Table of Contents

Table of Figures

Project Overview

The Project 2 focusses deeply on the aspect of a model development and its learning curve. The machine learning concepts involving the training, validation and testing of a logical programming model in determining the best and the worst features in a model to predict the progression of a disease. The objective is to try and tinker with the model's hyperparameters and possibly arrive at the most optimal solution.

Problem Statement 1

Question

The expectation is to generate plots of the Diabetes dataset features versus quantitative measure of disease progression for the following conditions:

1. A plot showing one feature that has a strong linear relationship with disease progression.
2. A plot showing one feature that has a strong nonlinear relationship with disease progression.
3. A plot showing one feature that has a weak relationship with disease progression.

Code

THE ENTIRE CODE BLOCK FOR THE PROBLEM STATEMENT 1 CAN BE FOUND IN THE APPENDIX OF THIS DOCUMENT.

Output

The dataset chosen to compare the disease progression in accordance with the features is the Diabetes dataset available on the *SKLearn* repository.

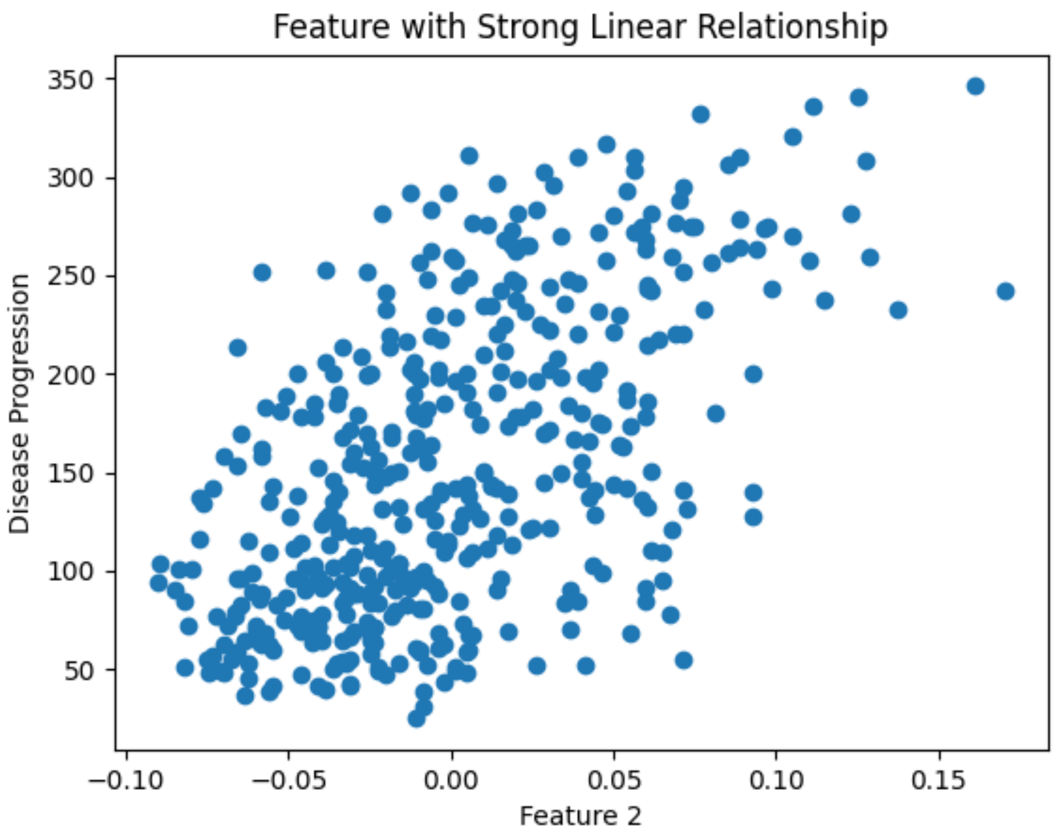

Figure 1: Feature vs Disease Progression for Strong Linear Relationship

The Figure 1 above represents the cluster plot for the strong linear relationship. For this plot the program has chosen Feature 2 to have this relationship among the other set of features.

