A Simple Learning Model: Perceptron

- Input vector $\mathbf{x} = [x_1, \ldots, x_d]^{\mathrm{T}}$.
- Give importance weights to the different inputs and compute a "Credit Score"

$$\text{“Credit Score”} = \sum_{i=1}^{d} w_i x_i.$$

- Approve credit if the "Credit Score" is acceptable.

$$\text{Approve credit if } \sum_{i=1}^{d} w_i x_i > \text{threshold}, \quad \text{(“Credit Score” is good)}$$

$$\text{Deny credit if } \sum_{i=1}^{d} w_i x_i < \text{threshold}. \quad \text{(“Credit Score” is bad)}$$

- How to choose the importance weights w_i

input x_i is important	$\Longrightarrow$	large weight $\lvert w_i \rvert$
input x_i beneficial for credit	$\Longrightarrow$	positive weight $w_i > 0$
input x_i detrimental for credit	$\Longrightarrow$	negative weight $w_i < 0$

A simple hypothesis set - the 'perceptron'

For input $\mathbf{x} = (x_1, \cdots, x_d)$ 'attributes of a customer'

$$\text{Approve credit if} \quad \sum_{i=1}^{d} w_i x_i > \text{threshold},$$

$$\text{Deny credit if} \quad \sum_{i=1}^{d} w_i x_i < \text{threshold}.$$

This linear formula $h \in \mathcal{H}$ can be written as

$$h(\mathbf{x}) = \text{sign}\left(\left(\sum_{i=1}^{d} w_i x_i\right) - \text{threshold}\right)$$

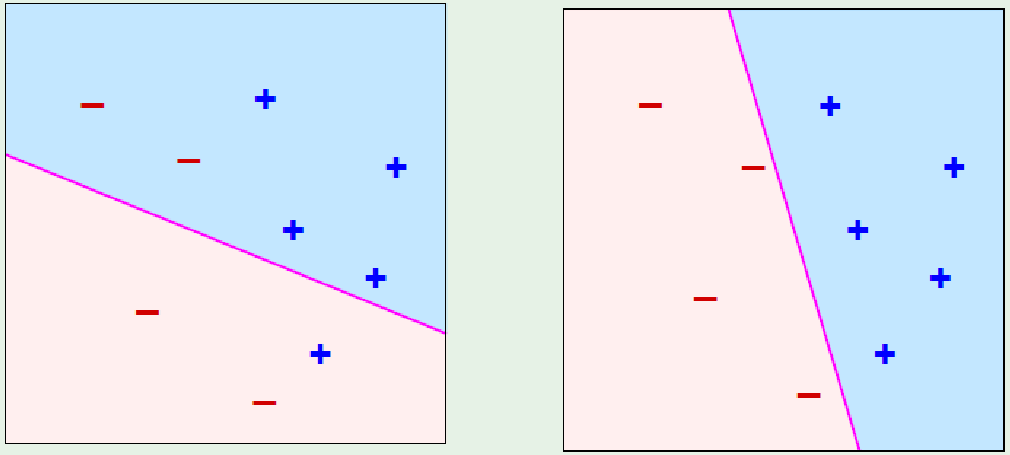
$$h(\mathbf{x}) = \text{sign}\left(\left(\sum_{i=1}^{d} w_i \; x_i\right) + \quad w_0 \quad\right)$$

Introduce an artificial coordinate $x_0 = 1$:

$$h(\mathbf{x}) = \text{sign}\left(\sum_{i=0}^{d} w_i \; x_i\right)$$

In vector form, the perceptron implements

$$h(\mathbf{x}) = \text{sign}(\mathbf{w}^{\intercal}\mathbf{x})$$



'linearly separable' data

A simple learning algorithm - PLA

The perceptron implements

$$h(\mathbf{x}) = \text{sign}(\mathbf{w}^\intercal \mathbf{x})$$

Given the training set:

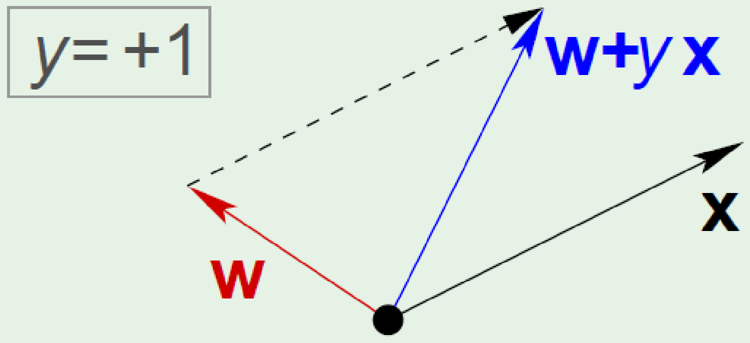
$$(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), \cdots, (\mathbf{x}_N, y_N)$$

