Comparison of Different Supervised Learning Approaches

Ahmad Mustapha Wali, 407.

DESCRIPTION OF THE MACHINE LEARNING APPROACH

1. Gaussian Naïve Bayes

Gaussian Naive Bayes is an extension of naive Bayes. Other functions may be used to estimate the data's distribution, but the Gaussian (or Normal) distribution is the simplest to deal with since it requires just the mean and standard deviation to be estimated from the training data.

For each class value, the mean and standard deviation of each input variable (x) is calculated.

$$\text{mean}(x) = 1/n * \sum (x)$$

Where n denotes the number of occurrences and x denotes the values associated with an input variable in your training data. The standard deviation may be calculated using the following equation:

$$\text{standard deviation}(x) = \sum (1/n * \sum (x_i\text{-mean}(x)^2)$$

This is the square root of the average squared difference between each value of x and the mean value of x, where n is the number of instances, x_i is the i'th instance's specific value of the x variable, mean(x) is defined above.

The parameters used were

parameters_2 = {'var_smoothing': [1e-9, 1e-3, 1e-6, 1, 2]}

2. Multinomial Naïve bayes

The Gaussian assumption mentioned before is far from the only straightforward assumption that may be used to characterize the generating distribution for each label. Another relevant example is multinomial naive Bayes, which assumes that the features are produced by a basic multinomial distribution. Multinomial naive Bayes is best suited for features that reflect counts or count rates since it represents the likelihood of detecting counts across several categories.

The concept is identical to that used before, except that instead of modeling the data distribution with a best-fit Gaussian distribution, we model it with a best-fit multinomial distribution. The parameters used were:

parameters_1 = {‘alpha’: [0.1, 1, 1.5, 2, 3, 5], ‘fit_prior’: [True, False]}

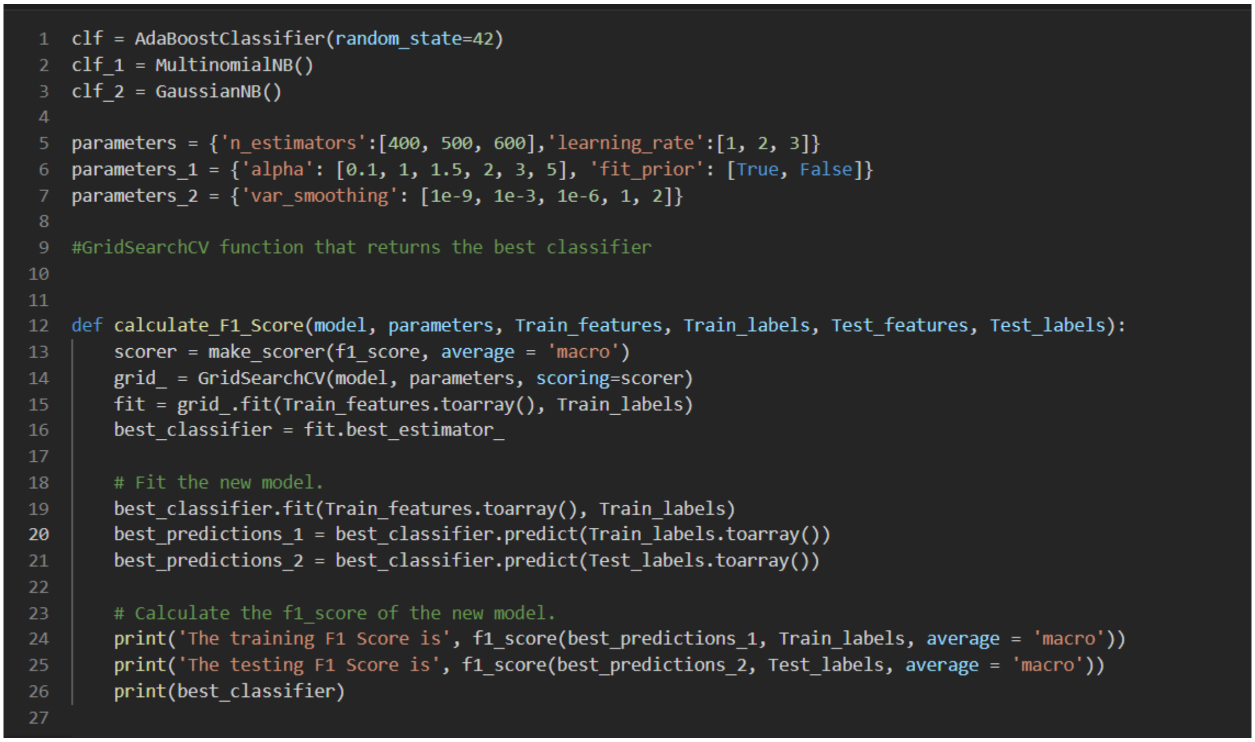
3. AdaBoost

AdaBoost (Adaptive Boosting) is a widely used boosting approach that combines numerous weak classifiers to create a single strong classifier. While no single classifier is capable of reliably predicting the class of an item, by grouping numerous weak classifiers and gradually learning from each other's incorrectly classified objects, we may construct one such strong model. The classifier given here might be any of your standard classifiers, ranging from Decision Trees (which are often used as the default) to Logistic Regression, and so on.

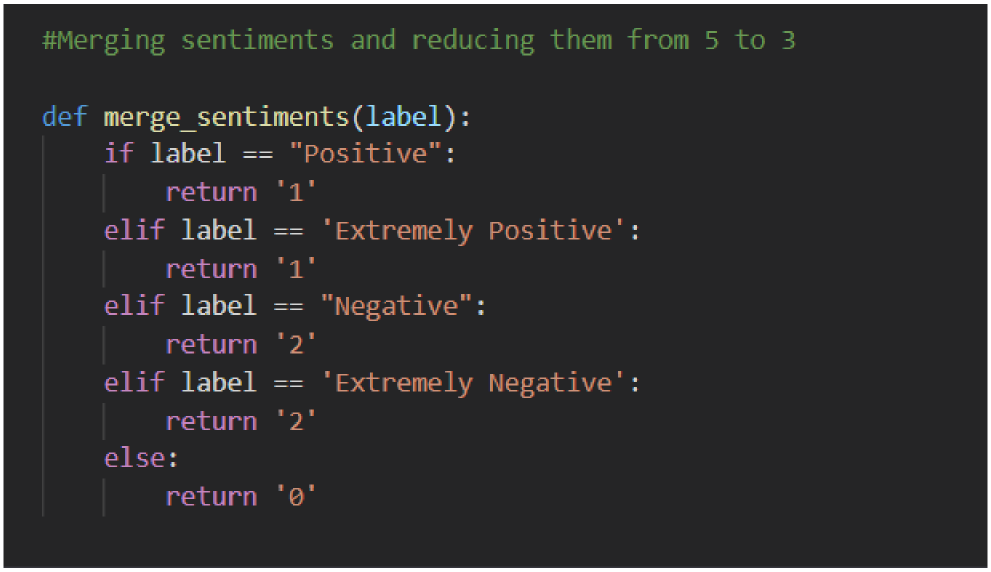
Rather than being a model in and of itself, AdaBoost may be put on top of any classifier to improve its accuracy by learning from its faults. This is why it is often referred to as the "best out-of-the-box classifier."