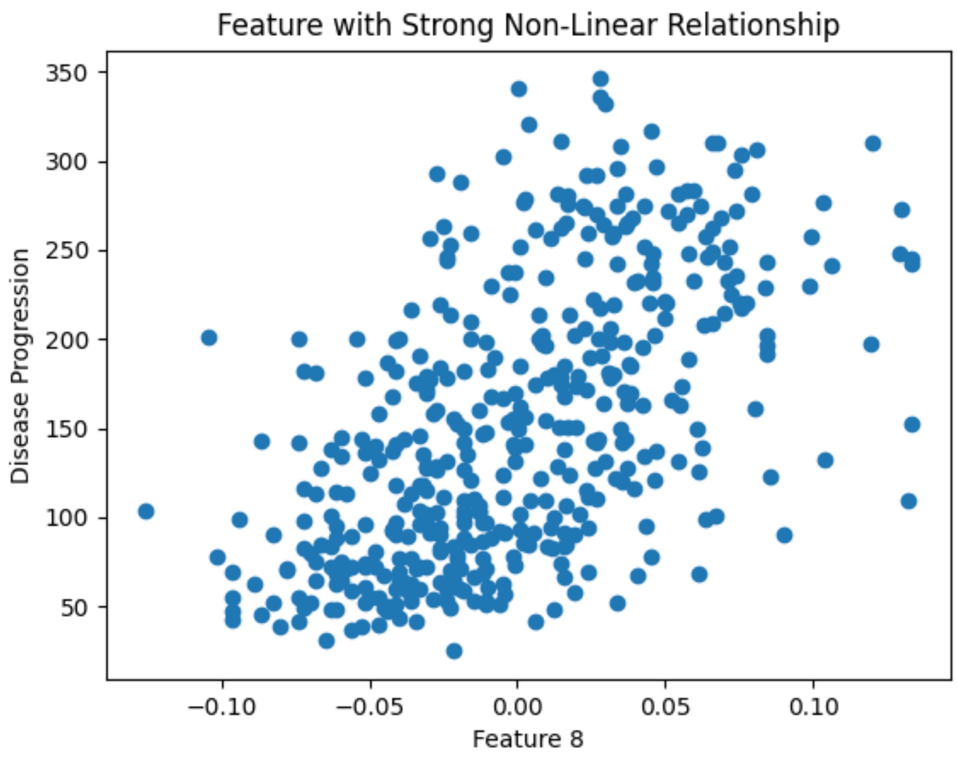

Figure 2: Feature vs Disease Progression for Strong Non-Linear Relationship

The Figure 2Figure 1 above represents the cluster plot for the strong non-linear relationship. For this plot the program has chosen Feature 8 to have this relationship among the other set of features.

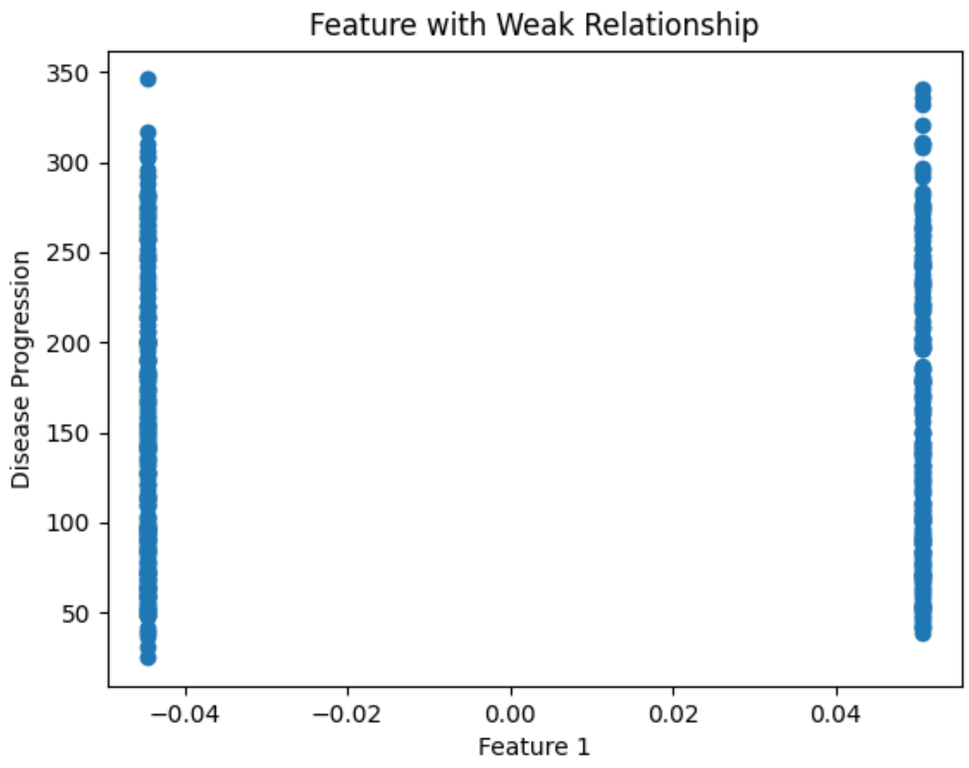

Figure 3: Feature vs Disease Progression for Weak Relationship

The Figure 3 above represents the cluster plot for the weak relationship. For this plot the program has chosen Feature 1 to have this relationship among the other set of features.

