Introduction

This study is attempted to analyze different Machine Learning models for document classification. The models chosen for this study are Support Vector Machines, K-Nearest Neighbors and Naive Bayes. These particular models are chosen because of their ease of implementation, popularity for different ML tasks, and most importantly for understanding their mathematical implementations, hyperparameters and using some optimizations and modifications at different stages of model building, finding out the model which performs the best for given performance metrics. This study aims to propose the best possible workflow for document classification problems.

Dataset

The dataset is the collection of newsgroups based on 10 different subjects. The groups include business, entertainment, food, graphics, historical, medical, politics, space, sports, technology. Each group includes 100 different text files containing articles related to the particular subject. Each text file contains somewhere between 1200 to 2500 words. In total, there were about 23,24,000 words after removing the stop words and after preprocessing the data, which became the vocabulary of the dataset, used for training the models.
The dataset is publicly available on kaggle:
https://www.kaggle.com/datasets/jensenbaxter/10dataset-text-document-classification

Preprocessing

After getting the dataset, the next important step is to preprocess and clean it. The text of the articles contains all sorts of characters which are not useful for model training. Only english words excluding stop words are to be kept in the dataset. The following flow was followed for preprocessing the text data of the articles. Firstly, the entire text was converted into lowercase, so that the model is case insensitive. Then all the tags, and special characters were removed. Then the data was tokenized or the sentences were broken down into individual words or tokens. Then the non-alpha tokens or numerical characters were removed, along with the punctuation symbols. Then the stop words were removed. NLTK provides all the stop words in the English language which were used to filter out the text without the stop words. The last step in preprocessing is to get the base words so that the tense of the words do not make a completely different word, for example sleep, slept, and sleeping must have the same meaning and base word. There are two different ways of getting the base word or normalizing the text: Stemming and Lemmatization.
1)Stemming: This involves trimming the last few characters from a word, for example trimming the ‘s’ from a plural word, or trimming ‘ing’ from a continuous form of word, to return a root word. The words after Stemming may not have any meaning in the English language, but all the different tenses of that word are converted into the same base word, which the model considers as the same word. NLTK provides the PorterStemmer technique which reduces the words in the text to their root form.

2)Lemmatization: Lemmatization is also the process of getting the root word from a given word, but this is different from Stemming, as Lemmatization involves getting the actual root word from the English language, which involves the parts of speech. WordNet Lemmatizer uses WordNet database to return the lemmas for the given word. Lemmatizer goes through the entire text document to understand the context in which the word is used and then returns the meaningful base word. This process is slower than Stemming, and as it turns out is not useful for the purpose of document classification. Lemmatization is more useful in building chatbots or where answering like humans is required but for the purpose of document classification, Stemming is better as no matter the context it returns the same base word, which is required if a document is to be classified based on the text.

Data Embedding

The next step involved creating vector representations of the textual data. The vectors were essentially the features of the text, which were used for model training. Data Embedding is used for extracting the most important features from the textual data which would enhance the model accuracy. There are several ways for vectorizing the data:

1) Bag of Words: This is the simplest technique for vectorization. It generates the number of distinct words and the count of each word that appeared in all the documents. The distinct words form the vocabulary of the model. It is called the Bag of Words approach because it creates the frequency distribution of the words without accounting for the order of their occurrence. The textual data is converted into a matrix form of occurrence of all the words in the document, with the number of features being the number of distinct words. It forms a binary matrix represented as 1 for the words appearing and 0 for not appearing. Bag of Words can be performed using different metrics like, simple 0 or 1 for appearing and not appearing word in the text, or using CountVectorizer, or using TF-IDF Vectorizer (Term Frequency- Inverse Document Frequency). Term Frequency (TF) indicates the number of occurrences of the word in the document, which indicates the importance of that word in that particular document. Document Frequency (DF) is the number of documents that contain that particular word. DF indicates how common the word is. Based on the TF-IDF scores for every word, they are assigned appropriate weights. IDF is defined as the logarithm of the total number of documents divided by the DF score for that word.