parameters = {'n_estimators':[400, 500, 600],'learning_rate':[1, 2, 3]}

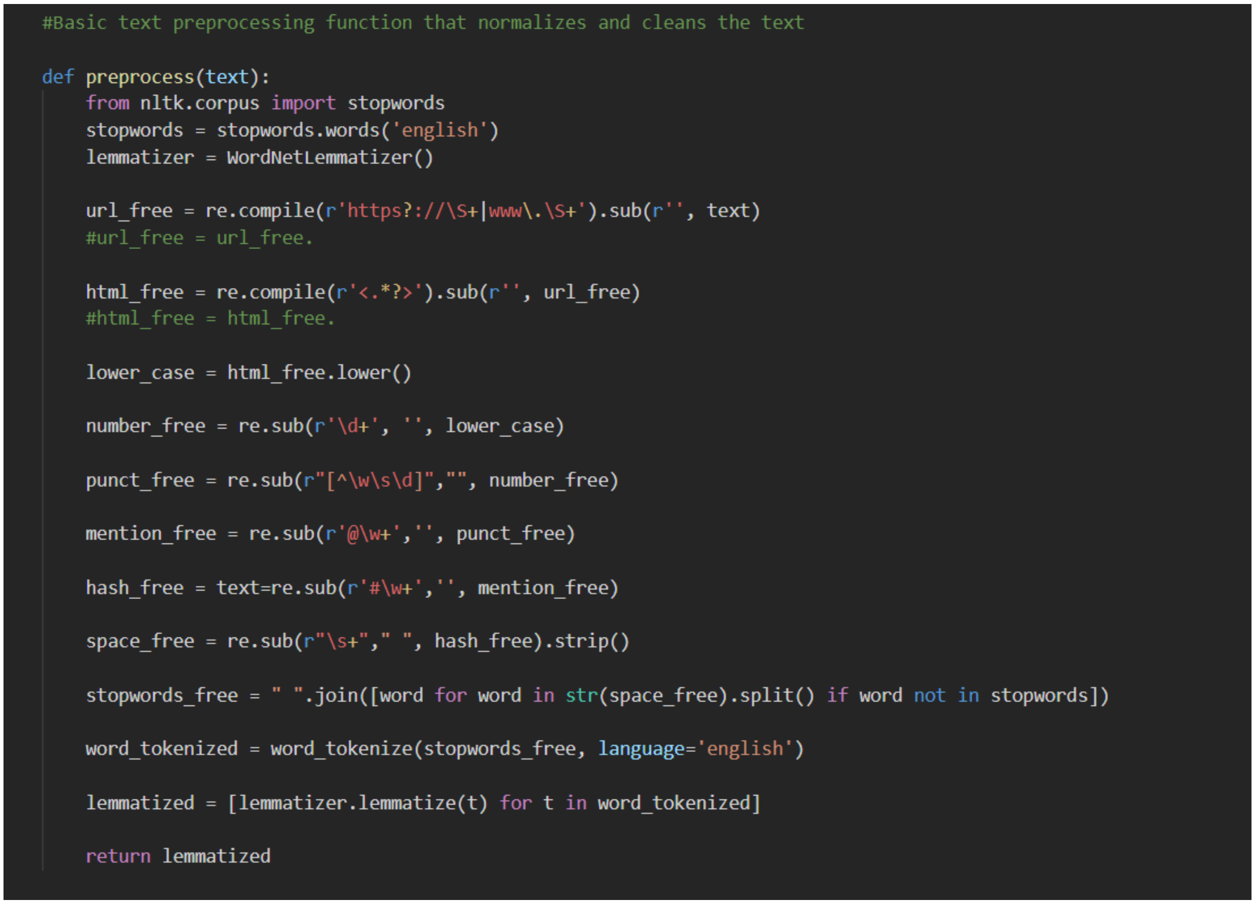
FUNCTIONS USED



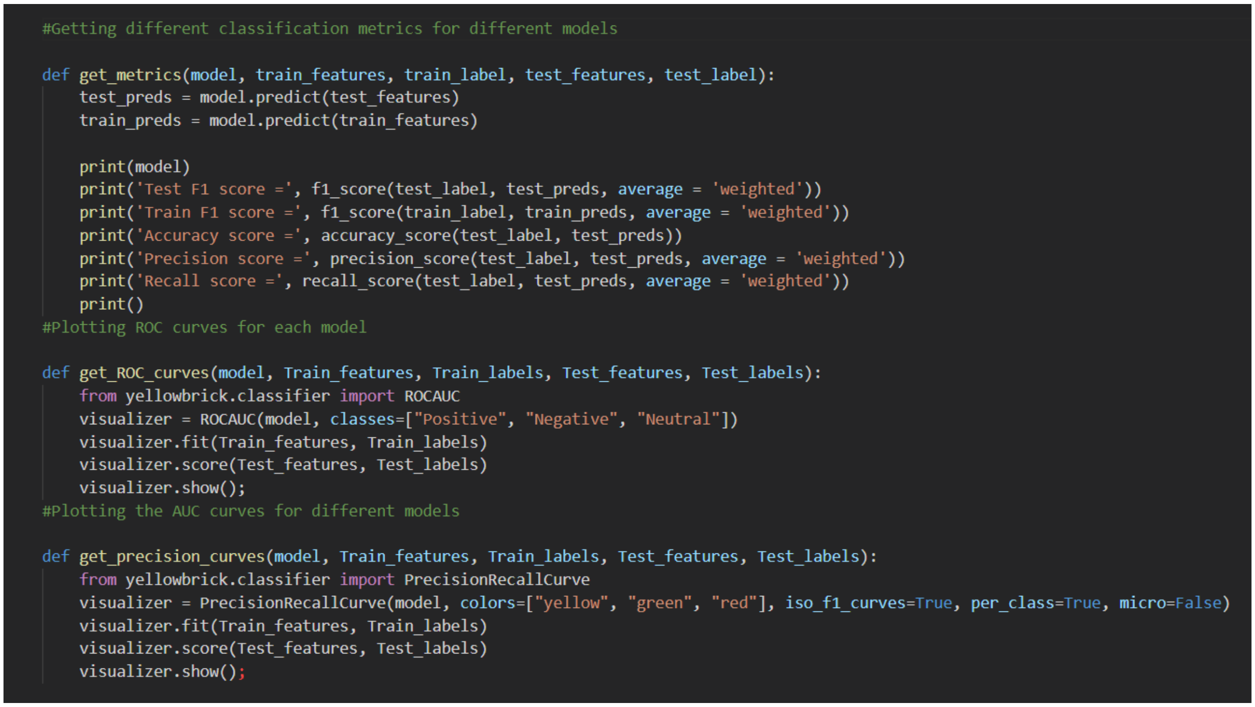
```python
clf = AdaBoostClassifier(random_state=42)
clf_1 = MultinomialNB()
clf_2 = GaussianNB()

parameters = {'n_estimators':[400, 500, 600],'learning_rate':[1, 2, 3]}
parameters_1 = {'alpha': [0.1, 1, 1.5, 2, 3, 5], 'fit_prior': [True, False]}
parameters_2 = {'var_smoothing': [1e-9, 1e-3, 1e-6, 1, 2]}

#GridSearchCV function that returns the best classifier


def calculate_F1_Score(model, parameters, Train_features, Train_labels, Test_features, Test_labels):
    scorer = make_scorer(f1_score, average = 'macro')
    grid_ = GridSearchCV(model, parameters, scoring=scorer)
    fit = grid_.fit(Train_features.toarray(), Train_labels)
    best_classifier = fit.best_estimator_

    # Fit the new model.
    best_classifier.fit(Train_features.toarray(), Train_labels)
    best_predictions_1 = best_classifier.predict(Train_labels.toarray())
    best_predictions_2 = best_classifier.predict(Test_labels.toarray())

    # Calculate the f1_score of the new model.
    print('The training F1 Score is', f1_score(best_predictions_1, Train_labels, average = 'macro'))
    print('The testing F1 Score is', f1_score(best_predictions_2, Test_labels, average = 'macro'))
    print(best_classifier)

