Implementing Canonical Models To Predict Admissions

Test 2, CSCI-297A: Introduction to Machine Learning
Washington and Lee University

Team Rocket: Abhi Jha '21, Tina Jin '21, Utkrist P. Thapa '21

October 20 2020

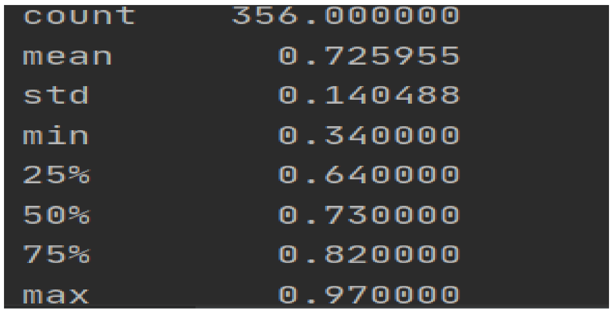
1 Exploratory Data Analysis

The original dataset that we have used in this assignment in order to predict admissions consists of 400 entry rows and 11 columns. Among these columns, the first column is simply the serial number which we omit from our feature set right away and the last 10 columns can be considered features which we will use to approximate the target values. The ninth column in the comma separated file made available to us for this task is the target value (Chance of Admit–continuous over 0.0-1.0). All entries in the dataset are numerical values with the exception of the 10th column (Race–Asian, latinx, white, african american). The 8th column is a binary feature (Research–YES/NO) and is denoted by either 1 or 0.

To examine the properties of the features and target in our dataset, we import the data into a Pandas dataframe and rename the column headers to abbreviated forms and split the categorical 'Race' column into four additional columns with binary YES/NO denoted by 1s and 0s. Then we reorder the columns in the dataframe such that the the column 'Chance of admit' (abbreviated as 'ADM') is the last column. We also drop any rows with missing values. The columns in our dataframe now look like this: ['GRE', 'TOEFL', 'URATE', 'SOP', 'LOR', 'CGPA', 'RES', 'SES', 'RACE_Asian', 'RACE_african american', 'RACE_latinx','RACE_white', 'ADM'].

Why not imputation?

Figure 1 shows the central tendency and dispersion measurements summarizing our dataset after dropping all rows with missing values. Initially we had 400 rows, leaving us with 356 rows after the drop. This reduces our dataset size by 11%, which we believe is acceptable. Furthermore, the data in the 356 rows seem to have a good range(34-97%) and a believable standard deviation of 14%. Hence, as long as we are able to work with original data, we decided to avoid synthesizing data which may hurt our model's accuracy.



```
count     356.000000
mean        0.725955
std         0.140488
min         0.340000
25%         0.640000
50%         0.730000
75%         0.820000
max         0.970000
```

Fig 1: Summary of the target column ‘ADM’.

Figure 2 explores the relationship between the columns in the dataset contained in Admission_Predict.csv by plotting scatterplots. We identify the features in the dataset as GRE score, TOEFL score, university rating, statement of purpose rating, letter of recommendation rating, cumulative GPA, research, socioeconomic status percentage, and race. Displayed on the bottom row are scatterplots of all twelve feature columns against our target value column (Chance of Admit abbreviated as ‘ADM’).

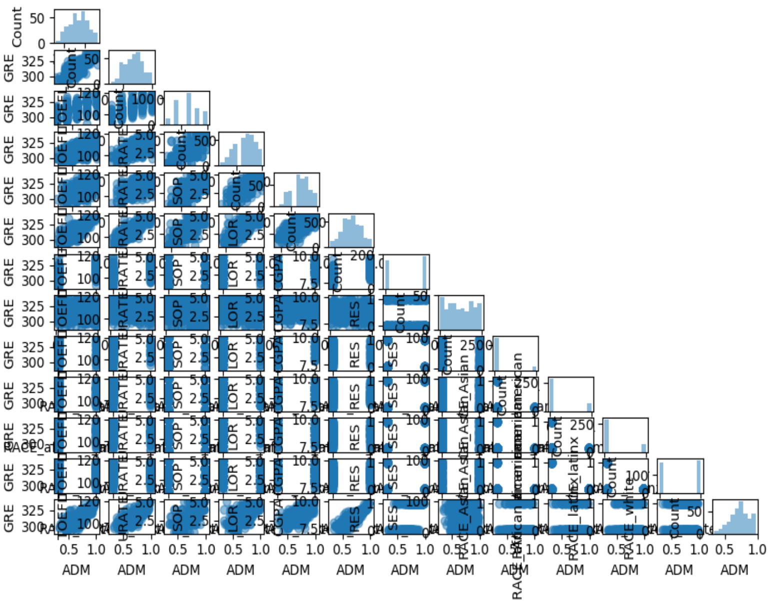


Fig 2: Scatterplot comparing the relationship between the columns of the our data set

