Matthew Jones (111277492)
Jeremy Herrmann (111575242)
CSE 353.01 Vibha Mane

Assignment 2 Report

Introduction

This project had the goal of reinforcing the topics covered in lecture by having us apply lecture knowledge through programming. We were to program, in either matlab or python, a decision tree classifier as well as a support vector machine, each for their own distinct datasets.

The decision tree classifier was used on the Titanic dataset, which contained information about the people on the Titanic . We were to use the data fields to help predict the class label “Survived” which indicated if a person on the Titanic survived the sinking of the Titanic.

The support vector machine was used on the Wisconsin Breast Cancer, which contained information about breast cancer patients from Wisconsin. We were to use the data fields to help predict the class label “Class” which indicated if a tumor was benign or malignant.

Decision Tree

Description

We used several python libraries that could help us generate a classification decision tree on our training data, and run it on our testing data. The libraries also allowed us to get really in depth statistics about our classification decision tree. Our classification decision tree predicted the “Survived” value of our Titanic dataset, which told us if a person survived the Titanic sinking. We were able to use the libraries to find the accuracy of our classification decision tree against the depth of the tree, and it also allowed us to perform k-fold cross-validation, which we did for a 5-fold example. Once we got all this data, we were then able to properly determine the adequacy of our model.

Equations - Classification Decision Tree:

- $H(x) = -\sum_{m=1}^{M} p_x(x_m) log(p_x(x_m))$, this solves for the entropy of a discrete random variable X, which takes M values, with $p_x(x)$ being the probability mass function of x.
- $H(y|x) = -\sum_{x} p_x(x) \sum_{y} p_{y|x}(y|x) log(p_{y|x}(y|x))$, this solves for the conditional entropy of y given x.
- $I(y;x) = -\sum_{x} p_x(x) \sum_{y} p_{x,y}(x,y) log \left\{ \frac{p_{x,y}(x,y)}{p_x(x) p_y(y)} \right\}$, this measures the information gain of the random variables x and y.

Algorithm and Implementation

```
def clean_dataset(data):
    data = data.drop('PassengerId', axis=1)
    data = data.drop('Name', axis=1)
    data = data.drop('Ticket', axis=1)
    number_rows = data.shape[0]
    # Counting the number of elements missing from each column
    columns_to_remove = []
    for column in data:
        if data[column].isna().sum() / number_rows >= 0.7:
            columns_to_remove.append(column)
    data = data.drop(columns_to_remove, axis=1)
    # The average age is about 29.7 years, and the peak is between 24 - 30 years, so we will replace all the missing
    # values with the mean.
    age_list = [age for age in data['Age'] if str(age) != "nan"]
    average_age = sum(age_list) / len(age_list)
    # Filling in the missing Age values using the Mean Value
    data['Age'] = data['Age'].fillna(average_age)
    # Grouping the Age into equal groups of values
    ages = [age for age in data['Age']]
    ages.sort()
    number_groups = 4
    age_groups = []
    for i in range(number_groups):
        amount_per_group = len(ages) / number_groups
        start_age = ages[int(i * amount_per_group)]
        end_age = ages[int((i + 1) * amount_per_group) - 1]
        age_groups.append((start_age, end_age))
    for i in data['Age'].index:
        for group in range(len(age_groups)):
            if age_groups[group][0] <= data['Age'][i] <= age_groups[group][1]:
                data['Age'].at[i] = group
```

```
    # Grouping the Fares into equal groups of values
    fares = [fare for fare in data['Fare']]
    fares.sort()
    number_groups = 4
    fare_groups = []
    for i in range(number_groups):
        amount_per_group = len(fares) / number_groups
        start_fare = fares[int(i * amount_per_group)]
        end_fare = fares[int((i + 1) * amount_per_group) - 1]
        fare_groups.append((start_fare, end_fare))
    for i in data['Fare'].index:
        for group in range(len(fare_groups)):
            if fare_groups[group][0] <= data['Fare'][i] <= fare_groups[group][1]:
                data['Fare'].at[i] = group
    # Dropping the missing rows of Embarked, since we only have two missing values
    data = data[data['Embarked'].notna()]
    # Replacing all male's with 0 and females with 1
    data['Sex'] = data['Sex'].replace('male', 0)
    data['Sex'] = data['Sex'].replace('female', 1)
    # Replacing all the Embarked categorical values with numerical values
    data['Embarked'] = data['Embarked'].replace('S', 0)
    data['Embarked'] = data['Embarked'].replace('Q', 1)
    data['Embarked'] = data['Embarked'].replace('C', 2)
    return data
```

