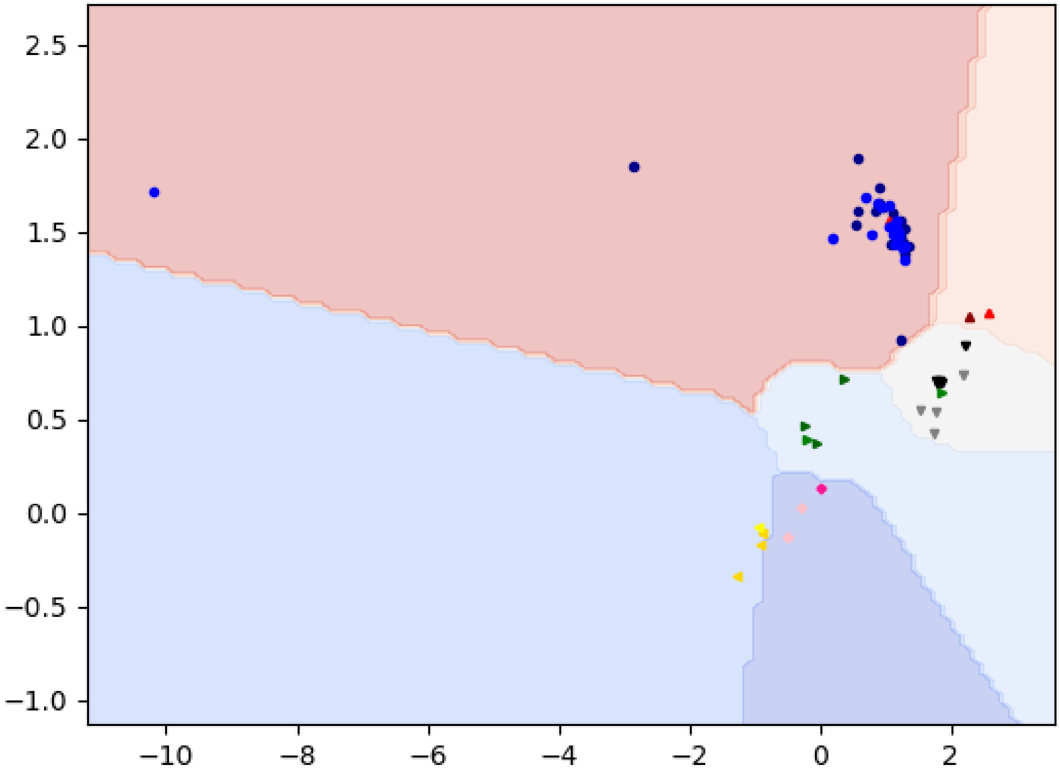
Trombini_Quentin_Week15

S15_01_a

```
from sklearn.model_selection import train_test_split
import sklearn.neural_network as nn
import sklearn.inspection as inspect
import numpy as np
import matplotlib.figure as figure
import matplotlib.backends.backend_agg as agg

with open("./S15/main.dat") as f:
    main_lines = f.readlines()
with open("./S15/photo.dat") as f:
    photo_lines = f.readlines()

dataset = []
improper = 0
for mline, pline in zip(main_lines,photo_lines):
    msplit = mline.split("|")
    psplit = pline.split("|")

    #r ID, U,B,V,R,I,J,H,K, absolute magnitude,spectral type
    useful = [msplit[0], psplit[8], psplit[9], psplit[10], pspli

    data_line = [int(useful[0])]
    try:
        for i in range(1,8):
            data_line.append(float(useful[i])-float(useful[i+1]
        #data_line = [float(useful[1])-float(useful[2]), float(u
        data_line.append(float(useful[-2]))
        data_line.append(useful[-1].strip())
        data_line[-1] = data_line[-1][:1].upper()
        dataset.append(data_line)
    except:
        improper +=1

print(dataset[:1])
print(f"{improper} removed for lacking some data, left {len(data

train, test = train_test_split(dataset,train_size=0.8,shuffle=T
print(f"Training dataset_size: {len(train)}, Test dataset_size: 

train_x =[]
train_y = []
test_x = []
test_y = []
for tr_line, te_line in zip(train,test):
    train_x.append(tr_line[1:3])
    train_y.append(tr_line[-1:])
    test_x.append(te_line[1:3])
    test_y.append(te_line[-1:])


train_x = np.array(train_x)
train_y = np.array(train_y)

label = list(set(train_y.flatten()))
print(label)

classifier = nn.MLPClassifier(max_iter=1000)
classifier.fit(train_x, train_y)

fig = figure.Figure()
canvas = agg.FigureCanvasAgg(fig)
ax = fig.add_subplot(111)
```

```
inspect.DecisionBoundaryDisplay.from_estimator(classifier, trair

prediction = classifier.predict(test_x)
for i in range (len (prediction)):
    subclass = prediction[i]
    match subclass:
        case "M":
            ax.plot(test_x[i][0], test_x[i][1], linestyle='None
        case "K":
            ax.plot(test_x[i][0], test_x[i][1], linestyle='None
        case "F":
            ax.plot(test_x[i][0], test_x[i][1], linestyle='None
        case "D":
            ax.plot(test_x[i][0], test_x[i][1], linestyle='None
        case "G":
            ax.plot(test_x[i][0], test_x[i][1], linestyle='None
        case "A":
            ax.plot(test_x[i][0], test_x[i][1], linestyle='None
        case "B":
            ax.plot(test_x[i][0], test_x[i][1], linestyle='None

for item,target in zip(train_x,train_y):
    match target:
        case "M":
            ax.plot(item[0], item[1], linestyle='None', marker=
        case "K":
            ax.plot(item[0], item[1], linestyle='None', marker=
        case "F":
            ax.plot(item[0], item[1], linestyle='None', marker=
        case "D":
            ax.plot(item[0], item[1], linestyle='None', marker=
        case "G":