Problem Statement 2

Question

A python code is expected that divided a dataset into three parts with the ratio: Training (70%), Validation (15%) and Testing (15%). It is imperative to note the value of the random state number so that the same program's output can be verified.

Code

THE ENTIRE CODE BLOCK FOR THE PROBLEM STATEMENT 2 CAN BE FOUND IN THE APPENDIX OF THIS DOCUMENT.

Output

```
Training set shape: (398, 30) (398,)
Validation set shape: (85, 30) (85,)
Testing set shape: (86, 30) (86,)
```

From the output derived it is understood that the code is working as intended. The code is implemented on the Breast Cancer Dataset from the *SKLearn* repository. The code also ensures that the data split between training and validation, testing is entirely separate and there is no overlap.

Category	Percentage split	Number of rows (samples)
Training	70%	398
Validation	15%	85
Testing	15%	86

From the above table, it can be inferred that the total number of rows (samples) in the data is $398 + 85 + 86 = 569$ which matches with the overall data samples available in the dataset. Also, 70% of 569 is 398 and 15% of 569 is 85 or 86 based on the selection of either floor or ceiling.

Problem Statement 3

Question

This question requires the usage of Linear Lasso (L1 Regularization) to determine 5 features that do the best job in predicting disease progression. The input of all hyperparameters should be changed and the corresponding best features or any other learned parameters are to be looked at in the output. The score value of the final model when the test data is applied is to be noted.

Code

THE ENTIRE CODE BLOCK FOR THE PROBLEM STATEMENT 3 CAN BE FOUND IN THE APPENDIX OF THIS DOCUMENT.

Output

This code uses the Linear Lasso (L1 Regularization) method in determining the best five features for the job to be done. This program uses the Breast Cancer dataset from the *SKLearn* database to compute the following results.

```
All Features:
mean radius: -0.0
mean texture: -0.0
mean perimeter: -0.0
mean area: -0.0
mean smoothness: -0.0
mean compactness: -0.0
mean concavity: -0.0
mean concave points: -0.0
mean symmetry: -0.0
mean fractal dimension: 0.0
radius error: -0.0
texture error: -0.0
perimeter error: -0.0
area error: -0.0
smoothness error: -0.0
compactness error: -0.0
concavity error: -0.0
concave points error: -0.0
symmetry error: -0.0
fractal dimension error: -0.0
worst radius: -0.0837061947395551
worst texture: -0.0
worst perimeter: -0.0
worst area: -0.0
worst smoothness: -0.0
worst compactness: -0.0
worst concavity: -0.0
worst concave points: -0.12349285739722492
worst symmetry: -0.0
worst fractal dimension: -0.0

Model Efficiency score: 0.46503869170904544

The best 5 features are:
worst fractal dimension : -0.0
worst symmetry : -0.0
mean texture : -0.0
mean perimeter : -0.0
mean area : -0.0

Final Score: 0.4875376493890684
```

Observations

There is an immediate observation that can be noticed is that the efficiency score of the model has improved from the validation set to the final test set. Although, this score is a very minor improvement, it still is significant.
The two major hyperparameters that can be altered which brings out a visible and desirable change to the out are:

- Alpha value of the lasso function
- Train, Validation and Test split ratio

Alpha value	Train : Validation : Test (ratio)	Efficiency Score	Names of selected features
0.2	80:15:5	0.487537649389	Worst fractal dimension; worst symmetry; mean texture; mean perimeter; mean area
0.1	70:24:6	0.613257188968	Worst fractal dimension; worst symmetry; mean texture; mean perimeter; mean area
0.4	60:20:20	-0.00460162674695	Worst fractal dimension; worst symmetry; mean texture; mean perimeter; mean area

From the table it can inferred that the most significant impacting hyperparameter for the model is the alpha score of the Lasso model. This hyperparameter determines the strength of the regularization model. Although, the ratio of training, validation and test does impact the overall efficiency score, but only to a certain degree of manipulation.

