# Blog Post Classification using Natural Language Processing


**Babandeep Singh**
UIN: 669265832
Email: bsingh8@uic.edu

**Rahul Gera**
UIN: 652701996
Email: rgera2@uic.edu

**Robin Beura**
UIN:658657743
Email: rbeura2@uic.edu



## Abstract

*Several supervised machine learning methods have been proposed in classifying numeric data into specified labels. However, classifying natural language, comparatively, is less explored as it possesses complicated challenges such as reading language at the machine level. This paper investigates the efficacy of using machine learning to categorize blogs. We design a text classification experiment to categorize blogs into three user groups: 'teenage', 'adults' and 'mature'. Our empirical findings indicate that leveraging machine learning models and the right feature vectorization can improve the content shown to users against randomly selected ones.*


## 1. Introduction

The number of blog posts is increasing exponentially. In 2004, Pew institute reported that 2-7% have written blog posts and about 11% read blogs who have access to the internet [1]. Technorati's web crawlers indicate that there are about 12,000 new blogs created each day; put another way, a new weblog is created every 7.4 seconds [2]. Given the popularity of blogs, it would be useful if we could devise a content classification system to automatically suggest the blogs to the right audience in terms of age group. However, it is difficult to group text into categories because of freestyle natural language. Bloggers at different levels write whatever is appealing to their mind, thus inventing sometimes new vocabulary and grammar. Nevertheless, some bloggers intentionally deviate from rules of language and decorum to create a spectacle for the sake of attracting a larger audience.

We used pure statistical learning methods to classify blogs conditioned on various feature vectorization techniques such as TF-IDF, count vectorizer and embeddings (Doc2vec and GloVe).

Section 2 discusses the Feature Engineering we used to curate data and make the machine readable. Section 3 discusses different statistical models used and their benefits. In section 4, we present empirical results and efficacy of different models. Section 5 concludes and finds and weighs any future implications.

## 2. Data Engineering

The dataset is acquired through Kaggle.com – Blog Posts Labeled with Age and Gender. Since the labeled dataset contains 3 discrete age groups (less than 17, 23 – 27, 33 and above), see Figure 1, this is a supervised multiclass classification problem.

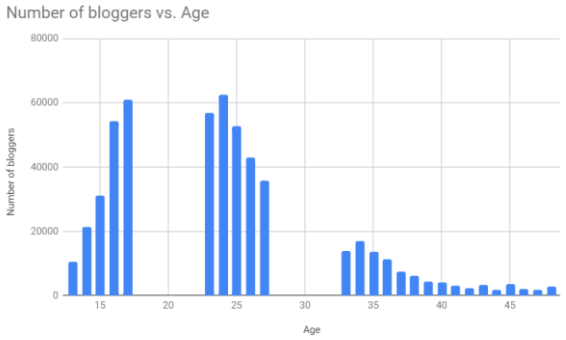


*Figure 1: Number of Bloggers and their age group*

The training and test dataset Contains 526,813 and 154,475 individual blog posts along with the age and gender of each blogger, respectively.

## 3. Data Preprocessing

First, we converted the numeric age into discrete categories (Mature, Minor, Adult) and manually harvested and classified 25000 blogs for each category using count vectorizer feature engineering technique with Naive Bayesian (Bernoulli) model to capture baseline accuracy of 55%). We used the Naïve Bayes classification algorithm because it was the fastest and most accurate classifier [3].

We further processed the text in a manner of removing punctuations such as "*,"* and "." and digits. We did not remove the stop words as we did not observe any difference and had an intuition that different authors might have different styles of writing, Furthermore, to make the text in same formatting, we transformed the text into lowercase text. We did not do lemmatization of verbs and nouns present as it was taking a lot of time for each combination and the improvement was not significant. As an additional feature, we removed words whose character length is less than 2 and greater than 16. For instance, test text *"I live in 49 S Maxwell Street, Maxwell's building*" transformed into *"live maxwell street maxwell building"*.

