Leonardo Gomez

CS 596

H6: Support Vector Machine

**Introduction:**

Support Vector Machine or SVM for short is a supervised learning model with associated learning algorithm that analyze data used for classification and regression analysis. An SVM model is a representation of the examples as points in space, mapped so that the examples of the separate categories are divided by a clear gap that is as wide as possible. New examples are then mapped into that same space and predicted to belong to a category based on which side of the gap they fall. However, some of the disadvantages is that the number of features is much greater than the number of samples, avoid over-fitting in choosing Kernel functions and regularization term is crucial and SVMs do not directly provide probability estimates, these are calculated using an expensive five-fold cross-validation.

**Results:**

Best kernel: linear c = 5.0

Confusion Matrix:

[[50 2]

[ 1 47]]

Average Accuracy: 0.97

Per-Class Precision: [0.98039216 0.95918367]

Per-Class Recall: [0.96153846 0.97916667]

\*\*\*\*\*5 Failures\*\*\*\*\*

Prediction: -1.0 Ground-truth: 1.0

Features: [ 1. 9.1 8.2 19.2 22.2 7.7]

Prediction: 1.0 Ground-truth: -1.0

Features: [ 0. 12.5 9.4 23.2 26. 10.8]

Prediction: -1.0 Ground-truth: 1.0

Features: [ 1. 10.8 9.5 22.5 26.3 9.1]

\*\*\*\*\*5 Successes\*\*\*\*\*

Correct Prediction: -1.0

Features: [ 0. 11.4 9. 22.7 24.8 10.1]

Correct Prediction: 1.0

Features: [ 1. 12.8 12.2 26.7 31.1 11.1]

