In the name of god

Assignment 1

Machine Learning

Mehrdad Baradaran

99222020

Master : Farahani

Q1 : Can gradient descent get stuck in a local minimum when training a logistic regression model? Why?

Gradient descent is an algorithm that can be used to update model parameters in order to minimize our loss function.

According to the formula of the gradient descent algorithm, it works as follows in each iteration:

$$w = w - \alpha \frac{\partial}{\partial w} J(w, b)$$

$$b = b - \alpha \frac{\partial}{\partial b} J(w, b)$$

So this algorithm continues until the convergence of the parameters. The point is that this algorithm depends on the cost function. And by deriving this function, we can update the parameters in each step.

In the above formula, w is an array of model parameters and b is our bias.

J is also called cost function.

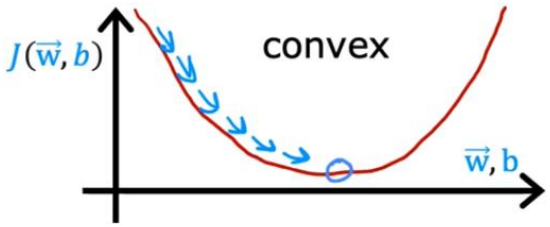
Now, if in logistic regression, our loss function is MSE, as can be seen in the picture below, unlike linear regression, the graph of the cost function will not be convex based on its parameters, which means that in addition to the global minimum, one or more local It will also have a minimum, as a result, the gradient descent algorithm may move towards the local minimum according to the initialization of the parameters.

Squared error cost

$$J(\vec{w}, b) = \frac{1}{m}\sum_{i=1}^{m}\frac{1}{2}(f_{\vec{w},b}(\vec{x}^{(i)}) - y^{(i)})^2$$

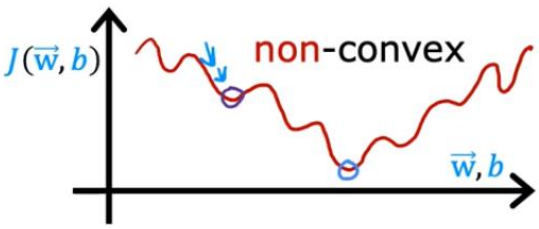
linear regression

$$f_{\vec{w},b}(\vec{x}) = \vec{w}\cdot\vec{x} + b$$



logistic regression

$$f_{\vec{w},b}(\vec{x}) = \frac{1}{1 + e^{-(\vec{w}\cdot\vec{x}+b)}}$$



For this purpose, we use a different cost function in the logistic regression model. Based on that, the cost value function has become a convex shape based on its variables, so that it has only one global minimum, and by using the gradient descent algorithm, we can converge the cost function towards it.

loss

$$L(f_{\vec{w},b}(\vec{x}^{(i)}), y^{(i)}) = -y^{(i)}\log\left(f_{\vec{w},b}(\vec{x}^{(i)})\right) - (1 - y^{(i)})\log\left(1 - f_{\vec{w},b}(\vec{x}^{(i)})\right)$$

cost

$$J(\vec{w}, b) = \frac{1}{m}\sum_{i=1}^{m}\left[L(f_{\vec{w},b}(\vec{x}^{(i)}), y^{(i)})\right]$$