```



```python
#Merging sentiments and reducing them from 5 to 3

def merge_sentiments(label):
    if label == "Positive":
        return '1'
    elif label == 'Extremely Positive':
        return '1'
    elif label == "Negative":
        return '2'
    elif label == 'Extremely Negative':
        return '2'
    else:
        return '0'
```



```python
#Basic text preprocessing function that normalizes and cleans the text

def preprocess(text):
    from nltk.corpus import stopwords
    stopwords = stopwords.words('english')
    lemmatizer = WordNetLemmatizer()

    url_free = re.compile(r'https?://\S+|www\.\S+').sub(r'', text)
    #url_free = url_free.

    html_free = re.compile(r'<.*?>').sub(r'', url_free)
    #html_free = html_free.

    lower_case = html_free.lower()

    number_free = re.sub(r'\d+', '', lower_case)

    punct_free = re.sub(r"[^\w\s\d]","", number_free)

    mention_free = re.sub(r'@\w+','', punct_free)

    hash_free = text=re.sub(r'#\w+','', mention_free)

    space_free = re.sub(r"\s+"," ", hash_free).strip()

    stopwords_free = " ".join([word for word in str(space_free).split() if word not in stopwords])

    word_tokenized = word_tokenize(stopwords_free, language='english')

    lemmatized = [lemmatizer.lemmatize(t) for t in word_tokenized]

    return lemmatized
```



```
#Getting different classification metrics for different models

def get_metrics(model, train_features, train_label, test_features, test_label):
    test_preds = model.predict(test_features)
    train_preds = model.predict(train_features)

    print(model)
    print('Test F1 score =', f1_score(test_label, test_preds, average = 'weighted'))
    print('Train F1 score =', f1_score(train_label, train_preds, average = 'weighted'))
    print('Accuracy score =', accuracy_score(test_label, test_preds))
    print('Precision score =', precision_score(test_label, test_preds, average = 'weighted'))
    print('Recall score =', recall_score(test_label, test_preds, average = 'weighted'))
    print()
#Plotting ROC curves for each model

def get_ROC_curves(model, Train_features, Train_labels, Test_features, Test_labels):
    from yellowbrick.classifier import ROCAUC
    visualizer = ROCAUC(model, classes=["Positive", "Negative", "Neutral"])
    visualizer.fit(Train_features, Train_labels)
    visualizer.score(Test_features, Test_labels)
    visualizer.show();
#Plotting the AUC curves for different models

def get_precision_curves(model, Train_features, Train_labels, Test_features, Test_labels):
    from yellowbrick.classifier import PrecisionRecallCurve
    visualizer = PrecisionRecallCurve(model, colors=["yellow", "green", "red"], iso_f1_curves=True, per_class=True, micro=False)
    visualizer.fit(Train_features, Train_labels)
    visualizer.score(Test_features, Test_labels)
    visualizer.show();
```

FEATURES USED

Two types of features were used;

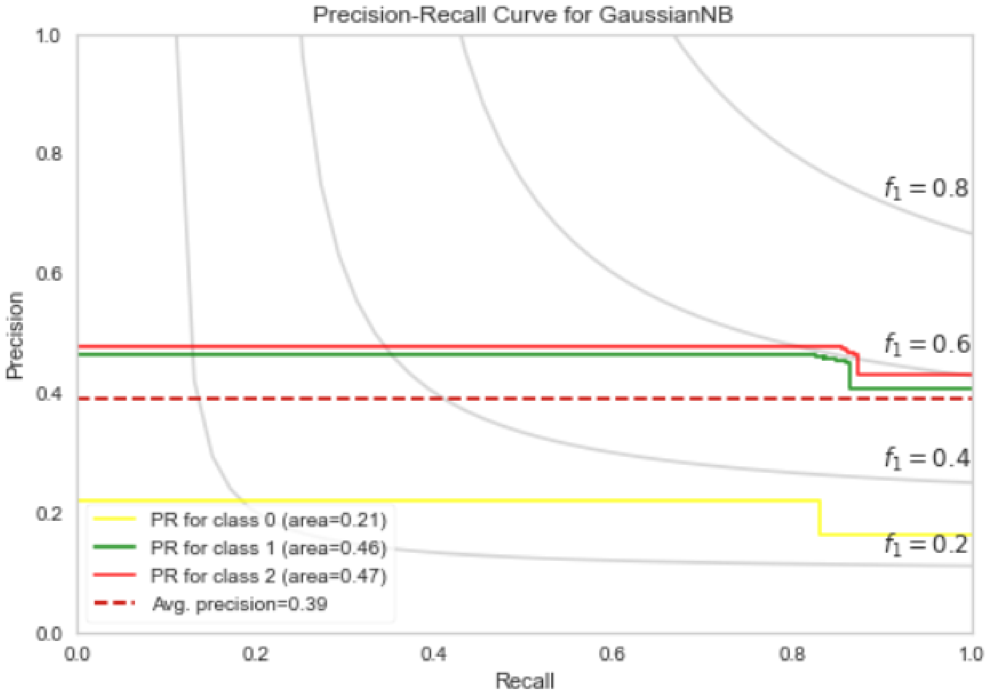
1. with 9616 features from the TF-Idf vectorizer.
2. with a maximum of 500 features set on the TF-Idf vectorizer.

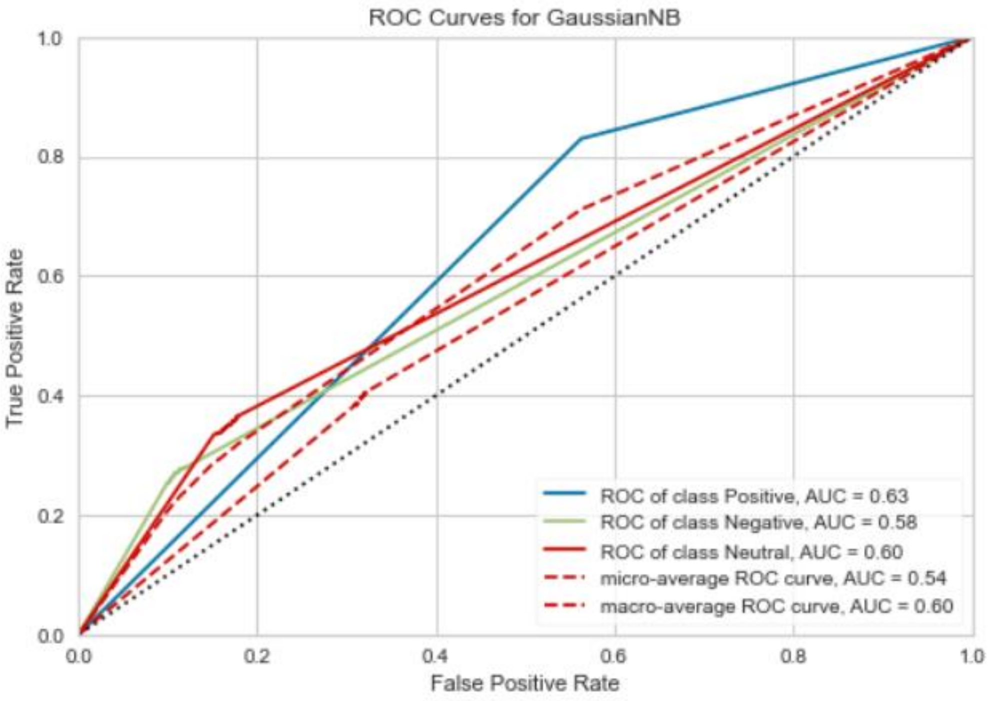
SUMMARY OF PERFORMANCE METRICS

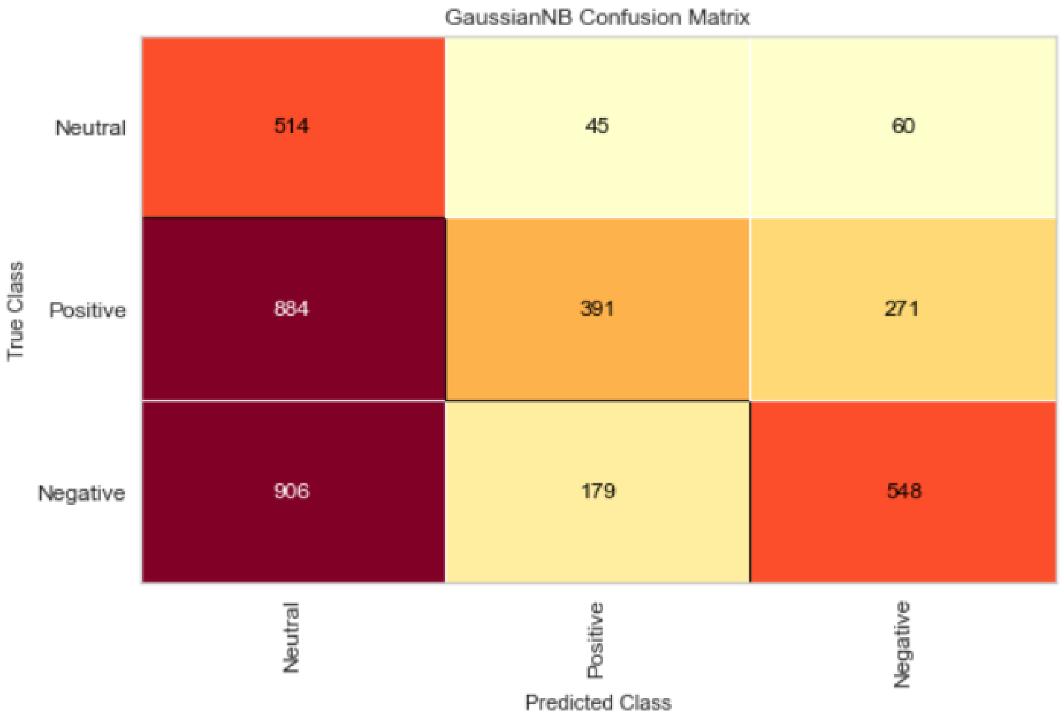
	Gaussian NB		Multinomial NB		AdaBoost	
	Feature 1	Feature 2	Feature 1	Feature 2	Feature 1	Feature 2
F1 Score (Training)	0.50946016	0.60921143	0.74176154	0.63522676	0.80701767	0.68254709
F1 Score (Testing)	0.39221584	0.57348967	0.66973808	0.61354458	0.76264901	0.63937811
Accuracy	0.38256977	0.55897841	0.68430753	0.60163244	0.75961032	0.63954713
Precision	0.56320892	0.63507642	0.67427080	0.64815021	0.77302990	0.63937527
Recall	0.38256977	0.55897840	0.68430753	0.60163243	0.75961032	0.63954713

