Gaussian Naive Bayes

Gaussian Naive Bayes is useful when working with continuous values whose probabilities can be modeled using Gaussian distributions whose means and variances are associated with each specific class (in this case, let's suppose *j=1,2, ... P*):

$$p(x_i|y_j) = \frac{1}{\sqrt{2\pi\sigma_j^2}} e^{-\frac{(x_i-\mu_j)^2}{2\sigma_j^2}}$$

Our goal is to estimate the mean and variance of each conditional distribution, using the maximum likelihood approach, which is quite easy, considering the mathematical nature of a Gaussian distribution. The likelihood for the whole dataset is as shown:

$$L(\mu, \sigma^2; X|Y) = log \prod_i p(x_i|y_i) = \sum_i log\, p(x_i|y_i)$$

Now, expanding the last term, we obtain the following expression (to avoid confusion, as the mean and the variance are associated with the y_i class, we are going to use the index *j* to indicate them, so they are excluded from the sum):

$$\sum_i log\, p(x_i|y_i) = \sum_i \left(-log\, \sqrt{2\pi} - log\, \sigma_j - \frac{(x_i-\mu_j)^2}{2\sigma_j^2} \right)$$

To maximize the likelihood, we need to compute the partial derivatives with respect to μ_j and σ_j (the first term is constant and can be removed):

$$\frac{\partial L}{\partial \mu_j} = \frac{\partial}{\partial \mu_j} \sum_i \left(-log\, \sigma_j - \frac{(x_i-\mu_j)^2}{2\sigma_j^2} \right) = -\sum_i \frac{(x_i-\mu_j)}{\sigma_j^2}$$

Setting the partial derivative equal to zero, we obtain a formula for the mean μ_j:

$$\mu_j = \frac{1}{n} \sum_i x_i$$

Analogously, we can compute the derivative with respect to σ_j:

$$\frac{\partial L}{\partial \sigma_j} = \frac{\partial}{\partial \sigma_j} \sum_i \left(-log\, \sigma_j - \frac{(x_i-\mu_j)^2}{2\sigma_j^2} \right) = -\frac{n}{\sigma_j} + \sum_i \frac{(x_i-\mu_j)^2}{\sigma_j^2}$$

Hence, the expression for the variance σ_j^2 is this:

$$\sigma_j^2 = \frac{\sum_i (x_i-\mu_j)^2}{n}$$

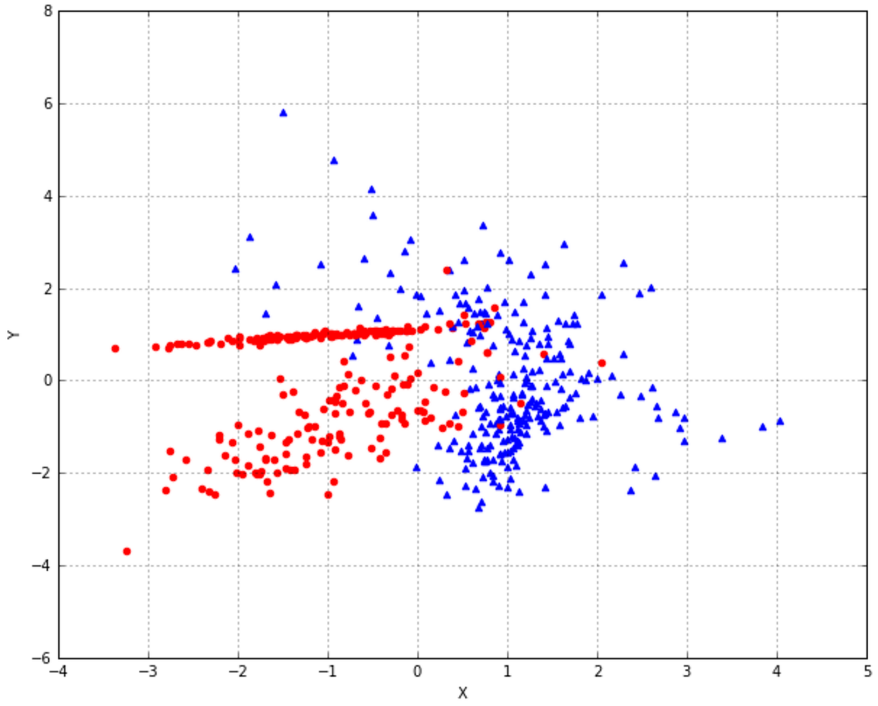
It's important to remember that the index *j* has been introduced as an auxiliary term, but in the actual computation of the likelihood, it refers to

the label assigned to the sample x_i.