Correct Prediction: 1.0

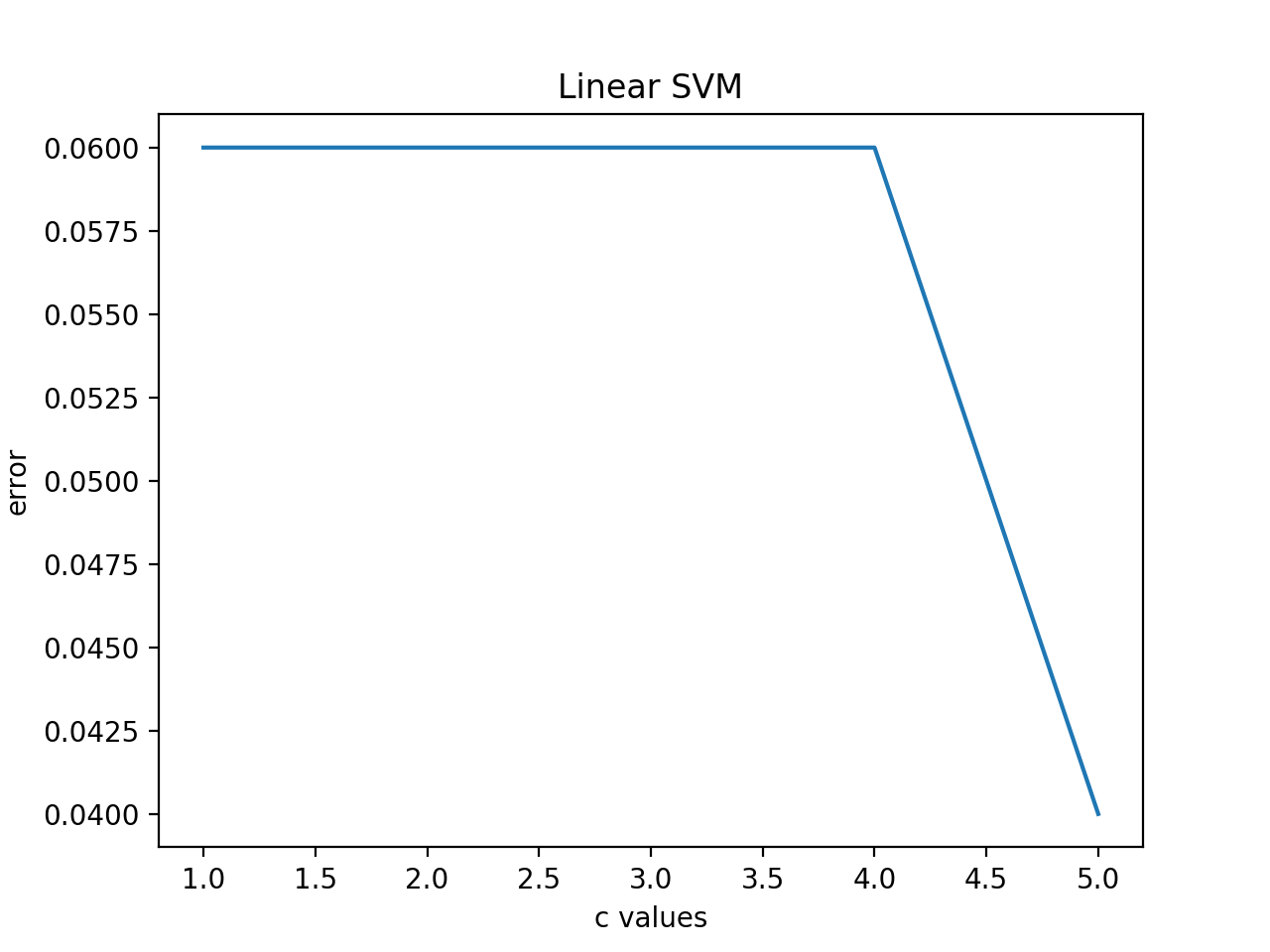
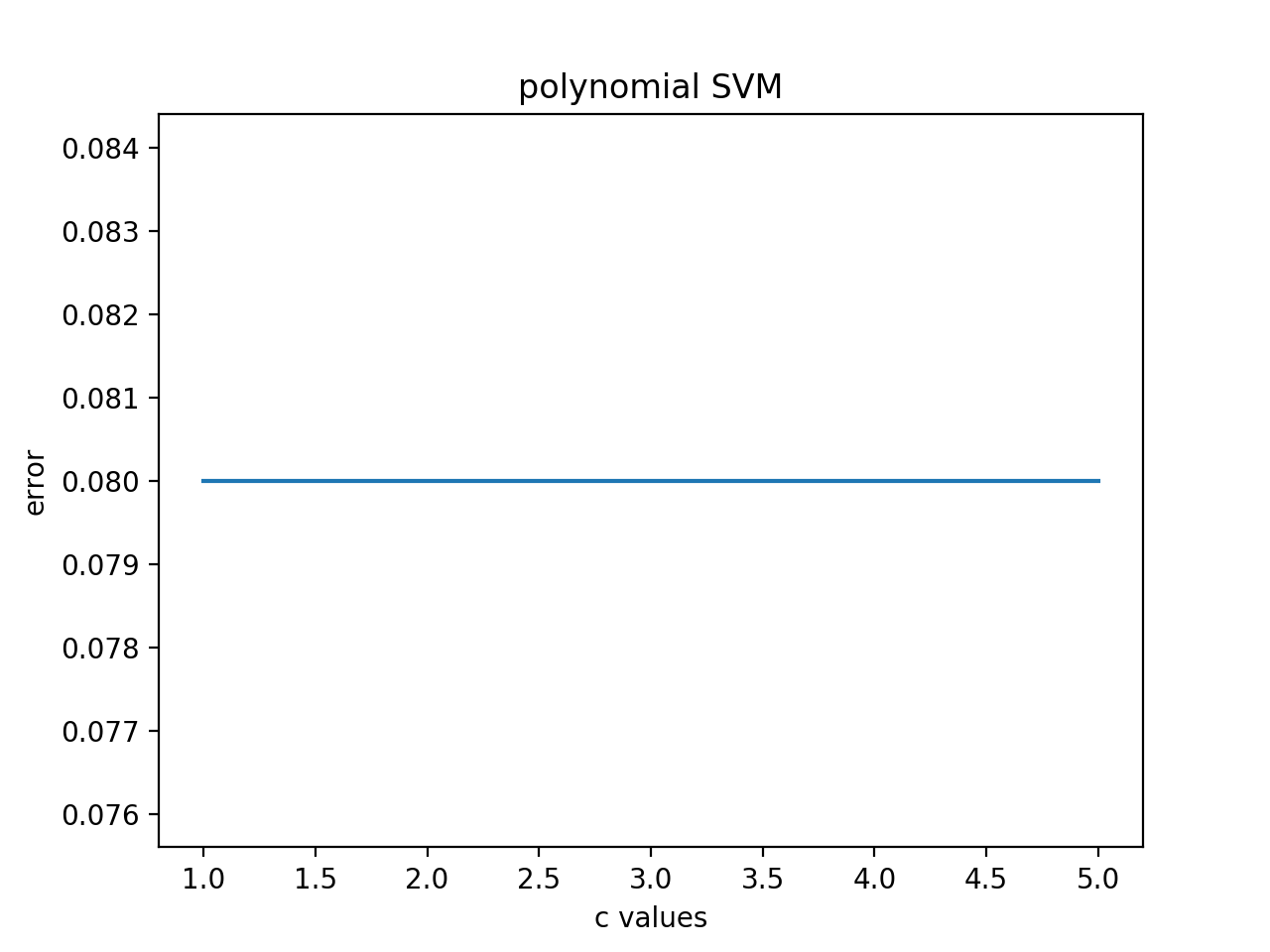
Features: [ 0. 16.2 14. 31.6 35.6 13.7]

