**PREDICTING BITTER PEPTIDES FROM SEQUENCE USING MACHINE LEARNING TECHNIQUES**

A Thesis

by

Nishitha Yendapally

Submitted to the College of Graduate Studies

Texas A&M University-Kingsville

in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

August 2022

Major Subject: Computer Science

PREDICTING BITTER PEPTIDES FROM SEQUENCE USING MACHINE LEARNING TECHNIQUES

A Thesis

by

NISHITHA YENDAPALLY

Approved as to style and content by:

|  |
| --- |
| Avdesh Mishra, Ph.D.  Committee Chair |

|  |  |  |  |
| --- | --- | --- | --- |
| Ayush Goyal, Ph.D.  Committee Member |  |  | Mais Nijim, Ph.D.  Committee Member |

|  |  |  |  |
| --- | --- | --- | --- |
| Lifford McLauchlan, Ph.D.  Interim Department Chair |  |  | Robert J Diersing, Ph.D.  Interim Vice President for Research  and Graduate Studies |

August 2022

**ABSTRACT**

Predicting Bitter Peptides from Sequence using Machine Learning Techniques

(August 2022)

Nishitha Yendapally, Bachelor of Technology in Computer Science

Jawaharlal Nehru Technological University - Hyderabad

Chairman of the Advisory Committee: Dr. Avdesh Mishra

In this post-genomic era, the prediction of peptide bitterness directly from the sequence is an important element in developing drugs and nutritional studies. Identifying bitter peptides using an experimental method such as *in-vivo* and *in-vitro* approaches that include “human taste panel studies” can be arduous, tiresome, time-consuming, and costly. In contrast to the state-of-the-art approaches, this thesis proposes to explore various machine learning techniques specifically, a stacking-based approach for an effective prediction of bitter peptides directly from the peptide sequence. In addition, this work explores various feature encoding and transformation techniques such as Term Frequency-Inverse Document Frequency, K-mers Feature Extraction, and Count Vectorizer. Finally, to validate the robustness, the proposed stacking-based model is compared with the other machine learning algorithms as well as an existing method by utilizing the standard performance evaluation metrics such as Sensitivity, Accuracy, Specificity, and Mathew’s Correlation Coefficient obtained through cross-validation and independent testing.

**DEDICATION**

This thesis is dedicated to my parents Mr. Venkatesham Yendapally and Mrs. Jayatha Yendapally, and my sister Ms. Harshitha Yendapally for their constant support without which I would not be able to complete this research work. I humbly submit my research work to help speed up and enhance the development of new drugs.

**ACKNOWLEDGEMENTS**

I express my sincere gratitude and thank my thesis supervisor, Dr. Avdesh Mishra, for providing constant motivation and guidance in all stages of my research. I am grateful to him for his valuable advice and prospective thinking which helped me to carry out research at TAMUK effortlessly.

I would like to thank Dr. Ayush Goyal and Dr. Mais Nijim for being part of my thesis committee and a constant source of encouragement throughout my research. I am also thankful to each of them for showing keen interest and reviewing my thesis. Moreover, I appreciate the Computer Science Department, TAMUK for the help and support during my academic course work at TAMUK.

Furthermore, I greatly express my sincere gratitude to Dr. Watshara Shoombuatong for providing the peptide dataset to conduct this research.

**TABLE OF CONTENTS**

Page

ABSTRACT iii

DEDICATION iv

ACKNOWLEDGEMENTS v

TABLE OF CONTENTS vi

LIST OF FIGURES vii

LIST OF TABLES ix

CHAPTER 1. INTRODUCTION 1

CHAPTER 2. DATA AND FEATURE COLLECTION ......................................................... 4

2.1 DATA SET 4

2.2 FEATURE REPRESENTATION 4

2.3 FEATURE EXTRACTION 5

2.2.1 K-MER EXTRACTION 5

2.2.2 NLP TECHNIQUES 5

2.2.2.1 TERM-FREQUENCY INVERSE DOCUMENT FREQUENCY (TFIDF) 6

2.2.2.2 COUNT VECTORIZER 7

CHAPTER 3. MACHINE LEARNING ALGORITHMS AND ASSESSMENT METRICS 8

3.1 MACHINE LEARNING METHODS 9

3.2 ARCHITECTURE OF THE MODEL 11

3.3 METRICS TO EVALUATE PERFORMANCE 12

CHAPTER 4. THE PROPOSED PREDICTION FRAMEWORK 14

4.1 STACKING BASED FRAMEWORK 14

CHAPTER 5. RESULTS AND CONCLUSIONS 16

5.1 PERFORMANCE COMPARISION ON THE BENCHMARK DATASET . 15

5.2 PERFORMANCE COMPARISIONS ON THE INDEPENDENT TEST DATASET 18

5.3 CONCLUSIONS 21

REFERENCES 23

VITA 25

APPENDIX 26

**LIST OF FIGURES**

Page

Fig 1. ARCHITECTURE OF THE GENERAL STACKING BASED MODEL. 12

[Fig 2. PROPOSED FRAMEWORK (STACKING-BASED MODEL) . 1](#_Toc75900459)5

[Fig 3. COMPARISION OF ACCURACY SCORES USING 10-FOLD CROSS VALIDATION ON THE BENCHMARK DATASET . 1](#_Toc75900459)8

[Fig 4. COMPARISION OF ACCURACY SCORES ON THE INDEPENDENT TEST DATASET. 2](#_Toc75900459)1

**LIST OF TABLES**

Page

Table 1. NUMBER OF BITTER AND NON-BITTER PEPTIDES. 4

Table 2. EVALUATION METRICS AND THEIR DEFINITION 13

Table 3. PERFORMANCE OF THE STACKING-BASED MODEL USING 10-FOLD CROSS-VALIDATION 16

Table 4. COMPARISON OF STACKING-BASED ALGORITHM WITH ML ALGORITHMS……… 17

Table 5. PERFORMANCE OF STACKING-BASED MODEL ON THE INDEPENDENT TEST DATASET 19

Table 4. COMPARISON OF STACKING-BASED ALGORITHM WITH ML ALGORITHMS ON THE INDEPENDENT TEST DATASET……… 20

**CHAPTER 1. INTRODUCTION**

Peptides are small chained amino acids and smaller versions of proteins. Amino acid chains with fewer than 20 are called, oligopeptides which include [dipeptides](https://en.wikipedia.org/wiki/Dipeptide), [tripeptides](https://en.wikipedia.org/wiki/Tripeptide), and [tetrapeptides](https://en.wikipedia.org/wiki/Tetrapeptide). They are found to be useful in the development of drugs to combat critical diseases. Many of these drugs have a bitter taste by nature. Significant efforts are being made to reduce the bitterness of these medicines to enhance taste and, as a result, increase drug compliance [1] [2] [3]. The experimental methods such as *in-vivo* and *in-vitro* approaches that include “human taste panel studies” [4], utilized to identify bitter peptides, are tedious, expensive, and time-consuming. In nutritional studies, identifying the bitter peptides plays an essential role. The need to create computational algorithms for quickly and accurately differentiating bitter from non-bitter peptides (BPs) is important, given that in the post-genomic age, the volume of peptides produced. In this thesis, a computer model called, the stacking-based model is introduced that utilizes features extracted from K-mers extracting technique, TFIDF, and a count vectorizer to predict peptide bitterness.

## 1.1 LITERATURE REVIEW

The literature presents several remarkable research on identifying bitter peptides. Several authors have used machine learning and statistical methods for identifying the peptides' bitterness [5] [6] [7]. The first method that utilizes a computational model for the prediction of bitterness in peptides is, iBitter-SCM [8]. It uses the scoring card approach with propensity ratings for 400-dipeptides and 20-amino acids to differentiate the bitterness and non-bitterness of the peptides. In their work, 10-fold CV and an independent-testing method to examine five machine-learning classifiers: K-Nearest Neighbour (KNN), Random Forest (RF), Nave Bayes (NB), Support Vector Machine (SVM), and Decision Tree (DT) are utilized. As reported, iBitter-SCM [8], obtained a 10-fold CV score and accuracy of 0.871 and 0.844, respectively on an independent test dataset. Although iBitter-SCM produced relatively high forecast accuracies as mentioned above, its prediction performance should be further improved before it can be utilized in real-world applications. One of the shortcomings of iBitter-SCM is that it utilizes a single feature descriptor. The other method is BERT4Bitter [5] which, applies bidirectional encoder representation from transformer (BERT) technique, a deep learning technique, for predicting BPs. Peptide sequences are the input for BERT4Bitter, which uses them to automatically generate feature descriptors without the requirement for deliberate feature encoding design or selection. BERT4Bitter uses raw peptide sequences instead of feature encodings that have been carefully designed and chosen. The prior approaches, however, mostly depend on features that must be manually and laboriously processed from raw peptide sequences. Amino acid index (AAI), Amino acid composition (AAC), pseudo amino acid composition (PseAAC), dipeptide composition (DPC), and tripeptide composition are examples of sequence-based feature encodings (TPC) that have been explored by prior approaches. Although BERT4Bitter yields better results, it is not directly comparable to the method proposed in this thesis because BERT4Bitter is based on deep learning techniques. Instead, the proposed approach is compared with iBitter-SCM, a method that utilizes individual machine learning algorithm.

To develop traditional machine learning predictors, it is well acknowledged that combining several feature descriptors may significantly enhance prediction performance when compared to a single feature descriptor. However, combining multiple feature descriptors may complicate the problem further as it leads to the problem of the curse of dimensionality. Moreover, combining feature descriptors may involve the inclusion of redundant and noisy information, resulting in poor prediction outcomes. While the identification of useful features using a feature selection algorithm can be employed to address the challenge of high-dimensional feature space, this procedure is time-consuming since it involves numerous manual, laborious, and trial-and-error attempts. Moreover, the development of traditional machine learning algorithms is rather difficult, requiring extracting features, feature significance identification, and prediction model optimization.

Compared with the iBitter-SCM, the stacking-based approach is observed to be more effective. In the proposed stacking-based approach, K-mers extraction technique is utilized to extract k-mers, specifically 2-mers, which are then passed to the NLP technique, called TFIDF and count vectorizer for feature extraction.

