Table of Contents

PROJECT OVERVIEW **1**

PROBLEM STATEMENT 1 .... **2**

QUESTION 2
CODE (LOGIC) 2
OUTPUT 3
OBSERVATIONS 3

PROBLEM STATEMENT 2 .... **4**

QUESTION 4
CODE (LOGIC) 4
OUTPUT 5
OBSERVATIONS 5

PROBLEM STATEMENT 3 .... **6**

QUESTION 6
CODE (LOGIC) 6
OUTPUT 7
OBSERVATIONS 7

PROBLEM STATEMENT 4 .... **8**

QUESTION 8
CODE (LOGIC) 8
OUTPUT 9
OBSERVATIONS 9

PROBLEM STATEMENT 5 .... **10**

QUESTION 10
CODE (LOGIC) 10
OUTPUT 11
OBSERVATIONS 11

PROJECT 1 CONCLUSION **12**

APPENDIX **13**

PROBLEM STATEMENT 1 13
Code *13*
PROBLEM STATEMENT 2 14
Code *14*
PROBLEM STATEMENT 3 15
Code *15*
PROBLEM STATEMENT 4 16
Code *16*
PROBLEM STATEMENT 5 17
Code *17*

Table of Figures

FIGURE 1: THE BEST FIT CURVE LINE FOR THE SCATTER PLOT OF THE DATA FROM THE ADVERTISING.CSV DATASET. 3

Project Overview

The project 1 is the initial approach into the world of Machine Learning. This involves but is not limited to designing model(s) as a tool to solve a problem of a particular nature. The designing of this model would include, writing and coding the logic for the model, training the model on a dataset for the model to consider boundary conditions. This model will be then checked for its robustness based on a scoring scale that is used to determine its overall effectiveness.

The project 1 consists of the following sub problems that are to be solved:

1. The prediction of return on investment into radio advertising in terms of unit sales.
    a. The problem expects a best fit curve, linear model coefficients and estimated number of units sold for a set budget.
    b. Additionally, the problem also expects the comparison of the model to a gradient decent example.
2. The problem focusses on the Decision Tree Regressor Algorithm that is to be implemented upon a medical Ski-kit Learn dataset to predict the diabetes progression based on a subset of medical features.
    a. The model is expected to list the seed value, names of the features, depth of the tree, the number of leaves of the tree, and the decision tree coefficient of the determination for the two best and worst combinations of the features.
    b. Additionally, the observations of the altering the seed value yielding towards the changes observed in the model's behavior is to be noted. Added to this, suggestions for improvement of the score, try an increase in the number of features to achieve higher scores (if possible), tradeoffs to the model in adding more features (if any), and choice of other hyperparameters to improve score is to be recorded.
3. This problem statement expects the use of logistic regression algorithm to predict the presence of cancer based on the processed medical image data in the breast cancer dataset. The data is to be extracted from the Scikit-Learn dataset.
    a. The model is expected to deliver the start seed value for the pseudo-random number generator and to provide two of the best and two of the worst combinations of features.
    b. Additionally, the problem also expects the theoretical understanding, comparisons, and suggestions for improvement for the model developed.
4. This problem has the same procedure as the above logistic regression problem. However, here the expectation is to implement the linear Support Vector Algorithm upon the Ski-kit learn dataset.
    a. The model is expected to deliver the start seed value for the pseudo-random number generator and to provide two of the best and two of the worst combinations of features.
    b. Additionally, a comparison needs to be drafted between logistic regression and the linear SVC in terms of the variance in the seed value chosen, try an increase in the number of features to achieve higher scores (if possible), shortcomings of this algorithm against Logistic regression, and choice of other hyperparameters to improve score.
5. The problem focusses on the K Neighbors Regressor algorithm that is to be implemented upon a medical Ski-kit Learn dataset to predict the diabetes progression based on a subset of medical features. The procedure of implementation is same as the decision tree regressor.

a. The problem also expects the to state the seed value chosen, names of the features, number of neighbors, and the K Nearest Neighbor coefficient of determination score for the two best and two worst combinations of features.
b. Additionally, the observations of the altering the seed value yielding towards the changes observed in the model's behavior is to be noted. Added to this, suggestions for improvement of the score, try an increase in the number of features to achieve higher scores (if possible), and choice of other hyperparameters to improve score is to be recorded.

Problem Statement 1

Question

Python code that uses the Scikit-Learn Linear Regression algorithm to predict the number of unit sales given an amount of money spent on radio advertising. You can extract the radio advertising data from the Advertising CSV file as follows (there are other approaches, so do what works for you):

```
df = pd.read_csv("Advertising.csv")
x_arr = df["radio"].to_numpy()
y_arr = df["sales"].to_numpy()
x_arr = x_arr.reshape(-1,1)
```

Since we are only interested in the best fit curve, you may use all the data for training.
Outputs: State the linear model coefficients determined by Linear Regression and the estimated # of units sold given a radio advertising budget of 23 M$ (units used for the radio advertising data).
Observations: How well do your coefficients match the results from our Gradient Descent example? Which method is faster for training the model?

