CS-7641 ASSIGNMENT1

SUPERVISED LEARNING

1 Packages and DataSet

In [1]:
```
import os
import numpy as np
import pandas as pd
import sklearn
from sklearn import preprocessing
from sklearn import metrics
from sklearn import tree
from sklearn.externals.six import StringIO
import pydotplus as pdp
import sklearn.metrics as mat
from matplotlib import pyplot
```

The packages I used for this assignment are common machine learning study tools, for the data processing the pandas package is needed, for learner modeling I import Decision Tree, MLP, KNN, ADABOOST, SVM as classifier tools from Sci-kit Learn toolkit in order to satisfy the supervised classification task requirements. As for visualization aspect, the matplotlib tool is required for diagramming, and specifically for decision tree display the Image method from IPython.Display is required.

As I am very interested in the point that the noise can lead to overfitting and the machine learning tools have much to do to avoid that,the data set I chose for this assignment are two specific data sets. The first data set is from UCI repository, it is a data set of student grade records consists of 649 records and each record contents 32 attributes. This is a 'clean' data set without much noise. On the contrary, another data set is a record of clients of a bank that is already split into training set of 10578 elements and testing set of 15929 and the attribute amount is 18. The bank data set are much more noisy than the grade dataset.For universal purpose I am trying to resplit the bank data set.

2 Data preprocessing

To do the work more accurately and smoothly, we need to preprocess the data.

In [2]:
```
dataset1 = pd.read_csv("./student-por.csv")
dataset1.dtypes
dataset1.head()
```

Out[2]:

	school	sex	age	address	famsize	Pstatus	Medu	Fedu	Mjob	Fjob	...	famrel	freetime	goout	Dalc	Walc	health	absences	G1	G2	G3
0	GP	F	18	U	GT3	A	4	4	at_home	teacher	...	4	3	4	1	1	3	4	0	11	11
1	GP	F	17	U	GT3	T	1	1	at_home	other	...	5	3	3	1	1	3	2	9	11	11
2	GP	F	15	U	LE3	T	1	1	at_home	other	...	4	3	2	2	3	3	6	12	13	12
3	GP	F	15	U	GT3	T	4	2	health	services	...	3	2	2	1	1	5	0	14	14	14
4	GP	F	16	U	GT3	T	3	3	other	other	...	4	3	2	1	2	5	0	11	13	13

5 rows × 33 columns

We read in the first dataset and we can see the data are of types of numerical and string, so we need to encode it for later calculation.

In [3]:
```
#label encode
from sklearn.preprocessing import LabelEncoder
var_to_encode = ['school','sex','address','famsize','Pstatus','Mjob','Fjob','reason','guardian','schoolsup','famsup',
       'paid','activities','nursery','higher','internet','romantic']
for col in var_to_encode:
    dataset1[col] = LabelEncoder().fit_transform(dataset1[col])
# Binarize G3<=11: G3=0    G3>11: G3=1
dataset1[['G3']] = preprocessing.Binarizer(threshold=11).transform(dataset1[['G3']])
x1=dataset1[dataset1.columns.drop('G3')]
y1= dataset1['G3']
```

Then split them into training set and data set, and do the same thing for the bank data set

```
In [4]: # divide dataset into train set and test set, size of test equals = 0.33*size of dataset
        from sklearn.model_selection import train_test_split
        x_train_1, x_test_1, y_train_1, y_test_1 = train_test_split(
            x1,y1,test_size=0.33, random_state=0)
```

```
In [5]: train = pd.read_csv("./MT_Train.csv")
        test = pd.read_csv('./MT_Test.csv')

        train['source']= 'train'
        test['source'] = 'test'
        dataset2=pd.concat([train, test],ignore_index=True)
        dataset2.dtypes
        dataset2.head()
```

Out[5]:

	SampleId	age	campaign	cons.conf.idx	cons.price.idx	contact	day_of_week	default	duration	education	...	job	loan	marital	month	n
0	NaN	56	1	-36.4	93.994	telephone	mon	no	261	basic.4y	...	housemaid	no	married	may	
1	NaN	37	1	-36.4	93.994	telephone	mon	no	226	high.school	...	services	no	married	may	
2	NaN	56	1	-36.4	93.994	telephone	mon	no	307	high.school	...	services	yes	married	may	
3	NaN	59	1	-36.4	93.994	telephone	mon	no	139	professional.course	...	admin.	no	married	may	
4	NaN	25	1	-36.4	93.994	telephone	mon	no	50	high.school	...	services	no	single	may	

To be more universal, I merged the origin train and test part of the bank dataset,and resplit them to 1:2, to avoid possible specific arrangement in the source data.

```
In [6]: dataset2.drop('default',axis=1,inplace=True)
        dataset2.drop('emp.var.rate',axis=1,inplace=True)

        from sklearn.preprocessing import LabelEncoder
        le = LabelEncoder()
        var_to_encode = ['job','marital','education','day_of_week','month','housing','loan','poutcome']
        for col in var_to_encode:
            dataset2[col] = le.fit_transform(dataset2[col])

        dataset2["contact"]=preprocessing.LabelBinarizer().fit_transform(dataset2["contact"])
        dataset2[["pdays"]] = preprocessing.Binarizer(threshold=998).transform(dataset2[["pdays"]])
        train_mod = dataset2.loc[dataset2['source']=='train']
        test_mod = dataset2.loc[dataset2['source']=='test']
        train=train_mod.copy()
        test=test_mod.copy()
        train.drop(['source','SampleId'],axis=1,inplace=True)
        test.drop(['source','y'],axis=1,inplace=True)
        train["y"]=preprocessing.LabelBinarizer().fit_transform(train["y"])

        x2=train[train.columns.drop('y')]
        y2=train['y']
        # divide dataset into train set and test set, size of test equals = 0.33*size of dataset
        from sklearn.model_selection import train_test_split
        x_train_2, x_test_2, y_train_2, y_test_2 = train_test_split(
            x2,y2,test_size=0.33, random_state=0)
```

