CS 182: Introduction to Machine Learning, Fall 2021
Homework 2

(Due on Tuesday, Nov. 2 at 11:59pm (CST))

Notice:

- Please submit your assignments via Gradescope. The entry code is KYJ626.
- Please make sure you select your answer to the corresponding question when submitting your assignments.
- Each person has a total of five days to be late without penalty for all the homeworks. Each late delivery less than one day will be counted as one day.

1. [10 points] Given the training data in Fig. 1, please use Naive Bayes classifier to predict the class (buys_computer) of the following new example: age> 40, income=high, student=no, credit_rating=excellent.

RID	age	income	student	credit_rating	Class: buys_computer
1	<=30	high	no	fair	no
2	<=30	high	no	excellent	no
3	31 . . . 40	high	no	fair	yes
4	>40	medium	no	fair	yes
5	>40	low	yes	fair	yes
6	>40	low	yes	excellent	no
7	31 . . . 40	low	yes	excellent	yes
8	<=30	medium	no	fair	no
9	<=30	low	yes	fair	yes
10	>40	medium	yes	fair	yes
11	<=30	medium	yes	excellent	yes
12	31 . . . 40	medium	no	excellent	yes
13	31 . . . 40	high	yes	fair	yes
14	>40	medium	no	excellent	no

Figure 1: The Buy Computer data.

Solution:

Let E = (age > 40, income = high, student = no, credit_rating = excellent)

E_1 is age > 40, E_2 is income = high, E_3 is student = no, E_4 is credit-rating = excellent

We need to compute P(yes | E) and P(no | E) and compare them.

$$P(\text{ yes } \mid \text{E}) = \frac{P(\text{E}_1 \mid \text{ yes })P(\text{E}_2 \mid \text{ yes })P(\text{E}_3 \mid \text{ yes })P(\text{E}_4 \mid \text{ yes })P(\text{ yes })}{P(E)}$$
$$P(\text{ no } \mid \text{E}) = \frac{P(\text{E}_1 \mid \text{ no })P(\text{E}_2 \mid \text{ no })P(\text{E}_3 \mid \text{ no })P(\text{E}_4 \mid \text{ no })P(\text{ no })}{P(E)}$$
$$P(\text{ yes }) = 9/14 = 0.643 \quad P(\text{ no }) = 5/14 = 0.357$$

$$P(\text{E1} \mid \text{ yes }) = 3/9 = 0.333 \quad P(\text{E1} \mid \text{ no }) = 2/5 = 0.4$$
$$P(\text{E2} \mid \text{ yes }) = 2/9 = 0.222 \quad P(\text{E2} \mid \text{ no }) = 2/5 = 0.4$$
$$P(\text{E3} \mid \text{ yes }) = 3/9 = 0.333 \quad P(\text{E3} \mid \text{ no }) = 4/5 = 0.8$$
$$P(\text{E4} \mid \text{ yes }) = 3/9 = 0.333 \quad P(\text{E4} \mid \text{ no }) = 3/5 = 0.6$$

$$P(\text{ yes } \mid \text{E}) = \frac{3/9 \times 2/9 \times 3/9 \times 3/9 \times 9/14}{P(\text{E})} = \frac{0.0053}{P(\text{E})} < P(\text{ no } \mid \text{E}) = \frac{2/5 \times 2/5 \times 4/5 \times 3/5 \times 5/14}{P(\text{E})} = \frac{0.0274}{P(\text{E})}$$

Hence, the Naïve Bayes classifier predicts buys_computer = no for the new example.

2. [20 points] Consider a two-class classification problem with L training samples $(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_L, y_L)$, where the input $\mathbf{x}_l \in \mathbb{R}^N$, $l = 1, \ldots, L$ contains N features. Suppose we use the logistic regression following the batch learning scheme, where the output is computed as

$$\hat{y}_l = \text{sigmoid}\left(\mathbf{w}^T \mathbf{x}_l\right).$$

Derive the update rule for parameter $\mathbf{w}$ under the following settings:

(a) $y_l \in \{0, 1\}$, $l = 1, \ldots, L$, where $P(y_l = 1 \mid \hat{y}_l) = \text{sigmoid}\left(\mathbf{w}^T \mathbf{x}_l\right)$ and $P(y_l = 0 \mid \hat{y}_l) = 1 - P(y_l = 1 \mid \hat{y}_l)$. [10 points]