On the bottom row, we observe that the non-categorical and non-binary columns all have a linear relationship with chances of admission with the exception of ‘SES’. This means that unlike the GRE score, or the TOEFL score, socioeconomic status does not have a strong linear correlation with chances of admission. We pulled up a correlation coefficient matrix in order to quantify the relationship as shown in Figure 3:

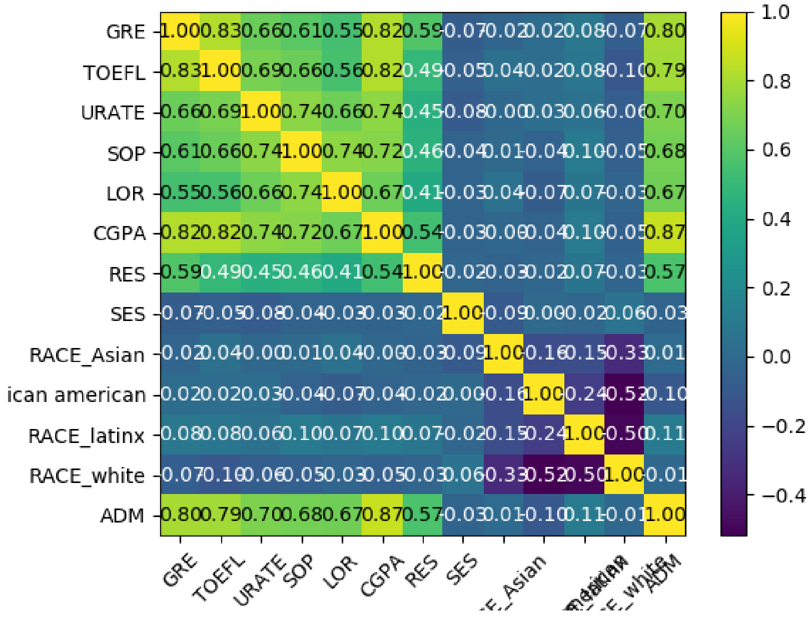
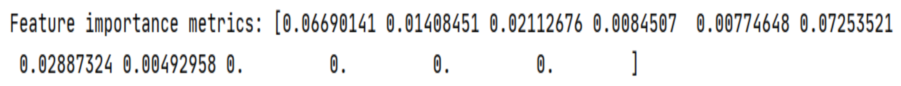


Fig 3: Correlation coefficient matrix showing color-coded linear correlations between the columns in the dataset

We see that 'SES' has very low correlation with every column in the dataset other than itself. Furthermore, we used a random forest classifier imported from sklearn in order to examine how much the random forest model depend on each feature, after converting chances of admission into binary values by setting a threshold (more on this in later sections). We found that race and socioeconomic status percentage had very little impact on chances of admission as shown in Figure 4:



```
Feature importance metrics: [0.06690141 0.01408451 0.02112676 0.0084507  0.00774648 0.07253521
 0.02887324 0.00492958 0.         0.         0.         0.        ]
```

Fig 4: Feature importance in classification obtained using a random forest. The order of index shown here is: 'GRE', 'TOEFL', 'URATE', 'SOP', 'LOR', 'CGPA', 'RES', 'SES', 'RACE_Asian', 'RACE_african american', 'RACE_latinx','RACE_white'.

Hence, we decided to omit 'SES' from our feature set. The reason we keep race is to make our model understand the race feature in the dataset so that it does not fall victim to the proxy problem when we attempt to apply the model in unseen data. Fortunately, we see that race does not have a big impact or correlation with chances of admission in this dataset which gives us confidence that our model will not be biased if we include race in the feature set.

Feature Scaling:

In order to visualize the distribution of our features, we plot histograms with matplotlib.pyplot as shown in Figure 5 and Figure 6.

