1. b

layers $1 \le l \le 3$ $\quad$ $l=1$ $\quad$ $1 \le j \le 5$

inputs $0 \le i \le d^2$ $\quad$ $l=2$ $\quad$ $1 \le j \le 6$

outputs $1 \le j \le d^3$ $\quad$ $(5+1)6 = 36$ #

2. d.

$20 \to 32 \to 18 \to 3$

$\quad\quad 31 \quad 17$

$(20\times31)+(32\times17)+(18\times3)=1218$

$20 \to 33 \to 17 \to 3$

$\quad\quad 32 \quad 16$

$(20\times32)+(33\times16)+(17\times3)=1219$ #

$20 \to 34 \to 16 \to 3$

$\quad\quad 33 \quad 15$

$(20\times33)+(34\times15)+(16\times3)=1218$

3. d.

$$q_k = \frac{e^{s_k}}{\sum_{k=1}^{K} e^{s_k}} \quad E = -\sum_{k=1}^{K} v_k \ln(q_k) \quad \frac{\partial E}{\partial q_k} = \frac{-v_k}{q_k}$$

$$\frac{\partial q_k}{\partial s_i} = \begin{cases} \frac{e^{s_k}}{\sum_{k=1}^{K} e^{s_k}} - \left(\frac{e^{s_k}}{\sum_{k=1}^{K} e^{s_k}}\right)^2 . & i=k \\ -\frac{e^{s_k} . s^{s_i}}{(\sum_{k=1}^{K} e^{s_k})^2} . & i \neq k \end{cases} = \begin{cases} q_k(1-q_k) . & i=k \\ -q_i q_k . & i \neq k \end{cases}$$

$$\frac{\partial E}{\partial s_k} = \sum_{k=1}^{K} \frac{\partial E}{\partial q_i} \cdot \frac{\partial q_i}{\partial s_k} = \frac{\partial E}{\partial q_k} \cdot \frac{\partial q_k}{\partial s_k} - \sum_{k \neq i} \frac{\partial E}{\partial q_i} \frac{\partial q_i}{s_k}$$

$$= v_k(1-q_k) + \sum k \neq i v_i q_k = -v_k + q_k \cdot \sum v_i$$

$$= q_k - v_k$$

掃描全能王 創建

5. e

$V_n \to \Sigma (r_m - w_m^T v)^2 = \| v w^* - y \|^2$ $w^* = (v^T v)^{-1} v^T y = \frac{1}{v} y$, $v = 2$ #

Linear Regression $\to \| xw - y \|^2 \to w = (X^T X)^{-1} X^T y$

6. b

$$\frac{\partial err}{\partial a_m} = -2 (r_{nm} - w_m^T v_n - a_m - b_n)$$

$$a_n \leftarrow a_m + \frac{\eta}{2} \cdot 2 (r_{nm} - w_m^T v_n - a_m - b_m)$$

$$a_m \leftarrow (1-\eta) a_m + \eta (r_{nm} - w_m^T v_n - b_m)$$ #

7. d.

(a) $x.x.o: [0.54 \times 0.08 \times 0.24 + 0.16 \times 0.92 \times 0.24 + 0.16 \times 0.08 \times 0.76] \cdot C_2^3 \fallingdotseq 0.18$

$x.x.x: 0.16 \times 0.08 \times 0.24 \fallingdotseq 0.003$

$0.18 + 0.003 = 0.183$ #

9. b.

$$\left(1 - \frac{1}{N}\right)^{0.5N} = \frac{1}{\left(\frac{N}{N-1}\right)^{0.5N}} = \frac{1}{\left(1 + \frac{1}{N-1}\right)^{0.5N}} = \sqrt{\frac{1}{\left(1 + \frac{1}{N-1}\right)^N}} \approx \sqrt{\frac{1}{e}} = 0.6065$$ #

11. a.

