به نام خدا

سید پوریا احمدی 9723002

بخش اول:

برای این بخش از این لینک کمک گرفته شده است.

https://www.youtube.com/watch?v=UX0f9BNBcsY

در ابتدا کتابخانه لازم برای خواندن دیتا، split کردن آن و .. Import است.

```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from matplotlib import pyplot as plt
```

سپس تابع fitرا داریم که به آموزش داده های train می پردازد.

در ورودی x, y را دریافت میکند که داده های train می باشند .

پارامتر های لازم را در این تابع set میکنیم که شامل سه مورد هستند. (n_iter که تعداد تکرار حلقه اصلی است، learning rate و lambda param)

سپس b را در ابتدا صفر قرار می دهیم و w را برداری به طول تعداد feature های x با مقادیر صفر قرار میدهیم.

حال به تعداد n_iter که حلقه بیرونی است این کار را تکرار می کنیم:

در هر iteration به تعداد داده ها ابتدا index آن و خود داده را که با xi مشخص کرده ایم مقدار دهی می شوند (در for index, xi ...) و سپس الگوریتم linear svm پیاده سازی می شود.

در نهایت بردار w و مقدار b برگردانده می شوند

```
def fit (X, y):
    learning_rate=0.01
    lambda_param=0.01
    n_iters=10000
    b = 0
    w = np.zeros(X.shape[1])
    for i in range(n_iters):
        for index, xi in enumerate(X):
            condition = (np.dot(xi, w) - b) * y[index]   < 1
            if condition:
                b -= learning_rate * y[index]
                w -= learning_rate * ( 2 * lambda_param * w - np.dot(xi, y[index]))

            else:
                w -=  learning_rate * (2 * lambda_param * w)

    return b, w
```

حال برای visualize کردن خطوط بدست آمده از تابع visualize استفاده میکنیم که داده ها و bو w را به عنوان ورودی میگیرد.

ابتدا با استفاده از amin و amax کمترین و بیشترین داده در x[:,0] را میگیریم.

سپس مقادیر hyperplane ها را بدست آورده و در نهایت آنها را در این تابع رسم میکنیم.

```
def visualize (X, y ,b ,w):
    fig = plt.figure()
    ax = fig.add_subplot(1, 1, 1)
    plt.scatter(X[:, 0], X[:, 1], marker="o", c=y)

    x0_1 = np.amin(X[:, 0])
    x0_2 = np.amax(X[:, 0])
    z0 = -w[0]
    z1 = w[1]

    x1_1 , x1_2 , x1_1_m , x1_2_m , x1_1_p , x1_2_p= (z0 * x0_1 + b ) / z1 , (z0 * x0_2 + b ) / z1 ,  (z0 * x0_1

    ax.plot([x0_1, x0_2], [x1_1, x1_2], "y--")
    ax.plot([x0_1, x0_2], [x1_1_m, x1_2_m], "k")
    ax.plot([x0_1, x0_2], [x1_1_p, x1_2_p], "k")

    x1_min = np.amin(X[:, 1])
    x1_max = np.amax(X[:, 1])
    ax.set_ylim([x1_min - 3, x1_max + 3])

    plt.show()
```

سپس تابه accuracy را داریم که دقت را حساب میکند و داده های پیشبینی شده را با y_test مقایسه می کند و دقت را اعلام میکند.

```
def accuracy(y_test, y_predict):
    acc = 0
    for i in range (len(y_test)):
        if y_test[i] == y_predict[i]:
            acc += 1
    print(acc/len(y_test))
```

