Machine Learning Nanodegree
Capstone Proposal

Mohammadmehdi Ezzatabadipour

February 18, 2018

1 Domain Background

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.

1.1 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.

2 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

[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

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.

3 Datasets and Inputs

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.

4 Solution Statement

I plan to use deep neural network (DNN) specially long short term memory Recurrent neural network (LSTM-RNN) in order to identify the toxic comments. One of the power of DNN, it is finding a group of words which are vital for a specific classification. In case of (LSTM-RNN), it is treating the comment as set the vectorized words like a time series and trying to learn how the words are aligned in a time series with respect to specific labels. 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.

5 Benchmark Model

I plan to make a pipeline of cleaning the data, tokenizing, removing frequent words, tf-idf (term frequency inverse document frequency) transformation and finally use multinomial naive bayes classifier for binary classfication. My choice of technique for benchmark is quite common used and proven to be working well for other projects such as spam email classifier. Certainly accuracy is not a good measure as most of the comments are not toxic, just classifying all comments as non-toxic gives us more than 90% accuracy. I also measure precision and recall and compare them to my solution output.

6 Evaluation Metrics

- **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.

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

- **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.
- **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.
- **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.

7 Project Design

7.1 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.

7.2 Preprocessing, Cleaning

In this section, I mainly use NLTK[9] (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.

```
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):
        return self

    def inverse_transform(self,X):
        pass

    def transform(self,X):
```

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

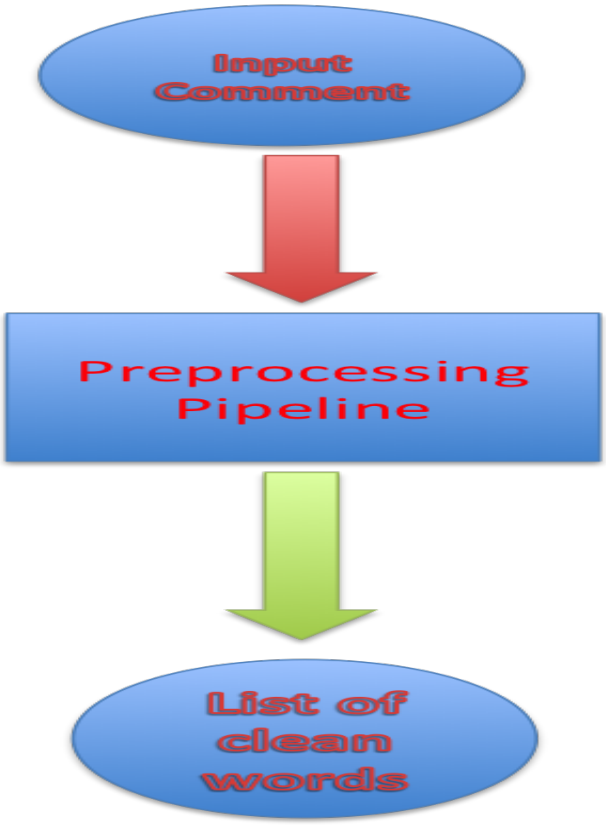

Figure 1: Preprocessing Workflow

```
        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)
```

7.3 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[10] for **word2vector**. 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. 2, we can get an idea, what it should be the fixed size of vector representation of each comment. 200 is very safe choice. However, it might make the entire algorithm quite time consuming.

