COSC 5557
Text Reviewer

Almountassir Bellah Aljazwe

April 25, 2024

1 Introduction

As technology has expanded to become an irreplaceable part of everyday life, it has allowed communication with anyone, in any part of the world, accessible through internet access. Although this has its well-known positives, the reality is it opens up the user to a potentially dangerous world, if they are not adept at using communication applications. Through a simple text, tweet, or blog post, one individual may have the ability to produce unbounded positivity; however, they also have the ability to produce a chain reaction of never-ending negativity. In addition to the fact that children can easily access these environments, the effects will only magnify throughout the future.

While an individual user cannot control what others type on the internet, they can control what they type; however, that is harder than it seems and the margin of error is too little to rely on one's self to have that discipline. This is where I believe machine learning can provide some additional support. In this wild-card project, I aim to use what I have learned to develop a machine-learning-based solution to the mentioned problem.

2 Idea

The main idea is simple. A machine learning pipeline is placed in between the user and the communication application. This pipeline is a layer of protection that takes in what the user is attempting to send, and reviews the content of the text. The text can be labeled as "green" (clean), "yellow" (suspicious), or "red" (unacceptable).

3 Dataset Collection

For this project, I had to construct my dataset as I could not find a dataset that was suitable for this particular idea. Although there was no dataset available, I managed to find a very diverse and large dataset that contains sentences mined from public web forums and blogs; the dataset can be found through the following web address : `https://digitalcommons.mtu.edu/mobiletext/1/`. Through this dataset folder, I went through the "sent_dev.txt" file and picked out 179 different sentences; each sentence was labeled manually with one of the given labels:

- Green (clean)
- Yellow (suspicious)
- Red (unacceptable)

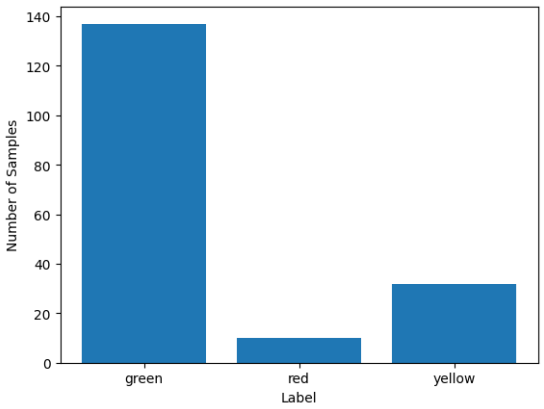
The "green" label is self-explanatory. The "yellow" label is given to sentences that are dependent on context; the sentence may be viewed as sarcasm, a joke between close friends, but it could also be viewed as unacceptable if it were sent to a person that is not known. The "red" label is given when the sentence is unacceptable; no matter the context or person, it simply should not be communicated due to its disturbing content. It is important to note that the process of manual labeling is based off of personal perspective, so a "yellow" label may be viewed as a "red" label to someone else. In addition, it fails to consider the context of the conversation; as a result, the "yellow" label is introduced to cover for the more unclear sentences.

4 Dataset Description

The dataset contains 179 rows (samples) and two columns; one is the sentence column, while the other is the target column which corresponds to the label of the sample sentence. The target column contains categorical values; each value is a string which corresponds to the sentence label. The dataset contains no missing values for all the provided samples.

A defining characteristic of this dataset is its imbalanced distribution of labels. While collecting the data from the mentioned resource, there was a greater ratio of "green" sentences than there was of the other labels; this, in a practical environment, should hopefully mirror the average text conversation or a series of tweets or posts.

Green	Yellow	Red
137/179	32/179	10/179
$\approx 77\%$	$\approx 18\%$	$\approx 6\%$



5 Experimental Set-up

5.1 The Pipelines

Going from raw text data to predicting the label of the raw text data requires a pipeline consisting of raw data pre-processing and a predictive machine learning model. To create the pipeline, Python's Sklearn "Pipeline" class is used. For the pipeline, there are three different steps. The first pipeline step is a text vectorizer; this is done through Sklearn's CountVectorizer class, which produces a matrix of token counts. The second step is an optional TF-IDF (Term Frequency - Inverse Document Frequency) matrix transformer; in other words, a pipeline can either have this as a second step or it can not. The third step is the predictive machine learning model; for this task, five different models were chosen :

- Support Vector Machine Classifier
- Logistic Regression
- Multinomial Naive Bayes Classifier
- Complement Naive Bayes Classifier
- Bernoulli Naive Bayes Classifier

For this task, I wanted to test out all possible pipeline combinations from the different steps. To do this, I created a function that allowed me to create a power set from a set of pipelines and a list of different pipeline step options. With the mentioned steps, there will be $(1 \times 2 \times 5 = 10)$ different pipeline combinations.

