Report - Naive Bayes

Description

This program is an implementation of a Bayesian classifier written in Python. Utilizing a data file, it classifies any record cases using the Bayesian classifier. It iterates through 10 cycles, continuously calculating the accuracy of the algorithm. In the end, it computes the average accuracy and standard deviation.

Files

The 'files' folder is used to store all necessary data. To run the program, place the data file in the 'files' folder. As a result of the program's execution, this folder is additionally filled with 5 files:

- **after.csv** - This file is formatted as .csv for data visualization using Pandas and Matplotlib. It contains classified data, including potentially incorrect data. It is only created when the show_graph() function is called.

- **before.csv** - This file is formatted as .csv for data visualization using Pandas and Matplotlib. It contains data before classifying them. It is only created when the show_graph() function is called.

- **cars_evaluation.trn** - This file is a part of the original data. It contains 70% of the data and is intended for training.

- **cars_evaluation.tst** - This file is a part of the original data. It contains 30% of the data and is intended for testing.

- **cars_evaluation_format.tst** - This file is a part of the original data and contains 30% of the data intended for classification. In practice, this means their classification has been removed.

Key functions

def show_graph():

This function visualizes original and freshly classified data in the form of two graphs.

def handle_files():

This function manages the original data file, separating it into two files: testing and training, in a 70/30 ratio. It also creates an additional file to undergo classification for later comparison.

def calculate_probability(car, keyword):

- car - a single data record
- keyword - class (unacc, acc, good, vgood)

This function calculates the probability of belonging to a specific class based on a given data record and classifier. Laplace's transformation is implemented to eliminate cases where the numerator/denominator is zero. Three functions are used for counting individual annotations, differing only in the condition in the if statement.

def check_both(w1, w2, i):

```
if w1 in c.split(',')[i] and w2 in
c.split(',')[6]:
```

def count_specific_word(w):

```
if w in c.split(',')[i]:
```

def count_word(w):

```
if w in c:
```

Results

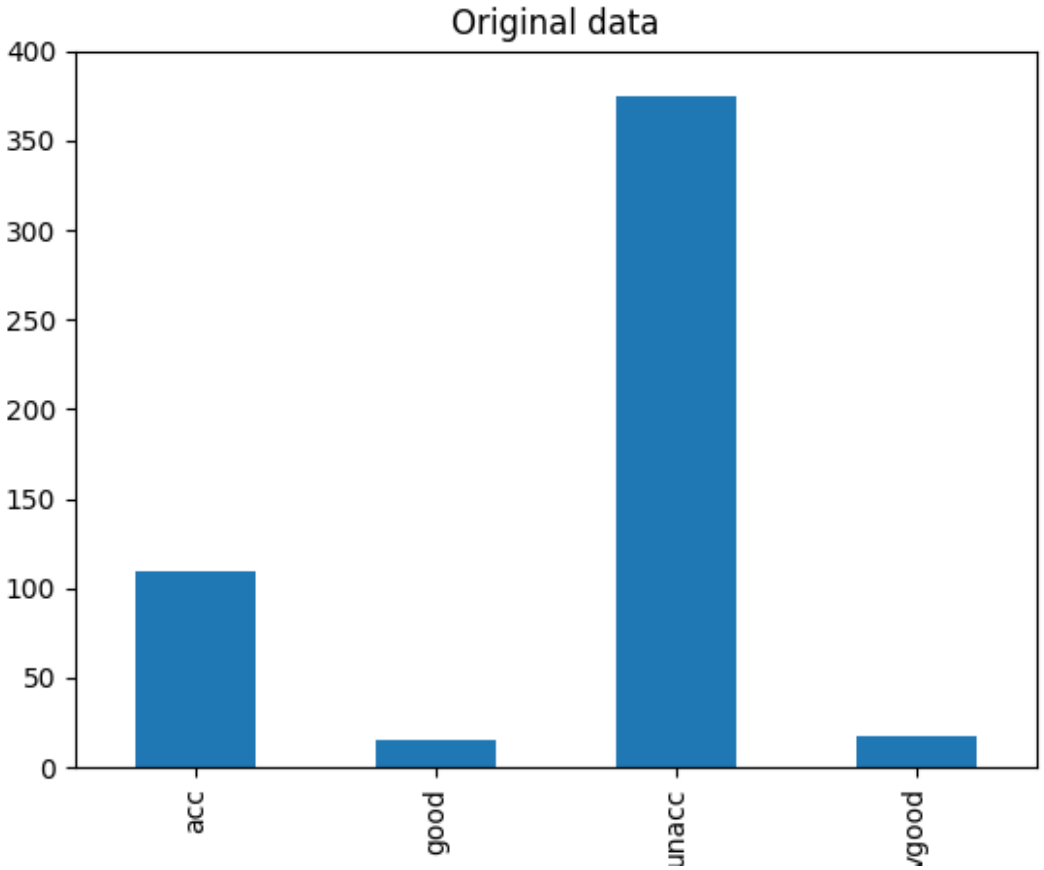
After 10 cycles of data randomization and classification, the results are as follows:

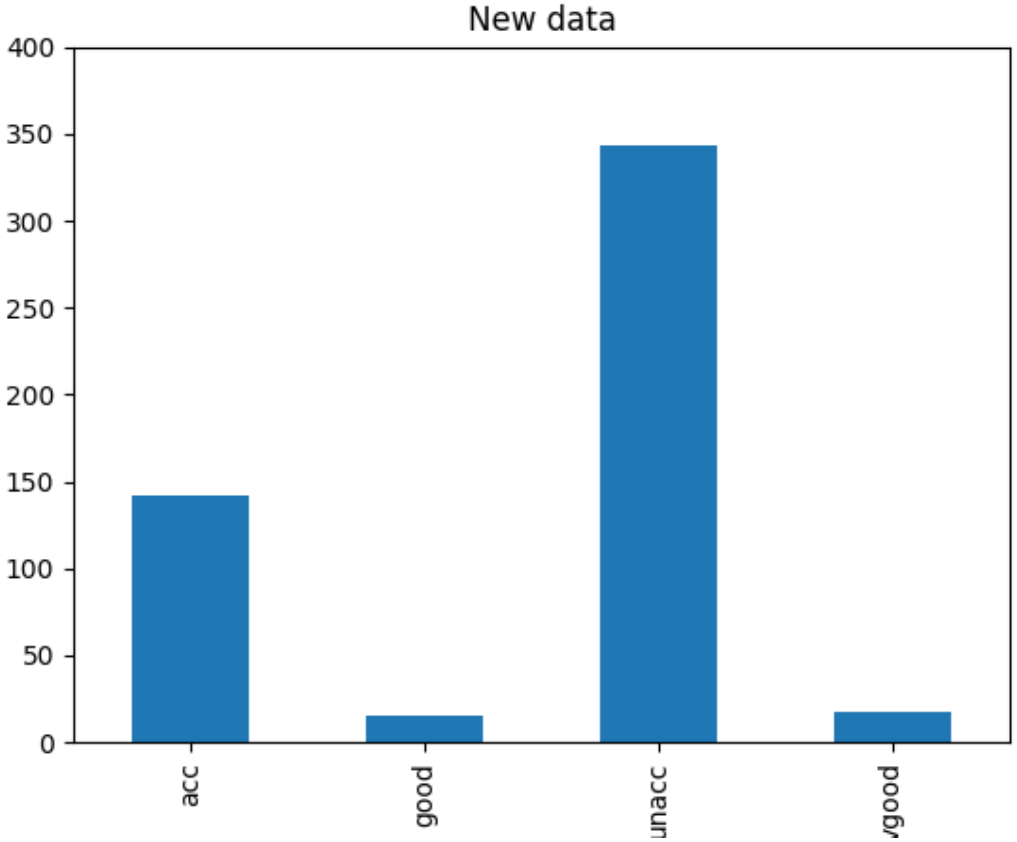
- Cycle 0 - Accuracy 91%
- Cycle 1 - Accuracy 91%
- Cycle 2 - Accuracy 93%
- Cycle 3 - Accuracy 92%
- Cycle 4 - Accuracy 92%
- Cycle 5 - Accuracy 95%
- Cycle 6 - Accuracy 93%
- Cycle 7 - Accuracy 92%
- Cycle 8 - Accuracy 92%
- Cycle 9 - Accuracy 91%

Average over 10 cycles: 92%
Standard deviation: 1.18321

In conclusion, the results range from 91-95%, indicating a low standard deviation, meaning the data is concentrated around the average.

Below are two graphs where the classification accuracywas 93%.


Original data
400
350
300
250
200
150
100
50
0
acc
good
unacc
vgood


New data
400
350
300
250
200
150
100
50
0
acc
good
unacc
vgood