This is the algorithm we used to preprocess our data, it puts things like strings into a numerical form that our library can use, as well as filling in missing values in age, grouping some of the numerical values into ranges, and getting rid of categories that do not relate to classification.

```
def split_dataset(dataset, independent_variables, response_variable):
    X = dataset[independent_variables]
    y = dataset[response_variable]
    return X, y


def partition_train_and_test(X, y, percent_test):
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, shuffle=True)
    return X_train, X_test, y_train, y_test
```

This is the algorithm we used to split our data into two distinct sets, training and testing

```
    classifiers = []
    results = []
    for depth in range(1, 10):
        classifier = fit_data(X_Train, Y_Train, depth)
        classifiers.append(classifier)
        accuracy = perform_analysis(classifier, X_Test, Y_Test)
        results.append((depth, accuracy))

        fn = Independent_Variables
        cn = [Dependent_Variable]

        fig, axes = plt.subplots(nrows = 1,ncols = 1,figsize = (4,4), dpi=600)
        tree.plot_tree(classifiers[depth-1],
                       feature_names = fn,
                       class_names="Survived",
                       filled = True);
        fig.savefig('Depth' + str(depth) + '.png')

    depth = [i[0] for i in results]
    accuracy = [i[1] for i in results]
    plt.scatter(depth, accuracy)
    plt.show()

    scores = cross_val_score(classifier, X, y, cv=5)
    print("Scores:", scores)
    print("Mean:", scores.mean())
    print("Standard Deviation:", scores.std())
```

This is the algorithm that prints out our tree, runs K-fold cross-validation with a k of 5, prints out our depth vs accuracy, and prints out tree statistics. It also prints out a visual of our tree, but we commented that out in our code, because it excessively printed 10 plots each time we ran our code.

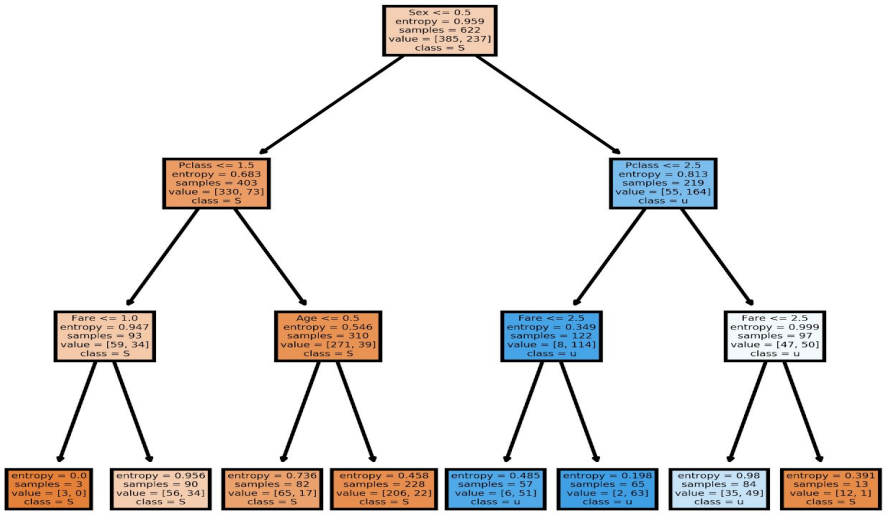
```
def generate_correlation_matrix(data, variables):
    correlation_matrix = data[variables].corr().round(2)
    correlation_plotter.heatmap(data=correlation_matrix, annot=True)
    plt.title('Correlation Matrix', fontsize=36)
    plt.show()
```

Lastly, we had an algorithm for generating a correlation matrix between our variables, and this will create and show it as a plot on the screen.