(b) $y_l \in \{-1, 1\}$, $l = 1, \ldots, L$, where $P(y_l = 1 \mid \hat{y}_l) = \text{sigmoid}\left(\mathbf{w}^T \mathbf{x}_l\right)$ and $P(y_l = -1 \mid \hat{y}_l) = \text{sigmoid}\left(-\mathbf{w}^T \mathbf{x}_l\right)$. [10 points]

(a) $y_l \in \{0, 1\}$, $l = 1, \ldots, L$, where $P(y_l = 1 \mid \hat{y}_l) = \text{sigmoid}\left(\mathbf{w}^T \mathbf{x}_l\right)$ and $P(y_l = 0 \mid \hat{y}_l) = 1 - P(y_l = 1 \mid \hat{y}_l)$.

Solution:

The likelihood function is

$$\mathcal{L} = \prod_{l=1}^{L} (\hat{y}_l)^{y_l} (1 - \hat{y}_l)^{1 - y_l}.$$

The loss function is accordingly deinfed as

$$\ell = -\log \mathcal{L} = -\sum_{l=1}^{L} \left(y_l \log \hat{y}_l + (1 - y_l) \log (1 - \hat{y}_l)\right).$$

Then the derivative of ℓ w.r.t. $\mathbf{w}$ can be obtained as

$$\begin{aligned}
\frac{\partial \ell}{\partial \mathbf{w}} &= -\sum_{l=1}^{L} \left(\frac{y_l}{\hat{y}_l} \hat{y}_l (1 - \hat{y}_l) \mathbf{x}_l - \frac{1 - y_l}{1 - \hat{y}_l} \hat{y}_l (1 - \hat{y}_l) \mathbf{x}_l \right) \\
&= -\sum_{l=1}^{L} \left(y_l (1 - \hat{y}_l) \mathbf{x}_l - (1 - y_l) \hat{y}_l \mathbf{x}_l \right) \\
&= \sum_{l=1}^{L} \left((\hat{y}_l - y_l) \mathbf{x}_l \right),
\end{aligned}$$

which leads to the update rule for $\mathbf{w}$:

$$\mathbf{w}_{\text{new}} = \mathbf{w} - \eta \sum_{l=1}^{L} \left((\hat{y}_l - y_l) \mathbf{x}_l \right)$$

with η being the learning rate.

(b) $y_l \in \{-1, 1\}$, $l = 1, \ldots, L$, where $P(y_l = 1 \mid \hat{y}_l) = \text{sigmoid}\left(\mathbf{w}^T \mathbf{x}_l\right)$ and $P(y_l = -1 \mid \hat{y}_l) = \text{sigmoid}\left(-\mathbf{w}^T \mathbf{x}_l\right)$.

Solution 1:

The likelihood function can be defined as

$$\mathcal{L} = \prod_{l=1}^{L} (\hat{y}_l)^{\frac{1}{2} + \frac{y_l}{2}} \left(\text{sigmoid}\left(-\mathbf{w}^T \mathbf{x}_l\right)\right)^{\frac{1}{2} - \frac{y_l}{2}}.$$

The loss function is accordingly deinfed as

$$\ell = -\log \mathcal{L} = -\sum_{l=1}^{L} \left(\left(\frac{1}{2} + \frac{y_l}{2} \right) \log \hat{y}_l + \left(\frac{1}{2} - \frac{y_l}{2} \right) \log \left(\text{sigmoid}\left(-\mathbf{w}^T \mathbf{x}_l\right)\right) \right).$$

