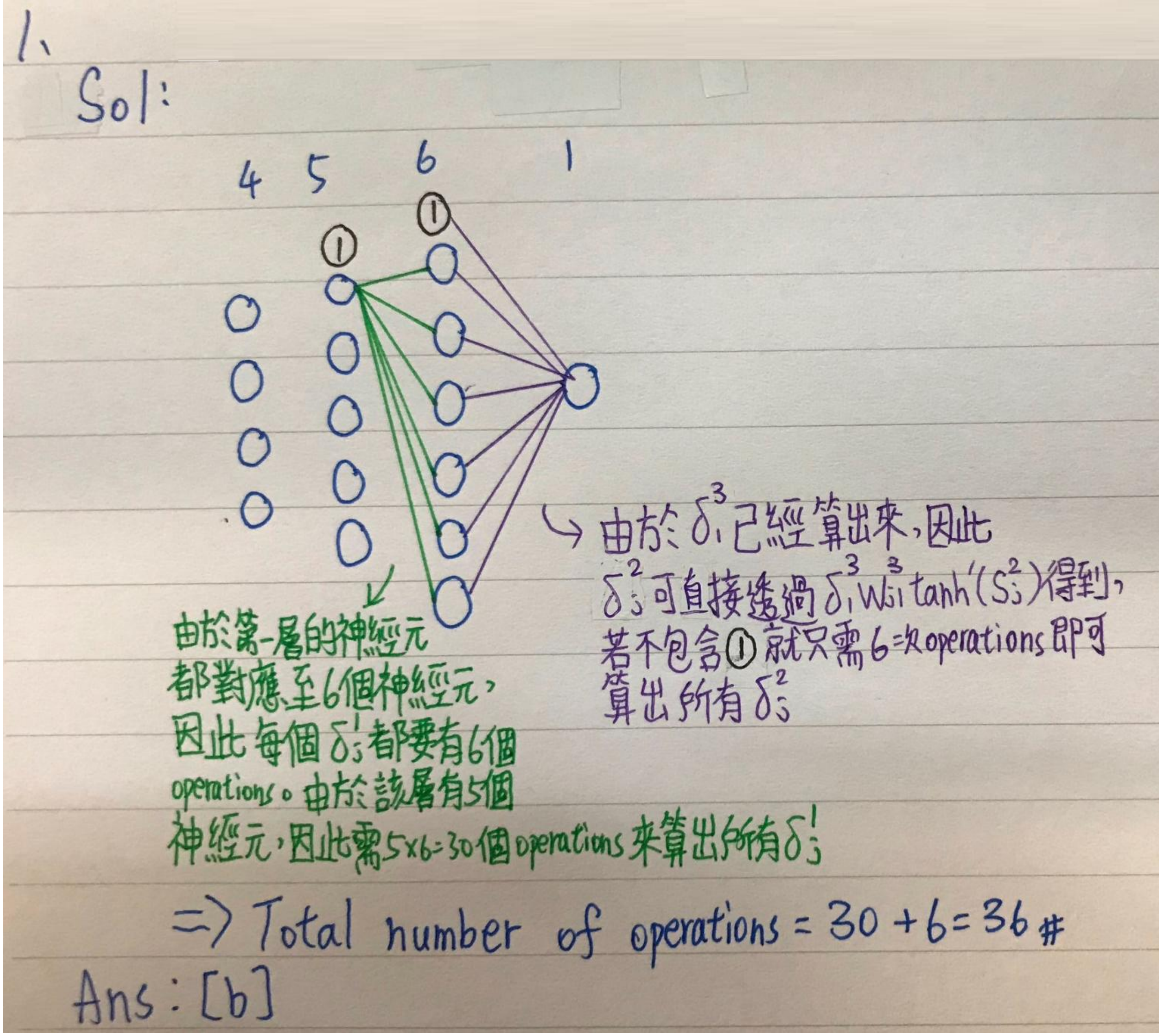
作業二

tags: 機器學習技法作業

第一題



1、

Sol:

由於第一層的神經元都對應至6個神經元，因此每個 δ_j^1 都要有6個 operations。由於該層有5個神經元，因此需 5×6=30 個 operations 來算出所有 δ_j^1

由於 δ_1^3 已經算出來，因此 δ_j^2 可直接透過 $\delta_1^3 w_{j1}^3 \tanh'(s_j^2)$ 得到，若不包含 ① 就只需6次 operations 即可算出所有 δ_j^2

$\Rightarrow$ Total number of operations = 30 + 6 = 36 #

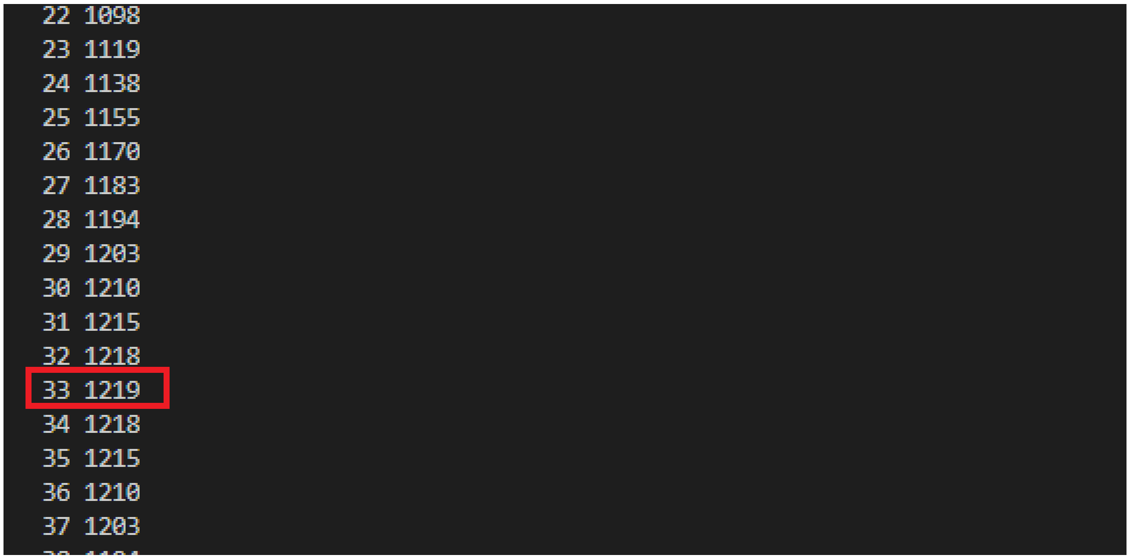
Ans: [b]

Ans: [b]

第二題

在相同數量的神經元下，越多隱藏層會讓權重的數量減少，因為每一層都會多浪費一個神經元給+1，但是前一層的神經元又不會連到他，因此權重數量減少。不過假如只有一層的話，權重的數量又不會比兩層多，因此如果想知道相同數量的神經元下，最多權重的情況是怎樣，就可以直接將兩層時所有的情況列出來。透過以下程式發現，可以得到權重最多為1219個。

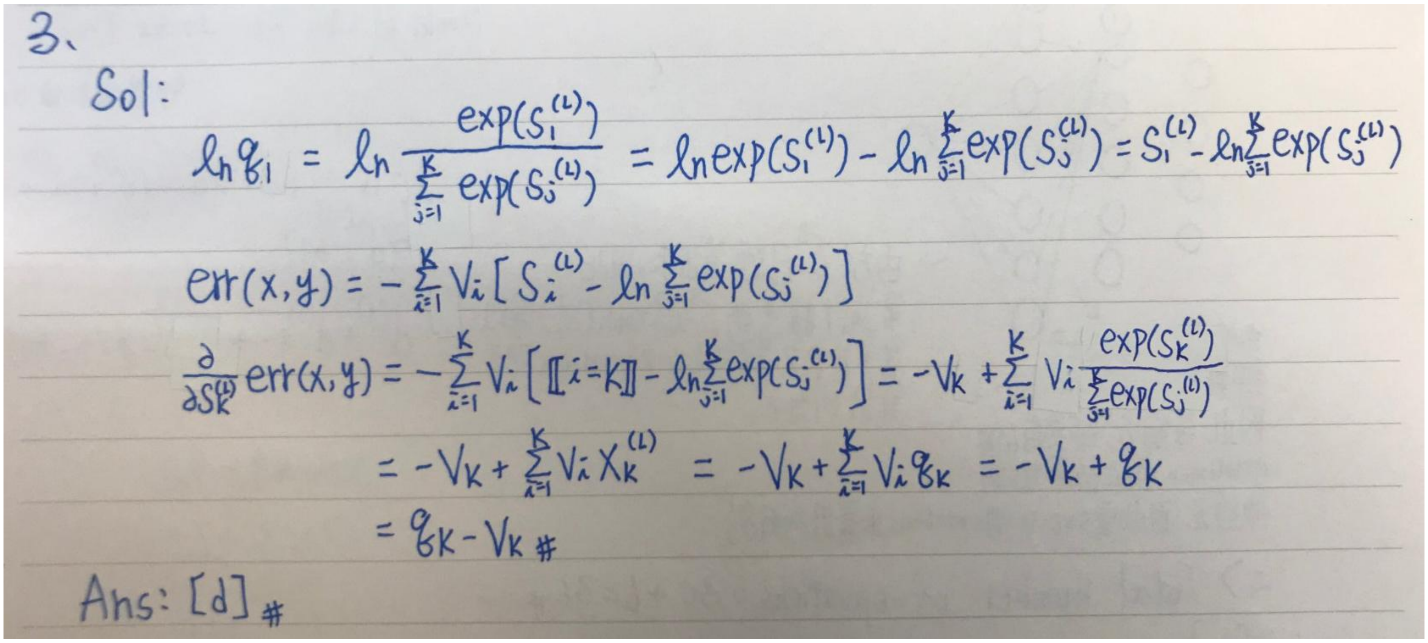
```
def hi(x):
    return 20*(x-1) + (x)*(50-x-1) + 3*(50-x)

def problem2():
    for (i, value) in zip( range(1,50), map(hi, range(1,50))):
        print(i, value)

problem2()
```



```
22 1098
23 1119
24 1138
25 1155
26 1170
27 1183
28 1194
29 1203
30 1210
31 1215
32 1218
33 1219
34 1218
35 1215
36 1210
37 1203
```