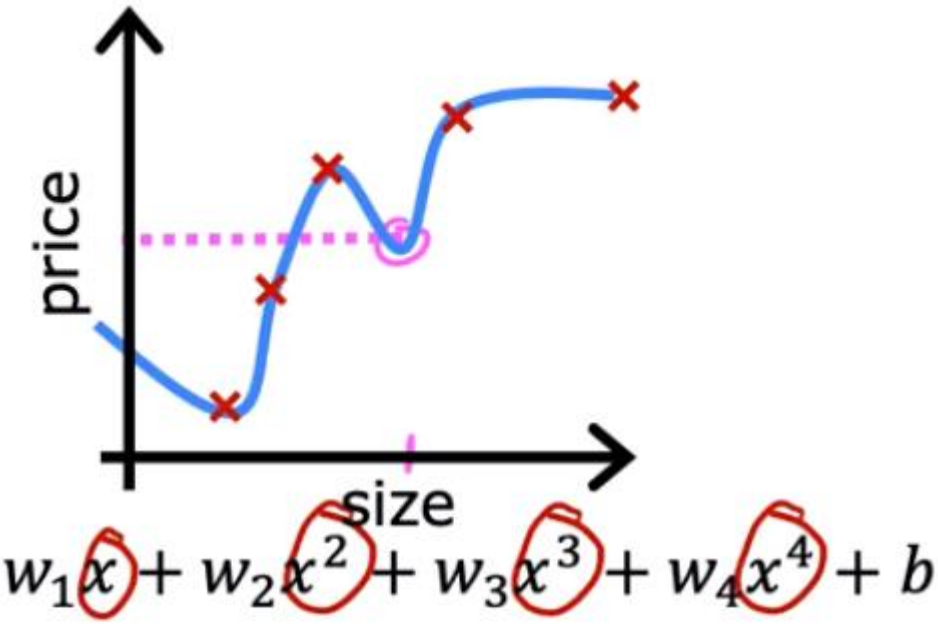
$$= -\frac{1}{m}\sum_{i=1}^{m}\left[y^{(i)}\log\left(f_{\vec{w},b}(\vec{x}^{(i)})\right) + (1 - y^{(i)})\log\left(1 - f_{\vec{w},b}(\vec{x}^{(i)})\right)\right]$$

As a result, by using this cost function, the gradient descent algorithm will not get stuck in the local minimum. And in general, it is very dependent on the cost function.

Q2 : Suppose you are using polynomial regression. You plot the learning curves and you notice that there is a large gap between the training error and the validation error. What is happening? What are three ways to solve this?

The loss in the training dataset is always lower than the validation dataset. And it means that we should expect a gap between the training and validation loss learning curves.

But the existence of a big gap between the training error and the validation error means the existence of high variance, and this happens when our model is fitted extremely well on the train data, which leads to overfitting.



overfit

- Fits the training set extremely well

high variance

Effective methods for addressing overfitting:

1 – collect more training data

But this method will not always be effective. For example, there may not be enough data to add.

2 – feature selection

In other words, it can be said that it reduced the complexity of the model.

But this method also has disadvantages and can cause good features to be lost.

3 - The last and best method to avoid overfitting is to use regularization.

In this method, we reduce the size of some parameters so that their effect is minimized, but we do not lose any feature and use all the features.

Q3 : Suppose you are using ridge regression and you notice that the training error and the validation error are almost equal and fairly high. Would you say that the model suffers from high bias or high variance? Should you increase the regularization hyper parameter α or reduce it?

When training error and the validation error are almost equal and fairly high so that underfitting occurs which means it has a high bias.

In using regularization, we add a regularization term to the cost function in addition to the previous terms so that we can tune the parameters in such a way that neither overfit nor underfit occurs.