Then the derivative of ℓ w.r.t. $\mathbf{w}$ can be obtained as

$$\begin{aligned}
\frac{\partial \ell}{\partial \mathbf{w}} &= -\sum_{l=1}^{L}\left(\frac{\frac{1}{2}+\frac{y_l}{2}}{\hat{y}_l}\hat{y}_l\left(1-\hat{y}_l\right)\mathbf{x}_l - \frac{\frac{1}{2}-\frac{y_l}{2}}{\text{sigmoid}\left(-\mathbf{w}^T\mathbf{x}_l\right)}\text{sigmoid}\left(-\mathbf{w}^T\mathbf{x}_l\right)\left(1-\text{sigmoid}\left(-\mathbf{w}^T\mathbf{x}_l\right)\right)\mathbf{x}_l\right) \\
&= -\sum_{l=1}^{L}\left(\left(\frac{1}{2}+\frac{y_l}{2}\right)\left(1-\hat{y}_l\right)\mathbf{x}_l - \left(\frac{1}{2}-\frac{y_l}{2}\right)\left(1-\text{sigmoid}\left(-\mathbf{w}^T\mathbf{x}_l\right)\right)\mathbf{x}_l\right) \\
&= -\sum_{l=1}^{L}\left(\left(\frac{1}{2}+\frac{y_l}{2}\right)-\left(\frac{1}{2}-\frac{y_l}{2}\right)-\hat{y}_l\left(\frac{1}{2}+\frac{y_l}{2}\right)+\text{sigmoid}\left(-\mathbf{w}^T\mathbf{x}_l\right)\left(\frac{1}{2}-\frac{y_l}{2}\right)\right)\mathbf{x}_l \\
&= -\sum_{l=1}^{L}\left(y_l-\hat{y}_l\left(\frac{1}{2}+\frac{y_l}{2}\right)+\left(1-\hat{y}_l\right)\left(\frac{1}{2}-\frac{y_l}{2}\right)\right)\mathbf{x}_l \\
&= -\sum_{l=1}^{L}\left(\frac{1}{2}+\frac{y_l}{2}-\hat{y}_l\right)\mathbf{x}_l.
\end{aligned}$$

which leads to the update rule for $\mathbf{w}$:

$$\mathbf{w}_{\text{new}} = \mathbf{w} - \eta\sum_{l=1}^{L}\left(\left(\frac{1}{2}+\frac{y_l}{2}-\hat{y}_l\right)\mathbf{x}_l\right)$$

with η being the learning rate.

Solution 2:

The likelihood function can be defined as

$$\mathcal{L} = \prod_{l=1}^{L}\text{sigmoid}\left(y_l\mathbf{w}^T\mathbf{x}_l\right).$$

The loss function is accordingly deinfed as

$$\ell = -\log\mathcal{L} = -\sum_{l=1}^{L}\log\left(\text{sigmoid}\left(y_l\mathbf{w}^T\mathbf{x}_l\right)\right).$$

Then the derivative of ℓ w.r.t. $\mathbf{w}$ can be obtained as

$$\begin{aligned}
\frac{\partial \ell}{\partial \mathbf{w}} &= -\sum_{l=1}^{L}\left(\frac{\text{sigmoid}\left(y_l\mathbf{w}^T\mathbf{x}_l\right)}{\text{sigmoid}\left(y_l\mathbf{w}^T\mathbf{x}_l\right)}\left(1-\text{sigmoid}\left(y_l\mathbf{w}^T\mathbf{x}_l\right)\right)y_l\mathbf{x}_l\right) \\
&= -\sum_{l=1}^{L}\left(\left(1-\text{sigmoid}\left(y_l\mathbf{w}^T\mathbf{x}_l\right)\right)y_l\mathbf{x}_l\right).
\end{aligned}$$

which leads to the update rule for $\mathbf{w}$:

$$\mathbf{w}_{\text{new}} = \mathbf{w} - \eta\sum_{l=1}^{L}\left(\left(1-\text{sigmoid}\left(y_l\mathbf{w}^T\mathbf{x}_l\right)\right)y_l\mathbf{x}_l\right)$$

with η being the learning rate.

3. [20 points] Consider a multi-class classification problem with L training samples $(\mathbf{x}_1, \mathbf{y}_1), \ldots, (\mathbf{x}_L, \mathbf{y}_L)$, where the input $\mathbf{x}_l \in \mathbb{R}^N$ contains N features and output $\mathbf{y}_l \in \mathbb{R}^M$ is a zero vector with one entry equals one to indicate the class of sample l, $l = 1, \ldots, L$. Suppose we use a two-layer neural network following the batch learning scheme, then the loss function can be defined as

$$\ell\left(\mathbf{W}_1, \mathbf{W}_2, \mathbf{b}_1, \mathbf{b}_2\right) = -\sum_{l=1}^{L} \mathbf{y}_l^T \log\left(\text{softmax}\left(\mathbf{W}_2 \text{sigmoid}\left(\mathbf{W}_1 \mathbf{x}_l + \mathbf{b}_1\right) + \mathbf{b}_2\right)\right),$$

where $\mathbf{W}_1$, $\mathbf{W}_2$, $\mathbf{b}_1$, and $\mathbf{b}_2$ are the training parameters. Derive the update rule for all parameters using gradient descent.

