Abstract

In this issue, we will learn an another algorithm linear classification - soft output - probability generation model: Naive Bayes Hypothesis.

Idea

In the last issue, GDA we learned makes an assumption that the dataset obeys Gaussian Dist. At the same time, Bernoulli Dist is introduced as a priori of the label to obtain the parameters via MAP.

Naive Bayes Hypothesis we learned in this issue assumes the relationship between the attributes of the data: the conditional independence hypothesis.

Algorithm

Normally, since there is p dimensions in x, we need to sample the joint distribution of p random variables to get the probability $p(x|y)$. However, as we know: A huge number of samples are needed to obtain the relative accurate probability approximation as to the high-dimensional space.

In the general directed probability graph model, different assumptions are made on the conditional independence relationship between various attribute dimensions, among which the simplest assumption is that one described in Naive Bayes model:

$$p(x|y) = \prod_{i=1}^{p} p(x_i|y)p(y) \tag{4}$$

In mathematical language:

$$x_i \perp x_j|y, \forall i \neq j \tag{5}$$

for a simple observation with Bayesian theorem:

$$p(y|x) = \frac{p(x|y)p(y)}{p(x)} = \frac{\prod_{i=1}^{p} p(x_i|y)p(y)}{p(x)} \tag{6}$$

Similar to GDA, here are some assumptions about the distributions of the data:

- x_i is a discrete variable:
  - Generally, we assume x_i obey Categorical Dist: $p(x_i = i|y) = \theta_i, \sum_{i=1}^{p} \theta_i = 1$
- x_i is a continuous variable:
  - Generally, we assume x_i obey Gaussian Dist: $p(x_i|y) = N(\mu_i, \Sigma_i)$
- Binary classification:
  - $y \sim Bernoulli(\phi) : p(y) = \phi^y(1-\phi)^{(1-y)}$
- Multi-classification:
  - $y \sim Categorical\ Dist \quad p(y_i) = \theta_i \quad \sum_{i=1}^{k} \theta_i = 1$

We are used to sample the dataset to estimate the parameters. we'll obtain the posteriori probability with Bayesian theorem when we are to predict test data after estimating the parameters.

Implement

```
import numpy as np
import os
os.chdir("../")
from models.linear_models import NaiveBayesClassifier

num_test = 100
x = np.linspace(0, 10, 1000)
k1, k2 = 0.1, 0.3
b1, b2 = 1, 2
x_train = x[:-num_test]
x_test = x[-num_test:]
v_1 = x_train * k1 + b1
v_2 = x_train * k2 + b2
train_data = np.r_[np.c_[x_train, v_1], np.c_[x_train, v_2]]
train_label = np.r_[np.ones_like(x_train), np.zeros_like(x_train)]

model = NaiveBayesClassifier()
model.fit(train_data, train_label)
print(model.get_params())

v_test = x_test * k2 + b2
data_test = np.c_[x_test, v_test]
print("accuary:", model.predict(data_test, 0))
```

```
([6.763511927008758, 0.0676351192700876], [4.499499499499499, 1.44994994994995],
[6.763511927008758, 0.6087160734307883], [4.499499499499499, 3.34984984984985])
accuary: 1.0
```