Code (Logic)

The logic of the code for the problem statement 1 can be defined by the following stages:

1. Model initialization and model training:

```
model = LinearRegression()
model.fit(x_arr,y_arr)
```

Here the *model* variable is initialized with the *Linear Regression* algorithm from the scikit-learn library. Subsequently, the model is then trained by fitting it with the respective input and output arrays. This helps in finding the best fit line that establishes a relationship between the budget allotted for radio marketing and the sales that can be predicted for the same based on the dataset.

2. The calculation of coefficient and intercept:

```
coefficients = model.coef_[0]
intercept = model.intercept_
```

At this stage, the slope of the best fit curve (line) is calculated by calculating the coefficients. And the intercept along this line is subsequently calculated. Mathematically, in the slope calculation equation $y = mx + b$, the slope of the line is m and the intercept is b.

3. Prediction of sales from the trained model:

```
budget = 23
estimated_sales = model.predict([[budget]])
```

The final stage is where the budget variable is assigned to create a start point for the model to predict the quantity of sales. As per the problem statement, the budget allotted is $ 23 Millions.

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

Output

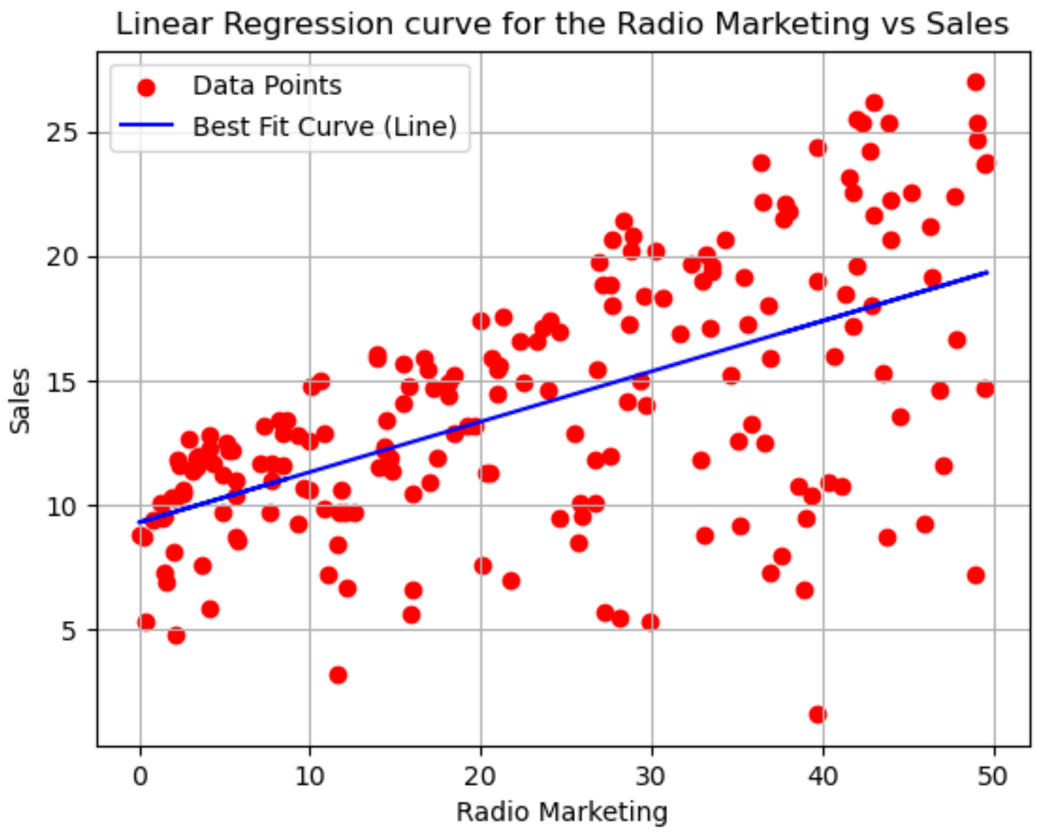

Figure 1: The Best Fit Curve Line for the scatter plot of the data from the Advertising.csv dataset.

Figure 1 represents the best fit curve for the data provided from the dataset. The graph is on the rising trend, i.e. with the increase in marketing via radio, sales has also exponentially increased to the most part (excluding a certain group of outliers).
The following output from the program code for problem statement 1 is as shown below:

```
Estimated Sales: 13.9690
Linear Model Coefficient (Slope): 0.2025
Linear Model Coefficient (Intercept): 9.3116
```

The output provides the linear model coefficients (Slope and Intercept) and the predicted sales for the given budget value.