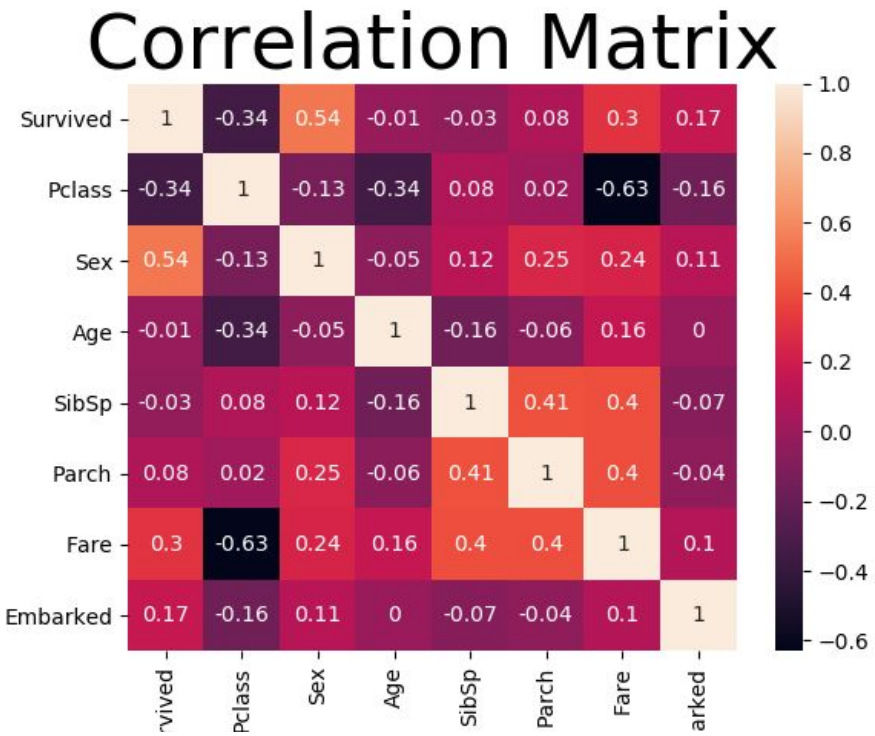
Libraries & Purposes:

- **Matplotlib:** Library that can create graphs which we used to create graphs of our regression models and their residual plots.
- **Pandas:** Library that has the ability to read excel spreadsheets into arrays which we used to take in the Boston Housing Dataset excel sheet as a multidimensional array.
- **Sklearn:** Library for machine learning, used for creation of the classifying decision tree
- **Seaborn:** Library used for plotting the correlation matrix between each of the variables

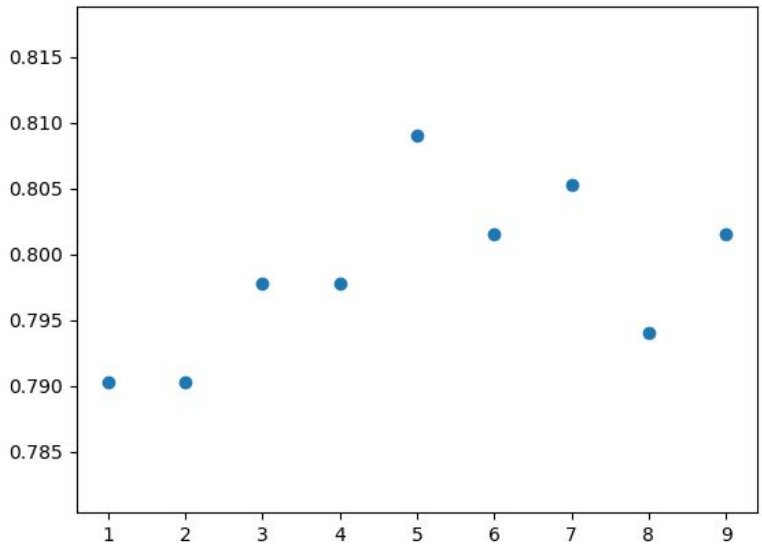
Results and Conclusions



This is an image of our classification decision tree when it has depth 3. Although we could output this image for any depth, we decided that depth 3 had enough info to show how our decision tree was working, without being too big to be contained in this document.



This is the correlation matrix our algorithm produced between all the different variables in the dataset that we used for classification.



This is the plot that was generated when we calculated the accuracy of prediction on the training and the test set, and plotted it against the depth of the tree. Our model did surprisingly well when handling the test data, and thus no overfitting has occurred.

A K-fold cross-validation is implemented by splitting your data into k distinct groups, using k-1 of them as training data, and using the last one as your test data. You repeat training and testing your classifier k times until you've used each k as a test data with all the other k's as your training data.

```
              precision    recall  f1-score   support

           0       0.79      0.83      0.81       166
           1       0.69      0.63      0.66       101

    accuracy                           0.75       267
   macro avg       0.74      0.73      0.73       267
weighted avg       0.75      0.75      0.75       267

Scores: [0.80337079 0.80337079 0.78651685 0.75280899 0.78531073]
Mean: 0.7862756300387227
Standard Deviation: 0.01846886049906726
```

The scores output is the different accuracies of the test results we got when running our classifier algorithm with a 5-fold cross-validation. This is the same accuracies we used in our accuracies vs depth portion, The classifier produces good output for every fold. This consistency shows that our classifier algorithm performs very well.

