_tag(wordpunct_tokenize(sent)):
                token = token.lower() if self.lower else token
                token = token.strip() if self.strip else token
                token = token.strip('_') if self.strip else token
                token = token.strip('*') if self.strip else token
                token = token.strip('#') if self.strip else token

                if token in self.stopwords:
                    continue

                if all(char in self.punct for char in token):
                    continue

                if len(token) <= 0:
                    continue

                lemma = self.lemmatize(token,tag)
                yield lemma

    def lemmatize(self,token,tag):
        tag ={
            'N' : wn.NOUN,
            'V' : wn.VERB,
            'R' : wn.ADV,
            'J' : wn.ADJ
        }.get(tag[0],wn.NOUN)
        return self.lemmatizer.lemmatize(token,tag)
```

3.3 Implementation

3.3.1 Vectorization, Embedding and Padding

By now, each comment turns into a clean list of lematized words. By exploring the vocabulary inside my training data, we create a bag of words, depending on its frequency, each word get a numeric index. From this step, the workflow bifurcates for the benchmark model versus our solution model.

- **Solution:** After creating bag of words, we use a technique called Embedding[22] for **word2vector**.

[22] https://www.tensorflow.org/tutorials/word2vec

Each word will be represented with a vector of arbitrary size, The similarity (For example: cosine similarity) between these new vectors quantifies the semantic similarity between their corresponding words.

- **Benchmark:** We have slighlty easier task, after creating bag of words with the corresponding indices, we start measuring the **tf** (term frequency) versus each comment. The outcome is a matrix; rows represent comments and columns represent terms (our vocabulary).

Along our solution workflow, the next step is padding the comments. As it is expected, the comments are coming with different lengths and sizes, we set the size of vector in which represents a comment to be fixed across all comments. If the comment is shorter, the extra elements are zeros. In Fig. 8, we can get an idea, what it should be the fixed size of vector representation of each comment. 100 is very safe choice. However, it might make the entire algorithm quite time consuming.

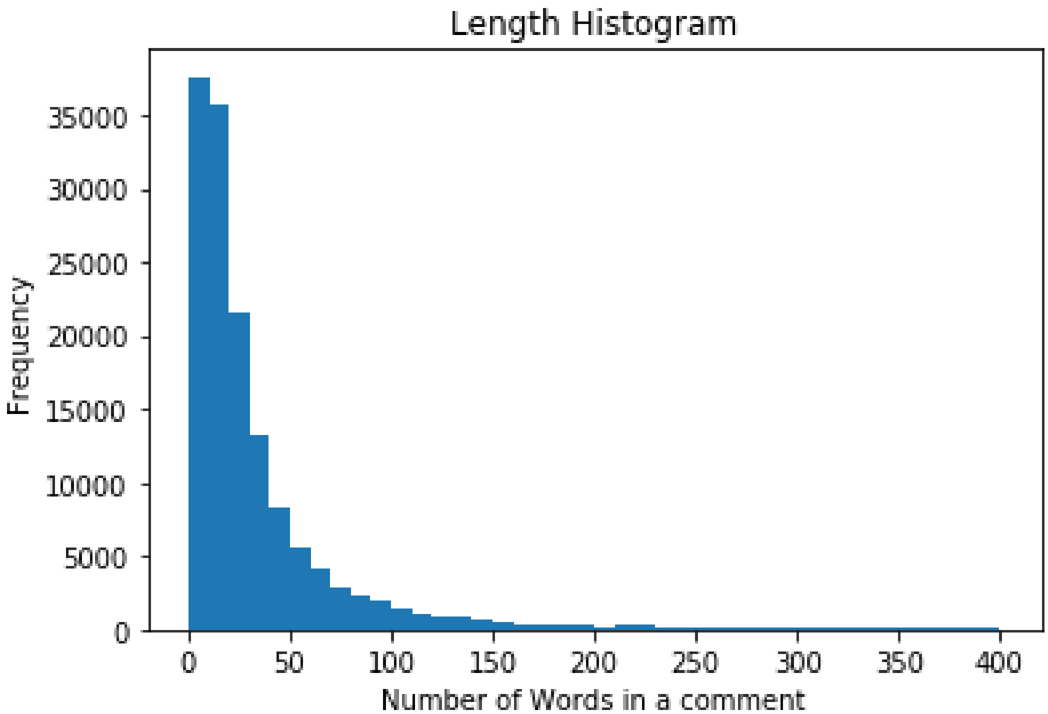


Figure 10: Histogram of Number of words per comment

3.3.2 Final Neural Netowrk Architecture

By now, each comment which is now transformed to a series of constnat number of vectors are ready to be passed like a time series to the choice of LSTM-RNN architecture. In order to explain, I make some assumptions: Let say if we make a decision for the lenght of each comment to be 100, and in the embedding section, we decide that each word will be represented with vector size of 128.