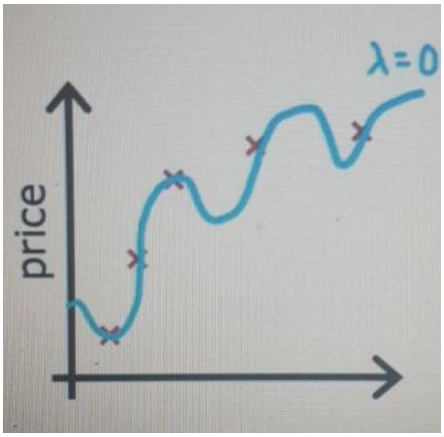
regularization term

$$+\frac{\lambda}{2m}\sum_{j=1}^{n} w_j^2$$

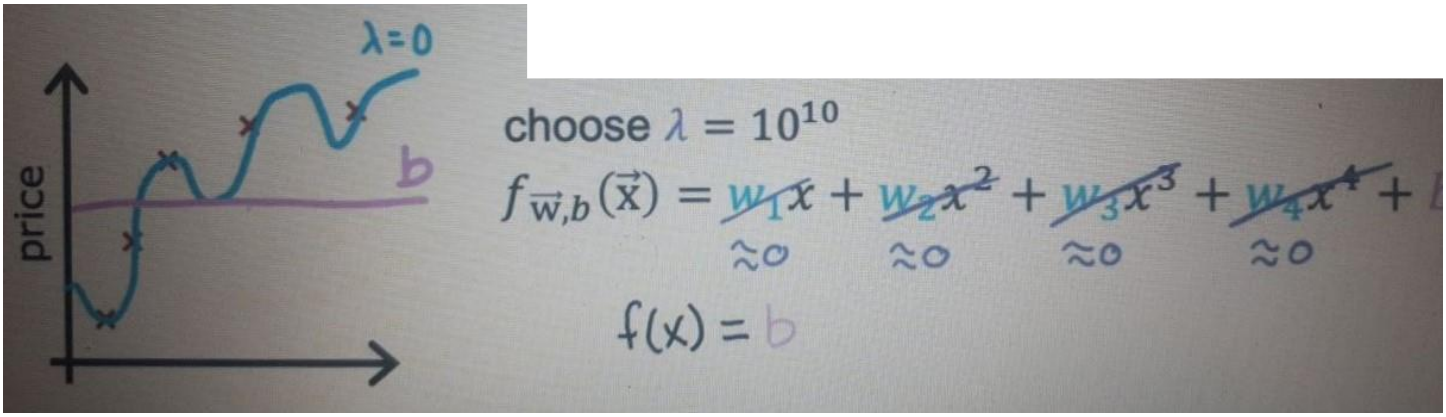
Now, with different values to the regularization hyperparameters, we show what should happen to alpha.

First of all, we would like to point out that here the lambda symbol is the same as the alpha mentioned in the question.

If lambda has a value of 0, our model reaches overfitting.



But if lambda has a very large value, then all parameters of the model tend to 0 and the model will become a constant function, as a result, underfitting occurs.



So, as the solution to fix high bias, according to the sayings, regularization hyper parameter alpha should be reduced in order to solve the problem of underfitting.

Q4 : Why would you want to use:

- Ridge regression instead of plain linear regression (i.e., without any regularization)?
- Lasso instead of ridge regression?
- Elastic net instead of lasso regression?

1. When your model is trained too much on train data, we end up with overfitting, in this case we use ridge regression. If we know that we can solve this problem with feature selection, we use lasso.

2. The working process of lasso is such that by using an L1 penalty, it can give some weights to the value of exactly 0. As a result, by implementing lasso, we will reach several sparse models. Finally, other than the important parameters, the rest of the parameters will be 0. Be careful that this model is only for when we make sure that there are several important features and the rest are useless, in case of doubt in this case, we should use ridge regression. lasso is a good and automatic method for feature selection.

3. Lasso may not work properly when the number of our training data is less than the number of features or when several parameters have correlations with each other, in this case we use elastic net.

Q5 : Implement Linear Regression with Mean Absolute Error as the cost function from scratch. Compare your results with the Linear Regression module of Scikit-Learn. Apply the model on Diabetes dataset.

As it is clear in the code, the outputs of Scikit-learn are far better and more accurate than the outputs of our model and provide less error.

Mean Absolute error of our model :

```
mae = 0
for i in range(len(y_pred)):
    mae += abs(y_test[i] - y_pred[i])

mae /= len(y_pred)
```

```
print("Mean Absolute Error:", mae)
```

```
Mean Absolute Error: 58.472985160921205
```

Mean Absolute error of scikit learn model :

```
mae = mean_absolute_error(y_test, y_pred)
```

```
print(f'Mean absolute error: {mae:.2f}')
```

```
Mean absolute error: 46.02
```

	Actual	Predict
0	321.0	138.221605
1	215.0	138.717756
2	127.0	136.940856
3	64.0	135.512468
4	175.0	136.057444
...	...	...
84	104.0	134.632395
85	49.0	135.078680
86	103.0	135.244613
87	142.0	137.815427
88	59.0	136.019977

Our Implement Model

	Actual	Predict
0	179.0	118.121704
1	168.0	129.621247
2	281.0	265.466444
3	60.0	122.126653
4	186.0	196.637082
...	...	...
128	311.0	167.756383
129	189.0	204.374845
130	68.0	196.750081
131	81.0	132.694810
132	96.0	64.362533

Scikit-Learn Model

There can be various reasons behind these results like choice of loss function, Optimization Algorithm used, etc. However, the prime goal of this notebook was to demonstrate the implementation of Multiple Linear Regression and not to perform better than Sklearn.