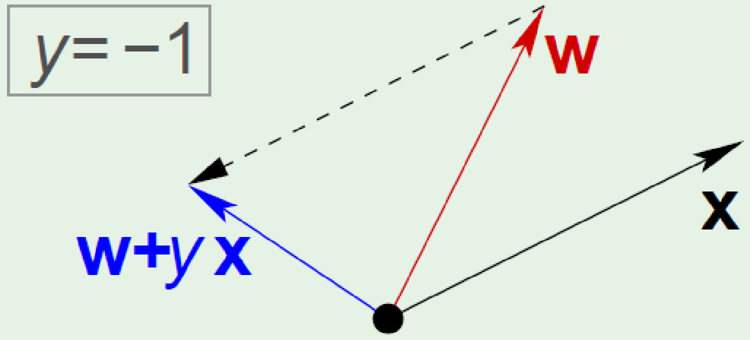
pick a misclassified point:

$$\text{sign}(\mathbf{w}^\intercal \mathbf{x}_n) \neq y_n$$

and update the weight vector:

$$\boxed{\mathbf{w} \leftarrow \mathbf{w} + y_n \mathbf{x}_n}$$





Iterations of PLA

- One iteration of the PLA:

$$\mathbf{w} \leftarrow \mathbf{w} + y\mathbf{x}$$

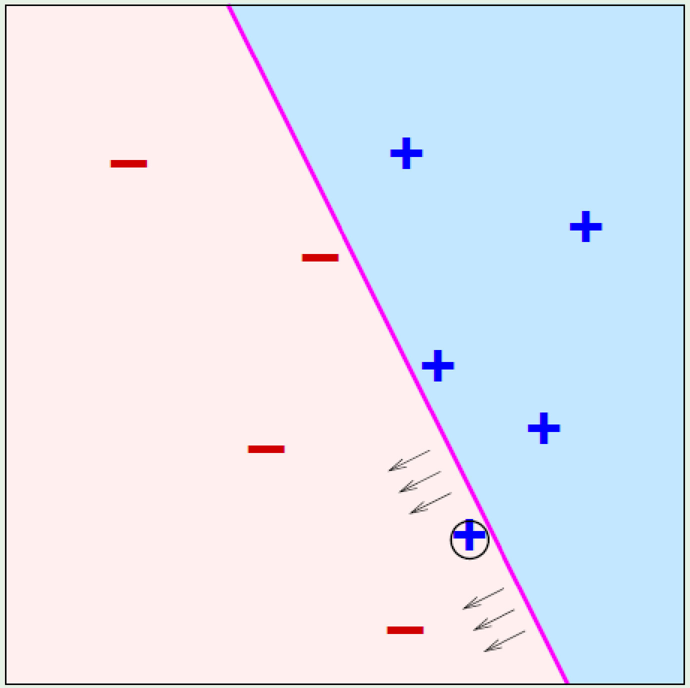
where $(\mathbf{x}, y)$ is a misclassified training point.

- At iteration $t = 1, 2, 3, \cdots$, pick a misclassified point from

$$(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), \cdots, (\mathbf{x}_N, y_N)$$

and run a PLA iteration on it.

- That's it!



The linear regression algorithm

1: Construct the matrix X and the vector $\mathbf{y}$ from the data set $(\mathbf{x}_1, y_1), \cdots, (\mathbf{x}_N, y_N)$ as follows

$$\underbrace{\mathrm{X} = \begin{bmatrix} -\mathbf{x}_1^{\top}- \\ -\mathbf{x}_2^{\top}- \\ \vdots \\ -\mathbf{x}_N^{\top}- \end{bmatrix}}_{\text{input data matrix}}, \qquad \underbrace{\mathbf{y} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_N \end{bmatrix}}_{\text{target vector}}.$$

2: Compute the pseudo-inverse $\mathrm{X}^{\dagger} = (\mathrm{X}^{\top}\mathrm{X})^{-1}\mathrm{X}^{\top}$.

3: Return $\mathbf{w} = \mathrm{X}^{\dagger}\mathbf{y}$.

Logistic regression algorithm

Initialize the weights at $t = 0$ to $\mathbf{w}(0)$

for $t = 0, 1, 2, \ldots$ **do**

  Compute the gradient

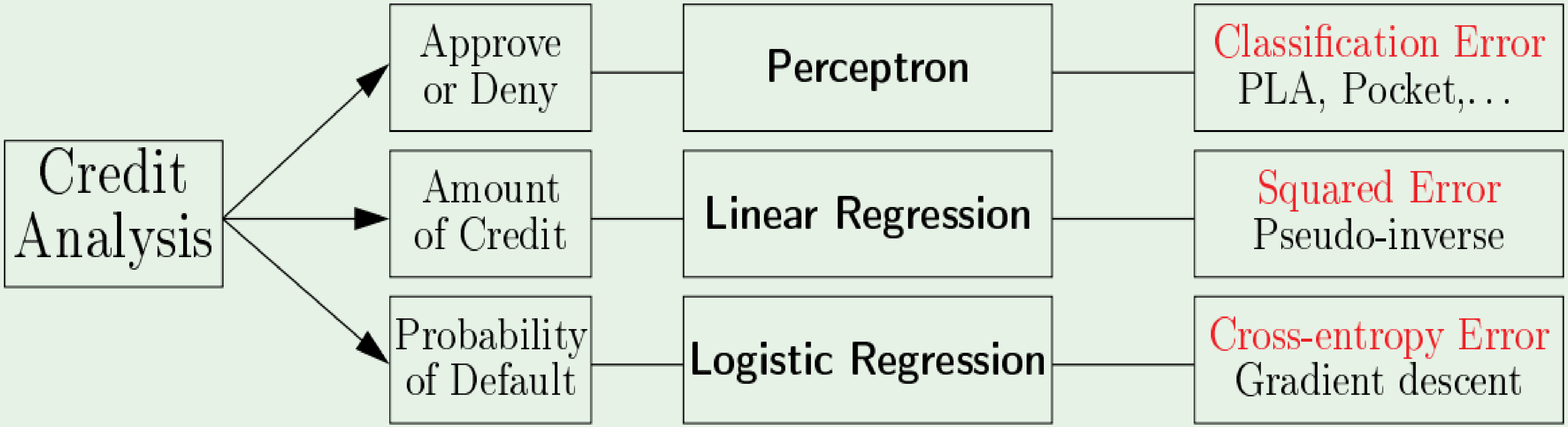
$$\nabla E_{\text{in}} = -\frac{1}{N} \sum_{n=1}^{N} \frac{y_n \mathbf{x}_n}{1 + e^{y_n \mathbf{w}^{\intercal}(t) \mathbf{x}_n}}$$