1. It could be observed that the AdaBoost classifier performed the best.
2. The multinomial classifier performed modestly, well beyond the Gaussian classifier.
3. Both the Gaussian and multinomial classifiers overfitted with the first feature.
4. The AdaBoost classifier did not overfit to any of the features.
5. The second feature greatly reduced overfitting overall, while also increasing the baseline performance.
6. The other classifiers' performance dropped marginally after reducing the feature size to 500.

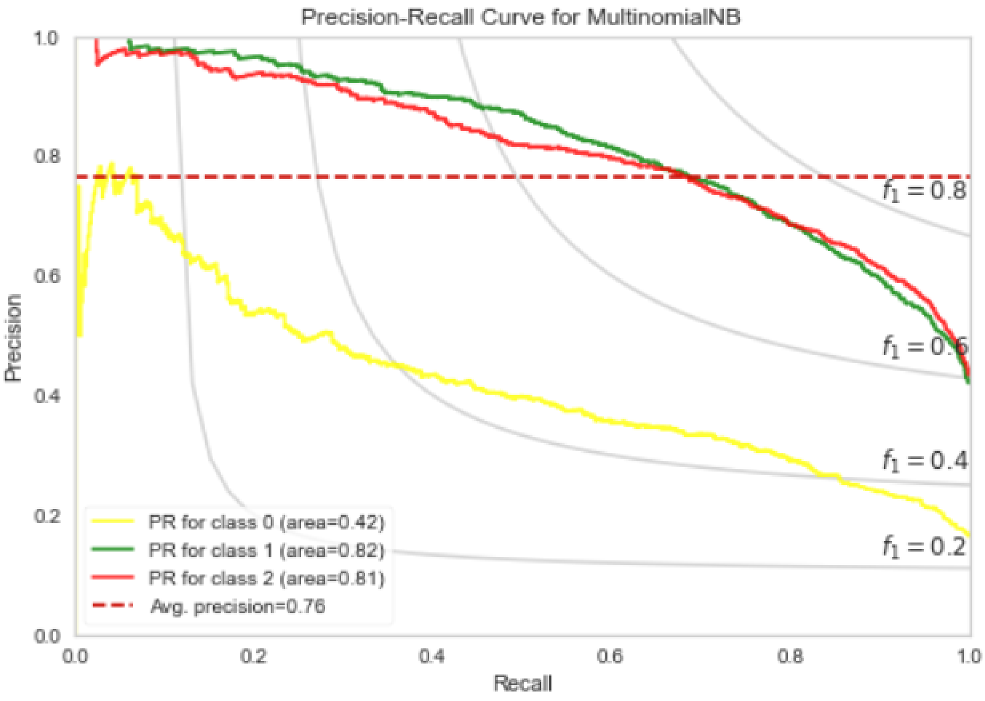
<u>GRAPHS FEATURE 1</u>

