1 Introduction

The goal of this project is to build a model that will predict elections. Every part of the previous chapters were a build up to create a model. Data has been collected, cleaned, and organized. Tweets have been evaluated with VADER. This chapter will cover the model building and the results from the model.

2 Approach

Part of planning and designing a model is an understanding of the model's purpose. What is the model going to do? Will it make a decision? A recommendation? These are questions that will determine the type and effectiveness of the model that's built.

The challenge with elections is that they are a zero-sum game. When a candidate wins an election, the other candidate loses. The approach taken is to remove the game entirely. Rather than focus on an election the model will focus on a candidate, and act as if they exist in a vacuum. It will predict a candidate's chances of winning their election independent of the other candidates' own probability. That means that the sum of the winning probabilities for all candidates in a given election is not always 1.

This approach will challenge the model constructing portion. This is a shift away from using the tweets as express data input. It's instead a higher-level approach. We will use the summary statistics per candidate as input instead.

3 A Machine Learning Overview

The layman's definition of machine learning is to "program a computer to accomplish a task that you don't know how to do". Most coding is to explicitly define a task. Machine learning offers the ability to teach an algorithm the rules of the task, and then allow the algorithm to find its own method to accomplish it. There are many subfields, but the type of learning we'll focus on is supervised learning.

Supervised learning is a type of learning where an algorithm is given input, output pairs and uses those pairs to infer the function. To put it more simply, we give the algorithm both the question and the answer, and it finds out the method to solve it. When creating a supervised learning model, it will need to be trained and tested. A full set of data is split into

training and testing sets. The model will use the training set to create the best map it can. The testing set evaluates the map's effectiveness.

A clean, extensive, and unbiased data set is crucial to a successful supervised learning model. The maps that the model creates is reliant entirely on data. A biased dataset will result in a biased model. A few examples of supervised learning include linear and logistic regression, perceptron learning, and naive bayes.

The specific problem in question is a binomial classification problem. Each input has two potential results - winning or losing. With binomial classification, there are a variety of models available. Due to a variety of reasons that will be thoroughly examined in the next section, we've chosen to proceed with the Naive Bayes Classifier, a machine learning technique based on Bayes Theorem of Conditional Probability.

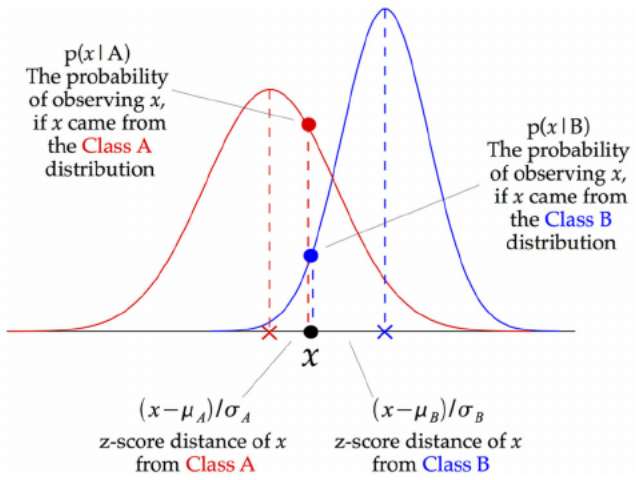
4 Understanding the Naive Bayes Classifier

The Naive Bayes classifier is a Bayesian classifier. All Bayesian classifiers are decision making systems that are based off of Bayes Theorem of Conditional Probability. Bayes Theorem makes a prediction based on observation.

Given that you have an object with feature A and a class B, the probability that the object belongs to class B based on observation A is:

$$P(A|B) = \frac{P(B|A)P(A)}{P(B)} \tag{1}$$

A Bayesian classifier uses that theorem and extends it to make a prediction. Imagine that there are classes A and B and an object with observation x.



The classifier uses Bayes Theorem, calculates $P(x|A)$ and $P(x|B)$ before making a decision based on the resulting probabilities. This example used one observation with two classes. What happens in the case where you have more than one observation? That's the problem the Naive Bayes classifier solves.

The classifier assumes that each parameter is independent from the other. Given a set of observations d and a class c_j, the probability that d came from c_j is:

$$P(d|c_j) = P(d_1|c_j) \cdot P(d_2|c_j) \cdot \ldots \cdot P(d_n|c_j) \tag{2}$$

which can be simplified to:

$$P(d|c_j) = \prod_{i=1}^{n} P(d_i|c_j) \tag{3}$$

This process repeats for all possible classes and the decision is the class with the highest probability. Thus we obtain the following classification rule:

$$P(c_j|d_1, \ldots, d_n) \propto P(c_j) \prod_{i=1}^{n} P(d_i|c_j) \implies \hat{c} = arg \max_{c_j} P(c_j) \prod_{i=1}^{n} P(d_i|c_j) \tag{4}$$

This rule determines the class $\hat{c}$ by choosing the class that maximizes probability. In this particular case, those classes are winning and losing an election.

There are several types of Naive Bayes classifiers. The one that we employed is the Guassian Naive Bayes. It's designed for models where the

parameters have continuous input. Using this model changes the way we calculate the conditional probability. With continuous input, we assume the probability to lie on a normal distribution. Keeping the same terminology as equation 2 we find $P(d_i|c_j)$ as such:

$$P(d_i|c_j) = \frac{1}{\sqrt{2\pi\sigma_{c_j}^2}} \exp\left(\frac{(d_i - \mu_{c_j})^2}{2\sigma_{c_j}^2}\right) \tag{5}$$

The parameters σ_{c_j} and μ_{c_j} are estimated using maximum likelihood.

