ML2016 HW2 Spam Classification

October 28, 2016

1 Logistic regression by gradient descent

1.1 Packages

```
import numpy as np
import pandas as pd
import random
import matplotlib.pyplot as plt
```

1.2 Preprocessing

I use **numpy** and **pandas** packages for data manipulation. The training data are loaded via `pandas.read_csv()` function. Training data are normalized using `pandas.DataFrame.mean()` and `pandas.DataFrame.std()` functions.

1.3 Define the sigmoid function

```
def sigmoid(z):
    return [1.0/(1.0+np.exp(-zi)) for zi in z]
```

1.4 Define the loss function

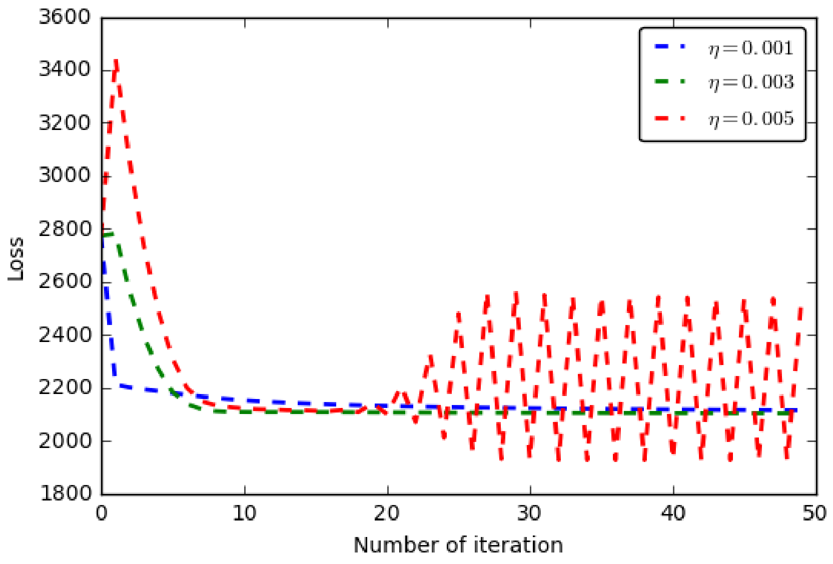
```
def computeLoss(X, y, coef):
    return -sum(np.log(sigmoid(y.multiply(np.dot(X, coef)))))
```

1.5 Main loop of gradient descent

```
# Set parameters
iteration = 200
eta = 0.003
alpha = eta # fixed learning rate
loss_history = list()
# Initialize coefficients to be all 0
coef = pd.Series(np.repeat(0.0, train_norm.shape[1], axis=0))
# Main loop of gradient descent
for ite in range(iteration):
    loss_history.append(computeLoss(train_norm, y, coef))
    temp = list()
    for i in range(train_norm.shape[1]):
        temp.append(coef[i] - alpha \
                                  * sum((sigmoid(np.dot(train_norm, coef)) - y) \
                                        .multiply(train_norm.ix[:,i])))
    for i in range(train_norm.shape[1]):
        coef[i] = temp[i]
```

1.6 Parameter determination

The parameters of gradient descent such as learning rate (`0.003`) and iteration time (`200`) are determined by grid-search method and some results are showed below:



2 Method2: Naive Bayes

2.1 Compute statistics

The other method I implement is the Naive Bayes algorithm. First, I compute the two prior probabilities: $P(\text{spam})$ and $P(\text{not spam})$. And the conditional mean and standard deviation for each feature, given spam or not spam, respectively. As showed below.

```
# Prior
p_spam = float(sum(train.ix[:,57])) / train.shape[0]
p_nonspam = 1 - p_spam
# Estimate mean and standard deviation for every features (0, 1, ..., 56)
train_spam = train[train.ix[:,57]==1]
means_spam = train_spam.mean()
std_spam = train_spam.std()
train_nonspam = train[train.ix[:,57]==0]
means_nonspam = train_nonspam.mean()
std_nonspam = train_nonspam.std()
```

2.2 Data cleansing

In order to compute the Gaussian PDF, I have to remove the 31-th feature since the standard deviation of the 31-th feature given the class = spam is 0.

```
# remove features with standard deviation = 0 (in each class)
rm_idx = [list(train.columns[std_spam == 0]), \
          list(train.columns[std_nonspam == 0])]
rm_idx = list(set([item for sublist in rm_idx for item in sublist]))
```