5.2 Pipeline Hyper-parameters

A key part of this process is hyper-parameter optimization. Not only do we have hyper-parameters for the machine learning models, but we also have important hyper-parameters for our pre-processing steps. The following tables describe the hyper-parameter spaces for each of the options in each pipeline step :

5.2.1 Pipeline Step 1

Table 1: CountVectorizer

Hyper-Parameter	Space Type	Range
Lowercase	Boolean	True or False
Stop Words	Categorical	"english" or None
Token Pattern	Categorical	Punctuation Included or Not Included

- Lowercase : whether to convert characters to lowercase, or not, before tokenizing.
- Stop Words : a list of words that are considered to be uninformative in representing the content of a text, by the CountVectorizer object.
- Token Pattern : a regular expression that defines the structure of a single token; if the definition is not met by an instance of a text, it won't be included as a token.

5.2.2 Pipeline Step 2

Table 2: TfidfTransformer

Hyper-Parameter	Space Type	Range
Use IDF	Boolean	True or False
Norm	Categorical	L1, L2, or None

- Use IDF : whether to enable inverse-document-frequency reweighting, or not.
- Norm : the normalization method.

5.2.3 Pipeline Step 3

Table 3: SVM Classifier

Hyper-Parameter	Space Type	Range
C	Float	[0.1, 1000]
Kernel	Categorical	Linear, RBF, or Sigmoid
Degree	Integer	[1, 10000]
Gamma	Categorical	Scale or Auto
Coef0	Float	[0.0, 1.0]
Tol	Float	[0.001, 1.0]

Table 4: Logistic Regression

Hyper-Parameter	Space Type	Range
Solver	Categorical	LBFGS
Penalty	Categorical	L2 or None
C	Float	[0.1, 1000]
Max Iterations	Integer	[50, 1000000]

Table 5: Multinomial Naive Bayes

Hyper-Parameter	Space Type	Range
Alpha	Float	[0.0, 1000000]

Table 6: Complement Naive Bayes

Hyper-Parameter	Space Type	Range
Alpha	Float	[0.0, 1000000]

Table 7: Bernoulli Naive Bayes

Hyper-Parameter	Space Type	Range
Alpha	Float	[0.0, 1000000]

5.3 Pipeline Evaluation

In order to fairly evaluate the optimized pipelines, nested resampling is done. The process follows the explanations by Dr. Bernd Bischl in the video titled : "I2ML - Evaluation - Resampling I" and Dr. Sebastian Raschka in the video titled : "11.5 Nested CV for Algorithm Selection (L11 Model Eval. Part 4)".

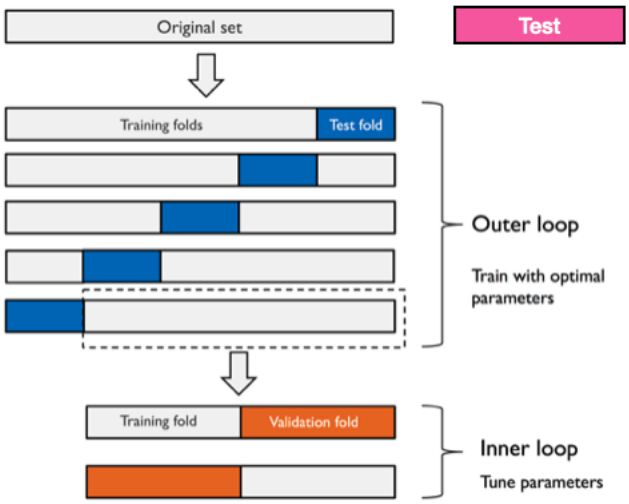


Figure 1: taken from Dr. Sebastian Raschka

In evaluating the optimized pipelines, the whole data set is used, contrary to the diagram above which leaves out a testing data set. The data set is fed into the nested resampling loops; it consists of an outer 10-fold cross validation loop and an inner 5-fold cross validation loop. For each outer loop, the data set is split up into a 4:1 ratio of training and testing samples. For each inner loop, the training set is split into an "inner-loop" training set and "inner-loop" validation set; the inner loop is used for hyper-parameter optimization.