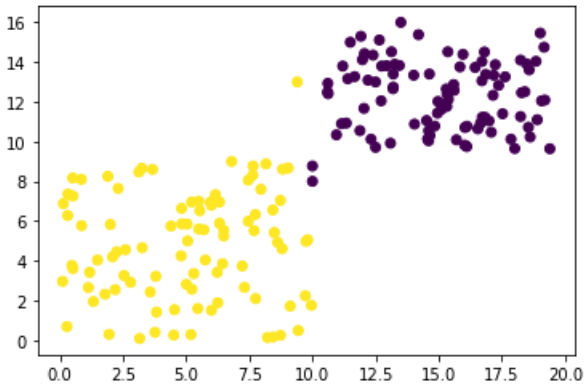
در نهایت در main

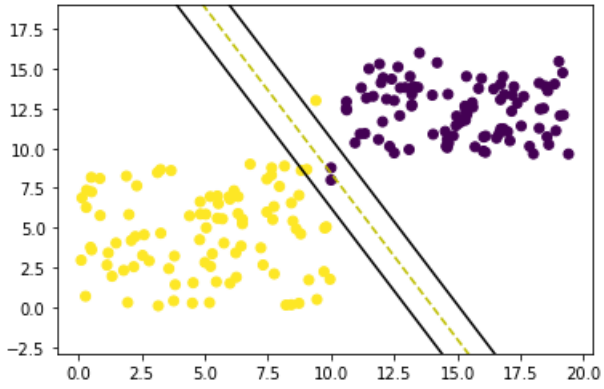
داده خوانده شده و داده ی هدف یعنی class از بقیه جدا می شود داده ها split می شوند و داده های x به آرایه numpy تبدیل می شوند و برای yها جاهایی که 0 هستند به -1 تبدیل می شوند و به آرایه numpyتبدیل می شوند. ابتدا داده ها را نمایش داده و سپس آموزش می دهیم و w ,b بدست آمده را پرینت میکنیم و در نهایت خطوط svm را پلات کرده و دقت را گزارش می دهیم .

برای predict داده ها از np.sign(np.dot(X_test, w) - b) استفاده میکنیم که علامت حاصله را مشخص میکند و به تابع accuracy پاس می دهیم.

```
if __name__ == "__main__":
    df=pd.read_csv("data1.csv")
    X = df.drop('Class', axis=1)
    y = df['Class']
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.1 )
    X_train = X_train.to_numpy()
    X_test = X_test.to_numpy()
    y_train = np.where(y_train == 0, -1, 1)
    y_test = np.where(y_test == 0, -1, 1)
    plt.scatter(X_train[:, 0], X_train[:, 1], marker="o", c=y_train)
    b,w = fit(X_train, y_train)
    print(b ,w)
    visualize (X_train, y_train , b, w)
    y_predict = np.sign(np.dot(X_test, w) - b)
    accuracy(y_test, y_predict)
```

```
-13.459999999999757 [-0.95904445 -0.45919427]
```



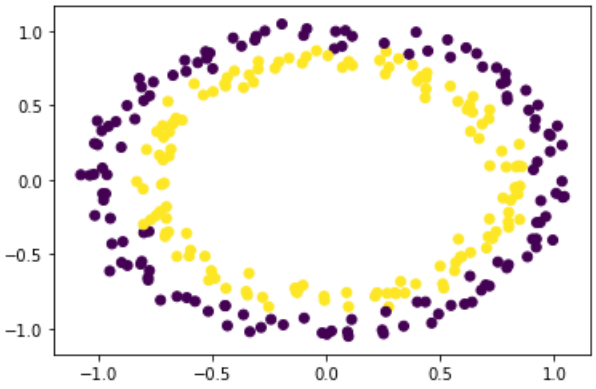


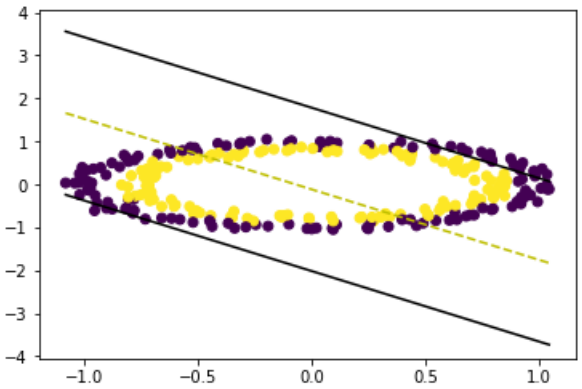

```
accuracy is :  1.0
```

همین کار را برای data2 انجام دادیم که مطابق انتظار جواب خوبی نگرفتیم که در عکس زیر مشاهده می شود.

```
if __name__ == "__main__":
    learning_rate=0.01
    lambda_param=0.01
    n_iters=10000
    df=pd.read_csv("data2.csv")
    X = df.drop('Class', axis=1)
    y = df['Class']
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.1 )
    X_train = X_train.to_numpy()
    X_test = X_test.to_numpy()
    y_train = np.where(y_train == 0, -1, 1)
    y_test = np.where(y_test == 0, -1, 1)
    plt.scatter(X_train[:, 0], X_train[:, 1], marker="o", c=y_train)
    b,w = fit(X_train, y_train)
    print(b ,w)
    visualize (X_train, y_train , b, w)
    y_predict = np.sign(np.dot(X_test, w) - b)
    accuracy(y_test, y_predict)
```

```
-0.060000000000000005 [0.8638463  0.52621122]
```






```
accuracy is :  0.3333333333333333
```

