**Applied Machine Learning and Life Cycle**

# Project Report

## Detecting Hate Speech in social media user-generated text using Natural Language Processing Techniques

Viraja Ketkar, Sumaira Afzal

School of Continuing Studies
York University

## 1. **Introduction**

With the increased use of social media platforms the problem of uncivilized speech has risen as well. A very common practice we see on social media platforms is a deliberate attack on person or group of people on the basis of attributes such as race, religion, ethnic origin, sexual orientation, disability, or gender. Due to uncivilized speech there is a lot of chaos and violence these days. Hate speech is an unfortunately common occurrence on the Internet. Often social media sites like Facebook and Twitter face the problem of identifying and censoring problematic posts while weighing the right to freedom of speech.

### Why this matters

The importance of detecting and moderating hate speech is evident from the strong connection between hate speech and actual hate crimes. Early identification of users promoting hate speech could enable outreach programs that attempt to prevent an escalation from speech to action.

### Problem Statement

Detect hate speech in tweets.For the sake of simplicity, we say a tweet contains hate speech if it has a racist or sexist sentiment associated with it. So, the task is to classify racist or sexist tweets from other tweets.

### Solution Statement

The proposed solution to this problem is to apply advanced machine learning algorithms such as ensemble methods, deep neural networks, and sequential algorithms.

### Goal

The goal of this project is to understand the components of a *Machine Learning Life Cycle* in order to apply them to practical business problem.

2. **Existing work**

Few existing applications in the industry:

**Workforce Analytics & Voice of Employee**

"crazy" good or bad?

If an employee calls their manager "crazy" in an anonymous survey, how do you know if they mean crazy good or crazy bad? And how long would it take you to read 10,000 surveys by hand? A Lexalytics text solution can show you in minutes how employees feel about their companies, managers, and work environments. Plus, we give you the context you need to understand exactly why they feel that way.

- Solution Profile: Lexalytics for People Analytics and Voice of Employee

**Voice of Customer (VoC)**

grow market share, build loyalty

Let your customers speak for themselves. Analyze thousands of social media comments, online reviews, and free-text survey responses in minutes. Hear what they're saying in their own words. Understand exactly how customers feel, and why they feel that way. Leverage that information to:

- Grow your market share and presence
- Build customer loyalty and improve retention
- Deliver innovative products in less time
- Perfect your go-to-market strategy

Solution Profile: Lexalytics for Voice of Customer

3.**Methodology**

### 3.1 Evaluation Metrics

We report the performance of each tweet analysis tool in terms of precision, recall, and f-measure for all classes.

### 3.2 Benchmark Model

For benchmark model we will use the algorithms outlined in this article and paper:

**1. A Novel, Gradient Boosting Framework for Sentiment Analysis in Languages where NLP Resources Are Not Plentiful: A Case Study for Modern Greek .**

| Algorithm | Accuracy | Precision (+) | Recall (+) | Precision (−) | Recall (−) |
|---|---|---|---|---|---|
| Decision Trees | 68.42% | 59.12% | 62.86% | 75.76% | 79.54% |
| SVM | 77.82% | 80.10% | 67.60% | 79.20% | 87.40% |
| Naive Bayes | 69.12% | 58.40% | 64.30% | 76.30% | 72.50% |
| Deep Learning | 72.00% | 67.50% | 74.80% | 82.40% | 78.00% |
| GBM | 84.20% | 82.40% | 77.90% | 86.30% | 88.30% |

**2. Baselines and Bigrams: Simple, Good Sentiment and Topic Classification**

## 3.3 Approach

Before diving into text classification lets see two categories of text classification:

**Topic Classification**

Topic classification is categorizing a text document into one of a predefined set of topics. In many topic classification problems, this categorization is based primarily on keywords in the text. SPAM filter, Inappropriate comment detection in discussion forums are two examples of topic classification.