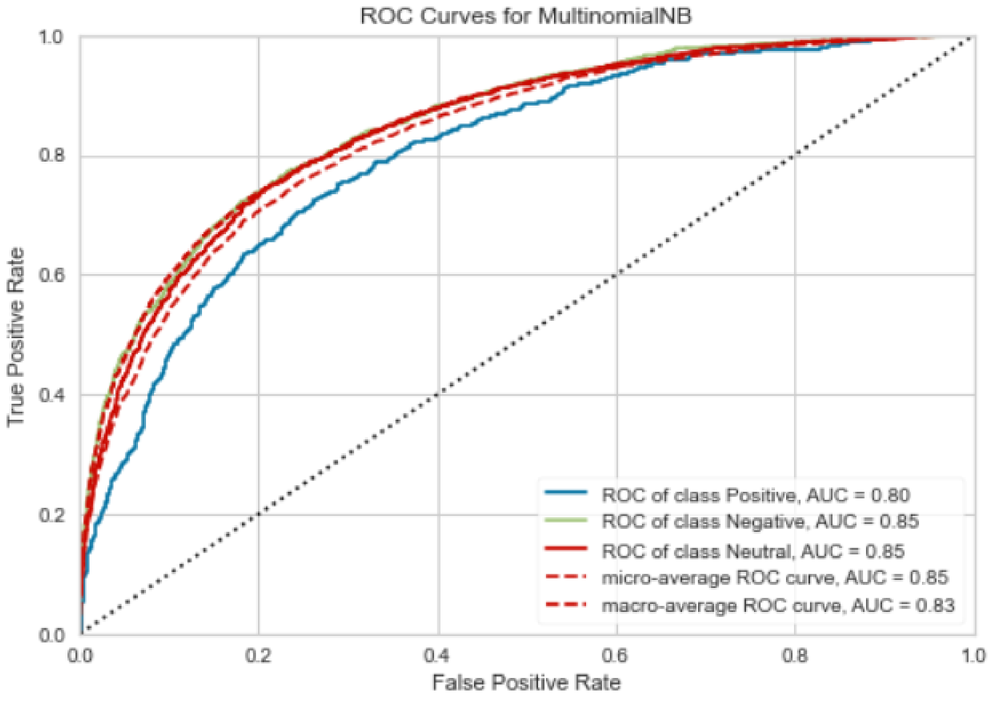


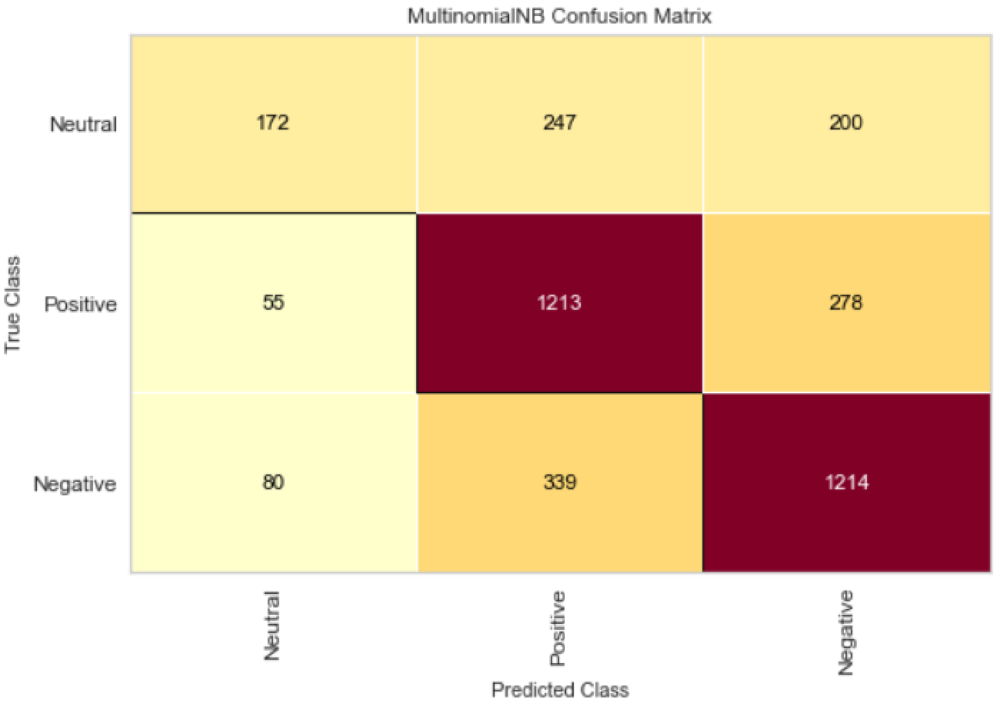


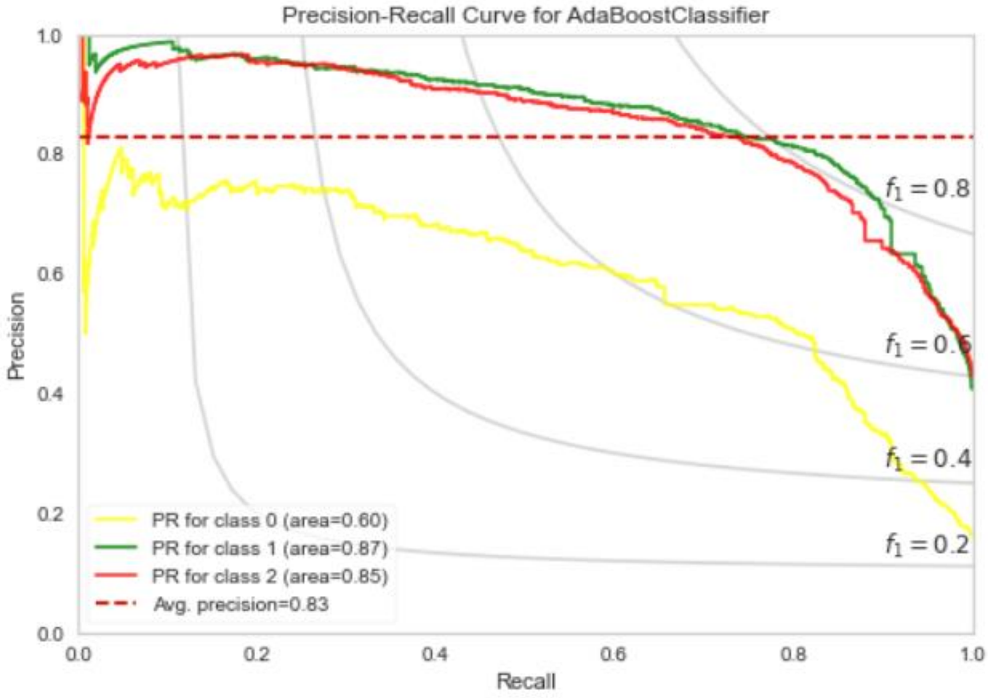

GaussianNB Confusion Matrix

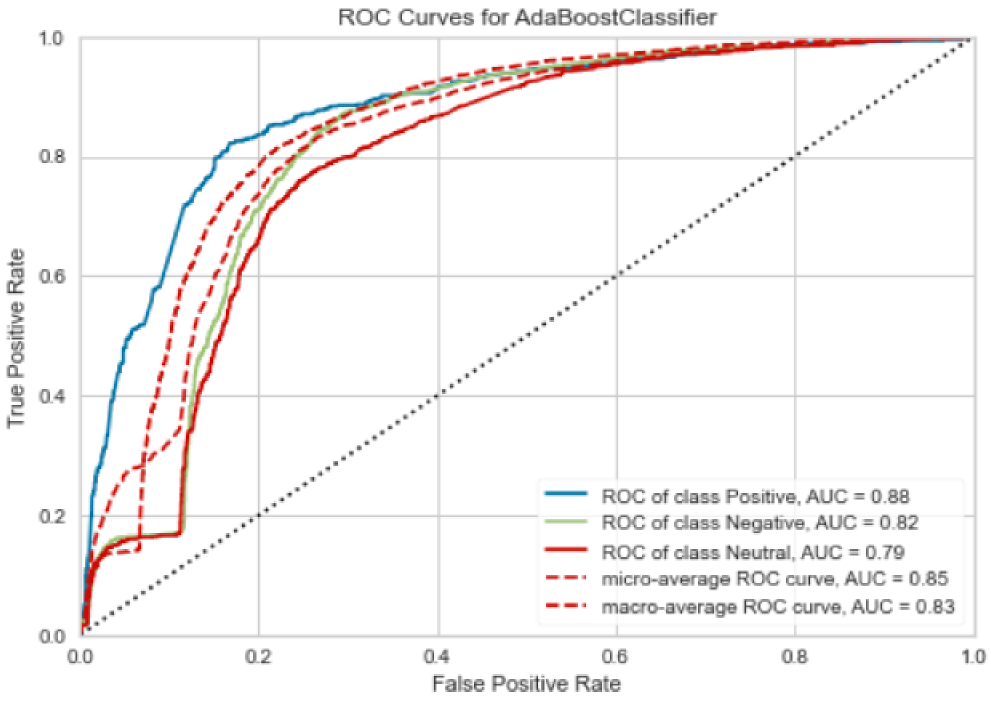
True Class	Neutral	Positive	Negative
Neutral	514	45	60
Positive	884	391	271
Negative	906	179	548

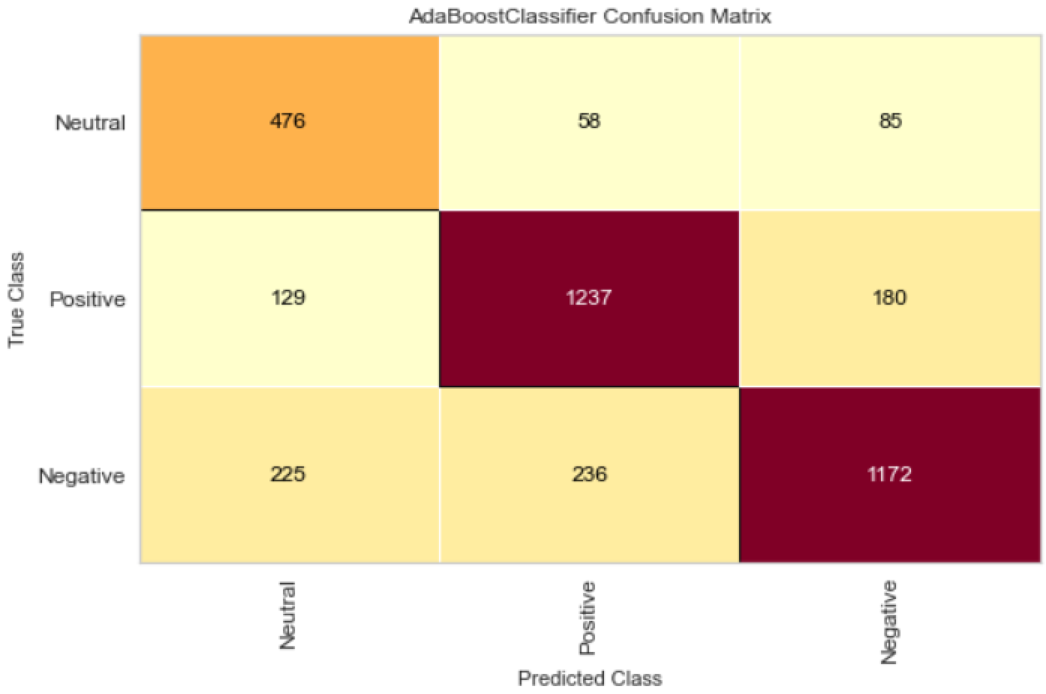

Precision-Recall Curve for MultinomialNB
Precision
Recall
1.0
0.8
0.6
0.4
0.2
0.0
0.0
0.2
0.4
0.6
0.8
1.0
$f_1 = 0.8$
$f_1 = 0.6$
$f_1 = 0.4$
$f_1 = 0.2$
PR for class 0 (area=0.42)
PR for class 1 (area=0.82)
PR for class 2 (area=0.81)
Avg. precision=0.76