Solution:

Let $\mathbf{a}_{1,l} = \mathbf{W}_1 \mathbf{x}_l + \mathbf{b}_1$, $\mathbf{h}_{1,l} = \text{sigmoid}(\mathbf{a}_{1,l})$, and $\mathbf{a}_{2,l} = \mathbf{W}_2 \mathbf{h}_{1,l} + \mathbf{b}_2$. We can get

$$\begin{aligned}
\ell &= -\sum_{l=1}^{L} \mathbf{y}_l^T \log\left(\text{softmax}\left(\mathbf{a}_{2,l}\right)\right) \\
&= -\sum_{l=1}^{L} \mathbf{y}_l^T \log\left(\frac{e^{\mathbf{a}_{2,l}}}{\mathbf{1}^T e^{\mathbf{a}_{2,l}}}\right) \\
&= -\sum_{l=1}^{L} \mathbf{y}_l^T \mathbf{a}_{2,l} + \sum_{l=1}^{L} \log\left(\mathbf{1}^T e^{\mathbf{a}_{2,l}}\right).
\end{aligned}$$

The differential of ℓ is

$$\begin{aligned}
d\ell &= d\left(-\sum_{l=1}^{L} \mathbf{y}_l^T \mathbf{a}_{2,l} + \sum_{l=1}^{L} \log\left(\mathbf{1}^T e^{\mathbf{a}_{2,l}}\right)\right) \\
&= -\sum_{l=1}^{L} \mathbf{y}_l^T d\mathbf{a}_{2,l} + \sum_{l=1}^{L} \frac{\mathbf{1}^T \left(e^{\mathbf{a}_{2,l}} \odot d\mathbf{a}_{2,l}\right)}{\mathbf{1}^T e^{\mathbf{a}_{2,l}}} \\
&= -\sum_{l=1}^{L} \mathbf{y}_l^T d\mathbf{a}_{2,l} + \sum_{l=1}^{L} \frac{\left(e^{\mathbf{a}_{2,l}}\right)^T d\mathbf{a}_{2,l}}{\mathbf{1}^T e^{\mathbf{a}_{2,l}}} \\
&= -\sum_{l=1}^{L} \mathbf{y}_l^T d\mathbf{a}_{2,l} + \sum_{l=1}^{L} \text{softmax}\left(\mathbf{a}_{2,l}\right)^T d\mathbf{a}_{2,l} \\
&= \sum_{l=1}^{L} \left(\text{softmax}\left(\mathbf{a}_{2,l}\right) - \mathbf{y}_l\right)^T \left(\left(d\mathbf{W}_2\right)\mathbf{h}_{1,l} + \mathbf{W}_2 d\mathbf{h}_{1,l} + d\mathbf{b}_2\right),
\end{aligned}$$

which leads to

$$\begin{aligned}
\frac{d\ell}{d\mathbf{W}_2} &= \sum_{l=1}^{L} \left(\text{softmax}\left(\mathbf{a}_{2,l}\right) - \mathbf{y}_l\right)\mathbf{h}_{1,l}^T \\
\frac{d\ell}{d\mathbf{h}_{1,l}} &= \mathbf{W}_2^T \left(\text{softmax}\left(\mathbf{a}_{2,l}\right) - \mathbf{y}_l\right) \\
\frac{d\ell}{d\mathbf{b}_2} &= \sum_{l=1}^{L} \left(\text{softmax}\left(\mathbf{a}_{2,l}\right) - \mathbf{y}_l\right).
\end{aligned}$$

