Gaze-based Autism Detection Using Videos

Importing libraries

In [1]:
```
# run this code

import numpy as np
import random
from sklearn.model_selection import KFold, cross_val_score
from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import SVC
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import *
from sklearn.preprocessing import Normalizer
from sklearn.metrics import confusion_matrix
from sklearn.metrics import classification_report, accuracy_score
import seaborn as sns
import matplotlib.pyplot as plt
from matplotlib.colors import ListedColormap
%matplotlib inline

#Note: these includes are just a guide. You are allowed to use anything
#from the packages outlined in "Environment.pdf" under Assignment 1 on Ca
```

Task 1: Data loading

In the **next** cell create methods to load the sensor data from numpy files.

In [2]:
```
#run this code

root = '/Users/gpanirudh/Downloads/Academics/UTK/Semester-1/ML/Assignment

def load_raw_data_autistic():
    test_X1= np.load(f'{root}vid_1_gaze_test_X.npy')
    test_Y1= np.load(f'{root}vid_1_gaze_test_Y.npy')

    test_X2= np.load(f'{root}vid_2_gaze_test_X.npy')
    test_Y2= np.load(f'{root}vid_2_gaze_test_Y.npy')

    #You can load more videos if you would like here (optional extra cred

    test_X3 = np.load(f'{root}vid_5_gaze_test_X.npy')
    test_Y3 = np.load(f'{root}vid_5_gaze_test_Y.npy')

    test_X4 = np.load(f'{root}vid_7_gaze_test_X.npy')
    test_Y4 = np.load(f'{root}vid_7_gaze_test_Y.npy')

    return (test_X1, test_X2, test_X3, test_X4), (test_Y1, test_Y2, test_

def load_raw_data_non_autistic():
    gaze_X1= np.load(f'{root}vid_1_gaze_GT_X.npy')
    gaze_Y1= np.load(f'{root}vid_1_gaze_GT_Y.npy')

    gaze_X2= np.load(f'{root}vid_2_gaze_GT_X.npy')
    gaze_Y2= np.load(f'{root}vid_2_gaze_GT_Y.npy')

    #You can load more videos if you would like here (optional extra cred

    gaze_X3= np.load(f'{root}vid_5_gaze_GT_X.npy')
    gaze_Y3= np.load(f'{root}vid_5_gaze_GT_Y.npy')

    gaze_X4= np.load(f'{root}vid_7_gaze_GT_X.npy')
    gaze_Y4= np.load(f'{root}vid_7_gaze_GT_Y.npy')

    return (gaze_X1, gaze_X2, gaze_X3, gaze_X4), (gaze_Y1, gaze_Y2, gaze_

def load_data_autistic():
    tuple_X, tuple_Y = load_raw_data_autistic()
    test_points_X = np.concatenate(tuple_X, axis=1)
    test_points_Y = np.concatenate(tuple_Y, axis=1)

    return test_points_X, test_points_Y

def load_data_non_autistic():
    tuple_X, tuple_Y = load_raw_data_non_autistic()
    gaze_GT_X=np.concatenate(tuple_X, axis=1)
    gaze_GT_Y=np.concatenate(tuple_Y, axis=1)

    return gaze_GT_X, gaze_GT_Y
```

In [3]:
```
#printing out our data...

autistic_X, autistic_Y = load_data_autistic()
control_X, control_Y = load_data_non_autistic()

print(autistic_X.shape, autistic_Y.shape)
print(control_X.shape, control_Y.shape)
```

```
(35, 2332) (35, 2332)
(25, 2332) (25, 2332)
```

Did you use additional files? (Yes or No) <Using additional files will give you 5 extra credit>

Yes

Task 2: choose wisely your features.

Add features that allow you to reliably identify users with autism symptoms.