This proves that our classification decision tree algorithm is well suited to predict whether someone survived the Titanic sinking based on the other factors seen in the dataset

Support Vector Machine

Description

We used several python libraries that could help us generate a support vector machine on our training data, and run it on our testing data. The libraries also allowed us to get really in depth statistics about our support vector machine. Our support vector machine predicted the "Class" value of our Wisconsin Breast

Cancer dataset, which told us if a tumor was benign or malignant. We were able to use the libraries to find the accuracy of our support vector machine with scatter plots as well as plot the decision region of the classifier. Once we got all this data, we were then able to properly determine the adequacy of our model.

Equations - Support Vector Machine:

- $\beta_0 + \beta_1 x_1 + ... + \beta_p x_p = 0$, this is the equation for a hyperplane in a p-dimensional space.

Algorithm and Implementation

```
def generate_correlation_matrix(data, variables):
    correlation_matrix = data[variables].corr().round(2)
    correlation_plotter.heatmap(data=correlation_matrix, annot=True)
    plt.title('Correlation Matrix', fontsize=36)
    plt.show()
```

Lastly, we had an algorithm for generating a correlation matrix between our variables, and this will create and show it as a plot on the screen. This allows us to quickly find the quality of variables that we will use to train our support vector machine

```
def perform_analysis(Classifier, X, Y):
    y_pred = Classifier.predict(X)

    print(confusion_matrix(Y, y_pred))
    print(classification_report(Y, y_pred))
```

This is the algorithm that performs analysis of the support vector machine, it prints out all of our output to the console of different reports like the confusion matrix and the classification report.

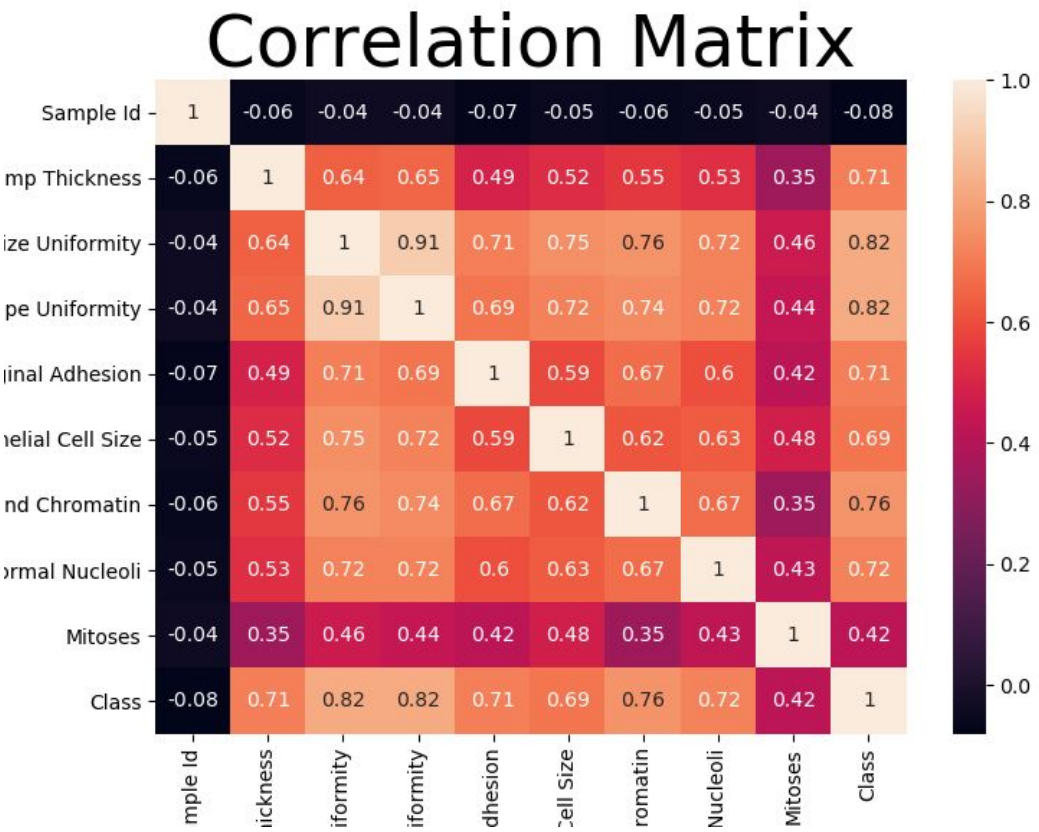
```
dataset["Bare Nuclei"] = dataset["Bare Nuclei"].apply(lambda x: int(x))
Independent_Variables = ["Cell Size Uniformity", "Mitoses", "Clump Thickness", "Cell Shape Uniformity",
                         "Marginal Adhesion", "Single Epithelial Cell Size", "Bland Chromatin", "Normal Nucleoli",
                         "Bare Nuclei"]
Dependent_Variable = "Class"
X, y = split_dataset(dataset, Independent_Variables, Dependent_Variable)

X_Train, X_Test, Y_Train, Y_Test = partition_train_and_test(X, y, 0.3)
classifier = fit_data(X_Train, Y_Train)

perform_analysis(classifier, X_Test, Y_Test)
dataset = drop_for_graph(dataset)
plt.figure(0)
correlation_plotter.pairplot(dataset, hue="Class", palette="husl", diag_kind="hist")
plt.title('MVN Simulated Data', color='#0000ff')
plt.show()

# 2D Version
pca = PCA(n_components=2)
Xreduced = pca.fit_transform(X)
X_Train, X_Test, Y_Train, Y_Test = partition_train_and_test(Xreduced, y, 0.3)
classifier = fit_data(X_Train, Y_Train)

# Code for showing decision regions
plot_decision_regions(np.array(X_Train), np.array(Y_Train), clf=classifier, legend=2, )
plt.figure(1, figsize=(20,20))
plt.show()
```