Since $\mathbf{h}_{1,l} = \sigma(\mathbf{a}_{1,l})$, we have

$$\begin{aligned}
d\ell &= \sum_{l=1}^{L} \left(\frac{\partial \ell}{\partial \mathbf{h}_{1,l}} \odot \text{sigmoid}'(\mathbf{a}_{1,l})\right)^T d\left(\mathbf{W}_1 \mathbf{x}_l + \mathbf{b}_1\right) \\
&= \sum_{l=1}^{L} \left(\frac{\partial \ell}{\partial \mathbf{h}_{1,l}} \odot \text{sigmoid}'(\mathbf{a}_{1,l})\right)^T \left(\left(d\mathbf{W}_1\right)\mathbf{x}_l + d\mathbf{b}_1\right),
\end{aligned}$$

indicating that

$$\frac{\partial \ell}{\partial \mathbf{W}_1} = \sum_{l=1}^{L} \left(\frac{\partial \ell}{\partial \mathbf{h}_{1,l}} \odot \text{sigmoid}'(\mathbf{a}_{1,l}) \right) \mathbf{x}_l^T$$
$$\frac{\partial \ell}{\partial \mathbf{b}_1} = \sum_{l=1}^{L} \left(\frac{\partial \ell}{\partial \mathbf{h}_{1,l}} \odot \text{sigmoid}'(\mathbf{a}_{1,l}) \right)^T .$$

Therefore, the update rules for $\mathbf{W}_1, \mathbf{W}_2, \mathbf{b}_1, \mathbf{b}_2$ are as follows:

$$\mathbf{W}_2^{update} = \mathbf{W}_2 - \alpha \sum_{l=1}^{L} \left(\text{softmax}\left(\mathbf{a}_{2,l}\right) - \mathbf{y}_l \right) \mathbf{h}_{1,l}^T$$
$$\mathbf{W}_1^{update} = \mathbf{W}_1 - \alpha \sum_{l=1}^{L} \left(\frac{\partial \ell}{\partial \mathbf{h}_{1,l}} \odot \text{sigmoid}'(\mathbf{a}_{1,l}) \right) \mathbf{x}_l^T$$
$$\mathbf{b}_2^{update} = \mathbf{b}_2 - \alpha \sum_{l=1}^{L} \left(\text{softmax}\left(\mathbf{a}_{2,l}\right) - \mathbf{y}_l \right)$$
$$\mathbf{b}_1^{update} = \mathbf{b}_1 - \alpha \sum_{l=1}^{L} \left(\frac{\partial \ell}{\partial \mathbf{h}_{1,l}} \odot \text{sigmoid}'(\mathbf{a}_{1,l}) \right)^T ,$$

where α is the learning rate.

4. [20 points] The problem of maximizing margin can be converted into the following equivalent problem

$$\begin{aligned} \underset{\mathbf{w},w_0}{\text{minimize}} \quad & \tfrac{1}{2}\|\mathbf{w}\|^2 \\ \text{subject to} \quad & y_i(\mathbf{w}^T\mathbf{x}_i + w_0) \geq 1, \quad i = 1, \cdots, N. \end{aligned}$$

(a) By introducing Lagrange multipliers $\{\alpha_i\}$, please give the Lagrangian function and the dual representation of the maximum margin problem. [10 points]

(b) Please show that the maximum of the margin $\gamma = \frac{1}{\|\mathbf{w}\|}$ is given by

$$\frac{1}{\gamma_{\max}^2} = \sum_{i=1}^{N} \alpha_i.$$

(Hint: $\{\alpha_i\}$ can be obtained by solving the dual representation of the maximum margin problem.) [10 points]

Solution:

1. The Lagrangian function is given by

$$L(\mathbf{w}, w_0, \boldsymbol{\alpha}) = \frac{1}{2}\|\mathbf{w}\|^2 - \sum_{i=1}^{N} a_i \left(y_i \left(\mathbf{w}^T\mathbf{x}_i + w_0\right) - 1\right). \tag{1}$$

The dual representation of the maximum margin problem is given by

$$\max_{\boldsymbol{\alpha}} \quad \widetilde{L}(\boldsymbol{\alpha}) = \sum_{i=1}^{N} \alpha_i - \frac{1}{2}\sum_{i=1}^{N}\sum_{j=1}^{N} \alpha_i\alpha_j y_i y_j \mathbf{x}_i^T\mathbf{x}_j \tag{2}$$

$$\text{subject to} \quad \alpha_i \geq 0, \quad i = 1, \cdots, N \tag{3}$$

$$\sum_{i=1}^{N} \alpha_i y_i = 0. \tag{4}$$

2. Because $\gamma = \frac{1}{\|\mathbf{w}\|}$, maximizing γ is equal to minimizing $\|\mathbf{w}\|$. Meanwhile, the primal problem is a convex problem and Slater's condition is satisfied. Hence, the optimal solution must satisfy KKT conditon. From the KKT condition of the primal problem we have