In [4]:
```
# Featurize each subject's gaze trajectory in the file
# This method is run once for each video file for each condition
def featurize_input(X, Y):
    out = []
    # Where i is each subject in the file
    for i in range(len(X)):
        X_cord = X[i]
        Y_cord = Y[i] #NOT USED (but you can add Y features if you like)
        fv = []

        # ADD CODE HERE #
        #add your features to fv (each feature here should be a single nu
        #you should replace the appends below with better features (mean,
        #fv.append(random.random())# use to test noise.. gives poor accur
        fv.append(X_cord[0]) #initial X position
        fv.append(X_cord[-1]) #final X position
        fv.append(Y_cord[0]) #initial Y position
        fv.append(Y_cord[-1]) #final Y position
        fv.append(min(X_cord))
        fv.append(max(X_cord))
        fv.append(min(Y_cord))
        fv.append(max(Y_cord))
        fv.append(np.median(X_cord))
        fv.append(np.median(Y_cord))

        out.append(fv)

    out = np.array(out)

    return out
```

In [5]:
```
# run this code

def load_data_autistic_fv():
    tuple_X, tuple_Y = load_raw_data_autistic()

    fv_list = []
    for X, Y in list(zip(tuple_X, tuple_Y)):
      fv_list.append(featurize_input(X, Y))

    fv = np.concatenate(tuple(fv_list), axis=1)

    return fv

def load_data_non_autistic_fv():
    tuple_X, tuple_Y = load_raw_data_non_autistic()

    fv_list = []
    for X, Y in list(zip(tuple_X, tuple_Y)):
      fv_list.append(featurize_input(X, Y))

    fv = np.concatenate(tuple(fv_list), axis=1)

    return fv
```

In [6]:
```
# run this code

# get feature engineered vectors
autistic_fv = load_data_autistic_fv()
control_fv = load_data_non_autistic_fv()

print(autistic_fv.shape, control_fv.shape)
```

```
(35, 40) (25, 40)
```

Question: Is there any problem since the number of autistic and control participants is not the same? Answer in the next cell. Justify your answer.

Answer:

I feel that due to the number of autistic and control participants not being the same, it might lead to an imbalance in the dataset. Due to this imbalance, we don't know how accurate our prediction will be. The prediction model will mostly be biased towards the majority class. Hence there is a need to balance the dataset.

I feel then when the difference is relatively small, there should not be much of an issue.

Task 3: balance the datasets.

Balance your datasets. Explain your method and **comment** about your decision.

Explanation:

I have included the code for balancing the datasets in **Task 6: Examine the effect of feature engineering**. I have used SMOTE Technique for balancing the dataset.

Since there is an imbalance in the dataset, this will become a blocker during training and hence leads to biased results if not resolved. Hence, I have used SMOTE, which a very popular oversampling technique. It creates minority data points in the class of minority samples.

Task 4: dealing with ground truth.

Add the ground truth (labels) to the data.