بخش بعدی :

حال برای دیتا 2 از nonlinear svm استفاده خواهیم کرد

ابتدا کتابخانه های لازم را import میکنبم :

```
import numpy as np
import matplotlib.pyplot as plt
from matplotlib.colors import ListedColormap
from sklearn.svm import SVC
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
```

سپس برای plot کردن decision boundary در این حالت یک تابع نوشتیم که برای نوشتن این تابع از (https://jakevdp.github.io/PythonDataScienceHandbook/05.07-support-vector-machines.html) استفاده شده است علاوه بر کانت های خود تابع کامنت هایی اضافه شده و کار هایی اضافه نیز انجام شده تا نشان داده شود فهم تابع انجام شده.

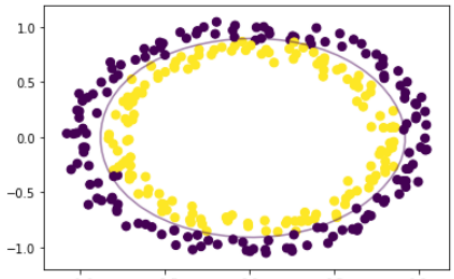
```
### https://jakevdp.github.io/PythonDataScienceHandbook/05.07-support-vector-machines.html
def plot_decision_boundary(model, ax=None):
    if ax is None:
        ax = plt.gca()

    xlim = ax.get_xlim()
    ylim = ax.get_ylim()

    # create grid to evaluate model
    x = np.linspace(xlim[0], xlim[1], 30)
    y = np.linspace(ylim[0], ylim[1], 30)
    Y, X = np.meshgrid(y, x)

# shape data
    xy = np.vstack([X.ravel(), Y.ravel()]).T

# get the decision boundary based on the model
    P = model.decision_function(xy).reshape(X.shape)

    # plot decision boundary
    ax.contour(X, Y, P,
               levels=[0], alpha=0.5,
               linestyles=['-'])

#You have everything you need to plot the decision boundary for this non-linear data. We can do that with
#a few lines of code that use the Matlibplot library, just like the other plots.

# plot data and decision boundary
```

سپس به تابع accuracy میرسم که عین قبل است

سپس به main میرسیم که مانند قبل داده خوانده شده تغییرات لازم انجام شده و در نهایت یک classifier (nonlinear_clf = svm.SVC(kernel='rbf', C=1.0)) که از rbf به عنوان کرنل استفاده میکند می سازیم و داده ها را با آن آموزش می دهیم و در نهایت داده ها را پلات میکنیم.

```python
if __name__ == "__main__":
    df=pd.read_csv("data2.csv")
    X = df.drop('Class', axis=1)

    y = df['Class']
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.1)
    X_train = X_train.to_numpy()
    X_test = X_test.to_numpy()
    y_train = np.where(y_train == 0, -1, 1)
    y_test = np.where(y_test == 0, -1, 1)
    X = X.to_numpy()
    y = y.to_numpy()
    plt.scatter(X[:, 0], X[:, 1], marker="o", c=y)
    nonlinear_clf = SVC(kernel='rbf', C=1.0)
    nonlinear_clf.fit(X_train, y_train)
    y_predict = nonlinear_clf.predict(X_test)
    accuracy(y_test, y_predict)
    plt.scatter(X[:, 0], X[:, 1], c=y, s=50)
    plot_decision_boundary(nonlinear_clf)
    plt.scatter(nonlinear_clf.support_vectors_[:, 0], nonlinear_clf.support_vectors_[:, 1], s=50, lw=1, facecolors='none')
    plt.show()
```

```
accuracy is :  1.0
```



بخش سوم :

برای این بخش از https://www.youtube.com/watch?v=Oq1cKjR8hNo&list=PLqnslRFeH2Upcrywf-u2etjdxxkL8nl7E&index=10 کمک گرفته شده است.

```python
from collections import Counter
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix
from matplotlib import pyplot as plt
import pandas as pd
```

```python
def bootstrap_sample(X, y):
    index = np.random.choice(X.shape[0], X.shape[0], replace=True)
    return X[index], y[index]
def most_common_label(y):
    return Counter(y).most_common(1)[0][0]
```

```python
def entropy(y):
    return -np.sum([p * np.log2(p) for p in np.bincount(y) / len(y) if p > 0])
```

```python
class Node:
    def __init__(
        self, feature=None, threshold=None, left=None, right=None, *, value=None
    ):
        self.feature = feature
        self.threshold = threshold
        self.left = left
        self.right = right
        self.value = value

    def is_leaf_node(self):
        return self.value is not None
```

همانطور که مشخص است کتابخانه ها import شده اند و کلاسی به نام گره دارین که ویژگی های آنرا مشخص میکند

تابعی به نام آنتروپی که آنرا حساب میکند .

