Q1: AdaBoost

(a)Decision Trees

In [ ]:

```
#import libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import cross_val_score
```

In [1]:

```
# import data
from sklearn.datasets import load_digits
dataset = load_digits()
X = dataset['data']
y = dataset['target']
```

In [4]:

```
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import cross_val_score
max_depth = [1,2,5]
for i in max_depth:
    clf = DecisionTreeClassifier(max_depth=i,random_state=0)
    score = cross_val_score(clf, X, y, cv=6)
    print(f"cross-validation score(max_depth = {i}",score)
```

```
cross-validation score(max_depth = 1 [0.2        0.2        0.19333333 0.2006689  0.19397993 0.19732441]
cross-validation score(max_depth = 2 [0.32       0.28333333 0.30666667 0.30100334 0.34448161 0.32107023]
cross-validation score(max_depth = 5 [0.60666667 0.57       0.63       0.68896321 0.70234114 0.60869565]
```

(b) Adaboost

In [5]:

```
from sklearn.ensemble import AdaBoostClassifier
max_depth = [1,2,5]
for i in max_depth:
    dtree = DecisionTreeClassifier(max_depth=i,random_state=0)
    clf = AdaBoostClassifier(base_estimator = dtree,random_state=0)
    # clf.fit(X, y)
    score = cross_val_score(clf, X, y, cv=6)
    print(f"cross-validation score(max_depth = {i}",score)
```

```
cross-validation score(max_depth = 1 [0.28333333 0.26666667 0.26333333 0.28093645 0.20401338 0.26755853]
cross-validation score(max_depth = 2 [0.57666667 0.72666667 0.59333333 0.7826087  0.72575251 0.51170569]
cross-validation score(max_depth = 5 [0.88333333 0.88333333 0.87       0.93979933 0.94314381 0.8729097 ]
```

(c) Discussion

Initialize: $w^0 = 1/N$

Iterate:

For t = 1:T:

1. Train weak classifier using distribution w^t

2. Get weak hypothesis h_t : X with error $e_t = sum(w^t)(mistakepoints)$

3. Choose alpha = $log(1 - e_t/e_t)/2$

4. Add to ensemble: $F_t(x) = F_{t-1}(x) + alpha_t h_t(x)$

5. Update weights: $w^{t+1} = w^t e^{-y_i alpha_t h_t(x)}$, renormalize w^{t+1}

Q2. XGBoost

(1) XGBoost with library

In [63]:

```
# Loading
from sklearn import datasets
import xgboost as xgb

iris = datasets.load_iris()
X = iris.data
y = iris.target
```

In [64]:

```
# Spliting data into train and test
from sklearn.model_selection import train_test_split

X_train, X_test, Y_train, Y_test = train_test_split(X, y, test_size=0.2)
# X_train = X_train[:,0].reshape((-1,1))
# X_test = X_test[:,0].reshape((-1,1))

# print(Y_test)
print(Y_train.shape)
print(X_train.shape)
```

```
(120,)
(120, 4)
```

In [65]:

```
# Data transform into DMatrix format
D_train = xgb.DMatrix(X_train, label=Y_train)
D_test = xgb.DMatrix(X_test, label=Y_test)
print(D_train)
```

```
<xgboost.core.DMatrix object at 0x7fc0824b0d30>
```

In [66]:

```
# Defining a XGboost model
param = {
    'eta': 0.5,
    'max_depth': 10,
    'objective': 'multi:softprob',
    'num_class': 3}

steps = 5  # The number of training iterations
```

In [67]:

```
# Train the model
model = xgb.train(param, D_train, steps)
```

In [68]:

```
# Model Evaluation
import numpy as np
from sklearn.metrics import precision_score, recall_score, accuracy_score

preds = model.predict(D_test)
best_preds = np.asarray([np.argmax(line) for line in preds])

print("Precision = {}".format(precision_score(Y_test, best_preds, average='macro')))
print("Recall = {}".format(recall_score(Y_test, best_preds, average='macro')))
print("Accuracy = {}".format(accuracy_score(Y_test, best_preds)))
```

```
Precision = 1.0
Recall = 1.0
Accuracy = 1.0
```

