**Twitter Sentiment Analysis, Research Project**

**Task:** Classify the sentiment or opinion expressed in tweets into one of the three classes: positive (1), negative (-1), or neutral (0) (which means no opinion).

**Training data:** the training data has 3 classes: positive (1), negative (-1), neutral (0)

**About the dataset**: We use a dataset from Kaggle – “Twitter Sentiment Analysis”. The dataset consists of tweets extracted from twitter and is pre-divided into train and test sets. This is a three class-classification problem and the dependent variable is either “Positive”, “Negative” or “Neutral”. The dataset has about 27k+ rows. The dataset looks almost balanced, and the distribution is as follows.

**Team Members Contributions**

|  |  |
| --- | --- |
| Muhammad Abrar Tariq | EDA, ML Model 1 |
| Ajay Sagar | Evaluation, Logistic Regression |
| Chinmay Tarwate | PPT, Decision Tree |
| Varun Bhalla | Cleaning, Baseline, SVM |

Table

Description automatically generatedChart, bar chart

Description automatically generated

**Introduction**

**Objective:**

The problem statement is to "predict" the sentiment (positive, negative or neutral) from the content of the tweets (“tweet” column). The ground truth (i.e., true class labels) is determined from the “sentiment” column. This is a three-class classification problem.

The problem proceeds in three stages:

1. **Text Cleaning & Processing**: We will clean up the raw tweet text using the various functions like removing punctuations, lemmatizing etc. and convert them into tokenized data
2. **Exploratory Analysis & Feature construction**: We create bag-of-words feature vectors and training labels from the processed text of tweets and the class columns respectively.
3. **Classification & Evaluation**: We use a dictionary-based method to create a baseline model. After the features are created, we use them to learn 4 machine learning models which can classify them based on their sentiment and evaluate the models against the baseline.

**Techniques Tried**

1. **Text Cleaning & Processing**

We create a function which processes raw text and convert it into tokens. All tokens must hold the following conditions.

1. The tokens must all be in lower case.
2. The tokens should appear in the same order as in the raw text.
3. The tokens must be in their lemmatized form.
   1. If a token cannot be lemmatized, the token is ignored.
4. All punctuations are to be removed. (used string.punctuation)
   1. Apostrophe of the form ‘s is ignored
      1. E.g. She’s will become she
   2. Other apostrophes should be omitted
      1. E.g. Don’t becomes don’t.
5. Words are separated when hyphen is encountered
6. All URLs are removed

Since stemming removes only the last few characters of a word, it leads to incorrect spellings and inconsistent meanings. Lemmatization considers the context as well and converts the words into their base form. We identify the Part of Speech (POS) tag for the words in all the tweets for the lemmatization to work. E.g. stemming of ‘Caring’ would give ‘Car’, However lemmatizing would give ‘Care’. However, Lemmatization is computationally expensive since it involves converting all the words to their base form depending on the content. Since our dataset was not that huge, we have decided to go ahead with lemmatizing as it would give us a higher accuracy.

Text

Description automatically generatedText

Description automatically generated

1. **Exploratory Analysis & Feature Construction**
2. After we have our tweets tokenized, we study and explore the dataset for useful patterns
3. From the tokenized data we use the TF-IDF feature vector methodology to create feature vectors.
4. To reduce the large corpus of words (and since we do not require all of them for classification) we prune the frequency distribution of the words (by and head and the tail)
   1. Very common words and very rare words hardly give any information regarding the similarity of two tweets
5. We create a sparse matrix for each of the tweets.
   1. Tweets which have words occurring in only that particular tweet are removed.
6. We also used grid search to find out the best way to create feature vectors.
   1. Since we study tokens as individual units and their relationships to sentiments, we also decided to study and examine which words tend to follow others immediately or that co-occur within the same documents.
   2. We analyzed n-grams where n=1,2,3,4 and 5 against the accuracy metric and found the best results when n=1 and n=2.
   3. After careful analysis we chose to go ahead with using only unigrams and bigrams since n>=3 gave little to no improvement on accuracy.

