Practical 2: Classification of objects using a radar signal and machine learning

180028981

April 2019

Contents

1 Introduction **1**

2 Loading and Cleaning the Data **1**
2.1 Datasets . . . 1
2.2 Loading the Data . . . 2
2.3 Cleaning the Data . . . 2

3 Analysing and Visualising the Data **2**

4 Preparing the Inputs and Choosing Features **4**
4.1 Splitting the Data . . . 4

5 Classifiers **4**
5.1 Logistic Regression . . . 4
5.1.1 Implementation . . . 4
5.1.2 Confusion Matrices . . . 5
5.2 Linear Support Vector Classifier . . . 5
5.2.1 Implementation . . . 5
5.2.2 Confusion Matrices . . . 6
5.3 Decision Tree . . . 7
5.3.1 Implementation . . . 7
5.3.2 Confusion Matrices . . . 7

6 Results and Evaluation **8**
6.1 Results . . . 9
6.2 Dataset XToClassify . . . 9

7 Running Instructions **9**

8 References **10**

9 Appendix **11**
9.1 XToClassify Results . . . 11

1 Introduction

The main aim of this assignment is to create a classification model while using real experimental and limited data that has not been analyzed before. There are two data sets corresponding to two tasks for this project: a binary classification task and a multiclass classification task. The report will focus on the following sections:

- Loading and Cleaning the Data
- Analysing and Visualising the Data
- Preparing the Inputs and Choosing a suitable Subset of Features
- Selecting and training a binary classification model
- Selecting and training a multiclass classification model
- Evaluating the Performance of the Model and Discussing the Results

The model will be trained using the scikit-learn library(Version 0.20.3).

2 Loading and Cleaning the Data

2.1 Datasets

The training data provided are radar readings along with the object that is being scanned. The end product will be a model that given some inputs from the radar, can identify what object is being read. Each line of data has 768 values. This is divided into three sections with the first 256 corresponding to the mean values of the readings, the second 256 corresponding to the min values of the readings and finally the last 256 corresponding to the max values of the reading. Each of these sections is divided into 4 receiver channels where each one contains 64 components. Below is a visual representation of how the data is divided.

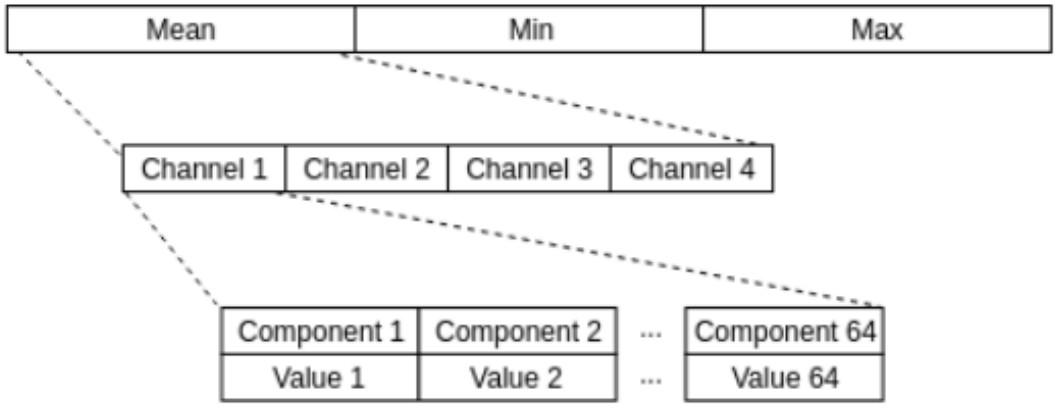


Figure 1: Visual Representation of the datasets.

2.2 Loading the Data

The data was loaded using the pandas library. The key values had to be hard-coded due to the formatting of the file. The following method was implemented, which takes an id (1 or 2) and retrieves the corresponding dataset (Binary or Multiclass).

```
def read_data(id):
    binary_file_path = "binary\\"
    multiclass_file_path = "multiclass\\"

    if id == 1:
        file_path = binary_file_path
        key = {'book': 0, 'plastic case': 1}
    elif id == 2:
        file_path = multiclass_file_path
        key = {'air': 0, 'book': 1, 'hand': 2, 'knife': 3, 'plastic case': 4}

    X = pd.read_csv(file_path+"X.csv", header=None)
    XToClassify = pd.read_csv(file_path+"XToClassify.csv", header=None)
    y = pd.read_csv(file_path+"y.csv", header=None)
    return X, y, XToClassify, key
```

2.3 Cleaning the Data

Through examination and testing the data seems to not be missing any values and be sufficiently clean. Furthermore, scaling the data is also not necessary as testing showed no improvement in the results in addition to the data already being in a relatively small range.

