Classification of Audit Firms - Python

Classification Summary

In the classification section, we have implemented the following models: k-Nearest Neighbours Classification, Logistic Regression, Linear Support Vector Machine, Kernelized Support Vector Machine and Decision Tree.

To start with this approach, we first merged the two datasets ('audit_risk' and 'trial'). We found out that there are a few columns in the 'trial' dataset which are the same as in the 'audit_risk' dataset and hence we removed them. We also found that a few columns in the 'trail' dataset are 10 times that of a few columns in the 'audit_risk' dataset and hence we dropped such columns. The column 'Detection_Risk' has only one value thorughout the dataset hence we have not considered this column in our analysis. More importantly, the Risk columns in both the datasets are different, hence we decided to consider the target column for our classification models as the Risk column found in 'audit_risk' dataset. The final shape of the dataset was 760 rows and 26 columns.

We have splitted the data in the ratio of 70:30 with a random state of 0. We have scaled the data using Min-Max Scaling method. In each of the classification models, we have used GridSearchCV to help us determine the best parameter for the particular model. The parameter values are passed in the variable called p_grid (in each model) and we have also shown which parameter gave us the best results by using the attribute best*params* For every model we have shown a plot of Train score and Test score.

In the end we have combined the test score and train score results and displayed it in form of a table for ease of comparison

In [1]:
```
import numpy as np
import scipy as spy
import pandas as pd
from sklearn import *
import matplotlib.pyplot as plt
import seaborn as sns
import warnings
warnings.filterwarnings('ignore')
```

Classification Data Prep

In [2]:
```
audit_risk = pd.read_csv("audit_risk.csv")
trial_df = pd.read_csv("trial.csv")
#Same columns as in audit_risk
trial_df = trial_df.drop(["Sector_score", "LOCATION_ID", "PARA_A", "PARA_B",
"TOTAL", "numbers", "Money_Value",
                    "Score", "SCORE_A", "History", "SCORE_B"], axis = 1)

#Same columns (multiples of 10)
trial_df = trial_df.drop(["District", "Marks", "MONEY_Marks" ,
                    "LOSS_SCORE",  "History_score"], axis = 1)
trial_df.columns = ['Loss', 'Risk_trial']
trial_df.columns
merged = pd.concat([audit_risk, trial_df], axis = 1)

final_df = merged.drop(['Audit_Risk', 'Risk_trial'], axis = 1)
#Filling NA with mean
final_df['Money_Value'] = final_df['Money_Value'].fillna(final_df['Money_Valu
e'].mean())

#Same value in throughout the column
final_df = final_df.drop(['Detection_Risk'], axis = 1)
#LOCATION_ID is of object datatype instead of integer
final_df["LOCATION_ID"].unique()
print(len(final_df[(final_df["LOCATION_ID"] == 'LOHARU') | (final_df["LOCATION
_ID"] ==  'NUH') | (final_df["LOCATION_ID"] == 'SAFIDON')]))
final_df = final_df[(final_df.LOCATION_ID != 'LOHARU')]
final_df = final_df[(final_df.LOCATION_ID != 'NUH')]
final_df = final_df[(final_df.LOCATION_ID != 'SAFIDON')]
final_df = final_df.astype(float)
len(final_df)
final_df = final_df.drop_duplicates(keep = 'first')
print(len(final_df))
final_df.shape
```

```
3
760
```