  Update the weights: $\mathbf{w}(t+1) = \mathbf{w}(t) - \eta \nabla E_{\text{in}}$

  Iterate to the next step until it is time to stop

Return the final weights $\mathbf{w}$

Summary of Linear Models

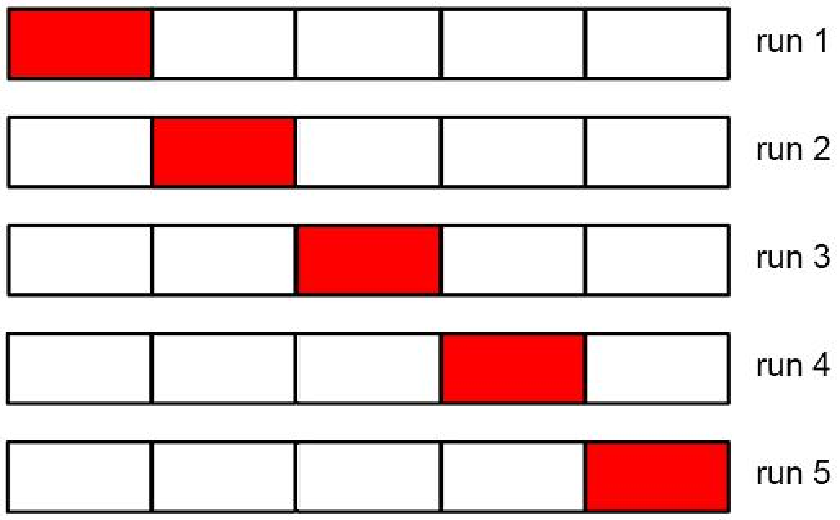


Validation

- Cross Validation
  - We split the training data into K **folds**; then, for each fold k ∈ {1, . . . ,K}, we train on all the folds but the k’th, and test on the k’th, in a round-robin fashion.
  - We then compute the error averaged over all the folds, and use this as a proxy for the test error.

  (Note that each point gets predicted only once, although it will be used for training K−1 times.)

  - It is common to use K = 5; this is called 5-fold CV.



Linear Independence

- The vectors {v1, v2, v3, …, vn} are linearly independent, if $\sum_{j=1}^{n}\left(a_j vj\right) = 0$, if and only if a_j = 0 for all j = 1,…,n
- The rank of a matrix is the maximum number of linearly independent column vectors (or row vectors).
- Nonsingular?

- Rank of an out-product matrix $xy^T \in IR^{m*n}$ with $x \in IR^m$ and y $\in IR^n$

Orthogonal and Orthonormal

- Two vectors x and y are orthogonal, if $x^T y$ = 0
- Can we say one vector x is orthogonal?
- Orthonormal? (set of orthogonal vectors be normalized)
- A matrix is called an orthogonal matrix, if its columns are orthonormal.

Eigenvalues and eigenvectors of a symmetric matrix

- The eigenvectors of a symmetric matrix are mutually orthogonal and its eigenvalues are real.
  - $A \in \mathbb{R}^{n \times n}$ is a symmetric matrix, if $A = A^T$.
- A symmetric matrix $A \in \mathbb{R}^{n \times n}$ can be written in the form $A = U \Lambda U^T$, where the columns of U (U is an orthogonal matrix) are the eigenvectors of A and Λ is a diagonal matrix, the diagonal elements of Λ which are the corresponding eigenvalues of A. This is called the eigendecomposition of A.

Sample Questions

Problem 1.2 Consider the perceptron in two dimensions: $h(\mathbf{x}) = \text{sign}(\mathbf{w}^{\mathrm{T}}\mathbf{x})$ where $\mathbf{w} = [w_0, w_1, w_2]^{\mathrm{T}}$ and $\mathbf{x} = [1, x_1, x_2]^{\mathrm{T}}$. Technically, $\mathbf{x}$ has three coordinates, but we call this perceptron two-dimensional because the first coordinate is fixed at 1.

(a) Show that the regions on the plane where $h(\mathbf{x}) = +1$ and $h(\mathbf{x}) = -1$ are separated by a line. If we express this line by the equation $x_2 = ax_1 + b$, what are the slope a and intercept b in terms of w_0, w_1, w_2?

From h(x) = + 1, we can get $w_1x_1 + w_2x_2 + w_0 > 0$, which means $x_2 \geq -(w_1x_1 + w_0) / w_2$ (if $w_2 \neq 0$)

The boundary is the line given by $x_2 = -\frac{w_1}{w_2}x_1 - \frac{w_0}{w_2}$.