As an example, we compare Gaussian Naive Bayes to a logistic regression using the ROC curves. The dataset has 300 samples with two features. Each sample belongs to a single class:

```
from sklearn.datasets import make_classification

nb_samples = 300

X, Y = make_classification(n_samples=nb_samples, n_features=2, n_informative=2, n_re
```

A plot of the dataset is shown in the following graph:



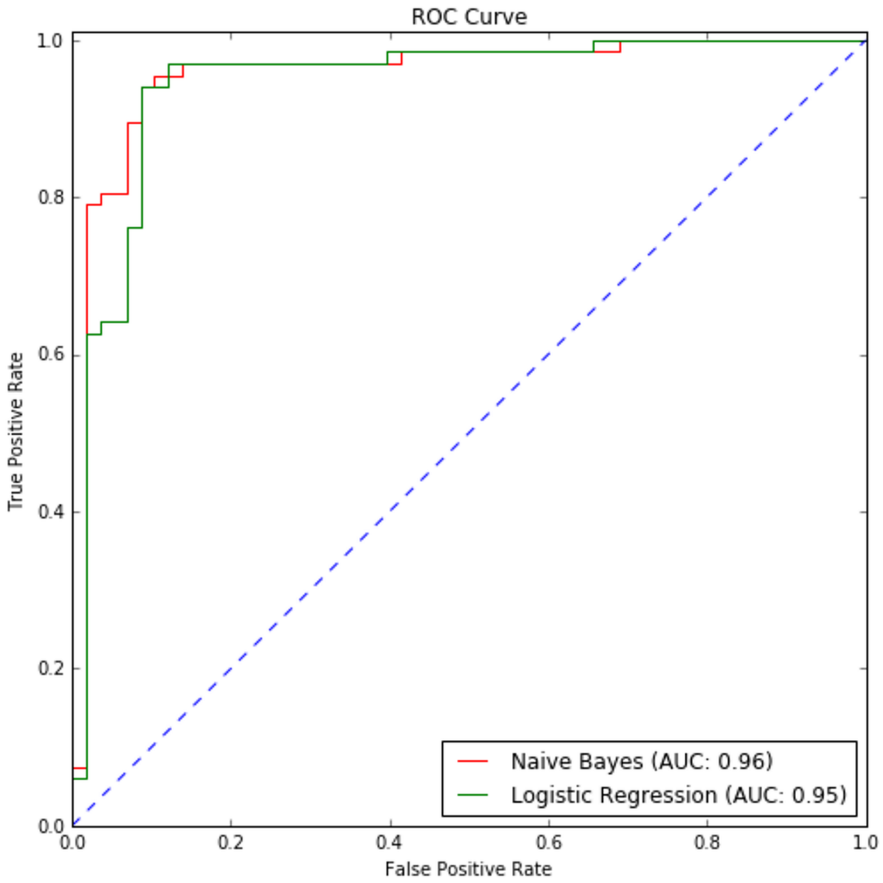
Dataset for Gaussian Naive Bayes test

Now we can train both models and generate the ROC curves (the Y scores for Naive Bayes are obtained through the `predict_proba` method):

```
from sklearn.naive_bayes import GaussianNB
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import roc_curve, auc
from sklearn.model_selection import train_test_split

X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.25)

gnb = GaussianNB()
gnb.fit(X_train, Y_train)
Y_gnb_score = gnb.predict_proba(X_test)

lr = LogisticRegression()
lr.fit(X_train, Y_train)
Y_lr_score = lr.decision_function(X_test)
```

```
fpr_gnb, tpr_gnb, thresholds_gnb = roc_curve(Y_test, Y_gnb_score[:, 1])
fpr_lr, tpr_lr, thresholds_lr = roc_curve(Y_test, Y_lr_score)
```

The resulting ROC curves (generated in the same way shown in the previous chapter) are shown in the following graph:



ROC curve comparing the scores of Naive Bayes versus a logistic regression

Naive Bayes' performance is slightly better than the logistic regression. However, the two classifiers have similar accuracy and **Area Under the Curve** (**AUC**). It's interesting to compare the performances of Gaussian and Multinomial Naive Bayes with the MNIST digit dataset. Each sample (belonging to 10 classes) is an 8 x 8 image encoded as an unsigned integer (0-255); therefore, even if each feature doesn't represent an actual count, it can be considered as a sort of magnitude or frequency:

```
from sklearn.datasets import load_digits
from sklearn.model_selection import cross_val_score

digits = load_digits()

gnb = GaussianNB()
mnb = MultinomialNB()

cross_val_score(gnb, digits.data, digits.target, scoring='accuracy', cv=10).mean()
0.81035375835678214

cross_val_score(mnb, digits.data, digits.target, scoring='accuracy', cv=10).mean()
0.88193962163008377
```

Multinomial Naive Bayes performs better than the Gaussian variant and the result is not really surprising. In fact, each sample can be thought of as a feature vector derived from a dictionary of 64 symbols and the effect of the Laplace coefficient can mitigate the deformations observed in a subset of the same digit class. The value of each feature (a pixel whose intensity is bounded between 0 and 16) is proportional to the count of each occurrence, so a multinomial distribution can better fit the data, while a Gaussian is slightly more limited by its mean and variance.