Problem Statement 4

Question

This question requires the usage of any of the fundamental regression algorithm to predict disease progression from a dataset using the top 5 best features that perform the best job for the same. The change and tuning done to the input via hyperparameters is to be documented along with the list of chosen final features in the output, the score value of the final model and the other output parameters. It is also important to note down observations on the efficiency of the model in data prediction, one example each from the test data that the model predicts poorly and well, along with a few pointers to improve model's efficiency.

Code

THE ENTIRE CODE BLOCK FOR THE PROBLEM STATEMENT 4 CAN BE FOUND IN THE APPENDIX OF THIS DOCUMENT.

Output

The code for this problem focusses on the breast cancer dataset from the *SKLearn* repository as well. Here, the Decision Tree Regressor is used to train the model and a score is drawn for the model's efficiency. The following the output drawn from a randomly chosen set of hyperparameters.

```
All Features:
mean radius: 0.0
mean texture: 0.05776928953399541
mean perimeter: 0.0
mean area: 0.0
mean smoothness: 0.0
mean compactness: 0.0
mean concavity: 0.0
mean concave points: 0.0
mean symmetry: 0.0
mean fractal dimension: 0.018254787295034974
radius error: 0.0
texture error: 0.0
perimeter error: 0.0
area error: 0.0
smoothness error: 0.0
compactness error: 0.010268317853457172
concavity error: 0.0
concave points error: 0.0
symmetry error: 0.0
fractal dimension error: 0.0
worst radius: 0.0
worst texture: 0.003917173107059755
worst perimeter: 0.7940915078230686
worst area: 0.010001608298821924
worst smoothness: 0.0
worst compactness: 0.0
worst concavity: 0.0
worst concave points: 0.10569731608856218
worst symmetry: 0.0
worst fractal dimension: 0.0

Model Efficiency score: 0.7319263583295914

The best 5 features are:
worst perimeter : 0.7940915078230686
worst concave points : 0.10569731608856218
mean texture : 0.05776928953399541
mean fractal dimension : 0.018254787295034974
compactness error : 0.010268317853457172

Final Score: 0.5859564164648912
```

Observations

It is to note the above output derived is from the Train: Validate: Test ratio of 70: 15: 15. Upon multiple iteration of change of this hyperparameter, it is noted that the Decision Tree Regressor is providing range of outputs that are falling out of pattern when the validation score (Model Efficiency score) and the test score (Final Score) are compared with one another.

Train: Validation: Test (Ratio)	Validation Score (Model Efficiency Score)	Model Testing Score (Final Score)	List of top 5 best features
70: 15: 15	0.7319263583295914	0.5859564164648912	worst perimeter; worst concave points; mean texture; mean fractal dimension; compactness error
50: 25: 25	0.4964539007092198	0.6984126984126984	worst concave points; area error; mean concave points; radius error; worst texture
40: 42: 18	0.6461538461538461	0.6915584415584415	worst concave points; worst perimeter; mean fractal dimension; worst fractal dimension; area error
95: 2.5: 2.5	0.5679012345679013	0.7583333333333333	worst radius; worst concave points; mean texture; mean concavity; mean radius

The above table indicates the validation that the Decision Tree Regression isn't an ideal choice to perform regression on this dataset. It is due to the results that vary in disproportion to the varying hyperparameters. With the certain cases where the training is very high and validation is low or when training is low and validation is the highest, the data points being output cannot be linearly matched for a pattern. Due to this, the list of selected features for each iteration is varying as well. Ideally, the **best-case example** would be the Split ratio of data of **70:15:15** and the **worst** would be **95:2.5:2.5**.
The suggestion(s) that one can provide for model improvement would be:

1. To define the problem further by collecting further information upon the desired output and the original behavior of the data along with trends and patterns.
2. To elaborately define the Decision Tree Regressor along with the path to be travelled by assigning the number of leaves.
3. To assign weights to the features and converting the solution to this problem via a more biased method.
4. To change the algorithm altogether and use a linear model to derive better results.

Problem Statement 5

Question

The above problem statements are to be conducted here as well for the cancer dataset and there is a need for the following:

1. 3 plots generation for features that are doing a good job.
2. 3 plots generation for features that are doing a poor job.

By separating classes for both. Added to this, the original dataset is to be split into Training, Validation and Testing. Upon this split dataset Linear Logistic Regression (with L1 Regularization) is to be performed to find a maximum 5 good features. These features will be provided as an input to a classifier model that classifies examples as benign and malignant.

Code

THE ENTIRE CODE BLOCK FOR THE PROBLEM STATEMENT 5 CAN BE FOUND IN THE APPENDIX OF THIS DOCUMENT.