            ax.plot(item[0], item[1], linestyle='None', marker=
        case "A":
            ax.plot(item[0], item[1], linestyle='None', marker=
        case "B":
            ax.plot(item[0], item[1], linestyle='None', marker=
fig.savefig("./S15/fig.png", dpi=100)
```

S15_01_b



S15_01_c

This code can be divided in three different part:

Data preprocessing:

```
for mline, pline in zip(main_lines,photo_lines):
    msplit = mline.split("|")
    psplit = pline.split("|")

    #r ID, U,B,V,R,I,J,H,K, absolute magnitude,spectral type
    useful = [msplit[0], psplit[8], psplit[9], psplit[10], pspli

    data_line = [int(useful[0])]
    try:
        for i in range(1,8):
            data_line.append(float(useful[i])-float(useful[i+1]
        #data_line = [float(useful[1])-float(useful[2]), float(u

        data_line.append(float(useful[-2]))
        data_line.append(useful[-1].strip())
        data_line[-1] = data_line[-1][:1].upper()
        dataset.append(data_line)
    except:
        improper +=1
```

In this code I will extract each line of data and extract the data I need to use then I will remove all improper data to keep only the line that contain all the information we need.

```
train, test = train_test_split(dataset,train_size=0.8,shuffle=Tr
print(f"Training dataset_size: {len(train)}, Test dataset_size:

train_x =[]
train_y = []
test_x = []
test_y = []
for tr_line, te_line in zip(train,test):
    train_x.append(tr_line[1:3])
    train_y.append(tr_line[-1:])
    test_x.append(te_line[1:3])
    test_y.append(te_line[-1:])
```

after that I will split the dataset in two different part : training and testing with the ratio of 80% in training and 20% in testing which is the default repartition in machine learning. and then extract the label from the number in each item.

Training

```
classifier = nn.MLPClassifier(max_iter=1000)
classifier.fit(train_x, train_y)
inspect.DecisionBoundaryDisplay.from_estimator(classifier, trai
prediction = classifier.predict(test_x)
```

I will then use the sklearn library to train the model to recognise which data is what using the training set and then make the prediction on the validation set.

Result

Then I will use mathplotlib to print the result as follow:

```
for item,target in zip(train_x,train_y):
    match target:
        case "M":
            ax.plot(item[0], item[1], linestyle='None', marker=
        case "K":
            ax.plot(item[0], item[1], linestyle='None', marker=
        case "F":
            ax.plot(item[0], item[1], linestyle='None', marker=
        case "D":
            ax.plot(item[0], item[1], linestyle='None', marker=
        case "G":
            ax.plot(item[0], item[1], linestyle='None', marker=
        case "A":
            ax.plot(item[0], item[1], linestyle='None', marker=
        case "B":
            ax.plot(item[0], item[1], linestyle='None', marker=
```

I will do this twice : once for the item in the training set and once for the item in the validation set