سپس کلاس random forest را داریم که در عکس زیر مشخص است.

```
class RandomForest:
    def __init__(self, n_trees=10, min_samples_split=2, max_depth=100, n_feats=None):
        self.n_trees = n_trees
        self.min_samples_split = min_samples_split
        self.max_depth = max_depth
        self.n_feats = n_feats
        self.trees = []
    def predict(self, X):
        tree_preds = np.swapaxes(np.array([tree.predict(X) for tree in self.trees]), 0, 1)
        return np.array([most_common_label(tree_pred) for tree_pred in tree_preds])
    def fit(self, X, y):
        self.trees = []
        for _ in range(self.n_trees):
            tree = DecisionTree(
                min_samples_split=self.min_samples_split,
                max_depth=self.max_depth,
                n_feats=self.n_feats,
            )
            X_samp, y_samp = bootstrap_sample(X, y)
            tree.fit(X_samp, y_samp)
            self.trees.append(tree)
```

در تابع fit به تعداد درخت هایی که داریم در یک حلقه یک درخت تصمیم هست که مقادیر لازم را به آن می دهیم.

حال به کاربرد bootstrap sample میرسیم که وظیفه دارد random subset ی انتخاب شود.

در ادامه درخت را آموزش می دهیم (که در تمرین قبلی بود) و در نهایت آن درخت را به لیست درخت هامون اضافه می کنیم.

حال نوبت به predict است که ابتدا به تعداد درخت های لیستی که داریم predict را برای آن درخت انجام می دهیم و در یک numpy array میریزیم. و بیشترین label پیش بینی شده را در غالب یک آرایه بر میگردانیم.

عکس کلاس درخت را در زیر قابل مشاهده است و کامنت گذاری شده است .

هرچند که در تمرین قبلی نیز استفاده شده است اما به طور کلی به ترتیب ابتدا داده ها آموزش داده می شوند و برای بزرگ کردن درخت یک تابع داریم که بر اساس information gain درخت را گسترش می دهد(تابع grow tree)

هم چنین توابعی در برای محاسبه information gain و split کردن هر گره داریم.

```python
class DecisionTree:
    def __init__(self, min_samples_split=2, max_depth=100, n_feats=None):
        self.min_samples_split = min_samples_split
        self.max_depth = max_depth
        self.n_feats = n_feats
        self.root = None

    def fit(self, X, y):
        self.n_feats = X.shape[1] if not self.n_feats else min(self.n_feats, X.shape[1])
        self.root = self._grow_tree(X, y)

    def predict(self, X):
        return np.array([self._traverse_tree(x, self.root) for x in X])

    def _grow_tree(self, X, y, depth=0):
        if depth >= self.max_depth or len(np.unique(y)) == 1 or X.shape[0] < self.min_samples_split:
            return Node(value=self._most_common_label(y))

        # greedily select the best split according to information gain
        best_feat, best_thresh = self._best_criteria(X, y, np.random.choice(X.shape[1], self.n_feats, replace=False))

        # grow the children that result from the split
        left_idxs, right_idxs = self._split(X[:, best_feat], best_thresh)
        left = self._grow_tree(X[left_idxs, :], y[left_idxs], depth + 1)
        right = self._grow_tree(X[right_idxs, :], y[right_idxs], depth + 1)
        return Node(best_feat, best_thresh, left, right)

    def _best_criteria(self, X, y, feat_idxs):
        best_gain = -1
        for feat_idx in feat_idxs:
            for threshold in np.unique(X[:, feat_idx]):
                gain = self._information_gain(y, X[:, feat_idx], threshold)
                if gain > best_gain:
                    best_gain = gain
                    split_idx = feat_idx
                    split_thresh = threshold

        return split_idx, split_thresh

    def _information_gain(self, y, X_column, split_thresh):

        # generate split
        left_idxs, right_idxs = self._split(X_column, split_thresh)
        # compute the weighted avg. of the loss for the children
        e_l, e_r = entropy(y[left_idxs]), entropy(y[right_idxs])
        n_l, n_r = len(left_idxs), len(right_idxs)
        child_entropy = (n_l / len(y)) * e_l + (n_r / len(y)) * e_r
        if len(left_idxs) == 0 or len(right_idxs) == 0:
            return 0
        # information gain is difference in loss before vs. after split
        return entropy(y) - child_entropy

    def _split(self, X_column, split_thresh):
        left_idxs = np.argwhere(X_column <= split_thresh).flatten()
        right_idxs = np.argwhere(X_column > split_thresh).flatten()
        return left_idxs, right_idxs

    def _traverse_tree(self, x, node):
        if node.is_leaf_node():
            return node.value

        if x[node.feature] <= node.threshold:
            return self._traverse_tree(x, node.left)
        return self._traverse_tree(x, node.right)

    def _most_common_label(self, y):
        counter = Counter(y)
        most_common = counter.most_common(1)[0][0]
        return most_common
```

