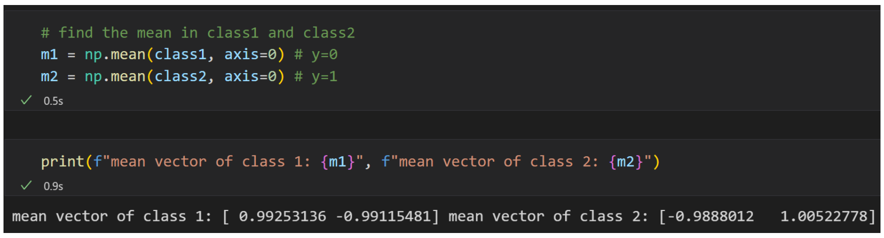
109550031_HW2

Part. 1, Coding

1. Compute the mean vectors mi (i=1, 2) of each 2 classes on training data



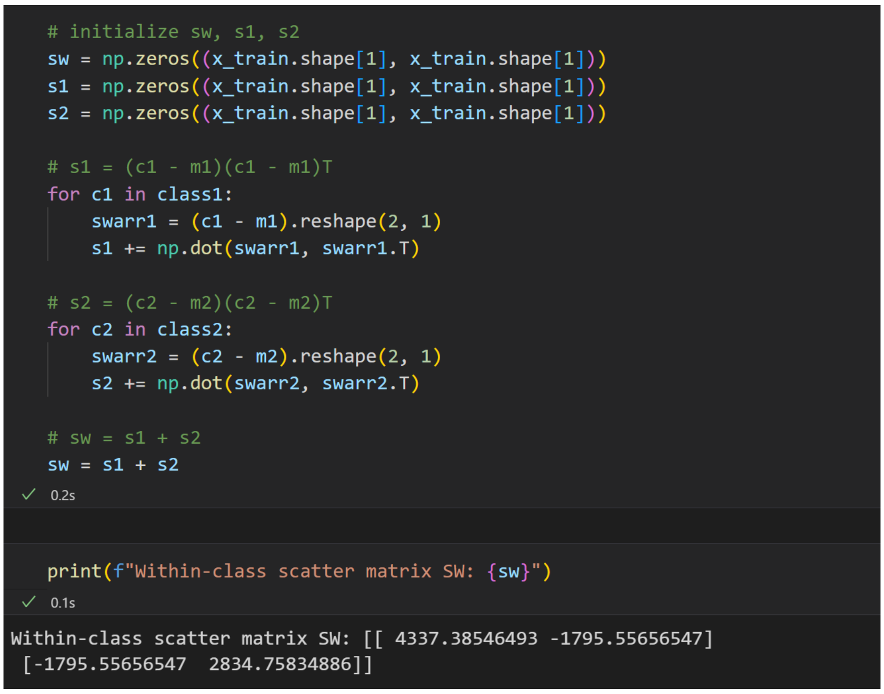
```python
# find the mean in class1 and class2
m1 = np.mean(class1, axis=0) # y=0
m2 = np.mean(class2, axis=0) # y=1
```

```python
print(f"mean vector of class 1: {m1}", f"mean vector of class 2: {m2}")
```

```
mean vector of class 1: [ 0.99253136 -0.99115481] mean vector of class 2: [-0.9888012   1.00522778]
```