**We use word clouds to see the top words in positive and negative classes.**

**Text

Description automatically generated**Text

Description automatically generated

**Class 1: Positive Words**

**Class -1: Negative Words**

1. **Classification & Evaluation**
   1. We create a baseline classifier which uses dictionary-based methods to classify the tweets
      1. Dictionary based methods are unsupervised techniques which predicted the sentiment of the text using knowledgebases, ontologies, databases, and lexicons specially curated and prepared just for sentiment analysis. They work well when the text is subjective and has various emotions like “sad”, “bad”, “good”, “happy” etc
      2. Most of these lexicons have a list of positive and negative polar words and some score associated with them. Scores are assigned to the text for which we want to compute the sentiment using this method.
      3. After aggregating these scores, we get the final sentiment. (We use AFINN lexicon dictionary for our classifier)
   2. After we create our features, we learn different machine learning models and compare it against our baseline classifier.

**ML Model 1: Support Vector Machines**

* + - 1. SVMs use kernel functions to determine the similarity/distance between two data points.
      2. We train our SVM on the dataset and perform 4-fold cross validation along with grid search to determine the best set of kernel and parameters.
      3. We evaluate the four different kernel functions linear – linear, poly, sigmoid and rbf and select the best one.

**ML Model 2: Logistic Regression**

1. We analyze the result of logistic regression with different penalties such as l1, l2 and choose the one that provides the maximum accuracy.
2. We apply grid search over the set of hyperparameters that need to be optimized and choose the best set of hyperparameters. The parameters analyzed are:
   * + - 1. Penalty: L1, L2.
         2. Since we have 3 classes to classify each tweet into, we analyze multiclass methods such as one vs rest and multinomial loss fit on the data.
         3. For the optimization of the linear regression, we analyze different algorithms such as "newton-cg", "sag", "lbfgs" and "liblinear"
3. Once we have the best parameters obtained we train the training dataset and report the results achieved on the test set.

**ML Model 3: Decision Tree**

1. Decision Tree use decision rules to predict the value of test data.
2. We train our model on the dataset and perform grid search to determine best set of parameters.
3. We evaluate different types of criterion (gini or entropy), splitter (best or random) and vary max depth. We train the dataset using the best set of parameters and report the results achieved on the test data.

**ML Model 4:**

**Results**

**Dataset**

1. **Baseline Classifier (Dictionary Based Method)**

|  |  |  |  |
| --- | --- | --- | --- |
| **Accuracy** | 0.637 | | |
|  | Precision | Recall | F-1 |
| **Positive Class (+1)** | 0.576 | 0.847 | 0.686 |
| **Negative Class (-1)** | 0.705 | 0.582 | 0.638 |
| **Neutral Class (0)** | 0.676 | 0.514 | 0.584 |

1. **Support Vector Machines Classifier**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Test Size = .33 | Training | Linear SVM Classifier | | | RBF SVM Classifier with C=1 | | |
| Accuracy | 0.64 | | | 0.68 | | |
|  | Precision | Recall | F-1 | Precision | Recall | F-1 |
| Positive Class (+1) | 0.66 | 0.73 | 0.69 | 0.69 | 0.77 | 0.67 |
| Negative Class (-1) | 0.61 | 0.59 | 0.60 | 0.59 | 0.69 | 0.64 |
| Neutral Class (0) | 0.64 | 0.61 | 0.62 | 0.73 | 0.62 | 0.67 |

**\*RBF was selected after 4-fold cross validation (Please see appendix for all the details)**

1. **Logistic Regression**

**The best parameters selected are : C = 1 , solver= liblinear , multiclass = ovr (refer appendix for all the results)**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Test Size = .33 | Training | Logistic Regression | | |
| Accuracy | 0.70 | | |
|  | Precision | Recall | F-1 |
| Positive Class (+1) | 0.60 | 0.72 | 0.66 |
| Negative Class (-1) | 0.73 | 0.64 | 0.74 |
| Neutral Class (0) | 0.74 | 0.76 | 0.69 |

1. **Decision Tree**

**The best parameters selected are: Criterion = gini, max depth = 300, splitter = random (refer appendix for all the results)**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Test Size = .33 | Training | Decision Tree | | |
| Accuracy | 0.67 | | |
|  | Precision | Recall | F-1 |
| Positive Class (+1) | 0.71 | 0.72 | 0.72 |
| Negative Class (-1) | 0.57 | 0.68 | 0.62 |
| Neutral Class (0) | 0.71 | 0.63 | 0.67 |

**Conclusions & Lessons Learnt**

We have achieved a significant improvement over the baseline method (unsupervised – dictionary-based method) with for the following.