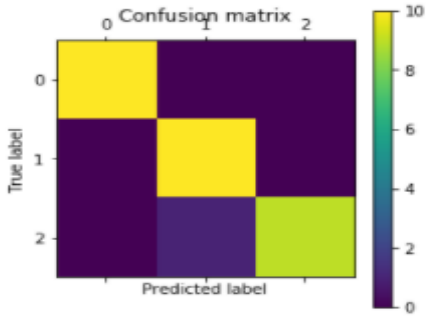
توابع accuracy و در نهایت main نیز هم در بخش 1و2 و هم در بخش 4 نیز توضیحات کافی داده شده است.

اما با توجه به اینکه species دارای سه گونه هستند y را به صورت آرایه از 1و2و3 تبدیل میکنیم که هر عدد نماینده یک گونه باشد و در نهایت داده های تست را پیش بینی کرده و دقت را محاسبه میکنیم.

```python
def accuracy(y_true, y_pred):
        cm = confusion_matrix(y_true, y_pred)
        print(cm)

        accuracy = np.sum(y_true == y_pred) / len(y_true)
        plt.matshow(cm)
        plt.title('Confusion matrix')
        plt.colorbar()
        plt.ylabel('True label')
        plt.xlabel('Predicted label')
        plt.show()
        return accuracy
```

```python
if __name__ == "__main__":
    df=pd.read_csv("data3.csv")
    X = df.drop('species', axis=1)
    y = df['species']
    X = X.to_numpy()
    y1 = np.where(y == 'Iris-setosa',1,-1)
    y2 = np.where(y == 'Iris-versicolor',2 ,-1)
    y3 = np.where(y == 'Iris-virginica',3,-1)
    y = y.to_numpy()
    for i in range(len(y)):
        if y1[i] == 1 :
            y[i] = 1
        elif y2[i] == 2:
            y[i] = 2
        elif y3[i] == 3:
            y[i] = 3
    y = y.astype(int)
    X_train, X_test, y_train, y_test = train_test_split(
        X, y, test_size=0.2)
    clf = RandomForest(n_trees=8, max_depth=10)

    clf.fit(X_train, y_train)
    y_pred = clf.predict(X_test)
    acc = accuracy(y_test, y_pred)

    print("Accuracy:", acc)
```

```
[[10  0  0]
 [ 0 10  0]
 [ 0  1  9]]
```




```
Accuracy: 0.9666666666666667
```

همچنین در مورد درخواست سوال درباره تعداد درخت ها به شکل زیر عمل میکنیم.