Output

The program runs over the dataset of Breast Cancer from the *SKLearn* repository. The following are the plots obtained for the three good and three poor performing features. Both the features that perform a good job and a poor job are separated by classes and are plotted alongside each other.

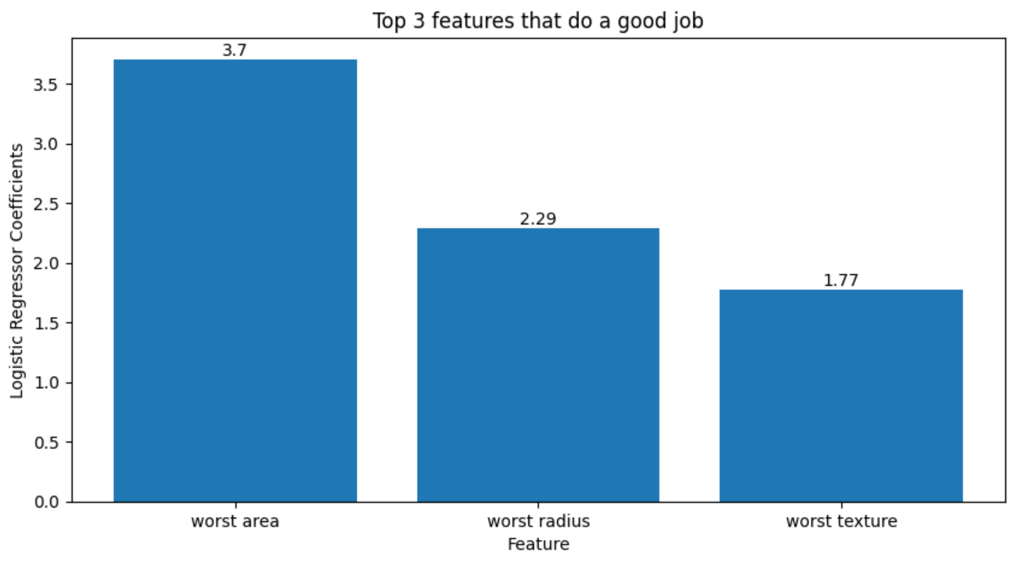

Figure 4: Three features that do a good job as per their Logistic Regressor Coefficients

Figure 4 has the score of each feature's logistic regressor coefficient upon the bar chart plotted The model has selected "Worst Area", "Worst Radius Feature" and "Worst Texture" to be the features that do a good job in predicting disease progression.

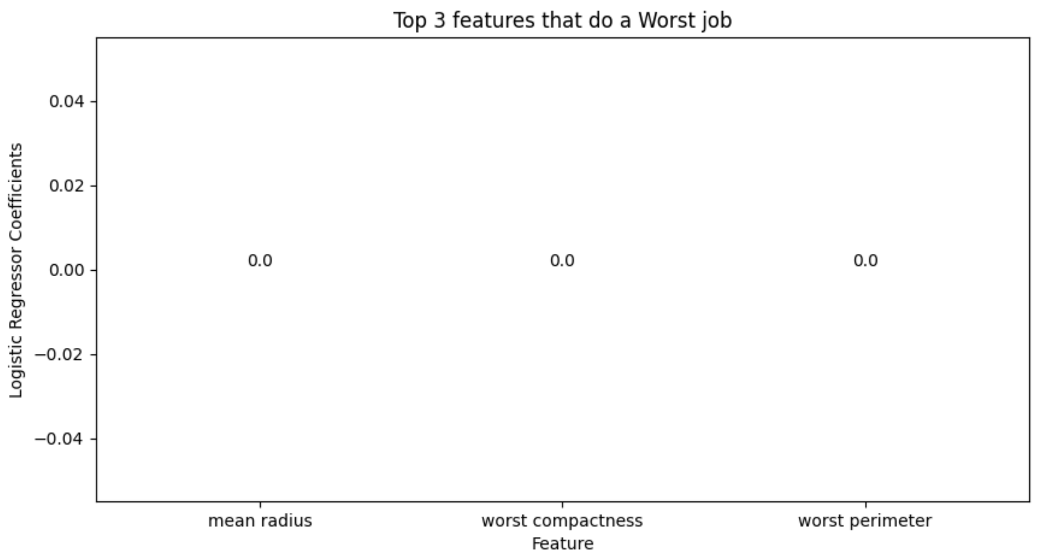

Figure 5:Three features that do a poor job as per their Logistic Regressor Coefficients