Q6 : Implement Linear Regression using the normal equation as the training algorithm from scratch. Apply the model on Diabetes dataset.

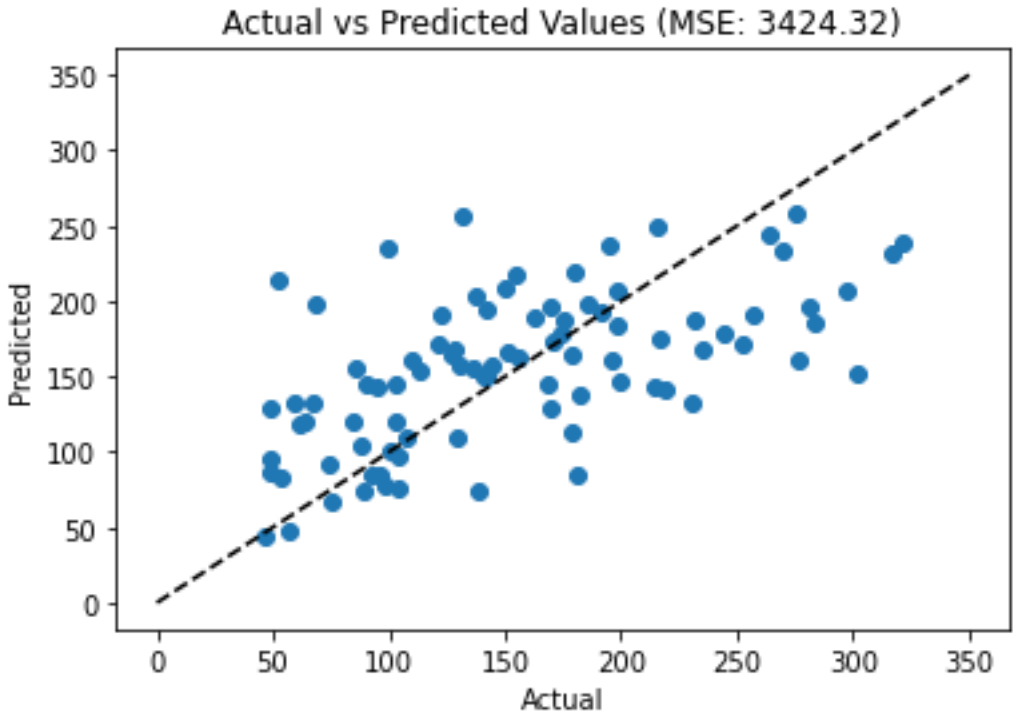
After importing the required libraries, we start loading the data and scaling them to get better results with good preprocessing. After that, we implement our own linear regression model with normal equation.

```
class LinearRegression:
    def __init__(self):
        self.theta = None

    def fit(self, X, y):
        X = np.insert(X, 0, 1, axis=1) # add bias term
        self.theta = np.linalg.inv(X.T @ X) @ X.T @ y

    def predict(self, X):
        X = np.insert(X, 0, 1, axis=1) # add bias term
        return X @ self.theta
```

At the end, we train the model and draw the outputs in a graph with their actual values, as can be seen, the results have provided much better results and predictions than the linear model implemented in question 5. and has less error in estimating test values.



	Actual	Predict
0	321.0	238.471452
1	215.0	248.931706
2	127.0	164.054042
3	64.0	120.307944
4	175.0	187.424221
...	...	...
84	104.0	76.773777
85	49.0	94.940469
86	103.0	145.295505
87	142.0	194.037764
88	59.0	132.785343

