Final Project DATA 2060

Machine Learning, from theory to algorithms

Edouard Clocheret, Junhui Huang, Eric Fan and Shivam Hingorani

Link to the github repository:
https://github.com/edouardclocheret/GaussianNaiveBayes.git

PART 1: Overview of Gaussian Naive Bayes for classification

OVERVIEW

Gaussian Naive Bayes is an extension of the conventional Naive Bayes algorithm. Like the original, the Gaussian variety of Naive Bayes is used for classification tasks, specifically those where the features are continuous and can be assumed to follow a Gaussian (i.e., normal) distribution. This means Gaussian Naive Bayes maintains the assumption of conditional independence across features (hence the description as "naive"). This algorithm is computationally efficient, which can be advantageous when using large datasets,and it serves as the basis for many more complex machine learning models/techniques.

REPRESENTATION

Multiclass Naive bayes Representation:

For a given input: $X = (x_1, x_2, \ldots, x_n)$,

First, we calcualte the prior probability for each class,

which is defined as the probability of each class based on the training data. This is donated as

$$P(C_k) = \frac{\text{Number of samples in class } C_k}{\text{Total number of samples}} = \frac{N_k}{N}$$

Second, For each class, we calculate the likelihood $P(X|C_k)$,

which is the proabability of observing feature X given that the data point belongs to a specific Class C_k

To calculate this, we use the Naive Bayes assumption that each feature x_i is independent of others given the class C. This allows us to calculate the product of the individual feature likelihoods:

$$P(X|C_k) = \prod_{i=1}^{n} P(x_i|C_k)$$

Third, We calculate the posterior Probability $P(C_k|X)$ for each class,

which is the goal of this algorithim, calculating the probability of each class C based on the feature X. Since $P(X)$ is the same for all classes, we can ignore it when comparing classes, so we don't need to calculate it explicitly.

$$P(C_k|X) \propto P(C_k) \cdot \prod_{i=1}^{n} P(x_i|C_k)$$

where

$$P(x_i|C_k) = \frac{1}{\sqrt{2\pi\sigma_k^2}} \exp\left(-\frac{(x_i - \mu_k)^2}{2\sigma_k^2}\right)$$

Last, We select the class C with the highest posterior probability,

which means that we After computing $P(C_k \mid X)$ for each class C_k , choose the class with the highest posterior probability as the predicted class:

$$C = \arg\max_{C_k} P(C_k \mid X) = \arg\max_{C_k} P(C_k) \cdot \prod_{i=1}^{n} P(x_i|C_k)$$

LOSS

For Gaussian Naive Bayes (GNB), the loss function normally not defined or refined. Because this type of model deduct based on **Maximum Likelihood Estimation** (MLE).

However, if we have to define a loss function when during the Training or Evaluation Process, we can define *Negative Log Likelihood* (NLL) to measure the classification loss.

The formula:

$$\text{NLL} = -\sum_{i=1}^{N} \log P(y_i|X_i)$$

y_i is the i th sample's true label, X_i is input sample feature, $P(y_i|X_i)$ is the probability of model predict.

Since Gaussian Naive Bayes assumes that features follow a Gaussian (normal) distribution, it calculates the conditional probability using the probability density function of a Gaussian distribution. For each class y and feature x_i, the conditional probability (P(x_i | y)) is given by:

$$P(x_i|y) = \frac{1}{\sqrt{2\pi\sigma_y^2}} \exp\left(-\frac{(x_i - \mu_y)^2}{2\sigma_y^2}\right)$$

Such that the negative log likelihood for Gaussian Naive Bayes is:

$$\text{NLL}_{\text{Gaussian}} = -\sum_{i=1}^{N} \left(\log P(y_i) + \sum_{j=1}^{d} \log \frac{1}{\sqrt{2\pi\sigma_{y_i,j}^2}} - \frac{(X_{ij} - \mu_{y_i,j})^2}{2\sigma_{y_i,j}^2} \right)$$

$\mu_{y_i,j}$ and $\sigma_{y_i,j}$ are sample j 's mean and variance under label y_i

Maximum Likelihood Estimation

Suppose we have a set of data points $X = \{x_1, x_2, \ldots, x_n\}$, and these data points are drawn from a normal distribution with mean μ and variance σ^2. We want to know μ and σ^2 by using MLE.