For each of the ten outer loop training sets, a hyper-parameter optimization algorithm is run to find an optimal hyper-parameter configuration. The optimization algorithm used is "Bayesian Optimization".

In the case of the "Bayesian Optimization" algorithm, 100 iterations of the algorithm are run; each iteration results in a hyper-parameter configuration that is chosen based off past evaluations (average cross validation accuracy scores from the inner loop folds); once all 100 iterations are run, a single hyper-parameter configuration is chosen from the inner loops.

When a hyper-parameter configuration is returned from the inner loops, the pipeline is configured to the hyper-parameter configuration; then, the pipeline is evaluated on an outer loop testing set; the evaluation result is kept as a single unbiased performance estimate for the single outer loop fold. This process repeats for each of the outer loop folds. At the end, there will be ten unbiased performance estimates for each of the ten outer loop folds.

5.4 Pipeline Hyper-parameter Selection

Independent of the "Optimized Pipeline Evaluation" process is the hyper-parameter selection process; this process is exclusively focused on the selection of hyper-parameters and is not focused on providing unbiased performance estimates for the optimized pipelines. To select the optimal hyper-parameters for each pipeline, "Bayesian Optimization" is run on the entire data set and a single hyper-parameter configuration is chosen.

In the case of the "Bayesian Optimization" algorithm, 100 iterations of the algorithm are run; each iteration results in a hyper-parameter configuration that is chosen based off past evaluations (average cross validation accuracy scores from the entire data set); once all 100 iterations are finished, a single hyper-parameter configuration is chosen.

5.5 Technologies Used

The source code for all the models and the text pre-processing steps was taken from the Sklearn Python library. The source code for the "Bayesian Optimization" algorithm was taken from the "baytune" Python library.

6 Results

6.1 Optimized Pipeline Evaluations

As mentioned before, ten individual performance estimates (accuracy scores) are stored from each of the ten outer loops of the nested resampling evaluation process. For each pipeline, the ten performance estimates are plotted :

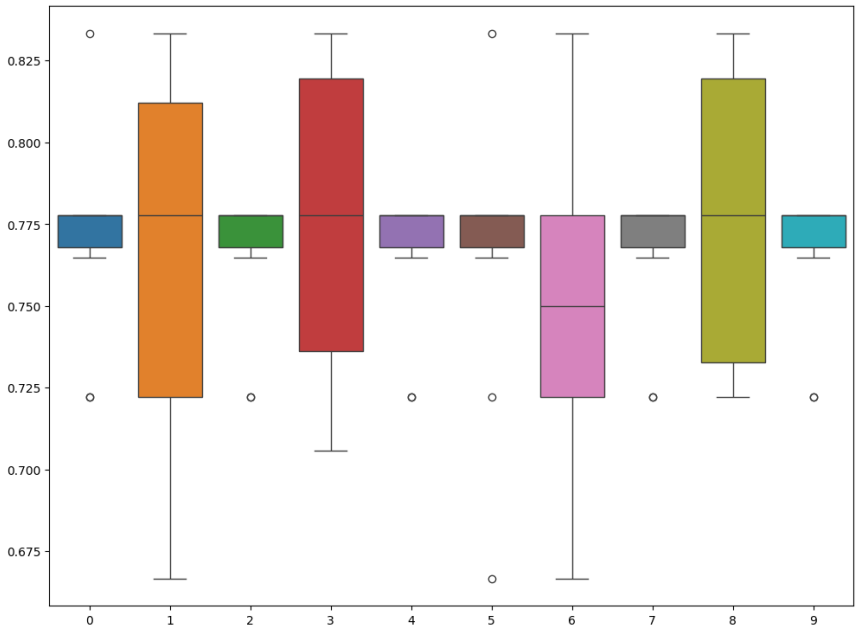


Figure 2: Box-plots of Optimized Pipeline Performance Estimates

To help distinguish the different box-plots, the following is a mapping of each of the numbers to their pipelines:

- Pipeline 0
    1. CountVectorizer
    2. TfidfTransformer
    3. SVM Classifier
- Pipeline 1
    1. CountVectorizer
    2. TfidfTransformer