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

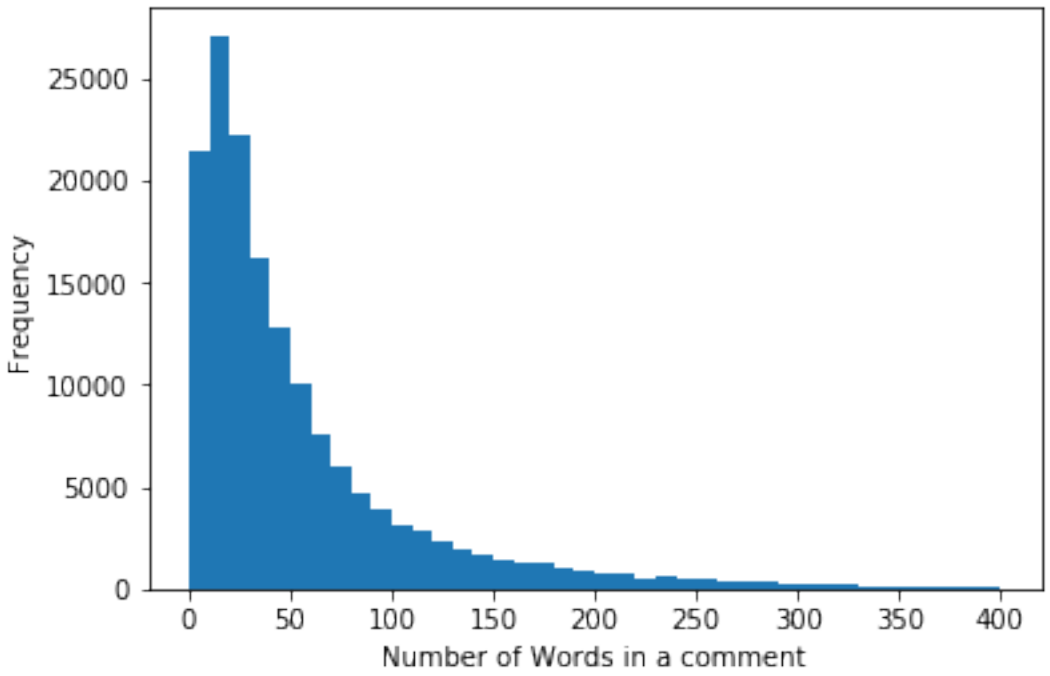

Figure 2: Histogram of Number of words per comment

7.4 Solution: 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 50, and in the embedding section, we decide that each word will be represented with vector size of 128.

```
"This is one of the toy architecture I used for my priliminary results"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_4 (Embedding)      (None, 50, 128)           2560000
_________________________________________________________________
lstm_layer (LSTM)            (None, 50, 60)            45360
_________________________________________________________________
global_max_pooling1d_4 (Glob (None, 60)                0
_________________________________________________________________
dropout_7 (Dropout)          (None, 60)                0
_________________________________________________________________
dense_7 (Dense)              (None, 50)                3050
_________________________________________________________________
dropout_8 (Dropout)          (None, 50)                0
_________________________________________________________________
dense_8 (Dense)              (None, 1)                 51
=================================================================
```

As you might see in the architecture above and also Fig. 3, the input of each LSTM layer is a matrix of size (50,128) which will be treated like a time-series of size (50) inside LSTM and the output of LSTM will be a vector of size (50,60). The way LSTM works, it recieves the vector of each time step (from 1 to 50) and tries to predicts the next coming input, then learns the inner memory between the each time step. The number of folds are 50 in our case.

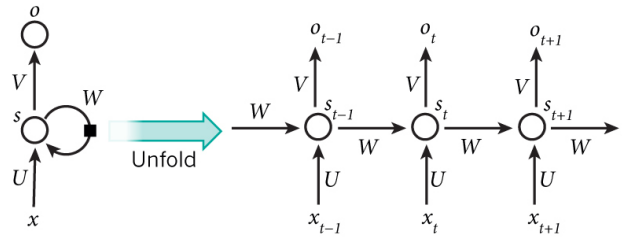

Figure 3: Simple Illustration of RNN Architecture [11]

We use max-pooling in order to reduce the dimention of the ouput of LSTM from (50,60) 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. I am still playing with paramters to get better performance.

7.5 Benchmark: Tf-idf + Multinomial Naive Bayes Classifier

After cleaning the data and tokenizing the words, the scikit-learn library of tf-idf is called to measure tf-idf of each token within the context of comment. That generates a numeric vector for each comment where the elements are tf-idf measurements of corresponding token (word). As a classifier, I choose Multnomial naive bayes (MultinomialNB) classifier from scikit-learn libraries.

8 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.

- **Support Vector Classifier (a.k.a SVC)** Another recommended option is using SVM for text processing and text classification [12]. It requires a grid search for hyper paramter tunning to get the best results.
- **Other DNN techniques (CNN)):** In a recent published paper [13], 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.

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