Problem Set 6

By Chu Zhuang

```
# import relevant packages
import random
import math
import numpy as np
import pandas as pd
import seaborn
import sklearn
from sklearn.svm import SVC
from sklearn.model_selection import train_test_split
import matplotlib as mpl
import matplotlib.pyplot as plt
```

Non Linear Separation

First of all, construct a clear non-linear two-class dataset with two features of 100 obersations:

```
#construct the dataset
X=[[random.uniform(-1,1),random.uniform(-1,1)] for i in range(100)]
Y0=[x1+x1*x1+x2+x2*x2 for (x1,x2) in X]
```

```
#transform into two classes
y_prob=[math.exp(y)/(1+math.exp(y)) for y in Y0]
Y=[prob>0.6 for prob in y_prob]
```

```
#convert into features and label dataset, as np format
np_X=np.array(X)     #features
np_Y=np.array(Y)     #labels
```

```
#split training and test dataset  70%/30%
from sklearn.model_selection import train_test_split
X_train,X_test, y_train, y_test =train_test_split(np_X,np_Y,test_size=0.3, random_state=4)
```

```
#function to plot ROC
import matplotlib.pyplot as plt
def plot_ROC_curve(model,data_feature,true_label,method_des):
    clf=model
    classes = clf.classes_
    try:
        probs = clf.predict_proba(data_feature)
    except AttributeError:
        print("The {} classifier does not apear to support prediction probabilties, so an ROC curve can't be
created. You can try adding `probability = True` to the model specification or use a different
model.".format(type(clf)))
        return
    predictions = clf.predict(data_feature)

    #setup axis for plotting
    fig, ax = plt.subplots(figsize = (5,5))

    #We can return the AUC values, in case they are useful
    aucVals = []
    for classIndex, className in enumerate(classes):        #Setup binary classes
        truths = [1 if c == className else 0 for c in true_label]
        predict = [1 if c == className else 0 for c in predictions]
        scores = probs[:, classIndex]

        #Get the ROC curve
        fpr, tpr, thresholds = sklearn.metrics.roc_curve(truths, scores)
        #fpr, tpr, thresholds = sklearn.metrics.roc_curve(truths, predictions)
        auc = sklearn.metrics.auc(fpr, tpr)
        aucVals.append(auc)

        #Plot the class's line
        ax.plot(fpr, tpr, label = "{} (AUC ${:.3f}$)".format(str(className).split(':')[0], auc))

    #Make the plot nice, then display it
    ax.set_title('Receiver Operating Characteristics')
    plt.plot([0,1], [0,1], color = 'k', linestyle='--')
    ax.set_xlabel('False Positive Rate')
    ax.set_ylabel('True Positive Rate')
    ax.legend(loc = 'lower right')
    plt.show()
    #plt.close()
```

1.1 Build linear SVM model: calculate the error rate and plot the graph

```
#build the SVM linear model
svm_linear=SVC(kernel = 'linear', probability = True)
svm_linear.fit(X_train,y_train)

#predict Y based on SVM, training
svm_pY=svm_linear.predict(X_train)
svm_pY_test=svm_linear.predict(X_test)
```

```
#calculate error rate, AUC score on Training Dataset
error_rate_linear=1 -  sklearn.metrics.accuracy_score(y_train,svm_pY)
auc_score_linear=sklearn.metrics.roc_auc_score(y_train,svm_pY)
print('Results for Training:')
print('Error Rate of linear SVM model:',round(error_rate_linear,4))
print('AUC score of best linear SVM model:',round(auc_score_linear,4))
```

```
Results for Training:
Error Rate of linear SVM model: 0.1
AUC score of best linear SVM model: 0.9
```

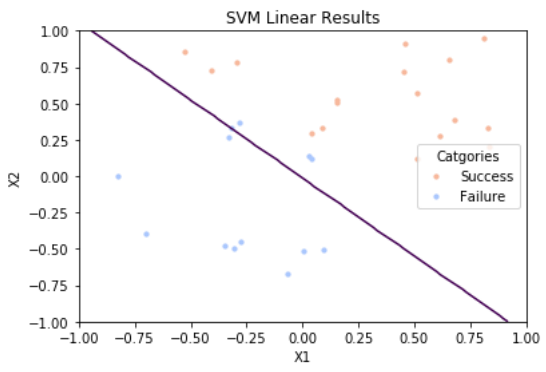
```
#calculate error rate, AUC score on Test Dataset
error_rate_linear_test=1 -  sklearn.metrics.accuracy_score(y_test,svm_pY_test)
auc_score_linear_test=sklearn.metrics.roc_auc_score(y_test,svm_pY_test)
print('Results for Test:')
print('Error Rate of linear SVM model:',round(error_rate_linear_test,4))
print('AUC score of best linear SVM model:',round(auc_score_linear_test,4))
```

```
Results for Test:
Error Rate of linear SVM model: 0.1
AUC score of best linear SVM model: 0.8846
```

```
#save the training and test results separately
method_label=[]
method_label.append('Linear SVM')
train_error_rate=[error_rate_linear]
train_auc=[auc_score_linear]
test_error_rate=[error_rate_linear_test]
test_auc=[auc_score_linear_test]
```

```
#plot prediction results based on Test dataset
labels=['Success','Failure']
pallet = seaborn.color_palette(palette='coolwarm', n_colors = len(set(np_Y)))

#plot points of success first
index_true=np.where(y_test==True)[0]
index_true=index_true.astype(np.int16)
x1=X_test[index_true,0]
x2=X_test[index_true,1]
plt.scatter(x1,x2,s=10,c=pallet[1],label=labels[0])

#plot points of failure
index_false=np.where(y_test==False)[0]
index_false=index_false.astype(np.int16)
x1=X_test[index_false,0]
x2=X_test[index_false,1]
plt.scatter(x1,x2,s=10,c=pallet[0],label=labels[1])

#np.meshgrid, set regions
xx, yy = np.meshgrid(np.linspace(-1,1,100), np.linspace(-1,1,100))

#predict the results of X1,X2, based on logistic regression bayes
Z = svm_linear.predict_proba(np.c_[xx.ravel(), yy.ravel()])
Z = Z[:,1].reshape(xx.shape)

#plot the contour and regions of success (in blue) or failure(in red)
plt.contour(xx, yy, Z, [0.5],color='r')

plt.legend(loc = 'center right', title = 'Catgories')
plt.xlabel('X1')
plt.ylabel('X2')
plt.title('SVM Linear Results')
plt.show()
```

```
'c' argument looks like a single numeric RGB or RGBA sequence, which should be avoided as value-mapping will
have precedence in case its length matches with 'x' & 'y'.  Please use a 2-D array with a single row if you
really want to specify the same RGB or RGBA value for all points.
'c' argument looks like a single numeric RGB or RGBA sequence, which should be avoided as value-mapping will
have precedence in case its length matches with 'x' & 'y'.  Please use a 2-D array with a single row if you
really want to specify the same RGB or RGBA value for all points.
C:\Users\zhuangchu\Anaconda3\lib\site-packages\ipykernel_launcher.py:27: UserWarning: The following kwargs
were not used by contour: 'color'
```



As we could see above clearly, there is a linear decision boundary in test data set, while some mis-classified datapoints (blue one above the decision boundary) since the datasets is non-linear separated.

Over all, for SVM linear model, error rate is around 0.1 for both Training and Test dataset. AUC score is little bit higher for training 0.9 than Testing 0.8846 for linear SVM model.