**Sentiment Analysis**

Sentiment analysis (SA) has become a very active research area in natural language processing (NLP) and has attracted increasing interest in data mining, Web mining, and text mining.The goal is to identify the polarity of text content: the type of opinion it expresses. This can take the form of a binary like/dislike rating, or a more granular set of options, such as a star rating from 1 to 5. One example of sentiment analysis include analyzing Twitter posts to analyze hate speech/offensive language in tweets. Here is the workflow to solve machine learning problem

- Step 1: Gather Data
- Step 2: Explore Data
- Step 2.5: Choose a Model
- Step 3: Prepare Data
- Step 4: Build, Train and Evaluate Model
- Step 5: Tune Hyperparameter

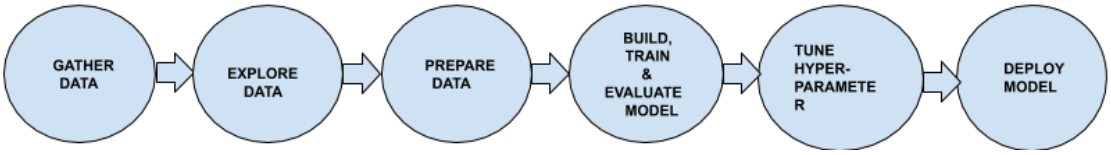



*Fig 1. Workflow to solve Machine Learning problem*

**Step 1: Gather Data**

Gathering data is the most important step in solving any supervised machine learning problem. Our text classifier can only be as good as the dataset it is built from.

We are tackling a specific problem, we used the positive/negative(Neutral/Hate or offensive language) polarity of the Twitter Tweets dataset. The dataset contains tweets posted by people on Twitter, it is labeled as **0 - Hate Speech, 1 - Offensive language and 2 - Neither.**

**Step 2: Explore Data**

Understanding the characteristics of our data beforehand enables us:

- to bring important aspects of the data into focus for further analysis.
- to build a better model
- to obtain a higher accuracy.
- to understand how much data we should use for training less data for training, or fewer computational resources.

The key to managing such an exploration is to be organized. Keeping records about the exploration, recording thoughts and ideas along the way, and organizing findings are all important. This is a complex undertaking, though possibly very rewarding.

**Load the Dataset**

First up, let's load the dataset into Python.

**Check the Dataset**

After loading the data, it's good practice to run some checks on it: so pick a few samples and manually check if they are consistent with our expectations e.g checking the missing values.

**Collect Key Metrics**

Once we have verified the data, we collected the following important metrics that can help characterize our text classification problem:

1. **Number of samples:** Total number of examples we have in the data.
2. **Number of classes:** Total number of topics or categories in the data.
3. **Number of samples per class:** Number of samples per class (topic/category). In a balanced dataset, all classes will have a similar number of samples; in an imbalanced dataset, the number of samples in each class will vary widely.
4. **Number of words:**Total number of words in dataset

5. Distribution of samples among classes:

| Metric Name | Metric Value |
|---|---|
| Number of samples | 23780 |
| Number of classes | 3 |
| Number of samples per class | 1 - 19190 \| 2 - 4163 \| 0 - 1430 |
| Number of words | 348721 |

*Table 1. Twitter dataset metrics summary*

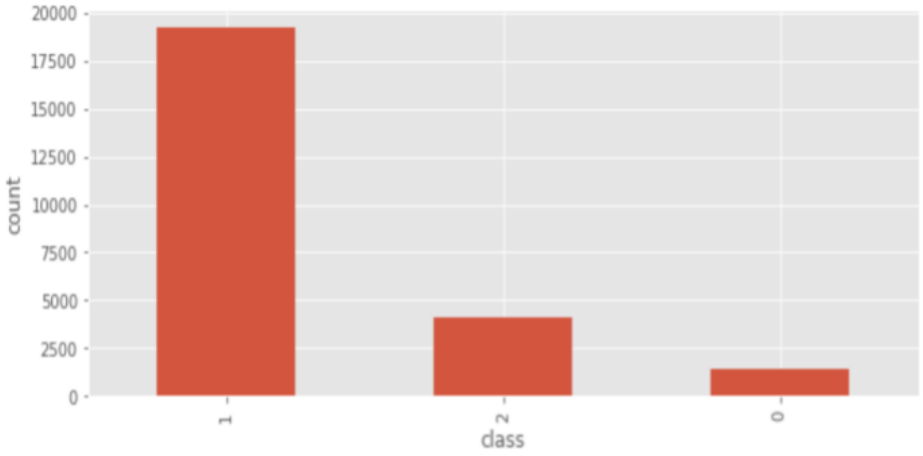



*Fig 2. Tweets representation per class*

**Step 3: Choose a Model**

At this point, we have assembled our dataset and gained insights into the key characteristics of our data. Next, based on the metrics we gathered in Step 2, we should think about which classification model we should use. This means/ asking questions such as, “How do we present the text data to an algorithm that expects numeric input?” (this is called data preprocessing and vectorization), “What type of model should we use?”, “What configuration parameters should we use for our model?”, etc.