1. **Dataset Support Vectors Machines C=1 works the best (\_\_\_\_ Accuracy on Test – 4-fold cross validation).**
2. **Logistic Regression with parameters C=1, multiclass = one-vs-rest(ovr) and liblinear optimizer provides the best results with a test accuracy of 0.70**
3. **Decision Tree with max depth of 300 and criterion = gini provides the best results with an accuracy of 0.67**

Since the naïve-method (dictionary-based method) does not take into account how words are combined in a sequence, we used an automatic approach where the sentiment analysis problem was modeled as a 3-class classification problem. For this process we extracted features from the text and used different machine learning models (like Support Vector Machines) as our statistical model. We tried various techniques like stemming, lemmatizing, tokenization and part-of-speech tagging.

We also found out that stemming did not give us great results however lemmatization offered better precision than stemming, but at the expense of recall. We found that unigrams and bigrams were sufficient and increasing the n-grams did not help with the accuracy.

Lessons Learnt: Feature Extraction looks like the most important part in text mining since we have to deal with subjectivity, tone, context, polarity and sarcasm. We should study how deep learning and other newer algorithms work on text data which can automatically extract features.

**Appendix**

1. **SVM Results**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **kernel** | **C** | **accuracy\_mean** | **accuracy\_min** | **accuracy\_max** |
| linear | 0.1 | 0.487656118 | 0.463057791 | 0.508412582 |
| linear | 1 | 0.593898258 | 0.564327485 | 0.61594733 |
| linear | 10 | 0.554768012 | 0.528508772 | 0.566934894 |
| linear | 100 | 0.543978228 | 0.527046784 | 0.55596196 |
| linear | 1000 | 0.537031779 | 0.510233918 | 0.555230432 |
| linear | 10000 | 0.529535212 | 0.501461988 | 0.540599854 |
| poly | 0.1 | 0.360942132 | 0.349670812 | 0.37088515 |
| poly | 1 | 0.532091018 | 0.519385516 | 0.550109729 |
| poly | 10 | 0.53995482 | 0.527046784 | 0.560351134 |
| poly | 100 | 0.507588409 | 0.501828822 | 0.5164594 |
| poly | 1000 | 0.432071286 | 0.422823702 | 0.446232626 |
| poly | 10000 | 0.422929848 | 0.40380395 | 0.445501097 |
| rbf | 0.1 | 0.368073336 | 0.356254572 | 0.383321141 |
| **rbf** | **1** | **0.60102759** | **0.581140351** | **0.633504023** |
| rbf | 10 | 0.59152012 | 0.567982456 | 0.60936357 |
| rbf | 100 | 0.590240346 | 0.565789474 | 0.610095099 |
| rbf | 1000 | 0.590240346 | 0.565789474 | 0.610095099 |
| rbf | 10000 | 0.590240346 | 0.565789474 | 0.610095099 |
| sigmoid | 0.1 | 0.488570396 | 0.457937089 | 0.518653987 |
| sigmoid | 1 | 0.59444637 | 0.567251462 | 0.614484272 |
| sigmoid | 10 | 0.537760901 | 0.523391813 | 0.548646672 |
| sigmoid | 100 | 0.498449645 | 0.477339181 | 0.512801756 |
| sigmoid | 1000 | 0.494789728 | 0.4835406 | 0.504023409 |
| sigmoid | 10000 | 0.498628651 | 0.476225311 | 0.510607169 |
|  |  |  |  |  |

