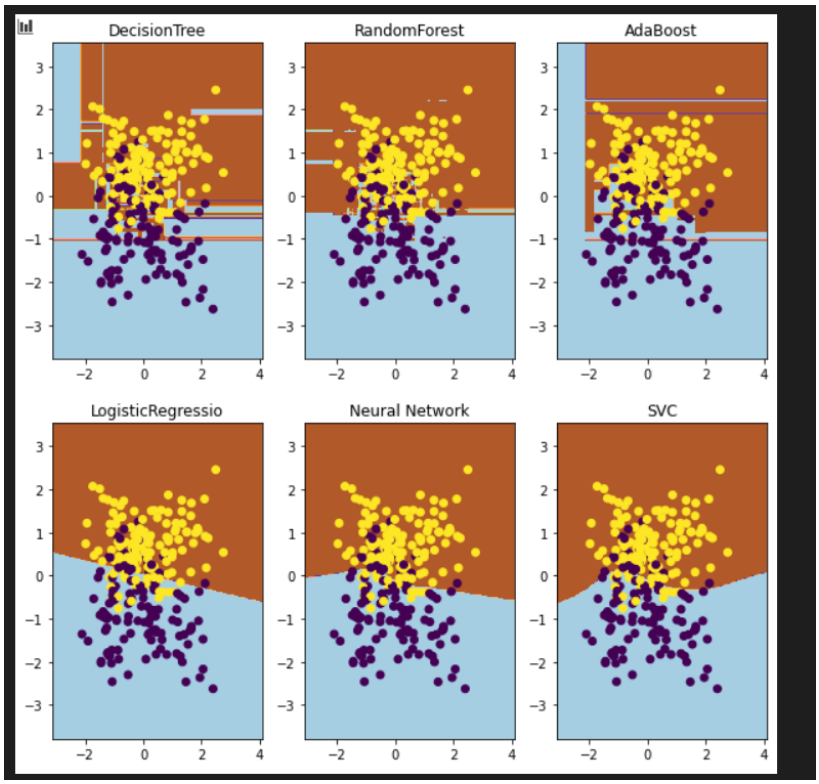
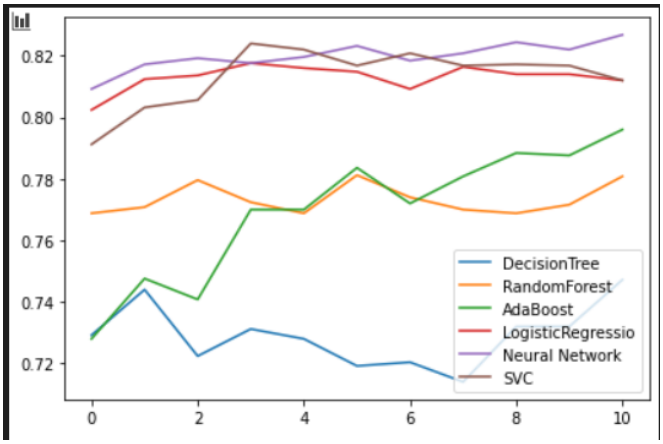
Q1

(a)



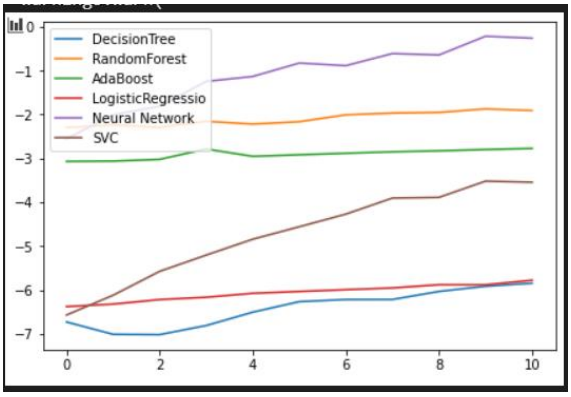

```
X,y = create_dataset()
clfdic = dict()
train_X,test_X,train_y,test_y = train_test_split(X,y,test_size=0.2,random_state = 45)
clf_DEC = DecisionTreeClassifier()
clf_DEC = clf_DEC.fit(train_X,train_y)
clfdic["DecisionTree"] = clf_DEC
clf_RAN = RandomForestClassifier()
clf_RAN = clf_RAN.fit(train_X, train_y)
clfdic["RandomForest"] = clf_RAN
clf_ADA = AdaBoostClassifier()
clf_ADA = clf_ADA.fit(train_X, train_y)
clfdic["AdaBoost"] = clf_ADA
clf_LOG = LogisticRegression()
clf_LOG = clf_LOG.fit(train_X, train_y)
clfdic["LogisticRegressio"] = clf_LOG
clf_NEU = MLPClassifier()
clf_NEU = clf_NEU.fit(train_X, train_y)
clfdic["Neural Network"] = clf_NEU
clf_SVC = SVC()
clf_SVC = clf_SVC.fit(train_X, train_y)
clfdic["SVC"] = clf_SVC
fig, ax = plt.subplots(2,3, figsize=(10,10))
i = 0
for key,value in clfdic.items():
    plotter(value, X, test_X, test_y, key, ax.flat[i])
    i=i+1
```