3 Decision Tree

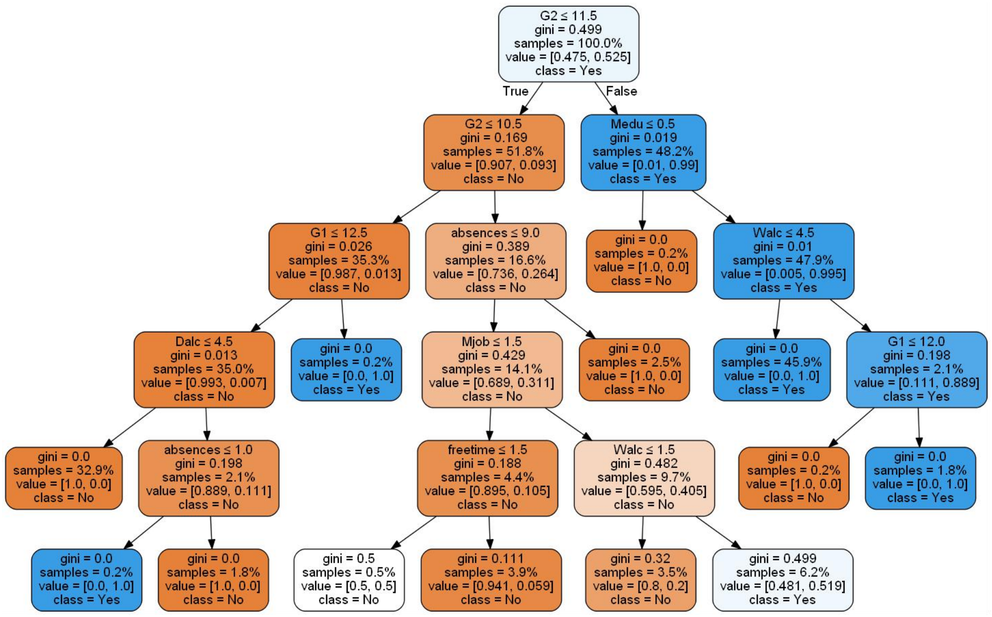
Decision Tree is a supervised learning method used for classification and regression. The goal is to create a model that predicts the value of a target variable by learning simple decision rules inferred from the data feature. In this assignment we do the decision tree classification task.

```
In [7]: from sklearn import tree
        from IPython.display import Image
        learner1=tree.DecisionTreeClassifier("gini",max_depth=4)
        learner1.fit(x_train_1, y_train_1)
        result=learner1.predict(x_test_1)
        learner2=tree.DecisionTreeClassifier("gini",max_depth=4)
        learner2.fit(x_train_2, y_train_2)
        result=learner2.predict(x_test_2)
        #print (result)
```

I trained 2 decision tree from both data, and as I mentioned before, the data of grades (student.csv) are 'clean' while data of bank client (MT.csv) are noisy.

```
In [8]: tree.export_graphviz(learner1,out_file='tree.dot',class_names=['No','Yes'],feature_names=x_train_1.columns,
                        filled=True, rounded=True, special_characters=True, proportion=True)
        # Note : Uncoverted Quotes (Yes) and Converted quotes (No)
        Image(filename='tree.png')
```

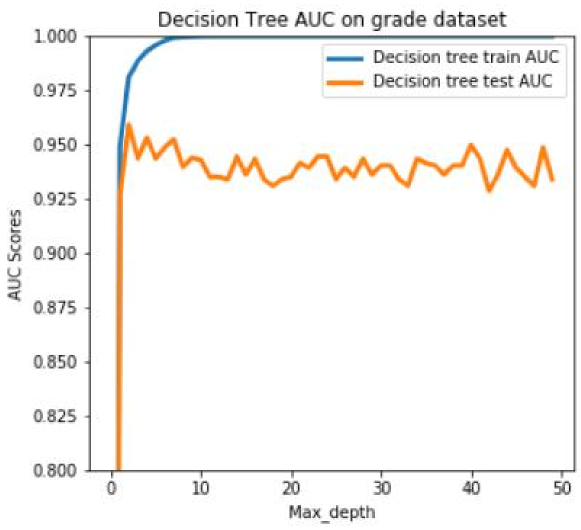
The visualization of this tree is of limited depth, and we can print the image of it:

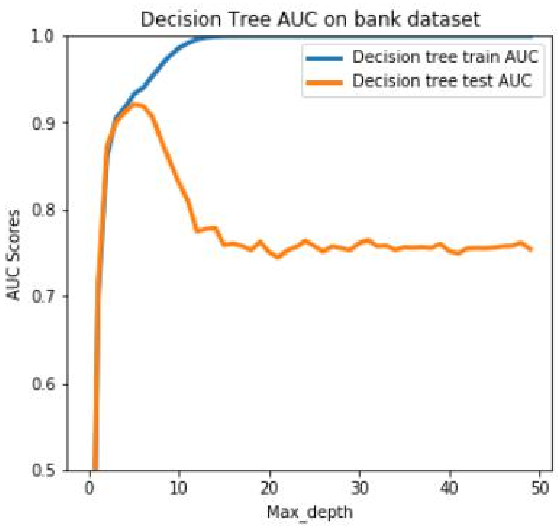


Then I made an iteration over the parameter depth. With the depth of tree growing up, the branches of the tree surpluses, and thus may lead to a potential overfitting. The decision tree also has the character of instability, and it will generate a completely different tree when data shuffles.

```
In [9]: maxdepth = 50
        tree_auc_trn1,tree_auc_tst1,tree_auc_trn2,tree_auc_tst2= np.zeros(maxdepth),\
                            np.zeros(maxdepth),np.zeros(maxdepth),np.zeros(maxdepth)
        for i in range(1,maxdepth):
            clf1 = tree.DecisionTreeClassifier(criterion="gini",max_depth=i)
            clf1 = clf1.fit(x_train_1, y_train_1)
            clf2 = tree.DecisionTreeClassifier(criterion="gini",max_depth=i)
            clf2 = clf2.fit(x_train_2, y_train_2)
            tree_auc_tst1[i] = mat.roc_auc_score(y_test_1, clf1.predict_proba(x_test_1)[:,1])
            tree_auc_trn1[i] = mat.roc_auc_score(y_train_1, clf1.predict_proba(x_train_1)[:,1])
            tree_auc_tst2[i] = mat.roc_auc_score(y_test_2, clf2.predict_proba(x_test_2)[:,1])
            tree_auc_trn2[i] = mat.roc_auc_score(y_train_2, clf2.predict_proba(x_train_2)[:,1])
```

```
pyplot.show()
```





The difference between two outcomes are distinct. We can see from each graph that the testing auc score in decision tree has the attribute of 'Rapid grow up- fall down- slow down' It can be easily understood that when the tree depth value rise, the classification are tending more 'detailed' with more branches. Thus the possibility of overfitting grows.

From dataset Grade which is clean, we can see the auc curve did not have a distinct ’down’ period, and that is because the data is clean, and no much ‘misleading teachers’ are misdirecting the learner, thus the difference of train auc value and test auc value is in reasonable range, we can say that this learner has much less risk of overfitting.On the contrary, the second diagram showed that the noisy Bank dataset are leading to a fatal overfitting with a sharp decrease on auc curve.

From the diagram we can also learn that one method to avoid possible overfitting is to restrict the max depth of the tree, and in the cases given, a max depth of 4-6 is reasonable. This is also a method of pruning.

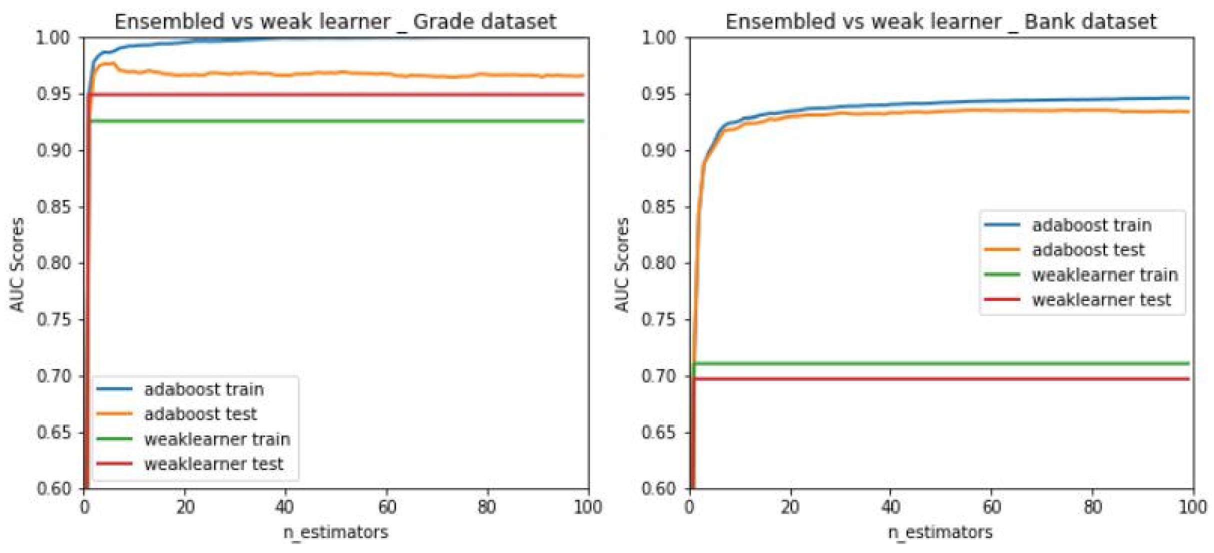
4 Boosting

In this task I use the adaBoost method from scikit learn. The function of boosting is to ensemble the ‘weak learners’, such as poor decision tree, and produce a ‘strong learners set’.