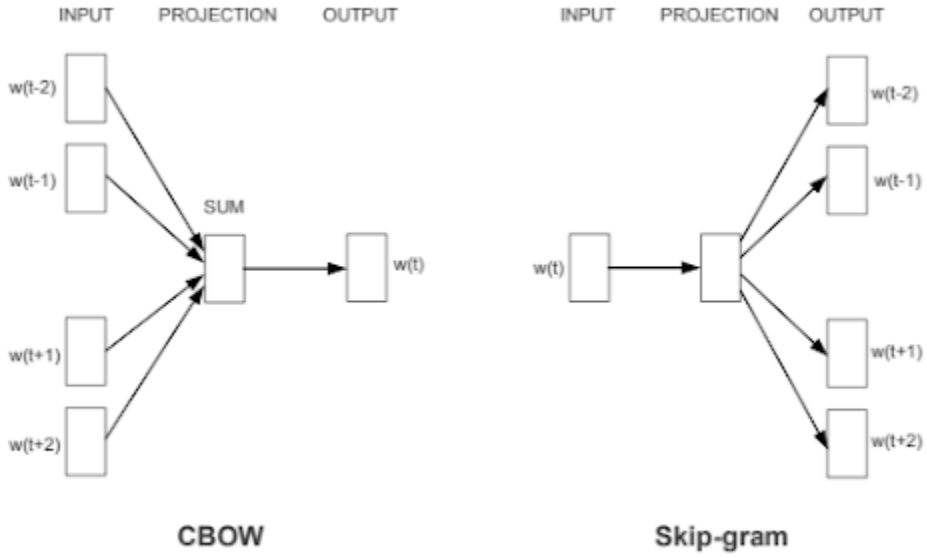
$$idf_i = \log\left(\frac{n}{df_i}\right)$$

The weights (TF-IDF score) is the multiplication of tf and idf score for the particular word ‘i’ in the particular text document ‘j’.

$$w_{i,j} = tf_{i,j} \times idf_i$$

The problem with this method is that the matrix would be a sparse matrix as most of the entries in the vector would be zero. Also, the dimensionality of the vector would be so high, which makes it very difficult for the model to be reduced and also computationally expensive.

2) Data Embedding: The Embeddings are generated to translate the large sparse vectors into lower dimensional space that preserves the semantic relationship of the text. The embeddings group words in such a way that the similar words would be closer to each other. The Embedding overcomes the problem of Bag of Words that it considers all words separately, without considering the semantically similar words as grouped together. The words with similar meaning are kept equidistant to each other in n-dimensional vector space. There are several ways to perform data embedding like using Word2Vec, GloVe, BERT, GPT by OpenAI and several other pretrained models.
    a) **Word2Vec**: Word2Vec is a three layered Neural Network- consisting of input, projection and output layers based model- which tries to learn the patterns and relationships between the words occurring near to each other in the text file. This model does not consider individual words but it trains the words against other words appearing close to the word from the vocabulary, which captures the meaning based on the context. There are two types of Word2Vec models:



        i) CBOW: This method is the Continuous Bag of Words, which uses the words that appear before and after the word within a certain window of observations to predict the current word. It uses the context and semantic understanding of the adjacent words to group words that appear to be

similar. The similar words are located in close vicinity of each other in the vector space.

ii) Skip Gram: This method is exactly the reverse of CBOW, as it predicts the previous and next few words for a certain window, given the current word.

It was found by Mikolov, et al that Skip-Gram works well for small-medium sized datasets. But CBOW is faster and more efficient to train. It also found that Skip-Gram predicts better for words which are less frequent, while CBOW is better for words which are more frequent.

The Word2Vec implementation provides several parameters like size, window, minimum count and sg. If sg=0, then the algorithm would perform CBOW otherwise it would be Skip Gram.

