Machine Learning (CS60050) - Assignment 2 Report

Naive Bayes Classifier for Spooky Author Identification

Group 3

Vanshita Garg - 19CS10064

Ashutosh Kumar Singh - 19CS30008

Introduction and Tasks

In this report, we analyse a Naive Bayes Classifier trained on the Spooky Author Identification Dataset. The tasks performed with regard to this are:

1. Create a binary feature matrix to extract features from each example of the dataset.
2. Randomly split the feature matrix into a 70-30 train-test split.
3. Train a Naive Bayes classifier on the train split to predict the author from the text. Report the test accuracy on the test split.
4. Implement a Naive Bayes classifier using Laplace correction on the same train split, and report the test accuracy on the same test split.

Dataset

The Spooky Author dataset contains text from works of fiction written by spooky authors of the public domain: Edgar Allan Poe, HP Lovecraft and Mary Shelley. The dataset has the following data fields:

- **id** - a unique identifier for each excerpt/sentence
- **text** - an excerpt/sentence written by one of the authors
- **author** - the author of the sentence (EAP: Edgar Allan Poe, HPL: HP Lovecraft; MWS: Mary Wollstonecraft Shelley)

The dataset has 19,579 training examples and is available in the '.csv' format. The number of entries for each author is as follows:

Author	Count	Percentage (%)
EAP	7900	40.35
HPL	5635	28.78
MWS	6044	30.87

The Naive Bayes Classifier Algorithm

- Our objective is to learn a classifier f that maps sentences from a sentence space D to a fixed set of classes (authors) C, i.e., $f : D \rightarrow C$.
- For a sentence d, the Naive Bayes Classifier returns, out of all possible classes $c \in C$, the class $\hat{c}$ that has the maximum posterior probability for the given sentence, i.e., $\hat{c} = \underset{c \in C}{argmax}\ P(c|d)$.

- Using Bayes' Theorem, we can write $P(c|d) = \frac{P(d|c) \cdot P(c)}{P(d)}$, where $P(c)$ is the prior probability of the class c, and $P(d|c)$ is the likelihood probability of the sentence d, given the class c.
- So, $\hat{c} = \underset{c \in C}{argmax}\ P(c|d)$
  $= \underset{c \in C}{argmax}\ \frac{P(d|c) \cdot P(c)}{P(d)}$
  $= \underset{c \in C}{argmax}\ P(d|c) \cdot P(c)$

  We can write a sentence as a collection (or tuple) of words w_1, w_2, ..., w_n.
  Then, $\hat{c} = \underset{c \in C}{argmax}\ P(d|c) \cdot P(c)$
  $= \underset{c \in C}{argmax}\ P(w_1, w_2, ..., w_n|c) \cdot P(c)$
- However, calculation of the term $P(w_1, w_2, ..., w_n|c)$ is computationally difficult, so we make two assumptions.
  1. The first assumption, known as the Naive Bayes assumption, is that the words (or features) are conditionally independent given the class c, i.e., $P(w_1, w_2, ..., w_n|c) = \prod_{i=1}^{n} P(w_i|c)$.
  2. The second assumption or simplification, also known as the bag-of-words simplification, is that the probability of encountering a specific word w_i is independent of the specific word position being considered. In other words, we can say that the words only encode word identity and not position.
- Thus, the final prediction of a Naive Bayes Classifier can be written as
  $\hat{c} = \underset{c \in C}{argmax}\ P(c) \cdot \prod_{i=1}^{n} P(w_i|c)$

Some Important Terms and Definitions

Abbreviations: TP - True Positive, FP - False Positive, TN - True Negative, FN - False Negative.

- $Accuracy = \frac{No.\ of\ examples\ correctly\ classified}{Total\ no.\ of\ examples}$
- $Precision\ (Positive\ Predictive\ Value) = \frac{TP}{TP + FP}$
- $Sensitivity\ (Recall) = \frac{TP}{FN + TP}$
- $Specificity = \frac{TN}{FP + TN}$
- $Negative\ Predictive\ Value = \frac{TN}{TN + FN}$
- $F1\ Score = 2 \times \frac{Precision \times Recall}{Precision + Recall}$
- $Prevalence = \frac{TP + FN}{Total\ no.\ of\ examples}$

- $Detection\,Rate = \dfrac{TP}{Total\ no.\ of\ examples}$
- $Detection\,Prevalence = \dfrac{TP + FP}{Total\ no.\ of\ examples}$
- $Balanced\,Accuracy = \dfrac{Sensitivity + Specificity}{2}$

Procedure and Results

1. Data Preprocessing - Constructing the Binary Feature Matrix

In the given dataset, each excerpt/sentence is a raw text. So, we make a bag-of-words simplification to extract features (words) from each sentence. Every excerpt/sentence is represented as a bag-of-words, i.e., an unordered set of the words used within the sentence ignoring their position but keeping note of their occurrence.
The binary feature matrix M will be of the size $r \times c$, where r is the number of examples and c is the size of the vocabulary consisting of all distinct words present in the dataset.