Figure 5 has the score of each feature's logistic regressor coefficient upon the bar chart plotted The model has selected "Mean Radius", "Worst Compactness Feature" and "Worst Perimeter" to be the features that do a poor job in predicting disease progression.

```
All Features:
mean radius: 0.0
mean texture: 0.0
mean perimeter: 0.0
mean area: 0.0
mean smoothness: 0.0
mean compactness: 0.0
mean concavity: 0.0
mean concave points: 1.6114609577869021
mean symmetry: 0.0
mean fractal dimension: 0.19724474369599282
radius error: 0.0
texture error: 0.1795119784644025
perimeter error: 0.0
area error: 0.0
smoothness error: 0.493253149464584
compactness error: 0.0
concavity error: 0.0
concave points error: 0.0
symmetry error: 0.0
fractal dimension error: 0.0
worst radius: 2.2881846361356297
worst texture: 1.7711170581707862
worst perimeter: 0.0
worst area: 3.6999390107548695
worst smoothness: 0.5976160838695855
worst compactness: 0.0
worst concavity: 0.5131897144609489
worst concave points: 0.6792700290806485
worst symmetry: 1.0316350918389607
worst fractal dimension: 0.0

Model Efficiency score: 0.9473684210526315

The best 5 features are:
worst area : 3.6999390107548695
worst radius : 2.2881846361356297
worst texture : 1.7711170581707862
mean concave points : 1.6114609577869021
worst symmetry : 1.0316350918389607

Final Score: 0.9736842105263158

Final score upon KNN Classification 0.9298245614035088

Example Values:
Benign:
['worst area: 1031.0', 'worst radius: 18.55', 'worst texture: 25.09', 'mean concave points: 0.09052', 'worst symmetry: 0.277']
['worst area: 783.6', 'worst radius: 16.01', 'worst texture: 28.48', 'mean concave points: 0.04951', 'worst symmetry: 0.2369']
['worst area: 803.7', 'worst radius: 16.11', 'worst texture: 29.11', 'mean concave points: 0.02957', 'worst symmetry: 0.2522']

Malignant:
['worst area: 591.2', 'worst radius: 14.19', 'worst texture: 24.85', 'mean concave points: 0.05603', 'worst symmetry: 0.3113']
['worst area: 347.3', 'worst radius: 10.65', 'worst texture: 22.88', 'mean concave points: 0.01116', 'worst symmetry: 0.2262']
['worst area: 663.5', 'worst radius: 14.73', 'worst texture: 21.7', 'mean concave points: 0.01473', 'worst symmetry: 0.2741']
```

The above is also the output from the program upon the terminal. Here, the score of all features is first printed and there after the Model Efficiency Score is printed.

This Model Efficiency score is the score based on the Validation split of the dataset. The five best features are then selected based on the score from the All-features list. The Final score is the based on the Testing split of the dataset. The top three example values for Benign and Malignant is also provided upon using K Nearest Neighbor Classifier (with n_neighbors = 5).

Observations

The flow of the program with the plotting of the good and poor features to the validation score and then the testing score, exemplifies that the data learning curve by the model has improved based on the ratio split of the Training, Validation and Testing.
The benign and malignant set of features after the K Nearest Neighbor classification is also displayed along with score to the top three values in each category.

Project 2 Conclusion

Project 2 has been the most challenging task that has been endured so far. The convolution in understanding and further continued learnings from each problem statement has not only been frustrating but also fruitful towards understanding each model and the concept of splitting data for Training, Validation and Testing in Machine Learning. The problems revolve only around this splitting concept and helps one solidify their reasoning upon the knowledge of these Machine Learning concepts.

Appendix

Problem Statement 1

Code

```
import matplotlib.pyplot as plt
from sklearn.datasets import load_diabetes
from sklearn.feature_selection import f_regression

#Loading the diabetes dataset
diabetes = load_diabetes()

#Obtaining the feature and target data
ft = diabetes.data
trgt = diabetes.target

#Computing scores from regression for strong linear and non-linear and weak
relationship
scores, x = f_regression(ft, trgt)

#Fetching the indices for the 3 types of relationships based on score
str_l_idx = scores.argmax()
str_nl_idx = scores.argsort()[-2]
wk_idx = scores.argmin()

#Plot 1: One feature with strong linear relationship with Disease progression
plt.scatter(ft[:, str_l_idx], trgt)
plt.xlabel('Feature {}'.format(str_l_idx))
plt.ylabel("Disease Progression")
plt.title('Feature with Strong Linear Relationship')
plt.show()

#Plot 2: One feature with strong non-linear relationship with Disease progression
plt.scatter(ft[:, str_nl_idx], trgt)
plt.xlabel('Feature {}'.format(str_nl_idx))
plt.ylabel("Disease Progression")
plt.title('Feature with Strong Non-Linear Relationship')
plt.show()

#Plot 3: One feature with Weak relationship with Disease progression
plt.scatter(ft[:, wk_idx], trgt)
plt.xlabel('Feature {}'.format(wk_idx))
plt.ylabel("Disease Progression")
plt.title('Feature with Weak Relationship')
plt.show()
```