calculate mean vector of two class

2. Compute the within-class scatter matrix sw on training data



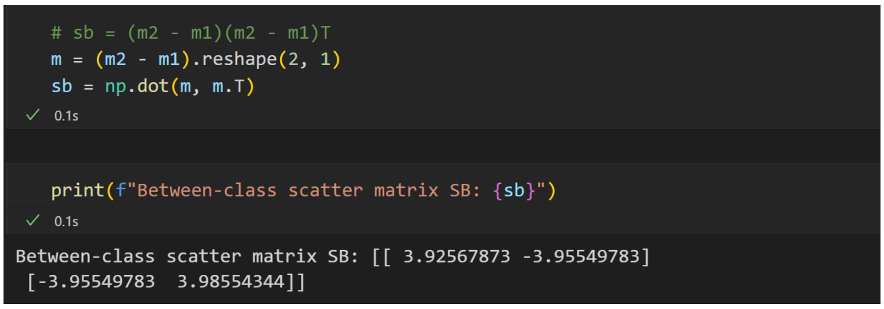
```python
# initialize sw, s1, s2
sw = np.zeros((x_train.shape[1], x_train.shape[1]))
s1 = np.zeros((x_train.shape[1], x_train.shape[1]))
s2 = np.zeros((x_train.shape[1], x_train.shape[1]))

# s1 = (c1 - m1)(c1 - m1)T
for c1 in class1:
    swarr1 = (c1 - m1).reshape(2, 1)
    s1 += np.dot(swarr1, swarr1.T)

# s2 = (c2 - m2)(c2 - m2)T
for c2 in class2:
    swarr2 = (c2 - m2).reshape(2, 1)
    s2 += np.dot(swarr2, swarr2.T)

# sw = s1 + s2
sw = s1 + s2
```

```python
print(f"Within-class scatter matrix SW: {sw}")
```

```
Within-class scatter matrix SW: [[ 4337.38546493 -1795.55656547]
 [-1795.55656547  2834.75834886]]
```

calculate s1 an s2 respectively and within-class scatter matrix sw = s1 + s2

3. Compute the between-class scatter matrix sb on training data



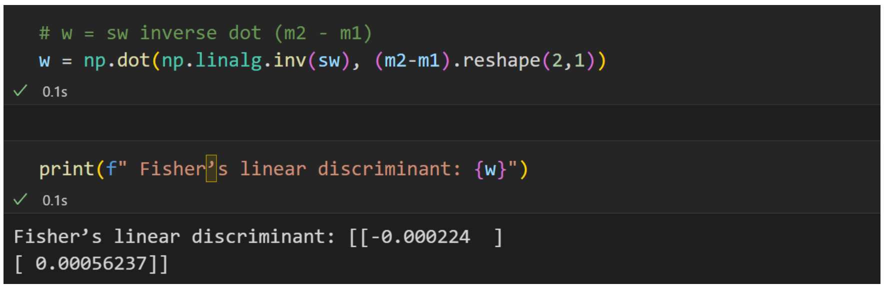
```python
# sb = (m2 - m1)(m2 - m1)T
m = (m2 - m1).reshape(2, 1)
sb = np.dot(m, m.T)
```

```python
print(f"Between-class scatter matrix SB: {sb}")
```

```
Between-class scatter matrix SB: [[ 3.92567873 -3.95549783]
 [-3.95549783  3.98554344]]
```

calculate between-class scatter matrix sb

4. Compute the Fisher’s linear discriminant w on training data



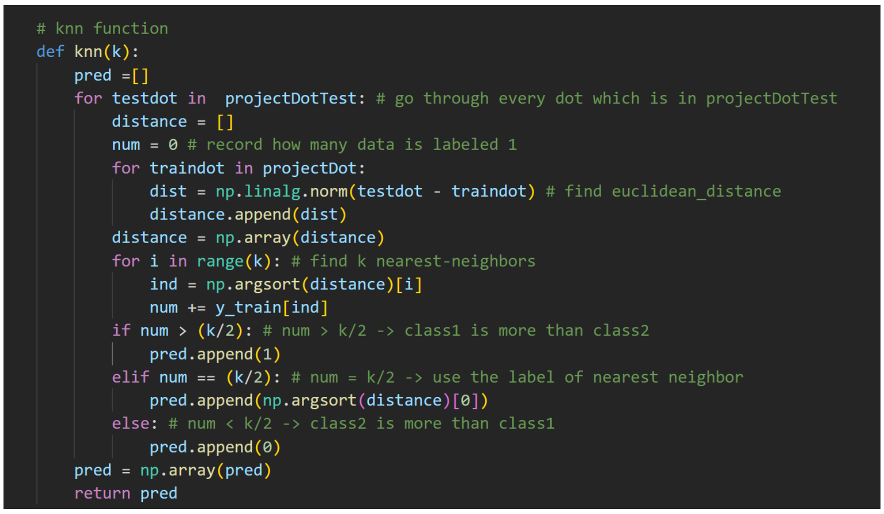
```
# w = sw inverse dot (m2 - m1)
w = np.dot(np.linalg.inv(sw), (m2-m1).reshape(2,1))
```
✓ 0.1s

```
print(f" Fisher’s linear discriminant: {w}")
```
✓ 0.1s

```
Fisher’s linear discriminant: [[-0.000224  ]
[ 0.00056237]]
```