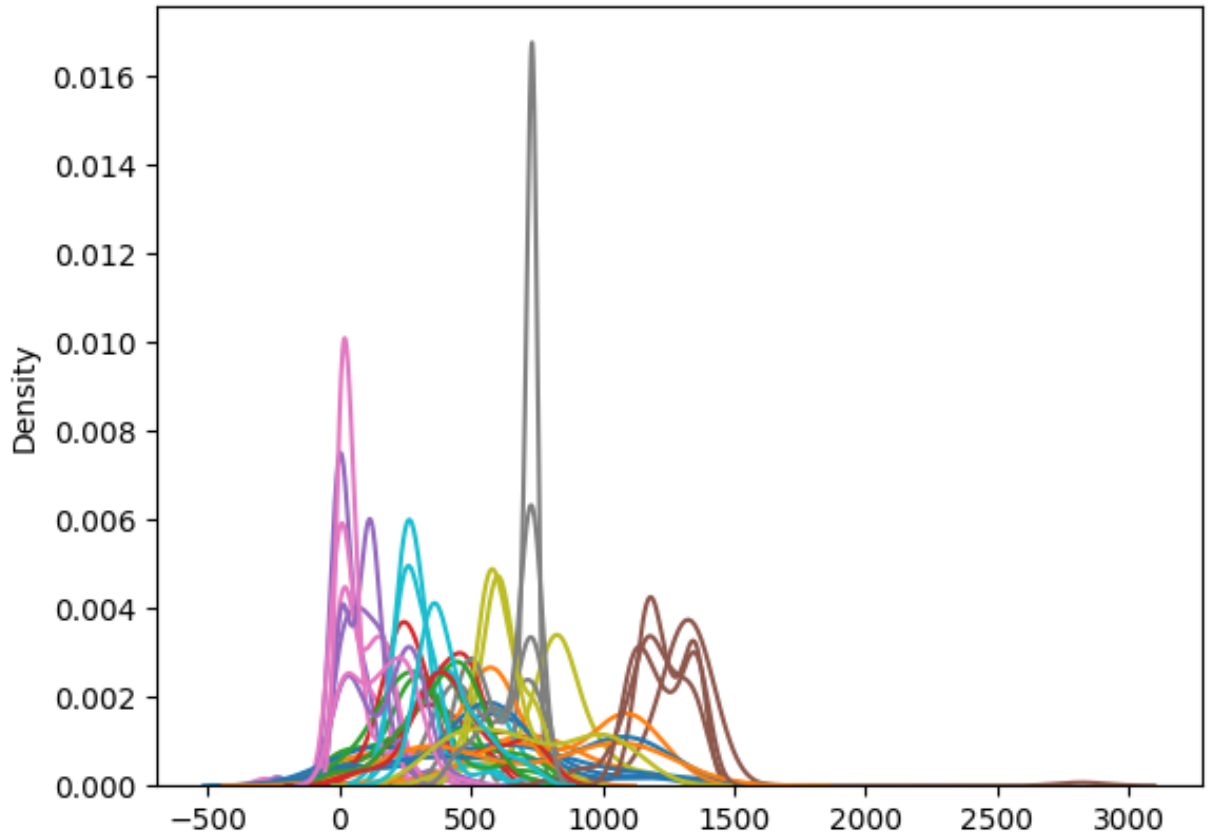
In [7]:
```
# run this code

#Assigning groundtruth conditions to each participant.
labels_aut = [1.0] * len(autistic_fv)
labels_control = [0.0] * len(control_fv)

#Make data and labels vectors (hint: np.concatenate)...
data = np.concatenate((autistic_fv, control_fv))
labels = np.concatenate((labels_aut, labels_control))

###SANITY CHECK###
print(data.shape) # data (expected output: (60, #) ) -- Note: your y-dim
print(labels.shape) # labels (exptected output: (60,) )

#NOTE: If you chose to rebalance your data (task 3) then you may have mor
```

```
(60, 40)
(60,)
```

Task 5: dealing with the spread of the features.

To know if we need to somehow normalize the data, it is useful to plot the spread of our features across the dataset. Write code to visualize the spread of our data (assuming that our data is contained in the variable 'X').

In [8]:
```
# run this code

for i in range(data.shape[1]):
    sns.kdeplot(data[:,i])
```