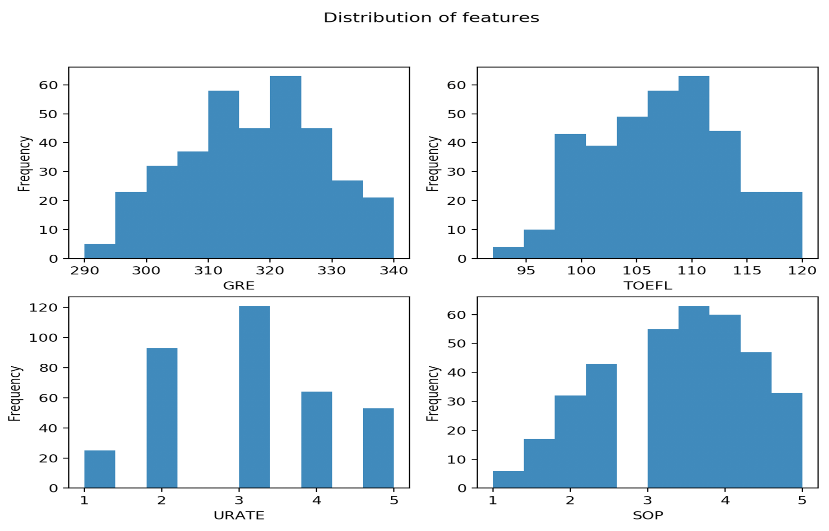


Fig 5: Histograms showing the distribution of feature columns GRE, TOEFL, URATE and SOP.

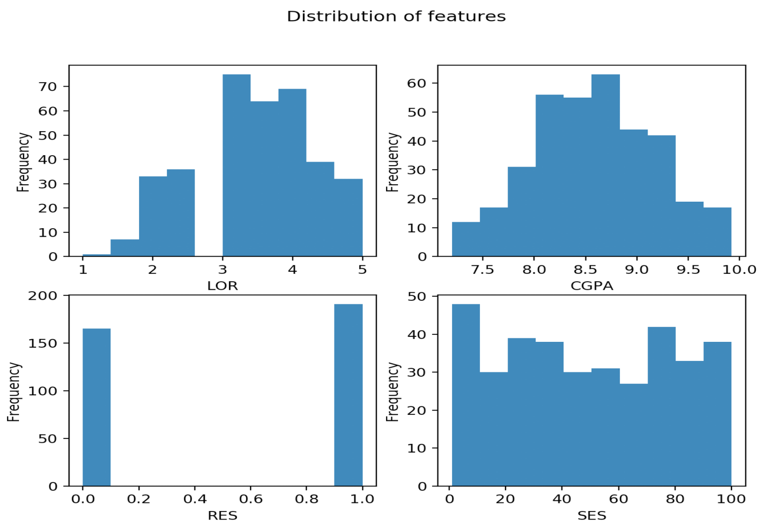


Fig 6: Histograms showing the distribution of feature columns LOR, CGPA, RES, SES.

Here, 'RES' is a binary feature and we did not bother plotting histograms for any of the 'RACE_' columns since those are binary features as well. Those aside, we observe that all feature columns have a normal distribution since they all follow the shape of the bell curve ('SES' is an exception to this but we have already omitted 'SES' from the feature set).

Since the features are normally distributed, we decided to use StandardScaler() from sklearn.preprocessing in order to scale our features without losing the original feature relationships shown in the scatterplots in Figure 2.

2 Model Selection

We decided to submit models for both continuous target and binary target. The target in the dataset is chances of admission, which is continuous over 0.0-1.0. To turn this continuous target to a binary classification problem, we decided to set a threshold that would define whether or not a person is admitted to college or not.

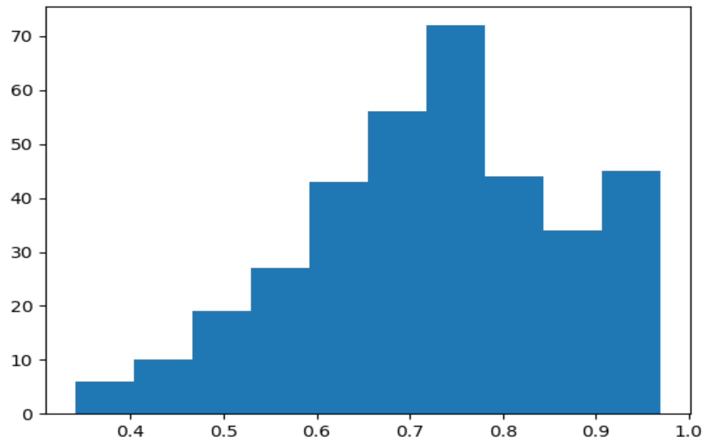


Fig 7: Histogram showing distribution of target column ‘ADM’