b) **PreTrained (Transfer Learning)**: There are many datasets which are trained on huge amounts of data, which have a large amount of corpus which can enhance the accuracy of our model. We have used the model trained on Google News dataset which has more than 100 billion words. The corpus of the training dataset massively increases. The first step in using the pretrained model is to leave out the words in our dataset which are not present in the vocabulary of the google dataset. Then the model generates 300 dimensional vector embeddings for the text document based on the context and similarity grouping. The embeddings are assigned to a text file such that the window defined for the prediction word matches the context in the google news dataset. Based on this grouping, similarity values are assigned to the individual words of the text file, and then the mean of all the embedding values is taken across the columns, which result in a 300 dimensional vector embedding for all the text files in our local dataset. The embeddings identify the most similar words, and rank them based on the amount of similarity with the given word. For example, the word “space” has following similar words as identified by the embeddings:

```
[('spaces', 0.6570690870285034),
 ('music_concept_ShockHound', 0.5850345492362976),
 ('Shuttle_docks', 0.5566749572753906),
 ('Space', 0.5478203296661377),
 ('Soviet_Union_Yuri_Gagarin', 0.5417766571044922),
 ('Shuttle_Discovery_blasts', 0.5352603197097778),
 ('Shuttle_Discovery_docks', 0.534925103187561),
 ('Shuttle_Endeavour_undocks', 0.532420814037323),
 ('Shuttle_Discovery_arrives', 0.5323426723480225),
 ('Shuttle_undocks', 0.523307740688324)]
```

Note that even if the word space is not used in any of the above phrases, it still identifies that the phrase “Soviet Union Yuri Gagarin” or “Shuttle Docks” is extremely closely related to the word “space”, based on the context and appearance of these words in the vicinity of the word “space” not only in our dataset but found out from the huge amounts of Google News Dataset. The generated vectors can be used directly in a Machine Learning model, or can be processed for dimensionality reduction to reduce the complexity and then passed on to the ML model.

Dimensionality Reduction

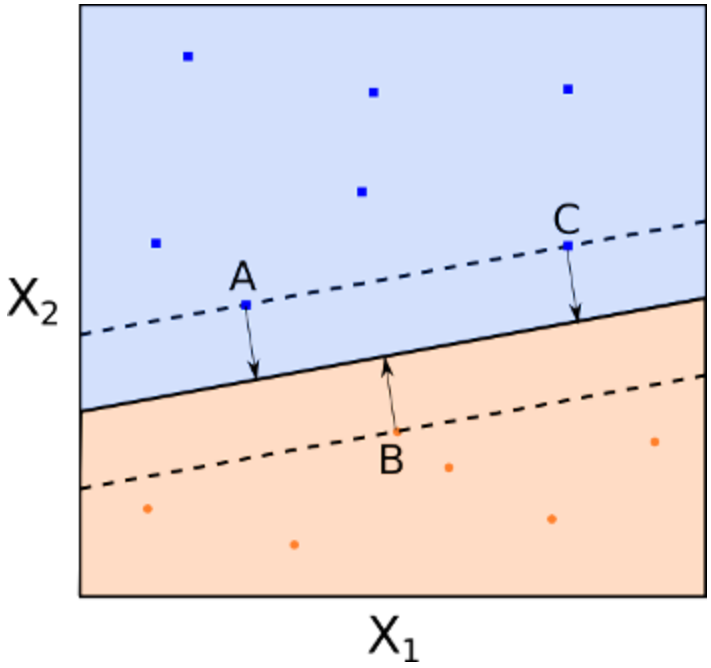
The model may become very complex due to lots of different words in the dataset, which leads to different columns for each word. So, Dimensionality reduction is performed to reduce the number of features or columns from the dataset, without losing on relevant information from all the features. There are different ways to implement dimensional reduction, most important of which is Principal Component Analysis (PCA). PCA decomposes the matrix by finding correlated features or dimensions that can be collapsed as one dimension which can act as a proxy for all the correlated columns that it replaces.
In this study, Truncated SVD is applied, as it is mostly used on sparse data matrices.SVD or Singular value Decomposition, reduces the matrix into its components to simplify the model. The reason PCA is not used here is that PCA is mostly performed on covariance matrix, but SVD directly decomposes the data matrix.

$$\mathbf{M} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^*$$

Every m*n matrix can be factored into U, sigma, and V*. U is an orthogonal matrix, sigma is a diagonal matrix, V* is also an orthogonal matrix. These can be called rotation, stretching and rotation matrix. The diagonal values in sigma are called singular values, the values in U and V are called the left and right singular vectors respectively. The Truncated SVD will give out the matrix with the specified number of components or columns.

Different ML models:

1) **SVM:**
   Support Vector Machines are Machine Learning techniques used for classification problems by finding n-dimensional hyperplanes which would separate the data points and classify them in different groups. The problem in SVM reduces to finding the hyperplanes which maximize the distance between the data points of different classes, known as the Maximal Margin Classifier. Support Vectors are the data points that lie closest to the hyperplane, and the hyperplane is chosen such that it is at maximum distance from all the support vectors.