Thus firstly I construct a decision tree with a max depth of 3 and max leaf nodes of 2 as the base learner, and then write an iteration on learner amounts up to 100.

```
In [10]: from sklearn import ensemble as bst
         from sklearn import tree

         base_tr1=tree.DecisionTreeClassifier(max_depth=3,max_leaf_nodes=2)
         base_tr1.fit(x_train_1,y_train_1)
         trscn1=mat.roc_auc_score(y_test_1, base_tr1.predict_proba(x_test_1)[:,1])
         trsct1=mat.roc_auc_score(y_train_1, base_tr1.predict_proba(x_train_1)[:,1])
         base_tr2=tree.DecisionTreeClassifier(max_depth=3,max_leaf_nodes=2)
         base_tr2.fit(x_train_2,y_train_2)
         trscn2=mat.roc_auc_score(y_test_2, base_tr2.predict_proba(x_test_2)[:,1])
         trsct2=mat.roc_auc_score(y_train_2, base_tr2.predict_proba(x_train_2)[:,1])

         max_learner=100
         ada_auc_tst1,ada_auc_trn1,tree_auc_tst1,tree_auc_trn1= np.tile(np.zeros(max_learner),(4,1))
         ada_auc_tst2,ada_auc_trn2,tree_auc_tst2,tree_auc_trn2= np.tile(np.zeros(max_learner),(4,1))
         for i in range(1,max_learner):
             ada1 = bst.AdaBoostClassifier(base_estimator=base_tr1,learning_rate=1,n_estimators=i,algorithm="SAMME.R")
             ada1.fit(x_train_1, y_train_1)
             ada_auc_tst1[i] = mat.roc_auc_score(y_test_1, ada1.predict_proba(x_test_1)[:,1])
             ada_auc_trn1[i] = mat.roc_auc_score(y_train_1, ada1.predict_proba(x_train_1)[:,1])
             tree_auc_tst1[i]= trsct1
             tree_auc_trn1[i]= trscn1
             ada2 = bst.AdaBoostClassifier(base_estimator=base_tr2,learning_rate=1,n_estimators=i,algorithm="SAMME.R")
             ada2.fit(x_train_2, y_train_2)
             ada_auc_tst2[i] = mat.roc_auc_score(y_test_2, ada2.predict_proba(x_test_2)[:,1])
             ada_auc_trn2[i] = mat.roc_auc_score(y_train_2, ada2.predict_proba(x_train_2)[:,1])
             tree_auc_tst2[i]= trsct2
```

```
pyplot.ylabel("AUC Scores")
pyplot.show()
```



```
In [11]: x1_scaled=preprocessing.scale(x1)
         x2_scaled=preprocessing.scale(x2)
         x_train_1s, x_test_1s, y_train_1s, y_test_1s = train_test_split(x1_scaled,y1,test_size=0.33, random_state=0)
         x_train_2s, x_test_2s, y_train_2s, y_test_2s = train_test_split(x2_scaled,y2,test_size=0.33, random_state=0)
```

We can see from the outcome that the first base learner is kind of a ‘Strong ’ one, and the tending towards overfitting is more. For second one is a ‘weaker’ one, and it does not seem to do much overfitting. We can see that the requirement for a boosting is a really, really weak learner, and that’s why we are to use the ‘decision tree stub’ as the base learner.

5 KNN

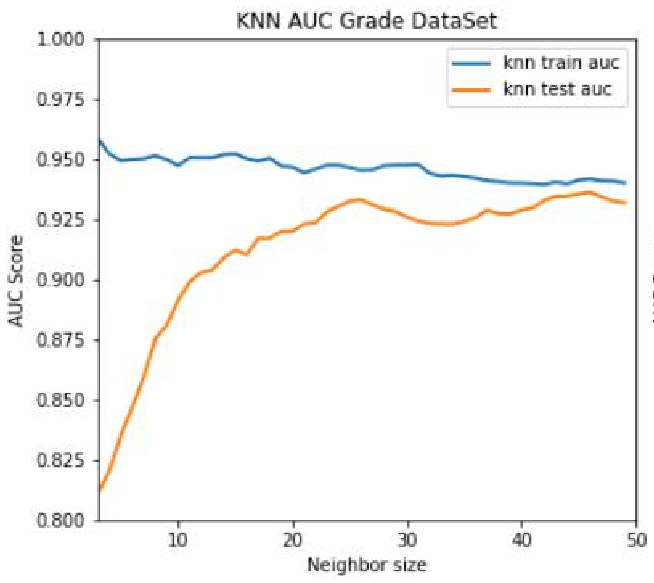
The algorithm of KNN is trying to find the nearest neighbors of points in multi-dimension space based on a method to measure the distance between distribution points. Apparently the important factor K is initial for the algorithm.So I write the iteration of the amount of k to see the auc curve performance.

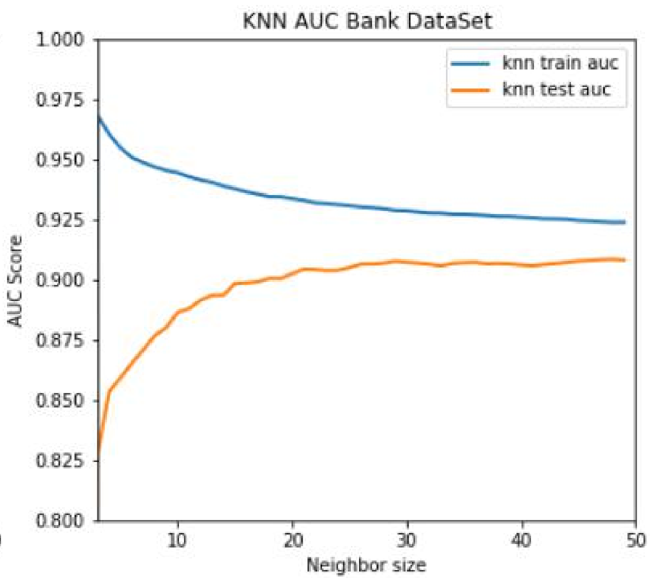
Before we get started, it is important to scale the data.How KNN will measure the variables to determine the distance should be under equal condition, and the weights are in concequence treated equal otherwise a variable with a large variance would dwarf a variable with a lower variance.

```
x1_scaled=preprocessing.scale(x1)
x2_scaled=preprocessing.scale(x1)
from sklearn.model_selection import train_test_split
x_train_1, x_test_1, y_train_1, y_test_1 = train_test_split(
    x1_scaled,y1,test_size=0.33, random_state=0)
from sklearn.model_selection import train_test_split
x_train_2, x_test_2 y_train_2, y_test_2 = train_test_split(
    x2_scaled,y2,test_size=0.33, random_state=0)
```

Then the iteration over k is to see the performance when k changes

```
In [12]: from sklearn import neighbors as nb

         kmax=50
         knn_auc_trn1,knn_auc_tst1,knn_auc_trn2,knn_auc_tst2=np.tile(np.zeros(kmax),(4,1))
         for i in range(3,kmax):
             clf1=nb.KNeighborsClassifier(n_neighbors=i,algorithm='auto', leaf_size=30, metric='minkowski',
                    metric_params=None, n_jobs=1, p=2, weights='uniform' )
             clf1 = clf1.fit(x_train_1s, y_train_1s)
             knn_auc_tst1[i] = mat.roc_auc_score(y_test_1s, clf1.predict_proba(x_test_1s)[:,1])
             knn_auc_trn1[i] = mat.roc_auc_score(y_train_1s, clf1.predict_proba(x_train_1s)[:,1])
             clf2=nb.KNeighborsClassifier(n_neighbors=i,algorithm='auto', leaf_size=30, metric='minkowski',
                    metric_params=None, n_jobs=1, p=2, weights='uniform' )
             clf2 = clf2.fit(x_train_2s, y_train_2s)
             knn_auc_tst2[i] = mat.roc_auc_score(y_test_2s, clf2.predict_proba(x_test_2s)[:,1])
             knn_auc_trn2[i] = mat.roc_auc_score(y_train_2s, clf2.predict_proba(x_train_2s)[:,1])
```





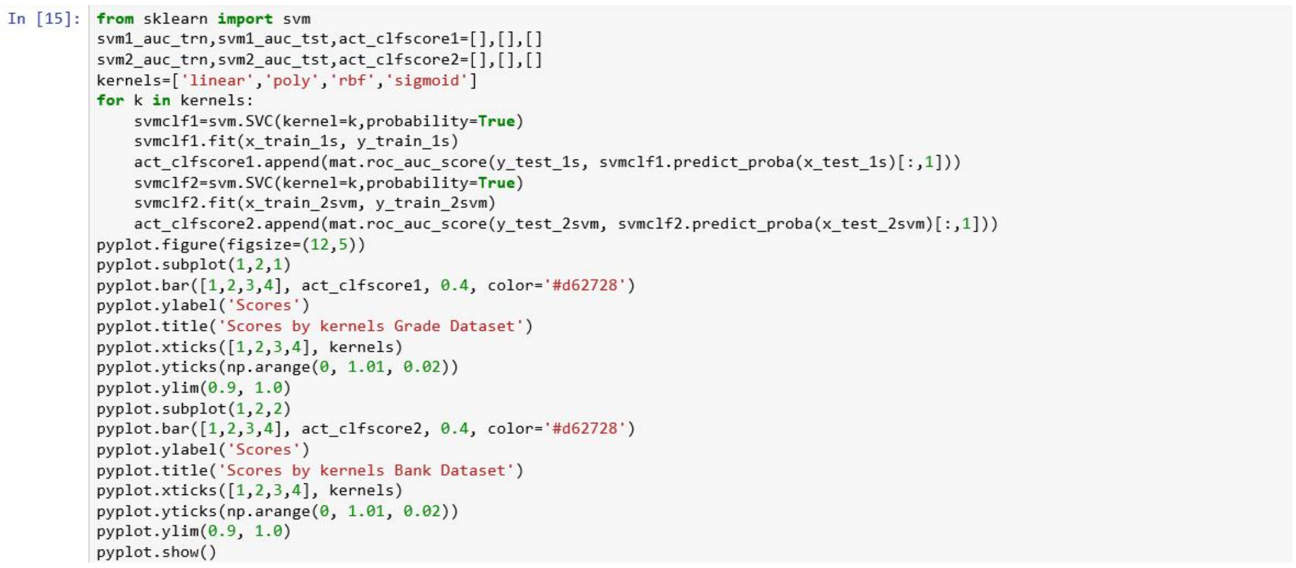
The outcome curves are similar, with the training auc and testing auc merging. The reason is quite simple, when we are at a low k value, the model is vulnerable since if several negative values are in the positive cluster, the boundary may be set, and thus the lower k is, the more complicated the model is, and apparently, it will lead to overfitting. We can see from the diagram that when the value of k increase, the difference between train auc curve and test auc curve decrease. That’s because when we have a larger k value,the model is less sophisticated and thus we face less overfitting.

Then we can easily understand that the kNN can deal with the noise when a larger k value is defined, and that is what we can see from the second diagram. However, notice that when the model has an unbalanced distribution, such as a really poor amount of one classification, the kNN might be retarded, and the err may rise. Thus cross validation are always needed for a kNN.

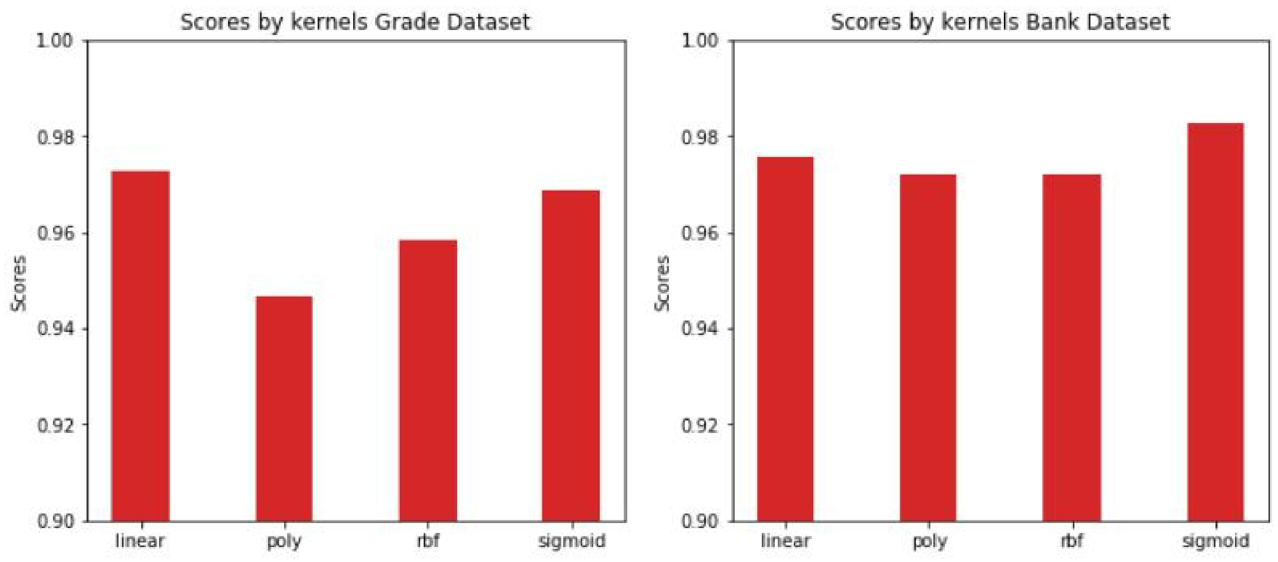
6 SVM Classifier

Unlike the kNN who is finding the neighbors,the SVM method is to define a boundary between the different classifications.The support vector machine is also effective in high dimensional spaces.

The first trial is to do with the kernel parameter, and as sklearn provided 4 types of kernels, we can compare the outcome by diagram.



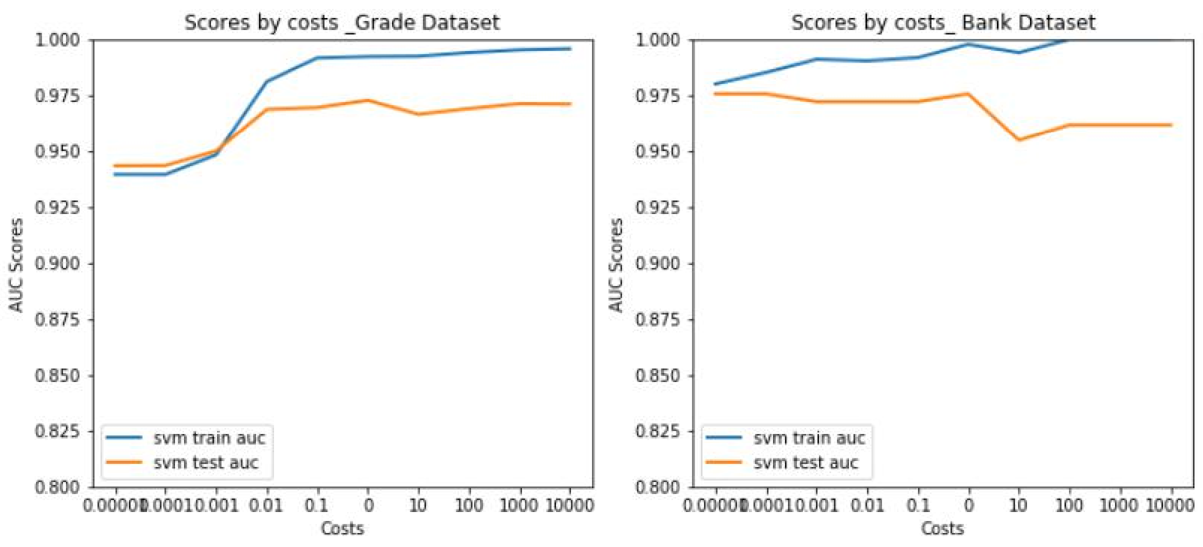
```
In [15]: from sklearn import svm
         svm1_auc_trn,svm1_auc_tst,act_clfscore1=[],[],[]
         svm2_auc_trn,svm2_auc_tst,act_clfscore2=[],[],[]
         kernels=['linear','poly','rbf','sigmoid']
         for k in kernels:
             svmclf1=svm.SVC(kernel=k,probability=True)
             svmclf1.fit(x_train_1s, y_train_1s)
             act_clfscore1.append(mat.roc_auc_score(y_test_1s, svmclf1.predict_proba(x_test_1s)[:,1]))
             svmclf2=svm.SVC(kernel=k,probability=True)
             svmclf2.fit(x_train_2svm, y_train_2svm)
             act_clfscore2.append(mat.roc_auc_score(y_test_2svm, svmclf2.predict_proba(x_test_2svm)[:,1]))
         pyplot.figure(figsize=(12,5))
         pyplot.subplot(1,2,1)
         pyplot.bar([1,2,3,4], act_clfscore1, 0.4, color='#d62728')
         pyplot.ylabel('Scores')
         pyplot.title('Scores by kernels Grade Dataset')
         pyplot.xticks([1,2,3,4], kernels)
         pyplot.yticks(np.arange(0, 1.01, 0.02))
         pyplot.ylim(0.9, 1.0)
         pyplot.subplot(1,2,2)
         pyplot.bar([1,2,3,4], act_clfscore2, 0.4, color='#d62728')
         pyplot.ylabel('Scores')
         pyplot.title('Scores by kernels Bank Dataset')
         pyplot.xticks([1,2,3,4], kernels)
         pyplot.yticks(np.arange(0, 1.01, 0.02))
         pyplot.ylim(0.9, 1.0)
         pyplot.show()
```



We can see from the diagram that the kernels selection are to conduct different outcomes. And they are based on the characteristics of the data.

The next discussion is over the cost value, which indicates the penalties of the wrong cost. The higher value of c, the harder the margin is, and the lower c indicates more tolerance.

```
In [16]: costs = np.power(10.0, range(-5,5))
         svm1_auc_tst,svm1_auc_trn,svm2_auc_tst,svm2_auc_trn  = np.tile(np.zeros(len(costs)),(4,1))
         for i in range(len(costs)):
             svmclf1 = svm.SVC(kernel = 'linear', C=costs[i], probability=True)
             svmclf1=svmclf1.fit(x_train_1s,y_train_1s)
             svm1_auc_tst[i] = mat.roc_auc_score(y_test_1s, svmclf1.predict_proba(x_test_1s)[:,1])
             svm1_auc_trn[i] = mat.roc_auc_score(y_train_1s, svmclf1.predict_proba(x_train_1s)[:,1])
             svmclf2 = svm.SVC(kernel = 'linear', C=costs[i], probability=True)
             svmclf2=svmclf2.fit(x_train_2svm,y_train_2svm)
             svm2_auc_tst[i] = mat.roc_auc_score(y_test_2svm, svmclf2.predict_proba(x_test_2svm)[:,1])
             svm2_auc_trn[i] = mat.roc_auc_score(y_train_2svm, svmclf2.predict_proba(x_train_2svm)[:,1])
         pyplot.figure(figsize=(12,5))
         pyplot.subplot(1,2,1)
         pyplot.title('Scores by costs _Grade Dataset')
         pyplot.plot(svm1_auc_trn, linewidth=2, label = "svm train auc")
         pyplot.plot(svm1_auc_tst, linewidth=2, label = "svm test auc")
         pyplot.legend()
         pyplot.xticks(range(len(costs)),['0.00001','0.0001','0.001','0.01','0.1','0','10','100','1000','10000','100000'])
         pyplot.ylim(0.8, 1.0)
         pyplot.xlabel("Costs")
         pyplot.ylabel("AUC Scores")
         pyplot.subplot(1,2,2)
         pyplot.title('Scores by costs_ Bank Dataset')
         pyplot.plot(svm2_auc_trn, linewidth=2, label = "svm train auc")
         pyplot.plot(svm2_auc_tst, linewidth=2, label = "svm test auc")
         pyplot.legend()
```



We can see from both diagram outcome that the increase value of cost can lead to the greater difference between train and test. By reviewing the margin definition, we can conclude that the error rate are to contribute this increase of gap most, while the lower c number indicates the greater buffer margin. Another disadvantage of the SVM algorithm is that it takes much more time on the giant amount of data, as the time complexity is high.

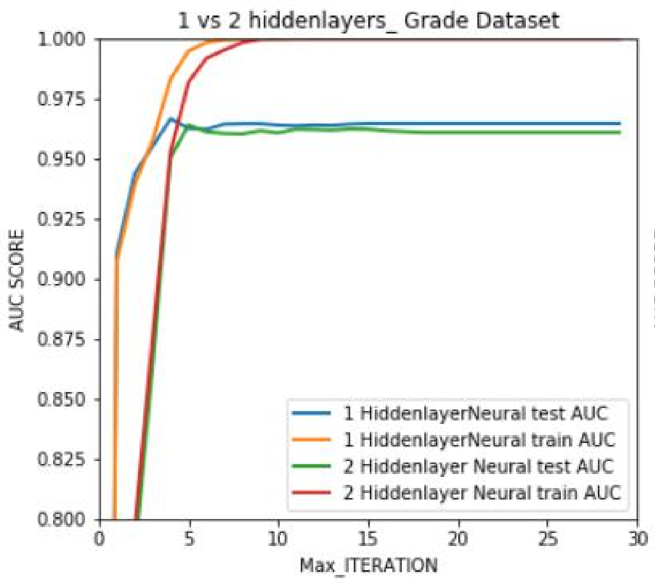
7 Neural Network

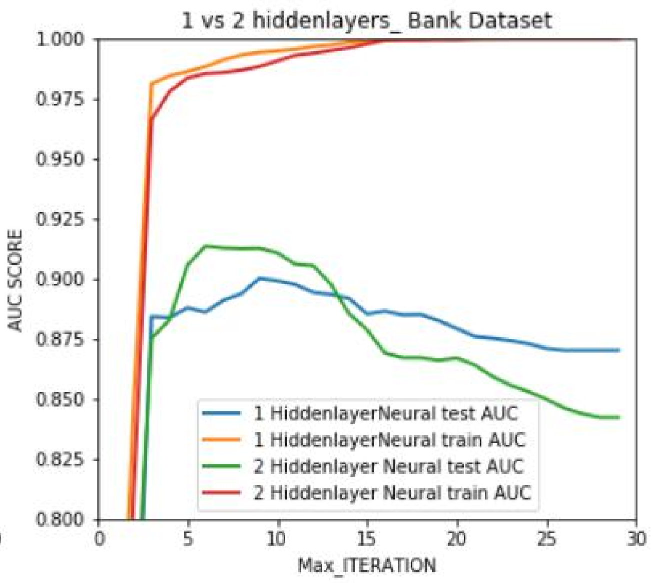
Before we start working on neural network, the data scaling is needed, thus we do preprocessing as following.

```
In [17]: from sklearn.preprocessing import StandardScaler
         scaler =StandardScaler()
         scaler.fit(x_train_1)
         x_train_1n = scaler.transform(x_train_1)
         x_test_1n = scaler.transform(x_test_1)
         y_train_1n=y_train_1
         y_test_1n=y_test_1
         x2_slice=x2[2000:3000]
         y2_slice=y2[2000:3000]
         from sklearn.model_selection import train_test_split
         x_train_2n, x_test_2n, y_train_2n, y_test_2n = train_test_split(
             x2_slice,y2_slice,test_size=0.33, random_state=0)
         from sklearn.preprocessing import StandardScaler
         scaler = StandardScaler()
         scaler.fit(x_train_2n)
         x_train_2n = scaler.transform(x_train_2n)
         x_test_2n = scaler.transform(x_test_2n)
```

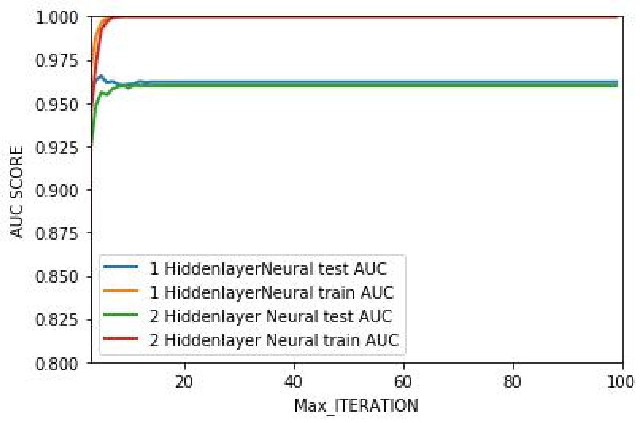
In this case I compare two situation of hidden layers- one layer and two layers, and to see the outcome when we increase the iteration amount of back propagation.

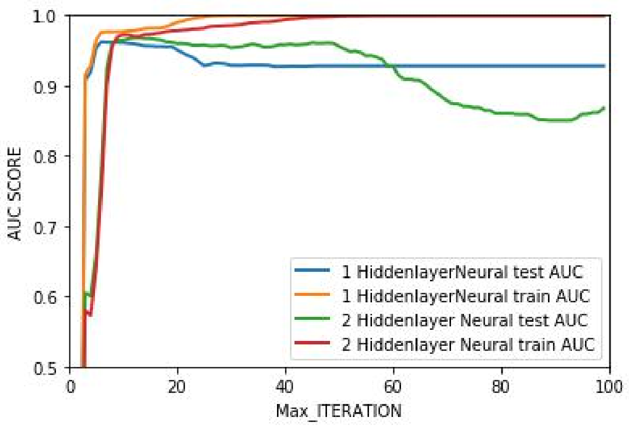
```
In [18]: from sklearn import neural_network as nw

         itermax=30
         nw_auc_trn1,nw_auc_tst1,nw_auc_trn2,nw_auc_tst2=np.tile(np.zeros(itermax),(4,1))
         nw_auc_trn3,nw_auc_tst3,nw_auc_trn4,nw_auc_tst4=np.tile(np.zeros(itermax),(4,1))
         for i in range(1,itermax):
             clf1= nw.MLPClassifier(solver='lbfgs',max_iter=i,alpha=1e-5,hidden_layer_sizes=(50), random_state=1)
             clf1 = clf1.fit(x_train_1n, y_train_1n)
             nw_auc_tst1[i] = mat.roc_auc_score(y_test_1n, clf1.predict_proba(x_test_1n)[:,1])
             nw_auc_trn1[i] = mat.roc_auc_score(y_train_1n, clf1.predict_proba(x_train_1n)[:,1])
             clf2= nw.MLPClassifier(solver='lbfgs',max_iter=i,alpha=1e-5,hidden_layer_sizes=(50, 50), random_state=1)
             clf2 = clf2.fit(x_train_1n, y_train_1n)
             nw_auc_tst2[i] = mat.roc_auc_score(y_test_1n, clf2.predict_proba(x_test_1n)[:,1])
             nw_auc_trn2[i] = mat.roc_auc_score(y_train_1n, clf2.predict_proba(x_train_1n)[:,1])
             clf3= nw.MLPClassifier(solver='lbfgs',max_iter=i,alpha=1e-5,hidden_layer_sizes=(50), random_state=1)
             clf3 = clf3.fit(x_train_2n, y_train_2n)
             nw_auc_tst3[i] = mat.roc_auc_score(y_test_2n, clf3.predict_proba(x_test_2n)[:,1])
             nw_auc_trn3[i] = mat.roc_auc_score(y_train_2n, clf3.predict_proba(x_train_2n)[:,1])
             clf4= nw.MLPClassifier(solver='lbfgs',max_iter=i,alpha=1e-5,hidden_layer_sizes=(50, 50), random_state=1)
             clf4 = clf4.fit(x_train_2n, y_train_2n)
             nw_auc_tst4[i] = mat.roc_auc_score(y_test_2n, clf4.predict_proba(x_test_2n)[:,1])
             nw_auc_trn4[i] = mat.roc_auc_score(y_train_2n, clf4.predict_proba(x_train_2n)[:,1])
```





We can see from this two diagram that the curve grows rapidly when we introduce the BP loop, and the two-layer auc has better performance at first, but gradually it has more tendency of splitting- that's the symbol of overfitting. It can be understood that the more layer the model has, the more accurate study model it is, and it will be stronger. And thus lead to the risk of overfitting. So to control this problem, we can control the layer amount of a neural network model.





It seems that from the last diagrams, the model's performance are not so good, but if we zoom to a larger dimension of iteration number, we can see that the auc curve are growing to higher accuracy amount, and are tending to remain stable when iteration times increases. It has the strong ability of learning and stability than the decision tree.

But we can also see from the second diagram above, that it does not perform really well with second layer introduced. Two possibilities may contribute to that, since it is from a slice of data, maybe the data quality is not sufficient, as the learning process get into greater accuracy, it in fact misleads. The second possibility is that as the second layer introduced, it is more sensitive to the iteration while the samples are not sufficient.

Thus in a nutshell, we can make these efforts to increase the ability of neural network learner:

1) Assure the quality of the dataset, to be more detailed, the training set should be large enough, with less noise and are typical.

2) When the learning ability is not sufficient, we can Increase the hidden layer to increase the ability of learning, while on the other hand we should control the iteration maximum to avoid overfitting.

3) To arrange other parameters, and find the best optimization, like we can do through grid search to confirm the best parameter sets.