3 Analysing and Visualising the Data

As mentioned in section 2.1, the data is split into three sections: mean, min, and max. The following method creates three sub plots for each of these sections showing the mean of the data. Each classification output is assigned a specific color.

```
    X = X.values
    y = y.values
    X_mean = X[:, 0:256]
    X_min = X[:, 256:2 * 256]
    X_max = X[:, 2 * 256:3 * 256]
    bound = list(range(0, 256))

    num_of_classes = len(np.unique(y))

    sections = [X_mean, X_min, X_max]
    titles = ["Mean", "Min", "Max"]
    plt.tight_layout()
    fontP = FontProperties()
    fontP.set_size('small')

    index = 0
    for X in sections:

        plt.subplot(3, 1, index+1).set_title(titles[index] + " Section")
        for c in range(num_of_classes):
```

```
            values = np.mean(X[np.where(y[:, 0] == c)], axis=0)
            plt.scatter(bound, values)

        plt.legend(key, prop=fontP)
        index += 1
    plt.tight_layout(h_pad=0.30)
    plt.show()
```

Below are the plots created with the binary and multiclass datasets.

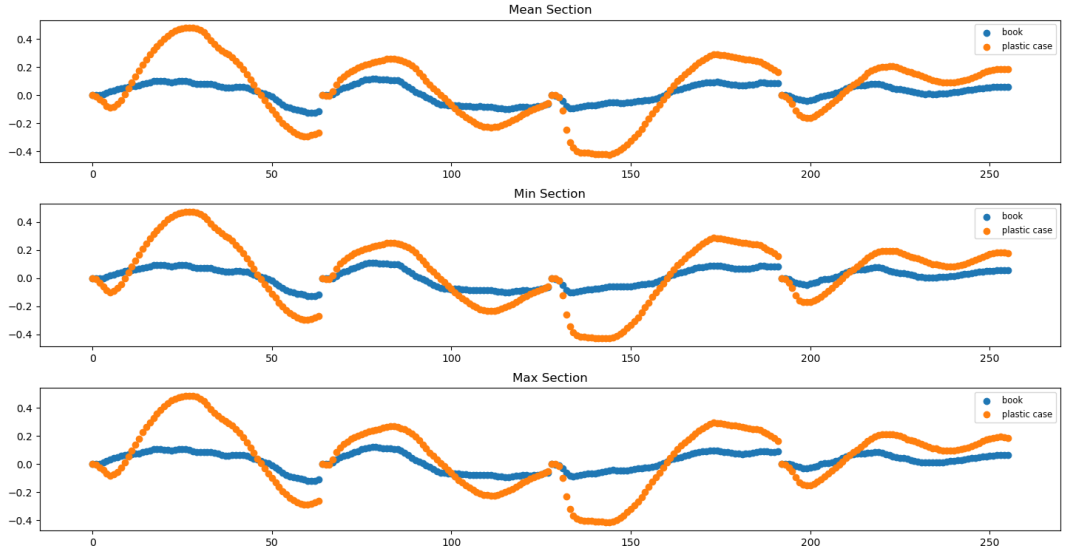


Figure 2: Binary data visualisation.

As it can be observed from figure 2, there is little overlap over the different classes which means that we can expect high accuracy results. Furthermore, the range of the data proves that there is no need for scaling.