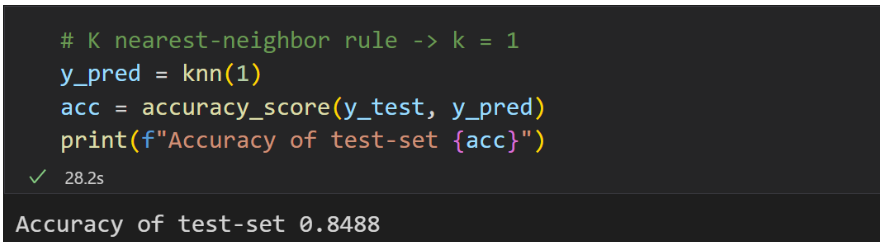
calculate Fisher’s linear discriminant w

5. Project the testing data by Fisher’s linear discriminant to get the class prediction by K-Nearest-Neighbor rule and report the accuracy score on testing data



```
# knn function
def knn(k):
    pred =[]
    for testdot in  projectDotTest: # go through every dot which is in projectDotTest
        distance = []
        num = 0 # record how many data is labeled 1
        for traindot in projectDot:
            dist = np.linalg.norm(testdot - traindot) # find euclidean_distance
            distance.append(dist)
        distance = np.array(distance)
        for i in range(k): # find k nearest-neighbors
            ind = np.argsort(distance)[i]
            num += y_train[ind]
        if num > (k/2): # num > k/2 -> class1 is more than class2
            pred.append(1)
        elif num == (k/2): # num = k/2 -> use the label of nearest neighbor
            pred.append(np.argsort(distance)[0])
        else: # num < k/2 -> class2 is more than class1
            pred.append(0)
    pred = np.array(pred)
    return pred
```

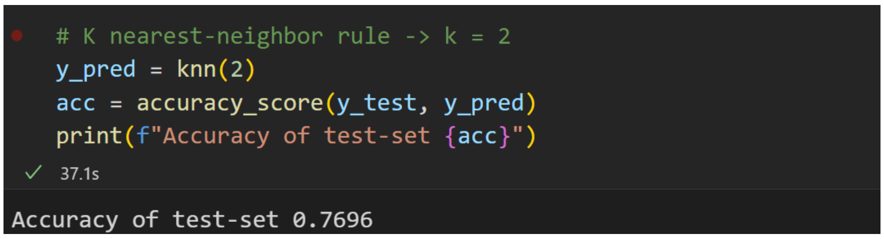
knn function which will return y_pred



```
# K nearest-neighbor rule -> k = 1
y_pred = knn(1)
acc = accuracy_score(y_test, y_pred)
print(f"Accuracy of test-set {acc}")
```
✓ 28.2s

```
Accuracy of test-set 0.8488
```