The PDF of the normal distribution is:

$$P(x|\mu, \sigma^2) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{(x - \mu)^2}{2\sigma^2}\right)$$

For a given dataset $X = \{x_1, x_2, \ldots, x_n\}$, the joint likelihood function is:

$$L(\mu, \sigma^2) = \prod_{i=1}^{n} P(x_i|\mu, \sigma^2)$$

Taking the log of the likelihood function, we get the log-likelihood function:

$$\log L(\mu, \sigma^2) = -\frac{n}{2}\log(2\pi\sigma^2) - \frac{1}{2\sigma^2}\sum_{i=1}^{n}(x_i - \mu)^2$$

Taking the derivative with respect to μ and σ^2 and setting them to zero, we obtain:

- The MLE for the mean:

$$\hat{\mu} = \frac{1}{n}\sum_{i=1}^{n} x_i$$

- The MLE for the variance:

$$\hat{\sigma}^2 = \frac{1}{n}\sum_{i=1}^{n}(x_i - \hat{\mu})^2$$

OPTIMIZER

- Optimizer: describe the numerical algorithm used to find the model parameters that minimize the loss for a training set. (Edouard)

The Gaussian assumption of the model is to consider the probabilities $P(x_i|C_k)$ to follow a Gaussian distribution:

$$P(x_i|C_k) = \frac{1}{\sqrt{2\pi\sigma_C^2}}\exp\left(-\frac{(x_i - \mu_C)^2}{2\sigma_C^2}\right)$$

The mean μ and variance σ estimators are derived from maximum likelihood estimation (MLE).

We use the log-loss because it is easier to calculate derivatives of a sum:

$$\text{LogLoss}(C, X) = -\log P(C) - \sum_{i=1}^{n} \log\left(\frac{1}{\sqrt{2\pi\sigma_{C,i}^2}}\exp\left(-\frac{(x_i - \mu_{C,i})^2}{2\sigma_{C,i}^2}\right)\right)$$

and simplifies as :

$$\text{LogLoss}(C, X) = -\log P(C) + \sum_{i=1}^{n}\left(\frac{(x_i - \mu_{C,i})^2}{2\sigma_{C,i}^2} + \log\sqrt{2\pi\sigma_{C,i}^2}\right)$$

We maximize the likelihood (= minimize the loss) by taking the logarithm (log-likelihood) and setting the derivatives with respect to $\mu_{C,i}$ and $\sigma_{C,i}$ to zero, we obtain:

$$\mu_{C_k,i} = \frac{1}{N_C}\sum_{j=1}^{N_C} x_i^{(j)}$$

and

$$\sigma_{C_k,i}^2 = \frac{1}{N_C}\sum_{j=1}^{N_C}\left(x_i^{(j)} - \mu_{C_k,i}\right)^2$$

Where

- N_C is the number of samples in class C.
- $x_i^{(j)}$ is the value of feature x_i for the j-th sample in class C.

In practice, we first separate the data by class and use the optimal estimators above for the mean and variance.

The training consists of computing and memorizing $\mu_{C_k,i}$ and $\sigma^2_{C_k,i}$ for all feature x_i and classes C_k.

These values are then used for the classifications to compute $P(x_i|C_k)$.

REFERENCES

1. Wikipedia, Naive Bayes Classifier https://en.wikipedia.org/wiki/Naive_Bayes_classifier
2. Geeks for geeks, Gaussian Naive Bayes, https://www.geeksforgeeks.org/gaussian-naive-bayes/
3. Brown DATA2060, Assignment 6
4. Scikit learn documentation, Naive Bayes, https://scikit-learn.org/1.5/modules/naive_bayes.html
5. Iqbal, Z., & Yadav, M. (2020). Multiclass Classification with Iris Dataset Using Gaussian Naive Bayes. International Journal of Computer Science and Mobile Computing, 9(4), 27–35. Retrieved from www.ijcsmc.com 6.UCI Machin e Learning Repository Iris Data Set.

PART 2: Model

This section is one code cell which contains the class of your ML algorithm and any other helper functions. Add headers to each method and explain to the viewer what each method does and what the inputs and outputs are. Please add comments to the code as well!