1.2 Build Non-linear SVM model (Radial Kernel): calculate the error rate and plot the graph

```
#build the SVM linear model
from sklearn.svm import SVC
svm_nlinear=SVC(kernel = 'rbf', gamma='auto',probability = True)
svm_nlinear.fit(X_train,y_train)

#predict Y based on SVM, training
svm_pY=svm_nlinear.predict(X_train)
svm_pY_test=svm_nlinear.predict(X_test)
```

```
#calculate error rate, AUC score on Training Dataset
error_rate_nlinear=1 -  sklearn.metrics.accuracy_score(y_train,svm_pY)
auc_score_nlinear=sklearn.metrics.roc_auc_score(y_train,svm_pY)
print('Results for Training:')
print('Error Rate of linear SVM model:',round(error_rate_nlinear,4))
print('AUC score of best linear SVM model:',round(auc_score_nlinear,4))
```

```
Results for Training:
Error Rate of linear SVM model: 0.0429
AUC score of best linear SVM model: 0.9542
```

```
#calculate error rate, AUC score on Test Dataset
error_rate_nlinear_test=1 -  sklearn.metrics.accuracy_score(y_test,svm_pY_test)
auc_score_nlinear_test=sklearn.metrics.roc_auc_score(y_test,svm_pY_test)
print('Results for Test:')
print('Error Rate of linear SVM model:',round(error_rate_nlinear_test,4))
print('AUC score of best linear SVM model:',round(auc_score_nlinear_test,4))
```

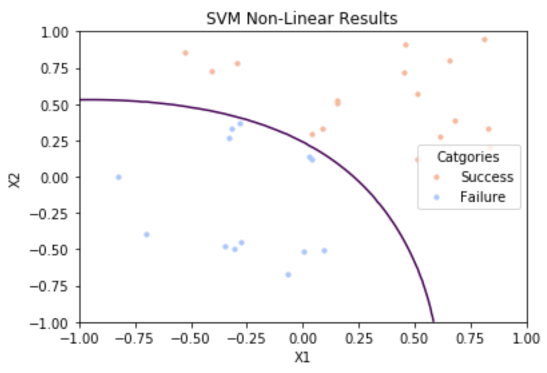
```
Results for Test:
Error Rate of linear SVM model: 0.0
AUC score of best linear SVM model: 1.0
```

```
#save the training and test results separately
method_label=['Linear SVM','Non-linear SVM']
train_error_rate=[error_rate_linear,error_rate_nlinear]
train_auc=[auc_score_linear,auc_score_nlinear]
test_error_rate=[error_rate_linear_test,error_rate_nlinear_test]
test_auc=[auc_score_linear_test,auc_score_nlinear_test]


df_svm=pd.DataFrame({'Train_Error':train_error_rate,'Train_AUC':train_auc,'Test_Error':test_error_rate,'Test_
AUC':test_auc},index=method_label)
```

```
#plot prediction results based on Test dataset
```

```
labels=['Success','Failure']
pallet = seaborn.color_palette(palette='coolwarm', n_colors = len(set(np_Y)))

#plot points of success first
index_true=np.where(y_test==True)[0]
index_true=index_true.astype(np.int16)
x1=X_test[index_true,0]
x2=X_test[index_true,1]
plt.scatter(x1,x2,s=10,c=pallet[1],label=labels[0])

#plot points of failure
index_false=np.where(y_test==False)[0]
index_false=index_false.astype(np.int16)
x1=X_test[index_false,0]
x2=X_test[index_false,1]
plt.scatter(x1,x2,s=10,c=pallet[0],label=labels[1])

#np.meshgrid, set regions
xx, yy = np.meshgrid(np.linspace(-1,1,100), np.linspace(-1,1,100))

#predict the results of X1,X2, based on logistic regression bayes
Z = svm_nlinear.predict_proba(np.c_[xx.ravel(), yy.ravel()])
Z = Z[:,1].reshape(xx.shape)

#plot the contour and regions of success (in blue) or failure(in red)
plt.contour(xx, yy, Z, [0.5],color='r')

plt.legend(loc = 'center right', title = 'Catgories')
plt.xlabel('X1')
plt.ylabel('X2')
plt.title('SVM Non-Linear Results')
plt.show()
```

```
'c' argument looks like a single numeric RGB or RGBA sequence, which should be avoided as value-mapping will
have precedence in case its length matches with 'x' & 'y'.  Please use a 2-D array with a single row if you
really want to specify the same RGB or RGBA value for all points.
'c' argument looks like a single numeric RGB or RGBA sequence, which should be avoided as value-mapping will
have precedence in case its length matches with 'x' & 'y'.  Please use a 2-D array with a single row if you
really want to specify the same RGB or RGBA value for all points.
C:\Users\zhuangchu\Anaconda3\lib\site-packages\ipykernel_launcher.py:27: UserWarning: The following kwargs
were not used by contour: 'color'
```



For the non-linear SVM model, on the same test dataset, we could see that it better captures the non-linear separation in the dataset; with only few data point close to the boundary and non mis-classification.

Both the Training error and test error of non-linear SVM outperforms the linear SVM model (as shown in the table below). Also for AUC score, the non-linear SVM model is higher than linear one, and even equals to 1 for non-linear SVM model on this non-linear dataset.

Overal,non-linear SVM performs better than linear SVM on the non-linear dataset.

```
#show training results for both SVM model
df_svm
```

```
.dataframe tbody tr th {
    vertical-align: top;
}

.dataframe thead th {
    text-align: right;
}
```

	Train_Error	Train_AUC	Test_Error	Test_AUC
Linear SVM	0.100000	0.900000	0.1	0.884615
Non-linear SVM	0.042857	0.954167	0.0	1.000000

SVM vs. Logistic Regression

1. Construct the dataset-overlapping and nolinear with observations:n=500, and features: p=2.

```
#construct the dataset
import math
X=[[random.uniform(-1,1),random.uniform(-1,1)] for i in range(500)]
Y0=[x1+x1*x1+x2+x2*x2+random.normalvariate(0,0.5) for (x1,x2) in X]
```

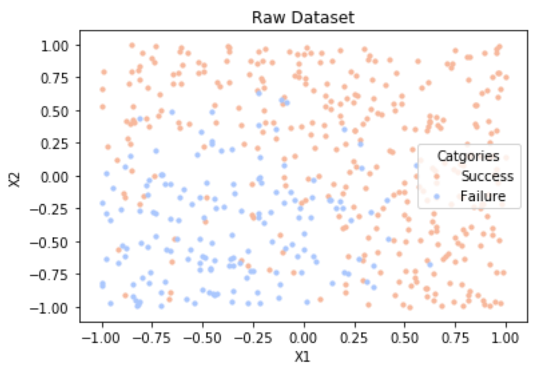
```
#transform into two classes
y_prob=[math.exp(y)/(1+math.exp(y)) for y in Y0]
Y=[prob>0.5 for prob in y_prob]
```

```
#convert into features and label dataset, as np format
np_X=np.array(X)     #features
np_Y=np.array(Y)     #labels
```

2. Plot the observations (Raw Dataset)

```
#plot the raw data
labels=['Success','Failure']
pallet = seaborn.color_palette(palette='coolwarm', n_colors = len(set(np_Y)))

#plot points of success first
index_true=np.where(np_Y==True)[0]
index_true=index_true.astype(np.int16)
x1=np_X[index_true,0]
x2=np_X[index_true,1]
plt.scatter(x1,x2,s=10,c=pallet[1],label=labels[0])

#plot points of failure
index_false=np.where(np_Y==False)[0]
index_false=index_false.astype(np.int16)
x1=np_X[index_false,0]
x2=np_X[index_false,1]
plt.scatter(x1,x2,s=10,c=pallet[0],label=labels[1])

plt.legend(loc = 'center right', title = 'Catgories')
plt.xlabel('X1')
plt.ylabel('X2')
plt.title('Raw Dataset')
plt.show()
```

```
'c' argument looks like a single numeric RGB or RGBA sequence, which should be avoided as value-mapping will have precedence in case its
length matches with 'x' & 'y'.  Please use a 2-D array with a single row if you really want to specify the same RGB or RGBA value for all
points.
'c' argument looks like a single numeric RGB or RGBA sequence, which should be avoided as value-mapping will have precedence in case its
length matches with 'x' & 'y'.  Please use a 2-D array with a single row if you really want to specify the same RGB or RGBA value for all
points.
```



4. Fit a Logistic Regression

```
#build linear logistic regression model
from sklearn.linear_model import LogisticRegression
lg_linear=LogisticRegression(solver='liblinear')
lg_linear.fit(np_X,np_Y)
```

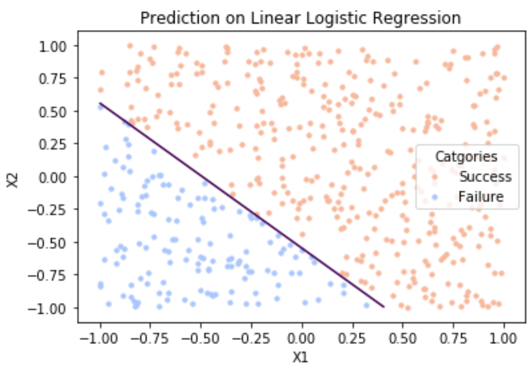
```
LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True,
                   intercept_scaling=1, l1_ratio=None, max_iter=100,
                   multi_class='warn', n_jobs=None, penalty='l2',
                   random_state=None, solver='liblinear', tol=0.0001, verbose=0,
                   warm_start=False)
```

5. Plot prediction for linear Logistic Regression

```
#predict Y based on train feature
lg_pY_linear=lg_linear.predict(np_X)
```

```
#plot the predicted results of linear Logistic Regression
labels=['Success','Failure']
pallet = seaborn.color_palette(palette='coolwarm', n_colors = len(set(lg_pY_linear)))

#plot points of success first
index_true=np.where(lg_pY_linear==True)[0]
index_true=index_true.astype(np.int16)
x1=np_X[index_true,0]
x2=np_X[index_true,1]
plt.scatter(x1,x2,s=10,c=pallet[1],label=labels[0])

#plot points of failure
index_false=np.where(lg_pY_linear==False)[0]
index_false=index_false.astype(np.int16)
x1=np_X[index_false,0]
x2=np_X[index_false,1]
plt.scatter(x1,x2,s=10,c=pallet[0],label=labels[1])

#np.meshgrid, set regions
xx, yy = np.meshgrid(np.linspace(-1,1,100), np.linspace(-1,1,100))

#predict the results of X1,X2, based on logistic regression bayes
Z = lg_linear.predict_proba(np.c_[xx.ravel(), yy.ravel()])
Z = Z[:,1].reshape(xx.shape)

#plot the contour and regions of success (in blue) or failure(in red)
plt.contour(xx, yy, Z, [0.5],color='r')

plt.legend(loc = 'center right', title = 'Catgories')
plt.xlabel('X1')
plt.ylabel('X2')
plt.title('Prediction on Linear Logistic Regression')
plt.show()
```

```
'c' argument looks like a single numeric RGB or RGBA sequence, which should be avoided as value-mapping will have precedence in case its
length matches with 'x' & 'y'.  Please use a 2-D array with a single row if you really want to specify the same RGB or RGBA value for all
points.
'c' argument looks like a single numeric RGB or RGBA sequence, which should be avoided as value-mapping will have precedence in case its
length matches with 'x' & 'y'.  Please use a 2-D array with a single row if you really want to specify the same RGB or RGBA value for all
points.
C:\Users\zhuangchu\Anaconda3\lib\site-packages\ipykernel_launcher.py:27: UserWarning: The following kwargs were not used by contour: 'color'
```



6. Non linear Logistic Regression

First of all, transform the X to derive non-linear predictors

```
#transform to non-linear features
X1=[(x1,x1*x1,x2,x2*x2,x1*x2) for (x1,x2) in X]
np_X1=np.array(X1)
```

```
#build non linear logistic regression model
lg_nlinear=LogisticRegression(solver='liblinear')
lg_nlinear.fit(np_X1,np_Y)
```

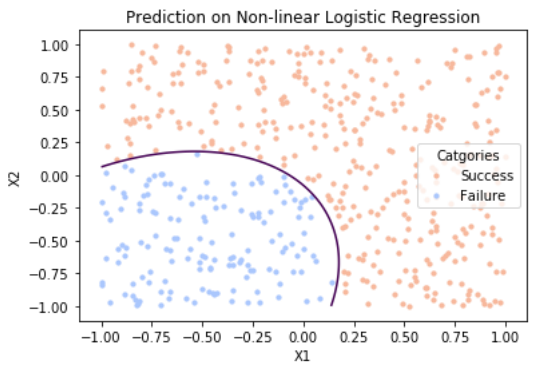
```
LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True,
                   intercept_scaling=1, l1_ratio=None, max_iter=100,
                   multi_class='warn', n_jobs=None, penalty='l2',
                   random_state=None, solver='liblinear', tol=0.0001, verbose=0,
                   warm_start=False)
```

7. Plot prediction for non-Linear Logistic Regression

```
#predict Y based on train feature
lg_pY_nlinear=lg_nlinear.predict(np_X1)
```

```
#plot the predicted results, colored according to the new labeled class
labels=['Success','Failure']
pallet = seaborn.color_palette(palette='coolwarm', n_colors = len(set(lg_pY_linear)))

#plot points of success first
index_true=np.where(lg_pY_nlinear==True)[0]
index_true=index_true.astype(np.int16)
x1=np_X[index_true,0]
x2=np_X[index_true,1]
plt.scatter(x1,x2,s=10,c=pallet[1],label=labels[0])

#plot points of failure
index_false=np.where(lg_pY_nlinear==False)[0]
index_false=index_false.astype(np.int16)
x1=np_X[index_false,0]
x2=np_X[index_false,1]
plt.scatter(x1,x2,s=10,c=pallet[0],label=labels[1])

#np.meshgrid, set regions
xx, yy = np.meshgrid(np.linspace(-1,1,100), np.linspace(-1,1,100))
xy=zip(xx.ravel(),yy.ravel())
xy1=[(xx,xx*xx,yy,yy*yy,xx*yy) for (xx,yy) in xy]
np_xy1=np.array(xy1)

#predict the results of X1,X2, based on logistic regression bayes
#Z = lg_nlinear.predict_proba(np.c_[xx.ravel(), yy.ravel()])
Z = lg_nlinear.predict_proba(np_xy1)
Z = Z[:,1].reshape(xx.shape)

#plot the contour and regions of success (in blue) or failure(in red)
plt.contour(xx, yy, Z, [0.5],color='r')

plt.legend(loc = 'center right', title = 'Catgories')
plt.xlabel('X1')
plt.ylabel('X2')
plt.title('Prediction on Non-linear Logistic Regression')
plt.show()
```

```
'c' argument looks like a single numeric RGB or RGBA sequence, which should be avoided as value-mapping will have precedence in case its
length matches with 'x' & 'y'.  Please use a 2-D array with a single row if you really want to specify the same RGB or RGBA value for all
points.
'c' argument looks like a single numeric RGB or RGBA sequence, which should be avoided as value-mapping will have precedence in case its
length matches with 'x' & 'y'.  Please use a 2-D array with a single row if you really want to specify the same RGB or RGBA value for all
points.
C:\Users\zhuangchu\Anaconda3\lib\site-packages\ipykernel_launcher.py:31: UserWarning: The following kwargs were not used by contour: 'color'
```

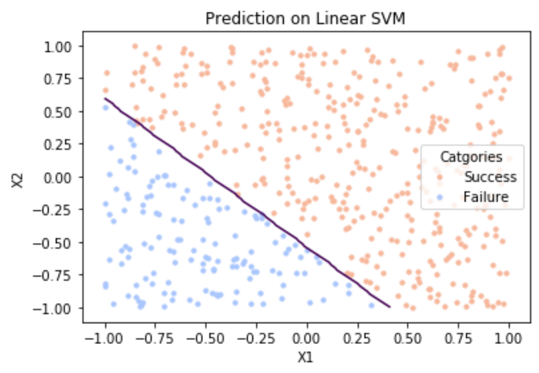


8. Linear SVM model

```
#build the SVM linear model
svm_linear=SVC(kernel = 'linear', probability = True)
svm_linear.fit(np_X,np_Y)

#predict Y based on SVM, training
svm_pY_linear=svm_linear.predict(np_X)
```

```
#plot the predicted results
labels=['Success','Failure']
pallet = seaborn.color_palette(palette='coolwarm', n_colors = len(set(svm_pY_linear)))

#plot points of success first
index_true=np.where(svm_pY_linear==True)[0]
index_true=index_true.astype(np.int16)
x1=np_X[index_true,0]
x2=np_X[index_true,1]
plt.scatter(x1,x2,s=10,c=pallet[1],label=labels[0])

#plot points of failure
index_false=np.where(svm_pY_linear==False)[0]
index_false=index_false.astype(np.int16)
x1=np_X[index_false,0]
x2=np_X[index_false,1]
plt.scatter(x1,x2,s=10,c=pallet[0],label=labels[1])

#np.meshgrid, set regions
xx, yy = np.meshgrid(np.linspace(-1,1,100), np.linspace(-1,1,100))

#predict the results of X1,X2, based on logistic regression bayes
Z = svm_linear.predict_proba(np.c_[xx.ravel(), yy.ravel()])
Z = Z[:,1].reshape(xx.shape)

#plot the contour and regions of success (in blue) or failure(in red)
plt.contour(xx, yy, Z, [0.5],color='r')

plt.legend(loc = 'center right', title = 'Catgories')
plt.xlabel('X1')
plt.ylabel('X2')
plt.title('Prediction on Linear SVM')
plt.show()
```

```
'c' argument looks like a single numeric RGB or RGBA sequence, which should be avoided as value-mapping will have precedence in case its
length matches with 'x' & 'y'.  Please use a 2-D array with a single row if you really want to specify the same RGB or RGBA value for all
points.
'c' argument looks like a single numeric RGB or RGBA sequence, which should be avoided as value-mapping will have precedence in case its
length matches with 'x' & 'y'.  Please use a 2-D array with a single row if you really want to specify the same RGB or RGBA value for all
points.
C:\Users\zhuangchu\Anaconda3\lib\site-packages\ipykernel_launcher.py:27: UserWarning: The following kwargs were not used by contour: 'color'
```



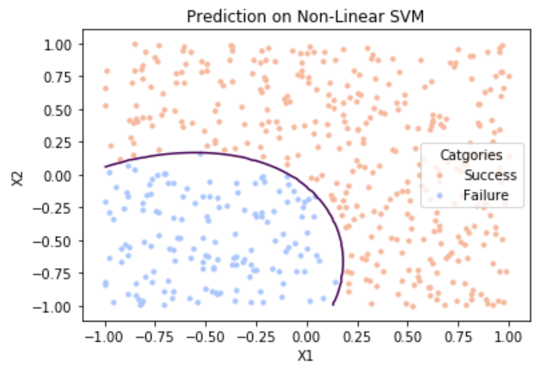
9. Non-Linear SVM model

```
#build the SVM linear model
svm_nlinear=SVC(kernel = 'rbf', gamma='auto',probability = True)
svm_nlinear.fit(np_X,np_Y)

#predict Y based on SVM, training
svm_pY_nlinear=svm_nlinear.predict(np_X)
```

```
#plot the predicted results
labels=['Success','Failure']
pallet = seaborn.color_palette(palette='coolwarm', n_colors = len(set(svm_pY_nlinear)))

#plot points of success first
index_true=np.where(svm_pY_nlinear==True)[0]
index_true=index_true.astype(np.int16)
x1=np_X[index_true,0]
```

```
x2=np_X[index_true,1]
plt.scatter(x1,x2,s=10,c=pallet[1],label=labels[0])

#plot points of failure
index_false=np.where(svm_pY_nlinear==False)[0]
index_false=index_false.astype(np.int16)
x1=np_X[index_false,0]
x2=np_X[index_false,1]
plt.scatter(x1,x2,s=10,c=pallet[0],label=labels[1])

#np.meshgrid, set regions
xx, yy = np.meshgrid(np.linspace(-1,1,100), np.linspace(-1,1,100))

#predict the results of X1,X2, based on logistic regression bayes
Z = svm_nlinear.predict_proba(np.c_[xx.ravel(), yy.ravel()])
Z = Z[:,1].reshape(xx.shape)

#plot the contour and regions of success (in blue) or failure(in red)
plt.contour(xx, yy, Z, [0.5],color='r')

plt.legend(loc = 'center right', title = 'Catgories')
plt.xlabel('X1')
plt.ylabel('X2')
plt.title('Prediction on Non-Linear SVM')
plt.show()
```

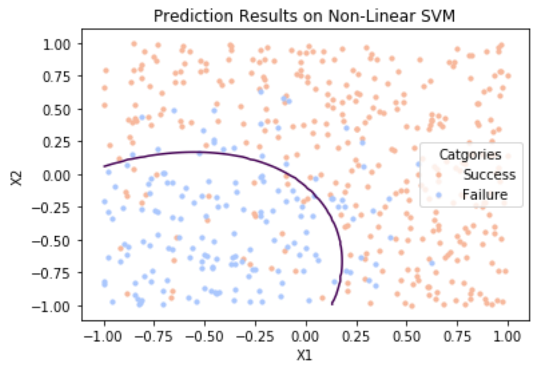
```
'c' argument looks like a single numeric RGB or RGBA sequence, which should be avoided as value-mapping will have precedence in case its
length matches with 'x' & 'y'.  Please use a 2-D array with a single row if you really want to specify the same RGB or RGBA value for all
points.
'c' argument looks like a single numeric RGB or RGBA sequence, which should be avoided as value-mapping will have precedence in case its
length matches with 'x' & 'y'.  Please use a 2-D array with a single row if you really want to specify the same RGB or RGBA value for all
points.
C:\Users\zhuangchu\Anaconda3\lib\site-packages\ipykernel_launcher.py:27: UserWarning: The following kwargs were not used by contour: 'color'
```



Also, for better visualization, I further plot the prediction results based on true label:

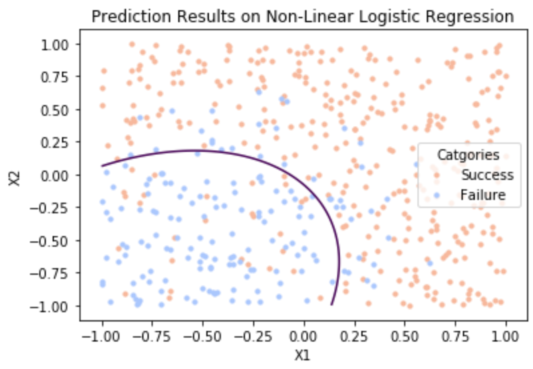
```
#plot the predicted results (Nonlinear SVM)
labels=['Success','Failure']
pallet = seaborn.color_palette(palette='coolwarm', n_colors = len(set(np_Y)))

#plot points of success first
index_true=np.where(np_Y==True)[0]
index_true=index_true.astype(np.int16)
x1=np_X[index_true,0]
x2=np_X[index_true,1]
plt.scatter(x1,x2,s=10,c=pallet[1],label=labels[0])

#plot points of failure
index_false=np.where(np_Y==False)[0]
index_false=index_false.astype(np.int16)
x1=np_X[index_false,0]
x2=np_X[index_false,1]
plt.scatter(x1,x2,s=10,c=pallet[0],label=labels[1])

#np.meshgrid, set regions
xx, yy = np.meshgrid(np.linspace(-1,1,100), np.linspace(-1,1,100))

#predict the results of X1,X2, based on logistic regression bayes
Z = svm_nlinear.predict_proba(np.c_[xx.ravel(), yy.ravel()])
Z = Z[:,1].reshape(xx.shape)

#plot the contour and regions of success (in blue) or failure(in red)
plt.contour(xx, yy, Z, [0.5],color='r')

plt.legend(loc = 'center right', title = 'Catgories')
plt.xlabel('X1')
plt.ylabel('X2')
plt.title('Prediction Results on Non-Linear SVM')
plt.show()
```

```
'c' argument looks like a single numeric RGB or RGBA sequence, which should be avoided as value-mapping will have precedence in case its
length matches with 'x' & 'y'.  Please use a 2-D array with a single row if you really want to specify the same RGB or RGBA value for all
points.
'c' argument looks like a single numeric RGB or RGBA sequence, which should be avoided as value-mapping will have precedence in case its
length matches with 'x' & 'y'.  Please use a 2-D array with a single row if you really want to specify the same RGB or RGBA value for all
points.
C:\Users\zhuangchu\Anaconda3\lib\site-packages\ipykernel_launcher.py:27: UserWarning: The following kwargs were not used by contour: 'color'
```



```
#plot the predicted results (Nonlinear Logistic Regression)
labels=['Success','Failure']
pallet = seaborn.color_palette(palette='coolwarm', n_colors = len(set(np_Y)))

#plot points of success first
index_true=np.where(np_Y==True)[0]
index_true=index_true.astype(np.int16)
x1=np_X[index_true,0]
x2=np_X[index_true,1]
plt.scatter(x1,x2,s=10,c=pallet[1],label=labels[0])

#plot points of failure
index_false=np.where(np_Y==False)[0]
index_false=index_false.astype(np.int16)
x1=np_X[index_false,0]
x2=np_X[index_false,1]
plt.scatter(x1,x2,s=10,c=pallet[0],label=labels[1])

#np.meshgrid, set regions
xx, yy = np.meshgrid(np.linspace(-1,1,100), np.linspace(-1,1,100))
xy=zip(xx.ravel(),yy.ravel())
xy1=[(xx,xx*xx,yy,yy*yy,xx*yy) for (xx,yy) in xy]
np_xy1=np.array(xy1)

#predict the results of X1,X2, based on logistic regression bayes
#Z = lg_nlinear.predict_proba(np.c_[xx.ravel(), yy.ravel()])
Z = lg_nlinear.predict_proba(np_xy1)
Z = Z[:,1].reshape(xx.shape)

#plot the contour and regions of success (in blue) or failure(in red)
plt.contour(xx, yy, Z, [0.5],color='r')

plt.legend(loc = 'center right', title = 'Catgories')
plt.xlabel('X1')
plt.ylabel('X2')
plt.title('Prediction Results on Non-Linear Logistic Regression')
plt.show()
```

```
'c' argument looks like a single numeric RGB or RGBA sequence, which should be avoided as value-mapping will have precedence in case its
length matches with 'x' & 'y'.  Please use a 2-D array with a single row if you really want to specify the same RGB or RGBA value for all
points.
'c' argument looks like a single numeric RGB or RGBA sequence, which should be avoided as value-mapping will have precedence in case its
length matches with 'x' & 'y'.  Please use a 2-D array with a single row if you really want to specify the same RGB or RGBA value for all
points.
C:\Users\zhuangchu\Anaconda3\lib\site-packages\ipykernel_launcher.py:31: UserWarning: The following kwargs were not used by contour: 'color'
```



10. As we could see from the visualizations above, for this dataset, both non-linear logistic regression and non-linear SVM performs close to each other and very well (capture the 2-degree polynomial relationship between Y and X).

    However it really depends on the dataset constructed. Since in this dataset, the relationship is simple (2 degree polynomial) and the non-linear logistic regression also built upon this function, so the non-linear logistic regression model yields good performance. **However, if the non-linear relationship is complex or unknown, the non-linear logistic regression model which needs to be built on pre-defined fixed models might fail. In this case, non-linear SVM is more flexible and could perform better, while it might not strictly find and follow the polynomial relationship in the dataset**.

    The performance of Non-linear logistic regression based on the right guess of the relationshop in the dataset; while SVM is more flexible and welcomes more complex non-linear relationship. However, if there is not a clear hyperplane to separate the predicted classes (no matter linear or non-linear), the SVM could fail as well.

Tuning Cost

11. Generate Data which are barely linearly separable:

```
#construct the dataset
X=[[random.uniform(-1,1),random.uniform(-1,1)] for i in range(500)]
Y0=[x1+x2+random.normalvariate(0,0.5) for (x1,x2) in X]
```

```
#transform into two classes
y_prob=[math.exp(y)/(1+math.exp(y)) for y in Y0]
Y=[prob>0.5 for prob in y_prob]
```

```
#convert into features and label dataset, as np format
np_X=np.array(X)     #features
np_Y=np.array(Y)     #labels
```

```
#split training and test dataset  70%/30%
X_train,X_test, y_train, y_test =train_test_split(np_X,np_Y,test_size=0.3, random_state=4)
```

12/13. Cross validation of Cost in linear-SVM:

```
#10-fold cross validation
from sklearn.model_selection import StratifiedKFold
skf = StratifiedKFold(n_splits=10,random_state=4)
#kf = StratifiedKFold(n_splits=10,random_state=4)

#define variable to save error rates
train_error_allc=[]
cv_error_allc=[]
train_error_allc2=[]
test_error_allc=[]

#tune cost parameters
cost=[0.001,0.01,0.05,0.1,0.5,1,10,100,1000,10000]
#np.linspace(0.1,1,10)

for i in range(len(cost)):
    svm_linear=SVC(C=cost[i], kernel = 'linear', probability = True)  #build the model for each cost
    train_error=[]
    cv_error=[]
    for train_index, test_index in skf.split(X_train,y_train):
        X_train10, X_test10 = X_train[train_index], X_train[test_index]
        y_train10, y_test10 = y_train[train_index], y_train[test_index]
        svm_linear.fit(X_train10,y_train10)

        #calculate accuracy based on CV training samples
        er_score=1-sklearn.metrics.accuracy_score(y_train10, svm_linear.predict(X_train10))
        train_error.append(er_score)
        #calculate accuracy based on test(validation) samples
        er_score=1-sklearn.metrics.accuracy_score(y_test10, svm_linear.predict(X_test10))
        cv_error.append(er_score)

    train_error_allc.append(np.mean(train_error))
    cv_error_allc.append(np.mean(cv_error))
    #calculate test error
    svm_linear.fit(X_train,y_train)
    er_score=1-sklearn.metrics.accuracy_score(y_test, svm_linear.predict(X_test))
    test_error_allc.append(er_score)
    er_score=1-sklearn.metrics.accuracy_score(y_train, svm_linear.predict(X_train))
    train_error_allc2.append(er_score)
```

```
df_cost=pd.DataFrame({'CV_Train_Error':train_error_allc,'CV_Error':cv_error_allc,'Test_Error':test_error_allc,'Train_Error_noCV':train_error
_allc2},index=cost)
df_cost
```

```
.dataframe tbody tr th {
    vertical-align: top;
}

.dataframe thead th {
    text-align: right;
}
```

	CV_Train_Error	CV_Error	Test_Error	Train_Error_noCV
0.001	0.488571	0.488571	0.513333	0.488571
0.010	0.164122	0.170971	0.160000	0.162857
0.050	0.160945	0.162400	0.153333	0.160000
0.100	0.159358	0.170971	0.153333	0.162857
0.500	0.158093	0.165257	0.173333	0.162857
1.000	0.156821	0.165257	0.173333	0.162857
10.000	0.156822	0.165257	0.166667	0.157143
100.000	0.156504	0.165257	0.166667	0.154286
1000.000	0.156504	0.165257	0.166667	0.154286
10000.000	0.156504	0.165257	0.166667	0.157143
100000.000	0.158409	0.165257	0.173333	0.160000

12. As we could see above, with the increasing of cost, the training errors decreases and at the same time CV error decreases as well (while a slightly larger than training error), which indicates that the bias decreases along with the increase of cost.However, the decrease of CV error is not inline with the decrease of cost; **after c=0.5, the cv error stablizes at 0.165257, while the error rate of training still goes down, indicating the potential of 'over-fitting' with larger cost value.**
    If considering both CV training error and CV error, the smallest best cost value is 100 (while 0.5 and 1 also perform quite close)
13. **When expanding to test dataset, we could see that however, when c=0.05/0.1, the model has the fewest test error; while the CV error and training error is not optimized then (slightly larger then the fewest errors for training and cv, c=0.5/1).** This discrepency between results of SVM classification on test dataset and training dataset further validates the **advantage of training but poorer at generalization** of high cost value. When training SVM model, it is important to keep in mind to loose the cost value a little bit which might permit better results for testing and further prediction.

Application: Predicting attitudes towards racist college professors

```
#load the data
df_gss_train=pd.read_csv('data/gss_train.csv')
df_gss_test=pd.read_csv('data/gss_test.csv')
df_gss_train.head()
```

```
.dataframe tbody tr th {
    vertical-align: top;
}

.dataframe thead th {
    text-align: right;
}
```

	age	attend	authoritarianism	black	born	childs	colath	colrac	colcom	colmil	...	partyid_3_Ind	partyid_3_Rep	
0	21	0	4	0	0	0	1	1	0	1	...	1	0	1
1	42	0	4	0	0	2	0	1	1	0	...	1	0	0
2	70	1	1	1	0	3	0	1	1	0	...	0	0	0
3	35	3	2	0	0	2	0	1	0	1	...	1	0	0
4	24	3	6	0	1	3	1	1	0	0	...	1	0	1

5 rows × 56 columns

Organize the data:

```
#organize the data to fit in model, for feature and predict value
#drop the predict value from the feature set
df_gss_train0=df_gss_train.drop('colrac',axis=1)
df_gss_test0=df_gss_test.drop('colrac',axis=1)

np_gss_train_feature=df_gss_train0.values
np_gss_train_y=df_gss_train['colrac'].values

np_gss_test_feature=df_gss_test0.values
np_gss_test_y=df_gss_test['colrac'].values
```

15. Fit in SVM model and tune Cost by 10-fold cross validation

```
#10-fold cross validation
skf = StratifiedKFold(n_splits=10,random_state=4)

#define variable to save error rates
#train_error_allc=[]
cv_error_allc=[]
train_error_allc2=[]
test_error_allc=[]
auc_allc=[]

#tune cost parameters
cost=[0.001,0.01,0.05,0.1,0.5,1,10]

for i in range(len(cost)):
    svm_linear=SVC(C=cost[i], kernel = 'linear', probability = True)  #build the model for each cost
    test_error=[]
    cv_error=[]
    for train_index, test_index in skf.split(np_gss_train_feature,np_gss_train_y):
        X_train10, X_test10 = np_gss_train_feature[train_index], np_gss_train_feature[test_index]
        y_train10, y_test10 = np_gss_train_y[train_index], np_gss_train_y[test_index]
        svm_linear.fit(X_train10,y_train10)

        #calculate accuracy based on test(validation) samples
        er_score=1-sklearn.metrics.accuracy_score(y_test10, svm_linear.predict(X_test10))
        cv_error.append(er_score)

    #train_error_allc.append(np.mean(train_error))
    cv_error_allc.append(np.mean(cv_error))
    #calculate test error
    svm_linear.fit(np_gss_train_feature,np_gss_train_y)
    er_score=1-sklearn.metrics.accuracy_score(np_gss_test_y, svm_linear.predict(np_gss_test_feature))
    test_error_allc.append(er_score)
    #calculate auc for test dataset
    auc_score=sklearn.metrics.roc_auc_score(np_gss_test_y, svm_linear.predict(np_gss_test_feature))
    auc_allc.append(auc_score)
    #er_score=1-sklearn.metrics.accuracy_score(np_gss_train_y, svm_linear.predict(np_gss_train_feature))
    #train_error_allc2.append(er_score)
```

```
df_cost=pd.DataFrame({'CV_Error':cv_error_allc,'Test_Error':test_error_allc,'Test_AUC':auc_allc},index=cost)
df_cost
```

```
.dataframe tbody tr th {
    vertical-align: top;
}

.dataframe thead th {
    text-align: right;
}
```

	CV_Error	Test_Error	Test_AUC
0.001	0.239674	0.219067	0.779386
0.010	0.205949	0.196755	0.800141
0.050	0.201224	0.206897	0.789176
0.100	0.203945	0.219067	0.776018
0.500	0.203251	0.215010	0.779791
1.000	0.205958	0.217039	0.777905
10.000	0.207314	0.219067	0.776018

From the table above we could see that when cost value equals to 0.05, the cv error is smallest (0.201224), while generally the averaged cv error is very close around 0.20 when the cost value changes from 0.01,0.05,0.5,1,10. However, when the cost value equals to 0.01, it yields the smallest **test error** which is even less than 0.20 (AUC score around 0.80),and for cost=0.05, its test error is the second smallest. Generally, the linear SVM performs well on this dataset with the **best tuned cost value as 0.01/0.05**.

16-1. Radial SVM

Fit in Radial SVM and Tune Cost and Gamma parameters by 10-fold cross validation.

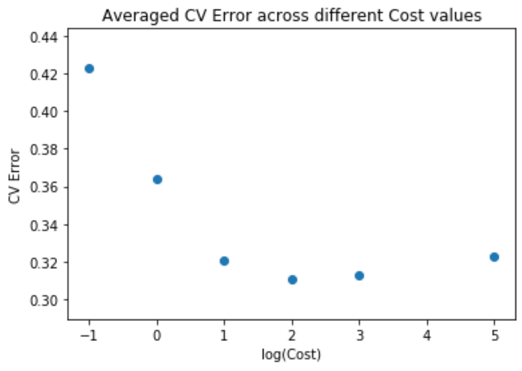
```
#10-fold cross validation
skf = StratifiedKFold(n_splits=10,random_state=4)

#define variable to save error rates
#train_error_allc=[]
cv_error_allc=[]
train_error_allc2=[]
test_error_allc=[]
auc_allc=[]

#tune cost parameters
cost=[0.1,1,10,100,1000,100000]
gamma=[0.00001,0.0001,0.001,0.01,0.1,1]
cg=[(c,g) for c in cost for g in gamma]

for i in range(len(cg)):
    svm_rbf=SVC(C=cg[i][0], kernel = 'rbf', gamma=cg[i][1], probability = True)  #build the model for each cost
    test_error=[]
    cv_error=[]
    for train_index, test_index in skf.split(np_gss_train_feature,np_gss_train_y):
        X_train10, X_test10 = np_gss_train_feature[train_index], np_gss_train_feature[test_index]
        y_train10, y_test10 = np_gss_train_y[train_index], np_gss_train_y[test_index]
        svm_rbf.fit(X_train10,y_train10)

        #calculate accuracy based on CV training samples
        #er_score=1-sklearn.metrics.accuracy_score(y_train10, svm_linear.predict(X_train10))
        #train_error.append(er_score)
        #calculate accuracy based on test(validation) samples
        er_score=1-sklearn.metrics.accuracy_score(y_test10, svm_rbf.predict(X_test10))
        cv_error.append(er_score)

    #train_error_allc.append(np.mean(train_error))
    cv_error_allc.append(np.mean(cv_error))
    #calculate test error
    svm_rbf.fit(np_gss_train_feature,np_gss_train_y)
    er_score=1-sklearn.metrics.accuracy_score(np_gss_test_y, svm_rbf.predict(np_gss_test_feature))
    test_error_allc.append(er_score)
    #calculate auc for test dataset
    auc_score=sklearn.metrics.roc_auc_score(np_gss_test_y, svm_linear.predict(np_gss_test_feature))
    auc_allc.append(auc_score)
    #er_score=1-sklearn.metrics.accuracy_score(np_gss_train_y, svm_rbf.predict(np_gss_train_feature))
    #train_error_allc2.append(er_score)
```

```
#aggregate the training and testing results
c=[c for (c,g) in cg]
g=[g for (c,g) in cg]
df_cost=pd.DataFrame({'CV_Error':cv_error_allc,'Test_Error':test_error_allc,'Test_AUC':auc_allc,'Cost':c,'Gamma':g},index=cg)
df_cost
```

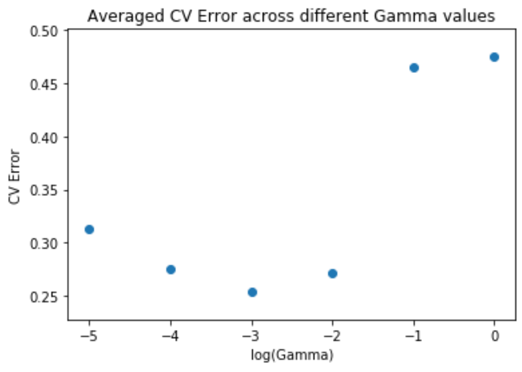
```
.dataframe tbody tr th {
    vertical-align: top;
}

.dataframe thead th {
    text-align: right;
}
```

	CV_Error	Test_Error	Test_AUC	Cost	Gamma
(0.1, 1e-05)	0.474678	0.462475	0.776018	0.1	0.00001
(0.1, 0.0001)	0.474678	0.462475	0.776018	0.1	0.00010
(0.1, 0.001)	0.293730	0.286004	0.776018	0.1	0.00100
(0.1, 0.01)	0.345790	0.310345	0.776018	0.1	0.01000
(0.1, 0.1)	0.474678	0.462475	0.776018	0.1	0.10000
(0.1, 1)	0.474678	0.462475	0.776018	0.1	1.00000
(1, 1e-05)	0.474678	0.462475	0.776018	1.0	0.00001
(1, 0.0001)	0.281549	0.255578	0.776018	1.0	0.00010
(1, 0.001)	0.235707	0.215010	0.776018	1.0	0.00100
(1, 0.01)	0.246490	0.261663	0.776018	1.0	0.01000
(1, 0.1)	0.469943	0.458418	0.776018	1.0	0.10000
(1, 1)	0.474678	0.462475	0.776018	1.0	1.00000
(10, 1e-05)	0.285553	0.247465	0.776018	10.0	0.00001
(10, 0.0001)	0.228270	0.208925	0.776018	10.0	0.00010
(10, 0.001)	0.214089	0.200811	0.776018	10.0	0.00100
(10, 0.01)	0.259356	0.271805	0.776018	10.0	0.01000
(10, 0.1)	0.461871	0.448276	0.776018	10.0	0.10000
(10, 1)	0.474678	0.462475	0.776018	10.0	1.00000
(100, 1e-05)	0.222832	0.204868	0.776018	100.0	0.00001
(100, 0.0001)	0.208643	0.198783	0.776018	100.0	0.00010
(100, 0.001)	0.235698	0.221095	0.776018	100.0	0.00100
(100, 0.01)	0.259356	0.271805	0.776018	100.0	0.01000
(100, 0.1)	0.461871	0.448276	0.776018	100.0	0.10000
(100, 1)	0.474678	0.462475	0.776018	100.0	1.00000
(1000, 1e-05)	0.205260	0.200811	0.776018	1000.0	0.00001
(1000, 0.0001)	0.205967	0.192698	0.776018	1000.0	0.00010
(1000, 0.001)	0.272189	0.249493	0.776018	1000.0	0.00100
(1000, 0.01)	0.259356	0.271805	0.776018	1000.0	0.01000
(1000, 0.1)	0.461871	0.448276	0.776018	1000.0	0.10000
(1000, 1)	0.474678	0.462475	0.776018	1000.0	1.00000
(100000, 1e-05)	0.212720	0.198783	0.776018	100000.0	0.00001
(100000, 0.0001)	0.254662	0.237323	0.776018	100000.0	0.00010
(100000, 0.001)	0.274212	0.239351	0.776018	100000.0	0.00100
(100000, 0.01)	0.259356	0.271805	0.776018	100000.0	0.01000
(100000, 0.1)	0.461871	0.448276	0.776018	100000.0	0.10000
(100000, 1)	0.474678	0.462475	0.776018	100000.0	1.00000

```python
#plot cross validation error for cost
import matplotlib.pyplot as plt
import math
cost0=[math.log(g,10) for g in cost]
y=df_cost.groupby('Cost').mean()['CV_Error']
y=y.values

plt.scatter(cost0,y);
plt.xlabel('log(Cost)');
plt.ylabel('CV Error');
plt.title('Averaged CV Error across different Cost values');
```



```
#plot cross validation error for Gamma
gamma0=[math.log(c,10) for c in gamma]
y=df_cost.groupby('Gamma').mean()['CV_Error']
y=y.values

plt.scatter(gamma0,y);
plt.xlabel('log(Gamma)');
plt.ylabel('CV Error');
plt.title('Averaged CV Error across different Gamma values');
```



```
#sor the model by smallest CV_Error
df_cost1=df_cost
df_cost1.sort_values(by = "CV_Error")[0:5]
```

```
.dataframe tbody tr th {
    vertical-align: top;
}

.dataframe thead th {
    text-align: right;
}
```

	CV_Error	Test_Error	Test_AUC	Cost	Gamma
(1000, 1e-05)	0.205260	0.200811	0.776018	1000.0	0.00001
(1000, 0.0001)	0.205967	0.192698	0.776018	1000.0	0.00010
(100, 0.0001)	0.208643	0.198783	0.776018	100.0	0.00010
(100000, 1e-05)	0.212720	0.198783	0.776018	100000.0	0.00001
(10, 0.001)	0.214089	0.200811	0.776018	10.0	0.00100

From the table and figure above, we could see that the smallest cv error is-0.20526 and test error-0.2008 for the radial kernel model, the corresponding gamma value is 0.00001 and cost value is 1000; while when gamma=0.0001 and cost value=1000, the **test error** is smallest,0.192698 (less than 0.2). Therefore **the best parameters for radial kernel SVM is gamma 0.0001/0.00001 and cost value=1000.**

Moreover, the best performance of the radial model is very close to the linear SVM, and it does not outperforms the linear model, which might indicate the 'linear' relationship of the high dimension features between classes; also considering the extreme small value of gamma, it is reasonable to infer that the non-linear boundary is very loose and broad and might close to a linear one.