Given that the best options might not be obvious, a naive solution would be to try every possible option exhaustively, pruning some choices through intuition. However, that would be tremendously expensive.

Our goal is to find the algorithm that **achieves close to maximum accuracy while minimizing computation time required for training.** We will try to implement different data processing techniques alternating and different model architectures. This will help us identify dataset parameters that influence optimal choices.

### Data preparation and Model Building Algorithm

```
1) Calculate the samples
2) Transform the classes into Binary classes
3) Clean the text
      a) Remove punctuation
      b) Convert words into lower case and split them
      c) Removing special characters
      d) Stemming
4) Split the dataset into training and testing dataset
5) Apply Feature Engineering techniques. We will apply two
   traditional models and one advance model for text feature
   extraction
      a) Bag of Words Model
      b) TF-IDF Model
      c) Word2Vec (future work for ensembles)
6) Score the importance of vectors (future work)
7) Build different models before and after balancing the data
      a) Single classifiers
      b) Deep learning and sequential models
      c) Ensemble models
8) Measure the Model performance with different hyper-parameters to
   find the best model configuration of the dataset
```

**Step 4: Prepare the Data**

**Text Preprocessing**

For this particular data set, our text cleaning step includes : HTML decoding, remove stop words, change text to lower case, remove punctuation, remove bad characters, and so on.

During the data exploration we found two major issues:

1. Multiple-Classes
2. Imbalanced dataset

To overcome multiple class issue we transformed the classes to Binary classes.Combining Hate Speech and Offensive language as 0 and Neither as 1 . Below table shows the multiple class data distribution.

| Class | Tweets |
|---|---|
| 0- Hate Speech | 1430 |
| 1- Offensive Language | 19190 |
| 2-Neither | 4163 |

*Table 2. Multiple class data distribution*

Records after transforming into Binary class:

| Class | Tweets |
|---|---|
| 0 - (Hate Speech + Offensive language) | 20620 |
| 1- Neither | 4163 |

*Table 3. Binary class data distribution*

**Split the dataset**

Before training, and even vectorizing, let's split our data into training and testing sets. It's important to do this before doing anything with the data so we have a fresh test set.

Before our data can be fed to a model, it needs to be transformed to a format the model can understand. First, the data samples that we have gathered may be in a specific order. We do not want any information associated with the ordering of samples to influence the relationship between texts

and labels. For example, if a dataset is sorted by class and is then split into training/validation sets, these sets will not be representative of the overall distribution of data. A simple best practice to ensure the model is not affected by data order is to always shuffle the data before doing anything else.

We have used scikit-learn test-train-split() and ratio is 80:20

**Feature Extraction**

Second, machine learning algorithms take numbers as inputs. This means that we will need to convert the texts into numerical vectors. There are three steps to this process:

**Bag of Words representation**

1. tokenizing strings and giving an integer id for each possible token, for instance by using white-spaces and punctuation as token separators.
2. counting the occurrences of tokens in each document.
3. normalizing and weighting with diminishing importance tokens that occur in the majority of samples / documents.

We call vectorization the general process of turning a collection of text documents into numerical feature vectors. This specific strategy (tokenization, counting and normalization) is called the Bag of Words or "Bag of n-grams" representation. Documents are described by word occurrences while completely ignoring the relative position information of the words in the document.

```
array([[0, 0, 0, ..., 0, 0, 0], [0, 0, 0, ..., 0, 0, 0], [0, 0, 0,
..., 0, 0, 0], ..., [0, 0, 0, ..., 0, 0, 0], [0, 0, 0, ..., 0, 0,
0], [0, 0, 0, ..., 0, 0, 0]],

dtype=int64) class scipy.sparse.csr.csr_matrix
```

This representation is used in conjunction with models that don't take ordering into account, such as logistic regression, multi-layer perceptrons, gradient boosting machines, support vector machines.

**Tf–idf term weighting**

Word counts are a good starting point, but are very basic. One issue with simple counts is that some words like "the" will appear many times and their large counts will not be very meaningful in the encoded vectors. An alternative is to calculate word frequencies, and by far the most popular method

is called TF-IDF. This is an acronym than stands for “Term Frequency – Inverse Document” Frequency which are the components of the resulting scores assigned to each word.