در یک حلقه از 1 تا 10 درخت را تست میکنیم . جواب ها به صورت زیر است : (چند بار تست گرفته شد)

```
Accuracy  for 1 tree is : 0.8666666666666667
Accuracy  for 2 tree is : 0.9
Accuracy  for 3 tree is : 0.9
Accuracy  for 4 tree is : 0.8666666666666667
Accuracy  for 5 tree is : 0.9
Accuracy  for 6 tree is : 0.9
Accuracy  for 7 tree is : 0.9
Accuracy  for 8 tree is : 0.9
Accuracy  for 9 tree is : 0.9
Accuracy  for 10 tree is : 0.9
Accuracy  for 11 tree is : 0.9
```

```
Accuracy  for 1 tree is : 0.9333333333333333
Accuracy  for 2 tree is : 0.8666666666666667
Accuracy  for 3 tree is : 0.9333333333333333
Accuracy  for 4 tree is : 0.9
Accuracy  for 5 tree is : 0.9333333333333333
Accuracy  for 6 tree is : 0.9
Accuracy  for 7 tree is : 0.9333333333333333
Accuracy  for 8 tree is : 0.9333333333333333
Accuracy  for 9 tree is : 0.9333333333333333
Accuracy  for 10 tree is : 0.9333333333333333
Accuracy  for 11 tree is : 0.9333333333333333
```

```
Accuracy  for 1 tree is : 0.9
Accuracy  for 2 tree is : 0.8666666666666667
Accuracy  for 3 tree is : 0.9666666666666667
Accuracy  for 4 tree is : 0.9666666666666667
Accuracy  for 5 tree is : 0.9666666666666667
Accuracy  for 6 tree is : 0.9666666666666667
Accuracy  for 7 tree is : 0.9666666666666667
Accuracy  for 8 tree is : 0.9666666666666667
Accuracy  for 9 tree is : 0.9666666666666667
Accuracy  for 10 tree is : 0.9666666666666667
Accuracy  for 11 tree is : 0.9666666666666667
```

```
Accuracy  for 1 tree is : 0.9
Accuracy  for 2 tree is : 0.9
Accuracy  for 3 tree is : 0.9
Accuracy  for 4 tree is : 0.9333333333333333
Accuracy  for 5 tree is : 0.9333333333333333
Accuracy  for 6 tree is : 0.9333333333333333
Accuracy  for 7 tree is : 0.9
Accuracy  for 8 tree is : 0.9666666666666667
Accuracy  for 9 tree is : 0.9333333333333333
Accuracy  for 10 tree is : 0.9333333333333333
Accuracy  for 11 tree is : 0.9333333333333333
```

همانطور که پیش بینی میشد به طور کلی وقتی تعداد درخت ها بالا برود نتایج بهتر می شود (در تعداد خیلی بالا ممکنه overfitting داشته باش

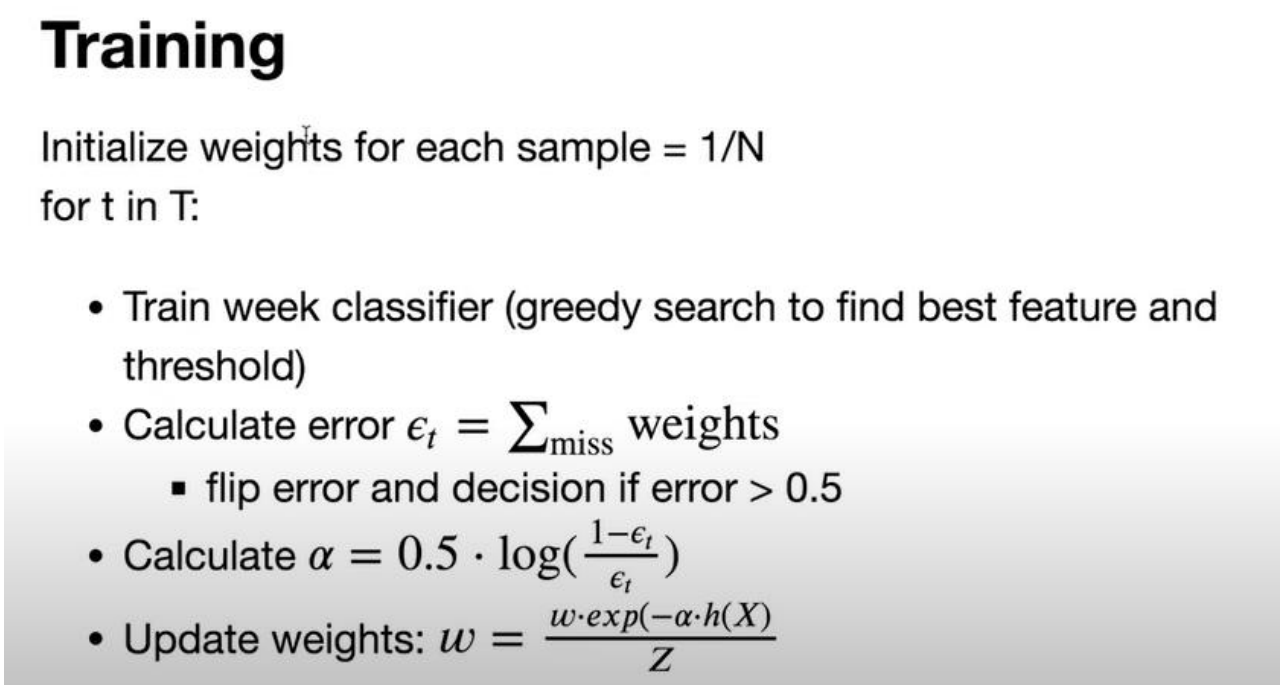
بخش چهارم:

برای این بخش از لینک (https://www.youtube.com/watch?v=wF5t4Mmv5us&list=PLqnslRFeH2Upcrywf-u2etjdxxkL8nl7E&index=14) کمک گرفته شده است.