## 4. Feature Extraction

In this step, we transformed further into feature vectors using the existing dataset. We implemented the following different ideas to obtain relevant features from our dataset.

In our case, However, the raw data, a sequence of symbols cannot be fed directly to the algorithms themselves as most of them expect numerical feature vectors with a fixed size rather than the raw text documents with variable length.

To address this, we used scikit-learn library as it provides utilities for the most common ways to extract numerical features from text content, namely:

- **Tokenizing** strings and giving an integer id for each possible token, for instance by using white-spaces and punctuation as token separators.
- **Counting** the occurrences of tokens in each document.
- **Normalizing** and weighting with diminishing importance tokens that occur in most samples / documents.

In this scheme, features and samples are defined as follows:

Each individual token occurrence frequency (normalized or not) is treated as a feature. The vector of all the token frequencies for a given document is considered a multivariate sample. A corpus of documents can thus be represented by a matrix with one row per document and one column per token (e.g. word) occurring in the corpus.

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 [4].

### a. Countvectorizer

CountVectorizer implements both tokenization and occurrence counting in a single class. Initially it converts a collection of text documents to a matrix of token counts which produces a sparse representation of the counts.

### b. Tf–idf

Due to the presence of functional words or perhaps other common word tokens, the information carried by them becomes minimal in regard to actual information. Feeding in the direct count data directly to a classifier, frequent terms will overshadow the rarer yet focused terms.

The Term-Frequency Inverse Document frequency - which is intended to reflect how important a word is to a document in a collection or corpus - is computed as:

$$tfidf(t) = tf(t) \times (\log\left(\frac{1+n}{1+df(t)}\right) + 1)$$

Where *n* is the total number of documents in the document set, and *df(t)* is the number of documents in the document set that contain term *t*. Note, we also incorporated bigram and trigram tf-idf so that we capture syntactic dependencies too in our feature vector.

## c. Word Embeddings

We experimented with two types of word embeddings - Doc2Vec and GloVe.

**GloVe -** Word embedding algorithms like word2vec and GloVe are key to the state-of-the-art results achieved by neural network models on natural language processing problems like machine translation.

Word embeddings are an improvement over simpler bag-of-word model word encoding schemes like word counts and frequencies that result in large and sparse vectors (mostly 0 values) that describe documents but not the meaning of the words.

We downloaded pre-trained GloVe 100 and trained a Word2Vec model from scratch with gensim library. We got ourselves a dictionary mapping word -> 100-dimensional vector which we used to build features. The simplest way to do that was by averaging word vectors for all words in a text. We built a Sklearn-compatible transformer that was initialized with a word -> vector dictionary. We used a tf-idf weighting scheme in this case.

The parameters passed were:

size: (default 100) The number of dimensions of the embedding, e.g., the length of the dense vector to represent each token (word).

window: (default 5) The maximum distance between a target word and words around the target word.

min_count: (default 5) The minimum count of words to consider when training the model; words with an occurrence less than this count will be ignored.

workers: (default 3) The number of threads to use while training.

sg: (default 0 or CBOW) The training algorithm, either CBOW (0) or skip gram (1).

**Doc2Vec -** The bag-of-words (BOW) has many disadvantages. The word order is lost, and thus different sentences can have the same representation, if the same words are used. Even though bag-of-n-grams considers the word order in short context, it suffers from data sparsity and high dimensionality. That is why we implemented a paragraph vector which transforms the whole document into a vector taking into consideration its order and sense of semantics. The texts can be of variable-length, ranging from sentences to documents.

The concept that Mikilov and Le have used was simple, yet clever: they have used the word2vec model, and added another vector (Paragraph ID below), like this:

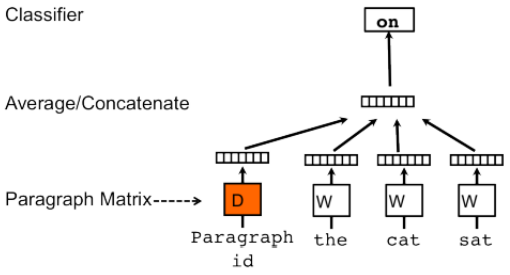


*Fig: PV-DM model*

The sketch above is a small extension to the CBOW model. But instead of using just words to predict the next word, another feature vector is added, which is document unique. So, when training the word vectors W, the

document vector D is trained as well, and at the end of training, it holds a numeric representation of the document.