Points A,B,C are the Support Vectors, and they are at a maximum distance from the hyperplane in the middle. The Maximal Margin Hyperplane is only dependent on the support vectors, and not on any other points, so SVM uses this to its advantage and stores only the support vectors in the memory instead of all the points, which saves a lot of memory.

The distance of any line, ax + by + c = 0 from a given point, (x0 , y0) is given by d = abs(ax0 ,by0+c)/(sq.root(a^2+b^2). In the same way, distance of a hyperplane equation: wTΦ(x) + b = 0 from a given point vector Φ(x0) is:

$$d_H(\phi(x_0)) = \frac{|w^T(\phi(x_0)) + b|}{||w||_2}$$

Where ||w||2 is the Euclidean normal form of distance. To find the Maximal Margin Hyperplane, the distance between the support vectors and the hyperplane should be maximized, and the support vectors are at the closest distance from the hyperplane. So, the SVM problem reduces to maximizing the minimum distance from the hyperplane.

$$w^* = arg_w max\ [min_n\ d_H(\phi(x_n))]$$

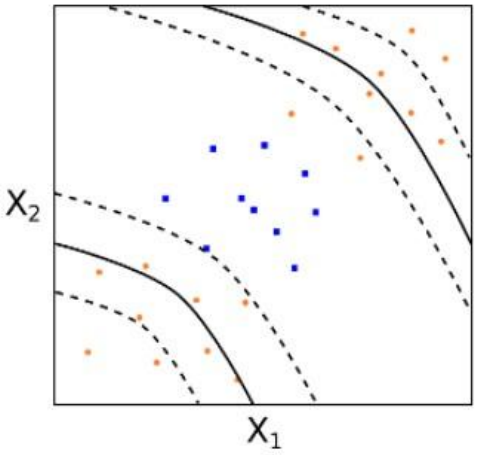
This works well when the boundaries are clearly defined, and no data points overlap other classes. This also works only when the classes are linearly separable. To incorporate for the separation of higher order data, and non-linear separations, SVM uses the kernel trick.

Kernel Functions:

To train for non-linear separations, the feature space can be enlarged by taking the square or cube, or any appropriate power of the data items. The functions used to enlarge this feature space are called the kernel functions. The kernel functions convert the lower dimensional space to a higher dimensional space, but the mapping of the values to higher powers is not done explicitly. The kernel functions are based on calculating the dot products of the data points which are represented as n dimensional vectors and extended to a higher degree by finding the power of the kernel function. There are different types of kernel functions which can be used depending on the nature of the data items:
1)Polynomial kernel
2)Radial Basis Function kernel (RBF)
3)Sigmoid kernel

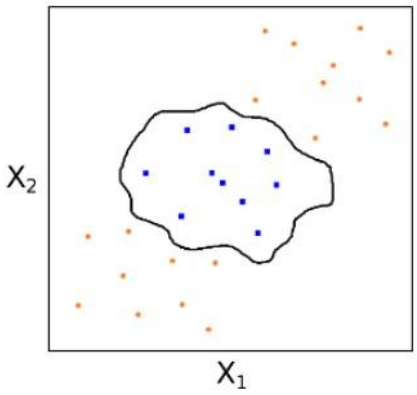
$$K(\vec{x}_i, \vec{x}_k) = (1 + \sum_{j=1}^{p} x_{ij} x_{kj})^d$$

This function is called the polynomial kernel with degree d, which enables the SVM to learn hyperplanes that are not just linear, but also of higher degree.



As the degree of the polynomial kernel grows, it becomes prone to overfitting the data, which leads to improper generalization of the model to new data points. When the data items of one class are enclosed within the data items of other classes, where there can be no clear polynomial function boundaries, the RBF kernel is used to classify.

$$K(\vec{x}_i, \vec{x}_k) = \exp\left(-\gamma \sum_{j=1}^{p} (x_{ij} - x_{kj})^2\right), \quad \gamma > 0$$



Hyperparameters of SVM:

1) Cost (C):

This hyperparameter controls the error in the SVM model. Low value of C indicates that the error is low, and high value indicates that error is high. Even though low C means that the error on the training set is low, it may not generalize well with the testing data. Low C might mean that the model fits the training data so well that there is minimal error, but it fails to provide a good accuracy on the testing data, which indicates that the model is overfitting the data. This is also called the bias-variance trade off in Machine Learning. High C may mean that the training data bias is high, but it will lead to low variance or a well generalizable model. On the other hand, a low C could lead to low training bias but very high variance. Generally the values of C are varied as 0.001,0.01,0.1,1,10,100, and the C value which gives the best testing accuracy is chosen.