(b&&c)




```
numset = [50,100,200,300,400,500,600,700,800,900,1000]
total = 0
pointdic = dict()
timedic = dict()
for key,value in clfdic.items():
    pointdic[key] = []
    timedic[key] = []
    for num in numset:
        start = time.time()
        for i in range (10):
            sample_x = [[] for i in range(len(train_X))]
            sample_list = [i for i in range(len(train_X))]
            sample_list = random.sample(sample_list, num)
            sample_x = [train_X[i] for i in sample_list]
            sample_y = [train_y[i] for i in sample_list]
            value.fit(sample_x,sample_y)
            y_pred = value.predict(test_X)
            total += accuracy_score(test_y,y_pred)
            #calculate accuracy at here
            #todo
        end = time.time()
        running_time = end-start
        timedic[key].append(math.log(running_time/10))
        pointdic[key].append(total/10)
        total = 0
#plt.gca().get_color_cycle(['blue','orange','green','salmon','red','yellow'])
for key,value in timedic.items():
    plt.plot(value,label = key)
plt.legend(loc = 'upper left' )
plt.tight_layout() # plot formatting
plt.show()
```

```
for key,value in pointdic.items():
    plt.plot(value,label = key)
plt.legend(loc = 'lower right' )
plt.tight_layout() # plot formatting
plt.show()
```



Firstly, we can find Decision tree is the simplest classifier which take shortest time and lowest accuracy.

Logistic Regression, Neural Network, random forest are more accurate than the others because base on bias-variance decomposition The less volatility, the better classifier and Neural Network takes the longest time in those classifier ,accuracy of random forest it too low, so logistic Regression takes not that much time which I think are the better classifiers

The accuracy of AdaBoost increase if the size of training set increase, the others wont be that much affected by change the size of training set

And also, time spend for every classifier will increase if the size of training set increase.

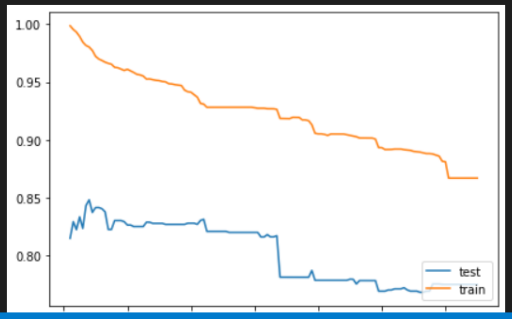
(d)

```
X,y = create_dataset(n = 2000,nf = 20,nr=12,ni = 8,random_state = 25)
train_X,test_X,train_y,test_y = train_test_split(X,y,test_size=0.5,random_state = 15)
clf_DEC = DecisionTreeClassifier(random_state=0)
clf_DEC = clf_DEC.fit(train_X,train_y)
y_pred = clf_DEC.predict(test_X)
print("test_acc: ",accuracy_score(test_y,y_pred))
y_pred = clf_DEC.predict(train_X)
print("train_acc: ",accuracy_score(train_y,y_pred))
```

```
test_acc:  0.814
train_acc:  1.0
```

(e)