Ans: [d]

第三題



3、

Sol:

$$\ln q_1 = \ln \frac{\exp(s_1^{(L)})}{\sum_{j=1}^{K} \exp(s_j^{(L)})} = \ln \exp(s_1^{(L)}) - \ln \sum_{j=1}^{K} \exp(s_j^{(L)}) = s_1^{(L)} - \ln \sum_{j=1}^{K} \exp(s_j^{(L)})$$

$$err(x,y) = -\sum_{i=1}^{K} v_i \left[s_i^{(L)} - \ln \sum_{j=1}^{K} \exp(s_j^{(L)}) \right]$$

$$\frac{\partial}{\partial s_k^{(L)}} err(x,y) = -\sum_{i=1}^{K} v_i \left[[\![i=k]\!] - \ln \sum_{j=1}^{K} \exp(s_j^{(L)}) \right] = -v_k + \sum_{i=1}^{K} v_i \frac{\exp(s_k^{(L)})}{\sum_{j=1}^{K} \exp(s_j^{(L)})}$$

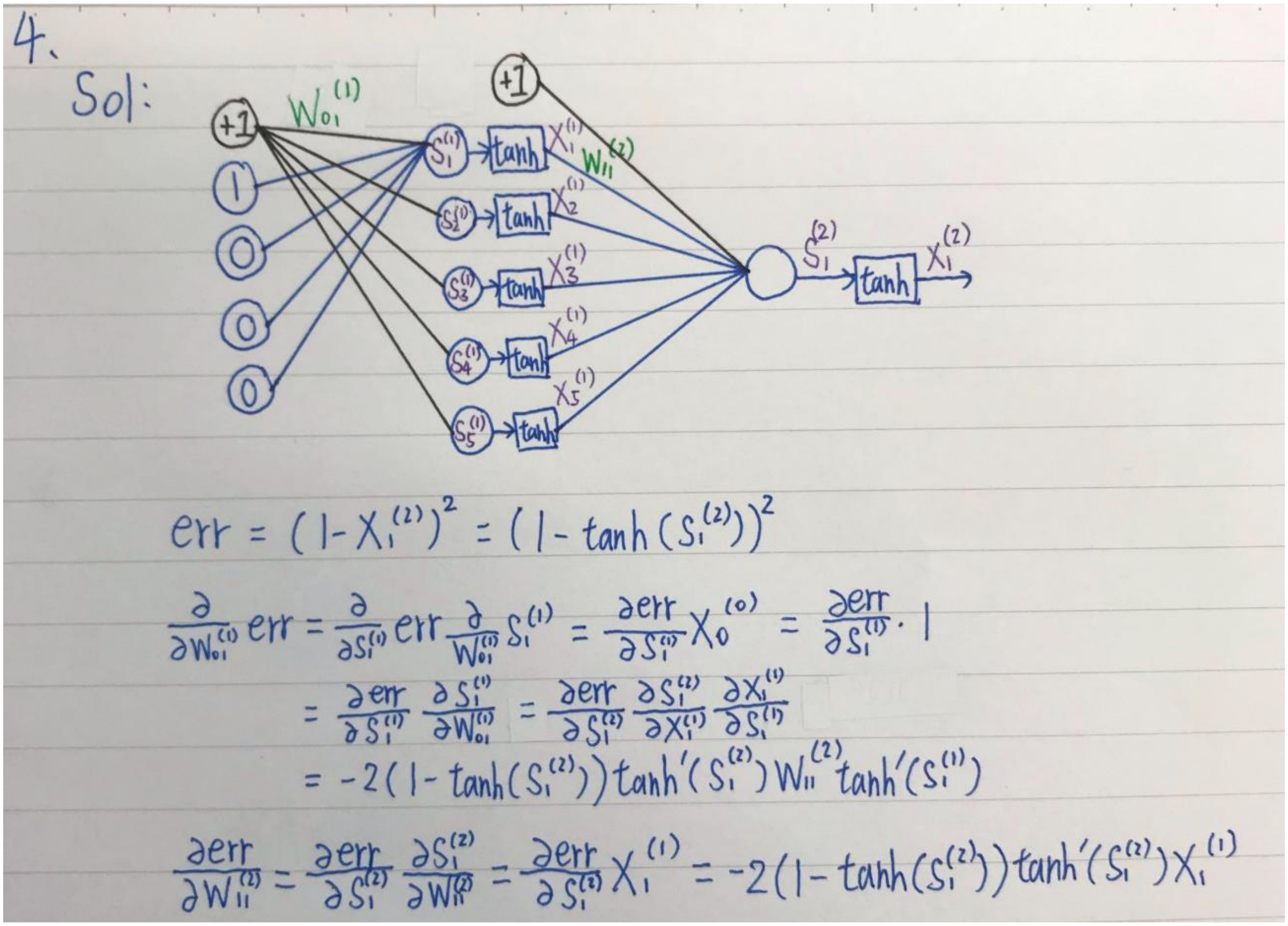
$$= -v_k + \sum_{i=1}^{K} v_i x_k^{(L)} = -v_k + \sum_{i=1}^{K} v_i q_k = -v_k + q_k$$

$$= q_k - v_k \ \#$$

Ans: [d] #

Ans: [d]

第四題



4、

Sol:

$$err = (1 - x_1^{(2)})^2 = (1 - \tanh(s_1^{(2)}))^2$$

$$\frac{\partial}{\partial w_{01}^{(1)}} err = \frac{\partial}{\partial s_1^{(1)}} err \frac{\partial}{w_{01}^{(1)}} s_1^{(1)} = \frac{\partial err}{\partial s_1^{(1)}} x_0^{(0)} = \frac{\partial err}{\partial s_1^{(1)}} \cdot 1$$

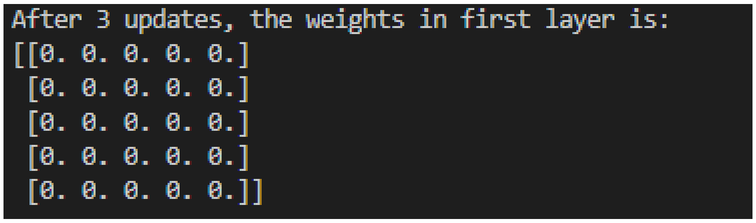
$$= \frac{\partial err}{\partial s_1^{(1)}} \frac{\partial s_1^{(1)}}{\partial w_{01}^{(1)}} = \frac{\partial err}{\partial s_1^{(2)}} \frac{\partial s_1^{(2)}}{\partial x_1^{(1)}} \frac{\partial x_1^{(1)}}{\partial s_1^{(1)}}$$

$$= -2(1 - \tanh(s_1^{(2)})) \tanh'(s_1^{(2)}) w_{11}^{(2)} \tanh'(s_1^{(1)})$$

$$\frac{\partial err}{\partial w_{11}^{(2)}} = \frac{\partial err}{\partial s_1^{(2)}} \frac{\partial s_1^{(2)}}{\partial w_{11}^{(2)}} = \frac{\partial err}{\partial s_1^{(2)}} x_1^{(1)} = -2(1 - \tanh(s_1^{(2)})) \tanh'(s_1^{(2)}) x_1^{(1)}$$

```
class NN_3L:
    def __init__(self):
        self.w1 = np.zeros((5, 5)).astype(float)
        self.w2 = np.zeros(6, ).astype(float)
        self.s1 = np.zeros(5, ).astype(float)
        self.s2 = np.zeros(1, ).astype(float)
        self.x0 = np.array([1, 1, 0, 0, 0]).astype(float)
        self.x1 = np.concatenate((np.ones(1, ), np.tanh(np.dot(self.w1, self.x0))),
        self.x2 = np.tanh(np.dot(self.w2, self.x1))
    def forward(self):
        self.s1 = np.dot(self.w1, self.x0)
        self.s2 = np.dot(self.w2, self.x1)
        self.x1 = np.concatenate((np.ones(1, ), np.tanh(self.s1)), None)
        self.x2 = np.tanh(self.s2)
    def backward(self):
        partial_w2 = np.multiply(np.multiply(-2*(1-np.tanh(self.s2)), self.Dtanh(se
        a = np.multiply(-2*(1-np.tanh(self.s2)), self.Dtanh(self.s2))
        b = np.multiply(a, self.w2[1:])
        c = np.multiply(b, self.Dtanh(self.s1))
        partial_w1 = np.dot(self.x0.reshape(5, 1), c.reshape(1, 5))
        self.w1 -= partial_w1
        self.w2 -= partial_w2

    def Dtanh(self, x):
        return (1-np.tanh(x)**2)

def problem3():
    NN = NN_3L()
    for i in range(3):
        NN.forward()
        NN.backward()
    print("After 3 updates, the weights in first layer is:\n{0} ".format(NN.w1))

if __name__ == "__main__":
    problem3()
```