Observations

As per the w and b scores from the Gradient Descent example after 15000 epochs, the difference in the value of the coefficients is largely insignificant (slight changes only after 10^{-5}). However, considering the computation power required and efficiency in the larger sense, Linear Regression fares far better. Considering the time complexity of both the approaches, Linear Regression uses an optimized approach when considering larger datasets.

Problem Statement 2

Question

Python code that uses the Scikit-Learn Logistic Regression algorithm to predict the presence of cancer based on the processed medical image data in the breast cancer dataset. For this part of the project, we will randomly select a subset of features for our classification problem. Use the following code to adjust the seed for the pseudo-random number generator that will randomly select 4 features:

```
seed = 123456
rng = np.random.default_rng(seed)
idx_feat = (np.floor(30*rng.uniform(size=4))).astype(int)
X = cancer["data"][:,idx_feat]
```

Again, you may use the entire set of examples in the dataset for training the model. There is no need to set aside a test set for this project.

Outputs: State the seed value, the feature names, and the logistic regression score for the two best and two worst combinations of features found during your testing. Don't worry about finding the optimal solution. We are only interested in gaining some familiarity with the variation in the accuracy of the model.

Questions: As you varied the "seed" value, what changes did you notice in the random selection of features? Were any of the features better in terms of increasing the logistic regression score? If you increased the number of features selected, would you get higher scores? What trade-off would you be making if you selected more features? What other hyperparameters could you tune to improve the scores?

Code (Logic)

The logic of the code for the problem statement 1 can be defined by the following stages:

1. Model initialization and model training:

```
model = LogisticRegression()
model.fit(x,y)
```

Here the *model* variable is initialized with the *Logistic Regression* algorithm from the scikit-learn library. Subsequently, the model is then trained by fitting it with the respective input and output arrays.

2. Deriving the best and the worst combination of features:

```
scores = []
for i in range(4):
    for j in range(4):
        if i != j:
            selected_features = [feature_names[i], feature_names[j]]
            selected_indices = [i,j]
            selected_x = x[:, selected_indices]
            model.fit(selected_x,y)
            score = model.score(selected_x,y)
            scores.append((selected_features, score))
```

Here, the scores array is initialized and a nested for loop is introduced. The key intent with the if statement inside the looping construct is to avoid the combination of two features from among the best set or the worst set is to be the same. Here the model is fit with the selected feature and the score is also calculated for the combination. The append method helps in combining the score for that set of features.

3. Sorting and segregating the output of best and worst combination of features:

```
scores.sort(key=lambda x: x[1], reverse=True)
best_selected_features, best_score  = scores[0]
worst_selected_features, worst_score  = scores[len(scores)-1]
```

The scores array is sorted in descending order of the scores received. Subsequently, the best and the worst score is determined, saved and then the output is provided.

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

Output

```
Seed Value: 123059
Selected Features: ['worst symmetry' 'worst texture' 'concave points error'
 'worst concave points']
Best combination of features: ['worst symmetry', 'worst concave points']
Best - Logistic Regression Score: 0.8418277680140598
Worst combination of features: ['concave points error', 'worst symmetry']
Worst - Logistic Regression Score: 0.6748681898066784
```

The output states that the seed value chosen is *123059* and the four random features that the model has chosen from the dataset are *worst symmetry, worst texture, concave points error, and worst concave points.* Post computation, the model has chosen the best combination of features to be *worst symmetry* and *worst concave points* with a score of *0.8418* and the worst combination of features to be *concave points error* and *worst symmetry* with a score of *0.6748*.

Observations

As the seed value was varied, it was noted that when the seed value decreased, the score of both the best and the worst-case conditions increased. i.e. when seed value was *123059* the best-case score was *0.8418* and the worst-case score was *0.6748.* When seed value was decreased to *123000* the best-case score was *0.9261* and the worst-case score was *0.8400*. Similarly, when the seed value increased, the score of both the best and the worst-case conditions decreased. i.e. When seed value was increased to *124000* the best-case score was *0.9086* and the worst-case score was *0.7996*.

However, it is imperative to note that both the best and the worst-case scenario scores increased than the set seed value of *123059*, when the seed value was increased and decreased.

When the number of features was doubled to a value of *8* from its original value *4*, it was observed that the difference between the values of the best-case and the worst-case scenarios was further apart. The new change in best-case and worst-case scenarios was *0.9173* and *0.6274* respectively. The tradeoff here would be that the model would start to lean farther away from the mean of the data set and increase the standard deviation, causing the reduction in the robustness of the trained model.