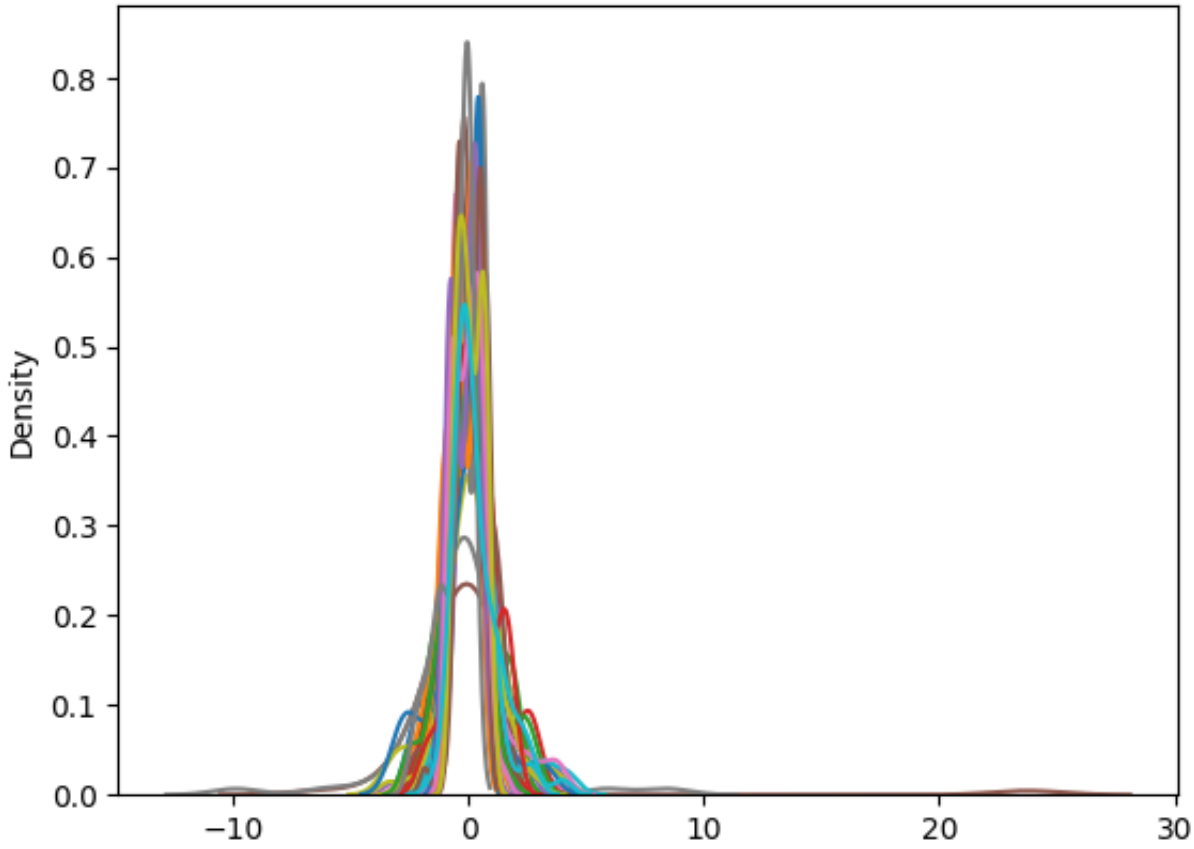
Question: What the previous plot tells us? Do we need to normalize our data? Answer in the next cell. Justify your answer.

Answer:

The plot shows that the range of data values are non-uniform. Hence, there is a need to normalize data. What it basically does is transforms data in a way that they have either similar distributions, or, make them dimensionless. It improves the accuracy by giving equal importance (weights) to each variable so that variables that have bigger numbers have a heavy influence on the result.

No matter what your answer was in the previous question, use one technique to normalize the data.

In [9]:
```
# run this code

scaler = RobustScaler()
data = scaler.fit_transform(data)
for i in range(data.shape[1]):
    sns.kdeplot(data[:,i])
```



Question: Why did you choose this normalization technique? What is it doing? Answer in the next cell.

Answer:

We have used this robust scaler technique because it is robust to the outliers since it uses the interquartile range. Robust scaler handles outliers in a better manner.

Task 6: Examine the effect of feature engineering

In this task you will run a machine learning classifier (SVM) on different combinations of features that you created in task 2.

Briefly explain your best model's features and why you think they worked well.

Keep in mind that for 100% credit you want at least one configuration (of feature engineering -- task 2) that provides above 95% accuracy when run through the SVM code below. Otherwise there will be a penalty of 10 points.