2) Gamma:

This is specifically related to the RBF kernel, which is responsible for the degree of curvature of the hyperplane decision boundary. If the Gamma value is high, it indicates that the curvature of the boundary is more, and if the Gamma value is low, the curvature is low. The Gamma values are varied as ‘scale’, ‘auto’, 0.0001,0.001,0.01,0.1,1. The value ‘scale’ means that it uses 1/(num_features * X.var()), and ‘auto’ means it uses 1/num_features. The Gamma value in conjunction with the C value which gives the highest accuracy is chosen as hyperparameters in the model.

2) K-Nearest Neighbor:

K-NN is a supervised algorithm that follows the principle that every data point falling near each other will fall in the same class. K-NN algorithm finds the similarity between the new incoming data and already available data and assigns the new data to the category most similar to the available categories. The best choice of values is highly data-dependent. In general, a larger K suppresses the effects of noise but blurs the classification boundaries.

Choosing the right value of K:

In order to select a suitable K for the data, the KNN algorithm was run multiple times with different K values, reducing the number of errors encountered while preserving the algorithm's

ability to make accurate predictions of choosing K to hit when data hasn't been seen before. The value of k chosen close to 1 makes the predictions less stable. On the other hand, increasing k makes predictions more stable due to averaging resulting in more accurate predictions. Also, based on a general thumb rule that the k value should be chosen as (sqrt(N)/2), where N is the number of data samples.

Math behind K-NN

Finding nearest neighbors can be thought of as finding the closest point from the input point. All the available points are stored and are further classified into classes using majority votes of K neighbors. While implementing K-NN, first step involves transformation of data points into their mathematical values (vectors). It works by calculating the distance between the each data point and test data and eventually finding the probability of points being similar to test data. Classification is based on which points share the highest probabilities. The distance function can be Euclidean, Minkowski, or Hamming distance.

- Euclidean Distance Function: Euclidean distance is the shortest distance between two points irrespective of their dimensionality. The distance between two points with coordinates (x,y) and (p,q) in the coordinate plane is given by

$$\text{dist}((x, y), (p, q)) = \sqrt{(x - p)^2 + (y - q)^2}$$

  The algorithm finds the k nearest neighbors of the given data point and assigns it to the class having the highest number of data points among all the other classes.

- Minkowski Distance Function: Minkowski distance is measured in N-dimensional space. Euclidean distance considers only 2-dimensional space whereas Minkowski is a generalization of the Euclidean distance.

  Consider two points X and Y such that

  $X = x_1, x_2, x_3, \ldots x_n$ and $Y = y_1, y_2, y_3, \ldots y_n$

  The distance between X and Y is given by:

$$D(X,Y) = \left(\sum_{i=1}^{n} |x_i - y_i|^p \right)^{1/p}$$

- Hamming Distance Function: Hamming distance is a metric for comparing two binary data strings of equal length. It is the number of bit positions in which the two bits are different. Hamming distance between two strings is calculated by XOR operation between the strings and counting the number of 1s in the resultant string.

  Suppose there are two strings 100101001 and 101111010.

  100101001 ⊕ 101111010 = 001010011. Since, the result contains four 1's, the Hamming distance, d(100101001 , 101111010) = 4.