The other hyper parameter that can be adjusted to further improve the score would be to increase the number of features in the set of best case and worst-case scenarios. When the number of features for each best and worst-case scenarios was programmed to be three in place of two, it was observed that the best-case scenario witnessed an improvement in score, however, the worst-case scenario remained the same.

Problem Statement 3

Question

Python code that uses the Scikit-Learn Decision Tree Regressor algorithm to predict diabetes progression based on a subset of medical features. Similar to the Logistic Regression task above, use the pseudo-random number generator and "seed" to randomly select 4 features for building a decision tree. Use the entire set of examples in the dataset for training the model.

Outputs: State the seed value, the feature names, the depth of the tree, the number of leaves of the tree and the decision tree coefficient of determination score for the two best and two worst combinations of features found during your testing. Don't worry about finding the optimal solution.

Questions: As you varied the "seed" value, what changes did you notice in the random selection of features? Were any of the features better in terms of increasing the decision tree score? If you increased the number of features selected, would you get higher scores? What trade-off would you be making if you selected more features? What other hyperparameters could you tune to improve the scores?

Code (Logic)

The logic of the code for the problem statement 1 can be defined by the following stages:

1. Model initialization and model training:

```
model = DecisionTreeRegressor()
model.fit(x,y)
```

Here the *model* variable is initialized with the *Decision Tree Regressor* algorithm from the scikit-learn library. Subsequently, the model is then trained by fitting it with the respective input and output arrays.

2. Deriving the best and the worst combination of features:

```
scores = []
for i in range(4):
    for j in range(4):
        if i != j:
            selected_features = [feature_names[i], feature_names[j]]
            selected_indices = [i,j]
            selected_x = x[:, selected_indices]
            model.fit(selected_x,y)
            score = model.score(selected_x,y)
            scores.append((selected_features, score))
```

Here, the scores array is initialized and a nested for loop is introduced. The key intent with the if statement inside the looping construct is to avoid the combination of two features from among the best set or the worst set is to be the same. Here the model is fit with the selected feature and the score is also calculated for the combination. The append method helps in combining the score for that set of features.

3. Sorting and segregating the output of best and worst combination of features:

```
scores.sort(key=lambda x: x[1], reverse=True)
best_selected_features, best_score  = scores[0]
worst_selected_features, worst_score  = scores[len(scores)-1]
```

The scores array is sorted in descending order of the scores received. Subsequently, the best and the worst score is determined, saved and then the output is provided.

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

Output

```
Seed Value: 125690
Selected Features: ['age', 'sex', 'bmi', 'bp', 's1', 's2', 's3', 's4', 's5', 's6']
Depth of the Tree: 18
Number of leaves of the Tree: 435
Best combination of features: ['sex', 'bmi']
Best - Decision Tree Regressor Score: 0.9904737073338544
Worst combination of features: ['bp', 'sex']
Worst - Decision Tree Regressor Score: 0.9079287193558537
```

The output states that the seed value chosen is *125690* and the random features from which the model has chosen best and worst-case scenarios are *age, sex, bmi, bp, s1, s2, s3, s4, s5, and s6.* Post computation, the depth of the tree is *18* and the number of leaves of the tree is *435*. Also, the model has chosen the best combination of features to be *sex* and *bmi* with a score of *0.9904* and the worst combination of features to be *bp* and *sex* with a score of *0.9079*.

Observations

As the seed value was varied, it was noted that when the seed value decreased the best-case scenario started to perform well but the worst-case scenario took a hit and dropped its score. i.e. When the seed value was *125690* the best and the worst-case was *0.9904* and *0.9079* respectively and with the decrease in seed value to *122486*, the best and the worst-case scenario changed to *0.9975* and *0.7048* respectively. Additionally, it was further surprising to witness that when the seed value was changed to *128990* the best and worst-case scenario changed to *0.9316* and *0.0018*. This indicated that the best-case scenario what is indirectly proportional to the change in seed value. But the worst-case scenario continued to drop in value with either increase or decrease in the value of the seed.

With the change in the seed value, there was change observed in the random features that was selected, however, this change in the features was an expected result. But the significance of the change in features can only be matched while comparing the score received for the combination of these features in the best and the worst-case scenarios.

The increase in number of features from four was done to a value of nine. The depth of the tree and the number of leaves of the tree remained same. However, there was a change observed in the combination of features that was selected for the best and the worst-case scenarios and the subsequent two scores changed as well to *0.9975* and *0.1409* respectively.

Here it clearly indicates that the best-case scenario again improves with a change (increase) in the number of features however the worst-case scenario still takes a hit. The trade-off will be that the worst-case scenario continues to deteriorate and provide the most undesired scores as there is a change with the number of features selected.