ROC Curves for MultinomialNB
True Positive Rate
False Positive Rate
1.0
0.8
0.6
0.4
0.2
0.0
0.0
0.2
0.4
0.6
0.8
1.0
ROC of class Positive, AUC = 0.80
ROC of class Negative, AUC = 0.85
ROC of class Neutral, AUC = 0.85
micro-average ROC curve, AUC = 0.85
macro-average ROC curve, AUC = 0.83

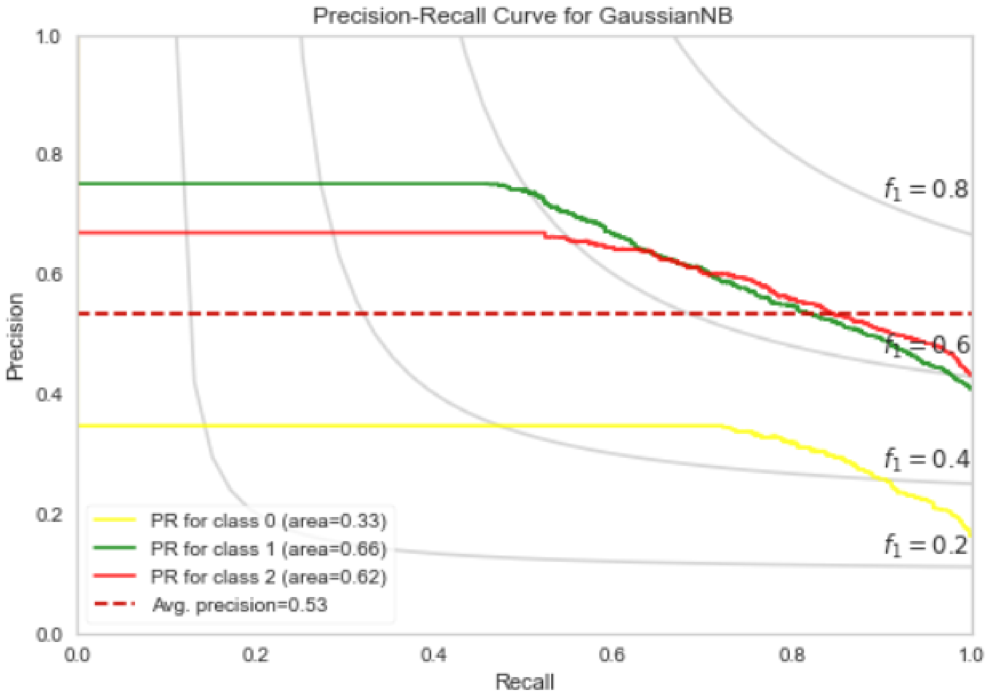

MultinomialNB Confusion Matrix
True Class
Neutral
Positive
Negative
172
247
200
55
1213
278
80
339
1214
Neutral
Positive
Negative
Predicted Class

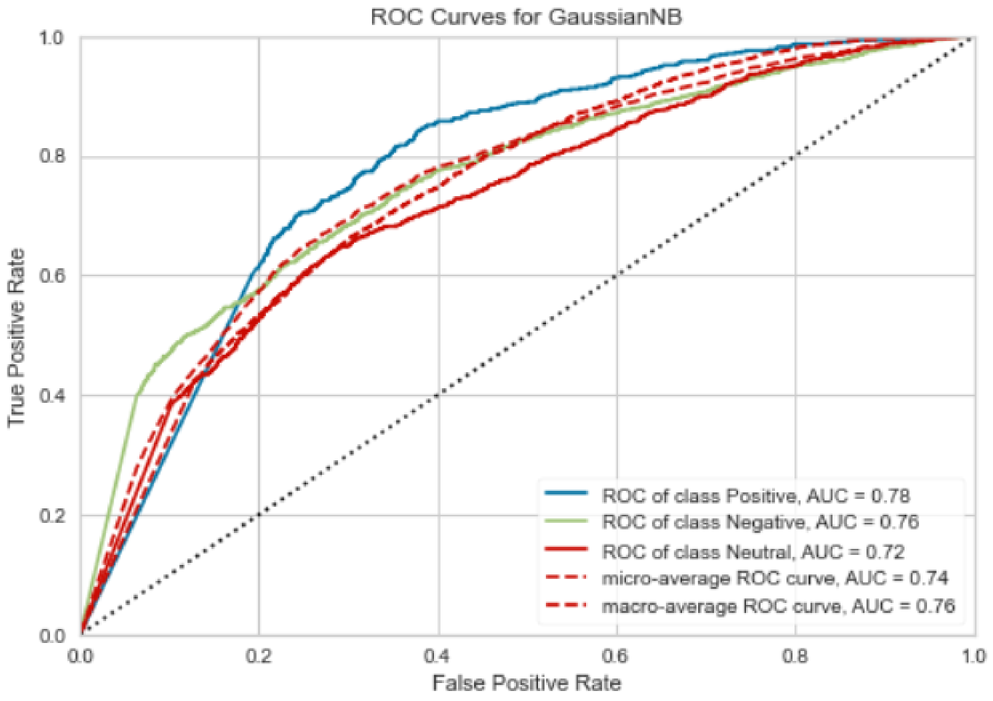

Precision-Recall Curve for AdaBoostClassifier
Precision
1.0
0.8
0.6
0.4
0.2
0.0
0.0
0.2
0.4
0.6
0.8
1.0
Recall
$f_1 = 0.8$
$f_1 = 0.6$
$f_1 = 0.4$
$f_1 = 0.2$
PR for class 0 (area=0.60)
PR for class 1 (area=0.87)
PR for class 2 (area=0.85)
Avg. precision=0.83