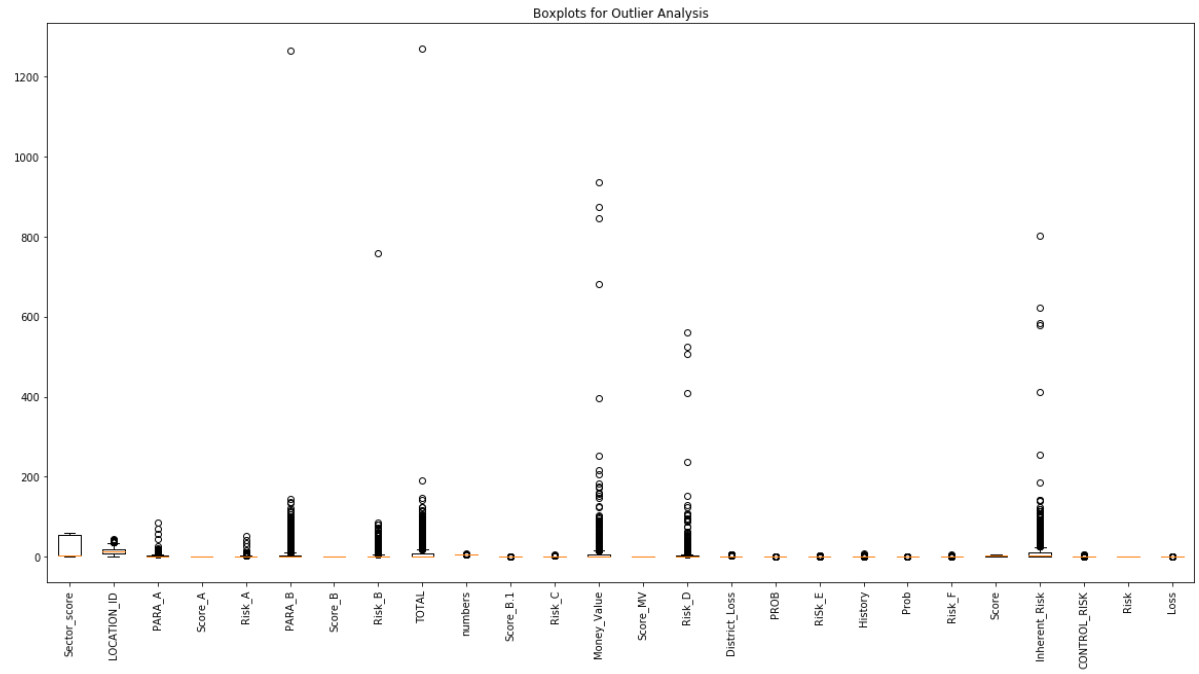
Out[2]: (760, 26)

Outlier Analysis

In [3]:
```
plt.rcParams['figure.figsize'] = (20.0, 10.0)
plt.boxplot(final_df.values)

plt.xticks(range(1,len(final_df.columns)+1), final_df.columns, rotation = 'ver
tical')
plt.title('Boxplots for Outlier Analysis')
```

Out[3]:
```
Text(0.5, 1.0, 'Boxplots for Outlier Analysis')
```



In [4]:
```
def outlier_removal(df, percentile):
    limit = df.quantile(percentile)
    df[df > limit] = df.median()

outlier_removal(final_df["PARA_A"], 0.75)
outlier_removal(final_df["Risk_A"], 0.75)
outlier_removal(final_df["PARA_B"], 0.75)
outlier_removal(final_df["Risk_B"], 0.75)
outlier_removal(final_df["TOTAL"], 0.75)
outlier_removal(final_df["Money_Value"], 0.75)
outlier_removal(final_df["Risk_D"], 0.75)
outlier_removal(final_df["Inherent_Risk"], 0.75)
```

In [5]:
```
import seaborn as sns
corr = final_df.corr()
corr.style.background_gradient(cmap='coolwarm')

cmap = sns.diverging_palette(220, 20, sep=20, as_cmap=True)
corr.style.background_gradient(cmap=cmap).set_precision(2)
```

Out[5]:

	Sector_score	LOCATION_ID	PARA_A	Score_A	Risk_A	PARA_B	Score_B	Ri
Sector_score	1	-0.052	-0.26	-0.42	-0.17	-0.068	-0.21	
LOCATION_ID	-0.052	1	0.02	0.079	0.0015	0.0065	0.13	
PARA_A	-0.26	0.02	1	0.64	0.91	0.11	0.33	(
Score_A	-0.42	0.079	0.64	1	0.46	0.16	0.57	
Risk_A	-0.17	0.0015	0.91	0.46	1	0.081	0.2	(
PARA_B	-0.068	0.0065	0.11	0.16	0.081	1	0.33	
Score_B	-0.21	0.13	0.33	0.57	0.2	0.33	1	
Risk_B	-0.04	0.03	0.086	0.14	0.076	0.68	0.27	
TOTAL	-0.18	-0.00077	0.46	0.46	0.39	0.47	0.3	
numbers	-0.15	0.0067	0.11	0.24	0.034	0.079	0.28	
Score_B.1	-0.17	-0.018	0.12	0.27	0.035	0.094	0.31	(
Risk_C	-0.16	-0.015	0.11	0.26	0.035	0.094	0.3	(
Money_Value	-0.11	-0.0029	0.14	0.16	0.16	0.12	0.21	(
Score_MV	-0.32	0.11	0.24	0.47	0.14	0.067	0.56	(
Risk_D	-0.094	0.018	0.15	0.16	0.18	0.1	0.2	(
District_Loss	-0.11	-0.11	0.054	0.086	0.03	-0.0015	-0.0077	(
PROB	-0.086	-0.0034	0.05	0.091	0.013	0.035	0.091	(
RiSk_E	-0.13	-0.097	0.059	0.1	0.027	-0.0034	0.012	(
History	-0.11	-0.082	0.091	0.18	0.067	0.11	0.2	(
Prob	-0.14	-0.054	0.12	0.26	0.066	0.058	0.31	(
Risk_F	-0.1	-0.089	0.077	0.15	0.06	0.11	0.17	
Score	-0.33	0.088	0.41	0.72	0.26	0.26	0.9	
Inherent_Risk	-0.18	0.044	0.32	0.4	0.27	0.32	0.23	
CONTROL_RISK	-0.16	-0.12	0.092	0.17	0.059	0.074	0.12	(
Risk	-0.39	0.063	0.33	0.62	0.21	0.16	0.63	
Loss	-0.082	0.0064	0.039	0.091	0.0034	0.04	0.097	(