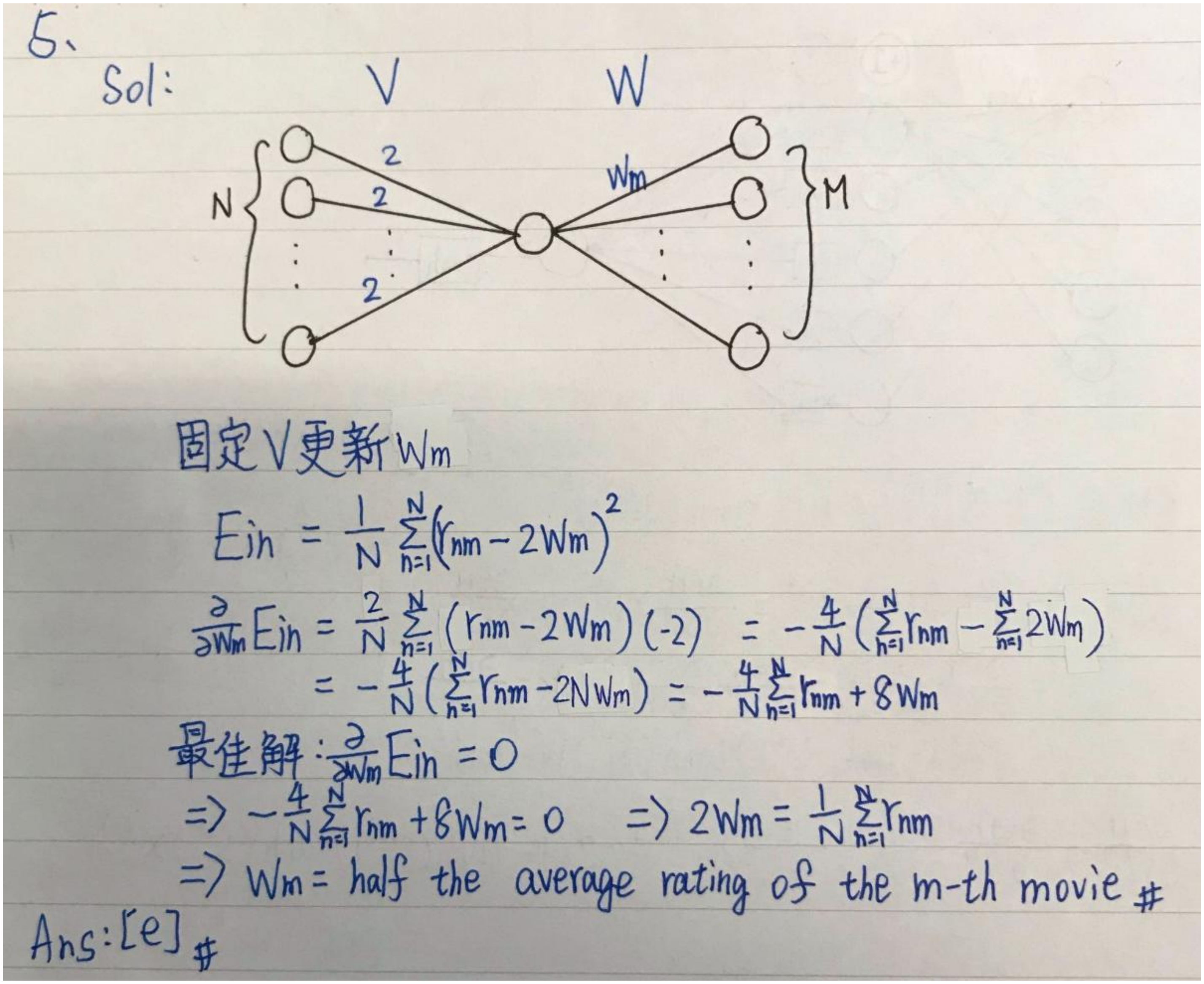
根據程式執行結果可知，經過三次的權重更新之後，第一層的所有權重都保持為0。原因是所有權重的初始值都為0，根據微分的結果可以發現第二層的權重將不會被更新，導致第一層的權重不會被更新：



```
After 3 updates, the weights in first layer is:
[[0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0.]]
```

Ans: [a]

第五題



5、

Sol:

固定V更新W_m

$$E_{in} = \frac{1}{N}\sum_{n=1}^{N}(r_{nm}-2W_m)^2$$

$$\frac{\partial}{\partial W_m}E_{in} = \frac{2}{N}\sum_{n=1}^{N}(r_{nm}-2W_m)(-2) = -\frac{4}{N}\left(\sum_{n=1}^{N}r_{nm}-\sum_{n=1}2W_m\right)$$

$$= -\frac{4}{N}\left(\sum_{n=1}^{N}r_{nm}-2NW_m\right) = -\frac{4}{N}\sum_{n=1}^{N}r_{nm}+8W_m$$

最佳解：$\frac{\partial}{\partial W_m}E_{in}=0$

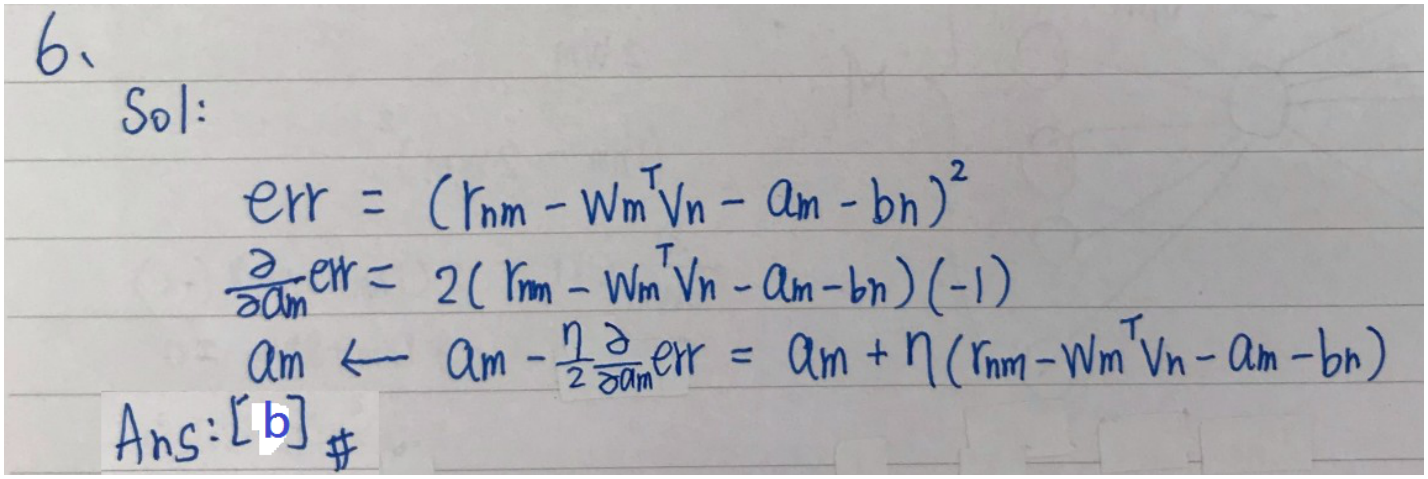
$\Rightarrow -\frac{4}{N}\sum_{n=1}^{N}r_{nm}+8W_m=0 \quad \Rightarrow 2W_m=\frac{1}{N}\sum_{n=1}^{N}r_{nm}$

$\Rightarrow$ W_m = half the average rating of the m-th movie #

Ans:[e] #

Ans: [e]

第六題



6、

Sol:

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

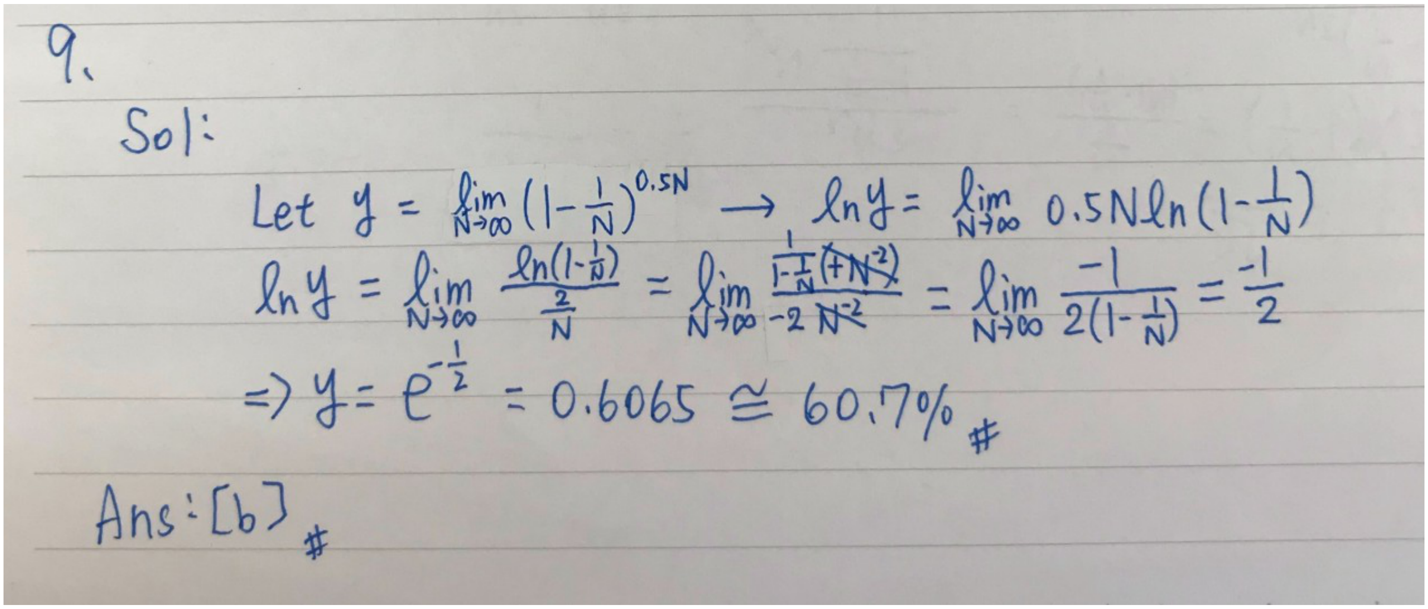
$$\frac{\partial}{\partial a_m} err = 2(r_{nm} - w_m^T v_n - a_m - b_n)(-1)$$