```
"This is my final architecture to create my classification"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_1 (Embedding)      (None, 100, 128)          2560000
_________________________________________________________________
lstm_layer (LSTM)            (None, 100, 60)           45360
_________________________________________________________________
global_max_pooling1d_1 (Glob (None, 60)                0
_________________________________________________________________
dropout_1 (Dropout)          (None, 60)                0
_________________________________________________________________
dense_1 (Dense)              (None, 50)                3050
_________________________________________________________________
dropout_2 (Dropout)          (None, 50)                0
_________________________________________________________________
dense_2 (Dense)              (None, 1)                 51
=================================================================
Total params: 2,608,461.0
Trainable params: 2,608,461.0
Non-trainable params: 0.0
_________________________________________________________________
```

As you might see in the architecture above and also Fig. 3, the input of each LSTM layer is a matrix of size (100,128) which will be treated like a time-series of size (100) inside LSTM and the output of LSTM will be a vector of size (100,60). The way LSTM works, it recieves the vector of each time step (from 1 to 100) and tries to predicts the next coming input, then learns the inner memory between the each time step. The number of folds are 100 in our case.

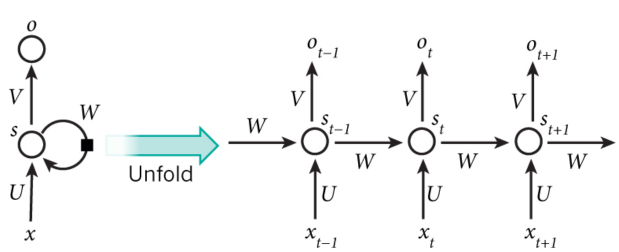


Figure 11: Simple Illustration of RNN Architecture [23]

We use max-pooling in order to reduce the dimention of the ouput of LSTM from (100,60) to a vector with size of (60). And then we pass the elements of the vector as input to hidden neural layer with 50 perceptrons. Finally, the output of hidden layer will be passed to last classifier layer. We incorporate **dropout** to avoid overfitting. Again, my final architecture might be slightly different from this architecture.

3.3.3 Implementation Challenges

Primarily I wanted to push the entire project inside the pipeline. Despite of a lot of efforts, I eventually made it work. However, I found it quite hard to debug and monitor the intermidiate outputs before final products (Extracting the intermediate results). Eventually, I gave up and design my machineary as step by step jupyter notebook cells instead of one synchronized pipeline. You can see some of my pipeline footprints specially in the preprocessign section. Regardless, still using the pipeline is so fascinating but not in the developement section.
As you noticed, I am using number of python libraries including sklearn, NLTK and keras. Some common techniques like term frequency, tokenizing are implemented in both **NLTK** and **scikit-learn**. I put a great deal of thinking which one I should use for what purpose. Eventually, I use **NLTK** for cleaning (preprocessing section like excluding stopwords, lemmatizing ...) and **scikit-learn** for **tf-idf, Multinomial NB and metrics**.
Although, I was so excited about using **LSTM-RNN**, however the primarily results were quite disappointing. it even made me think that the huge machinary of LSTM is like a overfitting section as a whole for this task. No need to mention that LSTM techniques are quite **time consuming**, it becomes a major issue when you are at hyper tunning section.

3.4 Refinement

In the beginning, I underestimated the importance of maximum number of features (the size of the vocabulary for bag of words). The right size of vocabulary has a importance specially for becnhmark model. My primary solution was $size_{vocabulary} = 5000$ with very poor performance for our metrics. Finally, I increase it to $size_{vocabulary} = 20000$ to be fixed accross my both benchmark model and my solution model.

Metrics	Value
Accuracy	0.920
Precision	0.993
Recall	0.176
F_1	0.298

Table 3: Metrics for **primary tf-idf Naive Classifier**

In the primary design of my solution model, the dropouts sections were missing. The padding size for each comment was smaller (50). The lemmatization section was not included in preprocessing section. Lemmatization helps to reach better performance without unnessary extension of vocabulary. Later I changed the padding to 100 to cover better context of the comments (Fig. 8).