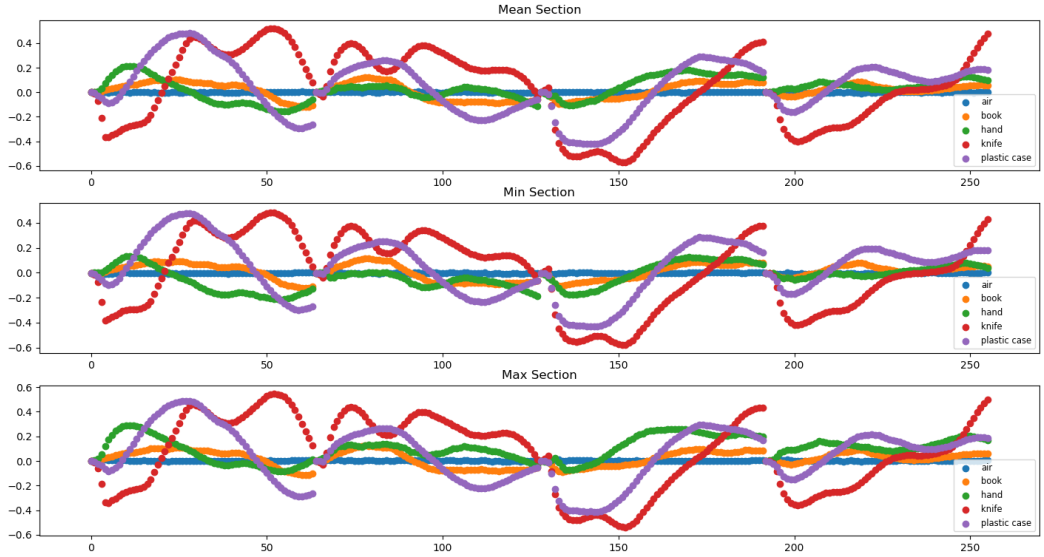


Figure 3: Multiclass data visualisation.

From figure 3, it can be observed that the classes present significantly more overlap and besides

a few outliers such as the knife class, the closer the values come to 0, the more difficult it is to distinguish them.

4 Preparing the Inputs and Choosing Features

4.1 Splitting the Data

The data is split into training and testing sets, with the ladder being 20% of the overall dataset. The data is split using this method:

```
1
2  def split_data(X, y):
3      X_train, X_testing, y_train, y_testing = train_test_split(X, y, test_size=0.20,
           shuffle=True,
4                                                              random_state=40)
5
6      return X_train, y_train, X_testing, y_testing
```

The data is shuffled before they are split.

5 Classifiers

Three algorithms were implemented in order to classify the dataset: Logistic Regression, Linear Support Vector, and Decision Tree. The library sklearn was used to implement all of these algorithms. The method used to create the matrices was taken from sklearn's documentation page. All the matrices have been normalized. Parts of the code snippets below may have been omitted if they were deemed irrelevant to the function of the algorithm.

5.1 Logistic Regression

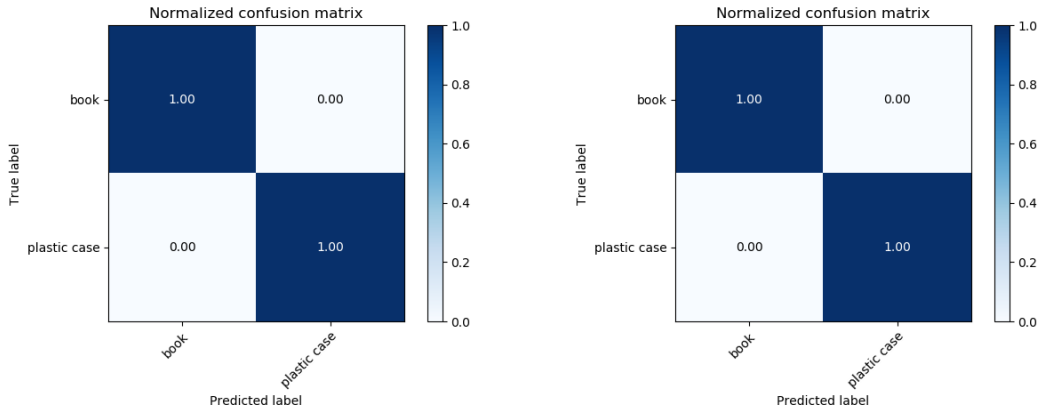
5.1.1 Implementation

The Logistic Regression algorithm calculates the probability for a variable belonging in a specific class. If the probability is above the threshold (Usually 50%), then that variable is deemed to belong in that class. In the case of multiclass classification, the variable is tested to see what is the probability of it belonging to a specific class vs all the rest of the classes. The highest value is the one chosen in the end. Below is the code used to implement this algorithm.

```
1  def logistic_regression(X_train, y_train, X_test, y_test, class_list, XToClassify):
2      model = LogisticRegression()
3      model.fit(X_train, y_train)
4      ...
5      classified_y = model.predict(XToClassify)
6      ...
7      return model, classified_y
```