2.3 Prediction

In the prediction part, I load all the necessary statistics derived from the training data, such as prior probabilities: $P(\text{spam})$ and $P(\text{not spam})$ and all the conditional means and standard deviations. Then, I define the following functions to compute all the conditional probabilities used in Naive Bayes algorithm:

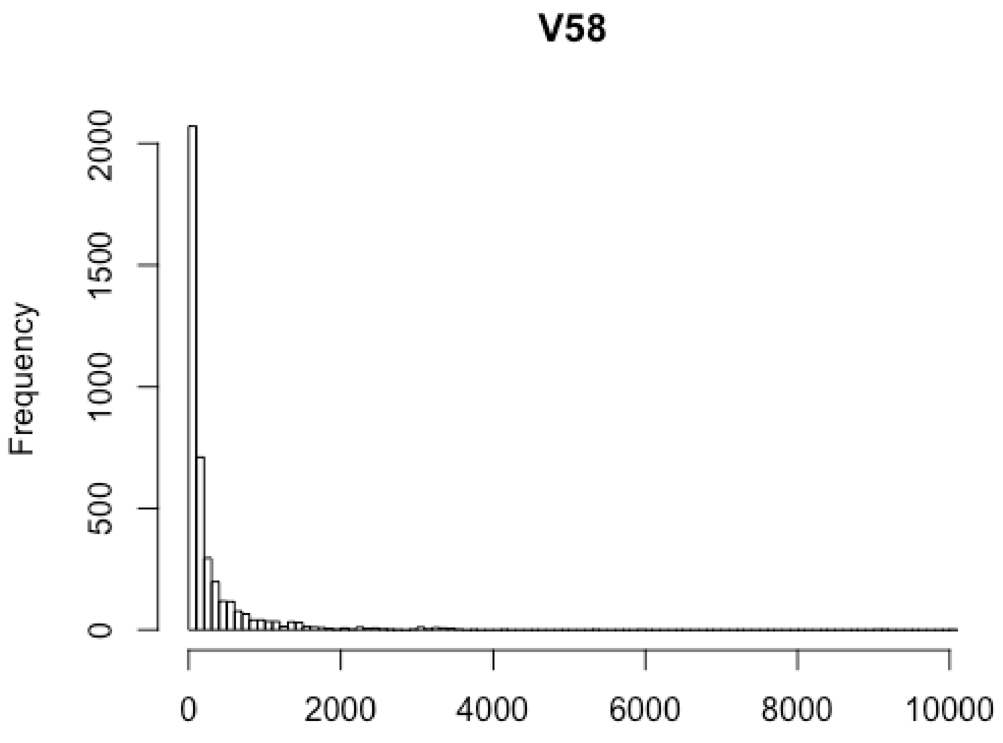
```
def gaussianPDF(x, mean, std):
    return (1 / (np.sqrt(2*np.pi) * std)) * np.exp(-(x-mean)**2/(2*std**2))
def probGiven0(x, dim):
    return gaussianPDF(x, means_nonspam[dim], std_nonspam[dim]) * p_nonspam
def probGiven1(x, dim):
    return gaussianPDF(x, means_spam[dim], std_spam[dim]) * p_spam
def prob0Givenx(x, dim):
    return probGiven0(x, dim) / (probGiven0(x, dim) + probGiven1(x, dim))
def prob1Givenx(x, dim):
    return probGiven1(x, dim) / (probGiven0(x, dim) + probGiven1(x, dim))
```

Now, I can compute the posterior probability for each sample in the testing data, and get the predicted class by simply comparing $P(\text{spam}|x)$ and $P(\text{not spam}|x)$:

```
result = list()
for i in range(test.shape[0]):
    prob0 = 1
    prob1 = 1
    for j in range(test.shape[1]):
        if j in rm_idx:
            continue
        prob0 *= prob0Givenx(test.ix[i,j], j)
        prob1 *= prob1Givenx(test.ix[i,j], j)
    result.append(1 if prob0 < prob1 else 0)
```

3 Performance comparison

The accuracy of the Naive Bayes model (around `0.76`) is much worse than that of the logistic regression model (around `0.92`). I think the main reason is that, in Naive Bayes approach, we assume that all the features are independent and Gussian distributed. This is a very strong assumption. As can be seen from the following histogram, most of the features are apparently **NOT** Gaussian distributed. Hence, it is expected that the simple Naive Bayes are not suitable for this dataset.



learning rate