This is the algorithm we used that created all of our plots and output data, whether it be the pair plots, correlation matrix, or decision region, it was all done in this method. It also happens to be the method that we fit our classifier support vector machine to the data.

Libraries & Purposes:

- **Numpy:** Library that has array functions that we used to turn our dataframe into an array at some points.
- **Matplotlib:** Library that can create graphs which we used to create graphs of our regression models and their residual plots.
- **Pandas:** Library that has the ability to read excel spreadsheets into arrays which we used to take in the Boston Housing Dataset excel sheet as a multidimensional array.
- **Sklearn:** Library for machine learning, used for creation of the classifying decision tree
- **Seaborn:** Library used for plotting the correlation matrix between each of the variables
- **Mlxtend:** Library used for plotting the decision regions of our support vector machine.

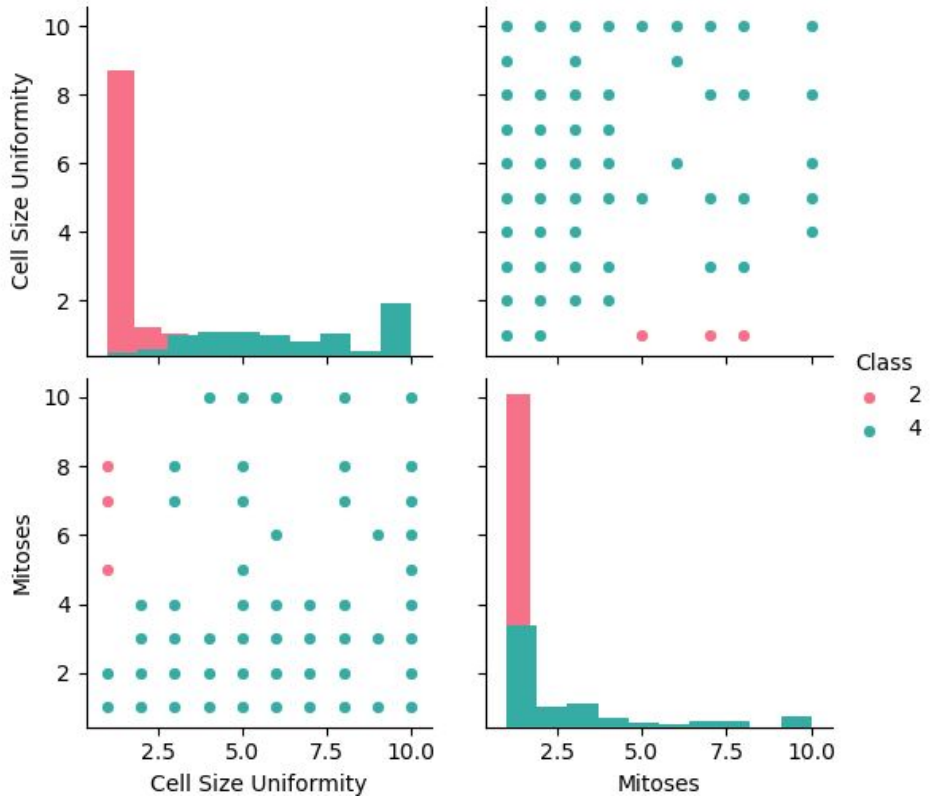
Results and Conclusions



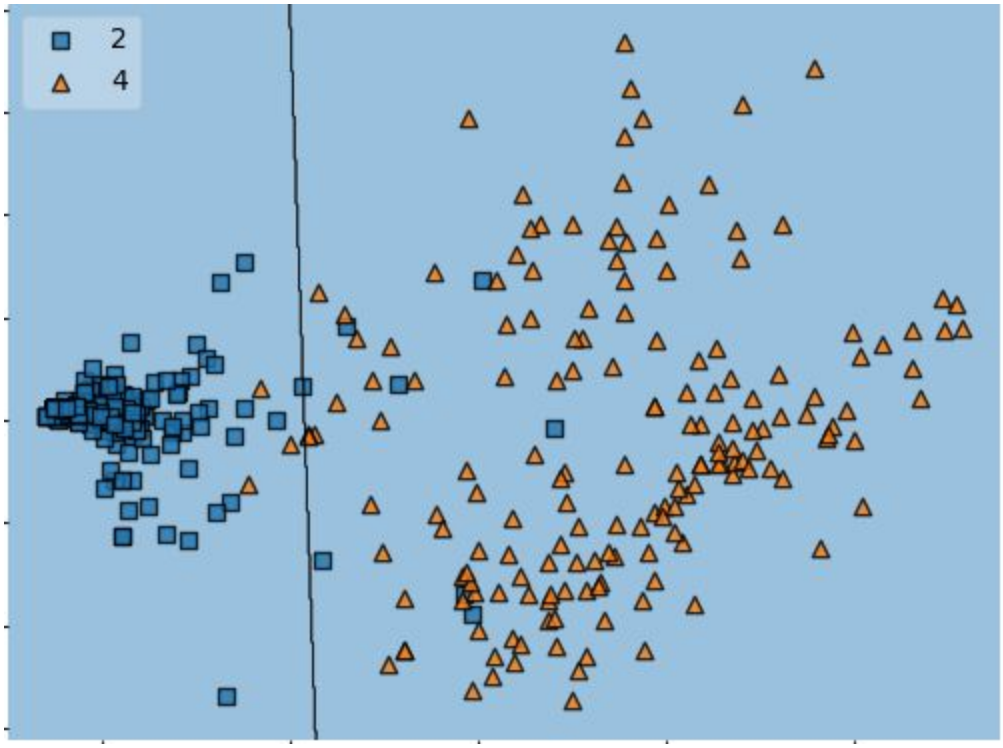
We generated a correlation matrix between all the variables in order to get a good understanding of our data before we programmed our support vector Machine.

	precision	recall	f1-score	support
2	0.97	0.91	0.94	131
4	0.85	0.95	0.90	74
accuracy			0.92	205
macro avg	0.91	0.93	0.92	205
weighted avg	0.93	0.92	0.92	205

This is the results printed by our analysis algorithm, that returns statistics about the support vector machine that was created.



We picked Cell Size Uniformity and Mitoses as our two selected independent variables that we used to create the decision region on our dataset. This was the plot that Seaborn.pairplot created for us. These were the two highest correlations we could find by looking at both the pair plots and the correlation matrix, that is why we used them to train our support vector machine.



This is the two dimensional decision region that our classifier came up with, it does a relatively decent job at splitting the data between the two different classes, malignant (4) and benign (2). We made this two dimensional decision region based off of the Cell Size Uniformity, and Mitoses category of the dataset.

References and Citations

Libraries:

- **Numpy:** https://numpy.org/
- **Matplotlib:** https://matplotlib.org/
- **Pandas:** https://pandas.pydata.org/
- **Sklearn:** https://scikit-learn.org/stable/
- **Seaborn:** https://seaborn.pydata.org/
- **Mlxtend:** http://rasbt.github.io/mlxtend/