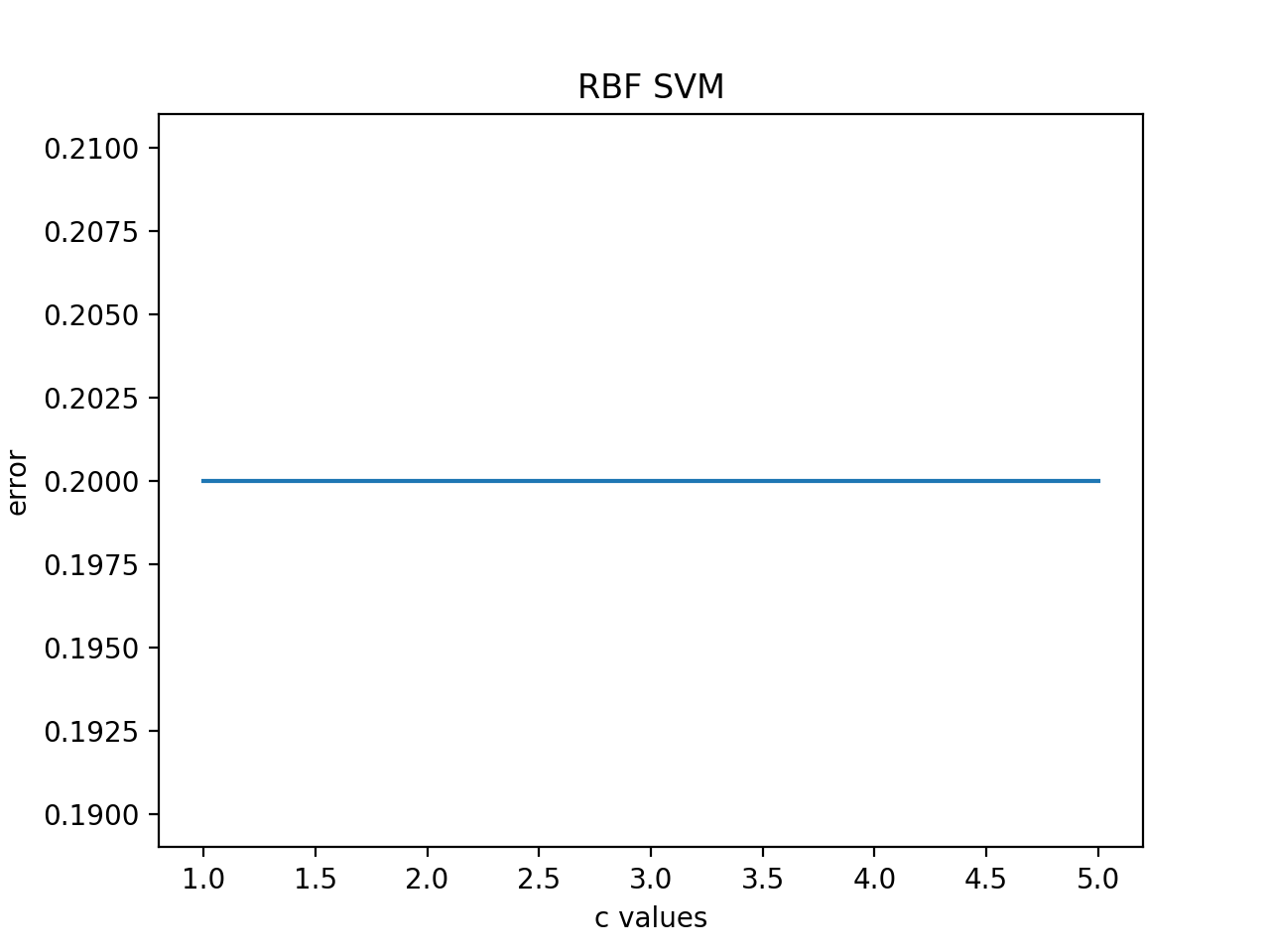
Correct Prediction: 1.0