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

Ans: [b] #

Ans: [b]

第九題



9、

Sol:

Let $y = \lim_{N\to\infty}(1-\frac{1}{N})^{0.5N} \longrightarrow \ln y = \lim_{N\to\infty} 0.5N\ln(1-\frac{1}{N})$

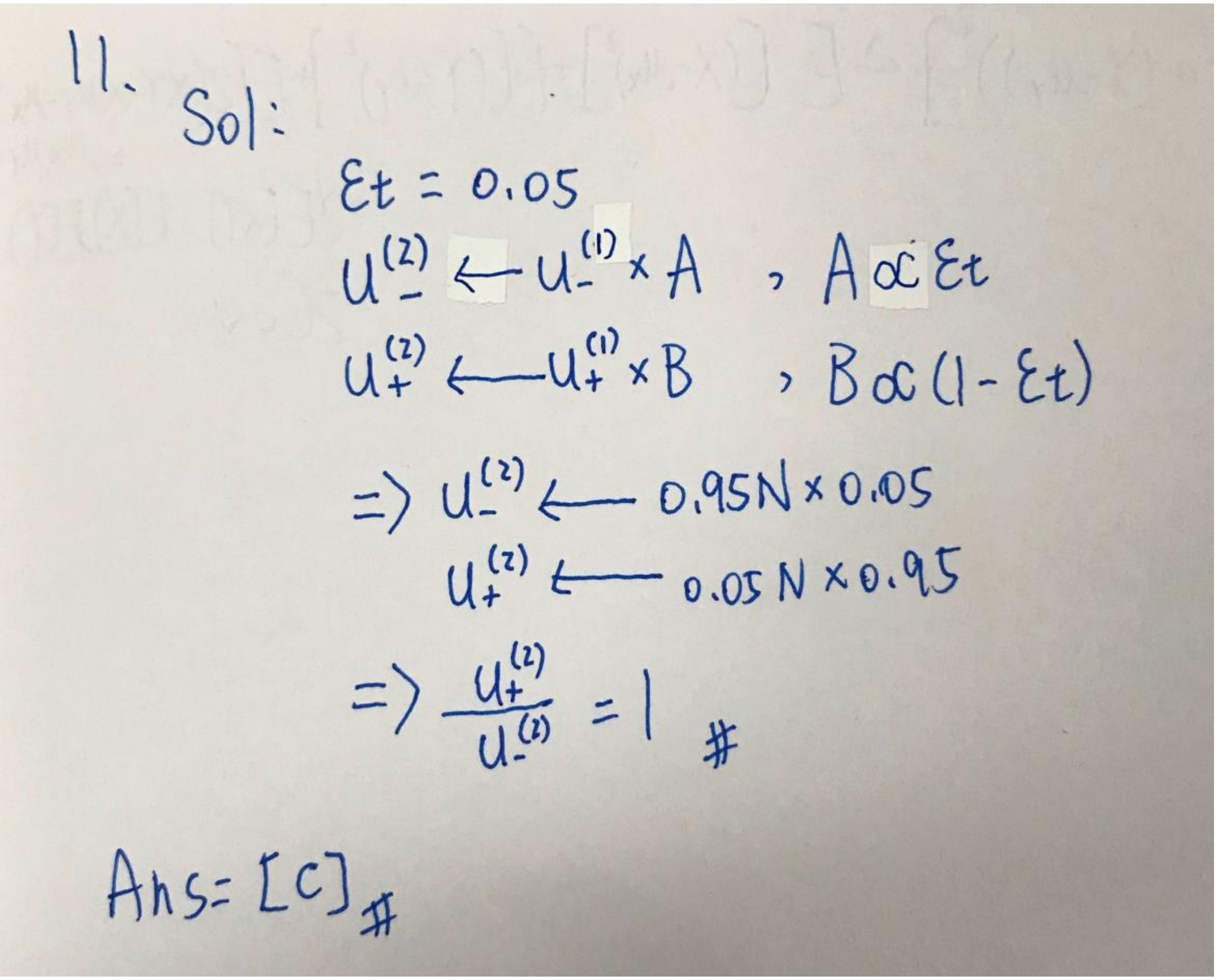
$$\ln y = \lim_{N\to\infty} \frac{\ln(1-\frac{1}{N})}{\frac{2}{N}} = \lim_{N\to\infty} \frac{\frac{1}{1-\frac{1}{N}}\cancel{(+N^{-2})}}{-2\cancel{N^{-2}}} = \lim_{N\to\infty} \frac{-1}{2(1-\frac{1}{N})} = \frac{-1}{2}$$

$\Rightarrow y = e^{-\frac{1}{2}} = 0.6065 \cong 60.7\%$ #

Ans: [b] #

Ans: [b]

第十一題



11.

Sol:

$\varepsilon_t = 0.05$

$u_-^{(2)} \leftarrow u_-^{(1)} \times A$, $A \propto \varepsilon_t$

$u_+^{(2)} \longleftarrow u_+^{(1)} \times B$, $B \propto (1-\varepsilon_t)$

$\Rightarrow u_-^{(2)} \longleftarrow 0.95N \times 0.05$

$u_+^{(2)} \longleftarrow 0.05N \times 0.95$

$\Rightarrow \frac{u_+^{(2)}}{u_-^{(2)}} = 1$ #

Ans= [c] #

Ans: [c]

第十四題

Ans: [e]

第十五題

Ans: [e]

第十六題

Ans: [a]

第十七題

Ans: [d]

第十八題

Ans: [b]

第十九題

這裡面許多演算法都非常直覺，而且也有許多可以應用的地方。其中我最喜歡的演算法是RandomForest，因為他可以做很多事，不但可以透過自己本身來達成validation，還可以用來做feature selection，而且執行起來效果也很不錯，我們在final project就使用他。

Ans: [c]

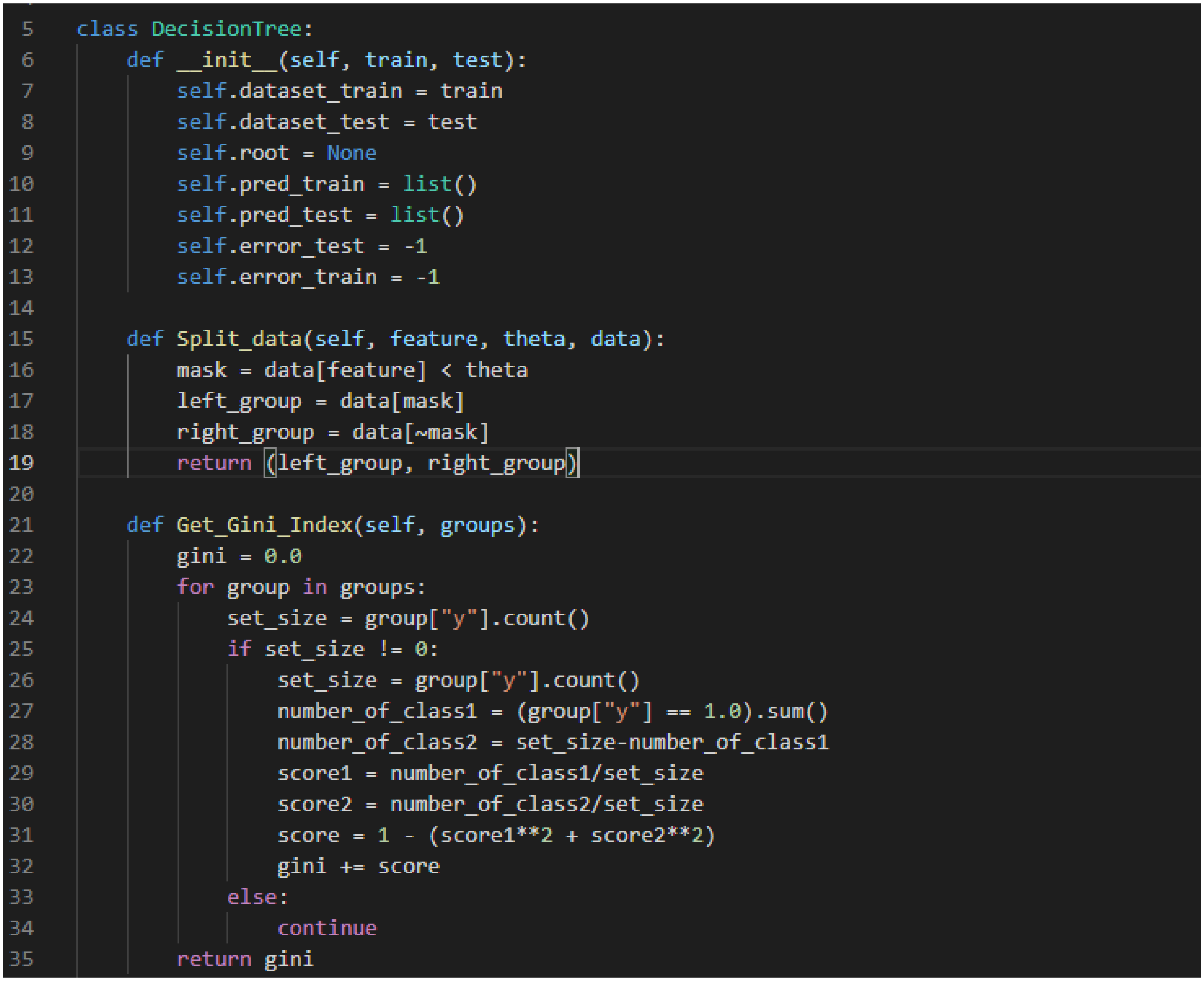
第二十題

數學的推導有些複雜，每次上課都要花很多時間才能搞懂，而且教到後面發現好多演算法都比他好，因此相較於其他部分，比較沒那麼喜歡這部分。不過這不是老師的問題，老師教得很認真，就是個人喜好而已。