When the number of features for each set of combinations (best or worst-case scenarios) was made to increase from two per set to three per set, unlike the other trials explained in the observation above, both the scores in this case (best and worst-case scenarios) performed extremely well with the best-case scenario having a score of *1.0* and the worst-case scenario having a score of *0.9079*, while still retaining the original value of random features being four. This proves that the other hyperparameter of choosing a larger set of combination of features to segregate into best and worst-case scenarios will be fruitful for this algorithm to work in its maximum efficiency.

Problem Statement 4

Question

Python code that uses the Scikit-Learn Support Vector Machine algorithm LinearSVC to predict the presence of cancer based on the processed medical image data in the breast cancer dataset. Follow the same procedure as was used for Logistic Regression.

Outputs: State the seed value, the feature names, and the LinearSVC score for the two best and two worst combinations of features found during your testing. Don't worry about finding the optimal solution.

Questions: As you varied the "seed" value, what changes did you notice in the random selection of features? Were any of the features better in terms of increasing the LinearSVC score? If you increased the number of features selected, would you get higher scores? What trade-off would you be making if you selected more features? What other hyperparameters could you tune to improve the scores?

Code (Logic)

The logic of the code for the problem statement 1 can be defined by the following stages:

1. Model initialization and model training:

```
model = LinearSVC()
model.fit(x,y)
```

Here the *model* variable is initialized with the *Linear Support Vector Machine* algorithm from the scikit-learn library. Subsequently, the model is then trained by fitting it with the respective input and output arrays.

2. Deriving the best and the worst combination of features:

```
scores = []
for i in range(4):
    for j in range(4):
        if i != j:
            selected_features = [feature_names[i], feature_names[j]]
            selected_indices = [i,j]
            selected_x = x[:, selected_indices]
            model.fit(selected_x,y)
            score = model.score(selected_x,y)
            scores.append((selected_features, score))
```

Here, the scores array is initialized and a nested for loop is introduced. The key intent with the if statement inside the looping construct is to avoid the combination of two features from among the best set or the worst set is to be the same. Here the model is fit with the selected feature and the score is also calculated for the combination. The append method helps in combining the score for that set of features.

3. Sorting and segregating the output of best and worst combination of features:

```
scores.sort(key=lambda x: x[1], reverse=True)
best_selected_features, best_score  = scores[0]
worst_selected_features, worst_score  = scores[len(scores)-1]
```

The scores array is sorted in descending order of the scores received. Subsequently, the best and the worst score is determined, saved and then the output is provided.

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

Output

```
Seed Value: 210058
Selected Features: ['worst fractal dimension' 'concavity error' 'worst concavity'
 'concave points error']
Best combination of features: ['worst fractal dimension', 'worst concavity']
Best - LinearSVC Score: 0.8488576449912126
Worst combination of features: ['concave points error', 'concavity error']
Worst - LinearSVC Score: 0.6274165202108963
```

The output states that the seed value chosen is *210058* and the four random features that the model has chosen from the dataset are *worst fractal dimension, concavity error, worst concavity, and concave points error.* Post computation, the model has chosen the best combination of features to be *worst fractal dimension* and *worst concavity* with a score of *0.8488* and the worst combination of features to be *concave points error* and *concavity error* with a score of *0.6274*.

Observations

As the seed value was varied, it was noted that when the seed value decreased to *200000*, the score of both the best-case scenario increased while there was a decrease in their score of the worst-case scenario, i.e., the best-case scenario score changed from *0.8488* to *0.7434* and the worst-case scenario score changed from *0.6274* to *0.6414*. On the contrary, when the seed value was increased to *270000* the best-case scenario and the worst-case scenario both increased in value to *0.9420* and *0.8453* respectively.

When the number of features was doubled to a value of *8* from its original value *4*, it was observed that the best-case scenario score remained the same while there was a decrease in the worst-case scenario from *0.6274* to *0.5008.* This clearly indicates that a change (increase) in the number of features that are considered, plays little effect upon the best-case scenario while the worst-case scenario deteriorates. The trade-off here is a reduction in the efficiency of the model, since the gap between the best-case scenario and the worst-case scenario widens with an increase in the number of features.

The other hyper parameter that can be altered here would be the increase of the set of features from two to three. Considering this scenario for the same see the value of *210058*, it can be observed that the best-case scenario rises slightly from *0.8488* to *0.8927*, and the worst-case scenario falls rather steeply in comparison to the best-case scenario from *0.6274* to *0.3743*. The observation here is that with individual changes made to certain set of hyper parameters is rather slightly insignificant and only a bigger change towards improvement can be observed when a combination of many hyper parameters or altered.