5.1.2 Confusion Matrices

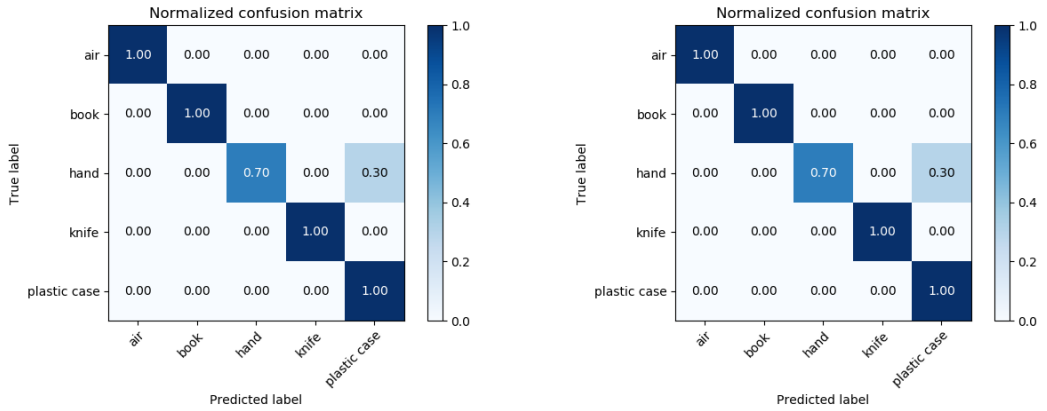
Below are the confusion matrices produces when running the Logistic Regression algorithm on the binary data set with the train and test subsets.



(a) Logistic Regression Binary Train Set (b) Logistic Regression Binary Test Set

Figure 4: Logistic Regression Performed on the Binary Data Set

Below are the confusion matrices produced when running Logistic Regression on the multiclass dataset.



(a) Logistic Regression Multiclass Train Set (b) Logistic Regression Multiclass Test Set

Figure 5: Logistic Regression Performed on the Multiclass Data Set

5.2 Linear Support Vector Classifier

5.2.1 Implementation

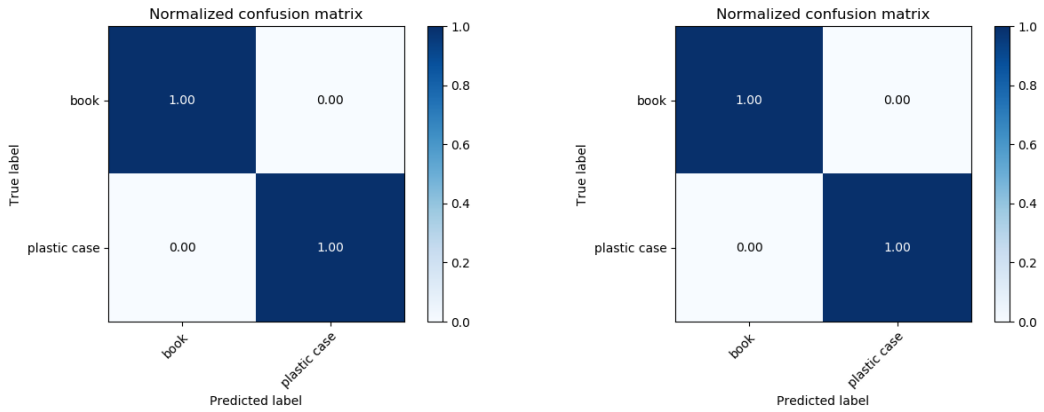
This algorithm can be used for both classification and regression projects. This algorithm tries to find an optimal hyper-plane that divides the points of the different classes. While this is effective on small and clean datasets in can experience drops in accuracy in more noise data sets. Below is the code used to implement the algorithm.

```
1 def linear_svc(X_train, y_train, X_test, y_test, class_list, XToClassify):
2     model = LinearSVC()
3     model.fit(X_train, y_train)
```

```
4     ...
5     classified_y = model.predict(XToClassify)
6     ...
7     return model, classified_y
```