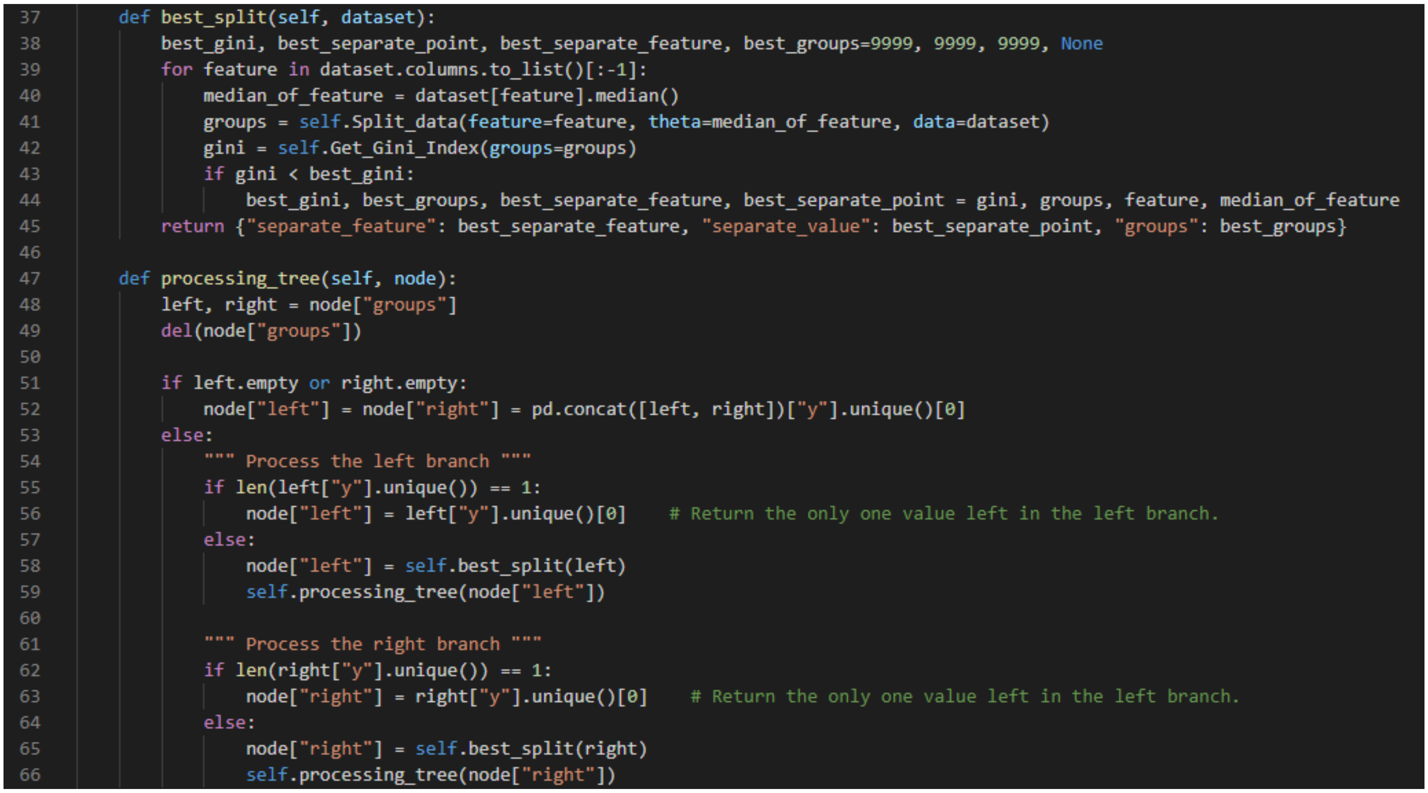
Ans: [a]

程式

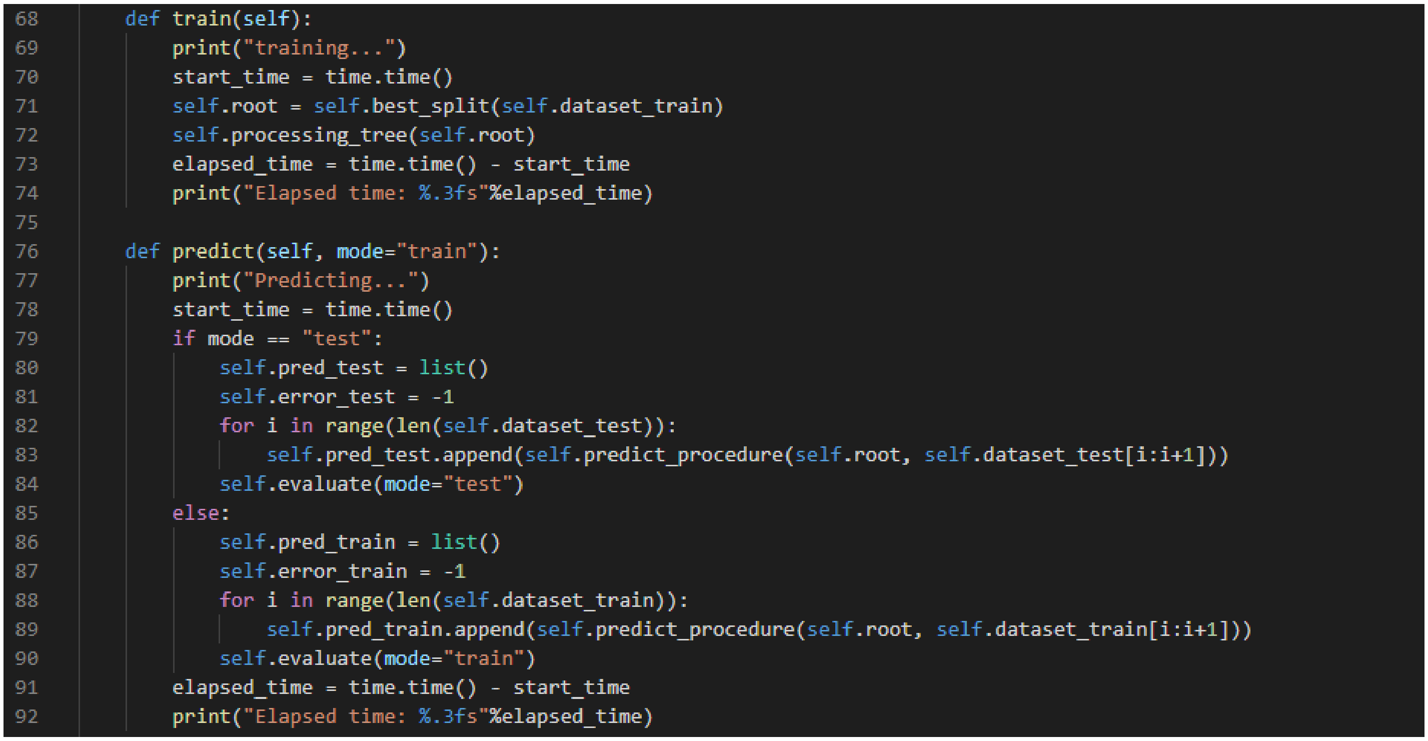
DecisionTree.py (http://DecisionTree.py)