$$\alpha_i \geqslant 0, i = 1, \cdots, N \tag{5}$$

$$y_i(\mathbf{w}^T\mathbf{x}_i + w_0) - 1 \geqslant 0, i = 1, \cdots, N \tag{6}$$

$$\alpha_i \left(y_i(\mathbf{w}^T\mathbf{x}_i + w_0) - 1\right) = 0, i = 1, \cdots, N \tag{7}$$

$$\mathbf{w} = \sum_{i=1}^{N} \alpha_i y_i \mathbf{x}_i \tag{8}$$

$$\sum_{i=1}^{N} \alpha_i y_i = 0. \tag{9}$$

According to (5), (6) and (7), when the primal problem reach its optimal value,

$$L(\mathbf{w}, w_0, \boldsymbol{\alpha}) = \frac{1}{2}\|\mathbf{w}\|^2. \tag{10}$$

Due to strong convexity, the optimal value of the original problem is equal to the optimal value of the dual problem. Therefore, combining (2), (8) and (10), we can get

$$\frac{1}{2}\|\mathbf{w}\|^2 = \sum_{i=1}^{N} \alpha_i - \frac{1}{2}\sum_{i=1}^{N}\sum_{j=1}^{N} \alpha_i\alpha_j y_i y_j \mathbf{x}_i^T\mathbf{x}_j = \sum_{i=1}^{N} \alpha_i - \frac{1}{2}\|\mathbf{w}\|^2. \tag{11}$$

Therefore, we have

$$\frac{1}{\gamma_{\max}^2} = \|\mathbf{w}\|^2 = \sum_{i=1}^{N} \alpha_i. \tag{12}$$

1. [30 points] *[Programming Problem]* Binary classification

   (a) Simple perceptron

      (1) Write down the analytical expressions of the cross-entropy error function and the gradient for one instance $(\mathbf{x}^{(\ell)}, y^{(\ell)})$. [6 points]

      (2) Learning a simple perceptron with batch GD (using the given initializations $\mathbf{w}^{\mathsf{init}}$ and w_0^{init}) based on the training set $(\mathbf{X}_{\mathsf{train}}, \mathbf{y}_{\mathsf{train}})$: use the training set and the validation set to obtain a good learning rate (you can set the maximum for iterations to 50 and try different learning rate in $[10^{-4}, 10^{-8}]$); output the learned model and evaluate its performance on the test set with the classification accuracy. [6 points]

      (3) Learning a simple perceptron with SGD (using the given initializations $\mathbf{w}^{\mathsf{init}}$ and w_0^{init}) based on the training set $(\mathbf{X}_{\mathsf{train}}, \mathbf{y}_{\mathsf{train}})$: use the training set and the validation set to obtain a good learning rate (you can set the maximum for iterations and try different learning rate); output the learned model and evaluate its performance on the test set with the classification accuracy. [6 points]

   (b) SVM

      (1) Use the function '`svm`' in package '`sklearn`' to do the binary classification and output the model. [6 points]

   (c) Try to compare the models learned from (a) and (b). [6 points]

   The dataset can be downloaded from Piazza.com, and you can use the following sample code to load the data.

```
import numpy as np
data = np.loadtxt('XXX.txt')
```

   (Note: You need to use the Jupyter Notebook to finish (a)-(2), (a)-(3), (b)-(1), and (c), export the notebook as PDF file, and then upload it onto Gradescope.)

Untitled

November 16, 2021

[10]:
```
from sklearn import svm
import numpy as np
```

[28]:
```
X_train = np.loadtxt('X_train.txt')
X_val = np.loadtxt('X_val.txt')
X_test = np.loadtxt('X_test.txt')
y_train = np.loadtxt('y_train.txt')
y_val = np.loadtxt('y_val.txt')
y_test = np.loadtxt('y_test.txt')
```