5.2.2 Confusion Matrices

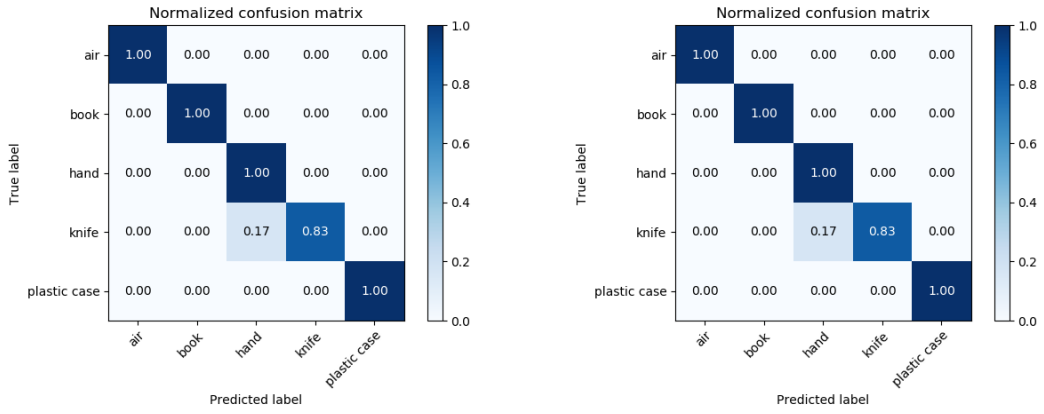
Below are the confusion matrices produces when running the Linear SVC algorithm on the binary data set with the train and test subsets.



(a) Linear SVC Binary Train Set (b) Linear SVC Binary Test Set

Figure 6: Linear Support Vector Classifier used on the Binary Data Set

Below are the confusion matrices produces when running the Linear SVC algorithm on the multiclass data set with the train and test subsets.



(a) Linear SVC Multiclass Train Set (b) Linear SVC Multiclass Test Set

Figure 7: Linear Support Vector Classifier used on the Multiclass Data Set

5.3 Decision Tree

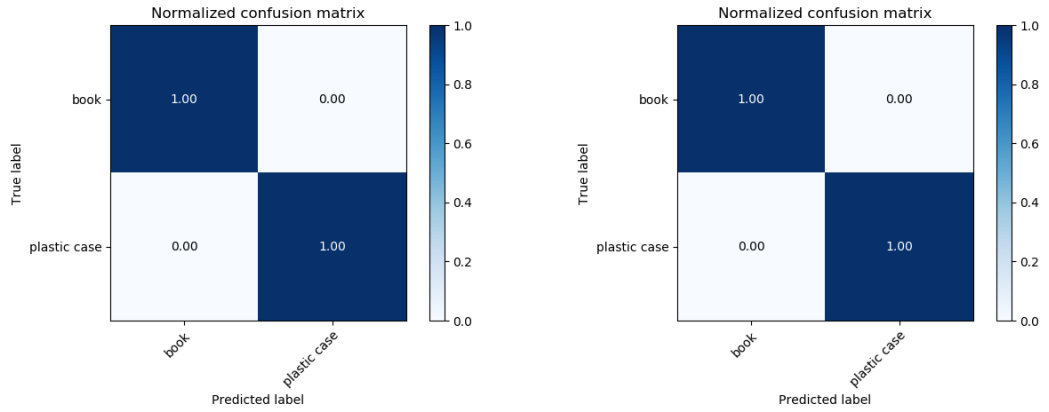
5.3.1 Implementation

Similar to Linear SVC, decision trees can be used for both classification and regression tasks. Decision trees use observations which are represented by branches to lead to a conclusion or a solution which is represented by the leaf nodes at the bottom. I decided to use decision trees as one of the algorithms to be tested because it can perform variable screening and feature selection on its own, which is a benefit since the data was not altered. Below is the code used to implement this algorithm.

```
1 def decision_tree(X_train, y_train, X_test, y_test, class_list, XToClassify):
2     model =    model = DecisionTreeClassifier()
3     model.fit(X_train, y_train)
4     ...
5     classified_y = model.predict(XToClassify)
6     ...
7     return model, classified_y
```

5.3.2 Confusion Matrices

Below are the confusion matrices produces when running the Decision Tree algorithm on the binary data set with the train and test subsets.