Problem Statement 5

Question

Python code that uses the Scikit-Learn KNeighborsRegressor algorithm to predict diabetes progression based on a subset of medical features. Follow the same procedure as was used for the Decision Tree Regressor.

Outputs: State the seed value, the feature names, the number of neighbors and the KNeighborsRegressor coefficient of determination score for the two best and two worst combinations of features found during your testing. Don't worry about finding the optimal solution.

Questions: As you varied the "seed" value, what changes did you notice in the random selection of features? Were any of the features better in terms of increasing the k-Nearest Neighbors score? If you increased the number of features selected, would you get higher scores? What trade-off would you be making if you selected more features? What other hyperparameters could you tune to improve the scores?

Code (Logic)

The logic of the code for the problem statement 1 can be defined by the following stages:

1. Model initialization and model training:

```
model = KNeighborsRegressor(n_neighbors=10)
model.fit(x,y)
```

Here the *model* variable is initialized with the *K Neighbor Regressor* algorithm from the scikit-learn library. Subsequently, the model is then trained by fitting it with the respective input and output arrays. It is also important to note that the number of nearest neighbors is assigned as *10*.

2. Deriving the best and the worst combination of features:

```
scores = []
for i in range(4):
    for j in range(4):
        if i != j:
            selected_features = [feature_names[i], feature_names[j]]
            selected_indices = [i,j]
            selected_x = x[:, selected_indices]
            model.fit(selected_x,y)
            score = model.score(selected_x,y)
            scores.append((selected_features, score))
```

Here, the scores array is initialized and a nested for loop is introduced. The key intent with the if statement inside the looping construct is to avoid the combination of two features from among the best set or the worst set is to be the same. Here the model is fit with the selected feature and the score is also calculated for the combination. The append method helps in combining the score for that set of features.

3. Sorting and segregating the output of best and worst combination of features:

```
scores.sort(key=lambda x: x[1], reverse=True)
best_selected_features, best_score  = scores[0]
worst_selected_features, worst_score  = scores[len(scores)-1]
```

The scores array is sorted in descending order of the scores received. Subsequently, the best and the worst score is determined, saved and then the output is provided.

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

Output

```
Seed Value: 453672
Selected Features: ['age', 'sex', 'bmi', 'bp', 's1', 's2', 's3', 's4', 's5', 's6']
Number of Neighbors: 10
Best combination of features: ['age', 'sex']
Best - KNeighbour Regressor Score: 0.3413959364324939
Worst combination of features: ['bp', 'age']
Worst - KNeighbour Regressor Score: 0.21184598682166422
```

The output states that the seed value chosen is *453672* and the random features from which the model has chosen best and worst-case scenarios are *age, sex, bmi, bp, s1, s2, s3, s4, s5, and s6.* Post computation, Number of neighbors is *10*. Also, the model has chosen the best combination of features to be *age* and *sex* with a score of *0.3413* and the worst combination of features to be *bp* and *age* with a score of *0.2118*.

Observations

Since the K Neighbor regressor worked with the same dataset as that of the decision tree regressor. The original seed value for the K Neighbor regressor in *453672* which yielded the best-case scenario value of *0.3413* and the worst-case scenario value of *0.2118* was changed to a seed value of *125690* (which was the original seed value used for decision tree regressor), the seal dead a best-case scenario score of *0.4330* and a worst-case scenario score of *0.2406*. This alone helps us understand that K Neighbor regressor isn't a worthy and efficient model that can be used for the concerned dataset. To further expand the vision on the behavior of the model towards the increase of seed value to *525205*, it was observed that the best-case scenario yielded a score of *0.4240* and the worst-case scenario yielded a score of *0.2146*.

When the number of selected features was increased from four to eight, the best-case scenario score increased from *0.3413* to *0.5316*, which was promising considering its prior performance however the worst-case scenario score dropped from *0.2118* to a score of *0.1126*.

Another trial involved the change in the hyper parameters, which included the change of the number of neighbors from ten to five that which only changed the best-case scenario from *0.3413* to *0.3996* and the worst-case scenario from *0.2118* to *0.2137*. Considering this very small change the other hyper parameter that was changed was to increase the number of feature sets for the best and the worst-case scenarios which was made from two to three for which the best-case scenario remained the same while the worst-case scenario changed from *0.2118* (original seed value) to *0.2832*. Therefore, the K Neighbor regressor has proven multiple times that it is not a best suit algorithm for the given dataset.

Project 1 Conclusion

Project 1 been entirely about trial and error and the process of learning during this approach. Project 1 has also provided many facets to understanding the behavior of well-known algorithms. It has also paved the way to help one understand that the robustness of an algorithm depends upon many factors and the combination of these factors.