From Figure 7, we see that the distribution of ‘ADM’ is not particularly skewed. From Figure 1, we know that the median of ‘ADM’ is 0.73. Hence, we decided that a threshold of 0.73 would be a good threshold to turn the continuous target into binary classes of 0s and 1s. Any number greater than or equal to 0.73 would be classified 1 and the rest would be 0.

For regression predicting the continuous target, we tried linear regression and random forest regressor. For binary classification predicting the binary classes after applying the threshold, we tried logistic regression with stochastic gradient descent, KNN, decision tree/random forest, SVM and Naive Bayes. The R^2 scores that measure the performance of regression models and the accuracies that measure the performance of classification models are given in the table below. For models that produce different results for random splits of train-test data, we run them 100 times and get the average accuracy.

	Type of model	Accuracy/R^2 score over 100 iterations
Linear Regression	Regression	**0.792**
Random Forest Regressor	Regression	0.773
Logistic Regression	Classification	**0.875**
KNN	Classification	0.827
Decision Tree	Classification	0.819
Random Forest	Classification	0.833 (20 iterations due to ensemble size)
SVM	Classification	0.875
Naive Bayes	Classification	0.873

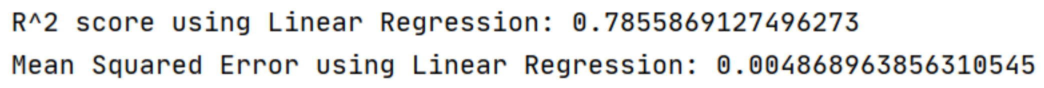
All models tabulated in the table above have been imported from sklearn (with the exception of logistic regression). We decided to choose the model with the best average accuracy over 100 iterations. Hence, we implement a logistic regression model for classification, and a linear regression model for continuous regression.

3 Linear Regression Model

In our implementation of linear regression, we implement a linear_regression() method along with other appropriate methods for training, predicting and calculating error. We do the preprocessing of data in function get_data. We import the data from the file into a Pandas dataframe, abbreviate the column headers, reorder the columns. Then we store the feature columns into a variable 'X' and the target column 'ADM' into a variable 'y'. Then we apply train_test_split imported from sklearn.model_selection to randomly split (x, y) into training and test sets (80-20 split). We apply the StandardScaler() to the features in order to make sure no feature overpowers another.

We implement linear regression using the default sklearn model LinearRegression(), which fits the training set, and predicts the values for the testing X values. The method returns evaluation metrics R^2 score and mean squared error, along with y_test, and y_pred, the predicted y values.

To understand the performance of the model, we print out the R^2 score and mean squared error of the model first. Though the R^2 score is not very high, we decide that it is understandable because the target variable changes in increments of 0.01. The $MSE = 0.005 < 0.01$, so the error is possibly caused by the sparsity of data points instead of the model itself.



```
R^2 score using Linear Regression: 0.7855869127496273
Mean Squared Error using Linear Regression: 0.004868963856310545
```

Fig 8: R^2 score and mean squared error of linear regression

Since the dataset has more than 2 features, it is hard to visualize the actual regression line. To visualize the performance though, we plot a graph of actual y values vs predicted y values, to see where the points lie around the ideal $f(x) = x$ line.