3. Logistic Regression

- Pipeline 2
    1. CountVectorizer
    2. TfidfTransformer
    3. Multinomial Naive Bayes
- Pipeline 3
    1. CountVectorizer
    2. TfidfTransformer
    3. Complement Naive Bayes
- Pipeline 4
    1. CountVectorizer
    2. TfidfTransformer
    3. Bernoulli Naive Bayes
- Pipeline 5
    1. CountVectorizer
    2. SVM Classifier
- Pipeline 6
    1. CountVectorizer
    2. Logistic Regression
- Pipeline 7
    1. CountVectorizer
    2. Multinomial Naive Bayes
- Pipeline 8
    1. CountVectorizer
    2. Complement Naive Bayes
- Pipeline 9
    1. CountVectorizer
    2. Bernoulli Naive Bayes

The statistical measurements from the pipelines' performance estimates are provided in the following tables :

Table 8: Bayesian Optimization

Pipeline	Avg. Accuracy Score	Standard Deviation
0	$\approx 77\%$	≈ 3
1	$\approx 77\%$	≈ 5
2	$\approx 77\%$	≈ 2
3	$\approx 78\%$	≈ 5
4	$\approx 77\%$	≈ 2
5	$\approx 77\%$	≈ 4
6	$\approx 75\%$	≈ 6
7	$\approx 77\%$	≈ 2
8	$\approx 78\%$	≈ 4
9	$\approx 77\%$	≈ 2

6.2 Pipeline Hyper-parameter Selection

The following tables provide the results of the hyper-parameter selection process that was run for each pipeline through the "Bayesian Optimization" algorithm :

Pipeline	Avg. 10-CV Accuracy	HP Configuration
0	$\approx 80\%$	Lowercase : True Stop Words : None Token Pattern : Punctuation Included Use IDF : True Norm : L2 C : 676.47 Kernel : RBF Degree : 5704 Gamma : Auto Coef0 : 0.75 Tol : 0.44
1	$\approx 78\%$	Lowercase : True Stop Words : None Token Pattern : Punctuation Not Included Use IDF : False Norm : None Penalty : L2 C : 2.04 Solver : LBFGS Max Iterations : 821980
2	$\approx 77\%$	Lowercase : True Stop Words : None Token Pattern : Punctuation Not Included Use IDF : True Norm : L2 Alpha : 1.0
3	$\approx 79\%$	Lowercase : True Stop Words : None Token Pattern : Punctuation Included Use IDF : True Norm : L2 Alpha : 164758.0
4	$\approx 77\%$	Lowercase : True Stop Words : None Token Pattern : Punctuation Not Included Use IDF : True Norm : L2 Alpha : 1.0

Pipeline	Avg. 10-CV Accuracy	HP Configuration
5	$\approx 78\%$	Lowercase : True Stop Words : English Token Pattern : Punctuation Not Included C : 845.68 Kernel : Sigmoid Degree : 9056 Gamma : Auto Coef0 : 0.50 Tol : 0.34
6	$\approx 78\%$	Lowercase : False Stop Words : None Token Pattern : Punctuation Included Penalty : L2 C : 66.29 Solver : LBFGS Max Iterations : 950903
7	$\approx 77\%$	Lowercase : True Stop Words : None Token Pattern : Punctuation Not Included Alpha : 1.0
8	$\approx 75\%$	Lowercase : False Stop Words : None Token Pattern : Punctuation Not Included Alpha : 107147.0
9	$\approx 77\%$	Lowercase : True Stop Words : None Token Pattern : Punctuation Not Included Alpha : 1.0

The following table compares the average 10-fold cross validation accuracy scores for the non-optimized pipelines and the average 10-fold cross validation accuracy scores for the (Bayesian) optimized pipelines :

Table 9: Non-optimized vs. Optimized Pipelines Comparison

Pipeline	Non-optimized Accuracy Score	Optimized Accuracy Score
0	$\approx 77\%$	$\approx 80\%$
1	$\approx 77\%$	$\approx 78\%$
2	$\approx 77\%$	$\approx 77\%$
3	$\approx 75\%$	$\approx 79\%$
4	$\approx 77\%$	$\approx 77\%$
5	$\approx 77\%$	$\approx 78\%$
6	$\approx 77\%$	$\approx 78\%$
7	$\approx 76\%$	$\approx 77\%$
8	$\approx 72\%$	$\approx 75\%$
9	$\approx 77\%$	$\approx 77\%$

6.3 Conclusion

From our results, we can analytically observe zero to slight performance increases after optimizing each of the pipelines. The highest optimized score was 80%, which was obtained by the 0th pipeline; the lowest average optimized performance estimate was 75%, which was obtained by the 8th pipeline. Considering that the baseline accuracy is approximately 77 %, these results are not satisfiable.