Procedure :

- After reading the data from the csv file, the first step is to tokenize the sentences, i.e., split the sentences into lists of words. This is done using using the python `re` package.
- After splitting the texts into words, we remove all the uninformative words (referred to as stopwords) like articles, prepositions, verbs, etc.
- Then, we construct the binary matrix M such that $M_{ij} = 1$ if and only if the j-th word is present in the text of the i-th example. In the matrix, each row corresponds to an example of the dataset and each column corresponds to a word in the vocabulary.

Results :

Shape of the Binary Feature Matrix M	(19579, 24951)
Total Number of Examples	19579
Vocabulary Size	24951

2. 70-30 Train-Test Split

Procedure :

- Split the dataset into a train-test split in the ratio 70:30.

Results :

Number of Examples in the Training Set	13705
Number of Examples in the Test Set	5874

3. Naive Bayes Classifier (without Laplace Correction)

Procedure :

- We already have the binary feature matrix corresponding to the bag-of-words. Now, we need to train a Naive Bayes Classifier.
- Learning the prior probabilities - The prior probability of the class c is the percentage of documents in the training set that are in class c. Given N, the total number of training text sentences, and N_c, the total number of examples belonging to the class c in the training set, the prior probability $\hat{P}(c) = \frac{N_c}{N}$
- Learning the likelihood probabilities - The likelihood $\hat{P}(w_i|c)$ can be seen as the probability that the word w_i is found in the sentences belonging to the class c. Thus, $\hat{P}(w_i|c) = \frac{count(w_i, c)}{\sum\limits_{w \in V} count(w, c)}$, V is the entire vocabulary, $count(w_i, c)$ is the number of times the word w_i is used within all sentences belonging to the class c. Thus, $\hat{P}(w_i|c)$ is the fraction of times the word w_i appears among all words in sentences of class c.
- To predict the author for a test example, we calculate the posterior probability for each class c using $\hat{P}(c) \prod\limits_i \hat{P}(w_i|c)$. Our prediction is the class with maximum posterior probability, i.e., $c_{MAP} = \underset{c \in C}{argmax}\ (\hat{P}(c) \prod\limits_i \hat{P}(w_i|c))$

Results :

Confusion Matrix

Pred \ True	EAP	HPL	MWS
EAP	2140	1103	931
HPL	130	538	59
MWS	150	53	770

Statistics by Class

	EAP	HPL	MWS
Precision (Positive Predictive Value)	0.5127	0.7400	0.7914
Sensitivity (Recall)	0.8843	0.3176	0.4375
Specificity	0.4111	0.9548	0.9507
Negative Predictive Value	0.8353	0.7754	0.7980
F1-Score	0.6491	0.4444	0.5635
Prevalence	0.4120	0.2884	0.2996
Detection Rate	0.3643	0.0916	0.1311
Detection Prevalence	0.7106	0.1238	0.1656
Balanced Accuracy	0.6477	0.6362	0.6941

Average Statistics

Average Precision	0.6814
Average Recall (Sensitivity)	0.5465
Macro-Averaged F1-Score	0.6065

Overall Statistics

Accuracy	58.6993%
95% Confidence Interval	(57.4402%, 59.9585%)

4. Naive Bayes Classifier with Laplace Correction

Procedure :

- Learning the prior probabilities - This remains same as we had done previously. Given N, the total number of training text sentences, and N_c, the total number of examples belonging to the class c in the training set, the prior probability $\hat{P}(c) = \frac{N_c}{N}$
- Learning the likelihood probabilities - In the previous method, if we have not seen any training example with the word w_i belonging to the class c, then the likelihood $\hat{P}(w_i|c)$ becomes 0, as a result of which the posterior probability of that class also becomes 0, which provides inaccurate predictions and hampers the accuracy. To deal with this, we add 1 to each count. So now, the likelihood $\hat{P}(w_i|c) = \frac{count(w_i, c) + 1}{(\sum\limits_{w \in V} count(w, c)) + |V|}$, where $|V|$ is the vocabulary size, and $count(w_i, c)$ is the number of times the word w_i is used within all sentences belonging to the class c.
- To predict the author for a test example, we again calculate the posterior probability for each class c. Our prediction is the class with maximum posterior probability, i.e., $c_{MAP} = \underset{c \in C}{argmax}\ (\hat{P}(c) \prod\limits_{i} \hat{P}(w_i|c))$

Results :

Confusion Matrix

Pred \ True	EAP	HPL	MWS
EAP	2031	190	157
HPL	150	1381	66
MWS	239	123	1537

Statistics by Class

	EAP	HPL	MWS
Precision (Positive Predictive Value)	0.8541	0.8647	0.8094
Sensitivity (Recall)	0.8393	0.8152	0.8733
Specificity	0.8995	0.9483	0.9120
Negative Predictive Value	0.8887	0.9268	0.9439
F1-Score	0.8466	0.8393	0.8401
Prevalence	0.4120	0.2884	0.2996
Detection Rate	0.3458	0.2351	0.2617
Detection Prevalence	0.4048	0.2719	0.3233
Balanced Accuracy	0.8694	0.8818	0.8927