Classification

In [6]:
```
classification_X = final_df.drop(["Risk"], axis = 1)
classification_y = final_df["Risk"]
```

In [7]:
```
from sklearn.model_selection import train_test_split

X_train_org, X_test_org, y_train, y_test = train_test_split(classification_X,
classification_y, test_size = 0.3, random_state = 0)
```

Splitting the data in 70:30 ratio

In [8]:
```
from sklearn.preprocessing import MinMaxScaler
scaler = MinMaxScaler()

X_train = scaler.fit_transform(X_train_org)
X_test  = scaler.transform(X_test_org)
```

Scale the data using Min - Max scaling method

KNN Classification

In [9]:
```
from sklearn.model_selection import GridSearchCV
from sklearn.neighbors import KNeighborsClassifier

knn = KNeighborsClassifier()
p_grid = {'n_neighbors':[3, 5, 7, 9, 11, 13, 15]}

grid_knn = GridSearchCV(knn, p_grid, cv = 10, scoring='roc_auc')
grid_knn.fit(X_train_org, y_train)
grid_knn.score(X_train_org, y_train)
```

Out[9]:
```
0.9850604363207547
```

In [10]:
```
grid_knn.score(X_test_org, y_test)
```

Out[10]:
```
0.9606377877237853
```

The following parameter gave the best result for the data:

In [11]:
```
grid_knn.best_params_
```

Out[11]:
```
{'n_neighbors': 5}
```

In [12]:
```
grid_knn.cv_results_['mean_test_score']
```

Out[12]:
```
array([0.94189957, 0.94706073, 0.94624739, 0.94597937, 0.94281844,
       0.94081211, 0.93477657])
```

In [13]:
```
y_knn_predict = grid_knn.predict(X_test_org)
y_knn_train_predict = grid_knn.predict(X_train_org)
```

In [14]:
```
from sklearn.metrics import roc_auc_score
print('Train roc_auc_score: %.2f'%roc_auc_score(y_knn_train_predict, y_train))
print('Test roc_auc_score: %.2f '%roc_auc_score(y_knn_predict, y_test))
```

```
Train roc_auc_score: 0.93
Test roc_auc_score: 0.91
```

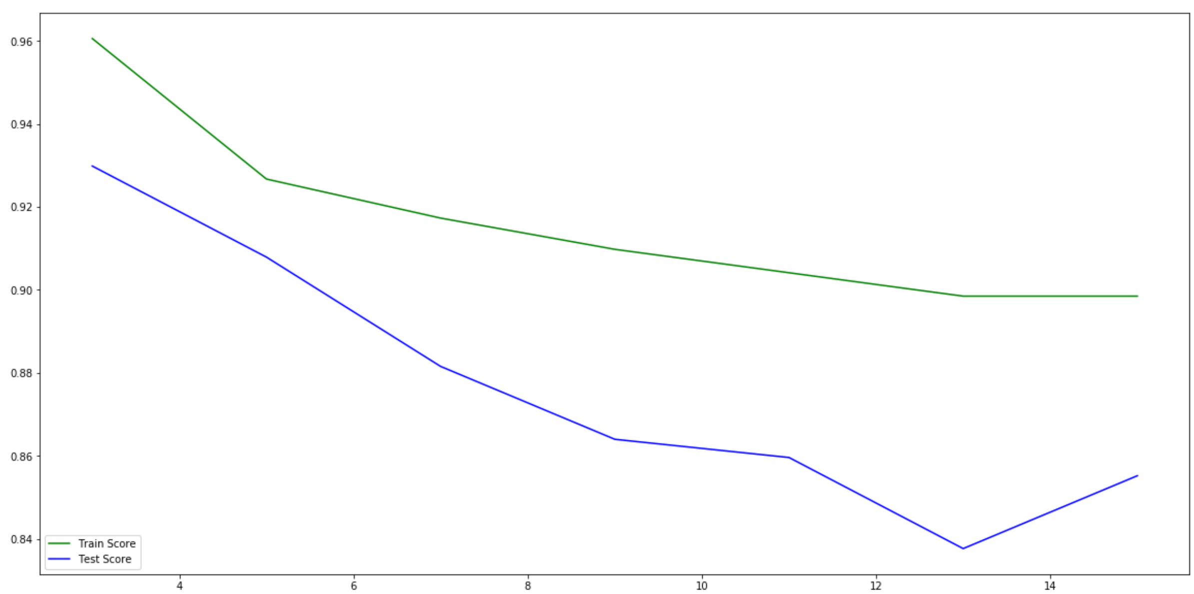
In [15]:
```
train_score_list = []
test_score_list = []
x_range = [3, 5, 7, 9, 11, 13, 15]

for alpha in x_range:
    model = KNeighborsClassifier(n_neighbors=alpha)
    model.fit(X_train_org,y_train)
    train_score_list.append(model.score(X_train_org,y_train))
    test_score_list.append(model.score(X_test_org, y_test))
```