In [ ]:
```
import numpy as np
import pandas as pd

def gaussian(x, mu, sigma2):

    sigma2 = np.where(sigma2 == 0, 1e-6, sigma2)#change by GF
    coeff = 1 / np.sqrt(2 * np.pi * sigma2)
    exponent = np.exp(-((x - mu) ** 2) / (2 * sigma2))

    return coeff * exponent

class GaussNaiveBayes :
```

```
    """ Gaussian Naive Bayes model for multiclass classification

    @attrs:
        n_classes:    the number of classes
        feature_dist:    a 3D (n_classes x n_features x 2) NumPy array of
                         the attribute distributions: mean and sigma2 for
                         each feature for each class

                        ex : [[[mu, sigma2],   of feature 1
                               [mu, sigma2],   of feature 2
                               [mu, sigma2]],  of feature 3 of class 1

                               [[mu, sigma2],  of feature 1
                               [mu, sigma2],   of feature 2
                               [mu, sigma2]]]  of feature 3 of class 2

        priors: a 1D NumPy array of the priors distribution

                        ex : [p_class1, p_class2, p_class3]
    """
    def __init__ (self):
        self.n_classes = None # computed at training
        self.feature_dist = None
        self.priors = None

        self.laplace_smoothing = 1 # Default value for the smoothing paramet


    def train(self, X_train, y_train):
        """ Trains the model, using maximum likelihood estimation.
        @params:
            X_train: a 2D (n_examples x n_features) numpy array
            y_train: a 1D (n_examples) numpy array of the corresponding labe
        @return:
            self.priors
            self.feature_dist
        """#change by GF
        self.n_classes = len(set(y_train))
        n_features = X_train.shape[1]

        self.priors = np.zeros(self.n_classes)
        self.feature_dist = np.zeros((self.n_classes, n_features, 2))

        n_examples = X_train.shape[0]

        for class_label in range(self.n_classes):
            X_class = X_train [y_train == class_label]

            # Computing prior with Laplace smoothing
            total_class = X_class.shape[0]
            a = self.laplace_smoothing
            self.priors[class_label] = (total_class +a) /(n_examples +a * se

            # Computing the moments (mean mu and variance sigma2)
            mu = np.mean(X_class, axis=0)
            sigma2 = np.var(X_class, axis = 0) + 1e-6
```

```
            # saving the mu and sigma for later predictions
            self.feature_dist [class_label, :, :] = np.vstack((mu , sigma2))
        return self.priors, self.feature_dist


    def predict(self, inputs):
        """ Outputs a predicted label for each input in inputs.

        @params:
            inputs: a 2D NumPy array containing inputs (n_requests * n_featu
        @return:
            a 1D numpy array of predictions
        """

        predictions = np.zeros(len(inputs))
        for index, input in enumerate(inputs) :

            logprobs = np.log(self.priors)

            for class_label in range(self.n_classes):
                mu = self.feature_dist[class_label, :, 0] # All mus for each
                sigma2 = self.feature_dist[class_label, :, 1]

                logprobs[class_label] += np.sum(np.log(gaussian(input, mu, s

            predictions[index] = np.argmax(logprobs)

        return predictions


    def accuracy(self, X_test, y_test):
        """ Outputs the accuracy of the trained model on a given dataset (da

        @params:
            X_test: a 2D numpy array of examples
            y_test: a 1D numpy array of labels
        @return:
            a float number indicating accuracy (between 0 and 1)
        """

        y_pred = self.predict(X_test)
        return np.mean(y_pred == y_test)
```

PART 3: Check Model

This section is a collection of code and markdown cells that contain the unit tests and a demonstration that your implementation can reproduce previous results.

There should be several unit tests. Create at least two or three unit tests per method and make sure that edge cases are properly handled. Explain either as comments or in a markdown cell what the goal of each test is and/or what edge case it tests for.

Find at least one previous work where your ML algorithm was applied on a public dataset. You can use, for example, the sklearn manual, a textbook, or a peer-reviewed publication. Include a markdown cell and describe the previous work. Demonstrate in a code cell that your implementation can successfully reproduce the previous work. Use citations and references.

In [ ]:
```
#Unit tests
import pytest

test_model1 = GaussNaiveBayes()

x1 = np.array([[0,0,1], [0,1,0], [1,0,1], [1,1,1], [0,0,1]])
y1 = np.array([0,0,1,1,0])
x_test1 = np.array([[1,0,0],[0,0,0],[1,1,1],[0,1,0], [1,1,0]])
y_test1 = np.array([0,0,1,0,1])

x2 = np.array([[0,0,1], [0,1,1], [1,1,1], [1,1,1], [0,0,0], [1,1,0]])
y2 = np.array([0,1,1,1,0,1])
x_test2 = np.array([[0,0,1], [0,1,1], [1,1,1], [1,0,0]])
y_test2 = np.array([0,1,1,0])

pri1, feature1 = test_model1.train(x1,y1)
assert (np.array(pri1) == pytest.approx(np.array([0.571,0.428]),0.01))# chec
feature1 = feature1.flatten()
cmp = np.array([[[0.00, 0.00], [0.333, 0.222], [0.667, 0.222]], [[1.00, 0.00
assert (feature1 == pytest.approx(cmp, abs = 0.01))# check if the train func

assert test_model1.accuracy(x1,y1) == 1. # check the accuracy function
assert test_model1.accuracy(x2,y2) >= .8
print("test complete")
```

```
test complete
```

PART 3.5: Real dataset

In [ ]:
```
#Load and preprocess data
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import accuracy_score

'''
The data set consists of 50 samples from each of three species of Iris (Iris
Four features were measured from each sample: the length and the width of th
'''
iris = load_iris()
X = iris.data
y = iris.target

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, rar
```

In [ ]:
```
#Train-test on real data
model1 = GaussNaiveBayes()
model1.train(X_train, y_train)
print("-------------------------------------------------------------")
print("Train accuracy:")
m1train = model1.accuracy(X_train, y_train)
print(m1train)#change by GF
print("-------------------------------------------------------------")
print("Test accuracy:")
m1test = model1.accuracy(X_test, y_test)
print(m1test)#change by GF
print("-------------------------------------------------------------")
```

```
-------------------------------------------------------------
Train accuracy:
0.95
-------------------------------------------------------------
Test accuracy:
1.0
-------------------------------------------------------------
```

In [ ]:
```
#Comparison with scikit-learn
'''
Use the sklearn model to test whether our model is correct or not.
'''
model = GaussianNB()
model.fit(X_train, y_train)

train_accuracy = accuracy_score(y_train, model.predict(X_train))
test_accuracy = accuracy_score(y_test, model.predict(X_test))

if train_accuracy == m1train and test_accuracy == m1test:
    print("Mission Complete")
```

```
Mission Complete
```

Description of previous work and comparision between our model.

In the previous work I found, which was a journel paper from International Journal of Computer Science and Mobile Computin. They apply the Gaussian Naive Bayes algorithm to classify the Iris dataset into its three species: Iris Setosa, Iris Versicolor, and Iris Virginica.

The dataset consists of 150 samples, with 50 samples per species, and includes four features for classification: sepal length, sepal width, petal length, and petal width. The goal is to utilize these features to predict the class of a flower. They also used scatter plot matrix as part of their EDA to study the relationships between pairs of features.

In their results, they evaluated the performance of the Gaussian Naive Bayes classifier using metrics such as precision (0.95), recall (0.95), F1-score (0.95), and accuracy

(0.95). Additionally, they used confusion matrix to explain the performance of the Gaussian Naive Bayes classifier.

The results from our algorithm closely match their findings and are also consistent with the outcomes produced by the built-in scikit-learn implementation.

PART 4: Contributions

One paragraph per team member and describe in detail what that team member's contribution is to the final project.

Edouard - Coded the core of the model (Train, predict, accuracy) and worked on the theory and likelyhood maximisation.

Eric- Coded the unit tests and worked on the theory (loss and optimizer).

Huang - Found previous work where Naive Bayes is applied on Iris dataset. Discusssion of previous work, and reproduction of the previous work. Introducted the represnetation of the Gaussian Naive Bayes.

Shivam - Worked on the previous work, and result analysis, limitations of the Gaussian Naive Bayes Model.

PART 5: Github repo

Link to the github repository:
https://github.com/edouardclocheret/GaussianNaiveBayes.git

PART 6: References

Collect all references in this section. Use the Harvard citation format.