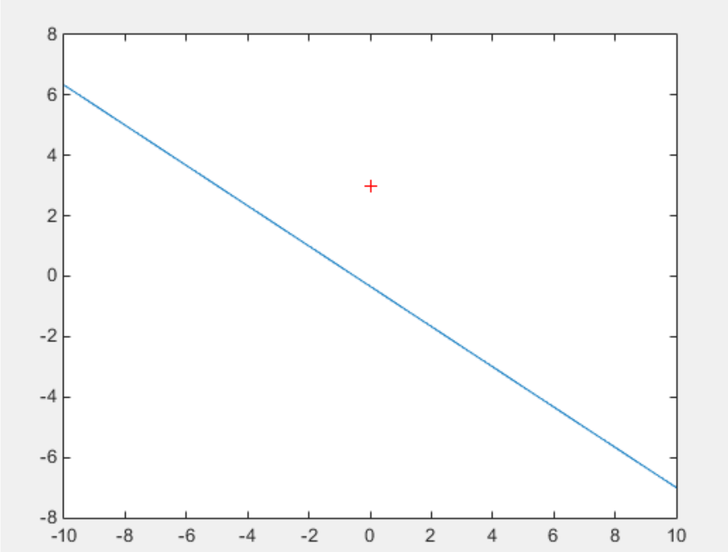
Problem 1.2 Consider the perceptron in two dimensions: $h(\mathbf{x}) = \text{sign}(\mathbf{w}^{\mathrm{T}}\mathbf{x})$ where $\mathbf{w} = [w_0, w_1, w_2]^{\mathrm{T}}$ and $\mathbf{x} = [1, x_1, x_2]^{\mathrm{T}}$. Technically, $\mathbf{x}$ has three coordinates, but we call this perceptron two-dimensional because the first coordinate is fixed at 1.

(b) Draw a picture for the cases $\mathbf{w} = [1, 2, 3]^{\mathrm{T}}$ and $\mathbf{w} = -[1, 2, 3]^{\mathrm{T}}$.

In more than two dimensions, the $+1$ and -1 regions are separated by a *hyperplane*, the generalization of a line.

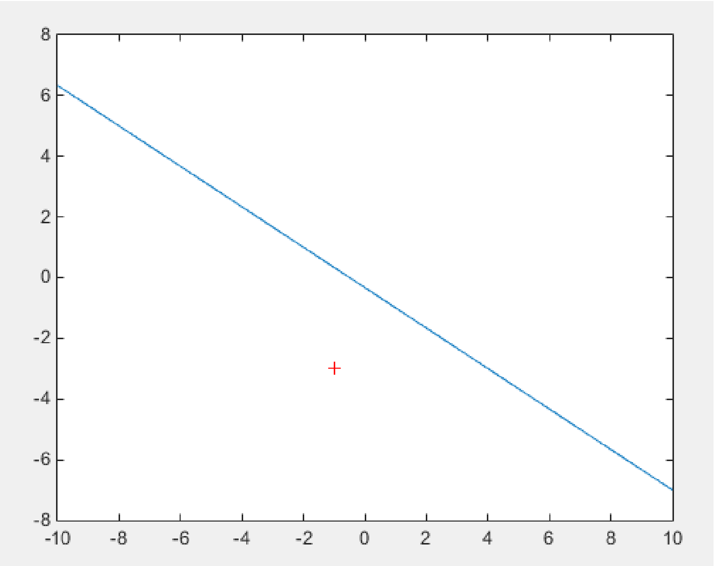
If w = $[1, 2, 3]^T$, we can get x_2 = -2/3 x_1 – 1/3.

$$h(x) = \begin{cases} +1, x_2 > -2/3\ x_1 - 1/3. \\ -1, x_2 \le -2/3\ x_1 - 1/3. \end{cases}$$



If w = $-[1, 2, 3]^T$, we can get x_2 = -2/3 x_1 – 1/3.

$$h(x) = \begin{cases} +1, x_2 < -2/3\ x_1 - 1/3. \\ -1, x_2 \ge -2/3\ x_1 - 1/3. \end{cases}$$



3. (10 points) Given $x \in \mathbb{R}^m$, $y \in \mathbb{R}^n$, show that the rank of matrix xy^T is one.

Outer product generates the matrix whose first row is $x_1(y_1, y_2, \dots, y_n)$, and the i^{th} row is $x_i(y_1, y_2, \dots, y_n)$. So, each row is the vector $(y_1, y_2, \dots, y_n)$ multiplied by scalars, and then every two rows of the matrix are linearly dependent to each other. Hence the rank is 1.

4. (10 points) Given $X = [x_1, x_2, \cdots, x_n] \in \mathbb{R}^{m\times n}$ where $x_i \in \mathbb{R}^m$ for all i, and $Y^T = [y^1, y^2, \cdots, y^n] \in \mathbb{R}^{p\times n}$ where $y^i \in \mathbb{R}^p$ for all i. Show that

$$XY = \sum_{i=1}^{n} x_i (y^i)^T.$$

$(XY)_{ij} = \sum_{k=1}^{n} x_{ki} y_j^k$ $\quad$ $(\sum_{k=1}^{n} x_{ki} (y_j^k)^T)_{ij} = \sum_{k=1}^{n} x_{ki} y_j^k$

So XY = $\sum_{k=1}^{n} x_{ki} (y_j^k)^T$

5. (10 points) Given $X \in \mathbb{R}^{m \times n}$, show that the matrix $X^T X$ is symmetric and positive semi-definite. When is it positive definite?

- A symmetric matrix $A \in \mathbb{R}^{n \times n}$ is positive semi-definite, if and only if $x^T A x \geq 0$, for any $x \in \mathbb{R}^n$.
  - All eigenvalues of A are non-negative.
  - $X^T A X$ for any $X \in \mathbb{R}^{n \times m}$ is positive semi-definite.

$\because (X^T X)^T = X^T X$
$\therefore X^T X$ is symmetric.
$\because \forall y \in R^n, y^T X^T Xy = (Xy)^T (Xy) \geq 0$
$\therefore$ It's positive semidefinite.

- A symmetric matrix $A \in \mathbb{R}^{n \times n}$ is positive definite, if and only if $x^T A x > 0$, for any $0 \neq x \in \mathbb{R}^n$.
  - All eigenvalues of A are positive.
  - All principal submatrices of A are positive definite.
  - All diagonal entries of A are positive.

When rank(X) = n, X is column full rank, it's positive definite.

Show the details on the blackboard.