The model above is called a Distributed Memory version of Paragraph Vector (PV-DM). It acts as a memory that remembers what is missing from the current context — or as the topic of the paragraph. While the word vectors represent the concept of a word, the document vector intends to represent the concept of a document.

We used the Doc2Vec model with gensim.

The parameters used were vector_size=64, window=2, min_count=1, workers=8, epochs = 20.

After some literature review and implementation, we found out that Doc2Vec usually outperforms simple averaging word2vec vectors.

## 5. Model

### a. Naive Bayes

After preprocessing and feature selection phases, the numbers of attributes will be significantly increased and are more precise for the use in building the classification model.

For the classification phase, Naïve Bayes is used as the classifier because of its simplicity and good performance in document and text classification [5].

Naïve Bayes classifier is the simplest instance of a probabilistic classifier. The output Pr(C | d) of a probabilistic classifier is the probability that a document *d* belongs to a class *C*. Each document contains terms which are given probabilities based on its number of occurrences within that document. With the supervised training, Naïve Bayes can learn the pattern of examining a set of test documents that have been well-categorized and hence comparing the contents in all categories by building a list of words as well as their occurrence.

$$y = argmax_y[P(y) * \prod_{i=1}^{n} P(x_i|y)]$$

Thus, such a list of word occurrences can be used to classify the new documents to their right categories, according to the highest posterior probability.

### b. Support Vector Machine

Support vector machines (or "SVMs") are another type of trainable classifier. SVMs are reportedly more accurate at classification than naive Bayesian networks in certain applications, Such as text classification for example [6]. They are also reportedly more accurate than neural networks in certain applications, such as reading handwritten characters for example [7]. Unfortunately, however, SVMs reportedly take longer to train than naive Bayesian classifiers. Thus, there is a need for methods and/or apparatus to build (or train) SVMs in an efficient manner.

The simplest form of an SVM defines a plane in the *n*-dimensional space (also referred to as a hyperplane) which separates feature vector points associated with objects "in a class' and feature vector points associated with objects "not in the class".

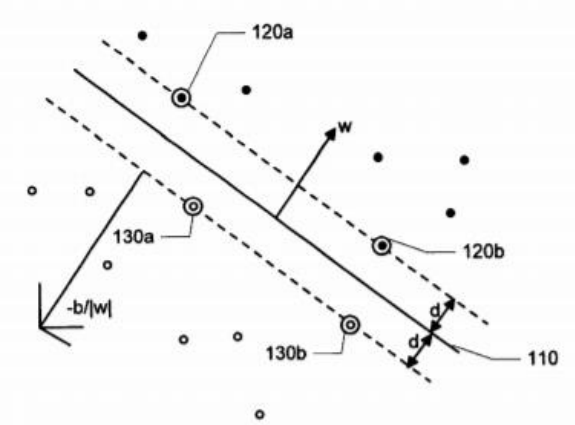


*Figure 2: SVM marginal separation*

In Figure 2, plane 110 separates feature vector points, denoted with blackened circles, associated with objects "in a class" from feature vector points, denoted with hollow circles, associated with objects "not in a class".

Several classes can be defined by defining several hyperplanes. The hyperplane defined by a trained SVM maximizes a distance (also referred to as a Euclidean distance) from it to the closest points (also referred to as "Support vectors') "in the class" and "not in the class". Referring again to Figure 2, support vectors 120, as well as support vectors 130, are located at a distance "*d*" from the plane 110. A hyperplane 110 is sought which maximizes these distances "*d*", so that the SVM defined by the hyperplane is robust to input noise.

A pair of adjustable parameters, **w** (a "weight vector") and **b** (a "threshold") can be defined such that all of the training data **X** (where **X** is a matrix composed of the features vectors **X**, of the training Set), having a known classification y, Satisfy the following constraints:

$$\vec{x}_i \cdot \vec{w} + b \geqq +1 \text{ for } y_i = +1$$

$$\vec{x}_i \cdot \vec{w} + b < -1 \text{ for } y_i = -1$$

where:
**w** is a weight vector parameter, **b** is a threshold parameter; and **y**, is a known classification associated with the $i^{th}$ example and is +1 if the example is "in the class" and -1 if the training example is "not in the class". Hence, the margin in the Sum of both inequality distances, namely $2/||w||$. By minimizing $||w||$, Subject to the constraints, the hyperplane providing the maximum margin can therefore be determined. [8]

## c. Logistic Regression

In a binary classification problem, where the response *y* falls into one of two categories, 0 or 1, logistic regression models the probability that *y* belongs to a particular category. The function that satisfies the property that a prediction is between 0 and 1 is the hypothesis function $0 \leq h_\theta(x) \leq 1$, defined as

$$h_\theta(x) = g(\theta^T x)$$

where the function *g* is the Softmax function

$$\text{softmax}(z) = \frac{e^z}{\sum_{i=1}^{k} e^{z_i}}$$

The output value of the hypothesis function is the estimated probability that the variable *y* is equal to 1 on a new input example x. More formally we can write this as

$$h_\theta(x) = P\ (y = 1 \mid x; \theta)$$

probability that y = 1, given feature x, parameterized by $\theta$.

The parameter vector θ is the vector of unknown linear regression coefficients of the *n*th order polynomial

$$\theta_o x_o + \theta_1 x_1 + \ldots + \theta_n x_n = \theta^T x.$$

High-dimensional vectors can be used in non-linear problems to get a more complex decision boundary, but the model will be more susceptible to overfitting, which means that it may fit the training set very well but fail to generalize to new examples.[9]

The training process of a classifier involves finding the best parameter θ vector for the logistic regression cost function $J(\theta)$ , given the dataset of *x* and *y* values. This optimization problem consists in minimizing the sum of the square difference between the output of the hypothesis $h_\theta(x)$ and class label *y*, which is finding parameters θ that minimize the function:

$$J(\theta) = \frac{1}{2m}\left[\sum_{i=1}^{m}(h_\theta(x^i) - y^i)^2 + \lambda \sum_{j=1}^{n} \theta_j^2\right]$$

where λ is the regularization parameter. This optimization problem can be solved with any standard numerical optimization algorithm, like the gradient descent or more advanced methods.

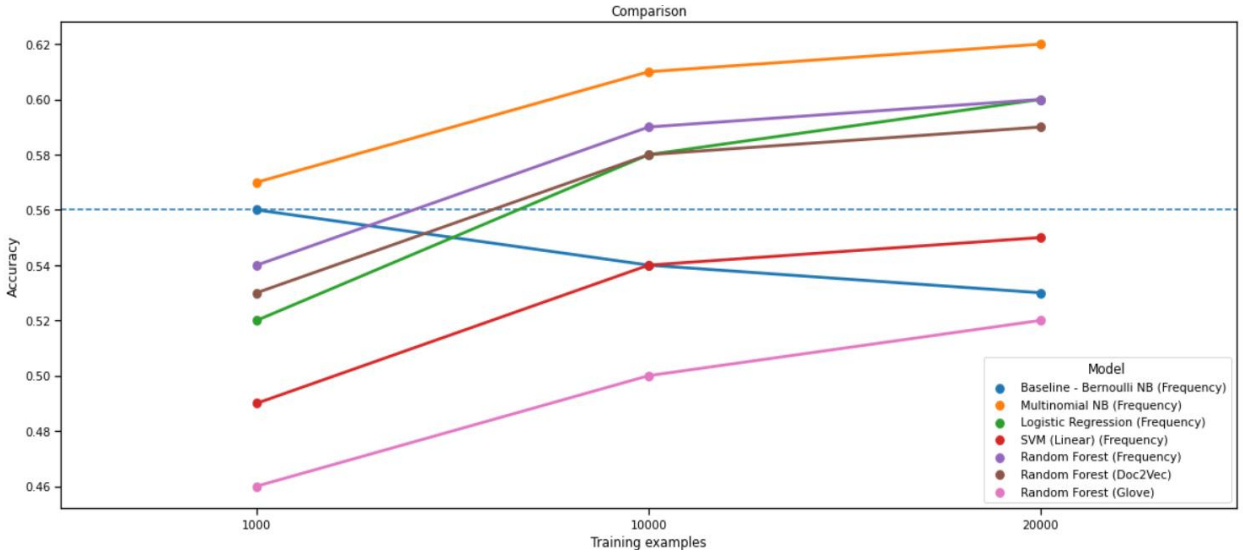


*Figure 3: Comparisons between Models*

### d. Random Forest

Random forests classifier (RFC) is one of the most successful ensemble learning techniques which have been proven to be very popular and powerful techniques in the pattern recognition and machine learning for high-dimensional classification and skewed problems [11]. A drawback associated with tree classifiers is their high variance. The reason for this lies in the hierarchical nature of the tree classifiers. An error that occurs in a node close to the root of the tree propagates all the way to the leaves. To make tree classification more stable, a decision forest methodology has been invented. A decision forest is an ensemble of decision trees. It can be seen as one classifier which contains several classification methods or one method but various parameters of work.

Consider a learning set made of $n$ vectors, $M \in X$ where X a set of numerical or symbolic observations and $N \in Y$ where Y is a set of class labels. For classification problems, a classifier is a mapping $X \rightarrow Y$. A new input vector is classified by each individual tree of the forest. Each tree yields a certain classification result. The principle of random forests is to build binary sub-trees using the training bootstrap samples coming from the learning sample L and selecting randomly at each node a subset of X. The decision forest chooses the classification which has the most votes overall the trees in the forest.

Random forest training is accomplished for each decision tree. In this method, each classifier's training set is generated by randomly drawing N examples, with replacement, with N the size of the original training set. The learning system generates a classifier from the sample and aggregates all the classifiers generated from the different trials to form the final classifier. To classify an instance, every classifier records a vote for the class to which it belongs, and the instance is labeled as a member of the class with the most votes. In case that more than one class jointly receives the maximum number of votes, then the winner is selected at random.

## 6. Results

The objective of this evaluation is twofold. First, it determines whether the preprocessing phase is useful to deduce better classification accuracy and performance when compared to the different statistical models. Second, it compares the classification accuracy and performance when different classifiers are applied on variable training examples.

A dataset with 5000 documents classified in three different categories is used for evaluation. The selected dataset contains three categories of document: Minor, Mature, Adult. All the three categories are easily differentiated.

Appendix 1 summarizes the results of using classifiers with different feature vectorizations and training examples to classify the posts. However, it surprisingly finds that the results of Multinomial Naive Bayes with count-vectorizer (57%) with 1000 training examples are closer to the baseline model. Although these results are better than a random guess, it is required to adjust the training examples in order to achieve better results.

Considering the increased training example, and same classifiers, we observed a significant increase in the accuracy in multinomial Naive Bayes with count vectorizer (62%). From Figure 3, we can observe that with increased training examples all the models are improving. We suspect that the poor performance of embedding is since bloggers tend to invent custom words and grammar which is not in line with the pretrained embedding models. However, the performance of these models also improved when we increased the training samples up to 20000.

## 7. Conclusion and Future Scope

In this study, we saw Naive Bayes is outperforming other statistical models such as Logistic Regression, Random Forest Classifier, Support Vector Machines. Through the implementation of different feature selection and training examples, it is demonstrated feature selection are two important steps to improve the mining quality. There are many words in the posts, therefore when we captured the terms from these documents, thousands of terms were found.

However, there are some terms that are useful and uninteresting to the results, it is then important to discover and interpret which features are useful and critical. Concerning numerous searching and selection techniques are available; it is encouraged to apply all these techniques and hence selects the best one for preprocessing of the data as well as to build the models. Furthermore, the performance of mining results is directly affected by the quality of data. So, the preprocessing and feature vectorizing phase is important to make the data being more precise (to achieve a better classification result) and even improve the time used to train and generalize the model.
Additionally, incorporating deep learning models such as neural networks would also improve the classification.

## 8. References


1. Amanda Lenhart, John Horrigan, and Deborah Fallow, "Content Creation Online", Pew Internet & American Life Project. http://www.pewinternet.org/pdfs/PIP_Content_Creation_Report.pdf
2. Sifry, David, "State of the Blogosphere, October 2004". http://www.sifry.com/alerts/archives/000387.html
3. McCallum, A. and Nigam K. "A Comparison of Event Models for Naive Bayes Text Classification". In AAAI/ICML-98 Workshop on Learning for Text Categorization, pp. 41-48. Technical Report WS-98-05. AAAI Press. 1998.
4. https://scikit-learn.org/stable/modules/feature_extraction.html#text-feature-extraction [accessed on 12/5/2020]
5. S. Chakrabarti, S. Roy, and M.V. Soundalgekar, "Fast and accurate text classification via multiple linear discriminant projection", The VLDB Journal The International Journal on Very Large Data Bases, 2003, pp. 170–185.
6. Joachims, Thorsten. "Text categorization with support vector machines: Learning with many relevant features." European conference on machine learning. Springer, Berlin, Heidelberg, 1998.
7. LeCun, Yann, et al. "Learning algorithms for classification: A comparison on handwritten digit recognition." Neural networks: the statistical mechanics perspective 261 (1995): 276.
8. Platt, John Carlton. "Methods and apparatus for building a support vector machine classifier." U.S. Patent No. 6,327,581. 4 Dec. 2001.
9. Allegorico, Carmine, and Valerio Mantini. "A data-driven approach for on-line gas turbine combustion monitoring using classification models." Second European Conference of the Prognostics and Health Management Society. 2014.
10. Bouaziz, Ameni, et al. "Short text classification using semantic random forest." International Conference on Data Warehousing and Knowledge Discovery. Springer, Cham, 2014.
11. Ho, Tin Kam. "Random decision forests." Proceedings of 3rd international conference on document analysis and recognition. Vol. 1. IEEE, 1995.
12. Amit, Yali, and Donald Geman. "Shape quantization and recognition with randomized trees." Neural computation 9.7 (1997): 1545-1588.
13. Breiman, Leo. "Bagging predictors (Technical Report 421)." University of California, Berkeley (1994).
14. Distribution Representations of Sentences and Documents by Le and Mikolov. https://cs.stanford.edu/~quocle/paragraph_vector.pdf