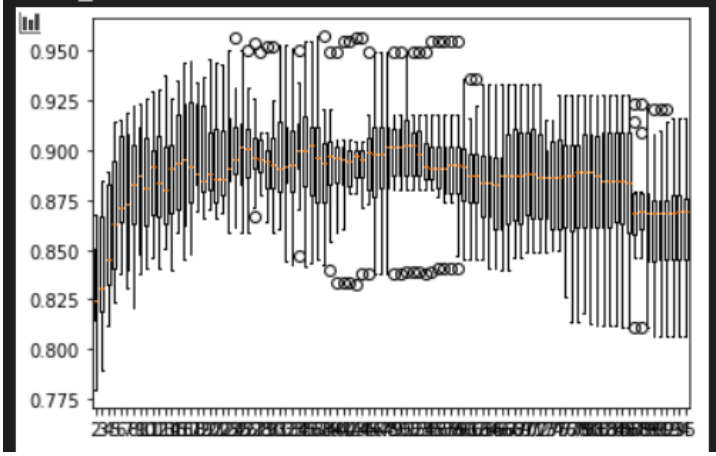
```
listtest = []
listtrain = []
for i in range(2,131):
    clf_DEC = DecisionTreeClassifier(min_samples_leaf=i,random_state= 0 )
    clf_DEC = clf_DEC.fit(train_X,train_y)
    y_pred = clf_DEC.predict(test_X)
    listtest.append(roc_auc_score(test_y, y_pred))
    y_pred = clf_DEC.predict_proba(train_X)

    listtrain.append(roc_auc_score(train_y, y_pred[:,1]))
plt.plot([i for i in range(2,131)],listtest,label = "test")
plt.plot([i for i in range(2,131)],listtrain,label = "train")
plt.legend(loc = 'lower right' )
plt.tight_layout() # plot formatting
plt.show()
```



(f)

```
highestk = 0
best = 0
tenfolds_x = np.array_split(train_X,10)
tenfolds_y = np.array_split(train_y,10)
box = [[] for i in range(94)]
for i in range(2,96):
    for j in range (10):
        test_X_f = tenfolds_x[j]
        test_y_f = tenfolds_y[j]
        #add 9 sets together
        train_X_f = tenfolds_x[:j]
        train_X_f.extend(tenfolds_x[j+1:])
        nine_x = np.reshape(train_X_f,(-1,20))
        #add 9 sets together
        train_y_f = tenfolds_y[:j]
        train_y_f.extend(tenfolds_y[j+1:])
        nine_y = list(np.array(train_y_f).flatten())
        #fit
        clf_DEC = DecisionTreeClassifier(min_samples_leaf=i,random_state = 0)
        clf_DEC = clf_DEC.fit(nine_x,nine_y)
        y_pred = clf_DEC.predict_proba(test_X_f)
        box[i-2].append(roc_auc_score(test_y_f,y_pred[:,1]))
for i in range(94):
    if sum(box[i]) > best:
        best = sum(box[i])
        highestk = i+2
print("the value of k that results in the highest average CV score is ",highestk)
clf_DEC = DecisionTreeClassifier(min_samples_leaf = highestk,random_state = 0)
clf_DEC = clf_DEC.fit(train_X,train_y)
y_pred = clf_DEC.predict(test_X)
print("test_acc: ",accuracy_score(test_y,y_pred))
y_pred = clf_DEC.predict(train_X)
print("train_acc: ",accuracy_score(train_y,y_pred))
plt.boxplot(box,positions = [i for i in range(2,96)])
plt.show()
```

```
the value of k that results in the highest average CV score is  24
test_acc:  0.825
train_acc:  0.881
```



(g)

```
g_dic=dict()
clf_DEC = DecisionTreeClassifier(random_state = 0)
g_dic["min_samples_leaf"] = [i for i in range(2,96)]
clf = GridSearchCV(clf_DEC,g_dic,scoring = 'roc_auc',cv = 10)
clf.fit(train_X, train_y)
clf.best_estimator_
```

```
DecisionTreeClassifier(min_samples_leaf=28, random_state=0)
```

The reason why it different to mine result is we split the data in 10 and use one of each to be the test data is not strict, randomly select 100 data from the train_data is better .

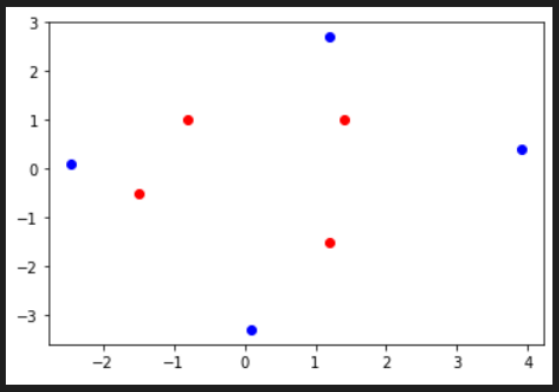
```
clf_DEC = DecisionTreeClassifier(min_samples_leaf = 28,random_state = 0)
clf_DEC = clf_DEC.fit(train_X,train_y)
y_pred = clf_DEC.predict(test_X)
print("test_acc: ",accuracy_score(test_y,y_pred))
y_pred = clf_DEC.predict(train_X)
print("train_acc: ",accuracy_score(train_y,y_pred))
```

```
test_acc:  0.828
train_acc:  0.878
```

Q2

(a)