The Naive Bayes is the optimal solution given the constraints of the problem. First, the classification problem means that the optimal solution is either a neural network, logistic regression, or Naive Bayes. The chosen approach has limited the size of our dataset. Instead of the three million tweets, the inputs are the hundred and fifty candidates. For smaller data sets, the Naive Bayes has been shown to outperform other models. Both regression and a neural network in crude terms weight their parameters to produce a final result. To ensure accurate weights, the training data needs to be extensive, something not available to us currently. The Naive Bayes probability based approach will take advantage of an unbiased dataset regardless of its size.

5 Implementation Naive Bayes in Python

The python library Sci-Kit Learn contains a variety of machine learning models. The machine learning can be imported from Sci-Kit, fitted with training data, and then tested using their predict method. Importing the library is done through the import statement.

```
from sklearn.naive_bayes import GuassianNB
```

The data to train and test the algorithm needs to be imported, split, and organized before any fitting can happen. The dataset has been stored as a CSV, and the pandas package will make importing and organization easy. They are stored in a dataframe, which is a python object that is similar to an excel table.

```
import pandas as pd

pd.read_csv(FILE_NAME)
```

The dataframe is sliced into data and labels. Data is the parameters that the algorithm will read and weight. These parameters include Tweet Count, Positive Tweets, Average Compound Sentiment Score, etc. Labels are the answers for the algorithm to check itself against. Did the candidate win or lose. To allow the algorithm to process the label, winning is a 1 and losing is a 0. Both the training and testing sets have a data and label variable.

```
train_data = np.asarray(senate.loc[:70, 'Count':'Average Compound'])
train_labels = np.asarray(senate.loc[70:, 'Winner'])
```

You'll notice that the inputs to the algorithm are all arrays of integers. Even when dealing with text, data needs to be tokenized or shaped into integers that can be organized into arrays for the algorithm to process. Shaping the information and then understanding the output is on the user.

Fitting the algorithm is easy. The gaussianNB object has a fit function that takes the training data and labels as input.

```
gnb = GaussianNB()
gnb.fit(train.data, train.labels)
```

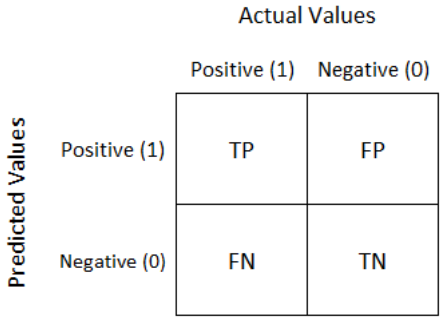
The algorithm is now trained. To evaluate, use the predict function on the testing set. The predict function will output the prediction, and not the conditional probabilities. To see those, use the predict_proba function.

```
output = gnb.predict(test_data)
probs = gnb.predict_proba(test_data)
```

With the ability to predict, we can now evaluate the effectiveness of Naive Bayes.

6 Evaluating the Model

Evaluating a model with just accuracy doesn't capture the full picture of how effective a machine learning model is. It is important to examine subsets of the results to understand how the model interacts with specific classifications. To do this, we use the confusion matrix.



The confusion matrix outlines the four potential outcomes for any prediction - true/false positives and true/false negatives. These have been used in the medical field for decades when assessing false positives, and gained popularity in machine learning in the 1980's.

6.1 Criteria

The confusion matrix provides the base for four criteria to evaluate the model. These are precision, accuracy, recall, and f1.

Precision is the ratio of true positives to the sum of true positives and false positives. This is a metric of how often the model is right given that it predicts a candidate to win. It's an important metric because it evaluates the power of the model in regards to predicting a winner.

$$PRECISION = \frac{TP}{TP + FP} \tag{6}$$

Recall is the ratio of true positives to the sum of true positives and false negatives. It's a good measure of how well the algorithm covers actual positive values.

$$RECALL = \frac{TP}{TP + FN} \tag{7}$$

Accuracy is the ratio of true positives to total items in the test set, It's an overall measure of how good the model is. On it's own it's an incomplete metric, but it fits in quite well with the other four.

$$ACCURACY = \frac{TP + TN}{TP + TN + FP + FN} \tag{8}$$

Last, the f1 score is the weighted average of the precision and recall. It takes false positives and false negatives into account.

$$F1 = 2 \cdot \frac{PR \cdot RE}{PR + RE} \tag{9}$$

6.2 Performance

To test the model, the data is split into training and testing. First the rows are shuffled and then 75% of the data goes to training while the other 25% goes to testing. Here is the resulting confusion matrix from that aplit:

Predicted/Actual	Positive	Negative
Positive	3	8
Negative	1	26

From those, we can calculate the metrics outlined in Section 6.1:

Metric	Score
Precision	0.75
Accuracy	0.763
Recall	0.33
F1	0.458

These results are encouraging - particularly the positive precision. Essentially, when the model does decide to predict a positive result, it is likely to be right. It's clear that the model we've implemented is conservative. The significant number of false negatives combine with the low number of false positives shows that the model only predicts positive when a significant amount of evidence shows it to be true.

The weak recall is potentially a reaction to class imbalance. The negative prediction rate is 0.76. If a larger test set with better balance was included, the greater number of true positives would push the recall higher.