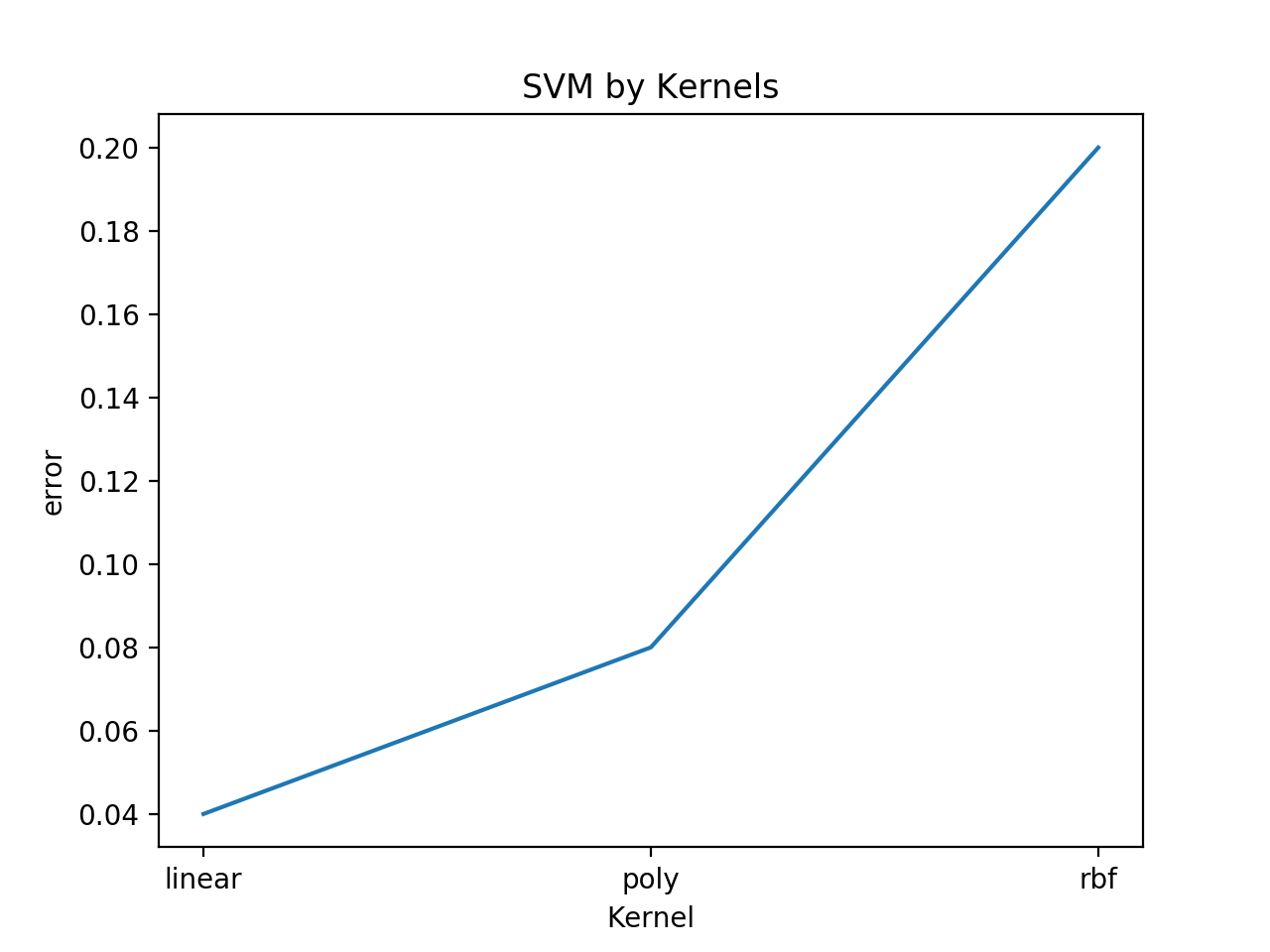
Features: [ 1. 9.1 8.1 18.5 21.6 7.7]

