Applied Data Mining Homework 01

Xun Zhao, xz2827

2019.1.29

Problem 1: Naive Bayes

1.

The naive Bayes classifier is based on an assumption that all the features are independent from each other. Meanwhile, in the case of this problem, it is mentioned that the input variables $\vec{x}_i, i = 1, 2, ..., N$, which contains 5 features $(x_i^{(1)}, x_i^{(2)}, ..., x_i^{(5)})$, has a spherical Guassian distribution. According to the defination of spherical Guassian, for all observation $\vec{x}_i$, $(x_i^{(1)}, x_i^{(2)}, ..., x_i^{(5)})$should be independent from each other. Thus, the training data set fits the requirement of naive Bayes classifier.

Estimation formula is as follows,

$$\begin{aligned}
\hat{y}_{new} &= f(\vec{x}_{new}) \\
&= \underset{c_k}{argmax} P(Y \in c_k) \cdot P(X = \vec{x}_{new} | Y \in c_k) \\
&= \underset{c_k}{argmax} P(Y \in c_k) \cdot \prod_{i=1}^{5} P(X^{(i)} = \vec{x}_{new}^{(i)} | Y \in c_k) \\
&= \underset{c_k}{argmax} \frac{N_{c_k}}{N_{c_1} + N_{c_2} + N_{c_3}} \cdot \frac{1}{\sqrt{(2\pi)^5 \prod_{i=1}^{5} \sigma_i^2}} exp[-\frac{1}{2} \sum_{i=1}^{5} \frac{(\vec{x}_{new}^{(i)} - \vec{\mu}^{(i)})^2}{\sigma_i^2}]
\end{aligned}$$

2.

In the estimation function given above, $\vec{\mu}$ is the mean vector of all $\vec{x}$ that belong to different classes c_k,

$$\vec{\mu}_k = (\frac{\Sigma \vec{x}^{(1)}}{N_{c_k}}, ..., \frac{\Sigma \vec{x}^{(5)}}{N_{c_k}}),\ \vec{x} \in c_k$$

and σ is the estimated variance of different classes and different features,

$$\sigma_k^{(i)} = \frac{1}{N_{c_k}} \Sigma(\vec{x}^{(i)} - \vec{\mu_k}^{(i)})^2,\ \vec{x} \in c_k$$

where $\vec{x}^{(i)}$ is the i-th feature of $\vec{x}$ that belongs to class c_k.

$P(Y \in c_k)$ is estimated as

$$P(Y \in c_k) = \frac{N_{c_k}}{N_{c_1} + N_{c_2} + N_{c_3}}$$

3.

I think the classifier works well under the independence assumption, because the distribution model is well defined and parameters can be estimated easily.

However in some ways, the behavior depends on how the data set is distributed and how the new data is given.

For example, if three classes are highly overlapped, the estimation function can get three high probabilities in all classes, and choose the largest one. In this case, it is more likely to draw a wrong conclusion because the difference among three probabilities is small and misleading.

For another example, even three classes are well divided, if the new data is far away from most training data points, it can also generate three low but similar probabilities that gives little information.

Problem 2: Perceptron

1.

The minimum empirical risk is $\frac{1}{22}$, for the boundary can classify all the solid circles to one class excpet one open circle ($\tilde{x}_2$), and other open circles into another class. Then there is only one misclassification, namely:

$$R = \frac{1}{n}\Sigma_{i=1}^{n} L(f(x_i), y_i) = \frac{1}{22}$$

2.