6. (10 points) Given $g(x,y) = e^x + e^{y^2} + e^{3xy}$, compute $\frac{\partial g}{\partial y}$.

$\frac{\partial g}{\partial y} = 2ye^{y^2} + 3xe^{3xy}$.

7. (25 points) Consider the matrix

$$A = \begin{pmatrix} 2 & 1 & 3 \\ 1 & 1 & 2 \\ 3 & 2 & 5 \end{pmatrix},$$

(a) Compute the eigenvalues and corresponding eigenvectors of A. You are allowed to use Matlab to compute the eigenvectors (but not the eigenvalues).

(b) What is the eigen-decomposition of A?

(c) What is the rank of A?

(d) Is A positive definite? Is A positive semi-definite?

(e) Is A singular?

- A symmetric matrix $A \in \mathbb{R}^{n \times n}$ can be written in the form $A = U \Lambda U^T$, where the columns of U (U is an orthogonal matrix) are the eigenvectors of A and Λ is a diagonal matrix, the diagonal elements of Λ which are the corresponding eigenvalues of A. This is called the eigendecomposition of A.

Exercise 3.6 Cross-entropy error measure

(a) More generally, if we are learning from ∓ 1 data to predict a noisy target $P(y|x)$ with candidate hypothesis h, Show that the maximum likelihood method reduces to the task of finding h that minimizes

$$E_{in}(w) = \sum_{n=1}^{N} [\![y_n = +1]\!] In \frac{1}{h(x_n)} + [\![y_n = -1]\!]\ In \frac{1}{1 - h(x_n)}$$

The likelihood p(y|x) = $\begin{cases} h(x) & for\ y = +1 \\ 1 - h(x) & for\ y = -1 \end{cases}$ also p(y|x) = $\theta(y\, w^T x)$

max $\prod_{n=1}^{N} p(y_n|x_n)$ <==> max ln($\prod_{n=1}^{N} p(y_n|x_n)$)

<==> max $\sum_{n=1}^{N} \ln p(y_n|x_n)$ <==> min - $\sum_{n=1}^{N} \ln p(y_n|x_n)$

<==> min $\sum_{n=1}^{N} ln \frac{1}{\theta(y_n w^T x_n)}$ = min $\sum_{n=1}^{N} [y_n = +1]\ ln \frac{1}{h(x_n)}$ + $[y_n = -1]\ ln \frac{1}{1-h(x_n)}$

Why called cross-entropy error:

For two probability distributions {p, 1-p} and {q, 1-q} with binary outcomes, the cross-entropy (from information theory) is

$$p log \frac{1}{q} + (1-p) log \frac{1}{1-q}$$

The In-sample error in part (a) corresponds to cross-entropy error measure on the data point (x_n, y_n),
with $p = [\![y_n = +1]\!]$ and $q = h(x_n)$

(b) For the case $h(x) = \theta(w^T w)$, argue that minimizing the in-sample error in part (a) is equivalent to minimizing the one in (3.9).

$$E_{in} = \frac{1}{N}\sum_{n=1}^{N} In(1 + e^{-y_n w^T x_n})$$

max $\prod_{n=1}^{N} p(y_n|x_n)$ <==> max ln($\prod_{n=1}^{N} p(y_n|x_n)$) <==> min - $\frac{1}{N}\sum_{n=1}^{N} \ln p(y_n|x_n)$

<==> min $\frac{1}{N}\sum_{n=1}^{N} \ln(\frac{1}{p(y_n|x_n)})$ <==> min $\frac{1}{N}\sum_{n=1}^{N} \ln(\frac{1}{\theta(y_n w^T x_n)})$

<==> min $\frac{1}{N}\sum_{n=1}^{N} \ln(1 + e^{-y_n w^T x_n})$

Exercise 3.7

- A correct example means $y_n$has same sign of $w^T x$, then $-y_n w^T x_n < 0$ and $\theta(-y_n w^T x_n)$≈0,so this point contributes little to gradient. On the other hand, for misclassified point, $-y_n w^T x_n > 0$ and $\theta(-y_n w^T x_n)$≈1 . This contributes to the gradient.

2. (20 points) Recall the objective function for linear regression can be expressed as

$$E(w) = \frac{1}{N}\|Xw - y\|^2,$$

as in Equation (3.3) of LFD. Minimizing this function with respect to w leads to the optimal w as $(X^T X)^{-1} X^T y$. This solution holds only when $X^T X$ is nonsingular. To overcome this problem, the following objective function is commonly minimized instead:

$$E_2(w) = \|Xw - y\|^2 + \lambda\|w\|^2,$$

where $\lambda > 0$ is a user-specified parameter. Please do the following:

(a) (10 points) Derive the optimal w that minimize $E_2(w)$.

(a) min $E_2(w) = || xw - y||^2 + \lambda\, ||W||^2 = w^T x^T x\, w - 2\, w^T x^T y + y^T y + \lambda\, w^T w$

$\frac{\partial E_2(w)}{\partial w} = (x^T x + x^T x)\, w - 2x^T y + 2\,\lambda\, I\, w = 0$, so we can get $w = (x^T x + \lambda\, I)^{-1} x^T y$

2. (20 points) Recall the objective function for linear regression can be expressed as

$$E(w) = \frac{1}{N}\|Xw - y\|^2,$$

as in Equation (3.3) of LFD. Minimizing this function with respect to w leads to the optimal w as $(X^T X)^{-1} X^T y$. This solution holds only when $X^T X$ is nonsingular. To overcome this problem, the following objective function is commonly minimized instead:

$$E_2(w) = \|Xw - y\|^2 + \lambda\|w\|^2,$$

where $\lambda > 0$ is a user-specified parameter. Please do the following:

(b) (10 points) Explain how this new objective function can overcome the singularity problem of $X^T X$.