Correct Prediction: -1.0

Features: [ 1. 10.8 9. 23. 26.5 9.8]







**Code:**

import numpy as np  
import download\_data as dl  
import matplotlib.pyplot as plt  
import sklearn.svm as svm  
from sklearn import metrics  
from conf\_matrix import func\_confusion\_matrix  
  
#CS 596, machine learning  
  
## step 1: load data from csv file.   
data = dl.download\_data('crab.csv').values  
  
n = 200  
#split data   
S = np.random.permutation(n)  
#100 training samples  
Xtr = data[S[:100], :6]  
Ytr = data[S[:100], 6:].ravel()  
# 100 testing samples  
X\_test = data[S[100:], :6]  
Y\_test = data[S[100:], 6:].ravel()  
  
## step 2 randomly split Xtr/Ytr into two even subsets: use one for training, another for validation.  
#############placeholder: training/validation #######################  
n2 = len(Xtr)  
S2 = np.random.permutation(n2)  
  
# subsets for training models  
x\_train= Xtr[S2[:50],:]  
y\_train= Ytr[S2[:50]]  
# subsets for validation  
x\_validation= Xtr[S2[50:],:]  
y\_validation= Ytr[S2[50:]]  
#############placeholder #######################  
  
## step 3 Model selection over validation set  
# consider the parameters C, kernel types (linear, RBF etc.) and kernel  
# parameters if applicable.   
  
  
# 3.1 Plot the validation errors while using different values of C ( with other hyperparameters fixed)   
# keeping kernel = "linear"  
#############placeholder: Figure 1#######################  
c\_parameters = []  
c\_range = np.arange(1.0,6.0,1.0)  
svm\_c\_error = []  
for c\_value in c\_range:  
 model = svm.SVC(kernel='linear', C=c\_value)  
 model.fit(X=x\_train, y=y\_train)  
 error = 1. - model.score(x\_validation, y\_validation)  
 svm\_c\_error.append(error)  
plt.plot(c\_range, svm\_c\_error)  
plt.title('Linear SVM')  
plt.xlabel('c values')  
plt.ylabel('error')  
#plt.xticks(c\_range)  
plt.show()  
  
index = np.argmin(svm\_c\_error)  
c\_parameters.append(c\_range[index])  
  
svm\_c\_error = []  
for c\_value in c\_range:  
 model = svm.SVC(kernel='poly', C=c\_value)  
 model.fit(X=x\_train, y=y\_train)  
 error = 1. - model.score(x\_validation, y\_validation)  
 svm\_c\_error.append(error)  
plt.plot(c\_range, svm\_c\_error)  
plt.title('polynomial SVM')  
plt.xlabel('c values')  
plt.ylabel('error')  
#plt.xticks(c\_range)  
plt.show()  
  
index = np.argmin(svm\_c\_error)  
c\_parameters.append(c\_range[index])  
  
svm\_c\_error = []  
for c\_value in c\_range:  
 model = svm.SVC(kernel='rbf', C=c\_value)  
 model.fit(X=x\_train, y=y\_train)  
 error = 1. - model.score(x\_validation, y\_validation)  
 svm\_c\_error.append(error)  
plt.plot(c\_range, svm\_c\_error)  
plt.title('RBF SVM')  
plt.xlabel('c values')  
plt.ylabel('error')  
#plt.xticks(c\_range)  
plt.show()  
  
index = np.argmin(svm\_c\_error)  
c\_parameters.append(c\_range[index])  
#############placeholder #######################  
  
  
# 3.2 Plot the validation errors while using linear, RBF kernel, or Polynomial kernel ( with other hyperparameters fixed)   
#############placeholder: Figure 2#######################  
  
kernel\_types = ['linear', 'poly', 'rbf']  
svm\_kernel\_error = []  
x = 0  
for kernel\_value in kernel\_types:  
 # your own codes  
 model = svm.SVC(kernel = kernel\_value, C =c\_parameters[x])  
 model.fit(X=x\_train, y=y\_train)  
 error = 1. - model.score(x\_validation, y\_validation)  
 svm\_kernel\_error.append(error)  
 x +=1  
  
plt.plot(kernel\_types, svm\_kernel\_error)  
plt.title('SVM by Kernels')  
plt.xlabel('Kernel')  
plt.ylabel('error')  
plt.xticks(kernel\_types)  
plt.show()  
  
best = np.argmin(svm\_kernel\_error)  
  
## step 4 Select the best model and apply it over the testing subset   
best\_kernel = kernel\_types[best]  
best\_c = c\_parameters[best] # poly had many that were the "best"  
model = svm.SVC(kernel=best\_kernel, C=best\_c)  
model.fit(X=x\_train, y=y\_train)  
## step 5 evaluate your results with the metrics you have developed in HA3,including accuracy, quantize your results.   
  
  
y\_pred = model.predict(X\_test)  
conf\_matrix, accuracy, recall\_array, precision\_array = func\_confusion\_matrix(Y\_test, y\_pred)  
print("Best kernel: {} c = {}".format(best\_kernel,best\_c))  
print("Confusion Matrix: ")  
print(conf\_matrix)  
print("Average Accuracy: {}".format(accuracy))  
print("Per-Class Precision: {}".format(precision\_array))  
print("Per-Class Recall: {}\n".format(recall\_array))  
  
success = (y\_pred == Y\_test)  
counter = 0  
print("\*\*\*\*\*5 Failures\*\*\*\*\*")  
for x in range(len(success)):  
 if(not(success[x])):  
 counter+=1  
 print("Prediction: {} Ground-truth: {}".format(y\_pred[x],Y\_test[x]))  
 print("Features: {}\n".format(X\_test[x]))  
 if (counter == 5):  
 break  
counter = 0  
print("\*\*\*\*\*5 Successes\*\*\*\*\*")  
for x in range(len(success)):  
 if(success[x]):  
 counter+=1  
 print("Correct Prediction: {}".format(y\_pred[x]))  
 print("Features: {}\n".format(X\_test[x]))  
 if (counter == 5):  
 break