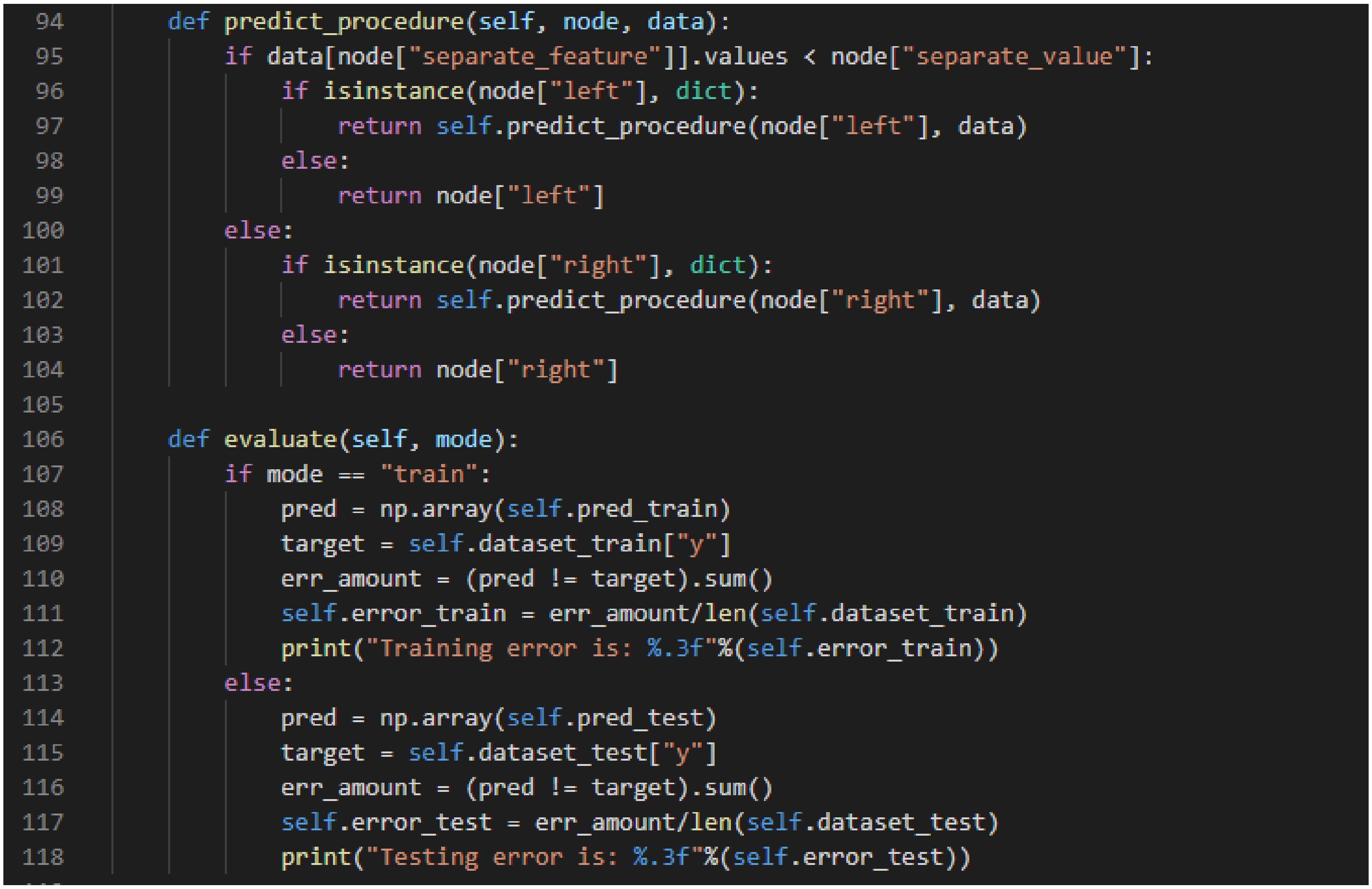
```python
class DecisionTree:
    def __init__(self, train, test):
        self.dataset_train = train
        self.dataset_test = test
        self.root = None
        self.pred_train = list()
        self.pred_test = list()
        self.error_test = -1
        self.error_train = -1

    def Split_data(self, feature, theta, data):
        mask = data[feature] < theta
        left_group = data[mask]
        right_group = data[~mask]
        return (left_group, right_group)

    def Get_Gini_Index(self, groups):
        gini = 0.0
        for group in groups:
            set_size = group["y"].count()
            if set_size != 0:
                set_size = group["y"].count()
                number_of_class1 = (group["y"] == 1.0).sum()
                number_of_class2 = set_size-number_of_class1
                score1 = number_of_class1/set_size
                score2 = number_of_class2/set_size
                score = 1 - (score1**2 + score2**2)
                gini += score
            else:
                continue
        return gini
```



```python
    def best_split(self, dataset):
        best_gini, best_separate_point, best_separate_feature, best_groups=9999, 9999, 9999, None
        for feature in dataset.columns.to_list()[:-1]:
            median_of_feature = dataset[feature].median()
            groups = self.Split_data(feature=feature, theta=median_of_feature, data=dataset)
            gini = self.Get_Gini_Index(groups=groups)
            if gini < best_gini:
                best_gini, best_groups, best_separate_feature, best_separate_point = gini, groups, feature, median_of_feature
        return {"separate_feature": best_separate_feature, "separate_value": best_separate_point, "groups": best_groups}

    def processing_tree(self, node):
        left, right = node["groups"]
        del(node["groups"])

        if left.empty or right.empty:
            node["left"] = node["right"] = pd.concat([left, right])["y"].unique()[0]
        else:
            """ Process the left branch """
            if len(left["y"].unique()) == 1:
                node["left"] = left["y"].unique()[0]    # Return the only one value left in the left branch.
            else:
                node["left"] = self.best_split(left)
                self.processing_tree(node["left"])

            """ Process the right branch """
            if len(right["y"].unique()) == 1:
                node["right"] = right["y"].unique()[0]    # Return the only one value left in the left branch.
            else:
                node["right"] = self.best_split(right)
                self.processing_tree(node["right"])
```

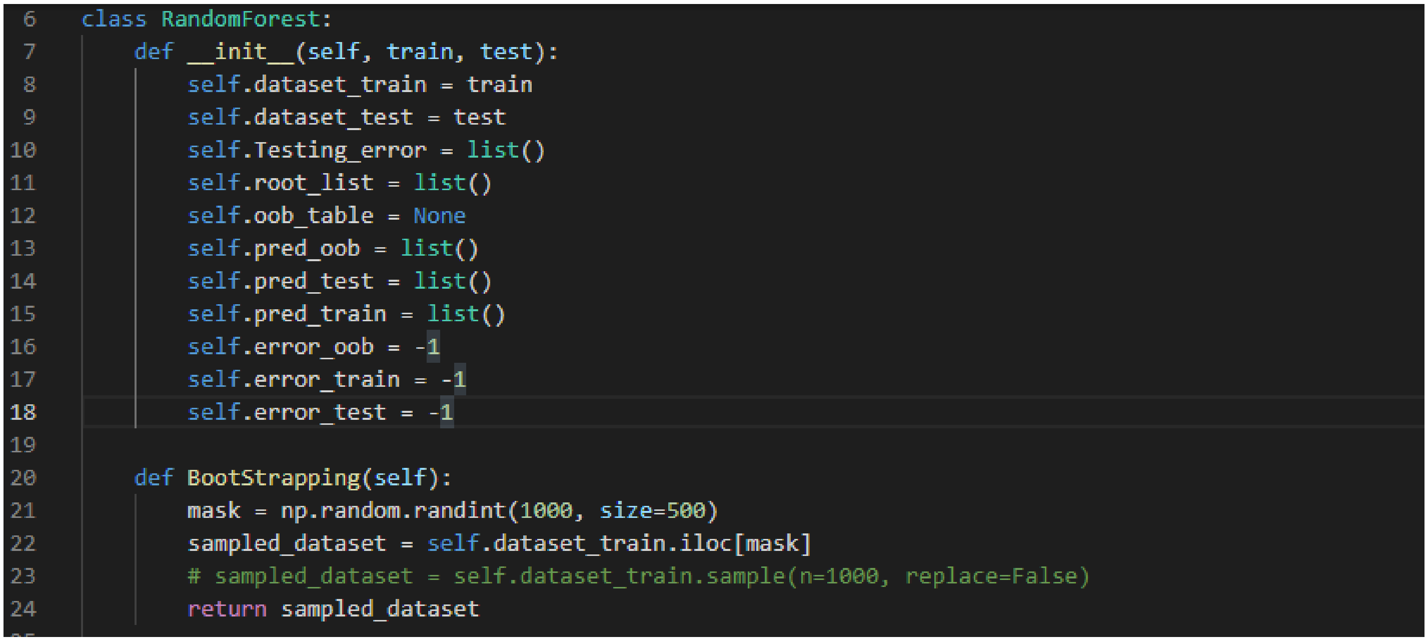


```python
    def train(self):
        print("training...")
        start_time = time.time()
        self.root = self.best_split(self.dataset_train)
        self.processing_tree(self.root)
        elapsed_time = time.time() - start_time
        print("Elapsed time: %.3fs"%elapsed_time)

    def predict(self, mode="train"):
        print("Predicting...")
        start_time = time.time()
        if mode == "test":
            self.pred_test = list()
            self.error_test = -1
            for i in range(len(self.dataset_test)):
                self.pred_test.append(self.predict_procedure(self.root, self.dataset_test[i:i+1]))
            self.evaluate(mode="test")
        else:
            self.pred_train = list()
            self.error_train = -1
            for i in range(len(self.dataset_train)):
                self.pred_train.append(self.predict_procedure(self.root, self.dataset_train[i:i+1]))
            self.evaluate(mode="train")
        elapsed_time = time.time() - start_time
        print("Elapsed time: %.3fs"%elapsed_time)
```