**\*average, min and max accuracies are calculated using 4-fold cross validation. Kernel RBF with C=1 is selected.**

1. **Logistic regression results**:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **optimizer** | **multiclass** | **penalty** | **c** | **accuracy\_mean** | **accuracy\_min** | **accuracy\_max** |
| lbfgs | auto | l2 | 0.01 | 0.654757967 | 0.654757967 | 0.654757967 |
| lbfgs | auto | l2 | 0.1 | 0.677803506 | 0.677803506 | 0.677803506 |
| lbfgs | auto | l2 | 1 | 0.68883008 | 0.68883008 | 0.68883008 |
| lbfgs | auto | l2 | 10 | 0.658065939 | 0.658065939 | 0.658065939 |
| lbfgs | auto | l2 | 100 | 0.609438747 | 0.609438747 | 0.609438747 |
| lbfgs | auto | l2 | 1000 | 0.578123277 | 0.578123277 | 0.578123277 |
| lbfgs | auto | l2 | 10000 | 0.566876172 | 0.566876172 | 0.566876172 |
| lbfgs | multinomial | l2 | 0.01 | 0.654757967 | 0.654757967 | 0.654757967 |
| lbfgs | multinomial | l2 | 0.1 | 0.677803506 | 0.677803506 | 0.677803506 |
| lbfgs | multinomial | l2 | 1 | 0.68883008 | 0.68883008 | 0.68883008 |
| lbfgs | multinomial | l2 | 10 | 0.658065939 | 0.658065939 | 0.658065939 |
| lbfgs | multinomial | l2 | 100 | 0.609438747 | 0.609438747 | 0.609438747 |
| lbfgs | multinomial | l2 | 1000 | 0.578123277 | 0.578123277 | 0.578123277 |
| lbfgs | multinomial | l2 | 10000 | 0.566876172 | 0.566876172 | 0.566876172 |
| lbfgs | ovr | l2 | 0.01 | 0.643180064 | 0.643180064 | 0.643180064 |
| lbfgs | ovr | l2 | 0.1 | 0.668320653 | 0.668320653 | 0.668320653 |
| lbfgs | ovr | l2 | 1 | 0.685301577 | 0.685301577 | 0.685301577 |
| lbfgs | ovr | l2 | 10 | 0.66710773 | 0.66710773 | 0.66710773 |
| lbfgs | ovr | l2 | 100 | 0.61947293 | 0.61947293 | 0.61947293 |
| lbfgs | ovr | l2 | 1000 | 0.582754438 | 0.582754438 | 0.582754438 |
| lbfgs | ovr | l2 | 10000 | 0.563237402 | 0.563237402 | 0.563237402 |
| liblinear | auto | l1 | 0.01 | 0.404234204 | 0.404234204 | 0.404234204 |
| liblinear | auto | l1 | 0.1 | 0.631050833 | 0.631050833 | 0.631050833 |
| liblinear | auto | l1 | 1 | 0.704267284 | 0.704267284 | 0.704267284 |
| liblinear | auto | l1 | 10 | 0.661484177 | 0.661484177 | 0.661484177 |
| liblinear | auto | l1 | 100 | 0.599184034 | 0.599184034 | 0.599184034 |
| liblinear | auto | l1 | 1000 | 0.572279193 | 0.572279193 | 0.572279193 |
| liblinear | auto | l1 | 10000 | 0.563457934 | 0.563457934 | 0.563457934 |
| liblinear | auto | l2 | 0.01 | 0.535450436 | 0.535450436 | 0.535450436 |
| liblinear | auto | l2 | 0.1 | 0.656522219 | 0.656522219 | 0.656522219 |
| liblinear | auto | l2 | 1 | 0.686404234 | 0.686404234 | 0.686404234 |
| liblinear | auto | l2 | 10 | 0.667328261 | 0.667328261 | 0.667328261 |
| liblinear | auto | l2 | 100 | 0.619362664 | 0.619362664 | 0.619362664 |
| liblinear | auto | l2 | 1000 | 0.582313375 | 0.582313375 | 0.582313375 |
| liblinear | auto | l2 | 10000 | 0.56511192 | 0.56511192 | 0.56511192 |
| liblinear | ovr | l1 | 0.01 | 0.404234204 | 0.404234204 | 0.404234204 |
| liblinear | ovr | l1 | 0.1 | 0.631050833 | 0.631050833 | 0.631050833 |
| **liblinear** | **ovr** | **l1** | **1** | **0.704267284** | **0.704267284** | **0.704267284** |
| liblinear | ovr | l1 | 10 | 0.661043114 | 0.661043114 | 0.661043114 |
| liblinear | ovr | l1 | 100 | 0.598081376 | 0.598081376 | 0.598081376 |
| liblinear | ovr | l1 | 1000 | 0.570073878 | 0.570073878 | 0.570073878 |
| liblinear | ovr | l1 | 10000 | 0.562796339 | 0.562796339 | 0.562796339 |
| liblinear | ovr | l2 | 0.01 | 0.535450436 | 0.535450436 | 0.535450436 |
| liblinear | ovr | l2 | 0.1 | 0.656522219 | 0.656522219 | 0.656522219 |