In [10]:
```
conda install -c conda-forge imbalanced-learn
```

```
Collecting package metadata (current_repodata.json): done
Solving environment: done


==> WARNING: A newer version of conda exists. <==
  current version: 4.12.0
  latest version: 22.9.0

Please update conda by running

    $ conda update -n base -c defaults conda



# All requested packages already installed.


Note: you may need to restart the kernel to use updated packages.
```

In [11]:
```
# run this code

xtrain, xtest, ytrain, ytest = train_test_split(data, labels, test_size=0

from imblearn.over_sampling import SMOTE
sm = SMOTE(random_state=42)
xtrain_res, ytrain_res = sm.fit_resample(xtrain, ytrain)

#training the model
clf = SVC() #note the default kernel here is 'rbf' - radial basis functio
clf.fit(xtrain_res, ytrain_res)
cv_scores = cross_val_score(clf, xtrain_res, ytrain_res, cv=10)
print('Average Cross Validation Score from Training:', cv_scores.mean(),

#testing the model
ypred = clf.predict(xtest)
cm = confusion_matrix(ytest, ypred)
cr = classification_report(ytest, ypred)

print('Confusion Matrix:', cm, sep='\n', end='\n\n\n')
print('Test Statistics:', cr, sep='\n', end='\n\n\n')

#This is what we will be grading (>95 expected)
print('Testing Accuracy:', accuracy_score(ytest, ypred))
```

```
Average Cross Validation Score from Training:
0.96


Confusion Matrix:
[[ 6  0]
 [ 0 12]]


Test Statistics:
              precision    recall  f1-score   support

         0.0       1.00      1.00      1.00         6
         1.0       1.00      1.00      1.00        12

    accuracy                           1.00        18
   macro avg       1.00      1.00      1.00        18
weighted avg       1.00      1.00      1.00        18



Testing Accuracy: 1.0
```

Explain your model performance:

Tha above classifier implements the default version of SVM, which is the radial basis function. The accurcay obtained using this was 1.0 (100%). Using the 10-fold cross validation, we got 0.96 as our average cross validation score from training.

We have applied SMOTE technique on the training dataset before using it for training the SVM classifier for the different kernels implemented.

Task 7: Modifying SVM

SVM allows for different kernels as you learned in class. Modify the code from task 6 to implement two other SVM kernels.

Briefly explain the differences you see between different kernels.

In [12]:
```
# code for classifier 1


#training the model
clf = SVC(kernel = 'linear') #linear kernel
clf.fit(xtrain_res, ytrain_res)
cv_scores = cross_val_score(clf, xtrain_res, ytrain_res, cv=10)
print('Average Cross Validation Score from Training:', cv_scores.mean(),

#testing the model
ypred = clf.predict(xtest)
cm = confusion_matrix(ytest, ypred)
cr = classification_report(ytest, ypred)

print('Confusion Matrix:', cm, sep='\n', end='\n\n\n')
print('Test Statistics:', cr, sep='\n', end='\n\n\n')

#This is what we will be grading (>95 expected)
print('Testing Accuracy:', accuracy_score(ytest, ypred))
```

```
Average Cross Validation Score from Training:
0.915


Confusion Matrix:
[[ 6  0]
 [ 1 11]]


Test Statistics:
              precision    recall  f1-score   support

         0.0       0.86      1.00      0.92         6
         1.0       1.00      0.92      0.96        12

    accuracy                           0.94        18
   macro avg       0.93      0.96      0.94        18
weighted avg       0.95      0.94      0.95        18



Testing Accuracy: 0.9444444444444444
```

Tha above classifier implements the linear kernel of SVM. The accurcay obtained using this was 0.944 (94.4%). Using the 10-fold cross validation, we got 0.915 as our average cross validation score from training.