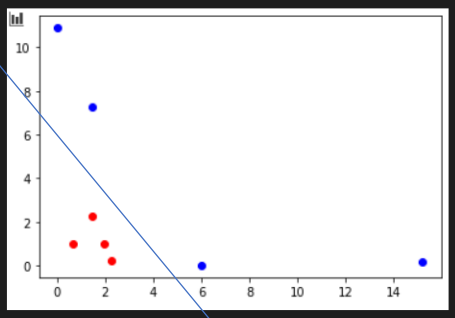
```
x1 = [-0.8,3.9,1.4,0.1,1.2,-2.45,-1.5,1.2]
x2 = [1,0.4,1,-3.3,2.7,0.1,-0.5,-1.5]
y = [1,-1,1,-1,-1,-1,1,1]
for i in range(8):
    if y[i] == 1:
        plt.scatter(x1[i],x2[i],color = "red")
    else:
        plt.scatter(x1[i],x2[i],color = "blue")
```




```
m = 0, d = 2
```

(b)

```
for i in range(8):
    if y[i] == 1:
        plt.scatter(x1[i]**2,x2[i]**2,color = "red")
    else:
        plt.scatter(x1[i]**2,x2[i]**2,color = "blue")
```




```
feature vector : [x1^2,x2^2,2*0.5x1*x2]
```

Yes it is linear separable

(c)

```
check = 0
w = np.array([1,1,1,1],dtype = float)
iterdict = dict()
Iter = 0
while check == 0 :
    check = 1
    for i in range (8):
        new_x = np.array([1,x1[i]**2,x2[i]**2,2**0.5*x1[i]*x2[i]])
        w_sum = w.dot(new_x)
        Iter+=1
        if y[i] * w_sum <=0:
            iterdict[Iter] = []
            check = 0
            w[0] = w[0] + 0.2*y[i]
            iterdict[Iter].append(w[0])
            w[1] = w[1] + 0.2*(x1[i]**2)*y[i]
            iterdict[Iter].append(w[1])
            w[2] = w[2] + 0.2*(x2[i]**2)*y[i]
            iterdict[Iter].append(w[2])
            w[3] = w[3] + 0.2*(2**0.5)*x1[i]*x2[i]*y[i]
            iterdict[Iter].append(w[3])
print(w)
field_names = ("Iteration No.",'w0','w1','w2','w3')
table = PrettyTable(field_names=field_names)
table.add_row([0,1,1,1,1])
for key,value in iterdict.items():
    table.add_row([key,round(value[0],4),round(value[1],4),round(value[2],4),round(value[3],4)])
print(table)
```

```
 3.          -0.62        -0.654        0.07793276]
```

Iteration No.	w0	w1	w2	w3
0	1	1	1	1
2	0.8	-2.042	0.968	0.5588
3	1.0	-1.65	1.168	0.9547
4	0.8	-1.652	-1.01	1.0481
7	1.0	-1.202	-0.96	1.2602
8	1.2	-0.914	-0.51	0.7511
9	1.4	-0.786	-0.31	0.5248
13	1.2	-1.074	-1.768	-0.3916
15	1.4	-0.624	-1.718	-0.1795
16	1.6	-0.336	-1.268	-0.6886
19	1.8	0.056	-1.068	-0.2926
22	1.6	-1.1445	-1.07	-0.2233
23	1.8	-0.6945	-1.02	-0.0112
24	2.0	-0.4065	-0.57	-0.5203
27	2.2	-0.0145	-0.37	-0.1243
30	2.0	-1.215	-0.372	-0.055
31	2.2	-0.765	-0.322	0.1571
32	2.4	-0.477	0.128	-0.352
36	2.2	-0.479	-2.05	-0.2587
40	2.4	-0.191	-1.6	-0.7678
43	2.6	0.201	-1.4	-0.3718
46	2.4	-0.9995	-1.402	-0.3025
47	2.6	-0.5495	-1.352	-0.0904
48	2.8	-0.2615	-0.902	-0.5995
54	2.6	-1.462	-0.904	-0.5302
55	2.8	-1.012	-0.854	-0.318
59	3.0	-0.62	-0.654	0.0779