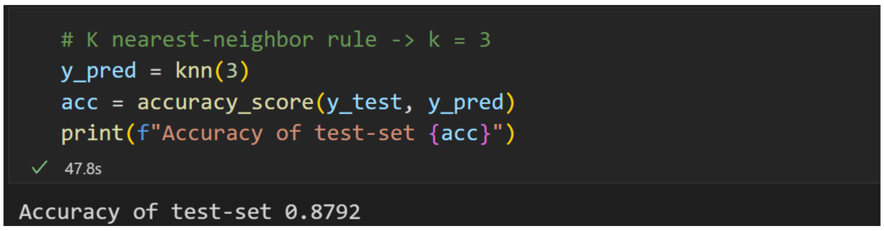
k = 1 → accuracy = 0.8488



```
# K nearest-neighbor rule -> k = 2
y_pred = knn(2)
acc = accuracy_score(y_test, y_pred)
print(f"Accuracy of test-set {acc}")
```
✓ 37.1s

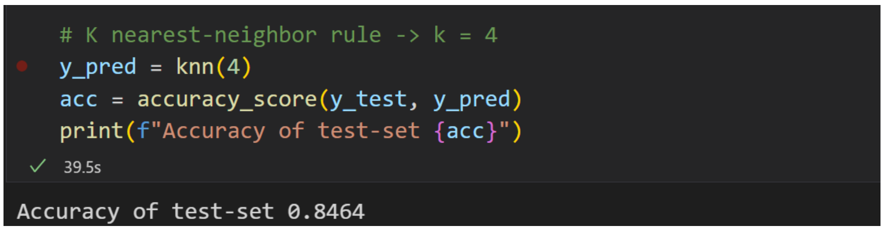
```
Accuracy of test-set 0.7696
```

k = 2 → accuracy = 0.7696



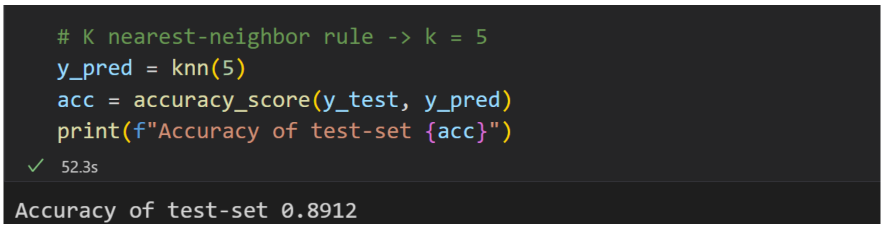
```
# K nearest-neighbor rule -> k = 3
y_pred = knn(3)
acc = accuracy_score(y_test, y_pred)
print(f"Accuracy of test-set {acc}")
✓ 47.8s

Accuracy of test-set 0.8792
```

k = 3 → accuracy = 0.8792



```
# K nearest-neighbor rule -> k = 4
y_pred = knn(4)
acc = accuracy_score(y_test, y_pred)
print(f"Accuracy of test-set {acc}")
✓ 39.5s

Accuracy of test-set 0.8464
```

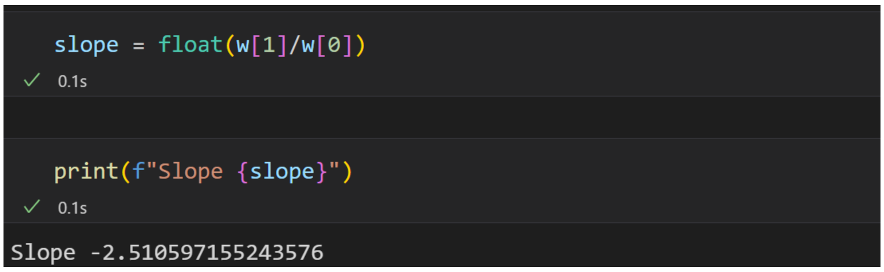
k = 4 → accuracy = 0.8464



```
# K nearest-neighbor rule -> k = 5
y_pred = knn(5)
acc = accuracy_score(y_test, y_pred)
print(f"Accuracy of test-set {acc}")
✓ 52.3s

Accuracy of test-set 0.8912
```