In [13]:
```
# code for classifier 2

#training the model
clf = SVC(kernel = 'sigmoid') #sigmoid kernel
clf.fit(xtrain_res, ytrain_res)
cv_scores = cross_val_score(clf, xtrain_res, ytrain_res, cv=10)
print('Average Cross Validation Score from Training:', cv_scores.mean(),

#testing the model
ypred = clf.predict(xtest)
cm = confusion_matrix(ytest, ypred)
cr = classification_report(ytest, ypred)

print('Confusion Matrix:', cm, sep='\n', end='\n\n\n')
print('Test Statistics:', cr, sep='\n', end='\n\n\n')

#This is what we will be grading (>95 expected)
print('Testing Accuracy:', accuracy_score(ytest, ypred))
```

```
Average Cross Validation Score from Training:
0.865


Confusion Matrix:
[[ 6  0]
 [ 1 11]]


Test Statistics:
              precision    recall  f1-score   support

         0.0       0.86      1.00      0.92         6
         1.0       1.00      0.92      0.96        12

    accuracy                           0.94        18
   macro avg       0.93      0.96      0.94        18
weighted avg       0.95      0.94      0.95        18



Testing Accuracy: 0.9444444444444444
```

Explanation:

Tha above classifier implements the sigmoid kernel of SVM. The accurcay obtained using this was 0.944 (94.4%). Using the 10-fold cross validation, we got 0.865 as our average cross validation score from training.

The default SVM kernel, that is, the radial basis function got the highest accuracy, 1.0(100%) and highest average cross validation score, that is, 0.96.

Both the linear and sigmoid kernels have achieved the same testing accuracy, that is, 0.944 (94.4%). They have a different cross validation score though. The average cross validation score is higher for the linear kernel is 0.915, whereas for sigmoid, it is 0.865.

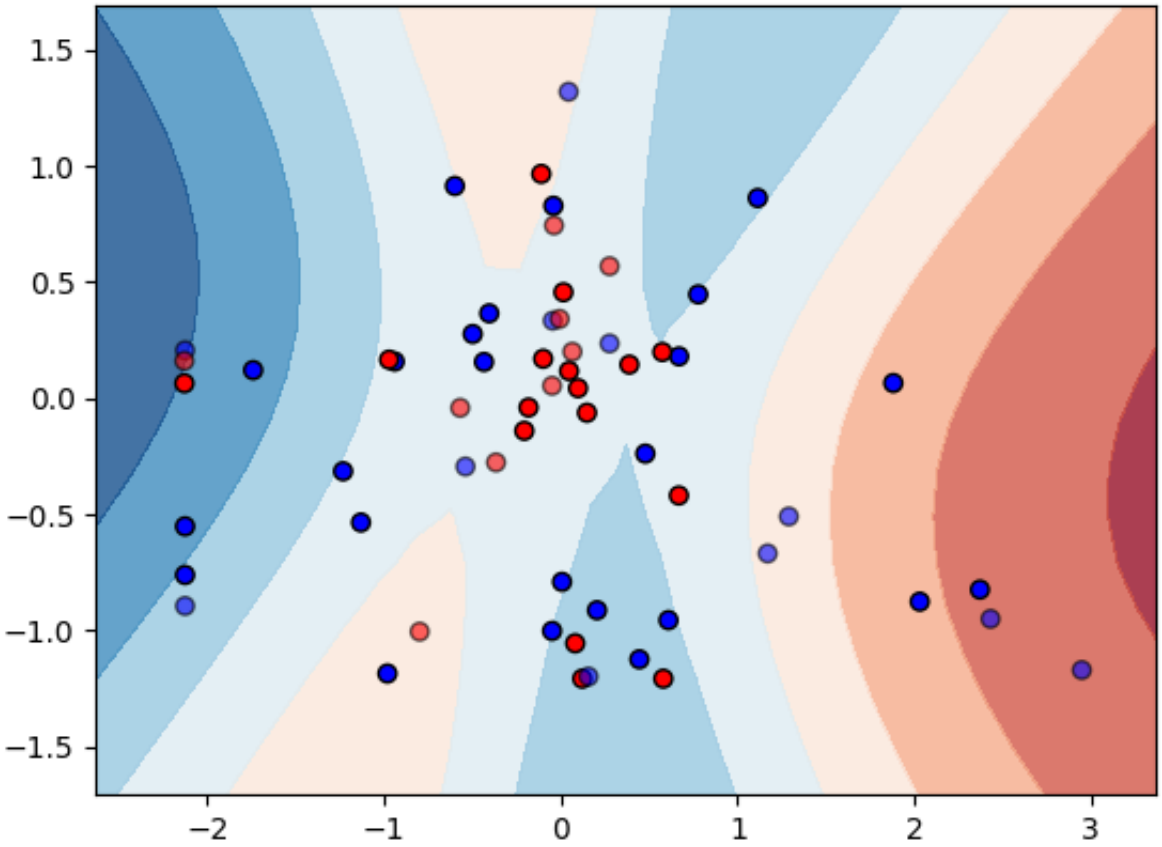
Looking at decision boundaries and spread of data