To classify, we need to compute the value of $\vec{v}_H \cdot \vec{x} + c$.

$$\begin{aligned}\vec{v}_H \cdot \vec{x}_1 + c &= \begin{pmatrix} \frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} \end{pmatrix} \cdot \begin{pmatrix} -3 \\ 0 \end{pmatrix} + \frac{1}{2\sqrt{2}} \\ &= -\frac{5}{4}\sqrt{2} < 0\end{aligned}$$

$$\begin{aligned}\vec{v}_H \cdot \vec{x}_2 + c &= \begin{pmatrix} \frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} \end{pmatrix} \cdot \begin{pmatrix} \frac{1}{2} \\ \frac{1}{2} \end{pmatrix} + \frac{1}{2\sqrt{2}} \\ &= \frac{\sqrt{2}}{4} > 0\end{aligned}$$

Thus, $\vec{x}_1$ is classifed to the class that is below the boundary (class -1), and $\vec{x}_2$ is classifed to the class above the boundary (class $+1$).

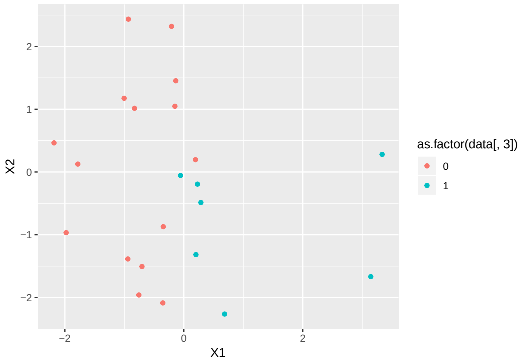
3.

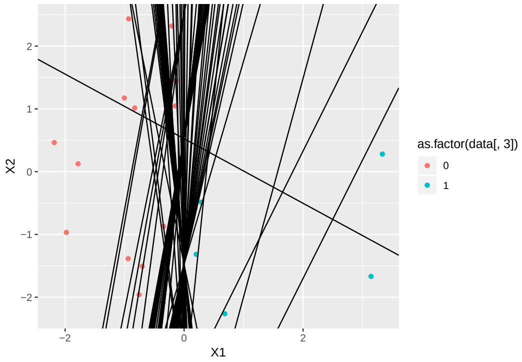
The algorithm will not have a solution, the value of $\vec{v}_H$ will fluctuate in a small range. It is because every time it iterate through all the points, there will always be some point(s) that is misclassified, and the $\vec{v}_H$ changes every time.

$$\vec{v}_{H,new} = \vec{v}_H + \alpha L(\hat{y}_i, y_i) \cdot \vec{x}_i$$

Basically, it is because the data points are linearly non-separable. Under the condition that learning rate α is constant, the result is not convergent.

The result of perceptron algorithm with $\alpha = 1$ is as follow, and the boundary lines are shown as black lines after refreshing $\vec{v}_H$ for 100 times,



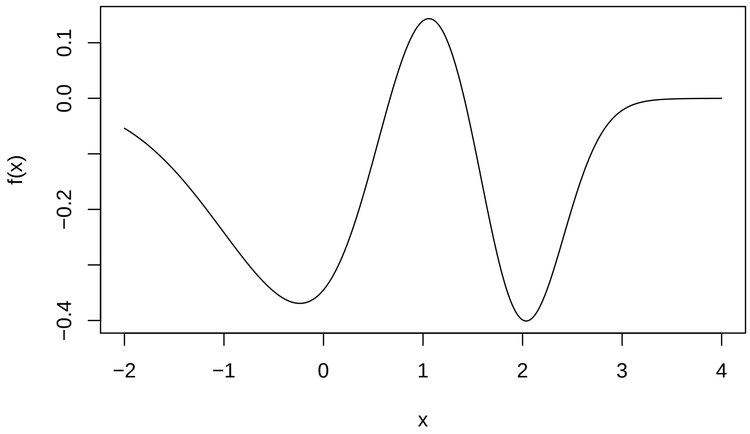


Problem 3: Gradient descent

1.