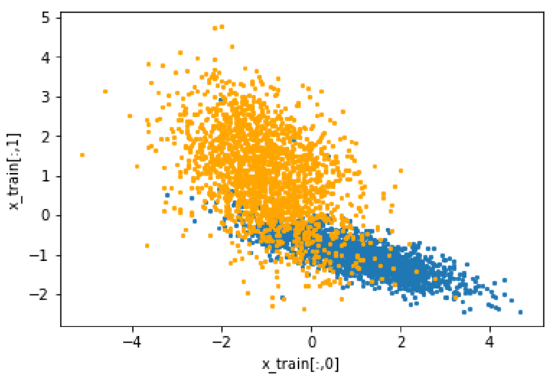
k = 5 → accuracy = 0.8912

6. Plot the 1) best projection line on the training data and show the slope and intercept on the title 2) colorize the data with each class 3) project all data points on your projection line.

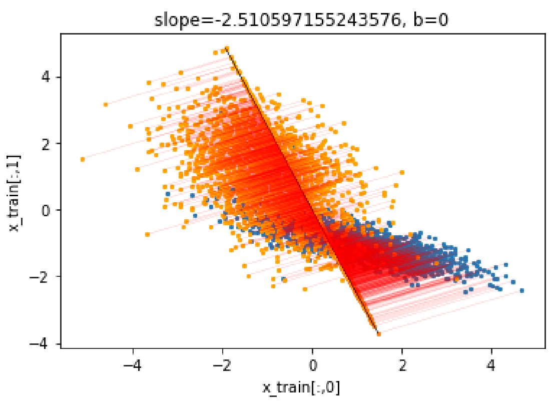


```
slope = float(w[1]/w[0])
✓ 0.1s

print(f"Slope {slope}")
✓ 0.1s

Slope -2.510597155243576
```

calculate the slope



training data



best projection line and all data points which are projected on the projection line

Part. 2, Questions

1. What's the difference between the Principle Component Analysis and Fisher’s Linear Discriminant?

The main difference is that the Fisher’s Linear Discriminant is supervised whereas Principle Component Analysis is unsupervised. Principle Component Analysis ignore class label, it focuses on capturing the direction of maximum variation in the data set. In contrast to Principle Component Analysis, Fisher’s Linear Discriminant focuses on finding a feature subspace that maximizes the separability between the groups.

2. Please explain in detail how to extend the 2-class FLD into multi-class FLD (the number of classes is greater than two).

When the number of classes(K) is greater than two, we need generalization forms for the within-class and between-class covariance matrices.

within-class covariance matrix (Sw)

Take the sum of the matrix-multiplication between the centralized input values and their transpose for each class.

between-class covariance (Sb)

For each class k=1,2,3,…,K, take the outer product of the local class mean(mk) and global mean(m). Then, scale it by the number of records in class k.

Fisher’s linear discriminant (w)

The optimal w is the eigenvector of $Sw^{-1}Sb$ that corresponds to the largest eigenvalue.

$$S_W = \sum_{k=1}^{K} S_k$$

$$S_k = \sum_{n \in C_k} (x_n - m_k)(x_n - m_k)^T$$

with-class covariance

$$S_B = \sum_{k=1}^{K} N_k (m_k - m)(m_k - m)^T$$

between-class covariance

$$W = max_{D'}(eig(S_W^{-1} S_B))$$

Fisher’s linear discriminant

Moreover, we can’t directly extend the objective to learn a multi-dimensional projection. A choice for the objective is

$$J(\mathbf{w}) = \mathrm{Tr}\left\{(\mathbf{W}\mathbf{S}_{\mathrm{W}}\mathbf{W}^{\mathrm{T}})^{-1}(\mathbf{W}\mathbf{S}_{\mathrm{B}}\mathbf{W}^{\mathrm{T}})\right\}$$

