Evaluation Report: Naive Bayes Classifier for Sentiment Analysis

---

Introduction:

This project explores sentiment analysis using a Naive Bayes classifier on a large-scale dataset. The primary goal is to classify tweets into positive, neutral, or negative sentiment categories. The project compares two approaches: a custom Naive Bayes implementation with token-level log-likelihoods, and a TF-IDF-based feature extraction pipeline using Scikit-learn's MultinomialNB. Further experiments were conducted using feature selection (SelectKBest) to optimize performance

Method 1: Naive Bayes from Scratch (Token Frequency)

1.1 Assumptions:

1. Bag of Words assumption: The order of words doesn't matter.
2. Conditional Independence: Each word contributes independently to the final class probability.

1.2 Tokenization:

1. We apply a lightweight tokenization process that simply splits each tweet into a sequence of individual words.
2. These tokenized lists directly feed into the frequency counting and log-likelihood calculations, forming the core of the probabilistic classification logic used to estimate sentiment.

1.3 Mathematics:

- For a sentiment class C and a tweet D made of tokens(words) t1,t2,…,tn:

$$P(C \mid D) \propto P(C) \cdot \prod_{i=1}^{n} P(t_i \mid C)$$

Bayes Theorem with Naive Assumption

$$\log P(C \mid D) = \log P(C) + \sum_{i=1}^{n} \log P(t_i \mid C)$$

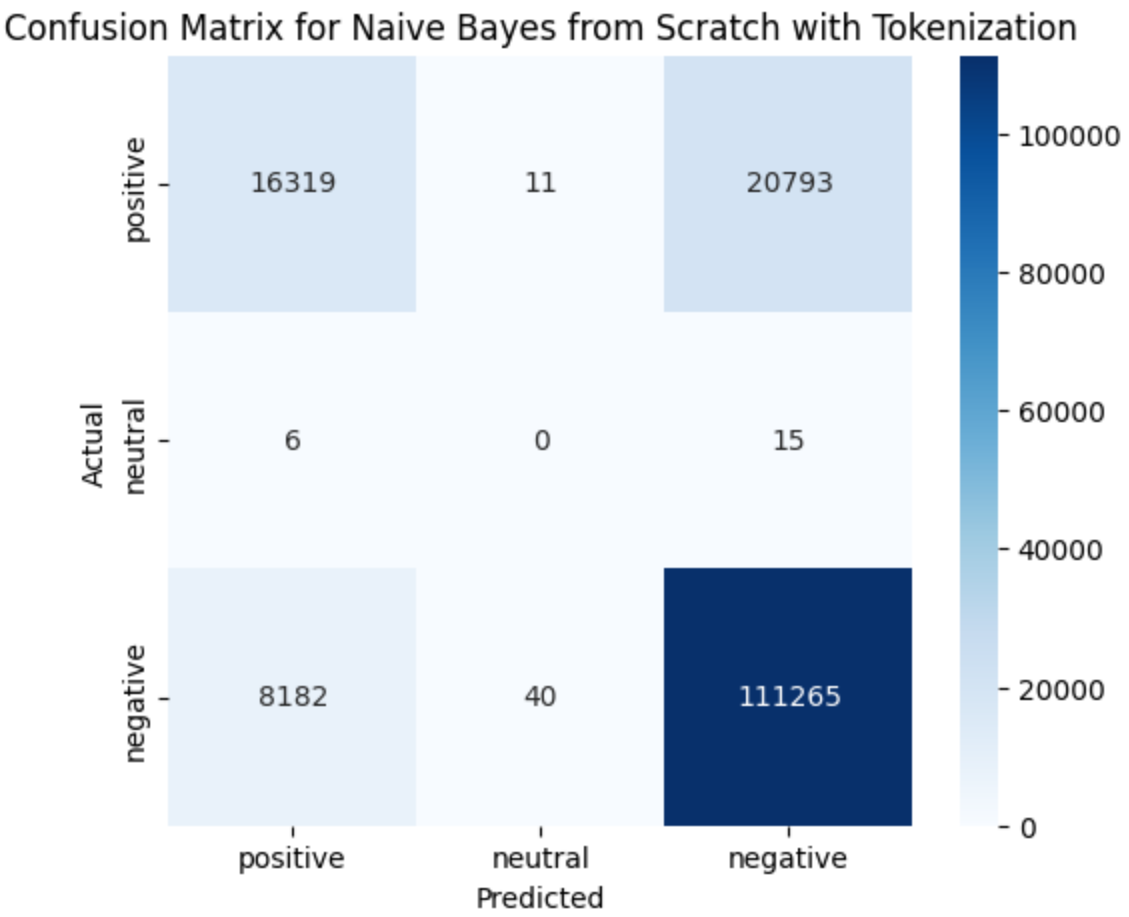
We take log to avoid underflow and simplify multiplication:

1.4 Why are we taking log?

1. Computers can only represent numbers down to a certain small value. If a number gets too close to zero (like 10^(-300)), it gets rounded to 0.This is called underflow, and it breaks computations
2. We don't actually need the probabilities themselves — just the relative magnitudes to find the maximum.
3. So instead of multiplying, We take the logarithm. This avoids underflow and turns multiplication into addition — which is faster and more stable.

1.5 Laplace Smoothing:

There can be words which do not occur in our dataset. So, the probability of that word occurring given a particular sentiment becomes zero. This would make the entire product of probabilities (and hence the classification) zero, which is undesirable. Hence we add a small constant (in our case 1) to ensure that no probability is zero.



1.6 Accuracy:

- Validation Accuracy: 81.34 %
- Test Accuracy: 81.46 %
- Classification Report:

	F-1 score	Support
Negative	0.88	119487
Positive	0.53	21
Neutral	0.00	37123
Macro avg	0.47	156631
Weighted avg	0.80	156631

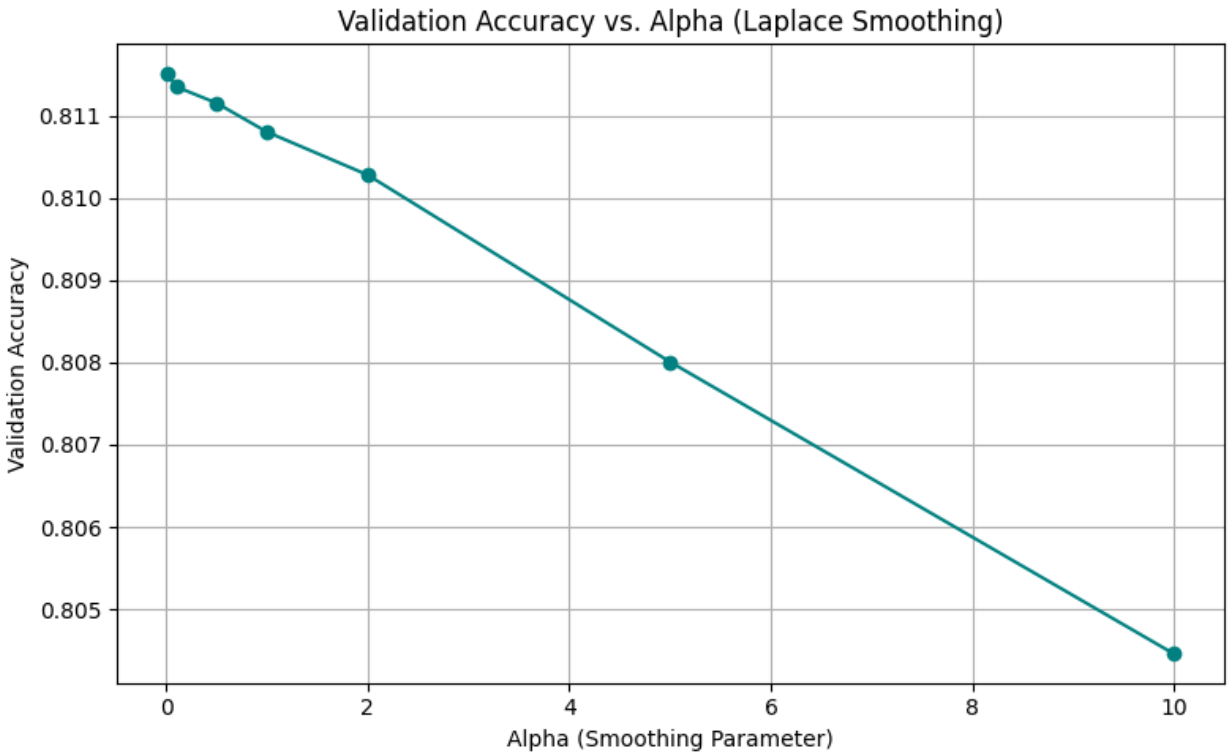
Method 2: TF-IDF + Multinomial Naive Bayes

2.1 Vectorization:

- TF-IDF features with up to 10,000 features and bigrams
- Sublinear term frequency scaling and L2 normalization