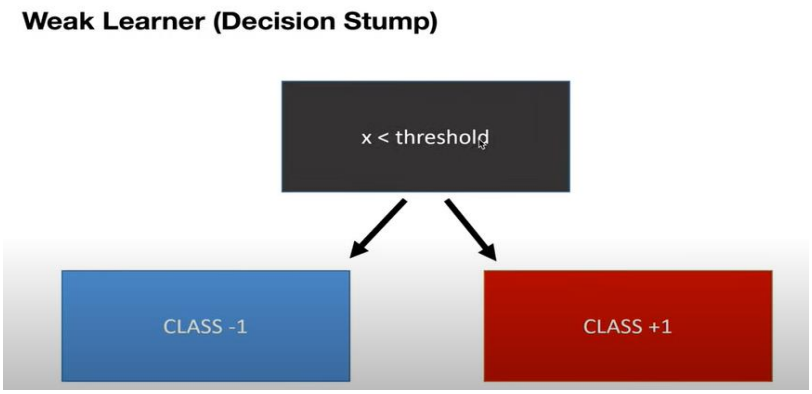
ابتدا کتابخانه های لازم را import میکنیم.

```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from matplotlib import pyplot as plt
from sklearn.metrics import confusion_matrix
```

همچنین الگوریتم پیاده سازی شده به این صورت است.



سپس باید یک کلاس به نام decision stump تعریف می کنیم که weak learner ما است و کار آن در عکس زیر مشخص است



همچنین کد پیاده سازی شده آن به این صورت است.

```python
# Decision stump used as weak classifier
class DecisionStump:
    def __init__(self):
        self.polarity = 1
        self.feature_idx = None
        self.threshold = None
        self.alpha = None
    def predict(self, X):
        predictions = np.ones(X.shape[0])
        X_column = X[:, self.feature_idx]

        # default case
        if self.polarity == 1:
            predictions[X_column < self.threshold] = -1
        else:
            predictions[X_column > self.threshold] = -1
        return predictions
```

سپس به خود adaboost می رسیم

در تابع fit ابتدا وزن ها را مقدار دهی میکنیم سپس لیستی از classifier ها می سازیم و در ابتدا لیستی خالی است.

حال iteration انجام می شود، که به تعداد classifier هایی که مشخص کرده ایم این حلقه انجام شود.(n_clf)

ما میخواهیم greedy search را انجام دهیم در ابتدا classifier را به وجود می آوریم. (استفاده از decision stump)

Min error رو تعریف می کنیم زیر می خواهیم بهترین feature value , split feature و split threshold را پیدا کنیم که min error حداقل شود. و در ابتدا مقدار بسیار زیادی قرار می دهیم .

حال رو کل feature ها iterate انجام می دهیم و در حلقه بعدی کل مقادیر unique را برای کل ویژگی ها پیدا میکنیم و در این حلقه polarity را برابر یک قرار می دهیم و prediction ها را انجام میدهیم و در مواردی که column های x ما از threshold کمتر باشند را منفی یک قرار میدهیم و error می شود تعداد مواردی که اشتباه دسته بندی شده اند.

اگر خطا بیشتر از 0.5 باشد نیز آنرا تغییر می دهیم و اگر از min errorکمتر باشد min error را آپدیت میکنیم و میخواهیم به عنوان دسته بند بهتر انتخابش کنیم و از فرمولی که در عکس بالا هست الفا را بدست می آوریم. و w هارا آپدست می کنیم و آنرا در لیستی که از دسته بند ها گذاشتیم ذخیره می کنیم.