With the help of project 1, it was easier to learn a couple of algorithms that are used to build models within machine learning. During project 1, it was established that even the most obvious looking robust algorithms, sometimes fail when there isn't a good match with the data set or when the input parameters aren't well defined.

Project 1 has helped improve skills related to python and a well-defined practical approach to machine learning, which would have been an enigma otherwise. In large contributions of experiences grafted from project 0 along with project 1, the analytical approach towards machine learning has surpassed a rookie mindset.

Appendix

Problem Statement 1

Code

```python
import pandas as pd
from sklearn.linear_model import LinearRegression
import numpy as np
import matplotlib.pyplot as plt

#Reading data from the file
data = pd.read_csv("Advertising.csv")

#Setting input and output arrays
x_arr = data["radio"].to_numpy().reshape(-1,1)
y_arr = data["sales"].to_numpy()

#Initialzing the model and fitting the model with the defined arrays
model = LinearRegression()
model.fit(x_arr,y_arr)

#Defining the coeffiecients and intercepts
coefficients = model.coef_[0]
intercept = model.intercept_

#Setting the budget as per the problem and predicting outcome
budget = 23
estimated_sales = model.predict([[budget]])

#Printing the necessary outputs
print ("Estimated Sales:", "{:.4f}".format(estimated_sales[0]))
print ("Linear Model Coefficient (Slope):", "{:.4f}".format(coefficients))
print ("Linear Model Coefficient (Intercept):", "{:.4f}".format(intercept))

#Plotting a scatter plot for the given data and outputs with the best fit curve
plt.scatter(x_arr,y_arr, color='red', label='Data Points')
plt.plot(x_arr,model.predict(x_arr), color='blue', label='Best Fit Curve (Line)')
plt.xlabel('Radio Marketing')
plt.ylabel('Sales')
plt.title('Linear Regression curve for the Radio Marketing vs Sales')
plt.legend()
plt.grid()
plt.show()
```

Problem Statement 2

Code

```
import numpy as np
from sklearn.datasets import load_breast_cancer
from sklearn.linear_model import LogisticRegression

#Setting the seed value and defining range
seed = 123059
rnge = np.random.default_rng(seed)

#Initializing the random features of size 4
rndm_feat = (np.floor(30 * rnge.uniform(size=4))).astype(int)

#Loading the breast cancer dataset
cancer = load_breast_cancer()

#Setting the subset of features and the target
x = cancer["data"][:,rndm_feat]
y = cancer["target"]

#Logistic Regression Model instance creation and fitting the model on the data
model = LogisticRegression()
model.fit(x,y)

#Getting the feature names
feature_names = cancer["feature_names"][rndm_feat]

#Printing the seed value and the selected feature names
print("Seed Value:", seed)
print("Selected Features:", feature_names)

#Deriving rhe logistic regression value for 2 best and 2 worst combination of features
scores = []
for i in range(4):
    for j in range(4):
        if i != j:
            selected_features = [feature_names[i], feature_names[j]]
            selected_indices = [i,j]
            selected_x = x[:, selected_indices]
            model.fit(selected_x,y)
            score = model.score(selected_x,y)
            scores.append((selected_features, score))

#Sorting the scores in decending order
scores.sort(key=lambda x: x[1], reverse=True)

#Features and the score is assigned accordinly based on the combination and the score
best_selected_features, best_score  = scores[0]
worst_selected_features, worst_score  = scores[len(scores)-1]

#Printing the best combination of features and its corresponding score
print("Best combination of features:", best_selected_features)
print("Best - Logistic Regression Score:", best_score)

#Printing the worst combination of features and its corresponding score
print("Worst combination of features:", worst_selected_features)
print("Worst - Logistic Regression Score:", worst_score)
```

Problem Statement 3

Code

```
import numpy as np
from sklearn.datasets import load_diabetes
from sklearn.tree import DecisionTreeRegressor

#Setting the seed value and defining range
seed = 125690
rnge = np.random.default_rng(seed)

#Initializing the random features of size 4
rndm_feat = (np.floor(10 * rnge.uniform(size=4))).astype(int)

#Loading the diabetes dataset
diabetes = load_diabetes()

#Setting the subset of features and the target
x = diabetes["data"][:, rndm_feat]
y= diabetes["target"]

#Decision Tree Regressor Model instance creation and fitting the model on the data
model = DecisionTreeRegressor()
model.fit(x,y)

#Getting the feature names
feature_names = diabetes["feature_names"]

#Printing the seed value and the selected feature names
print("Seed Value:", seed)
print("Selected Features:", feature_names)

#Getting the depth and number of leaves of the tree
depth = model.get_depth()
num_leaves = model.get_n_leaves()

#Printing the depth and number of leaves of the tree
print("Depth of the Tree:", depth)
print("Number of leaves of the Tree:", num_leaves)

#Deriving the Decision Tree regressor value for 2 best and 2 worst combination of features
scores = []
for i in range(4):
    for j in range(4):
        if i != j:
            selected_features = [feature_names[i], feature_names[j]]
            selected_indices = [i,j]
            selected_x = x[:, selected_indices]
            model.fit(selected_x,y)
            score = model.score(selected_x,y)
            scores.append((selected_features, score))

#Sorting the scores in decending order
scores.sort(key=lambda x: x[1], reverse=True)

#Features and the score is assigned accordinly based on the combination and the score
best_selected_features, best_score  = scores[0]
worst_selected_features, worst_score  = scores[len(scores)-1]

#Printing the best combination of features and its corresponding score
print("Best combination of features:", best_selected_features)
print("Best - Decision Tree Regressor Score:", best_score)

#Printing the worst combination of features and its corresponding score
print("Worst combination of features:", worst_selected_features)
print("Worst - Decision Tree Regressor Score:", worst_score)
```