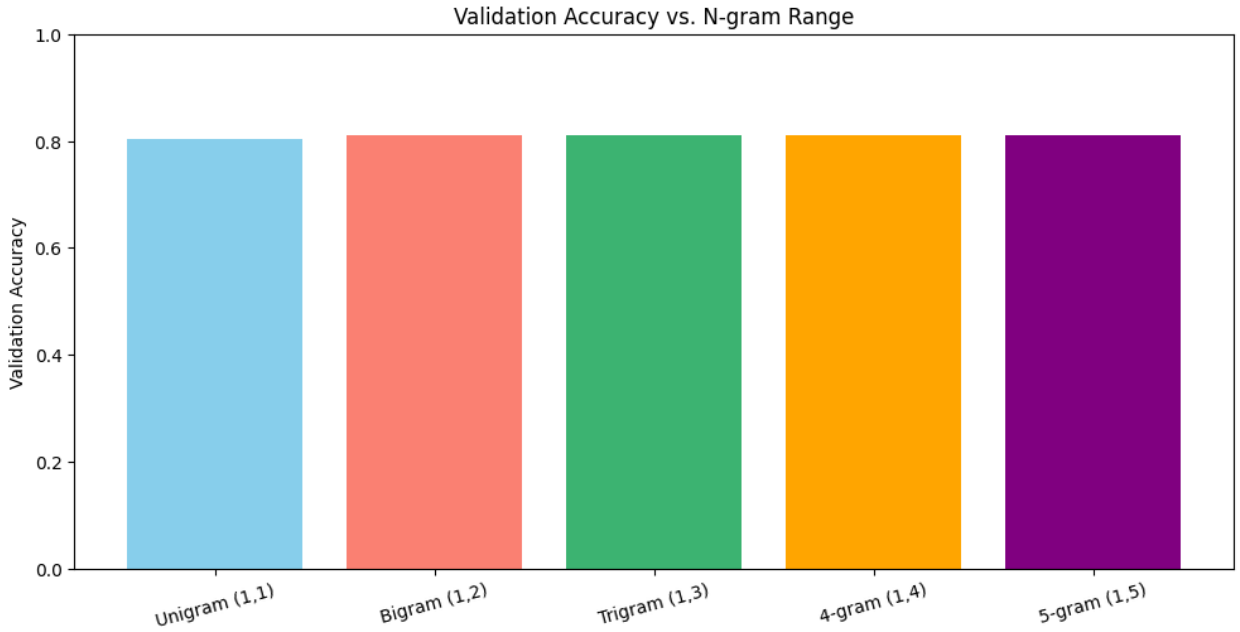
2.2 Classifier:

- MultinomialNB with custom class priors derived from training data
- Alpha smoothing parameters tested:



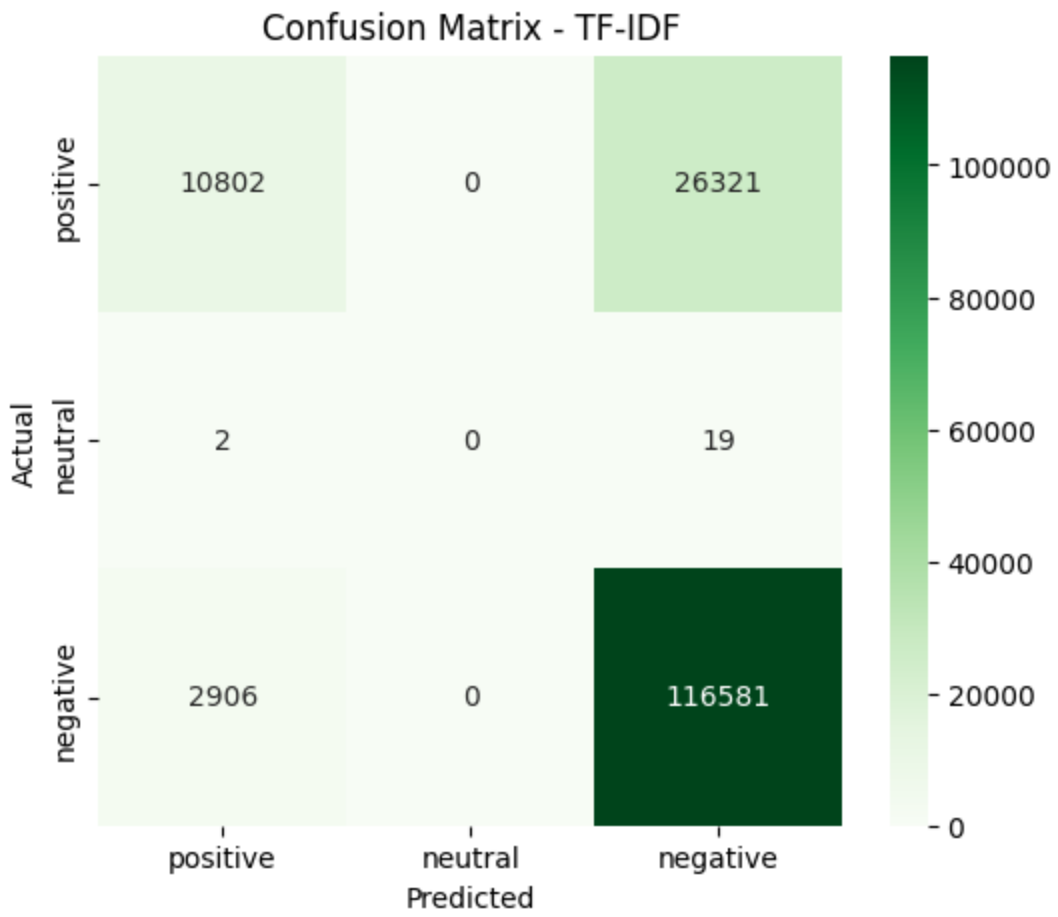
We can clearly see accuracy dropping with increase in alpha so we go for alpha=1

- N-Gram Range tested:



Chosen Metrics : alpha=1, n-gram range =2

Confusion Matrix:



Accuracy:

- Validation Accuracy: 81.08%
- Test Accuracy: 81.33%

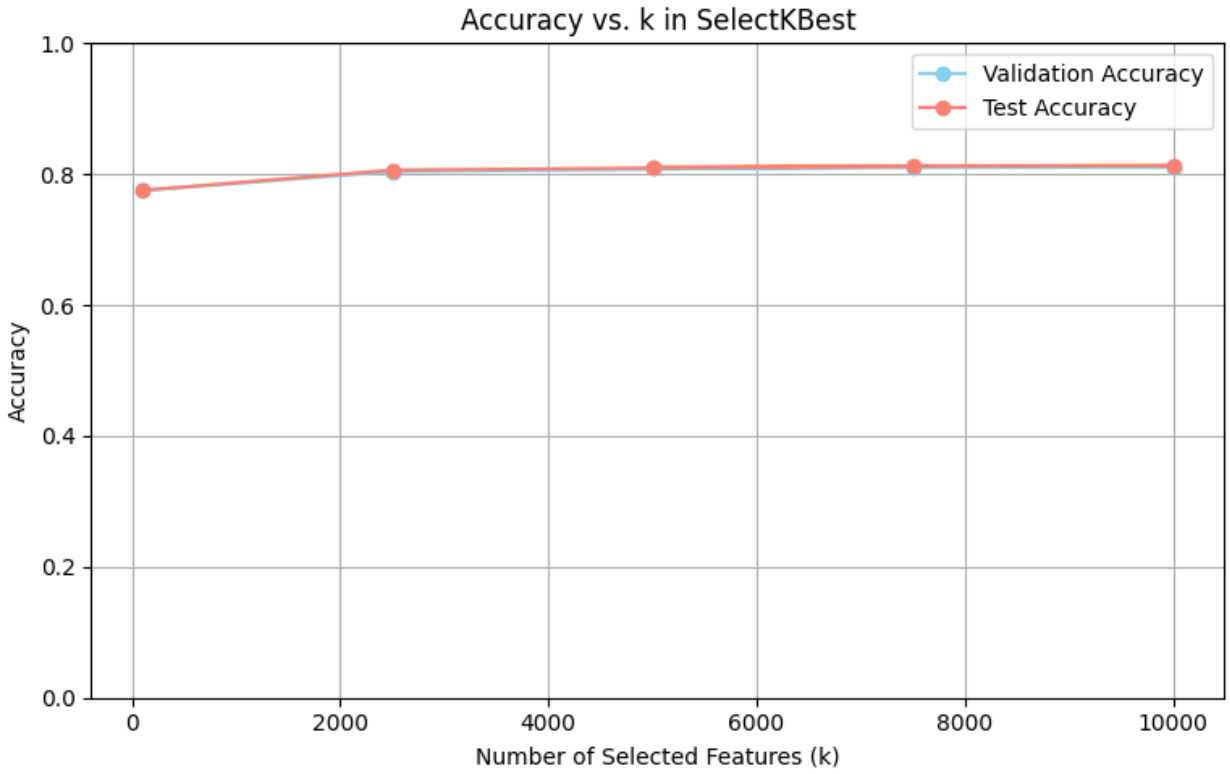
Method 3: TF-IDF + SelectKBest (Chi-Square Test)