- A symmetric matrix $A \in \mathbb{R}^{n\times n}$ is positive definite, if and only if $x^T A x > 0$, for any $0 \neq x \in \mathbb{R}^n$.
  - All eigenvalues of A are positive.
  - All principal submatrices of A are positive definite.
  - All diagonal entries of A are positive.

(b) if $\lambda > 0$, since $\forall\, y \neq 0, y^T(x^T x + \lambda I)y$ = $||Xy||^2$ + λ $||y||^2 > 0$, so $x^T x + \lambda I$ positive definite => full rank => is non-singular.

3. (35 points) In logistic regression, the objective function can be written as

$$E(w) = \frac{1}{N}\sum_{n=1}^{N} \ln\left(1 + e^{-y_n w^T x_n}\right).$$

Please

(a) (10 points) Compute the first-order derivative $\nabla E(w)$. You will need to provide the intermediate steps of derivation.

$$\nabla_w E_{in}(w) = \frac{1}{N}\sum_{i=1}^{N} \frac{1}{1+e^{-y_n w^T x_n}} \nabla_w (1+e^{-y_n w^T x_n})$$

$$= \frac{1}{N}\sum_{i=1}^{N} \frac{e^{-y_n w^T x_n}}{1+e^{-y_n w^T x_n}} \nabla_w (-y_n w^T x_n)$$

$$= \frac{1}{N}\sum_{i=1}^{N} \frac{e^{-y_n w^T x_n}}{1+e^{-y_n w^T x_n}} \nabla_w (-y_n x_n)$$

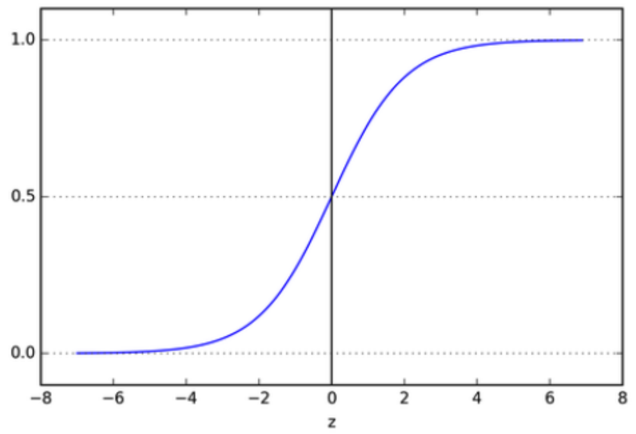
$$= \frac{1}{N}\sum_{i=1}^{N} \frac{y_n x_n}{1+e^{y_n w^T x_n}}$$

$$= \frac{1}{N}\sum_{i=1}^{N} -y_n x_n q(-y_n w^T x_n)$$

(b) (10 points) Once the optimal w is obtain, it will be used to make predictions as follows:

$$\text{Predicted class of } x = \begin{cases} 1 & \text{if } \theta(w^T x) \geq 0.5 \\ -1 & \text{if } \theta(w^T x) < 0.5 \end{cases}$$

where the function $\theta(z) = \frac{1}{1+e^{-z}}$ looks like



Explain why the decision boundary of logistic regression is still linear, though the linear signal $w^T x$ is passed through a nonlinear function θ to compute the outcome of prediction.

The decision can also be seen as: when $w^T x > 0$ then classify x as 1. Otherwise, classify x as -1. Since the $w^T x = 0$ is a linear boundary, the decision boundary of logistic regression is still linear.

(c) (5 points) Is the decision boundary still linear if the prediction rule is changed to the following? Justify briefly.

$$\text{Predicted class of } x = \begin{cases} 1 & \text{if } \theta(w^T x) \geq 0.9 \\ -1 & \text{if } \theta(w^T x) < 0.9 \end{cases}$$

(d) (10 points) In light of your answers to the above two questions, what is the essential property of logistic regression that results in the linear decision boundary?

- Yes. Although the threshold is changed from 0.5 to 0.9, the property in (b) still holds.

- The objective function of logistic regression is simultaneously increasing.

Programming exercises

- HW1 – perceptron for binary classification
- HW2 – logistic regression for binary classification

Dataset information

Original dataset:

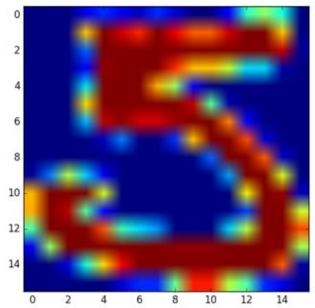
	sample numbers	image size	labels
Training dataset	1561	16*16	1 or 5
Testing dataset	424	16*16	1 or 5

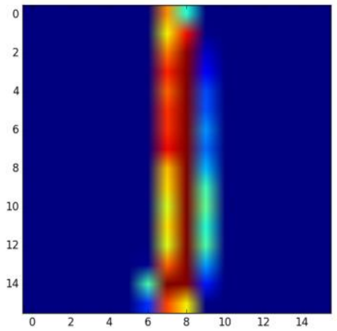
Extracted features:

	sample numbers	features	labels
Training dataset	1561	(1, x1, x2)	1 for 1 or -1 for 5
Testing dataset	424	(1, x1, x2)	1 for 1 or -1 for 5