Plotting the Train and Test Scores

In [16]:
```
plt.plot(x_range, train_score_list, c = 'g', label = 'Train Score')
plt.plot(x_range, test_score_list, c = 'b', label = 'Test Score')
plt.xscale('linear')
plt.legend(loc = 3)
plt.xlabel(r'$\alpha$')
```

Out[16]:
```
Text(0.5, 0, '$\\alpha$')
```



Add the final results to the table dataframe

In [17]:
```
table = [['knn', 'k = 5', grid_knn.score(X_train_org, y_train),
                 grid_knn.score(X_test_org, y_test), roc_auc_score(y_knn_train
_predict, y_train),
                 roc_auc_score(y_knn_predict, y_test) ]]
```

Linear SVM

In [18]:
```
from sklearn.svm import LinearSVC

linear_svc = LinearSVC()
p_grid = {'C':[0.001, 0.01, 0.1, 1, 10, 100]}

grid_linear_svc = GridSearchCV(linear_svc, p_grid, cv = 5, scoring='roc_auc',
return_train_score=True)
```

In [19]:
```
grid_linear_svc.fit(X_train, y_train)
```

Out[19]:
```
GridSearchCV(cv=5, error_score='raise-deprecating',
       estimator=LinearSVC(C=1.0, class_weight=None, dual=True, fit_intercept
=True,
     intercept_scaling=1, loss='squared_hinge', max_iter=1000,
     multi_class='ovr', penalty='l2', random_state=None, tol=0.0001,
     verbose=0),
       fit_params=None, iid='warn', n_jobs=None,
       param_grid={'C': [0.001, 0.01, 0.1, 1, 10, 100]},
       pre_dispatch='2*n_jobs', refit=True, return_train_score=True,
       scoring='roc_auc', verbose=0)
```

The following parameter gave the best result for the data:

In [20]:
```
grid_linear_svc.best_params_
```

Out[20]:
```
{'C': 1}
```

In [21]:
```
grid_linear_svc.cv_results_['mean_test_score']
```

Out[21]:
```
array([0.97604073, 0.98993044, 0.99681366, 0.9971112 , 0.99607762,
       0.99571116])
```

In [22]:
```
y_linear_svc_predict_train = grid_linear_svc.predict(X_train)
y_linear_svc_predict = grid_linear_svc.predict(X_test)
```