This is for exploration and will not be graded

In [14]:
```
#Give index of features that you are interested in looking at
sub_features = [0,1]

scores = []
cv = KFold(n_splits=3, random_state=42, shuffle=True)
for train_index, test_index in cv.split(data):
    #print("Train Index: ", train_index)
    #print("Test Index: ", test_index)
    X_train, X_test, y_train, y_test = data[train_index], data[test_index
```

In [15]:
```
h = .2  # step size in the mesh
x_min, x_max = data[:, 2].min() - .5, data[:, 2].max() + .5
y_min, y_max = data[:, 3].min() - .5, data[:, 3].max() + .5
xx, yy = np.meshgrid(np.arange(x_min, x_max, h),
                     np.arange(y_min, y_max, h))
h = .2  # step size in the mesh
x_min, x_max = data[:, sub_features[0]].min() - .5, data[:, sub_features[
y_min, y_max = data[:, sub_features[1]].min() - .5, data[:, sub_features[
xx, yy = np.meshgrid(np.arange(x_min, x_max, h),
                     np.arange(y_min, y_max, h))

cm = plt.cm.RdBu
cm_bright = ListedColormap(['#FF0000', '#0000FF'])

# Plot the training points
clf.fit(X_train[:,sub_features],y_train)
scores.append(clf.score(X_test[:,sub_features], y_test))
print('Mean Accuracy : ',str(np.mean(scores)))
print('This accuracy number is not used for evaluating your solution as i
if hasattr(clf, "decision_function"):
        Z = clf.decision_function(np.c_[xx.ravel(), yy.ravel()])
else:
        Z = clf.predict_proba(np.c_[xx.ravel(), yy.ravel()])[:, 1]
fig = plt.figure()

Z = Z.reshape(xx.shape)
plt.xlim(xx.min(), xx.max())
plt.ylim(yy.min(), yy.max())
plt.contourf(xx, yy, Z, cmap=cm, alpha=.8)

plt.scatter(X_train[:, sub_features[0]], X_train[:, sub_features[1]], c=y
plt.scatter(X_test[:, sub_features[0]], X_test[:, sub_features[1]], c=y_t


plt.show()
```

```
Mean Accuracy :  0.35
This accuracy number is not used for evaluating your solution as it only
looks at a subset of features.
```



Grading Strategy:

- Task1 0% (written for you)
- Task2 20%
- Task3 20%
- Task4 0% (written for you)
- Task5 20%
- Task6 25% (explanation)
- Task7 15% (5% per kernel, 5% explanation)

  There will be a -10 penalty if none of the classifiers you create (from tasks 6 and 7)
  have an accuracy of ~95.

  You can get +5 extra credit by modifying task 1 to use additional files (please indicate that you
  use additional files, there is a yes/no comment that you have to write in Task 1 to indicate that).

In [ ]: