Naive Bayes for Project Classification

Ubayd Abdul Majit

August 3, 2025

Introduction

This report details the implementation of a Multinomial Naive Bayes classifier built from scratch to categorize student projects (A, S, G, W) based on their text descriptions. The report covers the methodology for data representation, preprocessing techniques, the implementation of both a standard and an improved model, the evaluation procedure, and a discussion of the final results.

1 Data Representation and Preprocessing

To prepare the text data for classification, it was numerically converted and subjected to a rigorous cleaning process.

1.1 Data Representation

- **Standard Naive Bayes (SNB):** Utilized a Bag-of-Words (BoW) model with unigram term frequencies.
- **Improved Naive Bayes:** Employed TF-IDF (Term Frequency-Inverse Document Frequency) to score words by their importance. Key settings included n-grams (1-3), sublinear term frequency scaling, a tuned `max_df` (0.50-0.70), and L2 normalization.

1.2 Preprocessing

A custom `preprocess` function was created to perform the following steps:

1. **Lowercase Conversion:** Standardized text to lowercase.
2. **Punctuation Removal:** Eliminated non-informative characters.
3. **Tokenization:** Split text into individual word units.
4. **URL/Number Handling:** Replaced URLs and numbers with generic `<URL>` and `<NUM>` markers.
5. **Stop Word Removal:** Removed common, non-discriminative English words.
6. **Lemmatization:** Reduced words to their root form to group different inflections.

2 Naive Bayes Implementation and Improvements

2.1 Standard Naive Bayes (SNB) Implementation

A custom class, `MyNB`, was developed to implement the Multinomial Naive Bayes algorithm. Its core logic involves:

1. Estimating class priors based on their frequency in the training data.
2. Calculating the likelihood of each word given a class.
3. Applying 'add-alpha' (Laplace) smoothing (with $\alpha = 1.0$) to handle unseen words in the vocabulary.
4. Summing log probabilities to maintain numerical stability during calculations.

Classification is performed using Bayes' theorem under the assumption of conditional independence between words.

2.2 Model Improvements

The standard model was enhanced by integrating `TfidfVectorizer` and performing systematic hyperparameter tuning.

- **TF-IDF Scoring:** Provided more discriminative word scores compared to the simple frequency counts of BoW.
- **N-grams (1-3):** Captured phrase context beyond single words.
- **Sublinear TF Scaling (`sublinear_tf=True`):** Reduced the impact of words that appear with very high frequency.
- **L2 Normalization (`norm='l2'`):** Standardized feature vectors to account for document length.
- **Chi-squared (χ^2) Feature Selection:** Selected the top 'k' most relevant features (where 'k' was tuned between 15,500 and 17,500).
- **Hyperparameter Tuning:** The Naive Bayes alpha parameter and the TF-IDF `max_df` parameter were optimized via a grid search using 5-fold cross-validation.

3 Evaluation Procedure

3.1 Methodology

The models were evaluated using 5-fold stratified cross-validation (`StratifiedKFold`) to ensure robust performance estimates and maintain representative class distribution in each fold.

3.2 Performance Metric

The primary performance metric was accuracy, calculated as:

$$\text{Accuracy} = \frac{\text{Correct Predictions}}{\text{Total Predictions}}$$

The final score reported is the mean accuracy across all 5 cross-validation folds.

4 Results and Discussion

4.1 Standard Naive Bayes (SNB) Performance

The baseline SNB model (using BoW unigrams and $\alpha = 1.0$) established a strong initial performance, achieving a mean 5-fold CV accuracy of **0.96273 (96.27%)**. This result demonstrated the inherent effectiveness of the Naive Bayes algorithm for this text classification task.

4.2 Improved Naive Bayes Performance

The improved model, incorporating TF-IDF, n-grams, Chi-squared feature selection, and tuned hyperparameters, achieved a mean 5-fold CV accuracy of **0.99068 (99.07%)**.

Table 1: Model Performance Comparison

Model	**Key Features / Settings**	**Mean CV Accuracy**
Standard Naive Bayes	BoW (Unigrams, Freq.), Alpha=1.0	0.96273
Improved Naive Bayes	TF-IDF, N-grams (1,3), Chi2 (k=17500), Alpha=1.5e-6, MaxDF=0.55, SublinearTF=True	0.99068

Optimal Hyperparameters: The best-performing configuration was found with Naive Bayes $\alpha = 1.5 \times 10^{-6}$, $k_{\chi^2} = 17500$, TF-IDF N-gram Range = (1, 3), and max_df = 0.55.

4.3 Discussion

The improved model yielded a significant mean CV accuracy increase of **2.80%**. This gain is attributed to the combination of superior feature engineering (TF-IDF and n-grams), noise reduction through Chi-squared feature selection, and optimized smoothing from a finely-tuned alpha parameter. The high CV accuracy (0.99068) indicates strong generalization and aligns well with the Kaggle public leaderboard score (0.98818), confirming the model's robustness.

5 Conclusion

This project successfully demonstrated the implementation of a Multinomial Naive Bayes classifier from scratch. A baseline model achieved 96.27% accuracy, while an improved model reached 99.07% accuracy through careful feature engineering, systematic hyperparameter optimization, and data-driven feature selection. The final model's strong cross-validation and Kaggle results confirm its effectiveness and highlight the critical impact of these optimization techniques.

6 Future Works

Future enhancements could involve exploring ensemble methods, such as combining different Naive Bayes classifiers, to potentially achieve more stable predictions. Investigating advanced feature selection techniques like ReliefF or conducting a deeper error analysis on misclassified instances could also yield further improvements.

7 Full Implementation

The following is the complete Python code used for the implementation, from data loading to final submission.

```python
# ============================
# 1. IMPORTS + SETUP
# ============================
import pandas as pd
import numpy as np
import re, string
import time
import warnings
import nltk

# nltk resource download
try:
    nltk.data.find('corpora/stopwords')
except LookupError:
    nltk.download('stopwords', quiet=True)
try:
    nltk.data.find('corpora/wordnet')
except LookupError:
    nltk.download('wordnet', quiet=True)
try:
    nltk.data.find('corpora/omw-1.4')
except LookupError:
    nltk.download('omw-1.4', quiet=True)

from nltk.corpus import stopwords
from nltk.stem import WordNetLemmatizer
from sklearn.model_selection import StratifiedKFold
from sklearn.metrics import accuracy_score
from sklearn.feature_extraction.text import TfidfVectorizer, CountVectorizer
from sklearn.feature_selection import chi2

warnings.filterwarnings("ignore", category=UserWarning, module="sklearn.
    feature_extraction.text")
np.random.seed(42)

# ============================
# 2. TEXT PREPROC
# ============================
stop_w      = set(stopwords.words('english'))
lemr        = WordNetLemmatizer()
p_trans     = str.maketrans('', '', string.punctuation)

def preprocess(text: str) -> str:
    text = text.lower().translate(p_trans)
    toks = text.split()
    res = []
    for t in toks:
        if t.startswith('http'):
            res.append('<URL>')
        elif re.fullmatch(r'\d+', t):
            res.append('<NUM>')
        else:
            lem = lemr.lemmatize(t)
            if lem.isalpha() and lem not in stop_w:
                res.append(lem)
    return " ".join(res)

# ============================
# 3. LOAD DATA
# ============================
print("loading data...")
df      = pd.read_csv('train.csv')
test_df = pd.read_csv('test.csv')
```

```
63
64 print("preproc train data...")
65 x_p = df['Description'].map(preprocess)
66 y   = df['Class'].values
67 print(f"train data shape: {x_p.shape}, target shape: {y.shape}")
68
69 # ============================
70 # 4. MULTINOMIAL NB CLASS
71 # ============================
72 class MyNB:
73     def __init__(self, alpha=1.0):
74         self.alpha = alpha
75
76     def fit(self, x_s, y_trn):
77         self.clss_, cnts = np.unique(y_trn, return_counts=True)
78         n_samps, n_feats = x_s.shape
79         self.log_p_ = {
80             c: np.log(cnts[i] / n_samps)
81             for i, c in enumerate(self.clss_)
82         }
83         self.feat_log_prob_ = {}
84         for c_ix, c in enumerate(self.clss_):
85             msk = (y_trn == c)
86             xc = x_s[msk]
87             tc_c = np.asarray(xc.sum(axis=0)).ravel() + self.alpha
88             tot_tc_c = tc_c.sum()
89
90             if tot_tc_c == 0:
91                 self.feat_log_prob_[c] = np.full(n_feats, -np.log(n_feats))
92             else:
93                 self.feat_log_prob_[c] = np.log(tc_c / tot_tc_c)
94
95     def predict_log_proba(self, x_s):
96         jlls = []
97         for c in self.clss_:
98             lp_c = self.log_p_[c]
99             flp_c = self.feat_log_prob_[c]
100            jllc = x_s.dot(flp_c) + lp_c
101            jlls.append(jllc)
102        return np.vstack(jlls).T
103
104    def predict(self, x_s):
105        log_ps = self.predict_log_proba(x_s)
106        return self.clss_[np.argmax(log_ps, axis=1)]
107
108 # ========================================
109 # 4.A STANDARD NAIVE BAYES (SNB) - BASELINE
110 # ========================================
111 print("\ncalculating snb baseline performance...")
112 kf = StratifiedKFold(n_splits=5, shuffle=True, random_state=0)
113 snb_vectorizer = CountVectorizer(
114     preprocessor=None,
115     tokenizer=lambda txt: txt.split(),
116     ngram_range=(1,1),
117     binary=False
118 )
119 x_snb_feats = snb_vectorizer.fit_transform(x_p)
120 print(f"snb features shape: {x_snb_feats.shape}")
121
122 snb_cv_accs = []
123 snb_clf = MyNB(alpha=1.0)
124 for trn_ixs, val_ixs in kf.split(x_snb_feats, y):
125     x_trn_fld, x_val_fld = x_snb_feats[trn_ixs], x_snb_feats[val_ixs]
```

```
    y_trn_fld, y_val_fld = y[trn_ixs], y[val_ixs]
    snb_clf.fit(x_trn_fld, y_trn_fld)
    preds = snb_clf.predict(x_val_fld)
    snb_cv_accs.append(accuracy_score(y_val_fld, preds))
mean_snb_cv_acc = np.mean(snb_cv_accs)
print(f"snb baseline: mean 5-fold cv accuracy = {mean_snb_cv_acc:.5f}")

# ============================
# 5. GRID-SEARCH W/ 5-FOLD CV
# ============================
print("\n grid search for improved model...")
a_opts   = [1e-7, 5e-7, 1e-6, 1.5e-6, 2e-6]
k_opts   = [15500, 16000, 16500, 17000, 17500]
mdf_opts = [0.50, 0.55, 0.60, 0.65, 0.70]
best_cfg = {'acc': 0.0, 'snb_acc': mean_snb_cv_acc}
p_cnt = 0
tot_combs = len(mdf_opts) * len(k_opts) * len(a_opts)
print(f"total param combos for improved model: {tot_combs}")
st_gs = time.time()

for mdf_v in mdf_opts:
    for k_v in k_opts:
        for a_v in a_opts:
            p_cnt += 1
            if p_cnt % 10 == 0 or p_cnt == 1 or p_cnt == tot_combs:
                print(f"[{p_cnt}/{tot_combs} testing cfg: max_df={mdf_v},
    k_chi2={k_v}, alpha={a_v:.1e}...]")

            curr_vec = TfidfVectorizer(
                preprocessor=None,
                tokenizer=lambda txt: txt.split(),
                sublinear_tf=True,
                norm='l2',
                ngram_range=(1,3),
                min_df=1,
                max_df=mdf_v
            )
            x_tfidf_f_trn = curr_vec.fit_transform(x_p)
            act_k = min(k_v, x_tfidf_f_trn.shape[1])
            chi2_scrs, _ = chi2(x_tfidf_f_trn, y)
            curr_top_ix = np.argsort(chi2_scrs)[-act_k:]
            x_sel_trn = x_tfidf_f_trn[:, curr_top_ix]

            cv_accs = []
            for trn_ixs, val_ixs in kf.split(x_sel_trn, y):
                x_trn_fld, x_val_fld = x_sel_trn[trn_ixs], x_sel_trn[val_ixs]
                y_trn_fld, y_val_fld = y[trn_ixs], y[val_ixs]
                clf = MyNB(alpha=a_v)
                clf.fit(x_trn_fld, y_trn_fld)
                preds = clf.predict(x_val_fld)
                cv_accs.append(accuracy_score(y_val_fld, preds))

            mean_cv_acc_imp = np.mean(cv_accs)
            if mean_cv_acc_imp > best_cfg['acc']:
                best_cfg['acc'] = mean_cv_acc_imp
                best_cfg['alpha'] = a_v
                best_cfg['k_chi2'] = act_k
                best_cfg['vectorizer_obj_params'] = {
                    'ngram_range': (1,3), 'min_df': 1, 'max_df': mdf_v,
                    'sublinear_tf': True, 'norm': 'l2'
                }
                print(f"  *** new best cfg ({p_cnt}): acc={mean_cv_acc_imp:.5f}
     *** "
```

```
                        f"(a={a_v:.1e}, k={act_k}, mdf={mdf_v})")

et_gs = time.time()
print(f"\ngrid search done in {(et_gs - st_gs)/60:.2f} mins.")
print(f"\n>>> SNB CV Accuracy: {best_cfg.get('snb_acc', 'N/A'):.5f}")
print(f">>> Best Overall IMPROVED CFG Params:\n{best_cfg}")

# ============================
# 6. FINAL TRAIN + SUBMISSION
# ============================
if 'acc' in best_cfg and best_cfg['acc'] > 0:
    print("\nfull data w/ best improved cfg + prep submission...")
    best_v_params = best_cfg['vectorizer_obj_params']
    f_vec = TfidfVectorizer(
        preprocessor=None, tokenizer=lambda txt: txt.split(),
        ngram_range=best_v_params['ngram_range'], min_df=best_v_params['min_df'],
        max_df=best_v_params['max_df'], sublinear_tf=best_v_params['sublinear_tf'],
        norm=best_v_params['norm']
    )
    x_trn_f_tfidf = f_vec.fit_transform(x_p)
    x_tst_p = test_df['Description'].map(preprocess)
    x_tst_tfidf = f_vec.transform(x_tst_p)

    num_features_final = min(best_cfg['k_chi2'], x_trn_f_tfidf.shape[1])
    final_chi2_scores, _ = chi2(x_trn_f_tfidf, y)
    final_top_indices = np.argsort(final_chi2_scores)[-num_features_final:]
    x_trn_f_sel = x_trn_f_tfidf[:, final_top_indices]
    x_tst_f_sel = x_tst_tfidf[:, final_top_indices]

    print(f"final train feats shape: {x_trn_f_sel.shape}")
    print(f"final test feats shape: {x_tst_f_sel.shape}")

    f_nb_clf = MyNB(alpha=best_cfg['alpha'])
    f_nb_clf.fit(x_trn_f_sel, y)
    f_preds = f_nb_clf.predict(x_tst_f_sel)

    sub_df = pd.DataFrame({'Id': test_df['Id'], 'Class': f_preds})
    sub_fname = 'submission.csv'
    sub_df.to_csv(sub_fname, index=False)
    print(f"\n !!!! submission done: {sub_fname} (cv acc for best improved cfg == {best_cfg['acc']:.5f})")
else:
    print("\nno valid cfg for improved model. submission failed")
\end{ganttchart}
```

Listing 1: Full Python Implementation

Execution Results and Console Output

The following is the console output from running the script.

Listing 2: Console Output

```
loading data...
preproc train data...
train data shape: (4400,), target shape: (4400,)

calculating snb baseline performance...
snb features shape: (4400, 5450)
snb baseline: mean 5-fold cv accuracy = 0.96273
```

```
 grid search for improved model...
total param combos for improved model: 125
[1/125 testing cfg: max_df=0.5, k_chi2=15500, alpha=1.0e-07...]
  *** new best cfg (1): acc=0.99045 *** (a=1.0e-07, k=15500, mdf=0.5)
[10/125 testing cfg: max_df=0.5, k_chi2=16000, alpha=2.0e-06...]
  *** new best cfg (10): acc=0.99045 *** (a=2.0e-06, k=16000, mdf=0.5)
  *** new best cfg (12): acc=0.99068 *** (a=5.0e-07, k=16500, mdf=0.5)
[20/125 testing cfg: max_df=0.5, k_chi2=17000, alpha=2.0e-06...]
[30/125 testing cfg: max_df=0.55, k_chi2=15500, alpha=2.0e-06...]
[40/125 testing cfg: max_df=0.55, k_chi2=16500, alpha=2.0e-06...]
  *** new best cfg (49): acc=0.99068 *** (a=1.5e-06, k=17500, mdf=0.55)
[50/125 testing cfg: max_df=0.55, k_chi2=17500, alpha=2.0e-06...]
[60/125 testing cfg: max_df=0.6, k_chi2=16000, alpha=2.0e-06...]
[70/125 testing cfg: max_df=0.6, k_chi2=17000, alpha=2.0e-06...]
[80/125 testing cfg: max_df=0.65, k_chi2=15500, alpha=2.0e-06...]
[90/125 testing cfg: max_df=0.65, k_chi2=16500, alpha=2.0e-06...]
[100/125 testing cfg: max_df=0.65, k_chi2=17500, alpha=2.0e-06...]
[110/125 testing cfg: max_df=0.7, k_chi2=16000, alpha=2.0e-06...]
[120/125 testing cfg: max_df=0.7, k_chi2=17000, alpha=2.0e-06...]
[125/125 testing cfg: max_df=0.7, k_chi2=17500, alpha=2.0e-06...]

grid search done in 9.18 mins.

>>> SNB CV Accuracy: 0.96273
>>> Best Overall IMPROVED CFG Params:
{'acc': 0.9906818181818183, 'snb_acc': 0.9627272727272727, 'alpha': 1.5e-06, '
    k_chi2': 17500, 'vectorizer_obj_params': {'ngram_range': (1, 3), 'min_df':
    1, 'max_df': 0.55, 'sublinear_tf': True, 'norm': 'l2'}}

full data w/ best improved cfg + prep submission...
preproc test data...
final train feats shape: (4400, 17500)
final test feats shape: (1100, 17500)

 !!!! submission done: submission.csv (cv acc for best improved cfg == 0.99068)
```

References

1. Kibriya, A. M., Frank, E., Pfahringer, B., & Holmes, G. (2004). *Multinomial naive bayes for text categorization revisited.*

2. AI tools were used for researching grammatical errors.