16-2. Polynomial SVM

Fit in Polynomial SVM and Tune Cost, Degree and Gamma parameters by 10-fold cross validation.

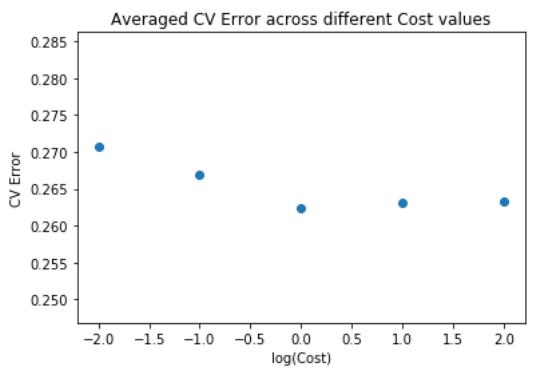
```python
#10-fold cross validation
from sklearn.model_selection import StratifiedKFold
skf = StratifiedKFold(n_splits=10,random_state=4)

#define variable to save error rates
#train_error_allc=[]
cv_error_allc=[]
train_error_allc2=[]
test_error_allc=[]
auc_allc=[]

#tune cost parameters
cost=[0.01,0.1,1,10,100]
#gamma=[1,10,100,1000]
degree=[1,3,5,7,9] #np.linspace(1,10,11)
#cgd=[(c,d,g) for c in cost for d in degree for g in gamma ]
cd=[(c,d) for c in cost for d in degree]

for i in range(len(cd)):
    svm_poly=SVC(C=cd[i][0], kernel = 'poly', degree=cd[i][1], gamma='auto',probability = True)  #build the model for each cost
    test_error=[]
    cv_error=[]
    for train_index, test_index in skf.split(np_gss_train_feature,np_gss_train_y):
        X_train10, X_test10 = np_gss_train_feature[train_index], np_gss_train_feature[test_index]
        y_train10, y_test10 = np_gss_train_y[train_index], np_gss_train_y[test_index]
        svm_poly.fit(X_train10,y_train10)

        #calculate accuracy based on CV training samples
        #er_score=1-sklearn.metrics.accuracy_score(y_train10, svm_linear.predict(X_train10))
        #train_error.append(er_score)
        #calculate accuracy based on test(validation) samples
        er_score=1-sklearn.metrics.accuracy_score(y_test10, svm_poly.predict(X_test10))
        cv_error.append(er_score)

    #train_error_allc.append(np.mean(train_error))
    cv_error_allc.append(np.mean(cv_error))
    #Because of very slow computation, for polynomial SVM
    #test error rate is not calculated
    #svm_poly.fit(np_gss_train_feature,np_gss_train_y)
    #er_score=1-sklearn.metrics.accuracy_score(np_gss_test_y, svm_poly.predict(np_gss_test_feature))
    #test_error_allc.append(er_score)
    #calculate auc for test dataset
    #auc_score=sklearn.metrics.roc_auc_score(np_gss_test_y, svm_poly.predict(np_gss_test_feature))
    #auc_allc.append(auc_score)
    #er_score=1-sklearn.metrics.accuracy_score(np_gss_train_y, svm_poly.predict(np_gss_train_feature))
    #train_error_allc2.append(er_score)
```

```python
#aggregate the training and testing results
c=[c for (c,d) in cd]
d=[d for (c,d) in cd]
df_cost=pd.DataFrame({'CV_Error':cv_error_allc,'Cost':c,'Degree':d},index=cd)
df_cost
```

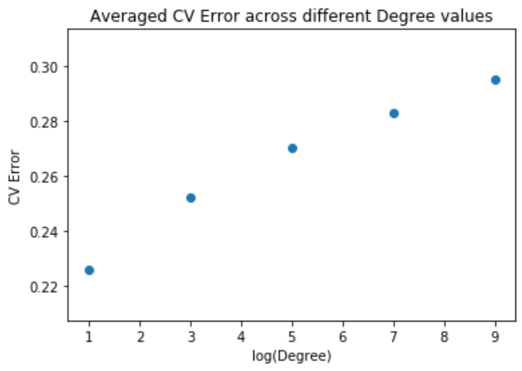
```css
.dataframe tbody tr th {
    vertical-align: top;
}

.dataframe thead th {
    text-align: right;
}
```

	CV_Error	Cost	Degree
(0.01, 1)	0.288242	0.01	1
(0.01, 3)	0.216829	0.01	3
(0.01, 5)	0.270158	0.01	5
(0.01, 7)	0.282978	0.01	7
(0.01, 9)	0.295072	0.01	9
(0.1, 1)	0.228909	0.10	1
(0.1, 3)	0.257310	0.10	3
(0.1, 5)	0.270158	0.10	5
(0.1, 7)	0.282978	0.10	7
(0.1, 9)	0.295072	0.10	9
(1, 1)	0.201890	1.00	1
(1, 3)	0.262054	1.00	3
(1, 5)	0.270158	1.00	5
(1, 7)	0.282978	1.00	7
(1, 9)	0.295072	1.00	9
(10, 1)	0.205278	10.00	1
(10, 3)	0.262054	10.00	3
(10, 5)	0.270158	10.00	5
(10, 7)	0.282978	10.00	7
(10, 9)	0.295072	10.00	9
(100, 1)	0.205963	100.00	1
(100, 3)	0.262054	100.00	3
(100, 5)	0.270158	100.00	5
(100, 7)	0.282978	100.00	7
(100, 9)	0.295072	100.00	9

```
#plot cross validation error for cost
#cost=[0.1,1,10,100]
cost0=[math.log(c,10) for c in cost]
y=df_cost.groupby('Cost').mean()['CV_Error']
y=y.values

plt.scatter(cost0,y);
plt.xlabel('log(Cost)');
plt.ylabel('CV Error');
plt.title('Averaged CV Error across different Cost values');
```



```
#plot cross validation error for Degree
#cost=[0.1,1,10,100]
#d0=[math.log(d,10) for d in degree]
y=df_cost.groupby('Degree').mean()['CV_Error']
y=y.values

plt.scatter(degree,y);
plt.xlabel('log(Degree)');
plt.ylabel('CV Error');
plt.title('Averaged CV Error across different Degree values');
```



```
#sor the model by smallest CV_Error
df_cost1=df_cost
df_cost1.sort_values(by = "CV_Error")[0:5]
```

```
.dataframe tbody tr th {
    vertical-align: top;
}

.dataframe thead th {
    text-align: right;
}
```

	CV_Error	Cost	Degree
(1, 1)	0.201890	1.00	1
(10, 1)	0.205278	10.00	1
(100, 1)	0.205963	100.00	1
(0.01, 3)	0.216829	0.01	3
(0.1, 1)	0.228909	0.10	1

For the polynomial SVM model, from parameters tuning, **when cost=1, degree=1, the model has the smallest cv error-0.2018, which is also very close to the performance of linear and radial kernel SVM**. Moreover, from the graph above we could clearly see that, the degree of best fit for this polynomial model is 1, which indicates the potentail **'linear' relationship between classes of the high-dimension features**; and maybe that is why the linear, radial kernel and polynomial modeld all generate good and close performance on this dataset.

However, even the degree of polynomial SVM is equal to 1, the best cost value is differnt from the linear SVM (1 rather than 0.5), but they are very close, which means a moderate level of penalty in linear model could already yield good performance in fitting; moreover, for radial kernel SVM, the cost value for the best model is quite large to lower bias, and the gamma is very small-indicating a broad non-linear separation, which further validates the guess that **there is a linear separation in the dataset that linear model could easily catch, while it need more penalty and fined parameters tuning for radial kernel SVM-non linear model to fit**.