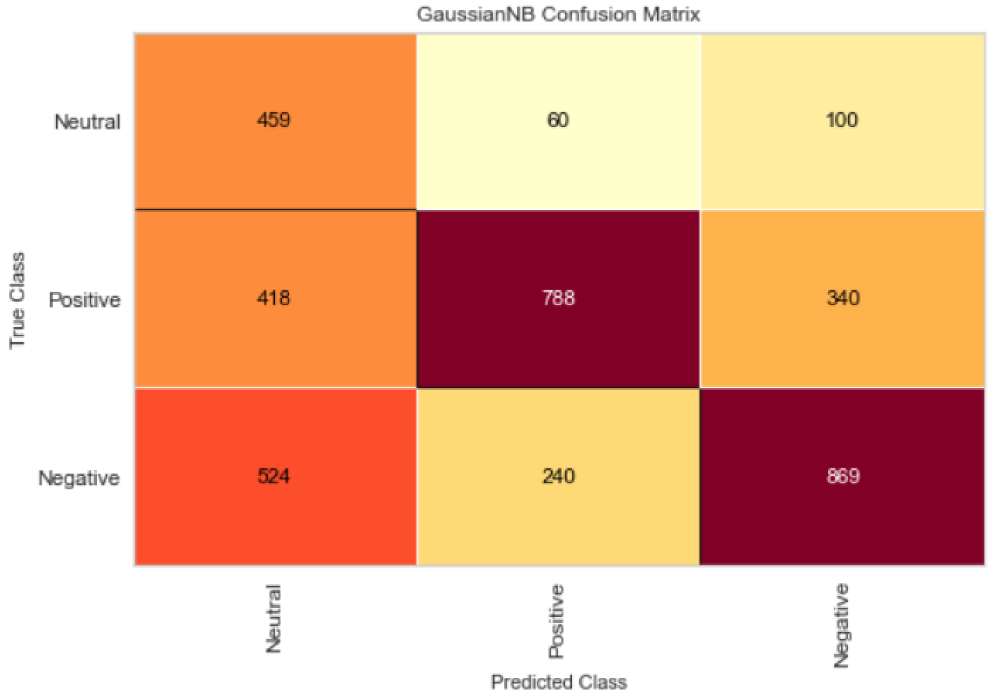




GRAPHS FEATURE 2

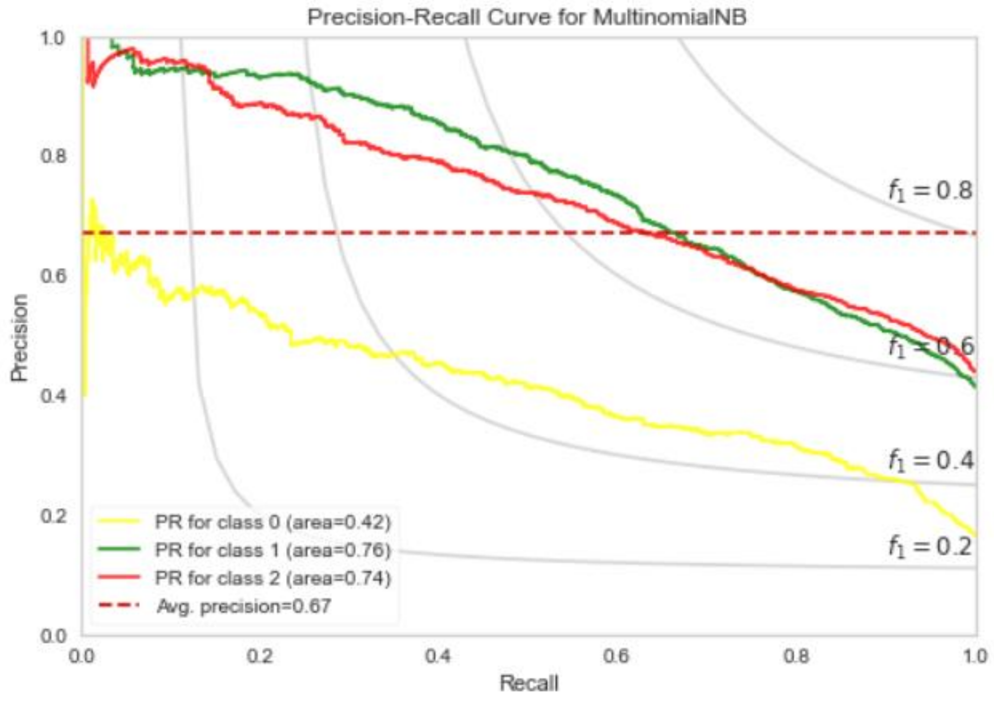

Precision-Recall Curve for GaussianNB
Precision
Recall
1.0
0.8
0.6
0.4
0.2
0.0
0.0
0.2
0.4
0.6
0.8
1.0
$f_1 = 0.8$
$f_1 = 0.6$
$f_1 = 0.4$
$f_1 = 0.2$
PR for class 0 (area=0.33)
PR for class 1 (area=0.66)
PR for class 2 (area=0.62)
Avg. precision=0.53

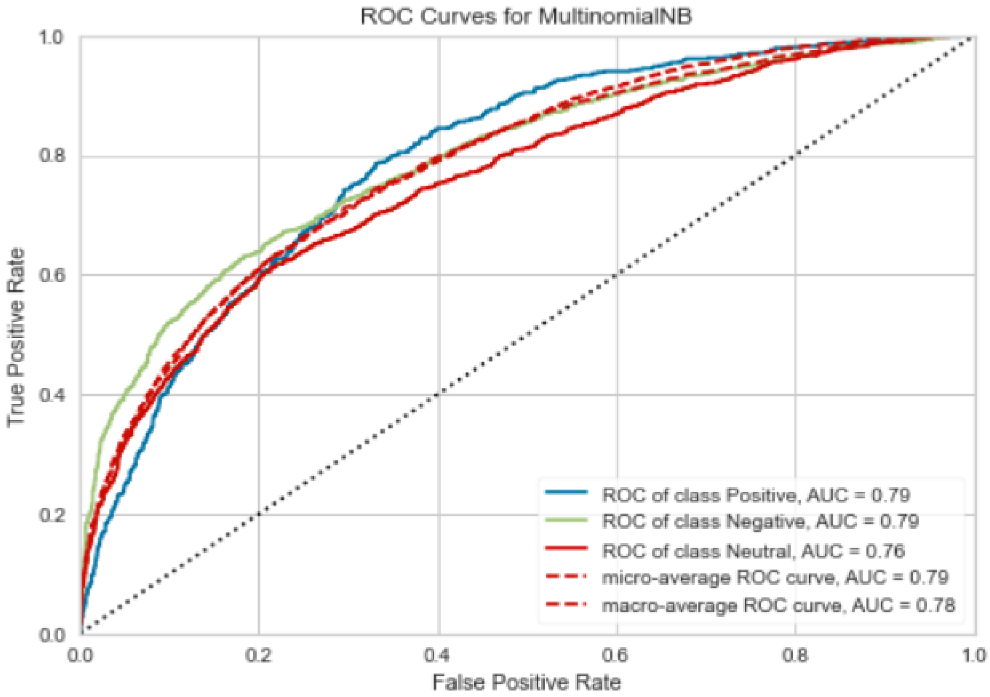

ROC Curves for GaussianNB
True Positive Rate
False Positive Rate
1.0
0.8
0.6
0.4
0.2
0.0
0.0
0.2
0.4
0.6
0.8
1.0
ROC of class Positive, AUC = 0.78
ROC of class Negative, AUC = 0.76
ROC of class Neutral, AUC = 0.72
micro-average ROC curve, AUC = 0.74
macro-average ROC curve, AUC = 0.76