[15]:
```
# BGD
def sigmoid(x):
    s = 1 / (1 + np.exp(-x))
    return s


w = np.loadtxt('w.txt')
w0 = np.loadtxt('w0.txt')
lr = 1e-4
num_classes = w.shape[0]
num_train = X_train.shape[1]
for iter in range(50):
    loss = 0.0
    dw = np.zeros(num_classes)
    dw0 = 0
    for i in range(num_train):
        z = w.dot(X_train[:, i])+w0
        y = sigmoid(z)
        loss -= (y_train[i])*np.log(y)+(1-y_train[i])*np.log(1-y)
        dw += (y_train[i]-y)*X_train[:, i]
        dw0 += y_train[i]-y
    dw /= num_train
    dw0 /= num_train
    loss /= num_train
    w += lr*dw
    w0 += lr*dw0

    if(iter % 10 == 0):
```

```
        print(loss)
```

```
0.6929827835630806
0.6921510364922884
0.6913210183724732
0.6904927256008473
0.6896661545770586
```

[16]:
```
acc = 0
num_train = X_test.shape[1]
for i in range(num_train):
    z = w.dot(X_test[:, i])+w0
    y = sigmoid(z)
    y = y > 0.5
    if(y == y_test[i]):
        acc += 1
print(acc/num_train)
```

```
0.985
```

[17]:
```
print('w = ', w)
print('w0 = ', w0)
```

```
w =  [-0.00113014 -0.00104271 -0.00108625 -0.00098743 -0.00107847 -0.00115366
 -0.0010352  -0.00113063 -0.00124894 -0.00105578 -0.00103827 -0.00099284
 -0.00100423 -0.0010292  -0.00106541 -0.00105379 -0.00120739 -0.00102632
 -0.00087948 -0.00090802]
w0 =  -1.4276968508357241e-06
```

[37]:
```
# SGD
w = np.loadtxt('w.txt')
w0 = np.loadtxt('w0.txt')
lr = 1e-4
num_classes = w.shape[0]
num_train = X_train.shape[1]
for iter in range(100):
    loss = 0.0
    dw = np.zeros(w.shape[0])
    dw0 = 0

    ind = np.random.randint(num_train)
    z = w.dot(X_train[:, ind]) + w0
    y = sigmoid(z)
    loss -= (y_train[ind])*np.log(y)+(1-y_train[ind])*np.log(1-y)
    dw += (y_train[ind]-y)*X_train[:, ind]
    dw0 += y_train[ind]-y
    w += lr*dw
    w0 += lr*dw0
```

[39]:
```
print('w = ', w)
print('w0 = ', w0)
```

```
w =  [-0.00199491 -0.0020234  -0.0023812  -0.00260795 -0.00264227 -0.00195734
 -0.00216467 -0.00238086 -0.00161627 -0.00249241 -0.00197703 -0.00201305
 -0.0009546  -0.00289491 -0.00218822 -0.0016476  -0.00120636 -0.00149334
 -0.00212926 -0.00163975]
w0 =  -0.0003965595477106169
```

[38]:
```
acc = 0
num_train = X_test.shape[1]
for i in range(num_train):
    z = w.dot(X_test[:, i])+w0
    y = sigmoid(z)
    y = y > 0.5
    if(y == y_test[i]):
        acc += 1
print(acc/num_train)
```

```
0.97
```

[12]:
```
# SVM
clf = svm.SVC(kernel = 'linear')
clf.fit(X_train.T, y_train.T)
print('w = ', clf.coef_)
print('w0 = ', clf.intercept_)
print('Accuracy on train set: ', clf.score(X_train.T, y_train.T))
print('Accuracy on val set: ', clf.score(X_val.T, y_val.T))
print('Accuracy on test set: ', clf.score(X_test.T, y_test.T))
```

```
w =  [[-0.57721262 -0.71570866 -0.55397539 -0.7321757  -0.65845498 -0.61498196
  -0.65883582 -0.60687084 -0.78410835 -0.55826668 -0.60384754 -0.45058823
  -0.64957957 -0.60249223 -0.53559436 -0.59535748 -0.60592019 -0.50241082
  -0.73408134 -0.60186219]]
w0 =  [0.06308533]
Accuracy on train set:  0.9795714285714285
Accuracy on val set:  0.975
Accuracy on test set:  0.98
```

1. In the BGD, we need process all the data points in a iteration. But in SGD, it only process a random data point.
2. The simple perceptron is to find a hyperplane that separates the two sets. And the SVM tries to maximize the distance between two closest opposite sample points.

(The answer is not unique, but you should not only compare the value of the results.)

[ ]:
```

```