3. By making use of Eq (1) ~ Eq (5), show that the Fisher criterion Eq (6) can be written in the form Eq (7).

$$y = \mathbf{w}^{\mathrm{T}}\mathbf{x} \qquad \text{Eq (1)}$$

$$\mathbf{m}_1 = \frac{1}{N_1}\sum_{n\in\mathcal{C}_1}\mathbf{x}_n \qquad \mathbf{m}_2 = \frac{1}{N_2}\sum_{n\in\mathcal{C}_2}\mathbf{x}_n \qquad \text{Eq (2)}$$

$$m_2 - m_1 = \mathbf{w}^{\mathrm{T}}(\mathbf{m}_2 - \mathbf{m}_1) \qquad \text{Eq (3)}$$

$$m_k = \mathbf{w}^{\mathrm{T}}\mathbf{m}_k \qquad \text{Eq (4)}$$

$$s_k^2 = \sum_{n\in\mathcal{C}_k}(y_n - m_k)^2 \qquad \text{Eq (5)}$$

$$J(\mathbf{w}) = \frac{(m_2 - m_1)^2}{s_1^2 + s_2^2} \qquad \text{Eq (6)}$$

$$J(\mathbf{w}) = \frac{\mathbf{w}^{\mathrm{T}}\mathbf{S}_{\mathrm{B}}\mathbf{w}}{\mathbf{w}^{\mathrm{T}}\mathbf{S}_{\mathrm{W}}\mathbf{w}} \qquad \text{Eq (7)}$$

$$S_W = \sum_{n\in C_1}(x_n - m_1)(x_n - m_1)^T + \sum_{n\in C_2}(x_n - m_2)(x_n - m_2)^T$$

$$S_B = (m_2 - m_1)(m_2 - m_1)^T$$

$$J(w) = \frac{(m_2 - m_1)^2}{s_1^2 + s_2^2} = \frac{(m_2 - m_1)^2}{\sum_{n\in C_1}(y_n - m_1)^2 + \sum_{n\in m_2}(y_n - m_2)^2}$$

$$= \frac{[W^T(m_2 - m_1)]^2}{\sum_{n\in C_1}(W^T x_n - W^T m_1)^2 + \sum_{n\in C_2}(W^T x_n - W^T m_2)^2}$$

$$= \frac{(W^T)^2 \cdot (m_2 - m_1)^2}{(W^T)^2\left[\sum_{n\in C_1}(x_n - m_1)^2 + \sum_{n\in C_2}(x_n - m_2)^2\right]}$$

$$= \frac{W^T \cdot W^T \cdot S_B}{W^T \cdot W^T \cdot S_W}$$

$$= \frac{W^T \cdot S_B \cdot W}{W^T \cdot S_W \cdot W} \quad \#$$

4. Show the derivative of the error function Eq (8) with respect to the activation ak for an output unit having a logistic sigmoid activation function satisfies Eq (9).

$$E(\mathbf{w}) = -\sum_{n=1}^{N} \{t_n \ln y_n + (1-t_n)\ln(1-y_n)\} \qquad \text{Eq (8)}$$

$$\frac{\partial E}{\partial a_k} = y_k - t_k \qquad \text{Eq (9)}$$

$$\frac{\partial y_n}{\partial a_n} = \frac{e^{a_n}}{\sum_i e^{a_i}} - \left(\frac{e^{a_n}}{\sum_i e^{a_i}}\right)^2 = y_n(1-y_n)$$

$$\frac{\partial E}{\partial a_n} = -t_n \frac{y_n(1-y_n)}{y_n} + (1-t_n)\frac{y_n(1-y_n)}{(1-y_n)}$$