Problem Statement 2

Code

```
from sklearn.model_selection import train_test_split
from sklearn.datasets import load_breast_cancer

#Initialzing the dataset
data = load_breast_cancer()

#Assigning features and target to the dataset
x = data.data
y = data.target

#Splitting ratios
train = 0.7
val = 0.15
test = 0.15

#Splitting the dataset for training and temporary set
x_train, x_temp, y_train, y_temp = train_test_split(x, y, train_size = train, 
random_state = 40)

#Splitting the remaining dataset into validation and testing set
x_val, x_test, y_val, y_test = train_test_split(x_temp, y_temp, test_size = 0.5, 
random_state = 40)

#Printing the shapes of the resulting datasets to confirm the split
print("Training set shape:", x_train.shape, y_train.shape)
print("Validation set shape:", x_val.shape, y_val.shape)
print("Testing set shape:", x_test.shape, y_test.shape)
```

Problem Statement 3

Code

```python
from sklearn.datasets import load_breast_cancer
from sklearn.linear_model import Lasso
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler

#Loading dataset, initializzing and splitting into training, validation and testing
data = load_breast_cancer()
ftr = data.data
trgt = data.target
ftr_train, ftr_test, trgt_train, trgt_test = train_test_split(ftr, trgt, test_size
= 0.2, random_state = 50)
ftr_train, ftr_val, trgt_train, trgt_val = train_test_split(ftr_train, trgt_train,
test_size = 0.25, random_state = 25)

#Scaling the features
scaler = StandardScaler()
ftr_train_scaled = scaler.fit_transform(ftr_train)
ftr_val_scaled = scaler.transform(ftr_val)
ftr_test_scaled = scaler.transform(ftr_test)

#Creation of Lasso regression model and fitting the data
lasso = Lasso(alpha = 0.2)
lasso.fit(ftr_train_scaled, trgt_train)

#Obtaining the lasso coeffiencents of 5 best features
coef = lasso.coef_
top_indices = coef.argsort()[-5:][::-1]
top_features = data.feature_names[top_indices]
feature_score = coef[top_indices]

#Calculating model efficiency score
score = lasso.score(ftr_val_scaled, trgt_val)

# Get all feature scores
all_feature_scores = coef

print("All Features:")
for feature, scr in zip(data.feature_names, all_feature_scores):
    print(f"{feature}: {scr}")

#Printing the results
print("\nModel Efficiency score:", score)
print("\nThe best 5 features are:")
for feature, scr in zip(top_features, feature_score):
    print(f"{feature} : {scr}")

#Applying the test data to the model and printing the final score
final_score = lasso.score(ftr_test_scaled, trgt_test)
print("\nFinal Score:", final_score)
```

Problem Statement 4

Code

```
import numpy as np
from sklearn.datasets import load_breast_cancer
from sklearn.tree import DecisionTreeRegressor
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler

#Loading dataset, initializzing and splitting into training, validation and testing
data = load_breast_cancer()
ftr = data.data
trgt = data.target
ftr_train, ftr_test, trgt_train, trgt_test = train_test_split(ftr, trgt, test_size
= 0.2, random_state = 50)
ftr_train, ftr_val, trgt_train, trgt_val = train_test_split(ftr_train, trgt_train,
test_size = 0.25, random_state = 25)

#Creation of Decision Tree regression model and fitting the data
dtr = DecisionTreeRegressor()
dtr.fit(ftr_train, trgt_train)

#Obtaining the feature importances of 5 best features
imp = dtr.feature_importances_
top_indices = np.argsort(imp)[-5:][::-1]
top_features = data.feature_names[top_indices]
feature_scores = imp[top_indices]

#Calculating model efficiency score
score = dtr.score(ftr_val, trgt_val)

# Get all feature scores
all_feature_scores = dtr.feature_importances_

print("All Features:")
for feature, scr in zip(data.feature_names, all_feature_scores):
    print(f"{feature}: {scr}")

#Printing the results
print("\nModel Efficiency score:", score)
print("\nThe best 5 features are:")
for feature, scr in zip(top_features, feature_scores):
    print(f"{feature} : {scr}")

#Applying the test data to the model and printing the final score
final_score = dtr.score(ftr_test, trgt_test)
print("\nFinal Score:", final_score)
```