Naive Bayes:

Naive Bayes Classifier is a collection of classification algorithms based on Bayes' theorem. It is a family of algorithms that share common principles i.e., each pair of classified features is independent of each other. Bayes' theorem determines the probability of an event occurring given the probability of another event that has already occurred. Bayes' theorem is expressed mathematically as the following equation:

$$P(A|B) = \frac{P(B|A)P(A)}{P(B)}$$

Let X, Y, and Z denote three sets of random variables. The variables in X are said to be conditionally independent of Y, given Z, 1f the following condition holds:

P(XIY, Z) = P(XIZ)

There are 3 kinds of Naive Bayes models under the Scikit-learn library:

1. **Gaussian model:** It is used for classification of data with continuous variables and assumes that every class follows a normal distribution.
2. **Multinomial model:** It is used for classification of data with discrete variables and assumes that every class follows and it works on Bernoulli trials.
3. **Bernoulli model:** It is useful when variables in classification are binary for example: It can classify a document as whether it occurs in a given class or does not occur in a given class.

There are many classifiers over the internet to classify numeric data but very few algorithms exist for classification of text data. Naive bayes is an ML algorithm for classifying text data which is a Bayesian model. This is called naive because it makes certain naive assumptions which reduces the accuracy of the algorithm.
Naive Bayes algorithm classifies text-data into different classes based on the probability of every class given document. After computation whichever class has the highest probability the document is classified into that class. It uses strong assumptions about independence of features to calculate the desired probability.

Math behind Naive Bayes:

With the conditional independence assumption, instead of computing the class-conditional probability for every combination of X, we only have to estimate the conditional probability of each X_i, given Y. The latter approach is more practical because it does not require a very large training set to obtain a good estimate of the probability. To classify a test record, the naive Bayes classifier computes the posterior probability for each class Y:

$$P(Y \mid \mathbf{X}) = \frac{P(Y) \prod_{i=1}^{d} P(X_i \mid Y)}{P(\mathbf{X})}$$

Since P(X) is fixed for every Y, it is sufficient to choose the class that maximizes the numerator term, P(Y) $\prod_{i=1}^{d}$ P(X,IY). In the next two subsections, we describe several approaches for estimating the conditional probabilities P(X,IY) for categorical and continuous attributes

Laplace Smoothing:

Data pre-processing for laplace smoothing:

As mentioned above the dataset needs to be pre-processed before running the algorithm onto it. The dataset used is modified into a dataframe containing the class ID , document ID , word ID and the frequency of that word for a given document in a class.
First the words are extracted from every document and class of the training dataset, then the words stemmed using the Lancaster Stemmer from the NLTK. After calculating some probabilities which are required in the algorithm, stop words(taken form NLTK) are also removed from the modified dataset.
Instead of creating vector representations of the textual data, a DataFrame is created. Every word extracted in the above process is assigned an ID, the class and documents are also assigned an ID independent of each other. For applying laplace smoothing we also need frequency of every word in a given document, for which iterating over every document we calculated the frequency of every word in a document.
Finally a DataFrame is constructed containing the class ID , document ID , word ID and the frequency of that word for a given document in a class, an example is shown below:

	Class_id	document_id	word_id	word_frequency
0	1	1	1	3
1	1	1	2	2
2	1	1	3	8
3	1	1	4	11
4	1	1	5	12
...	...	...	...	...
198712	10	1000	234	0
198713	10	1000	288	0
198714	10	1000	3193	0
198715	10	1000	187	0
198716	10	1000	97	0

198717 rows × 4 columns

Mathematical foundations and modifications:

The general Bayes' theorem is modified to find the required probability which is the probability of every class given document.
The first step is to calculate the probability of every class in the training dataset:

$$probability_{class} = \frac{number\ of\ documents\ in\ a\ given\ class}{total\ number\ of\ documents}$$

The second step it to generate the probability of every word in a given class:

$$probability(word \mid class) = \frac{word_{word,\ class}}{word_{allWords,\ class}}$$

Now, since some words will have zero probability as every word is not contained in every class we add a certain initial probability to every word due to which there is no word left having zero probability given class. This process is also called laplace smoothing and in the implemented naive bayes model we have shown the difference in errors after applying laplace smoothing and before applying laplace smoothing.

$$probability(word \mid class) = \frac{word_{word,\ class} + \alpha}{word_{allWords,\ class} + number\ of\ total\ words\ in\ the\ dataset}$$

In the applied model the value of alpha is taken as 1 which means that every word appears at least once in the given class and adding a factor of number of total words in the dataset normalizes the value of probability such that it may not take much higher values for redundant words.