This research proposes to explore various machine learning techniques specifically, stacking-based approaches for an effective prediction of bitter peptides directly from the peptide sequence. In addition, this work explores various feature encoding and transformation approaches such as TFIDF and K-mers extraction technique to extract useful features. Besides the feature encoding/extraction techniques utilized in this thesis, this work can be further improved by exploring other feature encoding techniques such as Pep2Vec (a Word2Vec-inspired technique) [5], FastText [5], and residue-wise contact energy matrix transformation.

**CHAPTER 2. DATA AND FEATURE COLLECTION**

This chapter explains the proposed dataset, feature extraction techniques, feature representation, and feature evaluation metrics in more detail.

# **2.1 DATASET**

In the proposed study, the BTP640 dataset that was previously established by Charoenkwan et al. [5] is utilized. It contains 320 BPs and 320 non-redundant non-BPs. To build the prediction model and validate its generalization capacity [7] [8] [11] [12] the BTP640 benchmark dataset is divided randomly into a BTP-CV (training subset) and BTP-TS (independent test subset) with 0.8:0.2 of split ratio. The dataset training subset contains 256 BPs and 256 non-BPs [9] that are utilized for validation and training. On the other hand, BTP-TS contains 64 BPs and 64 non-BPs that are utilized for independent testing. The previous research contains more details on the benchmark and independent test dataset [10] [7] [11] [6].

**Table1.** Number of BPs and Non-BPs

|  |  |  |  |
| --- | --- | --- | --- |
| **Dataset** | Total Number of peptides | Bitter-Peptides | Non-Bitter Peptides |
| Benchmark Dataset | 640 | 320 | 320 |
| Training Dataset (BTP-CV) | 512 | 256 | 256 |
| Testing Dataset (BTP-TS) | 128 | 64 | 64 |

**2.2 FEATURE REPRESENTATION**

A peptide sequence (P) can be represented as follows:

**P**= p1 p2 p3…pi … pN (1)

where pi and N stand for the protein P's ith residue and the length of the peptide, respectively. The residue pi is a member of the natural amino acid group, which also consists of the following amino- acids such as A, C, D, E, F, G, H, K, I, L, M, N, P, Q, R, S, T, V, W, and Y.

# **2.3 FEATURE EXTRACTION**

In this research, various peptide sequence-based feature extraction techniques for the accurate prediction of BPs are explored. The feature extraction techniques that are explored in this project are TFIDF and count vectorizer which utilizes 2-mers that are extracted utilizing the K-mer extraction technique.

**2.3.1 K-mers Extraction:**

This is the most crucial step in evaluating the sequence data in which the set of fixed-size chunks, rather than the sequence, can be efficiently analyzed by breaking a sequence down into its K-mers. Set operations are quicker and easier, and there are many widely available algorithms and strategies to work with them. K-mers are highly useful for sequence matching (string matching with n-grams has a rich history). In K-mers extraction, the substrings with length k in a string S or collection of strings is extracted, where k is a positive integer. In this research, we utilize the K-mers extraction technique to extract 2-mers, where the peptide sequences are broken down into the composition of two amino acids (dipeptides).

**2.3.2 NLP Techniques**

NLP-based approaches have so far been applied successfully in a variety of fields, including bioinformatics and drug development. The automatic representation of unprocessed raw data into a set of features is one of the most effective and advantageous NLP-based techniques (providing raw input data features). In this model NLP concepts are utilized where the sequence of peptides are transformed to the n-Dimensional word vector. In this technique amino acids sequences are represented as word. In this TFIDF and count vectorizer are utilized to perform the extraction of the features.

**2.3.2.1 Term Frequency Inverse Document Frequency (TFIDF)**

TFIDF is a statistical metric that assesses how frequently a word appears in a document within a collection of documents. It is a popular method for document representation in Natural Language Processing (NLP) that is based on information retrieval techniques [12]. This method is frequently used to assist model developers in representing documents [12] [13]. TFIDF is split into two parts TF (Term Frequency) and IDF (Inverse Document Frequency). TF indicates the number of times a word *i* (*ti*) appears in a particular document *j* (*dj*), and IDF denotes the inverse document frequency for the target word. In this proposed model, the K-mers (dipeptides) extraction technique is used to extract 2-mers which are then passed to the TFIDF module to convert the peptide sequence into a vector of 316-dimensions. 316 features are extracted from both BPs and Non-BPs sequences.

The Feature extraction are performed in the following way as shown in the below example. Let’s consider a document of sentences, where each word is considered as a token.

Example: Let us consider a corpus of documents as shown below.

corpus = [

'This is my Thesis document.',

'This document is my second document.',

'And this is my third one.',

'Is this my Thesis document?',

]

Output: The output of the TFIDF method include features: ['and', 'document', 'Thesis', 'is', 'one', 'second', 'my', 'third', 'this'].

**2.3.2.2 Count Vectorizer**

Count vectorizer is the other feature extraction technique which is used to convert the peptide sequence into a feature vector. This technique is utilized to convert a corpus of text into a vector of tokens/term counts or n-grams. Text data is preprocessed before creating the vector representation, making it a very versatile feature representation module for text. In this proposed model, first, 2-mers are extracted from the peptide sequences then count vectorizer is utilized to convert sequence of 2-mers into a numerical feature vector of 2-mers count. Through count vectorizer, 316 features were extracted.

The Feature extraction are performed in the following way as shown in the below example. Let’s consider a document of sentences, where each word is considered as a token.

Example: Let us consider a corpus of documents as shown below.

corpus = [

'This is my Thesis document.',

'This document is my second document.',

'And this is my third one.',

'Is this my Thesis document?',

]

Output: The output of the Count Vectorizer method include features: ['and', 'document', 'Thesis', 'is', 'one', 'second', 'my', 'third', 'this'] and the corresponding feature vectors can be represented as [[0 1 1 1 0 0 1 0 1], [0 2 0 1 0 1 1 0 1], [1 0 0 1 1 0 1 1 1], [0 1 1 1 0 0 1 0 1]].

**CHAPTER 3. MACHINE LEARNING ALGORITHM AND ASSESSMENT METRICS**

This chapter discusses the machine learning methods explored in this thesis. In addition, it describes a general architecture of the proposed stacking-based model that stacks individual machine learning algorithms for better performance. Additionally, it presents a discussion on the performance assessment techniques that are adopted to examine the effectiveness of the model.

**3.1 Machine Learning Methods**

Here, the machine learning methods explored in this work are described.

1. Naïve Bayes method is a supervised learning algorithm that is based onB . It uses a probabilistic machine learning scheme to learn the model.It is used for atasksTo classify samples, it makesa areBforIn this thesis, the Gaussian Naïve Bayes method is adopted to classify bitter peptides. maintainsthe property of In this implementation, parameter, var\_smoothing is set to a value of 10e-30.
2. s The parameters used for defining the tree in this model is random\_state with the value of 5.
3. Aan Here, parameter, n\_estimators is set to a value of 10.
4. a theis and the maximum depth of the tree is 2.
5. can be utilized both SVM is also known as Maximum Margin Classifier where it maximizes the margin of the hyperplane. The best hyperplane helps reduce the generalization error.

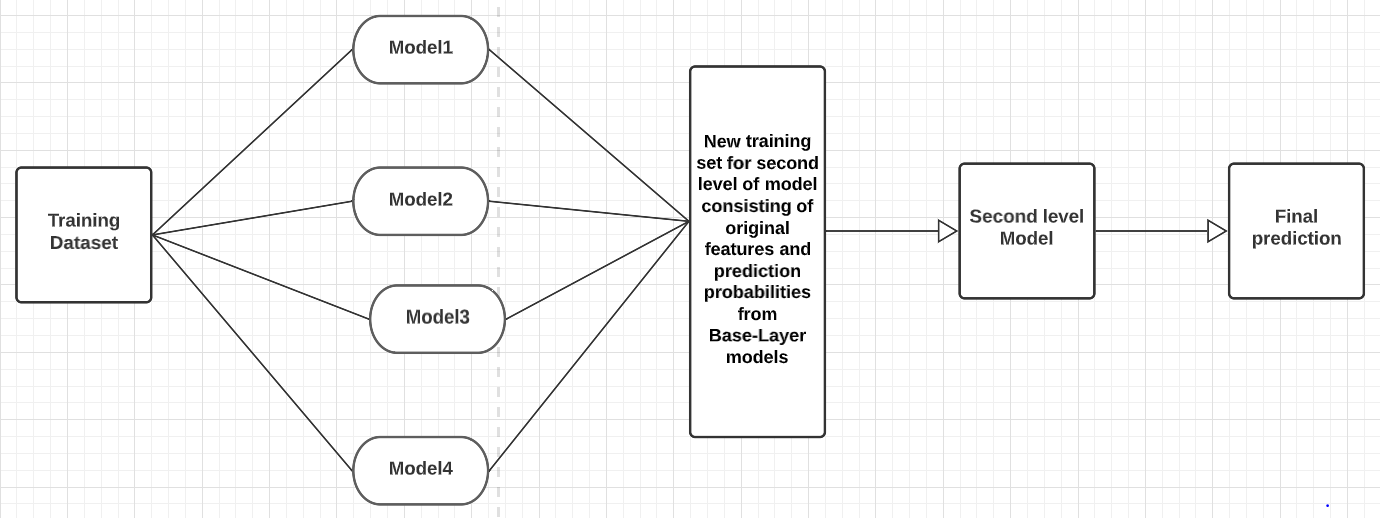
using kernel functionsthe Kernel functions are a technique for transforming data from its input form into the format needed for processing it. The term "Kernel" is used because the Support Vector Machine uses a collection of mathematical operations to provide the window through which the data can be manipulated. Therefore, the Kernel function typically changes the training set of data to enable a non-linear decision surface to transform to a linear equation in a higher number of dimension spaces. The linear kernel with the default gamma value (auto) is used for optimizing the SVMs model.

1. Logistic Regression is also a supervised machine learning approach that can be used both for both classification and also regression. It is using the sigmoid function.It is generally used and the default parameters are used in this implementation.
2. (KNN)problems In this implementation, all the parameters were set to their default value.
3. A Dmaking ons with the default parameters. It in ichand for . Here Lrepresents . In this implementation we, all the default values as optimization parameters.

**3.2 Architecture of the Model**

In this thesis, we utilize a stacking-based model to develop a predictor of BPs from the peptide sequence. Several studies have shown the superiority of a stacking-based model over traditional machine learning algorithms. Stacking helps in improving the performance of the model as the information from one or more models is combined to train a new model. This technique has been widely adopted because of its lower generalization error. It is due to low generalization error; the model yields a more accurate predictor. This thesis implements a 2-layer stacking framework that includes i.) Base-Layer and ii.) Meta-Layer.

The Base-Layer being the first stage, may contain several machine learning models. The models in the Base-Layer have been selected in such a way that their underlying operating principle is different so that they can pass valuable information to the Meta-Layer. The models in the Base-Layer are trained to generate prediction probabilities for each amino acid, which are consequently appended to the original input features of the amino acid and passed on to the Meta-Layer for improved training. The Meta-Layer produces the final predictions. Figure 1 shows a general architecture of the stacking-based model proposed in this work.



**Fig1:** Architecture of a general stacking-based model

## 3.3 Performance Assessment Metrics

Performance assessment is a very important step for the evaluation of ML/deep learning models. In this work, first, the model is trained on the Benchmark training data set (BTP-CV) and then tested on the independent test dataset (BTP-TS) that has both bitter and non-bitter peptides.

A 10-fold CV approach is utilized for comparing the outcomes of the suggested approach with the other Machine learning approaches. First, the data is divided into ten equal-sized folds, where 9-folds are utilized to train, and the remaining fold is utilized to test the method. This procedure continues until each fold is tested at least once. Table 2 shows the performance evaluation metrics (and the corresponding definitions) that are considered for an extensive evaluation of the proposed framework.

**Table 2**: Evaluation metrics and their definition.

|  |  |
| --- | --- |
| **Metric** | **Metric Definition** |
| Sensitivity (Sny) |  |
| Specificity (Spy) |  |
| Accuracy (Acy) |  |
| F1 score |  |
| Mathew’s correlation-coefficient (MaCC) |  |

**CHAPTER 4. THE PROPOSED PREDICTION FRAMEWORK**

In this chapter, the overall process adopted for the design and development of the stacking-based bitter peptides predictor is discussed.

## 4.1 Stacking-based Framework:

The procedures used to create a stacking-based framework for the classification of bitter peptides from the peptide sequence are explained below:

1. First, the BTP640 dataset is collected and utilized as a benchmark dataset. It consists of 640 peptide sequences of which 320 are bitter and the remaining 320 are non-bitter peptides.
2. Next, K-mers extraction techniques are used to extract 2-mers. Then these 2-mers are passed to the TFIDF feature extraction module.
3. Then, feature extraction is performed on the 2-mers sequence. Here, an NLP-based technique called, TFIDF is used to extract features from the peptides available in the dataset.
4. To devise a bitter peptide predictor, features extracted in Steps 2 and 3 are transformed into a feature matrix and used as an input to the machine learning algorithms including stacking. In the proposed model, stacking-based architecture employs three machine learning algorithms, Gaussian NB, Gradient boosting, and ADA-boosting algorithms in the Base-Layer and the Random Forest in the Meta-Layer.
5. Once the training dataset is trained using stacked model, it is evaluated on the entire training dataset as well as on a separate test dataset (independent test dataset). The suggested model's performance is assessed using the performance measures Sny, Spy, Acy, Bacc, and MaCC.

Diagram

Description automatically generated

**Fig2:** Proposed Framework **(**Stacking-based model )

**CHAPTER 5. RESULTS AND CONCLUSIONS**

This chapter describes the techniques used to evaluate the proposed method. Moreover, it discusses the results of this study in comparison with other Machine Learning Approaches. Finally, it concludes the thesis work by summarizing the proposed approach and findings.

## 5.1 Performance comparisons on the benchmark dataset:

Here, the performance of the proposed stacking-based architecture and other individual machine learning models is assessed on the benchmark training dataset. For this, a widely used method called, 10-fold CV is adopted.



**Table 3.** Performance of the Stacking-Based Model on the Benchmark Dataset



|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Benchmark Dataset** | **Sny %** | **Spy %** | **Acy %** | **Bacy %** | **MaCC %** |
| The Proposed Method | 78.5 | 85 | 80.4 | 80.4 | 60.58 |

From Table 3, it is evident that the stacking-based model can achieve sensitivity (Sny), accuracy (Acy), specificity (Spy), and Mathew’s correlation coefficient (MaCC) of 78.5%, 85%, 80.4%, and 60.58%, respectively. These scores indicate that the model can learn the significant pattern from the bitter dataset and differentiate between bitter and non-bitter peptides effectively. The results over the benchmark dataset show that the model has good prediction capabilities.

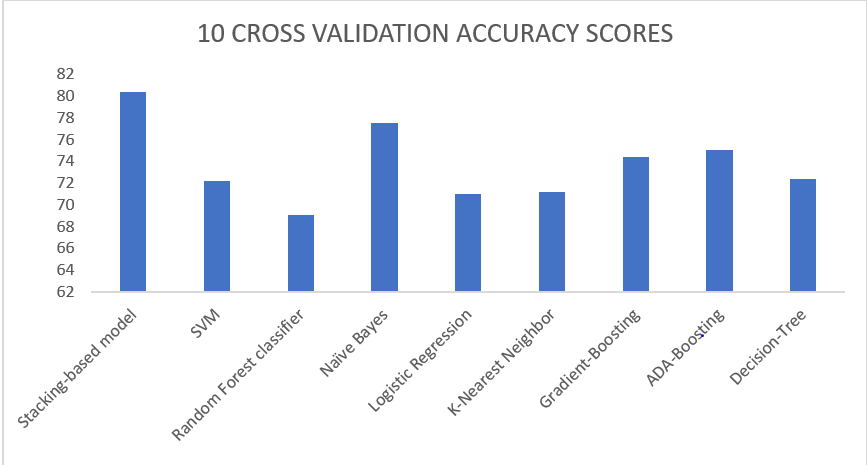
Next, to identify if the proposed stacking-based model is compared better with other individual machine learning models, a comparative study is done where the stacking-based model is compared with individual machine learning techniques. Table 4 presents the results of the comparison between the proposed stacking-based method with other individual machine learning methods.

**Table 4.** Comparison of the stacking-based algorithm with other machine learning algorithms

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Benchmark Dataset** | **Sny %** | **Spy %** | **Acy %** | **Bacy %** | **MaCC %** |
| Stacking-based model | 78.5 | **85.0** | **80.4** | **80.4** | **60.58** |
| SVM | 62.8 | 81.6 | 72.2 | 72.0 | 45.3 |
| Random Forest classifier | **89.8** | 48.4 | 69.1 | 69.1 | 42.0 |
| Naïve Bayes | 78.1 | 76.9 | 77.5 | 77.6 | 55.0 |
| Logistic Regression | 67.2 | 75.0 | 71.0 | 71.0 | 42.3 |
| K-Nearest Neighbor | 60.1 | 82.4 | 71.2 | 71.2 | 43.6 |
| Gradient-Boosting | 83.5 | 65.2 | 74.4 | 74.0 | 49.6 |
| ADA-Boosting | 64.8 | 85.1 | 75.0 | 75.0 | 51.0 |
| Decision-Tree | 66.7 | 78.1 | 72.4 | 72.4 | 45.2 |

From Table 4, it is evident that the proposed stacking-based method outperforms other machine learning models based on specificity(Sny), accuracy(Acy), balanced accuracy(Bacy), and Mathew’s correlation coefficient. The stacking-based model achieves a 10-fold CV specificity(Spy), accuracy(Acy), balanced accuracy(Bacy), and Mathew’s correlation coefficient(MaCC) of 85%, 80.4%, 80.4%, and 60.58%, respectively. Moreover, the sensitivity(Sny) of the Random Forest method is higher than the other methods. However, the sensitivity of the stacking-based methods is comparable with the top three methods Random Forest, Gradient-Boosting, and Naive Bayes. Fig 3 depicts the graph of comparison between stacking model and other machine learning models based on the accuracy scores using 10-Fold CV.

.



**Fig 3:** Comparison of accuracy scores using 10-fold cross validation.

**5.2 Performance comparison on the independent-test dataset:**

Here, the proposed stacking-based architecture and other machine learning methods performance is assessed on the independent test dataset. For this, each method is first trained using the benchmark-dataset and consequently, tested using an independent test dataset. It is made sure that none of the samples present on the independent test dataset is present in the training dataset. Moreover, the proposed method is compared with the existing bitter peptide prediction algorithm called iBitter-SCM, which is the closes method to the proposed algorithm. The findings of the stacking-based model on the independent test dataset as well as its comparison with the existing method are shown in Table 5.

**Table 5.** Comparison of the stacking-based model with the existing model on the independent test dataset.



|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |

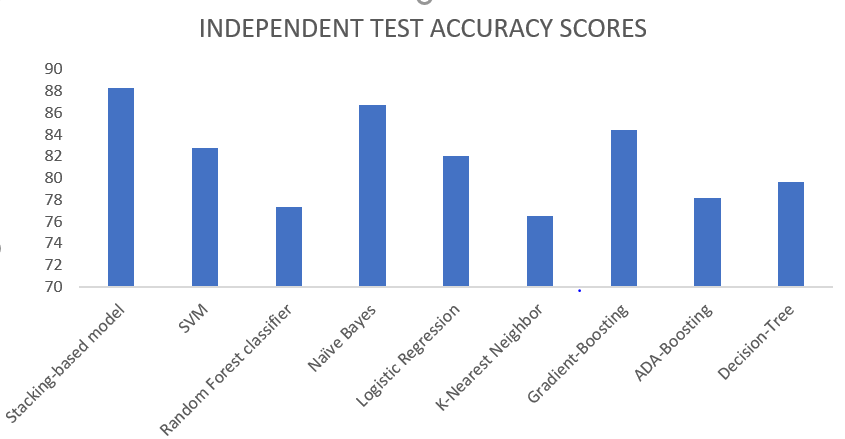
From Table 5, it is evident that the stacking-based model achieves sensitivity (Sny), specificity (Spy), accuracy (Acy), and Mathew’s correlation coefficient (MaCC) of 85.9%, 80.0%, 88.2%, and 76.6%, respectively. Moreover, the stacking-based method yields a percentage improvement of 4.62% and 7% based on critical metric, i.e., Acy and MaCC, respectively. performance comparison between the stacking-based model and the iBitter-SCM shows thatthe stacking-based modeloutperforms

Next, to identify if the proposed stacking-based model is better compared to other individual machine learning methods, a comparative study is done where, the stacking-based model is compared with individual machine learning methods on the independent test dataset. Table 6 presents the results of the comparison between the proposed stacking-based method with other individual machine learning methods.

**Table 6.** Performance comparison of the stacking-based model with other individual machine learning method on the independent test dataset.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Independent Test Dataset** | **Sny %** | **Spy %** | **Acy %** | **Bacy %** | **MaCC %** |
| Stacking-based model | 85.9 | **86.0** | **88.2** | **79.6** | **76.6** |
| SVM | 89.0 | 81.6 | 82.8 | 72.2 | 66.1 |
| Random Forest classifier | 93.7 | 48.4 | 77.3 | 69.1 | 57.8 |
| Naïve Bayes | 89.0 | 76.9 | 86.7 | 77.5 | 73.5 |
| Logistic Regression | 92.1 | 75.0 | 82.0 | 71.0 | 65.4 |
| K-Nearest Neighbor | 78.1 | 82.4 | 76.5 | 71.2 | 53.1 |
| Gradient-Boosting | **95.3** | 68.3 | 84.3 | 74.2 | 70.4 |
| ADA-Boosting | 68.7 | 85.1 | 78.1 | 75.0 | 57.2 |
| Decision-Tree | 71.8 | 78.1 | 79.6 | 72.5 | 60.1 |

From Table 6, it is evident that the stacking-based method outperforms other machine learning algorithms based on specificity (Spy), accuracy (Acy), balanced accuracy (Bacy), and Mathew’s correlation coefficient (MaCC). The stacking-based model achieves specificity(Spy), accuracy(Acy), balanced accuracy(Bacy), and Mathew’s correlation coefficient(MaCC) on the independent test dataset (BTP-TS) of 86%, 88.2%, 79.6%, and 76.6%, respectively. Moreover, the sensitivity of the Gradient-Boosting method is higher than the other methods. However, the sensitivity of the stacking-based methods is comparable with the top three methods Random Forest, Logistic Regression, and SVM. Fig 4 depicts the graph of comparison between stacking model and other machine learning models based on the accuracy scores on the independent Test datasets (BTP-TS).



**Fig 4:** Comparison of accuracy scores on the independent test dataset.

**5.3 Conclusions**

Accurately identifying bitter peptides from peptide sequences is critical for the effective development of novel medicines. Using a computational model that can precisely predict and identify bitter peptides from the sequence. Furthermore, there is a growing need for a powerful algorithm that can assess the peptide sequences to precisely identify bitter peptides given the recent quickly rising peptide count brought on by the success of the Genome project.

In this research, novel stacking-based models are proposed for the prediction of bitter peptides from the sequence. The work began by exploring feature extraction where K-mers extraction techniques followed by NLP techniques are utilized. The NLP (Natural language processing) technique used in this paper is TFIDF. Next, the features are used as an input to train the Machine learning algorithms and proposed stacking-based algorithm, and testing is performed on the independent test dataset. Lastly, the performance of the stacking-based model is assessed using different performance measures via training and independent testing. Moreover, the stacking-based model is compared with the other machine learning algorithms. The results demonstrate that the proposed approach achieves an improvement in accuracy and MaCC metric to be 4% and 7.8% more compare with the previously existing method. The advantage of the suggested method over the best current method suggests that this research has the potential to aid in the identification of significant bitter peptides and contribute to the creation of novel medications.

# **References**

|  |  |
| --- | --- |
| [1] | Dagan-Wiener,A. et al. (2017) Bitter or not? BitterPredict, a tool for predicting taste from chemical structure. Sci. Rep., 7, 1–13. |
| [2] | Huang,W. et al. (2016) BitterX: a tool for understanding bitter taste in humans. Sci. Rep., 6, 23450. |
| [3] | Pripp,A. and Ardo,Y. (2007) Modelling relationship between angiotensin-(I)-converting enzyme inhibition and the bitter taste of peptides. Food Chem., 102, 880–888. |
| [4] | Zheng, Suqing, et al. "e-Bitter: bitterant prediction by the consensus voting from the machine-learning methods." Frontiers in chemistry 6 (2018): 82. |
| [5] | Phasit Charoenkwan, Chanin Nantasenamat, Md Mehedi Hasan, Balachandran Manavalan, Watshara Shoombuatong, BERT4Bitter: a bidirectional encoder representations from transformers (BERT)-based model for improving the prediction of bitter peptides, Bioinformatics, 2021;, btab133, <https://doi.org/10.1093/bioinformatics/btab133> |
| [6] | Phasit Charoenkwan, Janchai Yana, Nalini Schaduangrat, Chanin Nantasenamat, Md. Mehedi Hasan, Watshara Shoombuatong, iBitter-SCM: Identification and characterization of bitter peptides using a scoring card method with propensity scores of dipeptides Genomics,Volume 112, Issue 4,2020,Pages 2813-2822,ISSN 0888-7543, |
| [7] | Arroyo-Ferna´ndez,I. et al. (2019) Unsupervised sentence representations as word information series: revisiting TF–IDF. Comput. Speech Language, 56,107–129. |
| [8] | Charoenkwan,P. et al. (2020a) iAMY-SCM: improved prediction and analysis of amyloid proteins using a scoring card method with propensity scores of dipeptides. Genomics, 112, 2813–2822. |
| [9] | Charoenkwan,P. et al. (2020b) iDPPIV-SCM: a sequence-based predictor for identifying and analyzing dipeptidyl peptidase IV (DPP-IV) inhibitory peptides using a scoring card method. J. Proteome Res., 19, 4125–4136. |
| [10] | Charoenkwan,P. et al. (2020c) iUmami-SCM: a novel sequence-based predictor for prediction and analysis of umami peptides using a scoring card method with propensity scores of dipeptides. J. Chem. Inf. Model., 60,6666–6678 |
| [11] | Charoenkwan,P. et al. (2020d) iBitter-SCM: identification and characterization of bitter peptides using a scoring card method with propensity scores of dipeptides. Genomics, 112, 2813-2822 |
| [12] | Wei,L. et al. (2020) Computational prediction and interpretation of cell-specific replication origin sites from multiple eukaryotes by exploiting stacking framework. Brief. Bioinform., 2020, bbaa275. |
| [13] | A. Aizawa, “An information-theoretic perspective of tf–idf measures.,” *Information Processing & Management,* pp. 45-65, 2003. |
| [14] | Chen,K. et al. (2016) Turning from TF-IDF to TF-IGM for term weighting in text classification. Expert Syst. Appl., 66, 245–260. |
| [15] | Mikolov,T. et al. (2013) Efficient estimation of word representations in vector space. arXiv preprint arXiv:1301.3781. |
| [16] | Tahir,M. et al. (2020) Prediction of N6-methyladenosine sites using convolution neural network model based on distributed feature representations. Neural Netw., 129, 385–391. |
| [17] | Wu,C. et al. (2019) PTPD: predicting therapeutic peptides by deep learning and word2vec. BMC Bioinformatics, 20, 1–8 |
| [18] | Aggarwala,V. and Voight,B.F. (2016) An expanded sequence context model broadly explains variability in polymorphism levels across the human genome. Nat. Genet., 48, 349–355 |
| [19] | Arroyo-Ferna´ndez,I. et al. (2019) Unsupervised sentence representations as word information series: revisiting TF–IDF. Comput. Speech Language, 56,107–129 |
| [20] | Asgari,E. and Mofrad,M.R. (2015) Continuous distributed representation of biological sequences for deep proteomics and genomics. PLoS One, 10,e0141287. |
| [21] | Habibi,M. et al. (2017) Deep learning with word embeddings improves biomedical named entity recognition. Bioinformatics, 33, i37–i48. |
| [22] | Hamid,M.-N. and Friedberg,I. (2019) Identifying antimicrobial peptides using word embedding with deep recurrent neural networks. Bioinformatics, 35, 2009–2016 |
| [23] | Hoque,M.T. et al. (2016) sDFIRE: sequence-specific statistical energy function for protein structure prediction by decoy selections. J Comput. Chem., 37, 1119–1124. |
| [24] | Iqbal,S. et al. (2015) Improved prediction of accessible surface area results in efficient energy function application. J. Theor. Biol., 380, 380–391. |
| [25] | Mishra,A. et al. (2016) Discriminate protein decoys from native by using a scoring function based on ubiquitous Phi and Psi angles computed for all atom. J. Theor. Biol., 398, 112–121. |
| [26] | Babu,M.M. et al. (2011) Intrinsically disordered proteins: regulation and disease. Curr. Opin. Struct. Biol., 21, 432. |
| [27] | Doszta´nyi,Z. et al. (2005) The pairwise energy content estimated from amino acid composition discriminates between folded and intrinsically unstructured proteins. J. Mol. Biol., 347, 827–839. |
| [28] | Xie,R. et al. (2020) DeepVF: a deep learning-based hybrid framework for identifying virulence factors using the stacking strategy. Brief. Bioinf.,2020,bbaa125 |
| [29] | Fischer,T. and Krauss,C., (2018)“ Deep learning with long short-term memory networks for financial market predictions,” Eur. J. Operat. Res, pp. 270,654-669. |
| [30] | Devlin,J.et.al (2018)“Bert: pre-training of deep bidirectional transformers for language understanding.,” arXiv preprint ., p. arXiv:1810.04805. |
| [31] | Hu,Q.et.al.(2015) “ A stacking-based approach to identify translated upstream open reading frames. In Arabidopsis Thaliana, International Symposium on Bioinformatics Research and Applications,,” Bioinformatics Research and Applications, , p. pp. 138–149. |
| [32] | Mishra,A.and Hoque,M.T. (2017)“ Three-dimensional ideal gas reference state based energy function,” Curr.Bioinformatics, pp. 12, 171–180. |
| [33] | Arroyo-Ferna´ndez,I. et al. (2019) Unsupervised sentence representations as word information series: revisiting TF–IDF. Comput. Speech Language, 56,107–129 |
| [34] | Asgari,E. and Mofrad,M.R. (2015) Continuous distributed representation of biological sequences for deep proteomics and genomics. PLoS One, 10,e0141287. |

**VITA**

NISHITHA YENDAPALLY

I am a graduate student of the Electrical Engineering and Computer Science department at Texas A&M University-Kingsville (TAMUK) pursuing a master’s degree in Computer Science. Before joining a master’s program in Computer Science at TAMUK, I completed my bachelor’s degree in Computer Science from the Jawaharlal Nehru Technological University, Hyderabad, India.

I have a special interest in Machine learning and Deep learning technologies. My thesis work has prepared me to become an ethical data science engineer and a competitive computer scientist.

**APPENDIX**

The BTP-640 dataset file contains peptide sequences in FASTA file format.

Step 1: Reading the dataset from the FASTA file and converting it to text or .csv file.

#Importing the pandas, numpy

from Bio import SeqIO

import pandas as pd

import numpy as np

import os

#Getting the current working directory

os.getcwd()

# output: 'C:\\Users\\Yendapally Nishitha\\peptide prediction'

#reading the fasta file

filepath='/Users/Yendapally Nishitha/peptide prediction/Training\_Positive.fasta'

seq\_objects=SeqIO.parse(filepath,'fasta')

sequences=[]

for seq in seq\_objects:

sequences.append(seq)

#Displaying the length of the sequences in FASTA file

len(sequences)

first\_record=sequences[1]

#Displaying of id, name, description for the sequences

first\_record.id

first\_record.description

#output: 'Negative'

Negative 1

first\_sequence=first\_record.seq

#finding the length of the first sequence

len(first\_sequence)

#output: 3

#Exporting the fasta file to text file after removing the ">", id, name and description

for record in sequences:

seq\_id=record.id

seq\_name=record.name

sequence=record.seq

sequence=[]

for record in sequences:

sequence= record.seq

print(sequence,file=open("testing\_data\_negative.txt","a"))

Step 2: After the data is collected in text file by removing the special characters, id, name and description of the sequences we perform k-mers extracting technique by giving the K=2 followed by feature extraction, and applying Machine Learning algorithms on the extracted features.

#importing numpy and pandas for

import numpy as np

import pandas as pd

from sklearn import \*

from sklearn.preprocessing import StandardScaler

# Reading the peptide sequencing dataset

data = pd.read\_table('actual dataset.txt')

data.head()

# function to convert sequence strings into k-mer words, default size = 2 ( words)

def getKmers(sequence, size=2):

return [sequence[x:x+size] for x in range(len(sequence) - size + 1)]

# converting data into short overlapping k-mers

data['words'] = data.apply(lambda x: getKmers(x['sequence']), axis=1)

data = data.drop('sequence', axis=1)

#print(data,file= open("feature\_file.txt","a"))

#Displaying the sequence of words

data\_texts = list(data['words'])

for item in range(len(data\_texts)):

data\_texts[item] = ' '.join(data\_texts[item])

#print(data\_texts)

y= data.iloc[:, 0].values

y\_train=y[0:512]

y\_test=y[512:]

##Applying Feature Extraction Techniques TFIDF and Count Vectorizer

from sklearn.feature\_extraction.text import CountVectorizer

cv = CountVectorizer()

X= cv.fit\_transform(data\_texts).toarray()

#from sklearn.feature\_extraction.text import TfidfVectorizer

#cv=TfidfVectorizer()

#X=cv.fit\_transform(data\_texts).toarray()

print(data\_texts)

print(X\_train.toarray())

print(X\_train.shape)

#removing all rows before 512

#X\_test=X[512:]

#removing all row after 512

#X\_train=X[0:512]

# Feature selection using Chi2

#from sklearn.feature\_selection import SelectKBest

#from sklearn.feature\_selection import chi2

#X= SelectKBest(chi2, k=3).fit\_transform(X,y)

#Feature selection using f\_regression

from sklearn.feature\_selection import SelectKBest

from sklearn.feature\_selection import f\_regression

fs = SelectKBest(score\_func=f\_regression, k=5)

test= fs.fit\_transform(X, y)

X\_test=X[512:]

X\_train=X[0:512]

print(X\_test.shape)

#output: (128, 316)

print(X\_train.shape)

#output: (512, 316)

**##1. Stacking Based model**

from sklearn import svm

from sklearn import model\_selection

from sklearn.linear\_model import LinearRegression

from sklearn.ensemble import AdaBoostClassifier

from sklearn.model\_selection import cross\_val\_score

from sklearn.model\_selection import cross\_val\_predict

from sklearn.svm import LinearSVC

from sklearn.pipeline import make\_pipeline

from sklearn.ensemble import RandomForestClassifier

from sklearn.ensemble import StackingClassifier

from sklearn.ensemble import BaggingClassifier

from sklearn.linear\_model import LogisticRegression

from sklearn.neighbors import KNeighborsClassifier

from sklearn.preprocessing import StandardScaler

from sklearn.naive\_bayes import MultinomialNB

from sklearn.naive\_bayes import GaussianNB

from sklearn.naive\_bayes import BernoulliNB

from sklearn.ensemble import GradientBoostingClassifier

from sklearn.metrics import accuracy\_score, precision\_score, confusion\_matrix, recall\_score, f1\_score, auc, matthews\_corrcoef

from sklearn.svm import SVC

#import xgboost as xgb

from sklearn.cluster import KMeans

from sklearn.feature\_selection import SelectFromModel

estimators=[('gnb',GaussianNB(var\_smoothing=10e-30)),('adb',AdaBoostClassifier(n\_estimators=10)),('bc',GradientBoostingClassifier(random\_state=5))]

model=StackingClassifier(estimators=estimators, final\_estimator=RandomForestClassifier(max\_depth=3, random\_state=10,min\_samples\_split=2))

y\_pred= cross\_val\_predict(model,X\_train,y\_train,cv=10,n\_jobs=-1)

outputFile=open('ADA\_stacking\_Final\_Test\_Results\_cross validation.txt','a')

confusion= confusion\_matrix(y\_train, y\_pred)

TP= confusion[1,1]

TN= confusion[0,0]

FP= confusion[0,1]

FN= confusion[1,0]

#specificity

Spe\_cla=(TN/float(TN+FP))

Acc\_Bal= 0.5\*((TP/float(TP+FN))+(TN/float(TN+FP)))

MCC\_cla= matthews\_corrcoef(y\_train, y\_pred)

F1\_cla=f1\_score(y\_train, y\_pred)

PREC\_cla=precision\_score(y\_train, y\_pred)

REC\_cla= recall\_score(y\_train, y\_pred)

Accuracy\_cla= accuracy\_score(y\_train, y\_pred)

Results='TFIDF Cross validation Results: \n'

outputFile.write(str(Results)+'\n')

outputFile.write('TP=%f\n'%TP)

outputFile.write('FP= %f\n'%FP)

outputFile.write('TN= %f\n'%TN)

outputFile.write('FN= %f\n'%FN)

outputFile.write('Recall/sensivity=%.5f\n'%REC\_cla)

outputFile.write('specificity= %.5f\n'%Spe\_cla)

outputFile.write('Accuracy\_balanced= %.5f\n'%Acc\_Bal)

outputFile.write('overall Accuracy= %.5f\n'%Accuracy\_cla)

outputFile.write('precision=%.5f\n'%PREC\_cla)

outputFile.write('F1=%.5f\n'%F1\_cla)

outputFile.write('MCC= %.5f\n'%MCC\_cla)

outputFile.close()

**## Training the model**

model.fit(X\_train,y\_train)

**#output:** StackingClassifier(estimators=[('gnb', GaussianNB(var\_smoothing=1e-29)),

('adb', AdaBoostClassifier(n\_estimators=10)),

('bc', GradientBoostingClassifier(random\_state=5))],

final\_estimator=RandomForestClassifier(max\_depth=3, random\_state=10))

y\_new=model.predict(X\_test)

y\_new

outputFile= open('ADA\_stacking\_Final\_Test\_Results\_cross validation.txt','a')

confuison = confusion\_matrix(y\_test, y\_new)

TP1= confusion[1,1]

TN1= confusion[0,0]

FP1= confusion[0,1]

FN1= confusion[1,0]

#specificity

SPEC\_cla= (TN1/float(TN1+FP1))

#Balanced accuracy

Acc\_Balance= 0.5\*((TP1/float(TP1+FN1))+(TN1/float(TN1+FP1)))

#Compute MCC

MCC\_cla= matthews\_corrcoef(y\_test, y\_new)

F1\_cla= f1\_score(y\_test, y\_new)

PREC\_cla= precision\_score(y\_test, y\_new)

REC\_cla= recall\_score(y\_test, y\_new)

Accuracy\_cla= accuracy\_score(y\_test, y\_new)

Results= 'Independent test Results:\n'

outputFile.write(str(Results)+'\n')

outputFile.write('TP= %f\n'%TP1)

outputFile.write('TN= %f\n'%TN1)

outputFile.write('FP= %f\n'%FP1)

outputFile.write('FN= %f\n'%FN1)

outputFile.write('Recall/sensivity= %.5f\n '%REC\_cla)

outputFile.write('specificity=%.5f\n'%SPEC\_cla)

outputFile.write('accuracy\_balanced= %.5f\n'%Acc\_Balance)

outputFile.write('overall\_accuracy= %.5f\n'% Accuracy\_cla)

outputFile.write('precision= %.5f\n'%PREC\_cla)

outputFile.write('F1=%.5f\n' %F1\_cla)

outputFile.write('MCC= %.5f\n'%MCC\_cla)

outputFile.close()

**##2. SVM**

from sklearn import svm

from sklearn import model\_selection

from sklearn.model\_selection import cross\_val\_score

from sklearn.model\_selection import cross\_val\_predict

from sklearn.svm import LinearSVC

from sklearn.pipeline import make\_pipeline

from sklearn.metrics import accuracy\_score, precision\_score, confusion\_matrix, recall\_score, f1\_score, auc, matthews\_corrcoef

from sklearn.svm import SVC

from sklearn.cluster import KMeans

model=SVC(kernel='linear', gamma='auto')

y\_pred= cross\_val\_predict(model,X,y\_train,cv=10,n\_jobs=-1)

outputFile=open('SVM\_Final\_Test\_Results\_cross validation.txt','a')

confusion= confusion\_matrix(y\_train, y\_pred)

TP= confusion[1,1]

TN= confusion[0,0]

FP= confusion[0,1]

FN= confusion[1,0]

#specificity

Spe\_cla=(TN/float(TN+FP))

Acc\_Bal= 0.5\*((TP/float(TP+FN))+(TN/float(TN+FP)))

MCC\_cla= matthews\_corrcoef(y\_train, y\_pred)

F1\_cla=f1\_score(y\_train, y\_pred)

PREC\_cla=precision\_score(y\_train, y\_pred)

REC\_cla= recall\_score(y\_train, y\_pred)

Accuracy\_cla= accuracy\_score(y\_train, y\_pred)

Results='TFIDF Cross validation Results: \n'

outputFile.write(str(Results)+'\n')

outputFile.write('TP=%f\n'%TP)

outputFile.write('FP= %f\n'%FP)

outputFile.write('TN= %f\n'%TN)

outputFile.write('FN= %f\n'%FN)

outputFile.write('Recall/sensivity=%.5f\n'%REC\_cla)

outputFile.write('specificity= %.5f\n'%Spe\_cla)

outputFile.write('Accuracy\_balanced= %.5f\n'%Acc\_Bal)

outputFile.write('overall Accuracy= %.5f\n'%Accuracy\_cla)

outputFile.write('precision=%.5f\n'%PREC\_cla)

outputFile.write('F1=%.5f\n'%F1\_cla)

outputFile.write('MCC= %.5f\n'%MCC\_cla)

outputFile.close()

model.fit(X\_train,y\_train)

#Output: SVC(gamma='auto', kernel='linear')

y\_new=model.predict(X\_test)

y\_new

outputFile= open('SVM\_Final\_Test\_Results\_cross validation.txt','a')

confuison = confusion\_matrix(y\_test, y\_new)

TP1= confusion[1,1]

TN1= confusion[0,0]

FP1= confusion[0,1]

FN1= confusion[1,0]

#specificity

SPEC\_cla= (TN1/float(TN1+FP1))

#Balanced accuracy

Acc\_Balance= 0.5\*((TP1/float(TP1+FN1))+(TN1/float(TN1+FP1)))

#Compute MCC

MCC\_cla= matthews\_corrcoef(y\_test, y\_new)

F1\_cla= f1\_score(y\_test, y\_new)

PREC\_cla= precision\_score(y\_test, y\_new)

REC\_cla= recall\_score(y\_test, y\_new)

Accuracy\_cla= accuracy\_score(y\_test, y\_new)

Results= 'Independent test Results:\n'

outputFile.write(str(Results)+'\n')

outputFile.write('TP= %f\n'%TP1)

outputFile.write('TN= %f\n'%TN1)

outputFile.write('FP= %f\n'%FP1)

outputFile.write('FN= %f\n'%FN1)

outputFile.write('Recall/sensivity= %.5f\n '%REC\_cla)

outputFile.write('specificity=%.5f\n'%SPEC\_cla)

outputFile.write('accuracy\_balanced= %.5f\n'%Acc\_Balance)

outputFile.write('overall\_accuracy= %.5f\n'% Accuracy\_cla)

outputFile.write('precision= %.5f\n'%PREC\_cla)

outputFile.write('F1=%.5f\n' %F1\_cla)

outputFile.write('MCC= %.5f\n'%MCC\_cla)

outputFile.close()

**##3. Random Forest classifier**

from sklearn.model\_selection import cross\_val\_score

from sklearn.model\_selection import cross\_val\_predict

from sklearn.ensemble import RandomForestClassifier

from sklearn.pipeline import make\_pipeline

from sklearn.metrics import accuracy\_score, precision\_score, confusion\_matrix, recall\_score, f1\_score, auc, matthews\_corrcoef

model=RandomForestClassifier(max\_depth=2, random\_state=0)

y\_pred= cross\_val\_predict(model,X,y\_train,cv=10,n\_jobs=-1)

outputFile=open('RandomForestClassifier\_Final\_Test\_Results\_cross validation.txt','a')

confusion= confusion\_matrix(y\_train, y\_pred)

TP= confusion[1,1]

TN= confusion[0,0]

FP= confusion[0,1]

FN= confusion[1,0]

#specificity

Spe\_cla=(TN/float(TN+FP))

Acc\_Bal= 0.5\*((TP/float(TP+FN))+(TN/float(TN+FP)))

MCC\_cla= matthews\_corrcoef(y\_train, y\_pred)

F1\_cla=f1\_score(y\_train, y\_pred)

PREC\_cla=precision\_score(y\_train, y\_pred)

REC\_cla= recall\_score(y\_train, y\_pred)

Accuracy\_cla= accuracy\_score(y\_train, y\_pred)

Results='TFIDF Cross validation Results: \n'

outputFile.write(str(Results)+'\n')

outputFile.write('TP=%f\n'%TP)

outputFile.write('FP= %f\n'%FP)

outputFile.write('TN= %f\n'%TN)

outputFile.write('FN= %f\n'%FN)

outputFile.write('Recall/sensivity=%.5f\n'%REC\_cla)

outputFile.write('specificity= %.5f\n'%Spe\_cla)

outputFile.write('Accuracy\_balanced= %.5f\n'%Acc\_Bal)

outputFile.write('overall Accuracy= %.5f\n'%Accuracy\_cla)

outputFile.write('precision=%.5f\n'%PREC\_cla)

outputFile.write('F1=%.5f\n'%F1\_cla)

outputFile.write('MCC= %.5f\n'%MCC\_cla)

outputFile.close()

#training the model

model.fit(X\_train,y\_train)

#output: RandomForestClassifier(max\_depth=2, random\_state=0)

y\_new=model.predict(X\_test)

y\_new

outputFile= open('RandomForestClassifier\_Final\_Test\_Results\_cross validation.txt','a')

confuison = confusion\_matrix(y\_test, y\_new)

TP1= confusion[1,1]

TN1= confusion[0,0]

FP1= confusion[0,1]

FN1= confusion[1,0]

#specificity

SPEC\_cla= (TN1/float(TN1+FP1))

#Balanced accuracy

Acc\_Balance= 0.5\*((TP1/float(TP1+FN1))+(TN1/float(TN1+FP1)))

#Compute MCC

MCC\_cla= matthews\_corrcoef(y\_test, y\_new)

F1\_cla= f1\_score(y\_test, y\_new)

PREC\_cla= precision\_score(y\_test, y\_new)

REC\_cla= recall\_score(y\_test, y\_new)

Accuracy\_cla= accuracy\_score(y\_test, y\_new)

Results= 'Independent test Results:\n'

outputFile.write(str(Results)+'\n')

outputFile.write('TP= %f\n'%TP1)

outputFile.write('TN= %f\n'%TN1)

outputFile.write('FP= %f\n'%FP1)

outputFile.write('FN= %f\n'%FN1)

outputFile.write('Recall/sensivity= %.5f\n '%REC\_cla)

outputFile.write('specificity=%.5f\n'%SPEC\_cla)

outputFile.write('accuracy\_balanced= %.5f\n'%Acc\_Balance)

outputFile.write('overall\_accuracy= %.5f\n'% Accuracy\_cla)

outputFile.write('precision= %.5f\n'%PREC\_cla)

outputFile.write('F1=%.5f\n' %F1\_cla)

outputFile.write('MCC= %.5f\n'%MCC\_cla)

outputFile.close()

**##4. Naïve Bayes**

from sklearn.model\_selection import cross\_val\_score

from sklearn.model\_selection import cross\_val\_predict

from sklearn.pipeline import make\_pipeline

from sklearn.naive\_bayes import MultinomialNB

from sklearn.metrics import accuracy\_score, precision\_score, confusion\_matrix, recall\_score, f1\_score, auc, matthews\_corrcoef

model=MultinomialNB(alpha=0.1)

y\_pred= cross\_val\_predict(model,X\_train,y\_train,cv=10,n\_jobs=-1)

outputFile=open('Naive\_Bayes\_Final\_Test\_Results\_cross validation.txt','a')

confusion= confusion\_matrix(y\_train, y\_pred)

TP= confusion[1,1]

TN= confusion[0,0]

FP= confusion[0,1]

FN= confusion[1,0]

#specificity

Spe\_cla=(TN/float(TN+FP))

Acc\_Bal= 0.5\*((TP/float(TP+FN))+(TN/float(TN+FP)))

MCC\_cla= matthews\_corrcoef(y\_train, y\_pred)

F1\_cla=f1\_score(y\_train, y\_pred)

PREC\_cla=precision\_score(y\_train, y\_pred)

REC\_cla= recall\_score(y\_train, y\_pred)

Accuracy\_cla= accuracy\_score(y\_train, y\_pred)

Results='TFIDF Cross validation Results: \n'

outputFile.write(str(Results)+'\n')

outputFile.write('TP=%f\n'%TP)

outputFile.write('FP= %f\n'%FP)

outputFile.write('TN= %f\n'%TN)

outputFile.write('FN= %f\n'%FN)

outputFile.write('Recall/sensivity=%.5f\n'%REC\_cla)

outputFile.write('specificity= %.5f\n'%Spe\_cla)

outputFile.write('Accuracy\_balanced= %.5f\n'%Acc\_Bal)

outputFile.write('overall Accuracy= %.5f\n'%Accuracy\_cla)

outputFile.write('precision=%.5f\n'%PREC\_cla)

outputFile.write('F1=%.5f\n'%F1\_cla)

outputFile.write('MCC= %.5f\n'%MCC\_cla)

outputFile.close()

#training the model

model.fit(X\_train,y\_train)

#Output: MultinomialNB(alpha=0.1)

y\_new=model.predict(X\_test)

y\_new

outputFile= open('Naive\_Bayes\_Final\_Test\_Results\_cross validation.txt','a')

confuison = confusion\_matrix(y\_test, y\_new)

TP1= confusion[1,1]

TN1= confusion[0,0]

FP1= confusion[0,1]

FN1= confusion[1,0]

#specificity

SPEC\_cla= (TN1/float(TN1+FP1))

#Balanced accuracy

Acc\_Balance= 0.5\*((TP1/float(TP1+FN1))+(TN1/float(TN1+FP1)))

#Compute MCC

MCC\_cla= matthews\_corrcoef(y\_test, y\_new)

F1\_cla= f1\_score(y\_test, y\_new)

PREC\_cla= precision\_score(y\_test, y\_new)

REC\_cla= recall\_score(y\_test, y\_new)

Accuracy\_cla= accuracy\_score(y\_test, y\_new)

Results= 'Independent test Results:\n'

outputFile.write(str(Results)+'\n')

outputFile.write('TP= %f\n'%TP1)

outputFile.write('TN= %f\n'%TN1)

outputFile.write('FP= %f\n'%FP1)

outputFile.write('FN= %f\n'%FN1)

outputFile.write('Recall/sensivity= %.5f\n '%REC\_cla)

outputFile.write('specificity=%.5f\n'%SPEC\_cla)

outputFile.write('accuracy\_balanced= %.5f\n'%Acc\_Balance)

outputFile.write('overall\_accuracy= %.5f\n'% Accuracy\_cla)

outputFile.write('precision= %.5f\n'%PREC\_cla)

outputFile.write('F1=%.5f\n' %F1\_cla)

outputFile.write('MCC= %.5f\n'%MCC\_cla)

outputFile.close()

**##5. Logistic Regression**

from sklearn import model\_selection

from sklearn.model\_selection import cross\_val\_score

from sklearn.model\_selection import cross\_val\_predict

from sklearn.pipeline import make\_pipeline

from sklearn.linear\_model import LogisticRegression

from sklearn.preprocessing import StandardScaler

from sklearn.metrics import accuracy\_score, precision\_score, confusion\_matrix, recall\_score, f1\_score, auc, matthews\_corrcoef

model=tree.DecisionTreeClassifier()

y\_pred= cross\_val\_predict(model,X,y\_train,cv=10,n\_jobs=-1)

outputFile=open('Decision\_Tree\_Final\_Test\_Results\_cross validation.txt','a')

confusion= confusion\_matrix(y\_train, y\_pred)

TP= confusion[1,1]

TN= confusion[0,0]

FP= confusion[0,1]

FN= confusion[1,0]

#specificity

Spe\_cla=(TN/float(TN+FP))

Acc\_Bal= 0.5\*((TP/float(TP+FN))+(TN/float(TN+FP)))

MCC\_cla= matthews\_corrcoef(y\_train, y\_pred)

F1\_cla=f1\_score(y\_train, y\_pred)

PREC\_cla=precision\_score(y\_train, y\_pred)

REC\_cla= recall\_score(y\_train, y\_pred)

Accuracy\_cla= accuracy\_score(y\_train, y\_pred)

Results='TFIDF Cross validation Results: \n'

outputFile.write(str(Results)+'\n')

outputFile.write('TP=%f\n'%TP)

outputFile.write('FP= %f\n'%FP)

outputFile.write('TN= %f\n'%TN)

outputFile.write('FN= %f\n'%FN)

outputFile.write('Recall/sensivity=%.5f\n'%REC\_cla)

outputFile.write('specificity= %.5f\n'%Spe\_cla)

outputFile.write('Accuracy\_balanced= %.5f\n'%Acc\_Bal)

outputFile.write('overall Accuracy= %.5f\n'%Accuracy\_cla)

outputFile.write('precision=%.5f\n'%PREC\_cla)

outputFile.write('F1=%.5f\n'%F1\_cla)

outputFile.write('MCC= %.5f\n'%MCC\_cla)

outputFile.close()

model.fit(X\_train,y\_train)

y\_new=model.predict(X\_test)

y\_new

outputFile= open('Decision\_Tree\_Final\_Test\_Results\_cross validation.txt','a')

confuison = confusion\_matrix(y\_test, y\_new)

TP1= confusion[1,1]

TN1= confusion[0,0]

FP1= confusion[0,1]

FN1= confusion[1,0]

#specificity

SPEC\_cla= (TN1/float(TN1+FP1))

#Balanced accuracy

Acc\_Balance= 0.5\*((TP1/float(TP1+FN1))+(TN1/float(TN1+FP1)))

#Compute MCC

MCC\_cla= matthews\_corrcoef(y\_test, y\_new)

F1\_cla= f1\_score(y\_test, y\_new)

PREC\_cla= precision\_score(y\_test, y\_new)

REC\_cla= recall\_score(y\_test, y\_new)

Accuracy\_cla= accuracy\_score(y\_test, y\_new)

Results= 'Independent test Results:\n'

outputFile.write(str(Results)+'\n')

outputFile.write('TP= %f\n'%TP1)

outputFile.write('TN= %f\n'%TN1)

outputFile.write('FP= %f\n'%FP1)

outputFile.write('FN= %f\n'%FN1)

outputFile.write('Recall/sensivity= %.5f\n '%REC\_cla)

outputFile.write('specificity=%.5f\n'%SPEC\_cla)

outputFile.write('accuracy\_balanced= %.5f\n'%Acc\_Balance)

outputFile.write('overall\_accuracy= %.5f\n'% Accuracy\_cla)

outputFile.write('precision= %.5f\n'%PREC\_cla)

outputFile.write('F1=%.5f\n' %F1\_cla)

outputFile.write('MCC= %.5f\n'%MCC\_cla)

outputFile.close()

**##6. K-Nearest Neighbor**

from sklearn import model\_selection

from sklearn.model\_selection import cross\_val\_score

from sklearn.model\_selection import cross\_val\_predict

from sklearn.pipeline import make\_pipeline

from sklearn.neighbors import KNeighborsClassifier

from sklearn.metrics import accuracy\_score, precision\_score, confusion\_matrix, recall\_score, f1\_score, auc, matthews\_corrcoef

model=KNeighborsClassifier(n\_neighbors=3)

y\_pred= cross\_val\_predict(model,X,y\_train,cv=10,n\_jobs=-1)

outputFile=open('K-Nearest\_Neighbor\_Final\_Test\_Results\_cross validation.txt','a')

confusion= confusion\_matrix(y\_train, y\_pred)

TP= confusion[1,1]

TN= confusion[0,0]

FP= confusion[0,1]

FN= confusion[1,0]

#specificity

Spe\_cla=(TN/float(TN+FP))

Acc\_Bal= 0.5\*((TP/float(TP+FN))+(TN/float(TN+FP)))

MCC\_cla= matthews\_corrcoef(y\_train, y\_pred)

F1\_cla=f1\_score(y\_train, y\_pred)

PREC\_cla=precision\_score(y\_train, y\_pred)

REC\_cla= recall\_score(y\_train, y\_pred)

Accuracy\_cla= accuracy\_score(y\_train, y\_pred)

Results='TFIDF Cross validation Results: \n'

outputFile.write(str(Results)+'\n')

outputFile.write('TP=%f\n'%TP)

outputFile.write('FP= %f\n'%FP)

outputFile.write('TN= %f\n'%TN)

outputFile.write('FN= %f\n'%FN)

outputFile.write('Recall/sensivity=%.5f\n'%REC\_cla)

outputFile.write('specificity= %.5f\n'%Spe\_cla)

outputFile.write('Accuracy\_balanced= %.5f\n'%Acc\_Bal)

outputFile.write('overall Accuracy= %.5f\n'%Accuracy\_cla)

outputFile.write('precision=%.5f\n'%PREC\_cla)

outputFile.write('F1=%.5f\n'%F1\_cla)

outputFile.write('MCC= %.5f\n'%MCC\_cla)

outputFile.close()

model.fit(X\_train,y\_train)

#output: KNeighborsClassifier(n\_neighbors=3)

y\_new=model.predict(X\_test)

y\_new

outputFile= open('K-Nearest\_Neighbor\_Final\_Test\_Results\_cross validation.txt','a')

confuison = confusion\_matrix(y\_test, y\_new)

TP1= confusion[1,1]

TN1= confusion[0,0]

FP1= confusion[0,1]

FN1= confusion[1,0]

#specificity

SPEC\_cla= (TN1/float(TN1+FP1))

#Balanced accuracy

Acc\_Balance= 0.5\*((TP1/float(TP1+FN1))+(TN1/float(TN1+FP1)))

#Compute MCC

MCC\_cla= matthews\_corrcoef(y\_test, y\_new)

F1\_cla= f1\_score(y\_test, y\_new)

PREC\_cla= precision\_score(y\_test, y\_new)

REC\_cla= recall\_score(y\_test, y\_new)

Accuracy\_cla= accuracy\_score(y\_test, y\_new)

Results= 'Independent test Results:\n'

outputFile.write(str(Results)+'\n')

outputFile.write('TP= %f\n'%TP1)

outputFile.write('TN= %f\n'%TN1)

outputFile.write('FP= %f\n'%FP1)

outputFile.write('FN= %f\n'%FN1)

outputFile.write('Recall/sensivity= %.5f\n '%REC\_cla)

outputFile.write('specificity=%.5f\n'%SPEC\_cla)

outputFile.write('accuracy\_balanced= %.5f\n'%Acc\_Balance)

outputFile.write('overall\_accuracy= %.5f\n'% Accuracy\_cla)

outputFile.write('precision= %.5f\n'%PREC\_cla)

outputFile.write('F1=%.5f\n' %F1\_cla)

outputFile.write('MCC= %.5f\n'%MCC\_cla)

outputFile.close()

**##7. Gradient-Boosting**

from sklearn.model\_selection import cross\_val\_score

from sklearn.model\_selection import cross\_val\_predict

from sklearn.ensemble import GradientBoostingClassifier

from sklearn.metrics import accuracy\_score, precision\_score, confusion\_matrix, recall\_score, f1\_score, auc, matthews\_corrcoef

model=GradientBoostingClassifier()

y\_pred= cross\_val\_predict(model,X,y\_train,cv=10,n\_jobs=-1)

outputFile=open('Gradient\_Boosting\_Final\_Test\_Results\_cross validation.txt','a')

confusion= confusion\_matrix(y\_train, y\_pred)

TP= confusion[1,1]

TN= confusion[0,0]

FP= confusion[0,1]

FN= confusion[1,0]

#specificity

Spe\_cla=(TN/float(TN+FP))

Acc\_Bal= 0.5\*((TP/float(TP+FN))+(TN/float(TN+FP)))

MCC\_cla= matthews\_corrcoef(y\_train, y\_pred)

F1\_cla=f1\_score(y\_train, y\_pred)

PREC\_cla=precision\_score(y\_train, y\_pred)

REC\_cla= recall\_score(y\_train, y\_pred)

Accuracy\_cla= accuracy\_score(y\_train, y\_pred)

Results='TFIDF Cross validation Results: \n'

outputFile.write(str(Results)+'\n')

outputFile.write('TP=%f\n'%TP)

outputFile.write('FP= %f\n'%FP)

outputFile.write('TN= %f\n'%TN)

outputFile.write('FN= %f\n'%FN)

outputFile.write('Recall/sensivity=%.5f\n'%REC\_cla)

outputFile.write('specificity= %.5f\n'%Spe\_cla)

outputFile.write('Accuracy\_balanced= %.5f\n'%Acc\_Bal)

outputFile.write('overall Accuracy= %.5f\n'%Accuracy\_cla)

outputFile.write('precision=%.5f\n'%PREC\_cla)

outputFile.write('F1=%.5f\n'%F1\_cla)

outputFile.write('MCC= %.5f\n'%MCC\_cla)

outputFile.close()

model.fit(X\_train,y\_train)

#output: GradientBoostingClassifier()

y\_new=model.predict(X\_test)

y\_new

outputFile= open('Gradient\_Boosting\_Final\_Test\_Results\_cross validation.txt','a')

confuison = confusion\_matrix(y\_test, y\_new)

TP1= confusion[1,1]

TN1= confusion[0,0]

FP1= confusion[0,1]

FN1= confusion[1,0]

#specificity

SPEC\_cla= (TN1/float(TN1+FP1))

#Balanced accuracy

Acc\_Balance= 0.5\*((TP1/float(TP1+FN1))+(TN1/float(TN1+FP1)))

#Compute MCC

MCC\_cla= matthews\_corrcoef(y\_test, y\_new)

F1\_cla= f1\_score(y\_test, y\_new)

PREC\_cla= precision\_score(y\_test, y\_new)

REC\_cla= recall\_score(y\_test, y\_new)

Accuracy\_cla= accuracy\_score(y\_test, y\_new)

Results= 'Independent test Results:\n'

outputFile.write(str(Results)+'\n')

outputFile.write('TP= %f\n'%TP1)

outputFile.write('TN= %f\n'%TN1)

outputFile.write('FP= %f\n'%FP1)

outputFile.write('FN= %f\n'%FN1)

outputFile.write('Recall/sensivity= %.5f\n '%REC\_cla)

outputFile.write('specificity=%.5f\n'%SPEC\_cla)

outputFile.write('accuracy\_balanced= %.5f\n'%Acc\_Balance)

outputFile.write('overall\_accuracy= %.5f\n'% Accuracy\_cla)

outputFile.write('precision= %.5f\n'%PREC\_cla)

outputFile.write('F1=%.5f\n' %F1\_cla)

outputFile.write('MCC= %.5f\n'%MCC\_cla)

outputFile.close()

**##8. ADA-Boosting**

from sklearn.model\_selection import cross\_val\_score

from sklearn.model\_selection import cross\_val\_predict

from sklearn.ensemble import AdaBoostClassifier

from sklearn.metrics import accuracy\_score, precision\_score, confusion\_matrix, recall\_score, f1\_score, auc, matthews\_corrcoef

from sklearn.svm import SVC

from sklearn.cluster import KMeans

model = AdaBoostClassifier(n\_estimators=100, random\_state=0)

y\_pred= cross\_val\_predict(model,X,y\_train,cv=10,n\_jobs=-1)

outputFile=open('AdaBoostClassifier\_Final\_Test\_Results\_cross validation.txt','a')

confusion= confusion\_matrix(y\_train, y\_pred)

TP= confusion[1,1]

TN= confusion[0,0]

FP= confusion[0,1]

FN= confusion[1,0]

#specificity

Spe\_cla=(TN/float(TN+FP))

Acc\_Bal= 0.5\*((TP/float(TP+FN))+(TN/float(TN+FP)))

MCC\_cla= matthews\_corrcoef(y\_train, y\_pred)

F1\_cla=f1\_score(y\_train, y\_pred)

PREC\_cla=precision\_score(y\_train, y\_pred)

REC\_cla= recall\_score(y\_train, y\_pred)

Accuracy\_cla= accuracy\_score(y\_train, y\_pred)

Results='TFIDF Cross validation Results: \n'

outputFile.write(str(Results)+'\n')

outputFile.write('TP=%f\n'%TP)

outputFile.write('FP= %f\n'%FP)

outputFile.write('TN= %f\n'%TN)

outputFile.write('FN= %f\n'%FN)

outputFile.write('Recall/sensivity=%.5f\n'%REC\_cla)

outputFile.write('specificity= %.5f\n'%Spe\_cla)

outputFile.write('Accuracy\_balanced= %.5f\n'%Acc\_Bal)

outputFile.write('overall Accuracy= %.5f\n'%Accuracy\_cla)

outputFile.write('precision=%.5f\n'%PREC\_cla)

outputFile.write('F1=%.5f\n'%F1\_cla)

outputFile.write('MCC= %.5f\n'%MCC\_cla)

outputFile.close()

model.fit(X\_train,y\_train)

y\_new=model.predict(X\_test)

y\_new

outputFile= open('AdaBoostClassifier\_Final\_Test\_Results\_cross validation.txt','a')

confuison = confusion\_matrix(y\_test, y\_new)

TP1= confusion[1,1]

TN1= confusion[0,0]

FP1= confusion[0,1]

FN1= confusion[1,0]

#specificity

SPEC\_cla= (TN1/float(TN1+FP1))

#Balanced accuracy

Acc\_Balance= 0.5\*((TP1/float(TP1+FN1))+(TN1/float(TN1+FP1)))

#Compute MCC

MCC\_cla= matthews\_corrcoef(y\_test, y\_new)

F1\_cla= f1\_score(y\_test, y\_new)

PREC\_cla= precision\_score(y\_test, y\_new)

REC\_cla= recall\_score(y\_test, y\_new)

Accuracy\_cla= accuracy\_score(y\_test, y\_new)

Results= 'Independent test Results:\n'

outputFile.write(str(Results)+'\n')

outputFile.write('TP= %f\n'%TP1)

outputFile.write('TN= %f\n'%TN1)

outputFile.write('FP= %f\n'%FP1)

outputFile.write('FN= %f\n'%FN1)

outputFile.write('Recall/sensivity= %.5f\n '%REC\_cla)

outputFile.write('specificity=%.5f\n'%SPEC\_cla)

outputFile.write('accuracy\_balanced= %.5f\n'%Acc\_Balance)

outputFile.write('overall\_accuracy= %.5f\n'% Accuracy\_cla)

outputFile.write('precision= %.5f\n'%PREC\_cla)

outputFile.write('F1=%.5f\n' %F1\_cla)

outputFile.write('MCC= %.5f\n'%MCC\_cla)

outputFile.close()

**##9. Decision-Tree**

from sklearn.model\_selection import cross\_val\_score

from sklearn.model\_selection import cross\_val\_predict

from sklearn import tree

from sklearn.pipeline import make\_pipeline

from sklearn.ensemble import GradientBoostingClassifier

from sklearn.metrics import accuracy\_score, precision\_score, confusion\_matrix, recall\_score, f1\_score, auc, matthews\_corrcoef

model=tree.DecisionTreeClassifier()

y\_pred= cross\_val\_predict(model,X,y\_train,cv=10,n\_jobs=-1)

outputFile=open('Decision\_Tree\_Final\_Test\_Results\_cross validation.txt','a')

confusion= confusion\_matrix(y\_train, y\_pred)

TP= confusion[1,1]

TN= confusion[0,0]

FP= confusion[0,1]

FN= confusion[1,0]

#specificity

Spe\_cla=(TN/float(TN+FP))

Acc\_Bal= 0.5\*((TP/float(TP+FN))+(TN/float(TN+FP)))

MCC\_cla= matthews\_corrcoef(y\_train, y\_pred)

F1\_cla=f1\_score(y\_train, y\_pred)

PREC\_cla=precision\_score(y\_train, y\_pred)

REC\_cla= recall\_score(y\_train, y\_pred)

Accuracy\_cla= accuracy\_score(y\_train, y\_pred)

Results='TFIDF Cross validation Results: \n'

outputFile.write(str(Results)+'\n')

outputFile.write('TP=%f\n'%TP)

outputFile.write('FP= %f\n'%FP)

outputFile.write('TN= %f\n'%TN)

outputFile.write('FN= %f\n'%FN)

outputFile.write('Recall/sensivity=%.5f\n'%REC\_cla)

outputFile.write('specificity= %.5f\n'%Spe\_cla)

outputFile.write('Accuracy\_balanced= %.5f\n'%Acc\_Bal)

outputFile.write('overall Accuracy= %.5f\n'%Accuracy\_cla)

outputFile.write('precision=%.5f\n'%PREC\_cla)

outputFile.write('F1=%.5f\n'%F1\_cla)

outputFile.write('MCC= %.5f\n'%MCC\_cla)

outputFile.close()

model.fit(X\_train,y\_train)

#Output: DecisionTreeClassifier()

y\_new=model.predict(X\_test)

y\_new

outputFile= open('Decision\_Tree\_Final\_Test\_Results\_cross validation.txt','a')

confuison = confusion\_matrix(y\_test, y\_new)

TP1= confusion[1,1]

TN1= confusion[0,0]

FP1= confusion[0,1]

FN1= confusion[1,0]

#specificity

SPEC\_cla= (TN1/float(TN1+FP1))

#Balanced accuracy

Acc\_Balance= 0.5\*((TP1/float(TP1+FN1))+(TN1/float(TN1+FP1)))

#Compute MCC

MCC\_cla= matthews\_corrcoef(y\_test, y\_new)

F1\_cla= f1\_score(y\_test, y\_new)

PREC\_cla= precision\_score(y\_test, y\_new)

REC\_cla= recall\_score(y\_test, y\_new)

Accuracy\_cla= accuracy\_score(y\_test, y\_new)

Results= 'Independent test Results:\n'

outputFile.write(str(Results)+'\n')

outputFile.write('TP= %f\n'%TP1)

outputFile.write('TN= %f\n'%TN1)

outputFile.write('FP= %f\n'%FP1)

outputFile.write('FN= %f\n'%FN1)

outputFile.write('Recall/sensivity= %.5f\n '%REC\_cla)

outputFile.write('specificity=%.5f\n'%SPEC\_cla)

outputFile.write('accuracy\_balanced= %.5f\n'%Acc\_Balance)

outputFile.write('overall\_accuracy= %.5f\n'% Accuracy\_cla)

outputFile.write('precision= %.5f\n'%PREC\_cla)

outputFile.write('F1=%.5f\n' %F1\_cla)

outputFile.write('MCC= %.5f\n'%MCC\_cla)

outputFile.close()