Show images

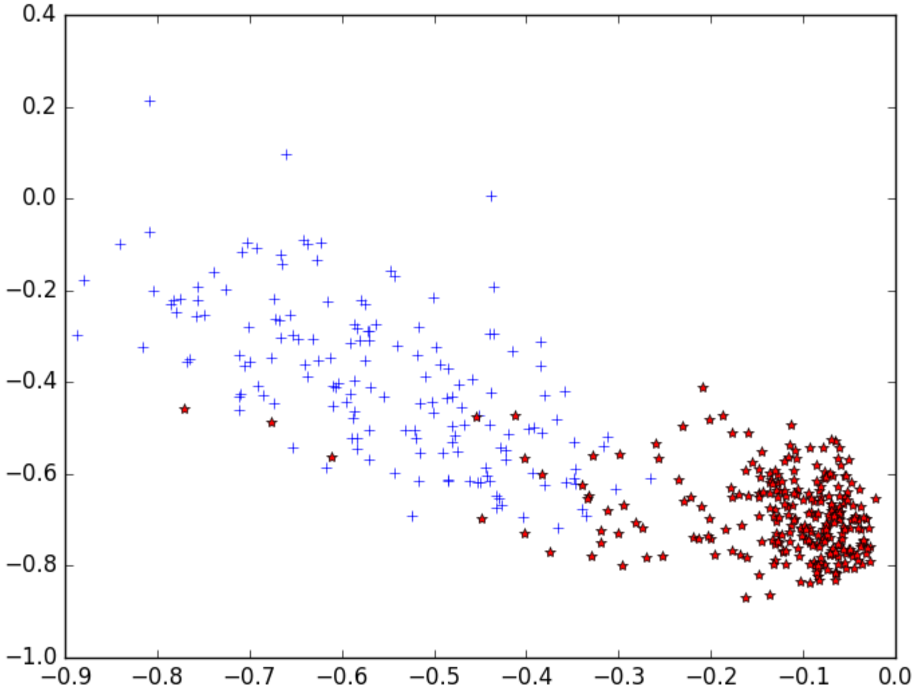
```python
def show_images(data):
    '''
    This function is used for plot image and save it.

    Args:
    data: Two images from train data with shape (2, 16, 16). The shape represents
    total 2
            images and each image has size 16 by 16.

    Returns:
        Do not return any arguments, just save the images you plot for your report.
    '''
    for i in range(len(data)):
        plt.clf()
        fig = plt.figure()
        plt.imshow(data[i,:])
        fig.savefig('traindata%d'%(i))
```





Show features

```python
def show_features(data, label):
    '''
    This function is used for plot a 2-D scatter
    plot of the features and save it.

    Args:
    data: train features with shape (1561, 2).
        The shape represents total 1561 samples and
        each sample has 2 features.
    label: train data's label with shape (1561,1).
        1 for digit number 1 and -1 for digit number 5.

    Returns:
    Do not return any arguments, just save
    the 2-D scatter plot of the features
    you plot for your report.
    '''
    n, _ = data.shape
    fig = plt.figure()
    for i in range(n):
        if label[i] == 1:
            plt.plot(data[i,0], data[i,1], 'r*')
        else:
            plt.plot(data[i,0], data[i,1], 'b+')
    fig.savefig('trainfeature')
```



Perceptron

A simple iterative method.

1: $\mathbf{w}(1) = \mathbf{0}$
2: **for** iteration $t = 1, 2, 3, \ldots$
3: the weight vector is $\mathbf{w}(t)$.
4: From $(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_N, y_N)$ pick any misclassified example.
5: Call the misclassified example $(\mathbf{x}_*, y_*)$,

$$\mathrm{sign}\left(\mathbf{w}(t) \bullet \mathbf{x}_*\right) \neq y_*.$$

6: Update the weight:

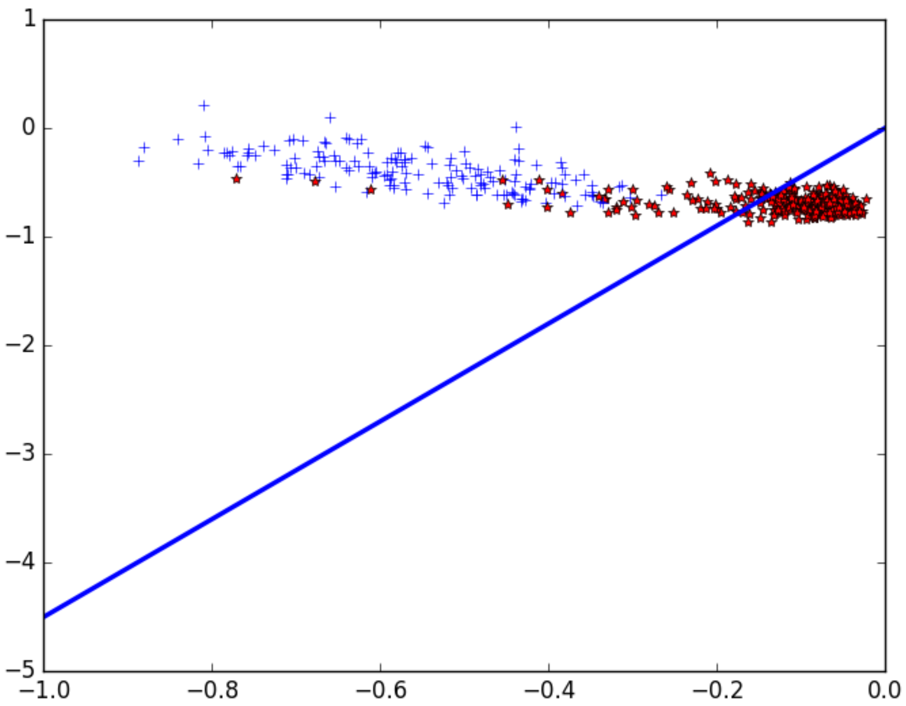
$$\mathbf{w}(t+1) = \mathbf{w}(t) + y_* \mathbf{x}_*.$$