The $f(x)$ is plotted as follows,

```
f = function(x){-dnorm(x, 0, 1) + 0.5 * dnorm(x, 1, 0.5) - 0.4 * dnorm(x, 2, 0.4)}
x = seq(-2, 4, length.out = 500)
graph = spline(x, f(x), n = 1000)
plot(graph, type = 'l', xlab = 'x', ylab = 'f(x)')
```



2.

```
f.prime = function(x, delta = 0.001){(f(x + delta) - f(x - delta)) / (2 * delta)}
print(f.prime(-2))
```

```
## [1] -0.1079819
```

3.

```
gred.des = function(x1, epsilon = 0.05){
    n = 1
    x.data = c(x1)
    while(n == 1 || abs(f.prime(x1)) > epsilon){
        pre = x1
        x1 = x1 - f.prime(x1) / n
        x.data = c(x.data, x1)
        n = n + 1
    }
    return(list(n, x.data, x1, f(x1)))
}
```

When setting $\epsilon = 0.05$ and $\alpha_n = \frac{1}{n}$:

```
result = gred.des(-2, epsilon = 0.05)
print(result[1][[1]]) # Iteration times
```

```
## [1] 14945
```

```
print(summary(result[2][[1]])) # All x_i
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
## -2.0000 -0.4180 -0.3529 -0.4037 -0.3253 -0.3095
```

```
print(result[3][[1]]) # Final x_n
```

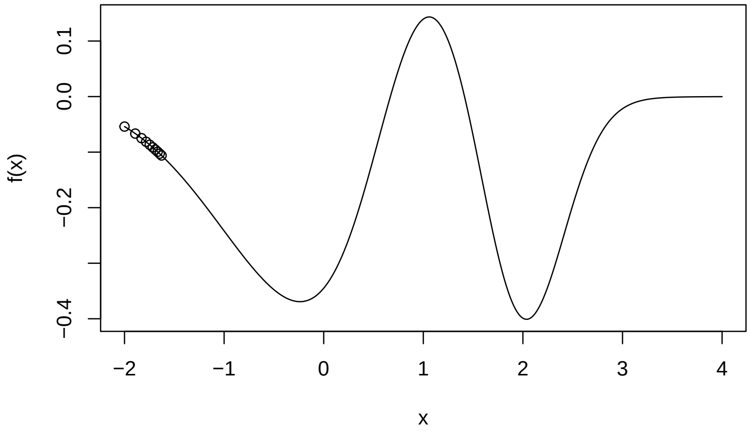
```
## [1] -0.3095104
```

```r
print(result[4][[1]]) # Minimum of f(x)
```

```
## [1] -0.3673588
```

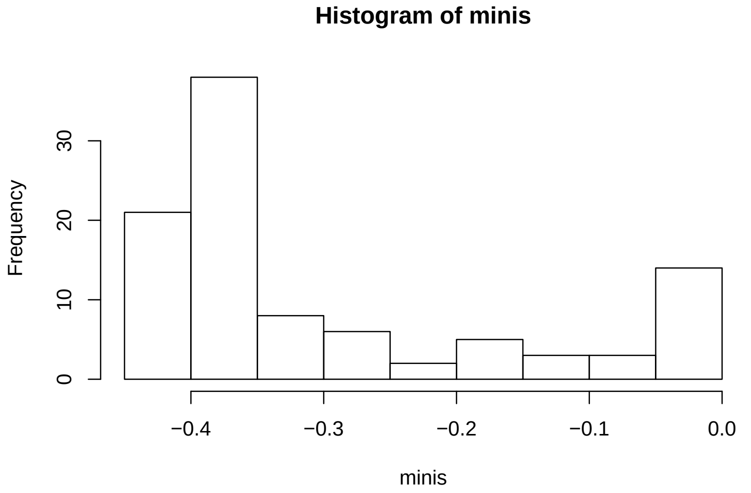
4.

x_1, ..., x_{10} are plotted as follows,



5.

The hist graph is as follows,



The minimum value of all 100 returns is as follows, which can be seen as the global minimum of $f(x)$.

```
## [1] -0.4009069
```