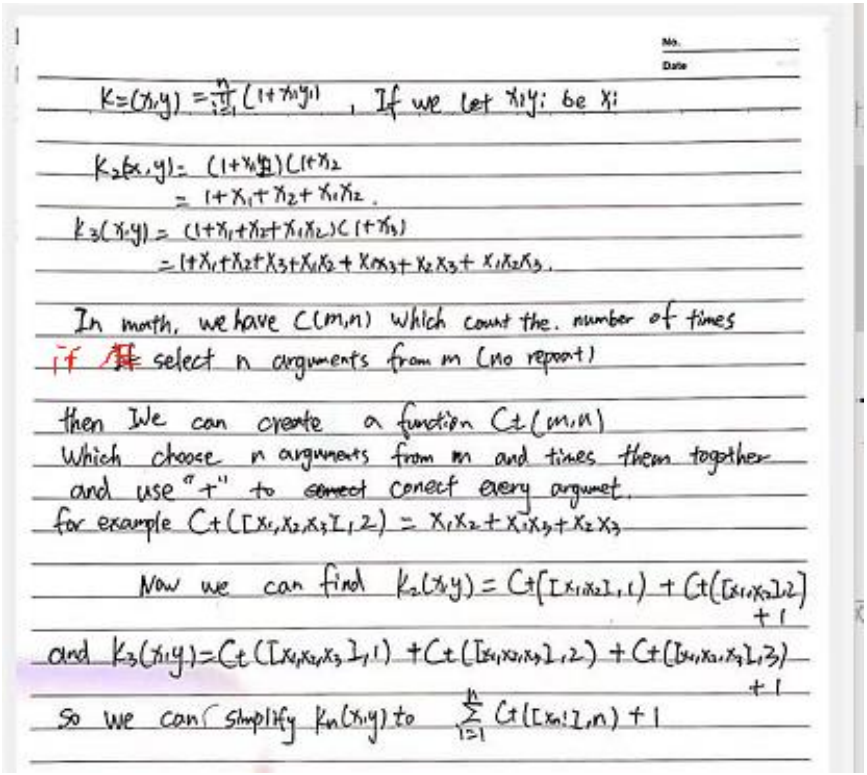
Last vector is [3,-0.62,-0.654,0.0779]

```
final_w = []
for i in range (8):
        new_x = np.array([1,x1[i]**2,x2[i]**2,2**0.5*x1[i]*x2[i]])
        w_sum = w.dot(new_x)
        final_w.append(w_sum)
field_names = ("xi",'ϕ(xi)','yiϕ^T*(xi)w*')
table = PrettyTable(field_names=field_names)
for i in range(8):
    y_final = final_w[i]*y[i]
    table.add_row([('(%s,%s)'%(x1[i],x2[i])),('(%f,%f,%f)'%(x1[i]**2,x2[i]**2,2**0.5*x1[i]*x1[i])),round(y_final,4)])
print(table)
```

xi	ϕ(xi)	yiϕ^T*(xi)w*
(-0.8,1)	(0.640000,1.000000,0.905097)	0.0
(3.9,0.4)	(15.210000,0.160000,21.510188)	-6.3629
(1.4,1)	(1.960000,1.000000,2.771859)	1.2851
(0.1,-3.3)	(0.010000,10.890000,0.014142)	-0.0
(1.2,2.7)	(1.440000,7.290000,2.036468)	-0.0
(-2.45,0.1)	(6.002500,0.010000,8.488817)	-0.7551
(-1.5,-0.5)	(2.250000,0.250000,3.181981)	0.0
(1.2,-1.5)	(1.440000,2.250000,2.036468)	0.0

All the yi*xi*wi > 0 so it is converged

(d



$K=(x,y) = \prod_{i=1}^{n}(1+x_i y_i)$, If we let $x_i y_i$ be X_i

$K_2(x,y) = (1+x_1 y_1)(1+x_2)$

$= 1+X_1+X_2+X_1X_2$.

$K_3(x,y) = (1+X_1+X_2+X_1X_2)(1+X_3)$

$= 1+X_1+X_2+X_3+X_1X_2+X_1X_3+X_2X_3+X_1X_2X_3$.

In math, we have C(m,n) which count the number of times if select n arguments from m (no repeat)

then We can create a function Ct(m,n)

Which choose n arguments from m and times them together and use "+" to ~~connect~~ conect every argumet.

for example $Ct([X_1,X_2,X_3],2) = X_1X_2+X_1X_3+X_2X_3$

Now we can find $K_2(x,y) = Ct([X_1,X_2],1) + Ct([X_1,X_2],2) + 1$

and $K_3(x,y) = Ct([X_1,X_2,X_3],1) + Ct([X_1,X_2,X_3],2) + Ct([X_1,X_2,X_3],3) + 1$

So we can simplify $K_n(x,y)$ to $\sum_{i=1}^{n} Ct([X_n],n) + 1$

so kn(x,y) = 1 + sum(from i = 1 to n) Ct([j for j in range(x1,xn)],n)