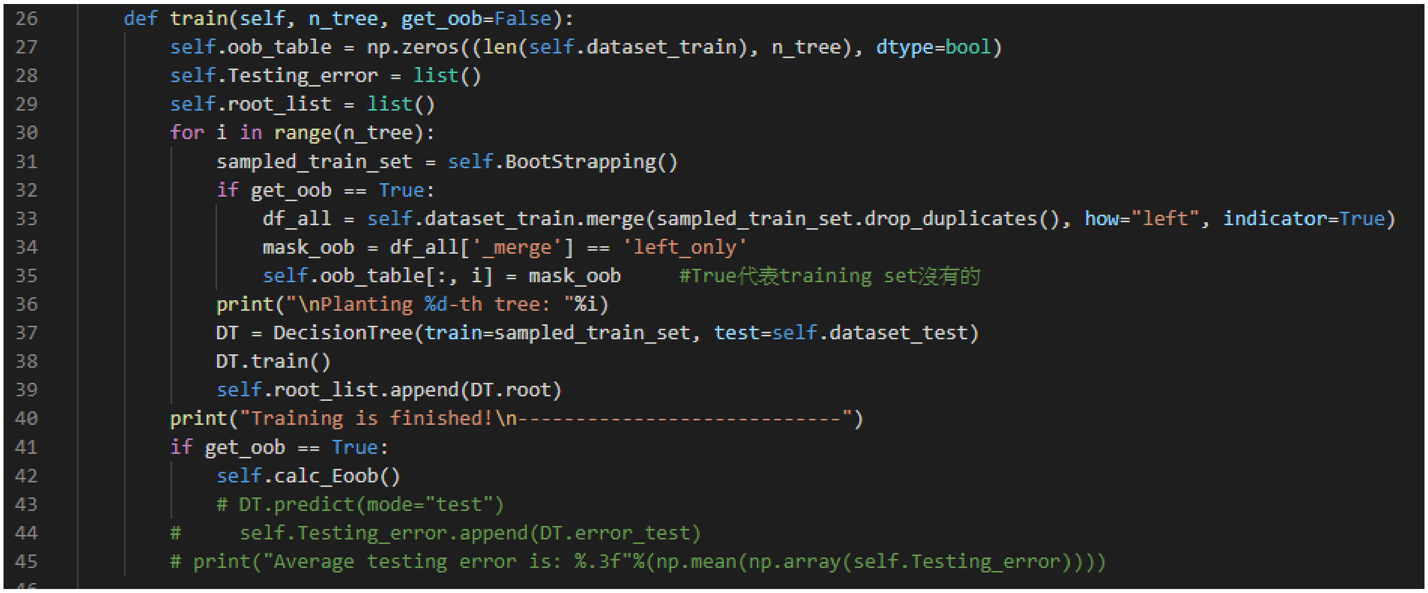


```python
    def predict_procedure(self, node, data):
        if data[node["separate_feature"]].values < node["separate_value"]:
            if isinstance(node["left"], dict):
                return self.predict_procedure(node["left"], data)
            else:
                return node["left"]
        else:
            if isinstance(node["right"], dict):
                return self.predict_procedure(node["right"], data)
            else:
                return node["right"]

    def evaluate(self, mode):
        if mode == "train":
            pred = np.array(self.pred_train)
            target = self.dataset_train["y"]
            err_amount = (pred != target).sum()
            self.error_train = err_amount/len(self.dataset_train)
            print("Training error is: %.3f"%(self.error_train))
        else:
            pred = np.array(self.pred_test)
            target = self.dataset_test["y"]
            err_amount = (pred != target).sum()
            self.error_test = err_amount/len(self.dataset_test)
            print("Testing error is: %.3f"%(self.error_test))
```

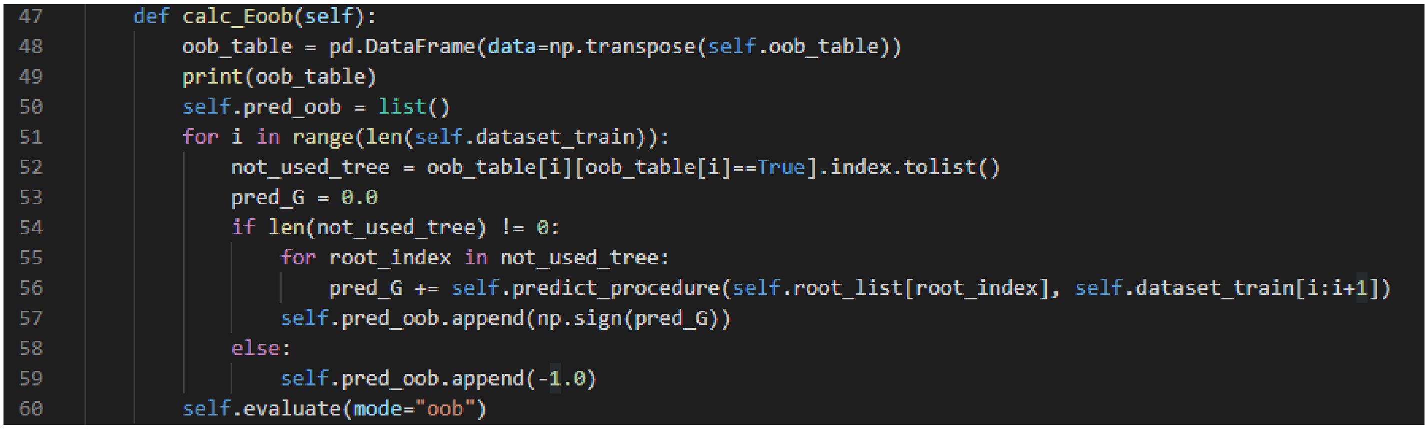
RandomForest.py (http://RandomForest.py)



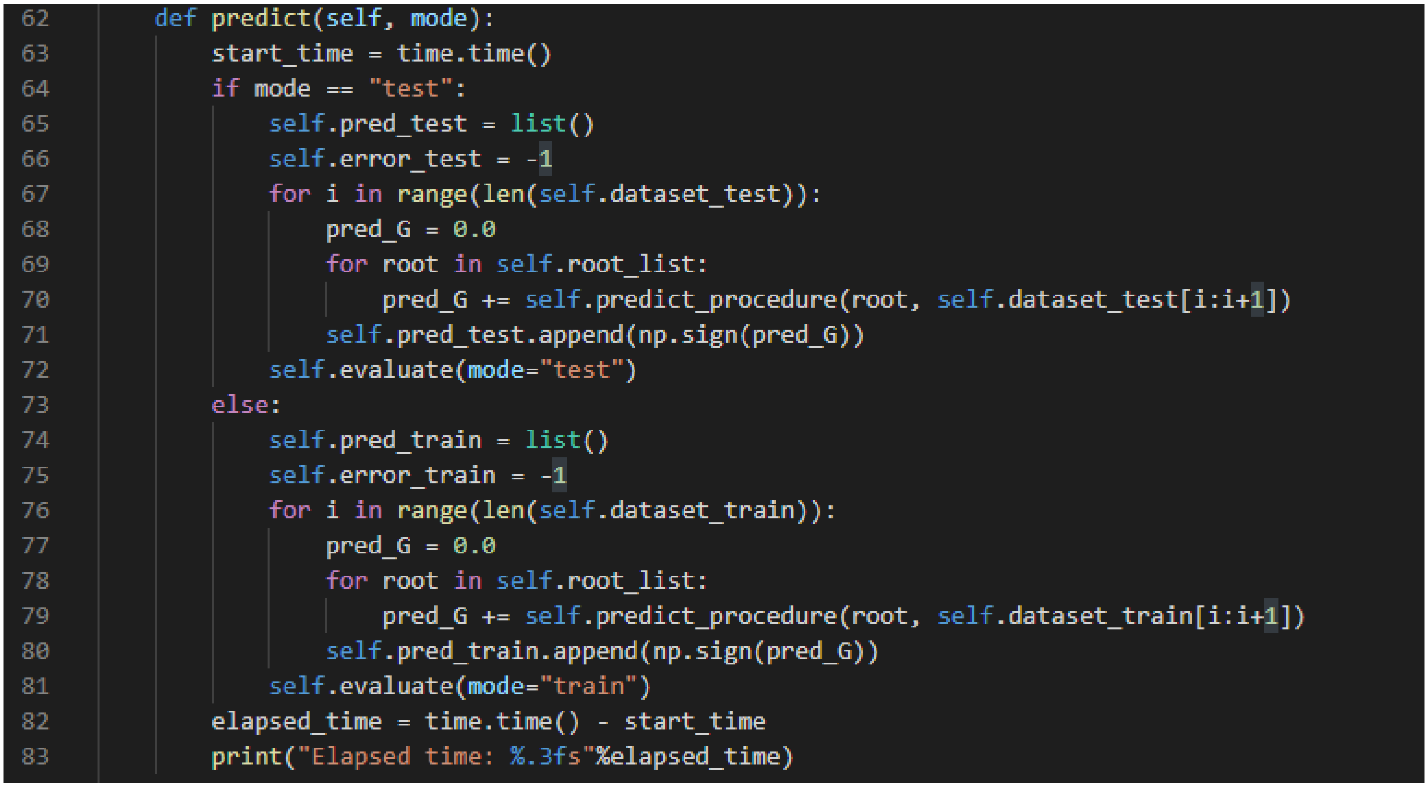
```python
class RandomForest:
    def __init__(self, train, test):
        self.dataset_train = train
        self.dataset_test = test
        self.Testing_error = list()
        self.root_list = list()
        self.oob_table = None
        self.pred_oob = list()
        self.pred_test = list()
        self.pred_train = list()
        self.error_oob = -1
        self.error_train = -1
        self.error_test = -1

    def BootStrapping(self):
        mask = np.random.randint(1000, size=500)
        sampled_dataset = self.dataset_train.iloc[mask]
        # sampled_dataset = self.dataset_train.sample(n=1000, replace=False)
        return sampled_dataset
```