7: $t \leftarrow t + 1$

```python
def perceptron(data, label, max_iter, learning_rate):
    '''
    The perceptron classifier function.

    Args:
    data: train data with shape (1561, 3), which means 1561 samples and
          each sample has 3 features.(1, symmetry, average internsity)
    label: train data's label with shape (1561,1).
           1 for digit number 1 and -1 for digit number 5.
    max_iter: max iteration numbers
    learning_rate: learning rate for weight update
    Returns:
        w: the seperater with shape (1, 3). You must initilize
        it with w = np.zeros((1,d))
    '''
    w = np.zeros((1,3))
    for i in range(max_iter):
        missample = []
        for j in range(len(data)):
            if sign(np.dot(data[j],np.transpose(w)))!=label[j]:
                missample.append((data[j], label[j]))
        if len(missample) == 0:
            break
        else:
            index = np.random.randint(len(missample))
            w = w + learning_rate*missample[index][0]*missample[index][1]
    return w
```

Show result

```python
def show_result(data, label, w):
    '''
    This function is used for plot the test data
    with the separators and save it.

    Args:
    data: test features with shape (424, 2). The shape
          represents total 424 samples and
          each sample has 2 features.
    label: test data's label with shape (424,1).
           1 for digit number 1 and -1 for digit number 5.

    Returns:
    Do not return any arguments, just save the
        image you plot for your report.
    '''
    show_features(data, label)
    w = np.reshape(w, (1,3))
    x = np.linspace(-1,0,2)
    y = -w[0,1]/w[0,2]*x -w[0,0]/w[0,2]
    plt.plot(x, y, linewidth=2.5, linestyle="-")
    plt.savefig('result')
```



Logistic regression

$$E_{\text{in}}(\mathbf{w}) = \frac{1}{N}\sum_{n=1}^{N} \ln(1 + e^{-y_n \cdot \mathbf{w}^{\mathsf{T}}\mathbf{x}})$$

Initialize at step $t = 0$ to $\mathbf{w}(0)$.
for $t = 0, 1, 2, \ldots$ **do**
  Compute the gradient

$$\mathbf{g}_t = \nabla E_{\text{in}}(\mathbf{w}(t)). \longleftarrow \text{(Ex. 3.7 in LFD)}$$

  Move in the direction $\mathbf{v}_t = -\mathbf{g}_t$.
  Update the weights:

$$\mathbf{w}(t+1) = \mathbf{w}(t) + \eta \mathbf{v}_t.$$

  Iterate 'until it is time to stop'.
end for
Return the final weights.

```python
def gradient(x, y, w):
    result = -y*x/(1+np.exp(y*np.dot(np.transpose(w),x)))
    return result[:,np.newaxis]


def logistic_regression(data, label, max_iter, learning_rate):
    '''
    The logistic regression classifier function.

    Args:
    data: train data with shape (1561, 3),
        which means 1561 samples and
        each sample has 3 features.(1, symmetry,
        average internsity)
    label: train data's label with shape (1561,1).
           1 for digit number 1 and -1 for digit number 5.
    max_iter: max iteration numbers
    learning_rate: learning rate for weight update

    Returns:
        w: the seperater with shape (3, 1). You must
        initilize it with w = np.zeros((d,1))
    '''
    n, d = data.shape
    w = np.zeros((d,1))
    for i in range(max_iter):
        gt = np.zeros((d,1))
        for j in range(n):
            gt = gt + gradient(np.transpose(data[j,:]), label[j],
        w = w - 1/n * learning_rate*gt
    print("w",w)
    return w
```

Accuracy

```python
def lrclassifier(x):
    return 1 if 1.0/(1+np.exp(-x))>0.5 else -1


def accuracy(x, y, w):
    '''
    This function is used to compute accuracy of a logsitic regression model.

    Args:
    x: input data with shape (n, d), where n
        represents total data samples and d represents
        total feature numbers of a certain data sample.
    y: corresponding label of x with shape(n, 1),
        where n represents total data samples.
    w: the seperator learnt from logistic regression
        function with shape (d, 1),
        where d represents total feature numbers
        of a certain data sample.

    Return
        accuracy: total percents of correctly classified
        samples. Set the threshold as 0.5,
        which means, if the predicted probability > 0.5,
        classify as 1; Otherwise, classify as -1.
    '''

    n, _ = x.shape
    mistakes = 0
    for i in range(n):
        if lrclassifier(np.dot(np.transpose(w),np.transpose(x[i,:]))) != y[i]:
            mistakes += 1
    return (n-mistakes)/n
```

Third order transformation

```
def thirdorder(data):
    '''
    This function is used for a 3rd order polynomial transform of the data.
    Args:
    data: input data with shape (:, 3) the first dimension represents
          total samples (training: 1561; testing: 424) and the
          second dimesion represents total features.

    Return:
        result: A numpy array format new data with shape (:,10), which using
        a 3rd order polynomial transformation to extend the feature numbers
        from 3 to 10.
        The first dimension represents total samples (training: 1561; testing: 424)
        and the second dimesion represents total features.
    '''
    n, _ = data.shape
    result = []
    for i in range(n):
        x1, x2 = data[i,0], data[i,1]
        order3 = [1, x1, x2, x1**2, x1*x2, x2**2, x1**3, x1**2*x2, x1*x2**2, x2**3]
        result.append(order3)
    return np.array(result)
```

1^{st} or 3^{rd} order?

- If 1^{st}: the accuracy between this two are similar and 3^{rd} order increases computation time.

- If 3^{rd}: focus on the accuracy is better than the 1^{st} one.