STATISTICAL LEARNING FROM DATA: APPLICATIONS IN PHYSICS

GAUSSIAN NAIVE BAYES

CONTENTS

1.Theory

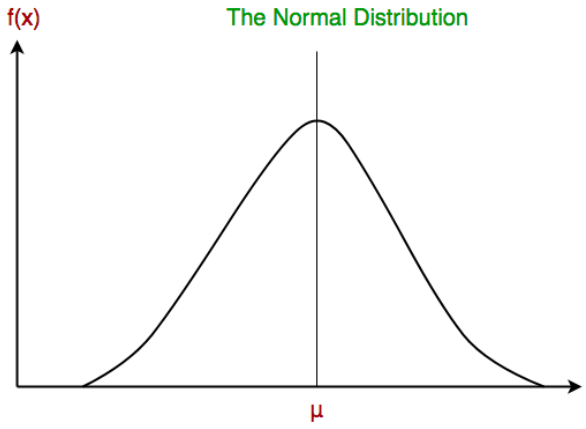
1.1 Bayes' Theorem

Bayes' Theorem finds the probability of an event occurring given the probability of another event that has already occurred. Bayes' theorem is stated mathematically as the following equation:

$$P(A|B) = \frac{P(B|A)P(A)}{P(B)}$$

1.2 GAUSSIAN NAIVE BAYES CLASSIFIER

Gaussian Naive Bayes is a machine learning classification technique based on a probabilistic approach that assumes each class follows a normal distribution. It assumes each parameter has an independent capacity of predicting the output variable. It is able to predict the probability of a dependent variable to be classified in each group.The combination of the prediction for all parameters is the final prediction that returns a probability of the dependent variable to be classified in each group. The final classification is assigned to the group with the higher probability.



The likelihood of the features is assumed to be Gaussian, hence, conditional probability is given by:

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

2. Homework 3

2.1 Application

```
import numpy as np



class GaussianNB:



   def __init__(self, priors=None, var_smoothing=1e-9):
       self.priors = priors
       self.var_smoothing = var_smoothing

   def logprior(self, class_ind):
       return np.log(self.class_priors_[class_ind])

   def loglikelihood(self, Xi, class_ind):
       # mu: mean, var: variance, Xi: sample (a row of X)
       # Get the class mean
       mu = self.theta_[class_ind, :]
       # Get the class variance
       var = self.var_[class_ind, :] + self.var_smoothing
       # Write the Gaussian Likelihood expression
       GaussLikelihood = (1 / np.sqrt(2 * np.pi * var)) * np.exp(-0.5 * ((Xi - mu)
** 2) / var)
       # Take the log of GaussLikelihood
       logGaussLikelihood = np.log(GaussLikelihood)
       # Return loglikelihood of the sample. Now you will use the "naive"
       # part of the naive bayes.
       return np.sum(logGaussLikelihood)

   def posterior(self, Xi, class_ind):
       logprior = self.logprior(class_ind)
```

```
        loglikelihood = self.loglikelihood(Xi, class_ind)
        # Return posterior
        return logprior + loglikelihood

    def fit(self, X, y):
        # Number of samples, number of features
        n_samples, n_features = X.shape
        # Get the unique classes
        self.classes_ = np.unique(y)
        # Number of classes
        n_classes = len(self.classes_)

        # Initialize attributes for each class
        # Feature means for each class, shape (n_classes, n_features)
        self.theta_ = np.zeros((n_classes, n_features))
        # Feature variances for each class shape (n_classes, n_features)
        self.var_ = np.zeros((n_classes, n_features))
        # Class priors shape (n_classes,)
        self.class_priors_ = np.zeros(n_classes)
        # Calculate class means, variances and priors
        for c_ind, c_id in enumerate(self.classes_):
            # Get the samples that belong to class c_id
            X_class = X[y == c_id]
            # Mean of each feature that belongs to class c_id
            self.theta_[c_ind, :] = np.mean(X_class, axis=0)
            # Calculate the variance of each feature that belongs to c_id
            self.var_[c_ind, :] = np.var(X_class, axis=0)
            # Calculate the priors for each class
            self.class_priors_[c_ind] = X_class.size / float(n_samples)

    def predict(self, X):
        y_pred = []
        for Xi in X: # Calculate posteriors for each sample
            posteriors = [] # For saving posterior values for each class
            # Calculate posterior probability for each class
            for class_ind in self.classes_:
                # Calculate posterior
                sample_posterior = self.posterior(Xi, class_ind)
                # Append the posterior value of this class to posteriors
                posteriors.append(sample_posterior)
            # Get the class that has the highest posterior prob. and
            # append the prediction for this sample to y_pred
            y_pred.append(self.classes_[np.argmax(posteriors)])
        return y_pred # Return predictions for all samples
```

2.2 Summary

Our code defines a Gaussian Naive Bayes classifier and It includes methods for initializing the classifier, calculating logarithmic prior and likelihood probabilities for Gaussian distributions, determining posterior probabilities, fitting the model to training data, and making predictions. The implementation assumes conditional independence of features given the class and is suitable for continuous data. A variance smoothing parameter is incorporated to prevent division by zero in the likelihood calculation.

3. Advantages vs Disadvantages of Naive Bayes Classifier

3.1 Advantages of Naive Bayes Classifier

- Easy to implement and computationally efficient.
- Effective in cases with a large number of features.
- Performs well even with limited training data.

3.2 Disadvantages of Naive Bayes Classifier

- Assumes that features are independent, which may not always hold in real-world data.
- Can be influenced by irrelevant attributes.
- May assign zero probability to unseen events, leading to poor generalization.