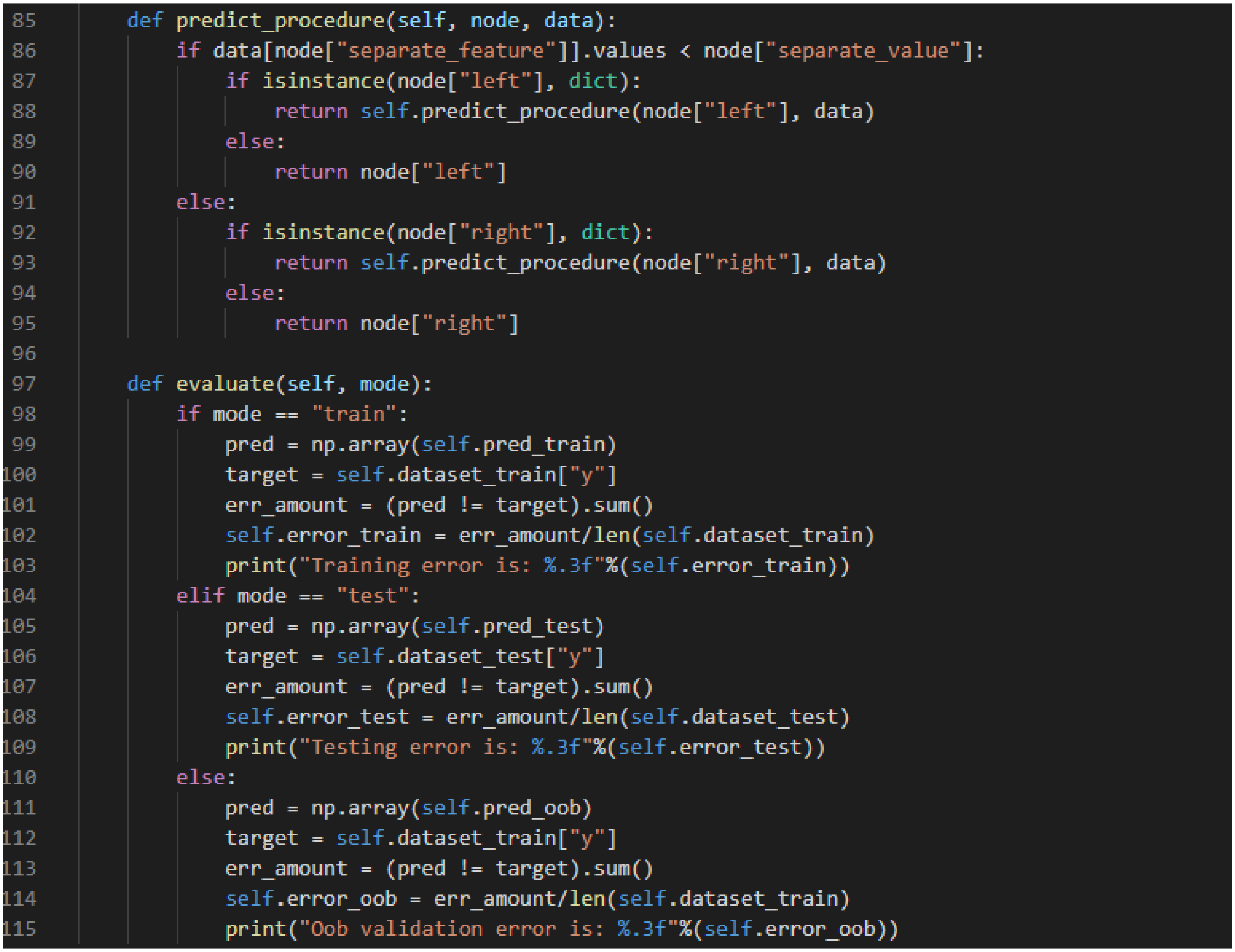
```
    def train(self, n_tree, get_oob=False):
        self.oob_table = np.zeros((len(self.dataset_train), n_tree), dtype=bool)
        self.Testing_error = list()
        self.root_list = list()
        for i in range(n_tree):
            sampled_train_set = self.BootStrapping()
            if get_oob == True:
                df_all = self.dataset_train.merge(sampled_train_set.drop_duplicates(), how="left", indicator=True)
                mask_oob = df_all['_merge'] == 'left_only'
                self.oob_table[:, i] = mask_oob     #True代表training set沒有的
            print("\nPlanting %d-th tree: "%i)
            DT = DecisionTree(train=sampled_train_set, test=self.dataset_test)
            DT.train()
            self.root_list.append(DT.root)
        print("Training is finished!\n---------------------------")
        if get_oob == True:
            self.calc_Eoob()
            # DT.predict(mode="test")
        #     self.Testing_error.append(DT.error_test)
        # print("Average testing error is: %.3f"%(np.mean(np.array(self.Testing_error))))
```



```
    def calc_Eoob(self):
        oob_table = pd.DataFrame(data=np.transpose(self.oob_table))
        print(oob_table)
        self.pred_oob = list()
        for i in range(len(self.dataset_train)):
            not_used_tree = oob_table[i][oob_table[i]==True].index.tolist()
            pred_G = 0.0
            if len(not_used_tree) != 0:
                for root_index in not_used_tree:
                    pred_G += self.predict_procedure(self.root_list[root_index], self.dataset_train[i:i+1])
                self.pred_oob.append(np.sign(pred_G))
            else:
                self.pred_oob.append(-1.0)
        self.evaluate(mode="oob")
```

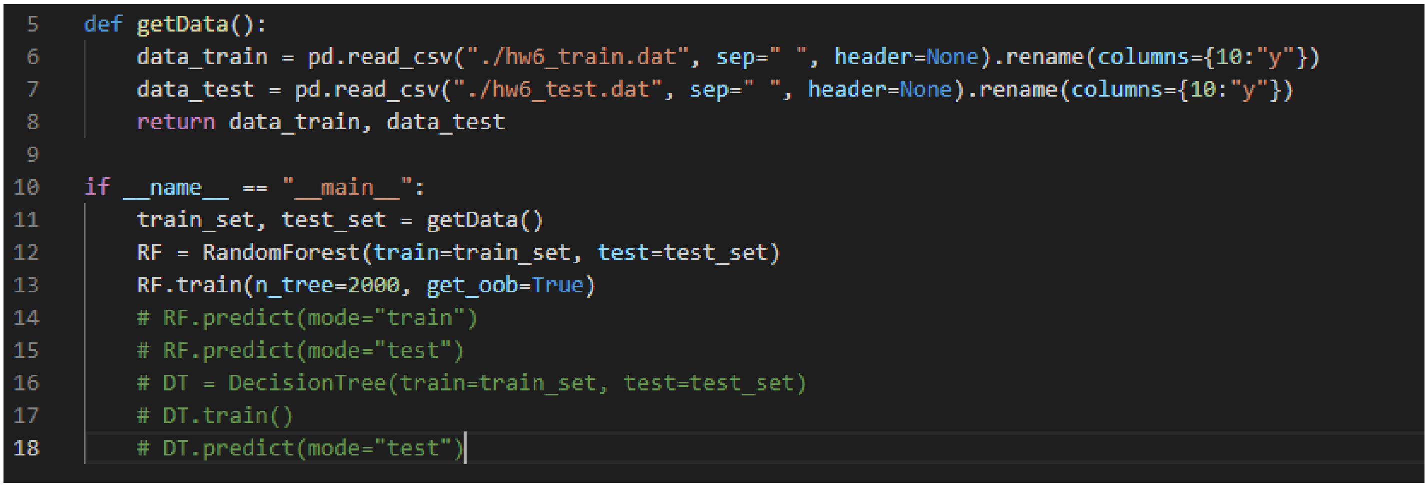


```
    def predict(self, mode):
        start_time = time.time()
        if mode == "test":
            self.pred_test = list()
            self.error_test = -1
            for i in range(len(self.dataset_test)):
                pred_G = 0.0
                for root in self.root_list:
                    pred_G += self.predict_procedure(root, self.dataset_test[i:i+1])
                self.pred_test.append(np.sign(pred_G))
            self.evaluate(mode="test")
        else:
            self.pred_train = list()
            self.error_train = -1
            for i in range(len(self.dataset_train)):
                pred_G = 0.0
                for root in self.root_list:
                    pred_G += self.predict_procedure(root, self.dataset_train[i:i+1])
                self.pred_train.append(np.sign(pred_G))
            self.evaluate(mode="train")
        elapsed_time = time.time() - start_time
        print("Elapsed time: %.3fs"%elapsed_time)
```



```python
    def predict_procedure(self, node, data):
        if data[node["separate_feature"]].values < node["separate_value"]:
            if isinstance(node["left"], dict):
                return self.predict_procedure(node["left"], data)
            else:
                return node["left"]
        else:
            if isinstance(node["right"], dict):
                return self.predict_procedure(node["right"], data)
            else:
                return node["right"]

    def evaluate(self, mode):
        if mode == "train":
            pred = np.array(self.pred_train)
            target = self.dataset_train["y"]
            err_amount = (pred != target).sum()
            self.error_train = err_amount/len(self.dataset_train)
            print("Training error is: %.3f"%(self.error_train))
        elif mode == "test":
            pred = np.array(self.pred_test)
            target = self.dataset_test["y"]
            err_amount = (pred != target).sum()
            self.error_test = err_amount/len(self.dataset_test)
            print("Testing error is: %.3f"%(self.error_test))
        else:
            pred = np.array(self.pred_oob)
            target = self.dataset_train["y"]
            err_amount = (pred != target).sum()
            self.error_oob = err_amount/len(self.dataset_train)
            print("Oob validation error is: %.3f"%(self.error_oob))
```

main.py (http://main.py)



```python
def getData():
    data_train = pd.read_csv("./hw6_train.dat", sep=" ", header=None).rename(columns={10:"y"})
    data_test = pd.read_csv("./hw6_test.dat", sep=" ", header=None).rename(columns={10:"y"})
    return data_train, data_test

if __name__ == "__main__":
    train_set, test_set = getData()
    RF = RandomForest(train=train_set, test=test_set)
    RF.train(n_tree=2000, get_oob=True)
    # RF.predict(mode="train")
    # RF.predict(mode="test")
    # DT = DecisionTree(train=train_set, test=test_set)
    # DT.train()
    # DT.predict(mode="test")
```