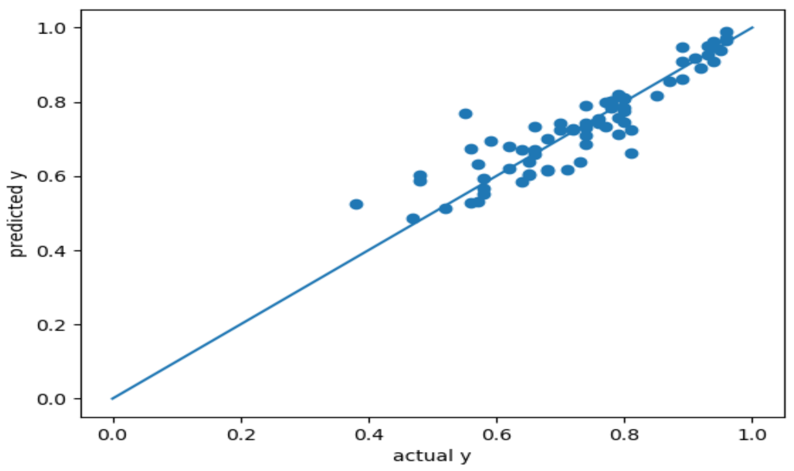


Fig 9: Actual y values vs predicted y values

4 Logistic Regression Classifier

In our implementation of logistic regression, we write a LogisticRegression() class with the appropriate methods for training, predicting and calculating accuracy. In the main function, we import the data from the file into a Pandas dataframe, abbreviate the column headers, reorder the columns and store the feature columns into a variable 'x'. Then, we convert the target column 'ADM' to binary values based on the condition (if value >= 0.73 value = 1 else value = 0). We store this column in a variable 'y'. Then we apply train_test_split imported from sklearn.model_selection to randomly split (x, y) into training and test sets (80-20 split). We apply the StandardScaler() to the features in order to make sure no feature overpowers another.

We implement logistic regression with stochastic gradient descent from Tensorflow. Hence, we split the training dataset into batches. Here, the number of batches determines the number of iterations.

Hyperparameters:

1. Number of Batches: 200
2. Learning Rate: 0.3
3. Batch size: 60

The reason why we have these values for hyperparameters can be explained with Figure 10 and Figure 11:

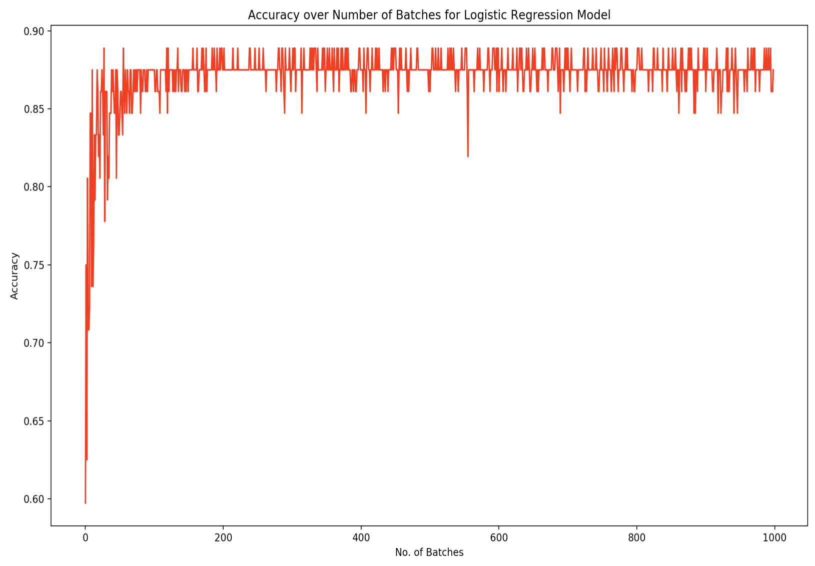


Fig 10: Graph showing the accuracy over the number of batches (iterations) for our logistic regression classifier