برای predict هم برای تمام دسته بند ها prediction * alpha را حساب میکنیم و سپس آنها را جمع میکنیم و در نهایت به علامت آنها توجه میکنیم.

```
class Adaboost:
    def __init__(self, n_clf=5):
        self.n_clf = n_clf
        self.clfs = []
    def fit(self, X, y):
        w = np.full(X.shape[0], (1 / X.shape[0]))
        self.clfs = []
        for _ in range(self.n_clf):
            clf = DecisionStump()
            min_error = float("inf")
            for xi in range(X.shape[1]):
                for threshold in np.unique(X[:, xi]):
                    p = 1
                    predictions = np.ones(X.shape[0])
                    predictions[X[:, xi] < threshold] = -1
                    error = sum(w[y != predictions])
                    if error > 0.5:
                        error = 1 - error
                        p = -1
                    if error < min_error:
                        clf.polarity = p
                        clf.threshold = threshold
                        clf.feature_idx = xi
                        min_error = error
            predictions = clf.predict(X)
            clf.alpha = 0.5 * np.log((1.0 - min_error + 1e-10) / (min_error + 1e-10))
            predictions = clf.predict(X)
            w *= np.exp(-clf.alpha * y * predictions)
            w /= np.sum(w)
            self.clfs.append(clf)
    def predict(self, X):
        return np.sign(np.sum([clf.alpha * clf.predict(X) for clf in self.clfs], axis=0))
```

برای قسمت pre processing با توجه به اینکه در missing value AGE داشتیم ابتدا جا های خالی را با mean پر کرده و سپس برای ستون sex هم label جدید قرا میدهیم تا همه ویژگی ها عددی باشند.

```
def pre_processing(dataframe):
#     print(dataframe.isna().sum())
    dataframe = dataframe.fillna(df['Age'].mean())
    dataframe = dataframe.drop('PassengerId', axis = 1)
#     print(dataframe['Sex'].unique())
    dataframe['Sex']= dataframe['Sex'].map({'male': 0, 'female': 1}).astype(int)
    return dataframe
```

همچنین برای اینکه بهتر بتوانیم نتیجه بگیریم داده هارا طبق این تابع scale میکنیم.

و در تابع accuracy دقت مدل محاسبه می شود و confusion matrix را نیز نمایش می دهیم

```python
def scale(X):
    new = X - np.mean(X, axis=0)
    return new / np.std(new, axis=0)
```

```python
def accuracy(y_true, y_pred):
    cm = confusion_matrix(y_true, y_pred)
    print(cm)
    accuracy = np.sum(y_true == y_pred) / len(y_true)
    plt.matshow(cm)
    plt.title('Confusion matrix')
    plt.colorbar()
    plt.ylabel('True label')
    plt.xlabel('Predicted label')
    plt.show()
    return accuracy
```

در نهایت در main داده ها را خوانده پیش پردازش را روی آنها انجام می دهیم .

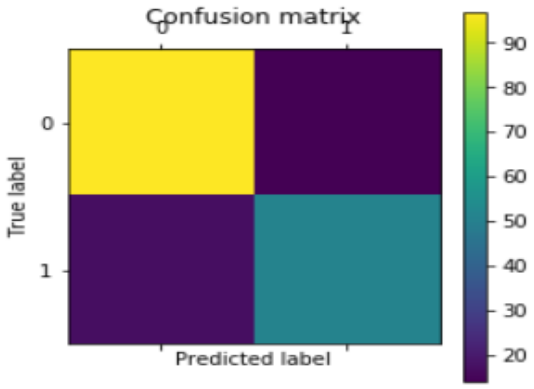
کلاس هدف را جدا کرده و داده هارا به numpy تبدیل کرده و int میکنیم .

داده های x را scale میکنیم.

جاهایی که y برابر 0 است منفی یک قرار می دهیم و split را انجام میدهیم و سپس داده های آموزش را به مدل می دهیم تا آموزش ببینند و دقت را در انتها نمایش می دهیم.

```python
if __name__ == "__main__":
    df=pd.read_csv("data4.csv")
    df = pre_processing (df)
    X = df.drop('Target_class', axis=1)
    y = df['Target_class']
    X = X.to_numpy()
    y = y.to_numpy()
    X = X.astype(int)
    y = y.astype(int)
    X = scale(X)
    y = np.where(y == 0, -1, 1)
    X_train, X_test, y_train, y_test = train_test_split(
        X, y, test_size=0.2 )
    # Adaboost classification with 5 weak classifiers
    clf = Adaboost(n_clf=5)
    clf.fit(X_train, y_train)
    y_pred = clf.predict(X_test)
    acc = accuracy(y_test, y_pred)
    print("Accuracy:", acc)
```

```
[[97 14]
 [17 51]]
```




```
Accuracy: 0.8268156424581006
```