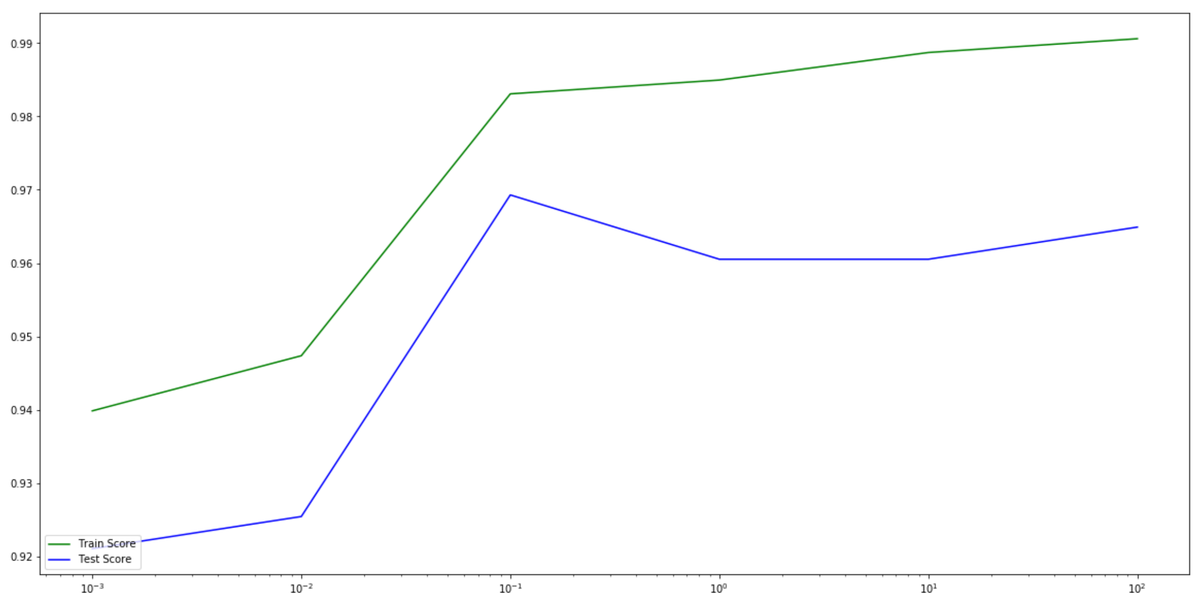
In [23]:
```
train_score_list = []
test_score_list = []
x_range = [0.001, 0.01, 0.1, 1, 10, 100]

for alpha in x_range:
    model = LinearSVC(C=alpha)
    model.fit(X_train,y_train)
    train_score_list.append(model.score(X_train,y_train))
    test_score_list.append(model.score(X_test, y_test))
```

Plotting the Train and Test Scores

In [24]:
```
plt.plot(x_range, train_score_list, c = 'g', label = 'Train Score')
plt.plot(x_range, test_score_list, c = 'b', label = 'Test Score')
plt.xscale('log')
plt.legend(loc = 3)
plt.xlabel(r'$\alpha$')
```

Out[24]:
```
Text(0.5, 0, '$\\alpha$')
```



In [25]:
```
table = table + [['LinearSVC', 'C = 1', grid_linear_svc.score(X_train, y_train
),
                                grid_linear_svc.score(X_test, y_test), roc_auc
_score(y_linear_svc_predict_train, y_train),
                                roc_auc_score(y_linear_svc_predict, y_test)]]
```

Add the results to table dataframe

Logistic Regression

In [26]:
```
from sklearn.linear_model import LogisticRegression
param_grid = {'C': [0.001, 0.01, 0.1, 1, 10, 100] }
logit = GridSearchCV(LogisticRegression(penalty='l2'), p_grid)
logit.fit(X_train, y_train)
```

Out[26]:
```
GridSearchCV(cv='warn', error_score='raise-deprecating',
       estimator=LogisticRegression(C=1.0, class_weight=None, dual=False, fit
_intercept=True,
          intercept_scaling=1, max_iter=100, multi_class='warn',
          n_jobs=None, penalty='l2', random_state=None, solver='warn',
          tol=0.0001, verbose=0, warm_start=False),
       fit_params=None, iid='warn', n_jobs=None,
       param_grid={'C': [0.001, 0.01, 0.1, 1, 10, 100]},
       pre_dispatch='2*n_jobs', refit=True, return_train_score='warn',
       scoring=None, verbose=0)
```

The following parameter gave the best result for the data:

In [27]:
```
logit.best_params_
```

Out[27]:
```
{'C': 10}
```

In [28]:
```
logit.cv_results_['mean_test_score']
```

Out[28]:
```
array([0.90977444, 0.92857143, 0.94172932, 0.96240602, 0.97180451,
       0.97180451])
```

In [29]:
```
y_log_predict_train = logit.predict(X_train)
y_log_predict = logit.predict(X_test)
```

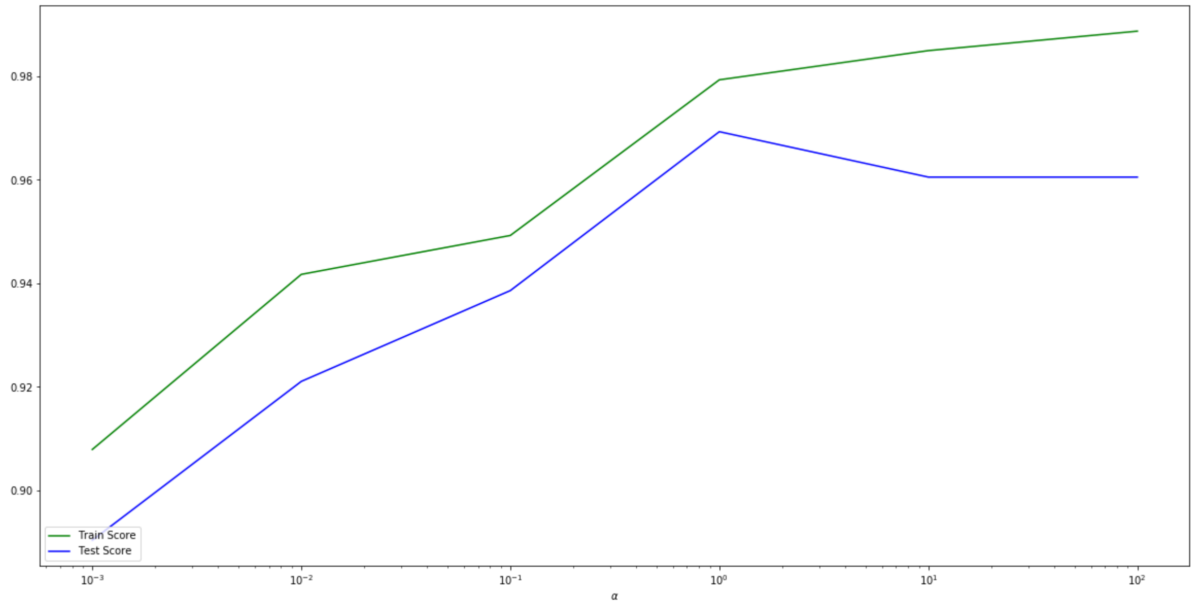
In [30]:
```
train_score_list = []
test_score_list = []
x_range = [0.001, 0.01, 0.1, 1, 10, 100]

for alpha in x_range:
    model = LogisticRegression(C=alpha)
    model.fit(X_train,y_train)
    train_score_list.append(model.score(X_train,y_train))
    test_score_list.append(model.score(X_test, y_test))
```

Plotting the Train and Test Scores

In [31]:
```
plt.plot(x_range, train_score_list, c = 'g', label = 'Train Score')
plt.plot(x_range, test_score_list, c = 'b', label = 'Test Score')
plt.xscale('log')
plt.legend(loc = 3)
plt.xlabel(r'$\alpha$')
```

Out[31]: Text(0.5, 0, '$\\alpha$')



Add the result to the table dataframe

In [32]:
```
table = table + [['Logistic Regression', 'C = 10', logit.score(X_train, y_trai
n),
                                logit.score(X_test, y_test), roc_auc_score(y_l
og_predict_train, y_train),
                                roc_auc_score(y_log_predict, y_test)]]
```

Kernalized SVM

In [33]:
```
from sklearn.svm import SVC
kernel_svc = SVC()
p_grid = {'C':[0.001, 0.01, 0.1, 1, 10, 100]}

grid_kernel_svc = GridSearchCV(kernel_svc, p_grid, cv = 5, scoring='roc_auc',
return_train_score=True)
```

In [34]:
```
grid_kernel_svc.fit(X_train, y_train)
```

Out[34]:
```
GridSearchCV(cv=5, error_score='raise-deprecating',
       estimator=SVC(C=1.0, cache_size=200, class_weight=None, coef0=0.0,
  decision_function_shape='ovr', degree=3, gamma='auto_deprecated',
  kernel='rbf', max_iter=-1, probability=False, random_state=None,
  shrinking=True, tol=0.001, verbose=False),
       fit_params=None, iid='warn', n_jobs=None,
       param_grid={'C': [0.001, 0.01, 0.1, 1, 10, 100]},
       pre_dispatch='2*n_jobs', refit=True, return_train_score=True,
       scoring='roc_auc', verbose=0)
```

The following parameter gave the best result for the data:

In [35]:
```
grid_kernel_svc.best_params_
```

Out[35]:
```
{'C': 100}
```

In [36]:
```
grid_kernel_svc.cv_results_['mean_test_score']
```

Out[36]:
```
array([0.97636429, 0.97629121, 0.97820909, 0.99518708, 0.99711016,
       0.9978535 ])
```

In [37]:
```
y_kernel_svc_predict_train = grid_kernel_svc.predict(X_train)
y_kernel_svc_predict = grid_kernel_svc.predict(X_test)
```

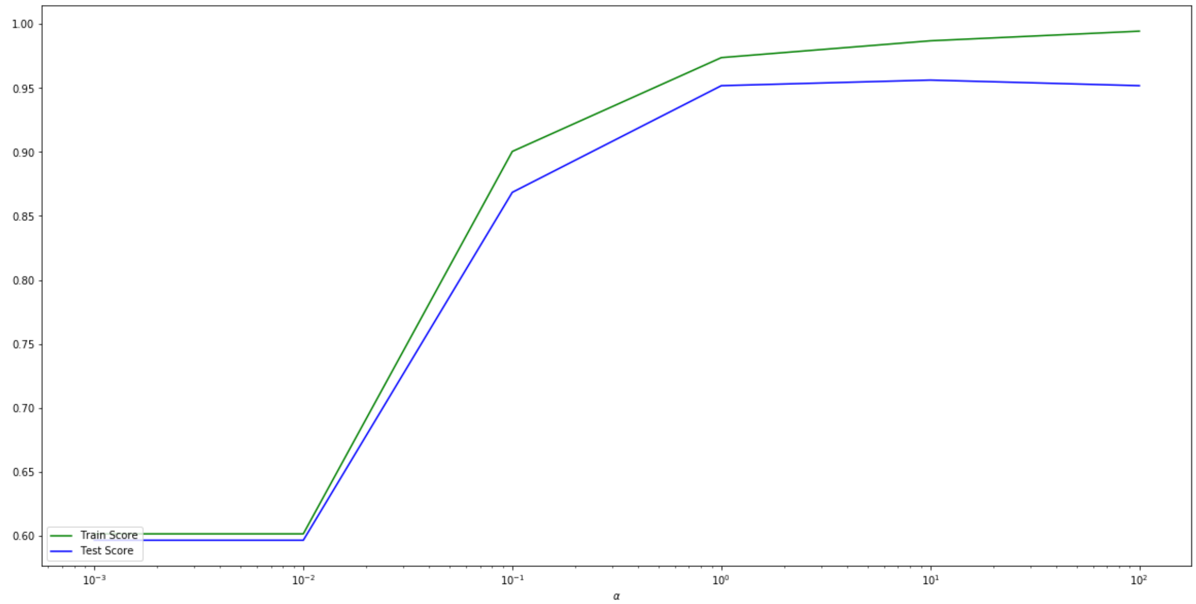
In [38]:
```
train_score_list = []
test_score_list = []
x_range = [0.001, 0.01, 0.1, 1, 10, 100]

for alpha in x_range:
    model = SVC(C=alpha)
    model.fit(X_train,y_train)
    train_score_list.append(model.score(X_train,y_train))
    test_score_list.append(model.score(X_test, y_test))
```

Plotting the Train and Test Scores

In [39]:
```
plt.plot(x_range, train_score_list, c = 'g', label = 'Train Score')
plt.plot(x_range, test_score_list, c = 'b', label = 'Test Score')
plt.xscale('log')
plt.legend(loc = 3)
plt.xlabel(r'$\alpha$')
```

Out[39]: Text(0.5, 0, '$\\alpha$')



Add the result to the table dataframe

In [40]:
```
table = table + [['Kernalized SVM', 'C = 100', grid_kernel_svc.score(X_train, y_train),
                                grid_kernel_svc.score(X_test, y_test), roc_auc_score(y_kernel_svc_predict_train, y_train),
                                roc_auc_score(y_kernel_svc_predict, y_test)]]
```

Decision Tree

In [41]:
```
from sklearn.tree import DecisionTreeClassifier

p_grid = {'max_depth': np.arange(4, 15)}

tree = GridSearchCV(DecisionTreeClassifier(), p_grid,cv = 5, scoring='roc_auc', return_train_score=True)
```

In [42]:
```
tree.fit(X_train_org, y_train)
```

Out[42]:
```
GridSearchCV(cv=5, error_score='raise-deprecating',
       estimator=DecisionTreeClassifier(class_weight=None, criterion='gini',
max_depth=None,
            max_features=None, max_leaf_nodes=None,
            min_impurity_decrease=0.0, min_impurity_split=None,
            min_samples_leaf=1, min_samples_split=2,
            min_weight_fraction_leaf=0.0, presort=False, random_state=None,
            splitter='best'),
       fit_params=None, iid='warn', n_jobs=None,
       param_grid={'max_depth': array([ 4,  5,  6,  7,  8,  9, 10, 11, 12, 1
3, 14])},
       pre_dispatch='2*n_jobs', refit=True, return_train_score=True,
       scoring='roc_auc', verbose=0)
```

The following parameter gave the best result for the data:

In [43]:
```
tree.best_params_
```

Out[43]:
```
{'max_depth': 4}
```

In [44]:
```
tree.cv_results_['mean_test_score']
```

Out[44]:
```
array([0.99248076, 0.99124625, 0.98505681, 0.98898544, 0.98898544,
       0.98186943, 0.98424144, 0.99135744, 0.98661344, 0.98505681,
       0.98898544])
```