Average Statistics

Average Precision	0.8427
Average Recall (Sensitivity)	0.8426
Macro-Averaged F1-Score	0.8427

Overall Statistics

Accuracy	84.2526%
95% Confidence Interval	(83.3211%, 85.1841%)

Analysis and Important Points

- As expected, the accuracy in the case of the Naive Bayes classifier with Laplace correction (84.2526%) is higher than the accuracy without Laplace correction (58.6993%). It is so because without Laplace correction, if we have not seen any training example with the word w_i belonging to the class c, then the likelihood $\hat{P}(w_i|c)$ becomes 0, as a result of which the posterior probability of that class also becomes 0, which provides inaccurate predictions and thus, hampers the accuracy.

- The closer the F1-Score is to 1, the better is our classification model. For the Naive Bayes Classifier without Laplace correction the macro-averaged F1-Score comes out to be 0.6065, while with Laplace correction, it is 0.8427. Hence, we can again see the improvement on using Laplace correction.

- The confusion matrix also gives us a good idea of the number of examples correctly and incorrectly classified.

- One important point to note is that, while computing the probabilities, we have not used log-likelihood, instead we have used the likelihood values as it is, and multiplied them while computing the posterior probabilities. This works fine in our case as we do not face any underflow issue while doing so.
  An issue that arises when using log-likelihood is that in the case of a Naive Bayes Classifier without Laplace correction, many likelihood probabilities are 0, so while calculating the log-likelihood, we face errors because of trying to calculate the logarithm of 0. One way to handle this could have been replacing all zeros by a very small value like 10^{-10}, however, the problem with this is that replacing

zeros by some small value in itself introduces a kind of Laplace smoothing, or in other words, it's effect is very less as compared to the strong effect created by having 0 in the product. Hence, in our calculations we have used the likelihood probabilities as it is.

Major Classes and Functions Used in the Code

1. class `NaiveBayes`

- `fit` : Initializes other attributes for the naive Bayes classifier, and calls the train method.
- `train` : Trains the naive Bayes classifier.
- `predict_one` : Predicts the classification of a single sample on the basis of the naive Bayes classifier created.
- `predict` : Determines the predictions of the Naive Bayes classifier on a set of test data.

2. Data Processing

- `tokenize` : Tokenizes a string (sentence) into a list of words after removing the stopwords.
- `process_data` : Constructs the r $\times$ c binary feature matrix M where r is the number of examples and c is the size of the vocabulary consisting of distinct words present in the dataset.
- `train_test_split` : Shuffles the data and splits the dataset into a train-test split.

3. Utility Functions

- `accuracy` : Calculates the accuracy given the predictions and true labels.
- `precision` : Calculates the precision (positive predictive value) value based on predictions and true labels.
- `sensitivity` : Calculates the sensitivity (recall) value based on predictions and true labels.
- `specificity` : Calculates the specificity value based on predictions and true labels.
- `negative_predictive_value` : Calculates the negative predictive value value based on predictions and true labels.
- `f1_score` : Calculates the F1-score based on predictions and true labels. It is the harmonic mean of the precision and recall.
- `prevalence` : Calculates the prevalence value based on predictions and true labels. Prevalence is the proportion of all positives in the total number of observations.
- `detection_rate` : Calculates the detection rate value based on predictions and true labels. Detection rate is the proportion of true positives in the total number of observations.
- `detection_prevalence` : Calculates the detection prevalence value based on predictions and true labels. Detection prevalence is the number of positive class predictions made as a proportion of all predictions.
- `balanced_accuracy` : Calculates the balanced accuracy value based on predictions and true labels. Balanced accuracy is the average of the sensitivity and the specificity.

- `confusion_matrix` : Calculates and returns the confusion matrix.
- `ci_95` : Calculates the 95% confidence interval of accuracy.
- `display_metrics` : Displays all the required metrics in a formatted manner.

Files Present

- `src` directory:
  - `train.csv` - Contains the data file for training and testing the Naive Bayes Classifier.
  - `requirements.txt` - Contains all the necessary dependencies with their versions.
  - `data_processing.py` - Contains necessary functions to process the raw textual data into tokenized words and create the binary feature matrix.
  - `naive_bayes.py` - Contains the model for the Naive Bayes Classifier.
  - `metrics.py` - Contains the functions to evaluate the performance of the model.
  - `utils.py` - Contains all helper functions required.
  - `main.py` - Main file for completing all tasks required.
- `output.txt` - The output for a run of the Naive Bayes Classifier.
- `ML_Assn_2_Report.pdf` - A report contain the step-wise description of the implementation and analysis of results.
- `README.md` - A README file describing the files present and the instructions to execute the code.

Instructions to Execute the Code

- Navigate to the `src` directory.
- Ensure you are using a latest version of Python3, and install all dependencies.
  `pip install -r requirements.txt`
- Execute the file `main.py`
  `python main.py`
- The relevant output will be displayed on the terminal or console.