$$= -t_n(1-y_n) + (1-t_n)\cdot y_n$$

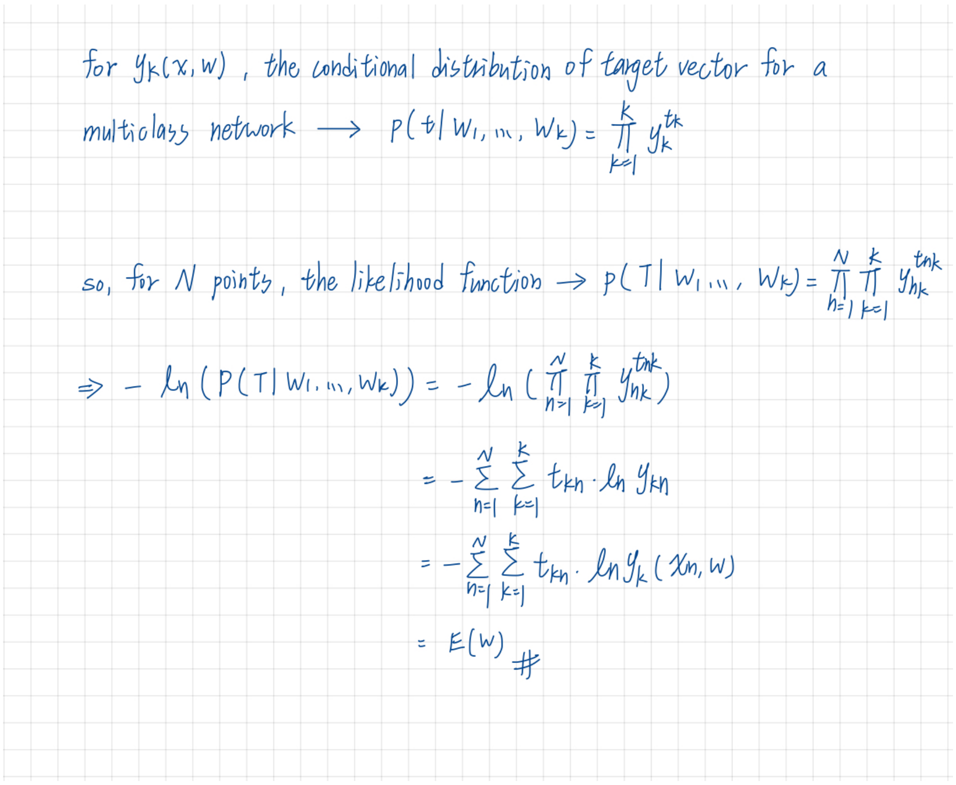
$$= -t_n + t_n\cdot y_n + y_n - t_n\cdot y_n$$

$$= y_n - t_n$$

so, $\frac{\partial E}{\partial a_k} = y_k - t_k$ #

5. Show that maximizing likelihood for a multiclass neural network model in which the network outputs have the interpretation yk(x, w)=p(tk=1 | x) is equivalent to the minimization of the cross-entropy error function Eq (10).

$$E(\mathbf{w}) = -\sum_{n=1}^{N}\sum_{k=1}^{K} t_{kn} \ln y_k(\mathbf{x}_n, \mathbf{w}) \qquad \text{Eq (10)}$$



for $y_k(x, w)$, the conditional distribution of target vector for a multiclass network $\longrightarrow$ $p(t|W_1, \dots, W_k) = \prod_{k=1}^{k} y_k^{t_k}$

so, for N points, the likelihood function $\rightarrow$ $p(T|W_1, \dots, W_k) = \prod_{n=1}^{N}\prod_{k=1}^{k} y_{nk}^{t_{nk}}$

$$\Rightarrow -\ln(P(T|W_1, \dots, W_k)) = -\ln\left(\prod_{n=1}^{N}\prod_{k=1}^{k} y_{nk}^{t_{nk}}\right)$$

$$= -\sum_{n=1}^{N}\sum_{k=1}^{k} t_{kn} \cdot \ln y_{kn}$$

$$= -\sum_{n=1}^{N}\sum_{k=1}^{k} t_{kn} \cdot \ln y_k(x_n, w)$$

$$= E(w)$$ #