Problem Statement 4

Code

```
import numpy as np
from sklearn.datasets import load_breast_cancer
from sklearn.svm import LinearSVC

#Setting the seed value and defining range
seed = 210058
rnge = np.random.default_rng(seed)

#Initializing the random features of size 4
rndm_feat = (np.floor(30 * rnge.uniform(size=4))).astype(int)

#Loading the breast cancer dataset
cancer = load_breast_cancer()

#Setting the subset of features and the target
x = cancer["data"][:, rndm_feat]
y = cancer["target"]

#Linear SVC Model instance creation and fitting the model on the data
model = LinearSVC()
model.fit(x,y)

#Getting the feature names
feature_names = cancer["feature_names"][rndm_feat]

#Printing the seed value and the selected feature names
print("Seed Value:", seed)
print("Selected Features:", feature_names)

#Deriving the Linear SVC value for 2 best and 2 worst combination of features
scores = []
for i in range(4):
    for j in range(4):
        if i != j:
            selected_features = [feature_names[i], feature_names[j]]
            selected_indices = [i,j]
            selected_x = x[:, selected_indices]
            model.fit(selected_x,y)
            score = model.score(selected_x,y)
            scores.append((selected_features, score))

#Sorting the scores in decending order
scores.sort(key=lambda x: x[1], reverse=True)

#Features and the score is assigned accordinly based on the combination and the score
best_selected_features, best_score  = scores[0]
worst_selected_features, worst_score  = scores[len(scores)-1]

#Printing the best combination of features and its corresponding score
print("Best combination of features:", best_selected_features)
print("Best - LinearSVC Score:", best_score)

#Printing the worst combination of features and its corresponding score
print("Worst combination of features:", worst_selected_features)
print("Worst - LinearSVC Score:", worst_score)
```

Problem Statement 5

Code

```
import numpy as np
from sklearn.datasets import load_diabetes
from sklearn.neighbors import KNeighborsRegressor

#Setting the seed value and defining range
seed = 453672
rnge = np.random.default_rng(seed)

#Initializing the random features of size 4
rndm_feat = np.floor(10 * rnge.uniform(size=4)).astype(int)

#Loading the diabetes dataset
diabetes = load_diabetes()

#Setting the subset of features and the target
x = diabetes["data"][:, rndm_feat]
y = diabetes["target"]

#K Neighbour Regressor Model instance creation and fitting the model on the data
model = KNeighborsRegressor(n_neighbors=10)
model.fit(x,y)

#Getting the feature names
feature_names = diabetes["feature_names"]

#Printing the seed value and the selected feature names
print("Seed Value:", seed)
print("Selected Features:", feature_names)

#Deriving the K Neighbour Regressor value for 2 best and 2 worst combination of features
scores = []
for i in range(4):
    for j in range(4):
        if i != j:
            selected_features = [feature_names[i], feature_names[j]]
            selected_indices = [i,j]
            selected_x = x[:, selected_indices]
            model.fit(selected_x,y)
            score = model.score(selected_x,y)
            scores.append((selected_features, score))

#Sorting the scores in decending order
scores.sort(key=lambda x: x[1], reverse=True)

#Features and the score is assigned accordinly based on the combination and the score
best_selected_features, best_score  = scores[0]
worst_selected_features, worst_score  = scores[len(scores)-1]

#Printing the number of neighbours and scores
print("Number of Neighbors:", model.n_neighbors)

#Printing the best combination of features and its corresponding score
print("Best combination of features:", best_selected_features)
print("Best - KNeighbour Regressor Score:", best_score)

#Printing the worst combination of features and its corresponding score
print("Worst combination of features:", worst_selected_features)
print("Worst - KNeighbour Regressor Score:", worst_score)
```