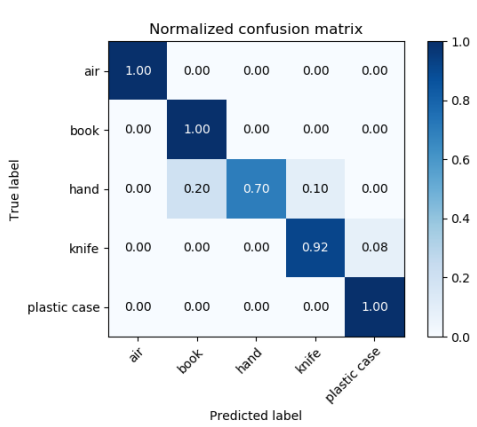


(a) Decision Tree Classifier Binary Train Set

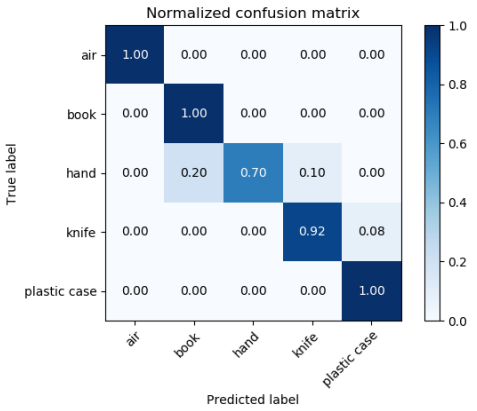
(b) Decision Tree Binary Test Set

Figure 8: Decision Tree used on the Binary Data Set

Below are the confusion matrices produces when running the Decision Tree algorithm on the multiclass data set with the train and test subsets.



(a) Decision Tree Multiclass Train Set



(b) Linear Support Vector Classifier Multiclass Test Set

Figure 9: Decision Tree used on the Multiclass Data Set

6 Results and Evaluation

The following method was implemented in order to retrieve certain metrics from each algorithm

```
def calc_scores(model, X_train, y_train, X_test, y_test, key):
    if int(len(key)) == 2:
        avg_mode = "binary"
    else:
        avg_mode = 'macro'

    y_train_pred = model.predict(X_train)

    pt1 = plot_confusion_matrix(y_train, y_train_pred, key)

    scores = cross_val_score(model, X_train, y_train, cv=5, scoring="accuracy")
    print("Cross validation scores: " + str(scores))
    print("Cross validation accuracy: " + str(np.mean(scores)))
    error = cross_val_score(model, X_test, y_test, cv=5,
                            scoring="neg_mean_squared_error").mean()
    print("SME Testing Set:", -error)
    y_test_pred = model.predict(X_test)
    pt2 = plot_confusion_matrix(y_test, y_test_pred, key)

    print("Accuracy scores: " + str(accuracy_score(y_test, y_test_pred)))
    print("Precision scores: " + str(precision_score(y_test, y_test_pred, average=
        avg_mode)))
    print("")
    return pt1, pt2
```

This method prints the accuracy and precision scores as well as the mean squared error and 5-fold cross validation.

6.1 Results

Below are two tables containing the results.

	Cross Validation Accuracy	**SME**	**Accuracy**	**Precision**
Logistic Regression	0.9857	0	1	1
Linear SVC	1	0	1	1
Decision Tree	0.9833	0	1	1

Table 1: Binary Test Set Results

	Cross Validation Accuracy	**SME**	**Accuracy**	**Precision**
Logistic Regression	0.9483	0.6722	0.975	0.98
Linear SVC	0.9492	1.00	0.95	0.9636
Decision Tree	0.9363	0.4722	0.925	0.9177

Table 2: Multiclass Test Set Results

When examining table 1, it can be observed that all the algorithms performed on par with each other and all were able to get near perfect scores, with every metric besides CV being the best result. Linear SVC is the only one that got a perfect 1 in its cross validation score. However, in table 2 we can observe that while accuracy and precision were high in all of the algorithms, Logistic Regression actually outperforms the rest with the highest scores in both accuracy and precision. All three algorithms had a relatively large squared mean error which may be a factor of the overlap of classes in the data set as it was observed in Figure 3 in Section 3.