In [34]:

```
# Dump Model (optional)
# model.dump_model('model.raw.txt')
```

In [69]:

```
# Write down your code here
eta = [0.1,0.3,0.5]
max_depth = [1,3,10]
for i in eta:
    for j in max_depth:
        param = {'eta': i, 'max_depth': j,  'objective': 'multi:softprob',  'num_class': 3}
        model = xgb.train(params = param, dtrain = D_train, num_boost_round=steps)
        preds = model.predict(D_test)
        best_preds = np.asarray([np.argmax(line) for line in preds])
        # print(best_preds)
        print(f"eta = {i},max_depth = {j}","Precision = {}".format(precision_score(Y_test, best_preds, average='ma
eta = 0.1,max_depth = 1 Precision = 0.9722222222222222 Recall = 0.9523809523809524 Accuracy = 0.9666666666666667
eta = 0.1,max_depth = 3 Precision = 1.0 Recall = 1.0 Accuracy = 1.0
eta = 0.1,max_depth = 10 Precision = 1.0 Recall = 1.0 Accuracy = 1.0
eta = 0.3,max_depth = 1 Precision = 0.9722222222222222 Recall = 0.9523809523809524 Accuracy = 0.9666666666666667
eta = 0.3,max_depth = 3 Precision = 1.0 Recall = 1.0 Accuracy = 1.0
eta = 0.3,max_depth = 10 Precision = 1.0 Recall = 1.0 Accuracy = 1.0
eta = 0.5,max_depth = 1 Precision = 1.0 Recall = 1.0 Accuracy = 1.0
eta = 0.5,max_depth = 3 Precision = 1.0 Recall = 1.0 Accuracy = 1.0
eta = 0.5,max_depth = 10 Precision = 1.0 Recall = 1.0 Accuracy = 1.0
```

I think max_depth is more sensitive, because when increase max_depth, Precision, Recall, Accuracy increase. When mex_depth = 3 and 10, the accuray are 1.0.

(2) XGBoost from scratch

In [13]:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import random
%matplotlib inline
```

In [2]:

```
# Data
year = [5,7,12,23,25,28,29,34,35,40,50,55]
salary = [82,80,103,118,172,127,204,189,99,166,221,240]
```

In [3]:

```
# Load the data
df = pd.DataFrame(columns=['Years','Salary'])
df.Years = year
df.Salary = salary
df.head()
# print(df.shape) # (10,2)
```

Out[3]:

	Years	Salary
0	5	82
1	7	80
2	12	103
3	23	118
4	25	172

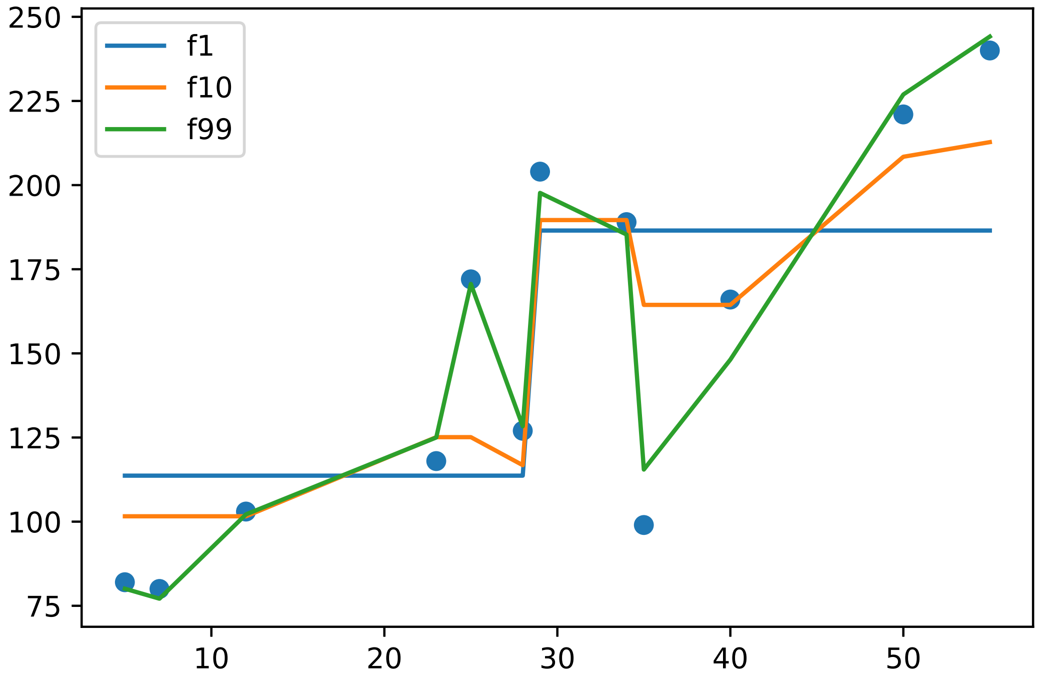
In [35]:

```
# Write down your code here
# f0 = np.mean(df["Salary"])
F = []
for i in range(100):
    if i>0:
        f = f + h
    else:
        f = np.mean(df["Salary"])
    F.append(f)
    y_f = df["Salary"]-f
    m = random.randrange(df.shape[0])
    h1 = np.mean(y_f[0:m+1])
    h2 = np.mean(y_f[m+1:])
    h = np.zeros(len(y_f))
    h[0:m+1] = h1
    h[m+1:] = h2
```

In [37]:

```
x = df["Years"]
plt.plot(x,F[1],label = "f1")
plt.plot(x,F[10],label = "f10")
plt.plot(x,F[99],label = "f99")
plt.scatter(x,df["Salary"])
plt.legend()
```

```
Out[37]: <matplotlib.legend.Legend at 0x7fc0501eb0d0>
```


f1
f10
f99
250
225
200
175
150
125
100
75
10
20
30
40
50