Implementation of **Naive Bayes** algorithm

Naive Bayes

Bayes' Theorem provides a way that we can calculate the probability of a piece of data belonging to a given class, given our prior knowledge. Bayes' Theorem is stated as:

- P(class|data) = (P(data|class) * P(class)) / P(data)

Where P(class|data) is the probability of class given the provided data. Naive Bayes is a classification algorithm for binary (two-class) and multiclass classification problems. It is called Naive Bayes or idiot Bayes because the calculations of the probabilities for each class are simplified to make their calculations tractable.

Rather than attempting to calculate the probabilities of each attribute value, they are assumed to be conditionally independent given the class value.

Data Preprocessing

Data is the most important part of machine learning to train, right and perfect calculations depend on training data.

- **Choose Data:** In this case, Data polarity dataset v2.0 (3.0Mb) (includes README v2.0): 1000 positive and 1000 negative processed reviews. Introduced in Pang/Lee ACL 2004. Released June 2004.
- **Label Data:** Data labeled with **'review', 'tag'** where review contains text which concludes positive and negative review in the tag.
- **Stop words:** Only the review section have text where a number of different stop words are used many times which have less weight in sentence or paragraph.
- **Separate By Class:** There needed to calculate the probability of data by the class they belong to, the so-called base rate. We can create a dictionary object where

each key is the class value and then add a list of all the records as the value in the dictionary.

- **Summarize Dataset:** needed two statistics from a given set of data. We'll see how these statistics are used in the calculation of probabilities in a few steps. The two statistics we require from a given dataset are the mean and the standard deviation (average deviation from the mean). The mean is the average value and can be calculated as:
    - mean = sum(x)/n * count(x)

  Where *x* is the list of values or a column we are looking for.

- **Training Data:** 1700 samples used for training a model out of 2000.
- **Testing Data:** 300 samples were used for training a model out of 2000.
- **Term Frequency - Inverse Document Frequency:** This technique is applied to quantify the words in the set of documents. Generally, compute a number instead of a word for each paragraph.

Model Training and Testing:

The program has many functions to data processing, cleaning, and calling each other but the Model of creating Naive Bayes class where calculating mean, variance, and priors after multiple examples go through on same equation which is programed, X has multiple numbers instead of words for calculations with formula, like a sentence, have 5 words in a variable so do 5 number X1, X2, X3, X4, X5, and y have a tag of belonging category.

```python
class NaiveBayes:
    def fit(self, X, y):
        n_samples, n_features = X.shape
        self._classes = np.unique(y)
        n_classes = len(self._classes)

        # calculate mean, var, and prior for each class
        self._mean = np.zeros((n_classes, n_features), dtype=np.float64)
        self._var = np.zeros((n_classes, n_features), dtype=np.float64)
        self._priors = np.zeros(n_classes, dtype=np.float64)

        for idx, c in enumerate(self._classes):
            X_c = X[y == c]
            self._mean[idx, :] = X_c.mean(axis=0)
            self._var[idx, :] = X_c.var(axis=0)
            self._priors[idx] = X_c.shape[0] / float(n_samples)

    def predict(self, X):
        y_pred = [self._predict(x) for x in X]
        return np.array(y_pred)

    def _predict(self, x):
        posteriors = []

        # calculate posterior probability for each class
        for idx, c in enumerate(self._classes):
            prior = np.log(self._priors[idx])
            posterior = np.sum(np.log(self._pdf(idx, x)))
            posterior = prior + posterior
            posteriors.append(posterior)

        # return class with highest posterior probability
        return self._classes[np.argmax(posteriors)]

    def _pdf(self, class_idx, x):
        mean = self._mean[class_idx]
        var = self._var[class_idx]
        numerator = np.exp(-((x - mean) ** 2) / (2 * var))
        denominator = np.sqrt(2 * np.pi * var)
        return numerator / denominator
```

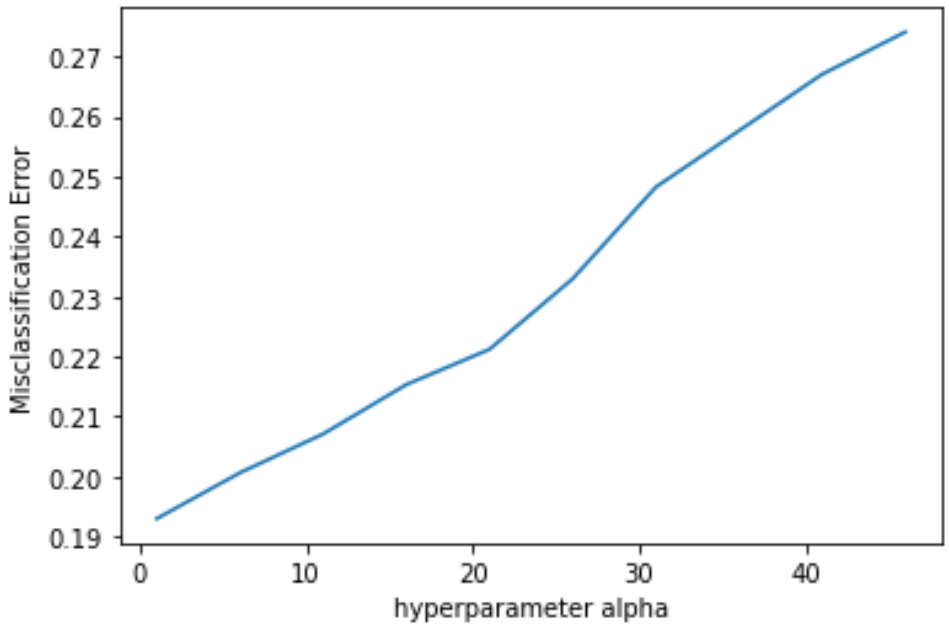
Training accuracy goes complete with 99% and test accuracy got 80 % correct. This was done with all data with the single epoch as there were many words in a single review almost led to 49000 features.

```
****Test accuracy is 80.33333333333333

****Train accuracy is 99%

In [3]:
```

Here is the smooth slope of accuracy going to words high with good accuracy.



In the testing phase, the cream color box represents the number of positive reviews the test got right and the Black color box represents the number of positive reviews the test got wrong. The orange color box represents the number of negative reviews the test got right and the purple color box represents the number of negative reviews the test got wrong