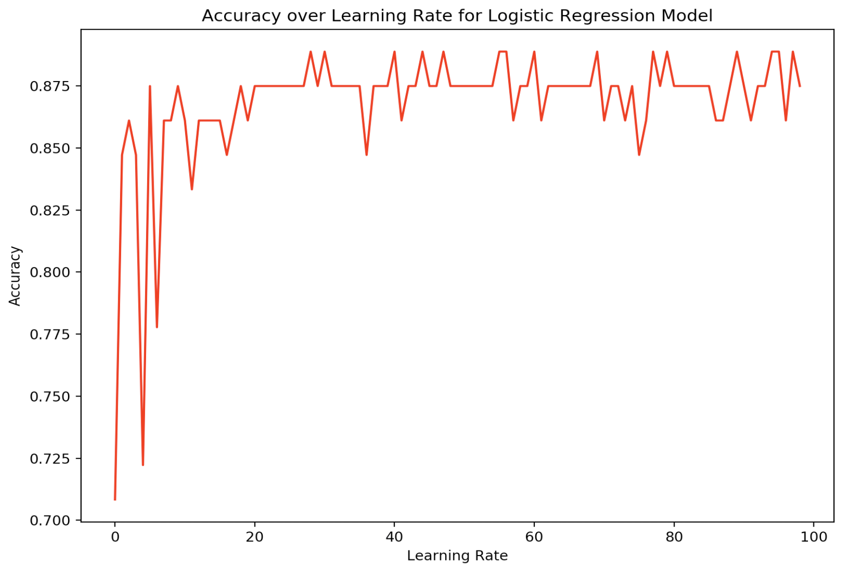


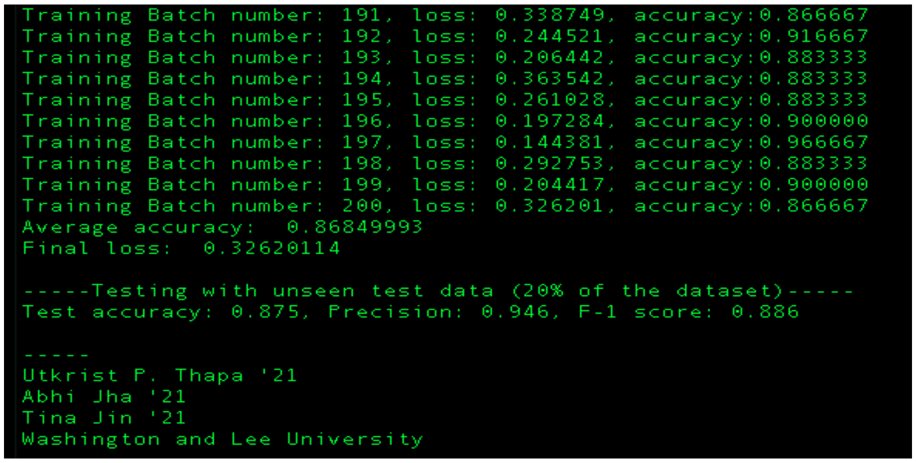
Fig 11: Graph showing the accuracy over learning rate (times 100) ranging from 0.1 to 1.

We observe from Figure 10 that the accuracy starts increasing swiftly over the first 100 batches and remains somewhat steady until 1000. We see the jagged lines in accuracy due to random batching and stochastic gradient descent. We decided that 200 batches would be an appropriate cutoff since the accuracy is not really increasing any further.

From Figure 11, we can see that the accuracy reaches a maximum and remains somewhat steady after 20 (lr = 0.2). We see a jagged graph here for the same reasons mentioned earlier. Hence, we decided to go with a learning rate of 0.3.

Finally, since the total number of rows in our dataframe is 356, we decided that a batch size of 60 would be appropriate.

Results:



```
Training Batch number: 191, loss: 0.338749, accuracy:0.866667
Training Batch number: 192, loss: 0.244521, accuracy:0.916667
Training Batch number: 193, loss: 0.206442, accuracy:0.883333
Training Batch number: 194, loss: 0.363542, accuracy:0.883333
Training Batch number: 195, loss: 0.261028, accuracy:0.883333
Training Batch number: 196, loss: 0.197284, accuracy:0.900000
Training Batch number: 197, loss: 0.144381, accuracy:0.966667
Training Batch number: 198, loss: 0.292753, accuracy:0.883333
Training Batch number: 199, loss: 0.204417, accuracy:0.900000
Training Batch number: 200, loss: 0.326201, accuracy:0.866667
Average accuracy:  0.86849993
Final loss:  0.32620114

-----Testing with unseen test data (20% of the dataset)-----
Test accuracy: 0.875, Precision: 0.946, F-1 score: 0.886

-----
Utkrist P. Thapa '21
Abhi Jha '21
Tina Jin '21
Washington and Lee University
```

Fig 12: Results from the logistic regression with stochastic gradient descent classifier implementation showing 87.5% accuracy, 94.6% precision and an F-1 score of 88.6% on unseen test data.

5 Libraries used:

1. TensorFlow
2. Scikit-Learn
3. Matplotlib
4. Pandas
5. NumPy
6. mlxtend.plotting

Team Rocket has neither given nor received any unacknowledged aid in this test. The resources we have used in order to be able to complete this implementation are as follows:

1. ”Python Machine Learning”, 3rd edition, Sebastian Raschka & Vahid Mirjalili
2. All resources posted to Canvas course site
3. TensorFlow official API documentation
4. Scikit-Learn official API reference