SENTIMENT CLASSFICATION ON PRODCUT REVIEWS

Subodh Sharma
Anand Rane
Kshitij Patil

MAY 10, 2020

This project aims to develop a gender classification system, when given twitter data for a user, it could tell the gender of the author. The levels of polarity are Male and Female. Finally, a fine-tuned model was used for predicting an appropriate label for a given twitter data.

Table of Contents

1. Introduction

This project aims to develop a gender classification system which on given a random twitter data can tell the gender of the author. The levels of polarity is Male and Female.

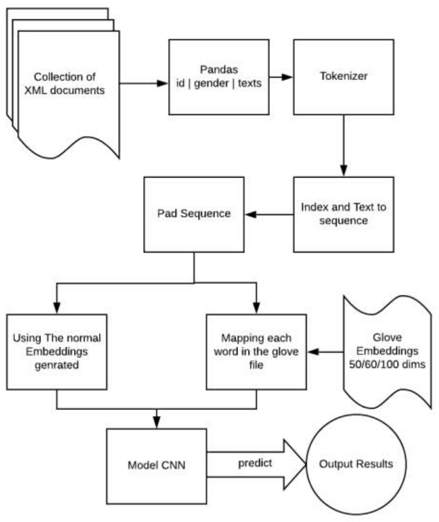
Our approach to predict the test dataset relies on various machine learning techniques. The flow of work is shown below:

2. Pre-processing and feature generation

2.1 Pre-processing

2.1.1 Pre-processing for Model 1 CNN

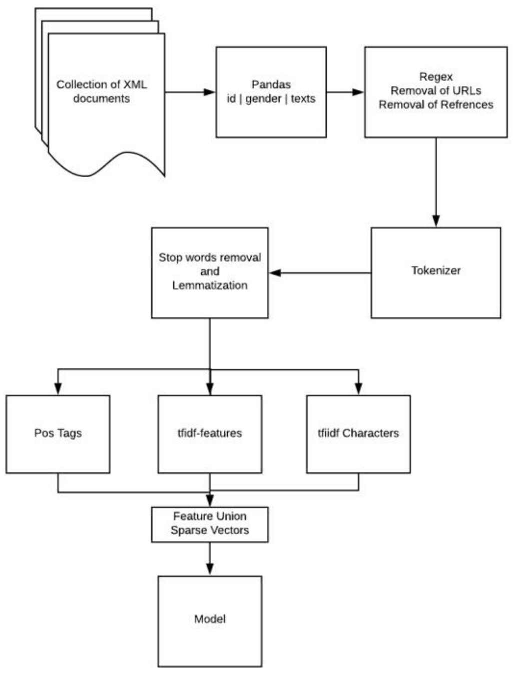
The model initially takes the texts in the embedding format by first tokenizing the each word and then running through a function called text_to_sequence () which these words and then finally adding the padding to words as for Deep learning models has to be of the same input size and the tweets are of variable sizes. The function truncates sentences which are bigger than the limit that we set and also adds padding to sentences smaller to that of the limit to generate the same input length. In our case the limit is set to 150 the same as that of the twitter's maximum limitation.


Collection of XML documents
Pandas id | gender | texts
Tokenizer
Index and Text to sequence
Pad Sequence
Using The normal Embeddings genrated
Mapping each word in the glove file
Glove Embeddings 50/60/100 dims
Model CNN
predict
Output Results

2.1.2 Pre-processing for Model 2 and 3

Processing Step	Use
Removing URLs	Removed all the URLS using the regular expression.
Removed user mentions	Removed user mentions from the tweet.
Removed hashtag	Splitting combined word into 2 if present in hashtag
Separating Connected Words	Splitting the connected words in hashtags
Concatenating Words	Concatenating words like aaaaahhhhh to ahh
Tokenizing	Tokenizing the tweet

Lowering	Decapitalizing the words
Removing Stop Words	Stop words are removed
Lemmatizing	Lemmatizing for getting adv, noun or verb
Unique Words	Taking unique number of tokens
1 char words	Words with only char are removed


Collection of XML documents
Pandas id | gender | texts
Regex Removal of URLs Removal of Refrences
Tokenizer
Stop words removal and Lemmatization
Pos Tags
tfidf-features
tfiidf Characters
Feature Union Sparse Vectors
Model

2.2 Feature generation

2.2.2 Feature Generation for Model 1

In the case of the model one there is limited explicit feature generation that has been done one of the key feature generation was that of mapping the words to the glove embeddings. Other on that N-gram is inherently given weightage depending on the size of the convolution kernel (N x Dims; where N is the size of kernel and Dims is the dimensions that are generated during the embedding phase for each word)

2.2.2 Feature Generation for Model 2 and Model 3

After Pre-processing the data, certain features from the data are extracted to build the model. Different models use different features for building the model. Supervised learning models SVC, SGD used features mentioned below:

We have extracted the following features from the tweet:

1. Unigrams, Bigrams, Trigrams, and Fourgrams from the words.
2. Bigrams, Trigrams, Fourgrams from the characters
3. Unigram, Bigram, Fourgram from the Parts of Speech Tags.
4. The words and chars which are occurring in all the documents are removed
5. Stop words from Words and Chars are removed
6. All the N grams are than converted to TF Vectors.
7. All the N grams except Parts of Speech are converted to IDF Vector

Benefit of using these features: The above features were helpful in creating a sparse vector of all the individual words in the matrix hence, the computation time and memory reduced significantly.

TFIDF provided probability of each word which was helpful in getting better accuracy of the model.

1. TF(w) = (Number of times term w appears in a document) / (Total number of terms in the document)
2. IDF(w) = log_e(Total number of documents / Number of documents with term w in it)

3. Models

3.1 Model 1 CNN

Using CNN for NLP based Gender Classification of twitter data. The main reasons for choosing such a methodology was to expand more than the bag-of-words based classification where the what an individual word (including Unigram, Bi-gram …. N-gram) words add value to a Document CNN also manages to maintain the usefulness of the order of the words i.e. the sematic importance of words in a sentence and therefore encoding more information a simple approach like the aforementioned bag of words.

The data was collated into one from where the id of an individual the gender and the tweet that they had posted was maintained. Particularly for the **CNN architecture** we did not *remove a Stop words, Special Words* etc. Rather we just used **Keras** inbuilt function of using word embeddings and also **Glove** word embedding to make words into features in **50,60- and 100-dimensions** vectors which map the vocabulary to vector spaces that represent them. In simpler terms what essentially happing semantically same words are mapped closer to each other like "*Queen*" and "*King*" or that the relationship of Female would be that of queen and similar relation of man would be that be of the king (strictly under rules of English Language).
Another reason for choosing is extracting the structure of the sentence an example of which can be the "cat is sitting" and the "dog is sitting" in both the cases we can generalize as an "animal is sitting" or even further generalizing it to "animal is action" this can be perfectly can be represented by word embeddings.

1.) *The simple word embedding by Keras*: Keras has its own library which can convert the words from the tweets into embeddings by using a unsupervised learning methodology. The major drawback of using such this library would be the embeddings would limit by the expanse of the twitter corporus that was provided to us. It would not be able to generalize to well due to this factor especially because the number of tweets where also ≈ 3000 tweets (with each tweet having an upper limit of 150 words).
2.) *Mapping to Glove word Embeddings:* Glove embeddings have been trained on a large English corporus by Stanford and has a proven record of better real-world word embeddings.

The above Diagram is a representation of how CNN can is used to get the Word Sequence from texts and generalizing the working of the Conv1d layer. Disclaimer: it is not the exact representation of the model we have used. The Model that we have proposed is given below.

The model is a simple build of 4 layers that is the Embedding layers that create the Word embeddings the Conv1d where the Word Sequences are removed then we flatten then input to put into the final sigmoid layer with 1 output (which will give the probability of the classification).

3.2 Model SVM

SVM is a supervised learning model, it draws a line in the hyperplane separating the two classes. Vectors are the coordinates in the hyperplane. The whole game is to identify the best hyperplane.

Using SVM we have achieved **84%** accuracy. Irrespective of the Hyper Parameter tuning and Adding more features the model accuracy is not improving.

We used TF-IDF of word n-grams, character n grams and parts of speech n grams, with C = 5.0, loss=l2.

C is the distance between the boundaries of this classes from the hyperplane.

Training: Model is trained on Sparse Vectors of TF-IDF from all the N grams except Parts of speech there we have used only TFs.

Hyper-parameter tuning: The model is hyper tuned with various values of C and loss functions. To the surprise there was not much improvement on accuracies.

Prediction: Used the trained classification model predict the 500 test labels and achieved 84% accuracy.

3.3 Model SGD

SGD is iterative algorithm we can use it in machine learning for optimizing different algorithms. The main reason to use this is because SVC is performing well but it is time consuming. We can use SGD with loss function 'hinge' which will behave like SVM and gives the benefits of optimization. We use hinge to to maximum margin classification which is similar to SVM but gives better result on the large dataset.

Using SGD, we have achieved 86% accuracy. The main problem with the SGD is that it the results are pretty inconsistent and there are little chances of re-producing the results, but the results remains somewhere around an average value.

We used TF-IDF of word n-grams, character n grams and parts of speech n grams, loss=l2.

Training: Model is trained on Sparse Vectors of TF-IDF from all the N grams except Parts of speech there we have used only TFs.

Hyper-parameter tuning: The model is hyper tuned with various values of C and loss functions. To the surprise there was not much improvement on accuracies.

Prediction: Used the trained classification model predict the 500 test labels and achieved 86% accuracy.

3.4 Discussion of model difference(s)

Features	CNN	SVC	SGD
Accuracy	51%	84%	86%

Hyper-parameter tuning	Learning Rate = 0.1
	Batch size = 32
	Filters (for the Conv1d layer) = 250
	Kernel Size = 4
	Parameter size = Default
	Number of Dimensions of word vectors = 50
		C = 5.0,
		Kernel = Linear
		Random state = 0
		Penalty = l2
		Dual = True
			Random state = 0
			Loss = hinge
			Penalty = l2

Data Used	Word Embeddings (Normal word embedding and glove)	TF-IDF for words, chars and POS	TF-IDF for words, chars and POS

4 Experiment setups

Model	Development Environment	Hyper Parameters/Parameters
CNN based classifier without Glove	Google Colab: Core: 2 RAM 12.72 GB (Default) DISK 107 GB (Default)	The Model Structure (Inclusion of more layers like Conv1d, Maxpooling etc.) Learning Rate Batch size Filters (for the Conv1d layer) Kernel Size Parameter size Number of Dimensions of word - - Vectors

SVC Linear Classifier	Personal Computer:
	Cores: i7- 6th generation
	RAM: 8 GB
		C = 5.0,
		Kernel = Linear
		Random state = 0
		Penalty = l2
		Dual = True

SGD Classifier	Amazon EC2 Instance
	C5x18 Large
	Cores: 72 (Logical)
	RAM: 144
		Random state = 0
		Loss = hinge
		Penalty = l2

5 Experimental results

5.1 Model 1

Almost all hyperparameter were giving the same results that is around the test average of 50.666 the final hyper parameters chosen are:

- Learning Rate = 0.1
- Batch size = 32
- Filters (for the Conv1d layer) = 250
- Kernel Size = 4
- Parameter size = Default
- Number of Dimensions of word vectors = 50

Validation Accuracy	Test Accuracy	Training Accuracy
65 %	51 %	80% (Overfitting)

Contrary to what we had initially thought; the word sequence models have little to no effect on the increase in the accuracy as we will see further the accuracy of models using bag-of-words have had better results.

5.2 Model 2 and 3 (SVC and SGD)

Initially the models were trained on the same dataset using n-grams for words and characters and POS tags.
We trained approximately 5 classifiers on the data and found out the best result from SVM and SGD.
We went on to improve these two classifiers but to the surprise by adding extra features like word count, emoticon counts, hashtag counts, Flesch Score, FRE score. Nothing improved the accuracy.

We tried using Grid Search on both the Classifiers but that also didn't help.
We have concluded the research on the dataset by keeping the same features from which we started initially.

We also learned that the best accuracy we have got from SGD has minimum chances of reproducing it. since it shuffles the dataset internally. But the main benefit is the accuracy remains in the close range of approximation.

6. Conclusion

In Conclusion, after testing out various features for the models that we had built we finalized on two feature sets that catered to the build of the Model 1 and Model 2/3.   Our initial idea of exploring deep learning models was to give weightage to word sequences than just using bag of words which lead by reading research paper for which we tested RNN, LSTMS and finally concluded with CNN to our surprise only achieving poor results. Which led us to use more traditional approaches like TF-IDF to text classification which was done by generating a sparse vector through a pipeline and was tested on a plethora models including Random Forest, Decision Trees, Naïve Bayes, SGD Classifier and SVC classifier finally finalizing on SGD Classifier which gave us results up to 86% of accuracy which holds true to Occam's Razor.

References

Kim, Y. (2014). Convolutional Neural Networks for Sentence Classification. Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP 2014), 1746–1751.
A Four Feature Types Approach for Detecting Bot and Gender of Twitter Users (Notebook for PAN at CLEF 2019 Johan Fernquist)
Twitter user profiling: Bot and Gender Identification (Notebook for PAN at CLEF 2019 Dijana Kosmajac and Vlado Keselj)
A guide to Text Classification(NLP) using SVM and Naive Bayes with Python(https://medium.com/@bedigunjit/simple-guide-to-text-classification-nlp-using-svm-and-naive-bayes-with-python-421db3a72d34)
SGD Sklearn (https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.SGDClassifier.html#sklearn.linear_model.SGDClassifier)
Understanding Convolutional Neural Networks for NLP (http://www.wildml.com/2015/11/understanding-convolutional-neural-networks-for-nlp/)
Support Vector Machines with Scikit-learn. (https://www.datacamp.com/community/tutorials/svm-classification-scikit-learn-python)
Text Classification Algorithms: A Survey (https://medium.com/text-classification-algorithms/text-classification-algorithms-a-survey-a215b7ab7e2d)
Hate Speech (https://github.com/t-davidson/hate-speech-and-offensive-language/tree/master/classifier)