Problem Statement 5

Code

```python
import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets import load_breast_cancer
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.neighbors import KNeighborsClassifier

#Loading dataset, initializzing and splitting into training, validation and testing
data = load_breast_cancer()
ftr = data.data
trgt = data.target
ftr_train, ftr_test, trgt_train, trgt_test = train_test_split(ftr, trgt, test_size = 0.2, random_state = 50)
ftr_train, ftr_val, trgt_train, trgt_val = train_test_split(ftr_train, trgt_train, test_size = 0.25, random_state = 25)

#Scaling the features
scaler = StandardScaler()
ftr_train_scaled = scaler.fit_transform(ftr_train)
ftr_val_scaled = scaler.transform(ftr_val)
ftr_test_scaled = scaler.transform(ftr_test)

#Creation of Logistic regression model and fitting the data
lr = LogisticRegression(penalty = 'l1', solver = 'liblinear')
lr.fit(ftr_train_scaled, trgt_train)

#Obtaining the Logistic Regression coeffiencents of 5 best features
coef = np.abs(lr.coef_)
top_indices = np.argsort(coef)[0][-5:][::-1]
top_features = data.feature_names[top_indices]
feature_score = coef[0][top_indices]

#Plotting the graph for 3 features that do a good job
good_ftr_indices = top_indices[:3]
good_ftr_names = data.feature_names[good_ftr_indices]
good_ftr_scores = coef[0][good_ftr_indices]

plt.figure(figsize = (10,5))
plt.bar(good_ftr_names, good_ftr_scores)
plt.title("Top 3 features that do a good job")
plt.xlabel("Feature")
plt.ylabel("Logistic Regressor Coefficients")

#Adding score value on top of each bar
for i, j in enumerate(good_ftr_scores):
    plt.text(i, j, str(round(j, 2)), ha = 'center', va = 'bottom')

plt.show()

#Obtaining the Logistic Regression coeffiencents of 5 worst features
worst_ftr_indices = np.argsort(coef)[0][:3]
worst_ftr_names = data.feature_names[worst_ftr_indices]
worst_ftr_scores = coef[0][worst_ftr_indices]

#Plotting the graph for 3 features that do a worst job
plt.figure(figsize = (10,5))
plt.bar(worst_ftr_names, worst_ftr_scores)
plt.title("Top 3 features that do a Worst job")
plt.xlabel("Feature")
plt.ylabel("Logistic Regressor Coefficients")

#Adding score value on top of each bar
for i, j in enumerate(worst_ftr_scores):
    plt.text(i, j, str(round(j, 2)), ha = 'center', va = 'bottom')

plt.show()

#Selecting 5 best features for classification
selected_ftr = data.data[:, top_indices]

#Developing the K Nearest Neighbour Classifier using the best features
knn = KNeighborsClassifier(n_neighbors = 5)
knn.fit(selected_ftr, trgt)

#Calculating model efficiency score
score = lr.score(ftr_val_scaled, trgt_val)

# Get all feature scores
all_feature_scores = coef[0]

print("All Features:")
for feature, scr in zip(data.feature_names, all_feature_scores):
    print(f"{feature}: {scr}")

#Printing the results
print("\nModel Efficiency score:", score)
print("\nThe best 5 features are:")
for feature, scr in zip(top_features, feature_score):
    print(f"{feature} : {scr}")

#Applying the test data to the model and printing the final score
final_score = lr.score(ftr_test_scaled, trgt_test)
print("\nFinal Score:", final_score)

# Applying the test data to the classifier and printing the final score
ftr_test_selected = ftr_test[:, top_indices]
final_score = knn.score(ftr_test_selected, trgt_test)
print("\nFinal score upon KNN Classification", final_score)

#Classifying examples as benign and malignant based on prediction score
pred = knn.predict(ftr_test_selected)
benign_examples = ftr_test_selected[pred == 0][:3]
malignant_examples = ftr_test_selected[pred == 1][:3]

# Printing example values of benign and malignant cases
print("\nExample Values:")
print("Benign:")
for i in benign_examples:
    print([f"{feature}: {value}" for feature, value in zip(top_features, i)])
print("\nMalignant:")
for i in malignant_examples:
    print([f"{feature}: {value}" for feature, value in zip(top_features, i)])
```