$+$ 5% $u_+ \frac{95}{100}$

$-$ 95% $u_- \frac{5}{100}$

$$\frac{u_+^{(2)}}{u_-^{(2)}} = \frac{\frac{95}{100}}{\frac{5}{100}} = 19$$

8. C

$xxxoo: (0.4)^3 \cdot (0.6)^2 \cdot C_3^5 = 0.23$ $0.23 + 0.075 + 0.01$

$xxxxo: (0.4)^4 (0.6) C_4^5 = 0.075$ $= 0.315 \fallingdotseq 0.32$ #

$xxxxx: (0.4)^5 = 0.01$

12. d

$$E = \sum_{n=1}^{N} u_n^{(t+1)} \left[(1-\epsilon_t)e^{-\alpha} + \epsilon_t e^{\alpha}\right]$$

$$= U_T \left[(1-\epsilon_t)e^{-\alpha} + \epsilon_t e^{\alpha}\right]$$

$$\leq U_T \cdot 2\sqrt{\epsilon(1-\epsilon)} \cdot \left[(1-\epsilon_t)e^{-\alpha} + \epsilon_t e^{\alpha}\right]$$

$$\leq U_T \cdot \exp\left(-2\left(\tfrac{1}{2}-\epsilon\right)^2\right) \cdot \left[(1-\epsilon_t)e^{-\alpha} + \epsilon_t e^{\alpha}\right]$$

$$= U_1 \cdot \exp\left(-2T\left(\tfrac{1}{2}-\epsilon\right)^2\right) \cdot \left[(1-\epsilon_t)e^{-\alpha} + \epsilon_t e^{\alpha}\right]$$

$$\leq \exp\left(-2T\left(\tfrac{1}{2}-\epsilon\right)^2\right) \#$$

13. d.

$$\min(u_+, 1-u_+) = \begin{cases} u_+ , & u_+ < 1-u_+ \\ 1-u_+ , & u_+ > 1-u_+ \end{cases} \xrightarrow[\text{normalize}]{\times \frac{1}{2}} \begin{cases} 2u_+ , & u_+ < 1-u_+ \\ 2-2u_+ , & u_+ > 1-u_+ \end{cases}$$

(d)

$$1-|u_+ - u_-| = 1-|u_+ - (1-u_+)|$$

$$\rightarrow \begin{cases} 1-(1-u_+)+u_+ = 2u_+ , & u_+ < 1-u_+ \\ 1-u_+ + (1-u_+) = 2-2u_+ , & u_+ > 1-u_+ \end{cases} \#$$

19. a

SVM是我很早就知道的模型 不過過去也只知道它大概的原理，這次透過這堂課，使我更瞭解SVM，因此能更適當的去應用它。

20. d

AdaBoost 跟 Gradient Boosting都是我從這堂課學到的新模型，不過 Gradient Boosting的原理推導較複雜，因此花了不少時間才瞭解它。