## Appendix 1:

| Number of Training Samples | Model | Feature Engineering | Accuracy | F1 score |
|---|---|---|---|---|
| 1000 | Baseline - Bernoulli NB | Frequency | 0.56 | 0.61 |
| | Baseline - Bernoulli NB | TF-IDF | 0.49 | 0.52 |
| | Baseline - Bernoulli NB | N-gram | 0.45 | 0.48 |
| | Multinomial NB | Frequency | 0.57 | 0.61 |
| | Multinomial NB | TF-IDF | 0.51 | 0.63 |
| | Multinomial NB | N-gram | 0.49 | 0.61 |
| | Gaussian NB | Doc2Vec | 0.4 | 0.45 |
| | Gaussian NB | Glove | 0.19 | 0.32 |
| | Logistic Regression | Frequency | 0.52 | 0.53 |
| | Logistic Regression | TF-IDF | 0.56 | 0.63 |
| | Logistic Regression | N-gram | 0.5 | 0.59 |
| | Logistic Regression | Doc2Vec | 0.52 | 0.58 |
| | Logistic Regression | Glove | 0.42 | 0.44 |
| | SVM (Linear) | Frequency | 0.49 | 0.49 |
| | SVM (Linear) | TF-IDF | 0.54 | 0.56 |
| | SVM (Linear) | N-gram | 0.48 | 0.51 |
| | SVM (Linear) | Doc2Vec | 0.52 | 0.57 |
| | SVM (Linear) | Glove | 0.42 | 0.44 |
| | Random Forest | Frequency | 0.54 | 0.6 |
| | Random Forest | TF-IDF | 0.54 | 0.62 |
| | Random Forest | N-gram | 0.49 | 0.56 |
| | Random Forest | Doc2Vec | 0.53 | 0.59 |
| | Random Forest | Glove | 0.46 | 0.57 |
| 10000 | Baseline - Bernoulli NB | Frequency | 0.54 | 0.57 |
| | Baseline - Bernoulli NB | TF-IDF | 0.48 | 0.5 |
| | Baseline - Bernoulli NB | N-gram | 0.46 | 0.48 |
| | Multinomial NB | Frequency | 0.61 | 0.63 |
| | Multinomial NB | TF-IDF | 0.6 | 0.65 |
| | Multinomial NB | N-gram | 0.54 | 0.61 |
| | Gaussian NB | Doc2Vec | 0.5 | 0.51 |
| | Gaussian NB | Glove | 0.38 | 0.41 |
| | Logistic Regression | Frequency | 0.58 | 0.59 |
| | Logistic Regression | TF-IDF | 0.6 | 0.63 |
| | Logistic Regression | N-gram | 0.52 | 0.57 |
| | Logistic Regression | Doc2Vec | 0.54 | 0.61 |
| | Logistic Regression | Glove | 0.44 | 0.49 |
| | SVM (Linear) | Frequency | 0.54 | 0.55 |
| | SVM (Linear) | TF-IDF | 0.57 | 0.58 |
| | SVM (Linear) | N-gram | 0.49 | 0.5 |
| | SVM (Linear) | Doc2Vec | 0.54 | 0.6 |
| | SVM (Linear) | Glove | 0.44 | 0.5 |
| | Random Forest | Frequency | 0.59 | 0.66 |
| | Random Forest | TF-IDF | 0.59 | 0.65 |
| | Random Forest | N-gram | 0.52 | 0.58 |
| | Random Forest | Doc2Vec | 0.58 | 0.64 |
| | Random Forest | Glove | 0.5 | 0.6 |
| 20000 | Baseline - Bernoulli NB | Frequency | 0.53 | 0.55 |
| | Baseline - Bernoulli NB | TF-IDF | 0.48 | 0.5 |
| | Baseline - Bernoulli NB | N-gram | 0.46 | 0.48 |
| | Multinomial NB | Frequency | 0.62 | 0.64 |
| | Multinomial NB | TF-IDF | 0.61 | 0.65 |
| | Multinomial NB | N-gram | 0.58 | 0.61 |
| | Gaussian NB | Doc2Vec | 0.56 | 0.57 |
| | Gaussian NB | Glove | 0.45 | 0.47 |
| | Logistic Regression | Frequency | 0.6 | 0.61 |
| | Logistic Regression | TF-IDF | 0.61 | 0.64 |
| | Logistic Regression | N-gram | 0.53 | 0.58 |
| | Logistic Regression | Doc2Vec | 0.59 | 0.63 |
| | Logistic Regression | Glove | 0.45 | 0.52 |
| | SVM (Linear) | Frequency | 0.55 | 0.55 |
| | SVM (Linear) | TF-IDF | 0.58 | 0.6 |
| | SVM (Linear) | N-gram | 0.48 | 0.5 |
| | SVM (Linear) | Doc2Vec | 0.59 | 0.66 |
| | SVM (Linear) | Glove | 0.47 | 0.57 |
| | Random Forest | Frequency | 0.6 | 0.66 |
| | Random Forest | TF-IDF | 0.59 | 0.66 |
| | Random Forest | N-gram | 0.53 | 0.59 |
| | Random Forest | Doc2Vec | 0.59 | 0.69 |
| | Random Forest | Glove | 0.52 | 0.61 |