1. Wikipedia, 2023. Naive Bayes Classifier. [online] Available at: https://en.wikipedia.org/wiki/Naive_Bayes_classifier [Accessed 8 Dec. 2024].
2. GeeksforGeeks, 2023. Gaussian Naive Bayes. [online] Available at: https://www.geeksforgeeks.org/gaussian-naive-bayes/ [Accessed 8 Dec. 2024].
3. Brown University, n.d. DATA2060, Assignment 6. [online] Brown University. Available at: [Accessed 8 Dec. 2024].
4. Scikit-learn, 2024. Naive Bayes. [online] Available at: https://scikit-learn.org/1.5/modules/naive_bayes.html [Accessed 8 Dec. 2024].

5. Iqbal, Z. and Yadav, M., 2020. Multiclass Classification with Iris Dataset Using Gaussian Naive Bayes. International Journal of Computer Science and Mobile Computing, 9(4), pp.27–35. Available at: Link [Accessed 8 Dec. 2024].

6. UCI Machine Learning Repository, n.d. Iris Data Set. [online] Available at: https://archive.ics.uci.edu/ml/datasets/Iris [Accessed 8 Dec. 2024].

Appendice: Student performance Dataset

In [ ]:
```python
#Import kaggle dataset and show training and testing error.
from google.colab import drive
import pandas as pd
from sklearn.model_selection import train_test_split, cross_val_score
from sklearn.naive_bayes import GaussianNB
from sklearn.pipeline import Pipeline
from sklearn.compose import ColumnTransformer
from sklearn.preprocessing import OrdinalEncoder, OneHotEncoder, MinMaxScale
drive.mount('/content/drive')
file_path = '/content/drive/My Drive/Student_performance_data.csv'
df = pd.read_csv(file_path)
y = df['GradeClass']
X = df.drop(columns=['GradeClass', 'StudentID'])

minmax_ftrs = ['Age', 'StudyTimeWeekly', 'Absences']
std_ftrs = ['GPA']

preprocessor = ColumnTransformer(
    transformers=[
        ('minmax', MinMaxScaler(), minmax_ftrs),
        ('std', StandardScaler(), std_ftrs)
    ]
)

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.1, rar
pipeline = Pipeline([
    ('preprocessor', preprocessor),
    ('model', GaussianNB())
])

pipeline.fit(X_train, y_train)

train_accuracy = pipeline.score(X_train, y_train)
test_accuracy = pipeline.score(X_test, y_test)

print("-----------------------------------------------------------")
print(f"Train Accuracy: {train_accuracy}")

print("-----------------------------------------------------------")
print(f"Test Accuracy: {test_accuracy}")

print("-----------------------------------------------------------")
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, cal
l drive.mount("/content/drive", force_remount=True).
------------------------------------------------------------
Train Accuracy: 0.7955390334572491
------------------------------------------------------------
Test Accuracy: 0.8
------------------------------------------------------------
```

In [ ]:

```python
from google.colab import drive
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler, LabelEncoder
import numpy as np

# Mount Google Drive
drive.mount('/content/drive')
file_path = '/content/drive/My Drive/Student_performance_data.csv'
df = pd.read_csv(file_path)
y = df['GradeClass']
X = df.drop(columns=['GradeClass', 'StudentID'])

# Standardize numerical features
numeric_features = ['Age', 'StudyTimeWeekly', 'Absences', 'GPA']
scaler = StandardScaler()
X[numeric_features] = scaler.fit_transform(X[numeric_features])

label_encoder = LabelEncoder()
y_encoded = label_encoder.fit_transform(y)

X_train, X_test, y_train, y_test = train_test_split(
    X, y_encoded, test_size=0.1, random_state=42, stratify=y_encoded
)

X_train_np = X_train.values
X_test_np = X_test.values
y_train_np = np.array(y_train)
y_test_np = np.array(y_test)

# Initialize and train the GaussNaiveBayes model
model1 = GaussNaiveBayes()
model1.train(X_train_np, y_train_np)

# Calculate train and test accuracies
print("------------------------------------------------------------")
m1train = model1.accuracy(X_train_np, y_train_np)
print(f"Test Accuracy: {m1train}")
print("------------------------------------------------------------")
print(f"Test Accuracy: {m1test}")
print("------------------------------------------------------------")
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, cal
l drive.mount("/content/drive", force_remount=True).
------------------------------------------------------------
Test Accuracy: 0.7978624535315985
------------------------------------------------------------
Test Accuracy: 0.7708333333333334
------------------------------------------------------------
```