14~18 cdadx

```python
import numpy as np
import pandas as pd
import random
from tqdm import tqdm


def split_data_label(data):
    label = data[:, -1]
    data = data[:, :-1]
    return data, label


data = np.loadtxt("./hw6_train.dat.txt")
x, y = split_data_label(data)
print(x.shape, y.shape)

data = np.loadtxt("./hw6_test.dat.txt")
x_test, y_test = split_data_label(data)
print(x_test.shape, y_test.shape)


class CART:
    def __init__(self):
        self.feature = None
        self.label = None
        self.n_samples = None
        self.gain = None
        self.left = None
        self.right = None
        self.threshold = None
        self.depth = 0
        self.root = None

    def fit(self, features, target):
        self.root = CART()
        self.root._grow_tree(features, target)

    def predict(self, features):
```

```
        return np.array([self.root._predict(f) for f in features])

    def _grow_tree(self, features, target):
        self.n_samples = features.shape[0]

        if len(np.unique(target)) == 1:
            self.label = target[0]
            return

        best_gain = 0.0
        best_feature = None
        best_threshold = None

        self.label = max([(c, len(target[target == c]))
                          for c in np.unique(target)], key=lambda x:
 x[1])[0]

        impurity_node = self._calc_impurity(target)

        for col in range(features.shape[1]):
            feature_level = np.unique(features[:, col])
            thresholds = (feature_level[:-
1] + feature_level[1:]) / 2.0

            for threshold in thresholds:
                target_l = target[features[:, col] <= threshold]
                impurity_l = self._calc_impurity(target_l)
                n_l = float(target_l.shape[0]) / self.n_samples

                target_r = target[features[:, col] > threshold]
                impurity_r = self._calc_impurity(target_r)
                n_r = float(target_r.shape[0]) / self.n_samples

                impurity_gain = impurity_node - \
                    (n_l * impurity_l + n_r * impurity_r)
                if impurity_gain > best_gain:
                    best_gain = impurity_gain
                    best_feature = col
```

```
                    best_threshold = threshold

        self.feature = best_feature
        self.gain = best_gain
        self.threshold = best_threshold
        self._split_tree(features, target)

    def _split_tree(self, features, target):
        features_l = features[features[:, self.feature] <= self.threshold]
        target_l = target[features[:, self.feature] <= self.threshold]
        self.left = CART()
        self.left.depth = self.depth + 1
        self.left._grow_tree(features_l, target_l)

        features_r = features[features[:, self.feature] > self.threshold]
        target_r = target[features[:, self.feature] > self.threshold]
        self.right = CART()
        self.right.depth = self.depth + 1
        self.right._grow_tree(features_r, target_r)

    def _calc_impurity(self, target):
        return 1.0 - sum([(float(len(target[target == c])) / float(target.shape[0])) ** 2.0 for c in np.unique(target)])

    def _predict(self, d):
        if self.feature != None:
            if d[self.feature] <= self.threshold:
                return self.left._predict(d)
            else:
                return self.right._predict(d)
        else:
            return self.label


# 14
```

```python
cart = CART()
cart.fit(x, y)
preds = cart.predict(x_test)
E = 1 - sum(preds == y_test)/len(y_test)
print(E)


def boostrap(x, y, N):
    indexs = [random.randint(0, N//2) for _ in range(N)]
    return x[indexs], y[indexs]


def predict(x, y, x_test, T):
    pred_tmp = np.zeros(y.shape)
    final_pred = []
    for i in tqdm(range(T)):
        tx, ty = boostrap(x, y, len(y)-1)
        cart = CART()
        cart.fit(tx, ty)
        pred = cart.predict(x_test)
        pred_tmp += pred
    for i in range(len(pred_tmp)):
        if pred_tmp[i] >= 0:
            final_pred.append(1)
        else:
            final_pred.append(-1)
    return np.array(final_pred)


# 15
pred = predict(x, y, x_test, 2000)
E = 1 - sum(pred == y_test)/len(y_test)
print(E)


def boostrap(x, y, N):
    indexs = [random.randint(0, N//2) for _ in range(N)]
    return x[indexs], y[indexs]
```

```python
def sign(x):
    if x >= 0:
        return 1
    else:
        return -1


def predict(x, y, x_test, T):
    pred_tmp = np.zeros(y.shape)
    final_pred = []
    for i in tqdm(range(T)):
        tx, ty = boostrap(x, y, len(y)-1)
        cart = CART()
        cart.fit(tx, ty)
        pred = cart.predict(x_test)
        pred_tmp += pred
    for i in range(len(pred_tmp)):
        final_pred.append(sign(pred_tmp[i]))
    return np.array(final_pred)


# 16
pred = predict(x, y, x, 2000)
E = 1 - sum(pred == y)/len(y)
print(E)

# 17
pred = predict(x, y, x_test, 2000)
E = 1 - sum(pred == y_test)/len(y_test)
print(E)
```