Third, now from the Bayesian Theorem the final formula for computing probability of class given document becomes:

$$prob(Class) = (probability_{class}) \times (\prod_{word=1}^{total\ words} prob(word \mid class)^{frequency\ of\ word})$$

Now, if the frequency of the word is too high then it's probability will be very high as compared to others. To smooth this we take the logarithm of probability.
So, our final model after all the modification becomes:

$$prob(Class) = log((probability_{class}) \times (\prod_{word=1}^{total\ words} prob(word \mid class)^{frequency\ of\ word}))$$

Results:

Various modifications were tried on different levels of the Machine Learning pipeline right from the preprocessing stage in which two different approaches of Stemming and Lemmatization were tried, then in the Vectorization or Data Embedding three different methods of Bag of Words (TF-IDF), Word2Vec embeddings and Transfer Learning technique which used pretrained weights on the Google News dataset, which consisted of vocabulary of 100 billion words were used. In Word2Vec, two different techniques of Continuous Bag of Words (CBOW), and Skip-Gram were used. In the next stage, the model accuracy while using dimensionality reduction and without using dimensionality reduction (Principal Component Analysis) was checked. The model has more features when dimensionality reduction is not used, but the model is fairly more complex and computationally inefficient than the one which uses dimensionality reduction. Finally three different Machine Learning models were used for document classification : Support Vector Machine (SVM), K-Nearest Neighbors (KNN), and Naive Bayes. The metrics used for the models are accuracy, precision, recall, and f1-score. All the above combinations were tested and compared based on the above metrics.

1) Baseline accuracy: Before applying any of the methods, a simple baseline accuracy was calculated on raw data by training the three models without any efficiency, and accuracy enhancing techniques. The models were trained without using any Data Embedding techniques, without using dimensionality reduction, and without using any cross validation techniques. For vectorizing the data simple frequency count using CountVectorizer was used.

Root Form	SVM	KNN	Naive Bayes
Stemming	0.8812	0.4663	0.8693
Lemmatization	0.8800	0.4525	0.8808

2) The next series of improvements were made to reduce model complexity. Models were compared on the basis of use of dimensionality reduction. This was done to check if there is significant reduction in accuracy if the features are reduced:

Differentiation	SVM	KNN	Naive Bayes
PCA	0.8913	0.8488	0.59778
Without PCA	0.8812	0.4663	0.8693

3) Testing for data embedding techniques. This model was trained using Stemming for preprocessing, and involves dimensionality reduction using PCA. The purpose is to compare Bag of Words (TF-IDF Vectorizer), Word2Vec (CBOW, Skip-Gram), and Pretrained model.

Data Embedding	SVM	KNN	Naive Bayes
TF-IDF	0.9562	0.9413	0.9144
CBOW	0.8959	0.6605	0.4913
Skip-Gram	0.9285	0.8922	0.8379
Pretrained model	0.9525	0.9050	0.8854

Conclusion

The following conclusions can be drawn from the study

1) Stemming and Lemmatization give similar results for all the three models. Stemming has marginally better results, and a slightly less complex model because the number of words in the sparse matrix are less as the root word of multiple words is the same without the context taken into consideration as opposed to the Lemmatization technique. Also, Stemming is preferred for tasks where context is not important.
2) SVM and KNN work best when dimensionality reduction is applied. Although SVM differs only slightly with dimensionality reduction, KNN performs badly without PCA. Naive Bayes being a probabilistic model, performs well with all the features. (For the rest of the study, SVM and KNN were used with Stemming and PCA reduction while Naive Bayes was used with Stemming and without PCA reduction.)
3) The best data embedding technique was found out to be TF-IDF and the Transfer Learning model on Google News dataset. TF-IDF proved to work on all the three ML models providing the best accuracy.
4) Skip-Gram performed better than CBOW in all the models, confirming Mikolov, et al findings that Skip-Gram works well for small-medium sized datasets with words that are less frequent.
5) Laplace Smoothing when applied as an extension of Naive Bayes reduces the error rate of the Naive Bayes model.
6) The study finally proposes that for document and text classification problems, the following workflow should be followed. Stemming for data preprocessing, TF-IDF Vectorizer, dimensionality reduction with PCA, and Support Vector Machine model.

1) Yadnesh P Mundhada 2019B3A70394P

2) **Yash Yadav 2021H1120269P**
3) **Shivansh Jain 2021AA1634P**