```
"This is my primary architecture to create my classification"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_1 (Embedding)      (None, 50, 128)          640000
_________________________________________________________________
lstm_layer (LSTM)            (None, 50, 60)           45360
_________________________________________________________________
global_max_pooling1d_1 (Glob (None, 60)                0
_________________________________________________________________
dense_1 (Dense)              (None, 50)                3050
_________________________________________________________________
dense_2 (Dense)              (None, 1)                 51
=================================================================
_________________________________________________________________
```

```
"This is my intermediate architecture to create my classification"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_1 (Embedding)      (None, 100, 128)          640000
_________________________________________________________________
lstm_layer (LSTM)            (None, 100, 60)           45360
_________________________________________________________________
global_max_pooling1d_1 (Glob (None, 60)                0
_________________________________________________________________
dense_1 (Dense)              (None, 50)                3050
_________________________________________________________________
dense_2 (Dense)              (None, 1)                 51
=================================================================
_________________________________________________________________
```

```
"This is my final architecture to create my classification"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_1 (Embedding)      (None, 100, 128)          2560000
_________________________________________________________________
lstm_layer (LSTM)            (None, 100, 60)           45360
_________________________________________________________________
global_max_pooling1d_1 (Glob (None, 60)                0
_________________________________________________________________
dropout_1 (Dropout)          (None, 60)                0
_________________________________________________________________
dense_1 (Dense)              (None, 50)                3050
_________________________________________________________________
dropout_2 (Dropout)          (None, 50)                0
_________________________________________________________________
dense_2 (Dense)              (None, 1)                 51
=================================================================
Total params: 2,608,461.0
Trainable params: 2,608,461.0
Non-trainable params: 0.0
_________________________________________________________________
```

3.4.1 Improvement through refinement

I included the performance results Table 4 from primary, intermediate and final architecture of my LSTM solution where you can see small gradual improvements. For the corresponding architectures, you can refer to previous subsection.

Metrics	Value
Accuracy	0.923
Precision	0.513
Recall	0.121
F_1	0.196

(a) Primary LSTM-RNN

Metrics	Value
Accuracy	0.933
Precision	0.653
Recall	0.412
F_1	0.505

(b) Intermediate LSTM-RNN

Metrics	Value
Accuracy	0.948
Precision	0.741
Recall	0.708
F_1	0.725

(c) Final LSTM-RNN

Table 4: Metrics Evaluation over **LSTM-RNN** Refinement

4 Results

4.1 Model Evaluation and Validation

Our dataset is contaning of 561809 comments where they are devided into three classes: 80% Training section, 10% Validation set and 10% testing set. I used binary cross entropy as my loss function and keep those weights which shows improvements in validation loss. The final architecture with final quantities are presented in my final architecture session Architecture 3.3.2. I plot the learning curve Fig 12 and I find 2-3 epchos is a good one for my trianing session to avoid underfitting and overfitting. The metrics are evaluated based on the performance of the model on the testing set. I also check the validity of learning curve, as the system starts overfiting over large number of epochs. At the end, the ouput of our model is a probability of a comment to be toxic. I set a threshhold equal to 0.5, which is higher than 0.5 will be labeled as toxic and less than 0.5 is labeled as non-toxic.

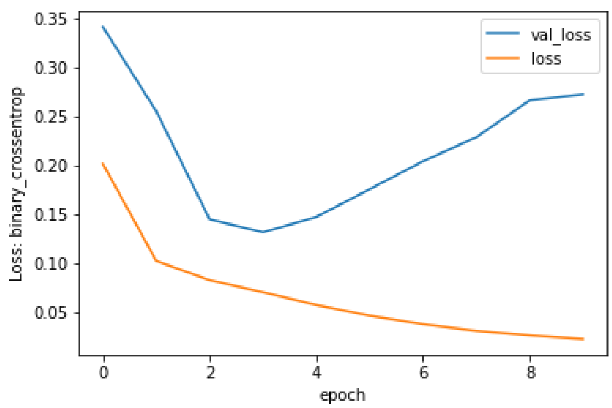


Figure 12: Learning curve

4.2 Justification

According to the metric performance Table 5, the final solution model is able to reach about 95% of accuracy with 71% of recall which is 20% higher than our benchmark performance Table 2 and also about 10% increase of F_1 score. In overal, our solution model identifies the toxic comments better than performance model. For instance, comparing the confusion matrix between the solution model and benchmark model (LSTM-RNN versus Multinomial NB) Fig 14 shows a significant progress of finding TP rightfully identification of toxic comments. Certainly there is a room for much better perfromance *precision*, *recall* and reaching higher *accuracy*. However through the current analysis $LSTM - RNN$ shows an acceptable performance.

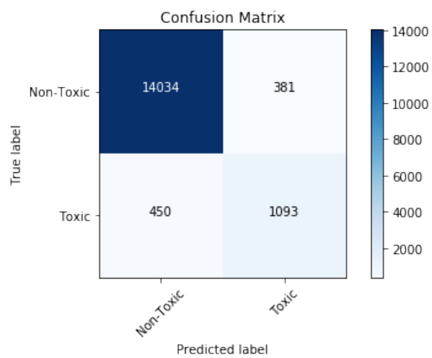
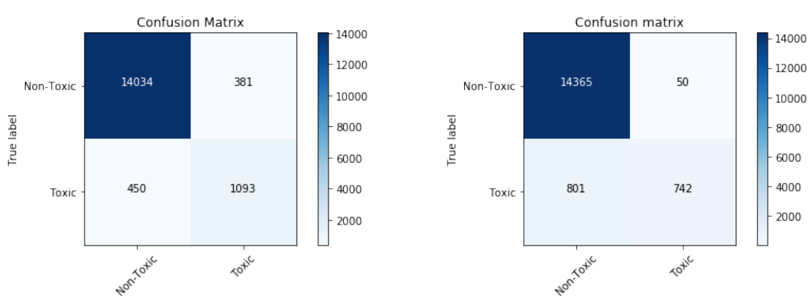


Figure 13: Confusion Matrix for LSTM-RNN

Metrics	Value
Accuracy	0.948
Precision	0.741
Recall	0.708
F_1	0.725

Table 5: Metrics Evaluation for **LSTM-RNN**



(a) LSTM-RNN **(b)** Tf-iDF+Multinomial NB

Figure 14: Performance Comparison between LSTM-RNN model versus Benchmark model

5 Conclusion

5.1 Free-Form Visualization

In the diagram below Fig 15, I briefly explain how the solution classifier as well as the benchmark model works. The description of how each step works is written in details in this article. The diagram briefly starts with a non-toxic comment and passes it through the classification pipline until it is labeled as non-toxic.

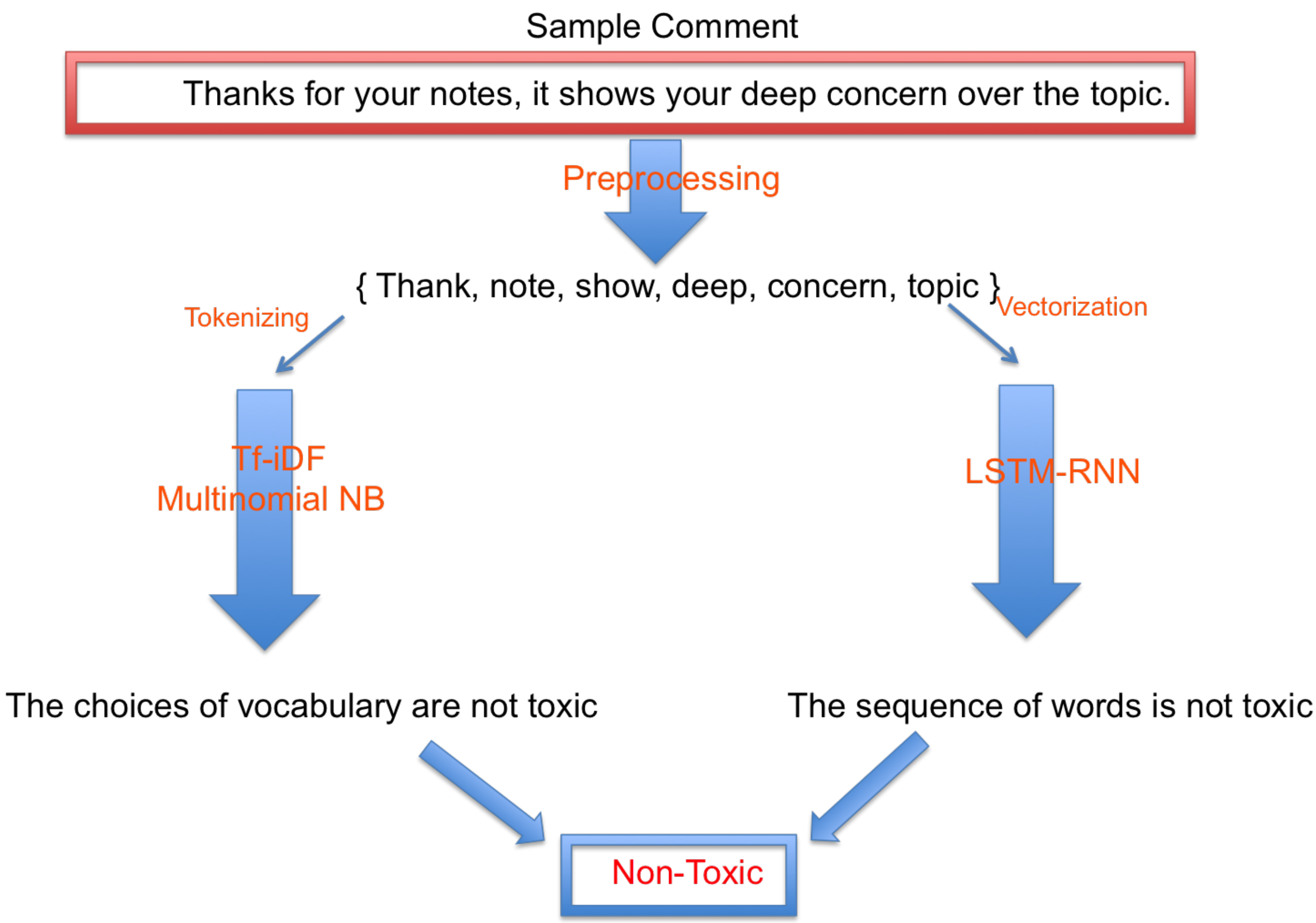