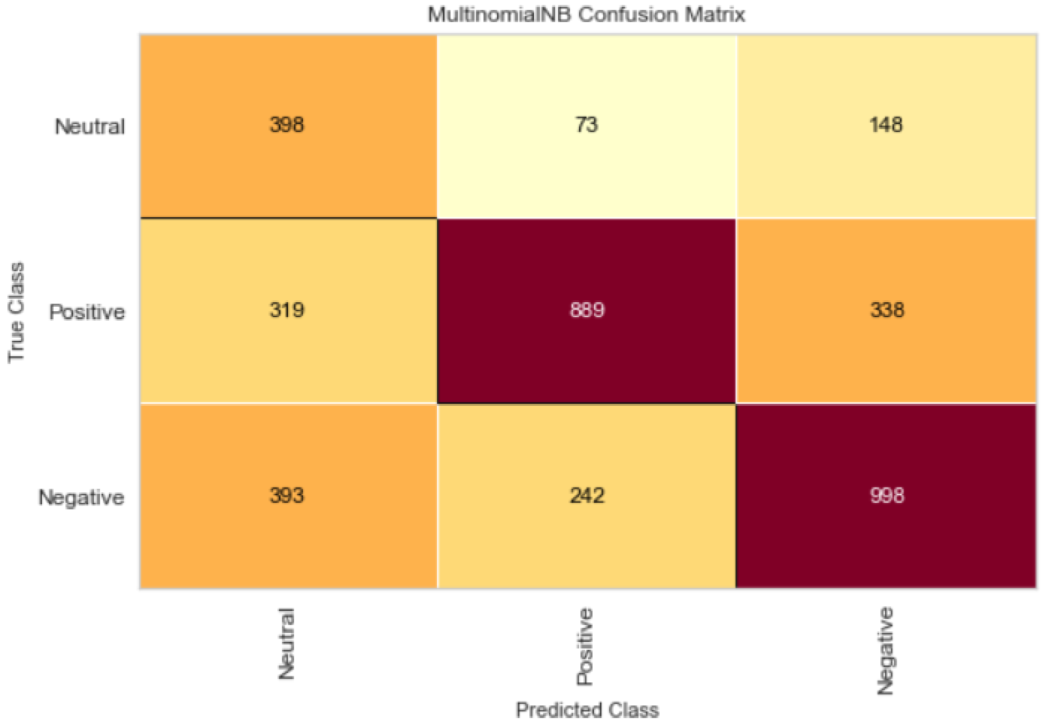


GaussianNB Confusion Matrix

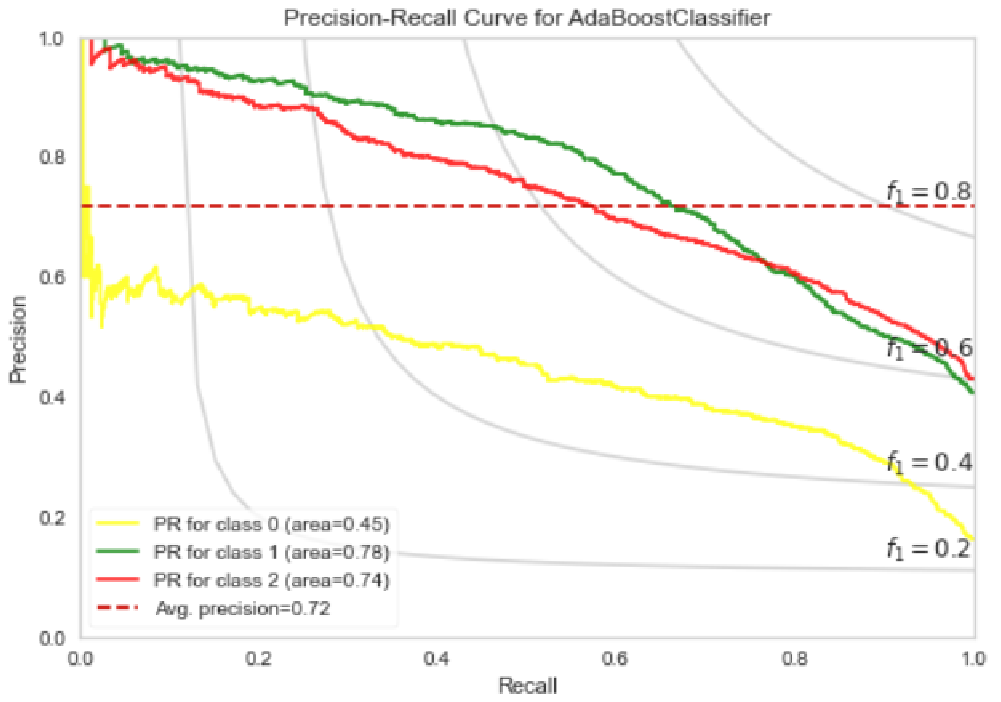
True Class \ Predicted Class	Neutral	Positive	Negative
Neutral	459	60	100
Positive	418	788	340
Negative	524	240	869

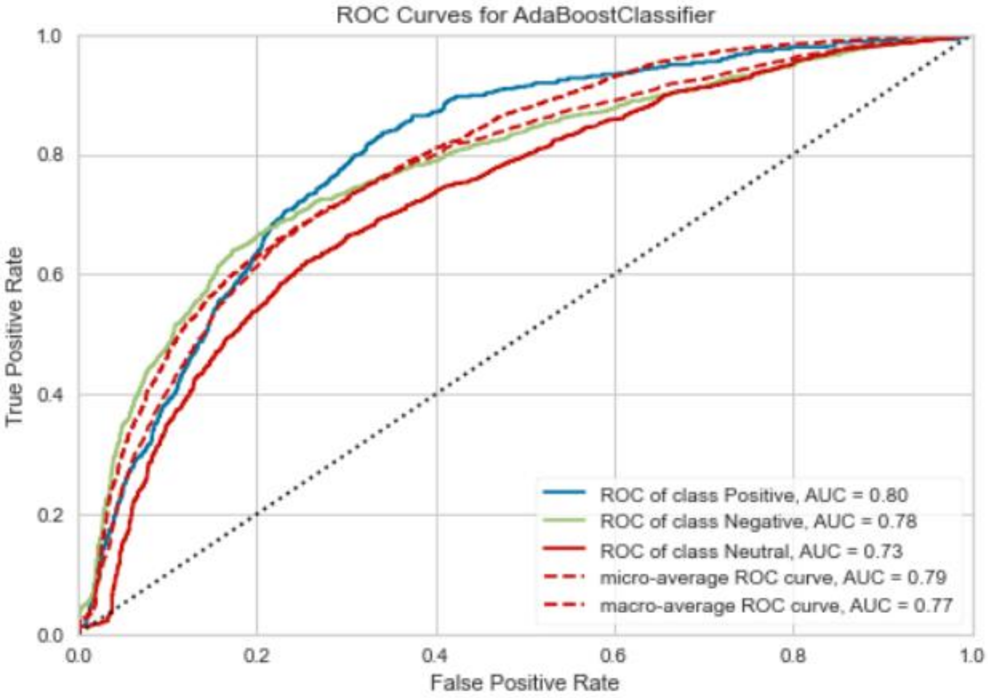


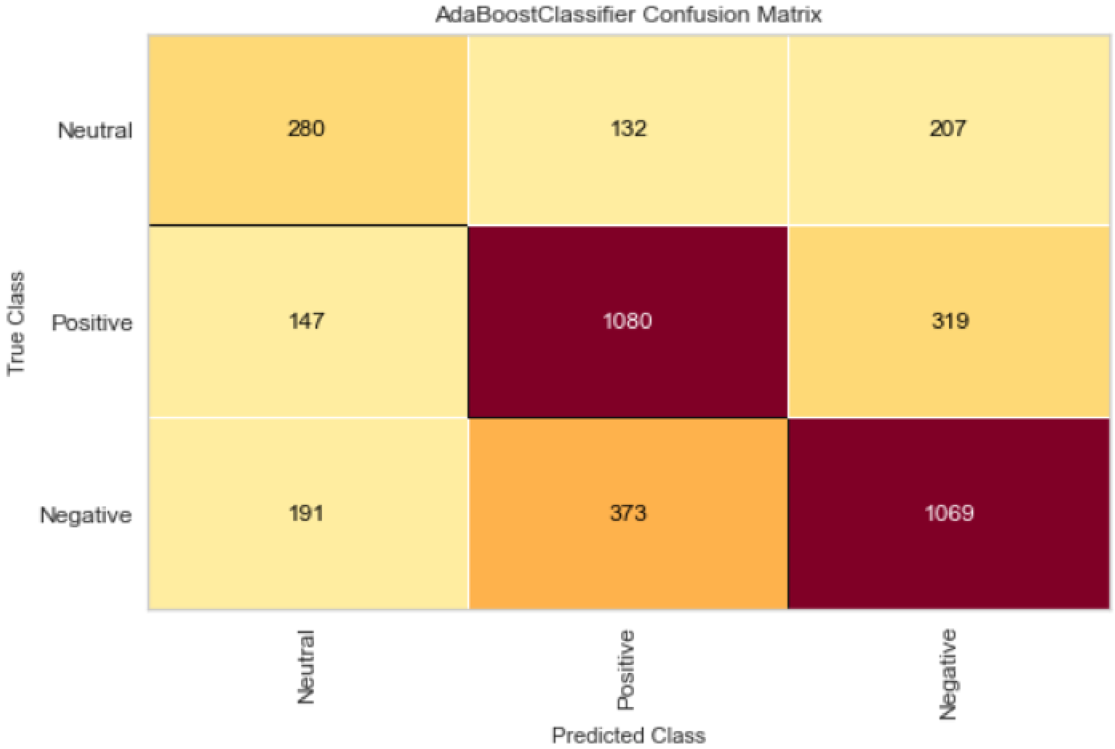




	Neutral	Positive	Negative
Neutral	398	73	148
Positive	319	889	338
Negative	393	242	998


Precision-Recall Curve for AdaBoostClassifier
Precision
Recall
1.0
0.8
0.6
0.4
0.2
0.0
0.0
0.2
0.4
0.6
0.8
1.0
$f_1 = 0.8$
$f_1 = 0.6$
$f_1 = 0.4$
$f_1 = 0.2$
PR for class 0 (area=0.45)
PR for class 1 (area=0.78)
PR for class 2 (area=0.74)
Avg. precision=0.72


ROC Curves for AdaBoostClassifier
True Positive Rate
False Positive Rate
1.0
0.8
0.6
0.4
0.2
0.0
0.0
0.2
0.4
0.6
0.8
1.0
ROC of class Positive, AUC = 0.80
ROC of class Negative, AUC = 0.78
ROC of class Neutral, AUC = 0.73
micro-average ROC curve, AUC = 0.79
macro-average ROC curve, AUC = 0.77



BIBLIOGRAPHY

Brownlee, J. (2020, August 15). *Naive Bayes for Machine Learning*. Machine Learning Mastery. https://machinelearningmastery.com/naive-bayes-for-machine-learning/

Kurama, V. (2021, April 9). *A Guide To Understanding AdaBoost*. Paperspace Blog. https://blog.paperspace.com/adaboost-optimizer/#:%7E:text=AdaBoost%20is%20an%20ensemble%20learning,turn%20them%20into%20strong%20ones.

VanderPlas, J. (2017). *Python Data Science Handbook: Essential Tools for Working with Data* (1st ed.). O'Reilly Media.