In [45]:
```
y_tree_predict_train = tree.predict(X_train_org)
y_tree_predict = tree.predict(X_test_org)
```

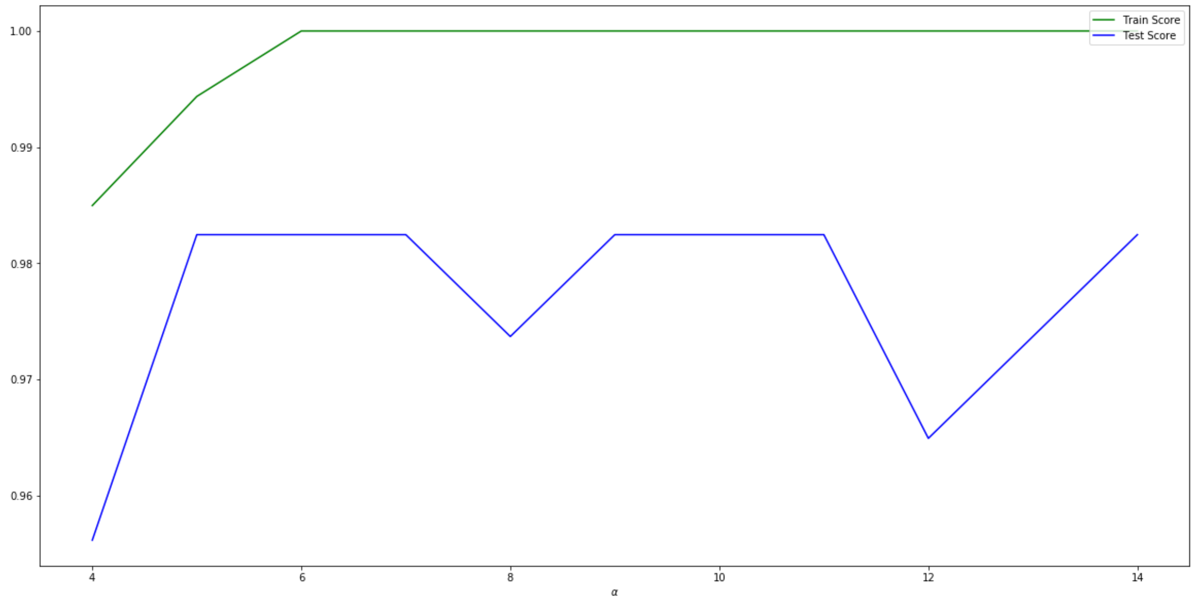
In [46]:
```
train_score_list = []
test_score_list = []
x_range = np.arange(4, 15)

for alpha in x_range:
    model = DecisionTreeClassifier(max_depth=alpha)
    model.fit(X_train_org,y_train)
    train_score_list.append(model.score(X_train_org,y_train))
    test_score_list.append(model.score(X_test_org, y_test))
```

Plot for the Train and Test Score

In [47]:
```
plt.plot(x_range, train_score_list, c = 'g', label = 'Train Score')
plt.plot(x_range, test_score_list, c = 'b', label = 'Test Score')
plt.xscale('linear')
plt.legend(loc = 1)
plt.xlabel(r'$\alpha$')
```

Out[47]: Text(0.5, 0, '$\\alpha$')



Adding the column names to the dataframe

In [48]:
```
table = table + [['Decision Tree', 'd = 4', tree.score(X_train_org, y_train),
                                tree.score(X_test_org, y_test), roc_auc_score(
y_tree_predict_train, y_train),
                                roc_auc_score(y_tree_predict, y_test)]]
```

In [49]:
```
table = pd.DataFrame(table,columns = ['Model name', 'Model parameter', 'Train
 accuracy',
                                              'Test accuracy', 'Train roc_auc
 score', 'Test roc_auc score'])
```

In [50]:
```
table.index = table['Model name']
```

The Train and Test accuracy for all the models:

In [51]:
```
table
```

Out[51]:

Model name	Model name	Model parameter	Train accuracy	Test accuracy	Train roc_auc score	Test roc_auc score
knn	knn	k = 5	0.985060	0.960638	0.930252	0.914714
LinearSVC	LinearSVC	C = 1	0.999175	0.996084	0.984316	0.966146
Logistic Regression	Logistic Regression	C = 10	0.984962	0.960526	0.984316	0.966146
Kernalized SVM	Kernalized SVM	C = 100	0.999926	0.997363	0.994515	0.959493
Decision Tree	Decision Tree	d = 4	0.997420	0.974744	0.985093	0.956039