Figure 15: Simple and descriptive visualization of how the classifiers work

5.2 Reflection

Preprocessing and cleaning data is my very challenging task of this project. I think it is very vital to any NLP real world challenges. For example **"dont"**, **"do not"** and **"don't"** have all similar meaning but without preprcessing they are considered as different tokens. The design of **preprocessing** section is really task specific. In my project, I have the intuition that the tense of verbs or the aversary of the similar words are not vital for my label identification. That is why I used lemmatization to reduce my vocabulary to four forms of **noun, verb, adjective, adverb**.

Using the very simple intuitive methods or architectures are quite useful in NLP tasks. Regardless of final performance, they provide a very good intuition about the task. A simple **tf** analysis subsection 2.2 determines the choice of vocabulary is really different from toxic to non-toxic comments. This factor enough can be by itself a very good classifier. Such simple analysis suggests that a simple combination of **tf-idf+multinomial NB** Table 2 can provide a very good performance and some time beats fancy techniques.

The last not the least, NLP task can be a very challenging problem and requires a constant research in order to make a better performance. Word2vec techniques, Embedding are some of the most amazing parts of experiences during this project. The fact that you can define an algebra over the words by transforming them into the vectors and trains a semantic algebric understanding are quite exciting. Learning The powerful machinary which enables quantification of the most unquantifable areas is the most important lesson I learnt throught the entire course of machine learning.

6 Improvements and Additional Algorithms

After this project, I plan to improve my NLP classifiers: first by using other algorithms which are mentioned below (SVC and CNN), secondly, extend my classifier to the overal goal of kaggle competition which is multilabel classifier. I construct the multilabel classifer by combining two sub-classifiers. The first subclassifier is toxic-comment classifer (binary classifier), which labels the comments as toxic and non-toxic then those comments which are classified as toxic will be pipelined to second subclassifier to ientify the type of toxicity. This project is helping me to create the first subclassfier.

1. **Support Vector Classifier (a.k.a SVC)** Another recommended option is using SVM for text processing and text classification [24]. It requires a grid search for hyper paramter tunning to get the best results.
2. **Other DNN techniques (CNN)):** In a recent published paper [25], there is a comparative study of using different DNN platforms for NLP purposes. CNN also proves to have a very high performance for various NLP tasks.

[24] http://scikit-learn.org/stable/modules/svm.html#svm
[25] https://arxiv.org/abs/1702.01923