Bernoulli Naive Bayes

If *X* is a Bernoulli-distributed random variable, it can have only two possible outcomes (for simplicity, let's call them *0* and *1*) and their probability is this:

$$p(X) = \begin{cases} p \ \ if \ \ X = 1 \\ q \ \ if \ \ X = 0 \end{cases} \quad where \ \ q = 1 - p \ \ and \ \ 0 < p < 1$$

In general, the input vectors x_i are assumed to be multivariate Bernoulli distributed and each feature is binary and independent. The parameters of the model are learned according to a frequency count. Hence, if there are *n* samples with *m* features, the probability for the i^{th} feature is this ($N_{x^{(i)}}$ counts the number of times the $i^{th} = 1$):

$$p_i = \frac{N_{\bar{x}^{(i)} = 1}}{n}$$

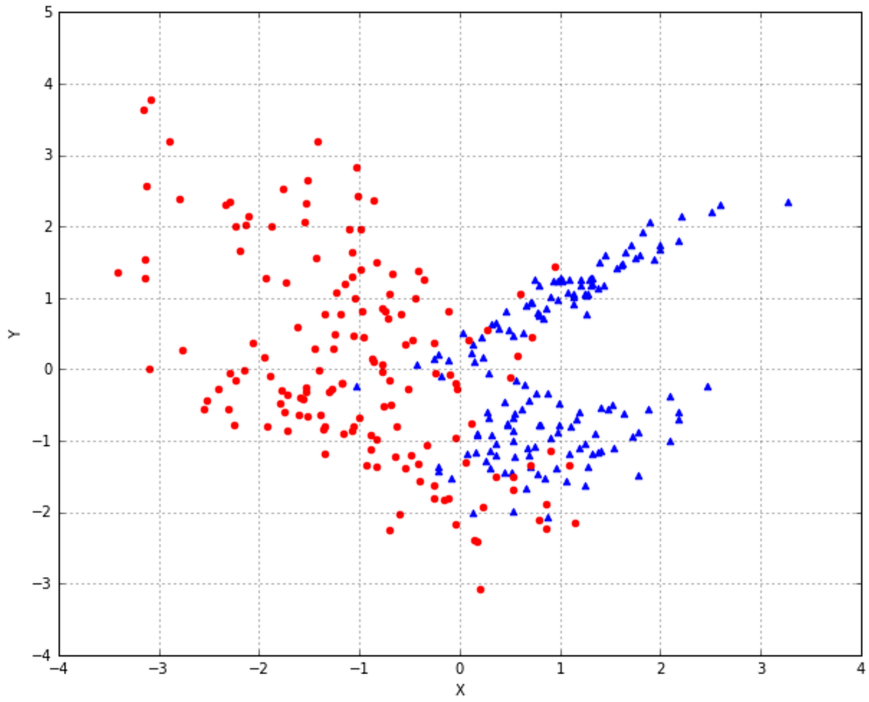
To test this algorithm with scikit-learn, we're going to generate a dummy dataset. Bernoulli Naive Bayes expects binary feature vectors; however, the `BernoulliNB` class has a `binarize` parameter, which allows us to specify a threshold that will be used internally to transform the features:

```
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
```

We have generated the bidimensional dataset shown in the following graph:



Dataset for Bernoulli Naive Bayes test

We have decided to use `0.0` as a binary threshold, so each point can be characterized by the quadrant where it's located. Of course, this is a rational choice for our dataset, but Bernoulli Naive Bayes is envisaged for binary feature vectors or continuous values, which can be precisely split with a predefined threshold:

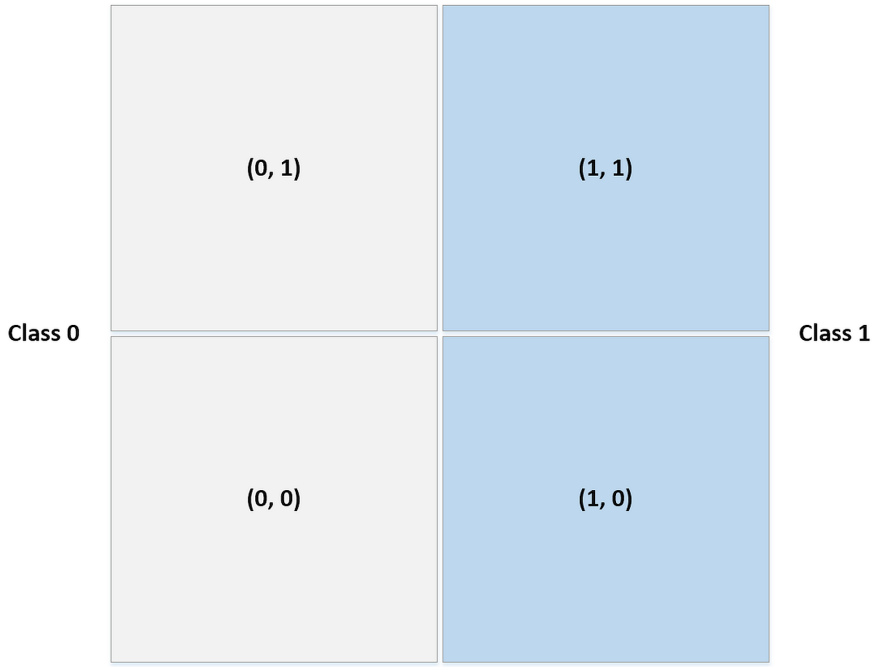
```
from sklearn.naive_bayes import BernoulliNB
from sklearn.model_selection import train_test_split

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

bnb = BernoulliNB(binarize=0.0)
bnb.fit(X_train, Y_train)

print(bnb.score(X_test, Y_test))
0.85333333333333339
```

The score is rather good, but if we want to understand how the binary classifier worked, it's useful to see how the data has been internally binarized:



Structure of the binarized dataset

Now, checking the Naive Bayes predictions, we obtain the following:

```
data = np.array([[0, 0], [0, 1], [1, 0], [1, 1]])

print(bnb.predict(data))
[0, 0, 1, 1]
```