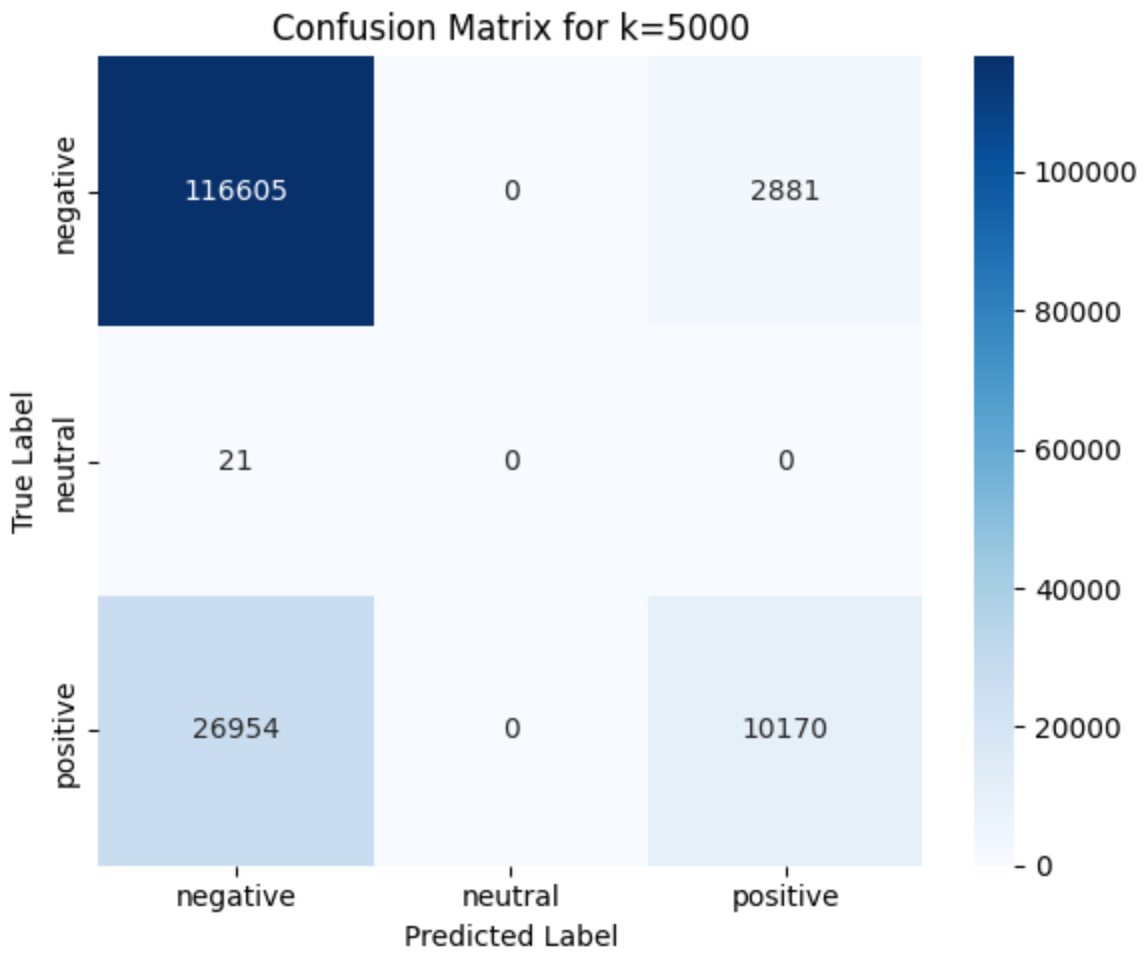
Why SelectKBest and not PCA?

SelectKBest does not create new features, it just selects the best k features out of the existing features, whereas Pca creates entirely new features. PCA can even create negative features which break Naive bayes. SelectKbest is preferred for sparse, high dimensional data.

- Tested for k values: [100, 2500, 5000, 7500, 10,000]
- Best accuracy obtained around k = 7500.

Accuracy vs. k Plot:





Summary Table:

k-value	Validation Accuracy	Test Accuracy
100	77.5%	77.55%
2500	80.40%	80.63%
5000	80.94%	81.11%
7500	81.09%	81.30%
10000	81.08%	81.33%