6.2 Dataset XToClassify

One of the tasks of this assignment was to assign y values to unlabeled X data. This values can be found in the Appendix of this report and also in a folder when the program is run. (More information in Section 7 below)

7 Running Instructions

The code produces some warnings which may be obtrusive when trying to view the results. Therefor it is recommended to run the program in the following manner:
`python -W ignore Classifier.py` The file needs to be placed in the same directory as two folders named binary and multiclass which contain the X, y, and XToClassify csv files that were provided for the purposes of this coursework. The Confusion Matrices will be saved in a folder in the same directory as the python file in order to avoid intrusions when running the program. The XToClassify y values will also be saved in a separate folder in the same directory.

8 References

- VanderPlas J. (2016), Python Data Science Handbook, 1st Edition, O'Reilly Media
- Machine Learning Lecture Notes from St. Andrews University, written by Terzić K. and Harris-Birtill D.
- scikit-learn Version 0.20.3
- Figure 1 (2019) was provided by David Harris-Birtill

9 Appendix

9.1 XToClassify Results

All three algorithms produced the same output for the binary classifier. In the table below is the output.

ID	**y**
1	1
2	1
3	1
4	1
5	1
6	1
7	1
8	1
9	1
10	1
11	0
12	0
13	0
14	0
15	0
16	0
17	0
18	0
19	0
20	0

Table 3: Binary Classifier Output (Three algorithms)

Below are the results for the multiclass classifiers.

ID	Decision Tree	Linear Regression	Linear SCV
1	2.00E+00	2.00E+00	2.00E+00
2	2.00E+00	2.00E+00	2.00E+00
3	2.00E+00	2.00E+00	2.00E+00
4	2.00E+00	1.00E+00	2.00E+00
5	0.00E+00	4.00E+00	2.00E+00
6	2.00E+00	2.00E+00	2.00E+00
7	2.00E+00	2.00E+00	2.00E+00
8	2.00E+00	2.00E+00	2.00E+00
9	2.00E+00	2.00E+00	2.00E+00
10	2.00E+00	2.00E+00	2.00E+00
11	0.00E+00	0.00E+00	0.00E+00
12	0.00E+00	0.00E+00	0.00E+00
13	0.00E+00	0.00E+00	0.00E+00
14	0.00E+00	0.00E+00	0.00E+00
15	0.00E+00	0.00E+00	0.00E+00
16	0.00E+00	0.00E+00	0.00E+00
17	0.00E+00	0.00E+00	0.00E+00
18	0.00E+00	0.00E+00	0.00E+00
19	0.00E+00	0.00E+00	0.00E+00
20	0.00E+00	0.00E+00	0.00E+00
21	3.00E+00	3.00E+00	3.00E+00
22	3.00E+00	3.00E+00	3.00E+00
23	3.00E+00	3.00E+00	3.00E+00
24	3.00E+00	3.00E+00	3.00E+00
25	2.00E+00	3.00E+00	3.00E+00
26	3.00E+00	3.00E+00	3.00E+00
27	3.00E+00	3.00E+00	1.00E+00
28	3.00E+00	3.00E+00	3.00E+00
29	3.00E+00	3.00E+00	3.00E+00
30	2.00E+00	3.00E+00	3.00E+00
31	1.00E+00	1.00E+00	1.00E+00
32	1.00E+00	1.00E+00	1.00E+00
33	1.00E+00	1.00E+00	1.00E+00
34	1.00E+00	1.00E+00	1.00E+00
35	1.00E+00	1.00E+00	1.00E+00
36	1.00E+00	1.00E+00	1.00E+00
37	1.00E+00	1.00E+00	1.00E+00
38	1.00E+00	1.00E+00	1.00E+00
39	1.00E+00	1.00E+00	1.00E+00
40	1.00E+00	1.00E+00	1.00E+00
41	4.00E+00	4.00E+00	4.00E+00
42	4.00E+00	4.00E+00	4.00E+00
43	4.00E+00	4.00E+00	4.00E+00
44	4.00E+00	4.00E+00	4.00E+00
45	4.00E+00	4.00E+00	4.00E+00
46	4.00E+00	4.00E+00	4.00E+00
47	4.00E+00	4.00E+00	4.00E+00
48	4.00E+00	4.00E+00	4.00E+00
49	4.00E+00	4.00E+00	4.00E+00
50	4.00E+00	4.00E+00	4.00E+00

Table 4: Multiclass Classifier Output