| liblinear | ovr | l2 | 1 | 0.686404234 | 0.686404234 | 0.686404234 |
| liblinear | ovr | l2 | 10 | 0.667328261 | 0.667328261 | 0.667328261 |
| liblinear | ovr | l2 | 100 | 0.619362664 | 0.619362664 | 0.619362664 |
| liblinear | ovr | l2 | 1000 | 0.582313375 | 0.582313375 | 0.582313375 |
| liblinear | ovr | l2 | 10000 | 0.56511192 | 0.56511192 | 0.56511192 |
| newton-cg | auto | l2 | 0.01 | 0.654537435 | 0.654537435 | 0.654537435 |
| newton-cg | auto | l2 | 0.1 | 0.677582975 | 0.677582975 | 0.677582975 |
| newton-cg | auto | l2 | 1 | 0.68883008 | 0.68883008 | 0.68883008 |
| newton-cg | auto | l2 | 10 | 0.658176205 | 0.658176205 | 0.658176205 |
| newton-cg | auto | l2 | 100 | 0.609218216 | 0.609218216 | 0.609218216 |
| newton-cg | auto | l2 | 1000 | 0.578013011 | 0.578013011 | 0.578013011 |
| newton-cg | auto | l2 | 10000 | 0.565001654 | 0.565001654 | 0.565001654 |
| newton-cg | multinomial | l2 | 0.01 | 0.654537435 | 0.654537435 | 0.654537435 |
| newton-cg | multinomial | l2 | 0.1 | 0.677582975 | 0.677582975 | 0.677582975 |
| newton-cg | multinomial | l2 | 1 | 0.68883008 | 0.68883008 | 0.68883008 |
| newton-cg | multinomial | l2 | 10 | 0.658176205 | 0.658176205 | 0.658176205 |
| newton-cg | multinomial | l2 | 100 | 0.609218216 | 0.609218216 | 0.609218216 |
| newton-cg | multinomial | l2 | 1000 | 0.578013011 | 0.578013011 | 0.578013011 |
| newton-cg | multinomial | l2 | 10000 | 0.565001654 | 0.565001654 | 0.565001654 |
| newton-cg | ovr | l2 | 0.01 | 0.643180064 | 0.643180064 | 0.643180064 |
| newton-cg | ovr | l2 | 0.1 | 0.668210387 | 0.668210387 | 0.668210387 |
| newton-cg | ovr | l2 | 1 | 0.685301577 | 0.685301577 | 0.685301577 |
| newton-cg | ovr | l2 | 10 | 0.667217995 | 0.667217995 | 0.667217995 |
| newton-cg | ovr | l2 | 100 | 0.61947293 | 0.61947293 | 0.61947293 |
| newton-cg | ovr | l2 | 1000 | 0.582533907 | 0.582533907 | 0.582533907 |
| newton-cg | ovr | l2 | 10000 | 0.563898997 | 0.563898997 | 0.563898997 |
| sag | auto | l2 | 0.01 | 0.654647701 | 0.654647701 | 0.654647701 |
| sag | auto | l2 | 0.1 | 0.677582975 | 0.677582975 | 0.677582975 |
| sag | auto | l2 | 1 | 0.68883008 | 0.68883008 | 0.68883008 |
| sag | auto | l2 | 10 | 0.658176205 | 0.658176205 | 0.658176205 |
| sag | auto | l2 | 100 | 0.609438747 | 0.609438747 | 0.609438747 |
| sag | auto | l2 | 1000 | 0.578895137 | 0.578895137 | 0.578895137 |
| sag | auto | l2 | 10000 | 0.568750689 | 0.568750689 | 0.568750689 |
| sag | multinomial | l2 | 0.01 | 0.654537435 | 0.654537435 | 0.654537435 |
| sag | multinomial | l2 | 0.1 | 0.677582975 | 0.677582975 | 0.677582975 |
| sag | multinomial | l2 | 1 | 0.68883008 | 0.68883008 | 0.68883008 |
| sag | multinomial | l2 | 10 | 0.658176205 | 0.658176205 | 0.658176205 |
| sag | multinomial | l2 | 100 | 0.609549013 | 0.609549013 | 0.609549013 |
| sag | multinomial | l2 | 1000 | 0.578674606 | 0.578674606 | 0.578674606 |
| sag | multinomial | l2 | 10000 | 0.567868563 | 0.567868563 | 0.567868563 |
| sag | ovr | l2 | 0.01 | 0.643069798 | 0.643069798 | 0.643069798 |
| sag | ovr | l2 | 0.1 | 0.668210387 | 0.668210387 | 0.668210387 |
| sag | ovr | l2 | 1 | 0.685301577 | 0.685301577 | 0.685301577 |
| sag | ovr | l2 | 10 | 0.667217995 | 0.667217995 | 0.667217995 |
| sag | ovr | l2 | 100 | 0.619362664 | 0.619362664 | 0.619362664 |
| sag | ovr | l2 | 1000 | 0.58297497 | 0.58297497 | 0.58297497 |
| sag | ovr | l2 | 10000 | 0.571507333 | 0.571507333 | 0.571507333 |
| saga | auto | l2 | 0.01 | 0.654537435 | 0.654537435 | 0.654537435 |

1. **Decision Tree model results:**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Criterion** | **Max\_depth** | **Split** | **accuracy\_mean** | **accuracy\_min** | **accuracy\_max** |
| entropy | 1 | best | 0.433896 | 0.433896 | 0.433896 |
| entropy | 1 | random | 0.421877 | 0.421877 | 0.421877 |
| entropy | 2 | best | 0.458154 | 0.458154 | 0.458154 |
| entropy | 2 | random | 0.448341 | 0.448341 | 0.448341 |
| entropy | 5 | best | 0.514279 | 0.514279 | 0.514279 |
| entropy | 5 | random | 0.483956 | 0.483956 | 0.483956 |
| entropy | 7 | best | 0.536553 | 0.536553 | 0.536553 |
| entropy | 7 | random | 0.509648 | 0.509648 | 0.509648 |
| entropy | 10 | best | 0.554967 | 0.554967 | 0.554967 |
| entropy | 10 | random | 0.536553 | 0.536553 | 0.536553 |
| entropy | 100 | best | 0.653655 | 0.653655 | 0.653655 |
| entropy | 100 | random | 0.666997 | 0.666997 | 0.666997 |
| entropy | 200 | best | 0.652994 | 0.652994 | 0.652994 |
| entropy | 200 | random | 0.667769 | 0.667769 | 0.667769 |
| entropy | 300 | best | 0.644393 | 0.644393 | 0.644393 |
| entropy | 300 | random | 0.663138 | 0.663138 | 0.663138 |
| entropy | 500 | best | 0.635903 | 0.635903 | 0.635903 |
| entropy | 500 | random | 0.654317 | 0.654317 | 0.654317 |
| gini | 1 | best | 0.433455 | 0.433455 | 0.433455 |
| gini | 1 | random | 0.421877 | 0.421877 | 0.421877 |
| gini | 2 | best | 0.457603 | 0.457603 | 0.457603 |
| gini | 2 | random | 0.448341 | 0.448341 | 0.448341 |
| gini | 5 | best | 0.514941 | 0.514941 | 0.514941 |
| gini | 5 | random | 0.483956 | 0.483956 | 0.483956 |
| gini | 7 | best | 0.537104 | 0.537104 | 0.537104 |
| gini | 7 | random | 0.509648 | 0.509648 | 0.509648 |
| gini | 10 | best | 0.557063 | 0.557063 | 0.557063 |
| gini | 10 | random | 0.537325 | 0.537325 | 0.537325 |
| gini | 100 | best | 0.658617 | 0.658617 | 0.658617 |
| gini | 100 | random | 0.669864 | 0.669864 | 0.669864 |
| gini | 200 | best | 0.6638 | 0.6638 | 0.6638 |
| gini | 200 | random | 0.679127 | 0.679127 | 0.679127 |
| gini | 300 | best | 0.662918 | 0.662918 | 0.662918 |
| gini | 300 | random | 0.670857 | 0.670857 | 0.670857 |
| gini | 500 | best | 0.655861 | 0.655861 | 0.655861 |
| gini | 500 | random | 0.673944 | 0.673944 | 0.673944 |

1. **ML Model 4**