**Term Frequency:** This summarizes how often a given word appears within a document. Inverse Document Frequency: This downscales words that appear a lot across documents. Without going into the math, TF-IDF are word frequency scores that try to highlight words that are more interesting, e.g. frequent in a document but not across documents.

The TfidfVectorizer will tokenize documents, learn the vocabulary and inverse document frequency weightings, and allow us to encode new documents. Alternately, if you already have a learned CountVectorizer, you can use it with a TfidfTransformer to just calculate the inverse document frequencies and start encoding documents.The same create, fit, and transform process is used as with the CountVectorize.

```
(1, 26961)

 [[0. 0. 0. ... 0. 0. 0.]]
```

| Representation | Results |
|---|---|
| BOW | `array([[0, 0, 0, ..., 0, 0, 0], [0, 0, 0, ..., 0, 0, 0], [0, 0, 0, ..., 0, 0, 0], ..., [0, 0, 0, ..., 0, 0, 0], [0, 0, 0, ..., 0, 0, 0], [0, 0, 0, ..., 0, 0, 0]], dtype=int64)` `class scipy.sparse.csr.csr_matrix` |
| Tf–idf term weighting | `(1, 26961) [[0. 0. 0. ... 0. 0. 0.]]` |

*Table 4.Feature Extraction summary*

**Step 5: Build, Train, and Evaluate Model**

We tried different model building approaches:

1. First approach: Train different models(MultinomialNB,LSVC,LR) with BOW or tf-idf matrix on imbalanced dataset using Pipeline class in Scikit-Learn that behaves like a compound classifier.
2. Second approach: Train deep learning Models (LSTM,CNN,RNN) using word embeddings
3. Third approach: Train different ensemble models (NB, Knn) with BOW on balanced dataset using SMOTE()

1. **First approach: Model training with BOW and tf-idf matrix using Pipeline class in Scikit-Learn.Following figure explains this approach:**

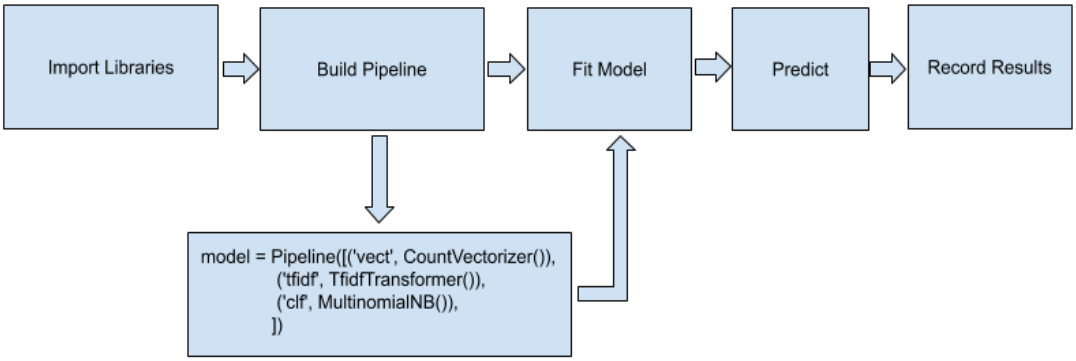



*Fig 3. Model training approach using Scikit-learn Pipeline class*

2. **Second approach: Train deep learning Models (LSTM,CNN,RNN) using word embeddings**

In deep learning text classification we followed this approach :

- Download data from balanced data
- Process the dataset
- Build neural network with LSTM Build neural network with LSTM and CNN Use pre-trained GloVe word embeddings Word Embeddings from Word2Vec.

3. **Third approach: Train different ensemble models with BOW on balanced dataset using SMOTE()**

**Ensemble-Learning**

The purpose of this part of the study is to compare the performance of different classifiers for polarity determination in highly imbalanced short text datasets using features learned by word embedding techniques. Several base classifiers and ensembles have been investigated with and without SMOTE (Synthetic Minority Over-sampling Technique). Using a dataset of tweets from twitter, the results show without SMOTE (Synthetic Minority Over-sampling Technique). The results show that applying Word2Vec with ensemble and SMOTE can achieve improvement on accuracy for the baseline for imbalanced datasets.For now we have implemented BOW to train ensemble models.

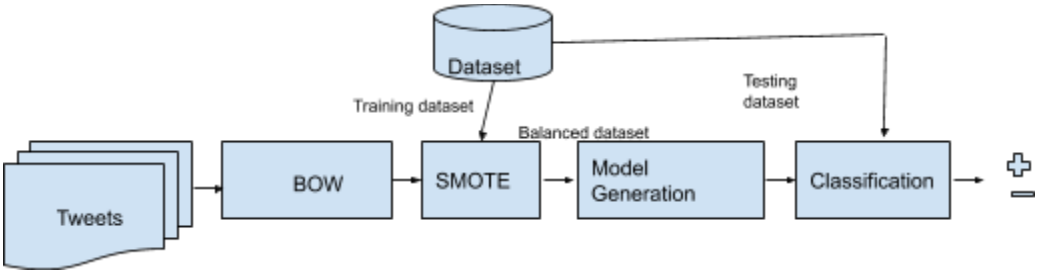



*Fig . Ensemble model train approach*

Several ensemble learning techniques are evaluated to generate computational models including: Bagging, Boosting, Voting and Random Forests. We followed these steps for training Ensemble models:

***Text Preprocessing***

Before feeding the data to ML models we cleaned the data

***Preparing for training***

*Tokenize the data*

We can not feed the string to ML models so After text cleaning and removing stop words, we have only over 2 million words to work with! We will convert our text documents to a matrix of token

counts using the CountVectorizer to tokenize (CountVectorizer). This is the approach which was used before word embedding. It used the concept of Bag of words where words are represented in the form of encoded vectors. It is a sparse vector representation where the dimension is equal to the size of vocabulary. If the word occurs in the dictionary, it is counted, else not. To overcome the shortcomings of BOW (It ignores order of the word, for example, this is bad = bad is this. It ignores the context of words) we will try Word2Vec to overcome this limitation.

***Balance the data***

We balanced the data by over-sampling using SMOTE. SMOTE was applied to the training set only for few reasons:

- **to ensure evaluating models using real test set not synthesized test set.**
- **if we apply SMOTE on whole dataset we will face the data leak problem.**
- **Additionally, evaluating computational models using synthesized samples results in high performers which is, in fact, incorrect.**

SMOTE was applied using imbalanced-learn toolbox.We got following results:

```
unique, counts = np.unique(y_train_res, return_counts=True)
print(list(zip(unique, counts)))
```

```
[(0,16532), (1,16532)]
```

We evaluated the classifiers as :

- Bagging Classifier using two different base-estimators: k-NN and Decision trees,
- AdaBoost Classifier using Decision tree as base estimator,
- For Voting Based Ensemble Classifier, we evaluated combinations of base classifiers. The combination is composed of four base classifiers, namely: LinearSVC, Logistic Regression, Random Forest Classifier and MultinomialNB. With voting-based ensemble classifier.

Table 7 summarizes the results of each classifier.We trained Various Classifiers with these parameters:

| | Name | Type | Parameters |
|---|---|---|---|
| clf1 | Gaussian NB (GNB) | Single classifier | |
| clf2 | k-NN | Single classifier | k=5 |
| clf3 | Decision Tree | Single classifier | Gini index, min_sample_split= 2 |
| eclf1 | RandomForest | Ensemble: Randomization | n_estimators = 20 |
| eclf2 | Bagging | Ensemble:Bagging | Estimator = Decision tree |
| eclf3 | Boosting | Ensemble:Boosting | Estimator = Decision tree |

| | | | |
|---|---|---|---|
| eclf4 | Voting | Ensemble:Voting | Estimators = LSVC, RF,LR,NB |

*Table 7. Summary of evaluated ensemble classifier*

**Step 6: Tune Hyperparameters**

We can tune hyperparameters using Grid Search CV for different algorithms

**Ensemble Models:**

- Random Forest: Tune the number of estimators
- K-nn: We should consider tuning the number of k
- Bagging classifiers: To achieve good results we should tune the base_estimators,
- Boosting classifiers: For optimal results we should tune n_estimators,

**Deep Learning Models**

- **Number of layers in the model**: We must be careful in choosing this value. **Too many layers** will allow the model to learn too much information about the training data, causing **overfitting**. **Too few layers** can limit the model's learning ability, causing **underfitting**. For text classification datasets, models with two layers performed well.
- **Number of units per layer**: The units in a layer must hold the information for the transformation that a layer performs. For the first layer, this is driven by the number of features. In subsequent layers, the number of units depends on the choice of expanding or contracting the representation from the previous layer. It is recommended **minimize the information loss between layers.**
- **Dropout rate**: Dropout layers are used in the model for **regularization**. They define the fraction of input to drop as a precaution for overfitting. Recommended range: 0.2–0.5.
- **Learning rate**: This is the rate at which the neural network weights change between iterations. A large learning rate may cause large swings in the weights, and we may never find their optimal values. A low learning rate is good, but the model will take more iterations to converge. It is a good idea to start low, say at 1e-4. If the training is very slow, increase this value. If your model is not learning, try decreasing learning rate.

There are couple of additional hyperparameters we tuned that are specific to our model:

1. **Kernel size**: The size of the convolution window. Recommended values: 3 or 5.
2. **Embedding dimensions**: The number of dimensions we want to use to represent word embeddings—i.e., the size of each word vector. Recommended values: 50–300. In our experiments, we used GloVe embeddings with 100 dimensions with a pre- trained embedding layer.

## 4. **Results**

Summary of classification report and accuracy with first approach on imbalanced dataset

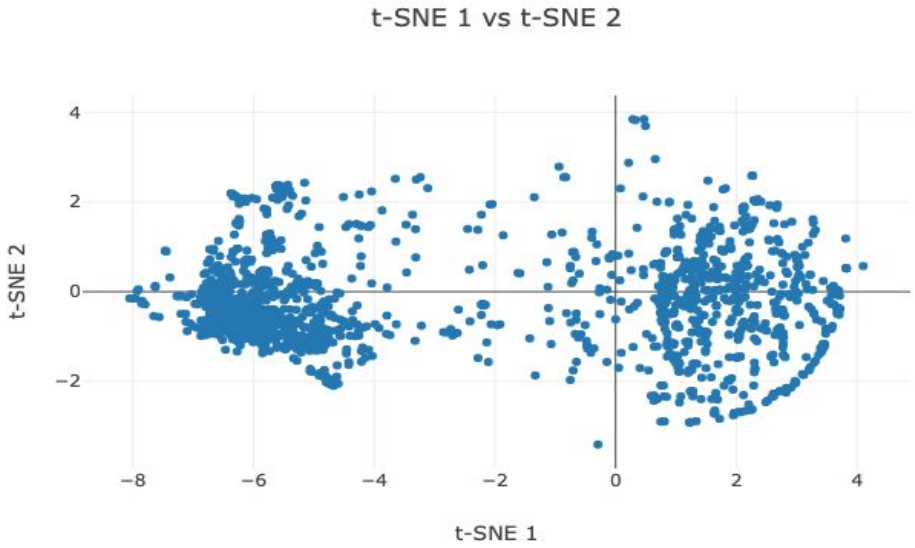
| Classifier | Without SMOTE | |
|---|---|---|
| | Classification report | Accuracy |
| NB | precision recall f1-score support<br><br>0 0.84 1.00 0.91 10301<br>1 0.98 0.07 0.13 2091<br><br>micro avg 0.84 0.84 0.84 12392<br>macro avg 0.91 0.54 0.52 12392<br>weighted avg 0.86 0.84 0.78 12392 | 84.32 |
| LSVC | precision recall f1-score support<br><br>0 0.85 1.00 0.92 10301<br>1 0.91 0.12 0.21 2091<br><br>micro avg 0.85 0.85 0.85 12392<br>macro avg 0.88 0.56 0.56 12392<br>weighted avg 0.86 0.85 0.80 12392 | 84.87 |
| LR | precision recall f1-score support<br><br>0 0.95 0.97 0.96 10301<br>1 0.83 0.75 0.79 2091<br><br>micro avg 0.93 0.93 0.93 12392<br>macro avg 0.89 0.86 0.87 12392<br>weighted avg 0.93 0.93 0.93 12392 | 93.10 |

*Table 5. Summary of results without SMOTE*

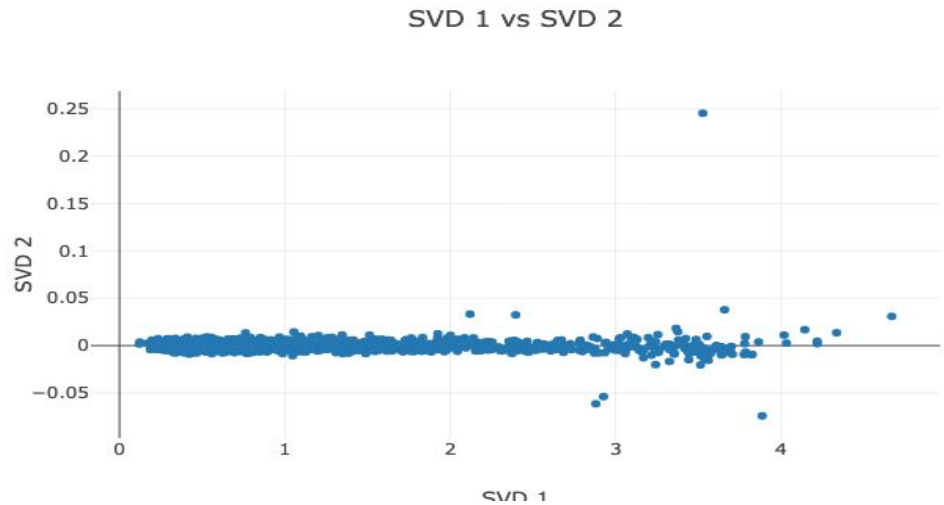
Summary of results using second approach on balanced dataset.

| Classifier | With SMOTE | | | |
|---|---|---|---|---|
| | 10 epoch | | 3 epoch | |
| | Time | Accuracy | Time | Accuracy |
| LSTM<br>vocab= 20000<br>memory units=100<br>Drop-out rate= 0.2 | 17 s | 0.99 | 17 s | 0.99 |
| CNN<br>pool-size=4 | 14 s | 0.99 | 10 s | 0.98 |
| Glove<br>Embedding vector size=100 | 6 s | 0.95 | 5 s | 0.85 |

*Table 6. Summary deep learning models results*




## Word Embeddings from Word2Vec



We recorded the Performance comparison of various base and ensemble classifiers with SMOTE on test dataset.

| Classifier | With SMOTE | | | | | |
|---|---|---|---|---|---|---|
| | | **precision** | **recall** | **f1-score** | **support** | ***Acc*** |
| clf1(NB) | 0 | 0.95 | 0.97 | 0.96 | 4105 | 92.97 |
| | 1 | 0.84 | 0.74 | 0.78 | 852 | |
| | micro avg | 0.93 | 0.93 | 0.93 | 4957 | |
| | macro avg | 0.89 | 0.85 | 0.87 | 4957 | |
| | weighted avg | 0.93 | 0.93 | 0.93 | 4957 | |
| clf2(k-nn) | 0 | 1.00 | 0.55 | 0.71 | 4105 | 62.73 |
| | 1 | 0.32 | 1.00 | 0.48 | 852 | |
| | micro avg | 0.63 | 0.63 | 0.63 | 4957 | |
| | macro avg | 0.66 | 0.77 | 0.59 | 4957 | |
| | weighted avg | 0.88 | 0.63 | 0.67 | 4957 | |
| clf3(DT) | 0 | 0.98 | 0.93 | 0.96 | 4104 | 92.79 |
| | 1 | 0.74 | 0.91 | 0.81 | 853 | |
| | micro avg | 0.93 | 0.93 | 0.93 | 4957 | |
| | macro avg | 0.86 | 0.92 | 0.88 | 4957 | |
| | weighted avg | 0.94 | 0.93 | 0.93 | 4957 | |
| eclf1 (RF) | 0 | 0.95 | 0.95 | 0.95 | 4104 | 91.16 |
| | 1 | 0.74 | 0.74 | 0.74 | 853 | |
| | micro avg | 0.91 | 0.91 | 0.91 | 4957 | |
| | macro avg | 0.85 | 0.84 | 0.84 | 4957 | |
| | weighted avg | 0.91 | 0.91 | 0.91 | 4957 | |
| eclf2 (Bagging) | 0 | 0.99 | 0.93 | 0.96 | 4104 | 93.04 |
| | 1 | 0.73 | 0.95 | 0.82 | 853 | |
| | micro avg | 0.93 | 0.93 | 0.93 | 4957 | |
| | macro avg | 0.86 | 0.94 | 0.89 | 4957 | |
| | weighted avg | 0.94 | 0.93 | 0.93 | 4957 | |
| eclf3 (Boosting) | 0 | 0.98 | 0.92 | 0.95 | 4104 | 92.25 |
| | 1 | 0.71 | 0.93 | 0.81 | 853 | |
| | micro avg | 0.92 | 0.92 | 0.92 | 4957 | |
| | macro avg | 0.85 | 0.93 | 0.88 | 4957 | |
| | weighted avg | 0.94 | 0.92 | 0.93 | 4957 | |
| eclf4 (Voting) | Avg 82.45 | | | | | Avg **94.38** |

Confusion Matrix for Ensemble models

| Classifier | Confusion Matrix |
|---|---|
| NB | [[4002.25  120.3 ]<br>[ 218.15  616.3 ]] |
| LR | [[3922.2   200.35]<br>[  79.15  755.3 ]] |
| LSVC | [[3949.4   173.15]<br>[ 102.5   731.95]] |
| k-nn | [[2261 1844]<br>[   3  849]] |
| Ensemble Classifier | |
| Decision Tree | [[3826  278]<br>[  79  774]] |
| RandomForest | [[3887  217]<br>[ 221  632]] |
| Bagging | [[3799  305]<br>[  40  813]] |
| Boosting | [[3780  324]<br>[  60  793]] |
| Voting Classifier | [[4024.2   97.6]<br>[ 180.8  654.4]] |

5. **Discussion**

Classification is primary form of text analysis and is widely used in variety of domains and applications. The applied text analysis has two most difficult parts:

1. **Curation and collection of domain specific corpus and build models on upon.**
2. **The second difficult part is composing an analytical solution for application-specific problem.**

It may not necessarily be immediately obvious how to compose application solutions , but it helps to understand the most language -aware data products are actually composed of multiple models and submodels.

Using a dataset of tweets, the results show that applying BOW with ensemble and SMOTE to train Ensemble Models can achieve more improvement on average in *F*1 score which is a weighted average of precision and recall and is considered a better performance measure than accuracy for the baseline. Accuracy is also improved in Ensemble models.

## 6. **Conclusion and recommendation**

As we can see the process of selecting an optimal model is complex,iterative and subsequently more intricate.When we see this in applied text analysis the search for the most optimal model follows a common workflow:create a corpus, select a vectorization technique, fit a model and evaluate using cross-validation.At application time it is recommended to select the model with best results based on cross-validation and use it to make predictions.

Importantly, classification offers metrics such as precision, recall, accuracy and f1 score that can be used to guide our selection of algorithms.However not all ,machine learning problems can be formulated as supervised learning problems.

Few metrics of evaluation in a classification and regression problems are recommended:

- Confusion matrix are generally used only with class output models.
- Gain and Lift chart are mainly concerned to check the rank ordering of the probabilities.
- Root Mean Squared Error (RMSE) is the most popular evaluation metric used in regression problems
- To check model is an overfit or not k-fold cross validation is widely used .If the performance metrics at each of the k times modelling are close to each other and the mean of metric is highest.K-Fold gives us a way to use every single datapoint which can reduce this selection bias to a good extent. Also, K-fold cross validation can be used with any modelling technique.

**Future Work:**

- We will apply word embeddings with ensembles and SMOTE to achieve more improvement.
- We will try different metrics of evaluation for model selection.

## 7. References/bibliography.


1. Tackling imbalance dataset in text classification https://towardsdatascience.com/yet-another-twitter-sentiment-analysis-part-1-tackling-class-imbalance-4d7a7f717d44
2. Addressing the problem of Unbalance Dataset in Sentiment Analysis

   https://www.scitepress.org/papers/2012/41426/41426.pdf

3. SMOTE https://arxiv.org/pdf/1106.1813.pdf
4. Data Imbalance Problem in Text Classification https://ieeexplore.ieee.org/document/5669059
5. Scikit-learn model evaluation https://scikit-learn.org/stable/modules/model_evaluation.html
6. A Novel, Gradient Boosting Framework for Sentiment Analysis in Languages where NLP Resources Are Not Plentiful: A Case Study for Modern Greek https://www.mdpi.com/1999-4893/10/1/34/pdf
7. Baselines and Bigrams: Simple, Good Sentiment and Topic Classification https://www.aclweb.org/anthology/P12-2018
8. Main approaches in NLP Coursera Course
9. How to handle Imbalanced Classification Problems in machine learning? Analytics Vidhya
10. Analytics Vidhya Data science tutorials
11. Datacamp Tutorial Hyperparameter Tuning
12. Google AI tutorials