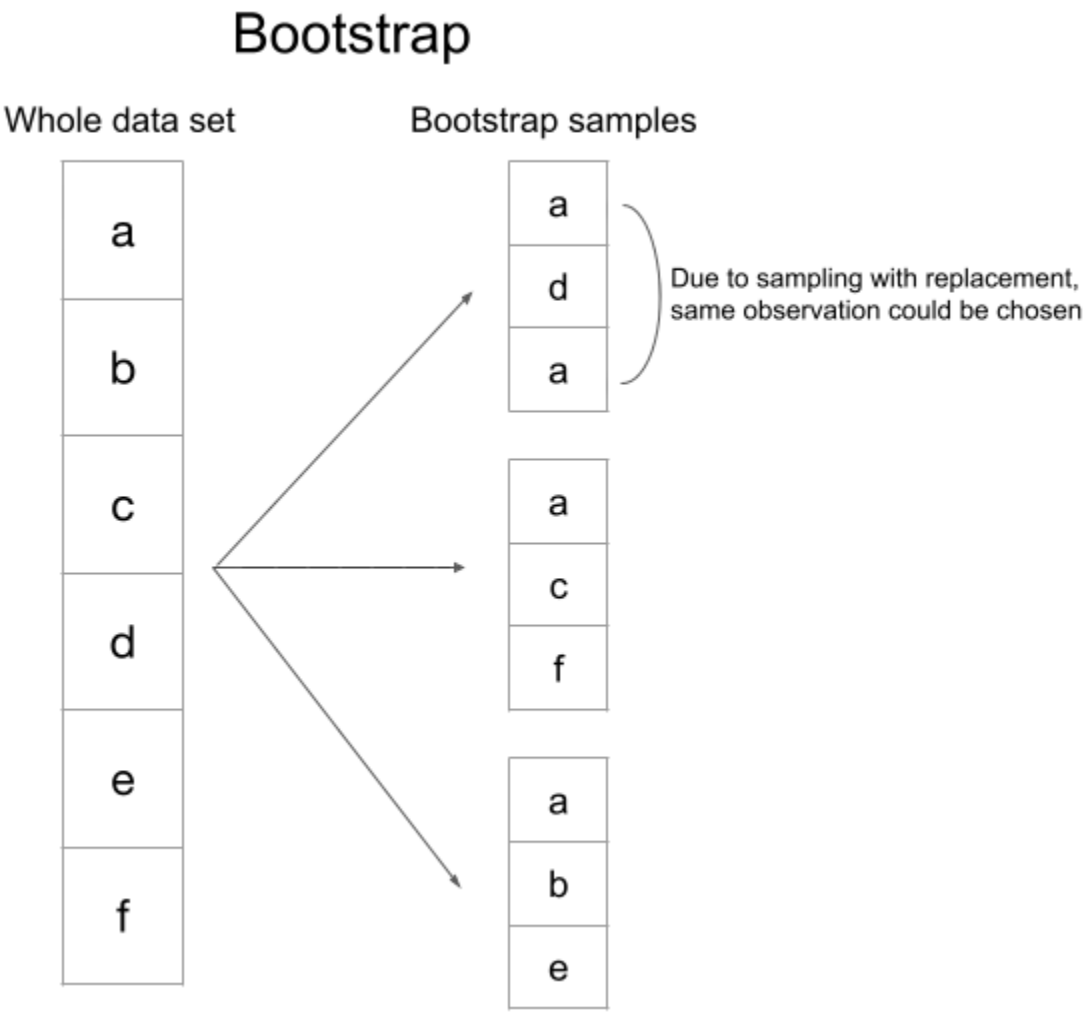
Our Linear regression learning with normal equation

But there are two basic problems in this algorithm. First, if the number of data points exceeds a certain limit, this process will be very long, and on the other hand, this method only works in linear regression models, and unlike gradient descent, it will not work in other models.

Q7 : Compare bootstrapping with cross-validation. In which conditions we should use bootstrapping?

Bootstrapping and cross-validation are two common techniques used in machine learning to estimate the performance of a model on unseen data. Both techniques involve generating multiple samples from the original dataset and training the model on these samples.

Bootstrapping involves randomly selecting samples (with replacement) from the dataset to create new samples, which are then used to train and evaluate the model. The results from each sample are aggregated to provide an estimate of the model's performance. Bootstrapping is particularly useful when the dataset is small or when the underlying distribution is not well-known.



Cross-validation involves partitioning the dataset into multiple folds and training the model on all but one fold, which is used for evaluation. This process is repeated for each fold, and the results are aggregated to provide an estimate of the model's performance. Cross-validation is particularly useful when the dataset is large and there is enough data to partition it into multiple folds.

In general, bootstrapping is preferred when the dataset is small or when the underlying distribution is not well-known, while cross-validation is preferred when the dataset is large and there is enough data to partition it into multiple folds. However, there are situations where bootstrapping may be preferred over cross-validation. For example, when the dataset is highly imbalanced, cross-validation may not provide a representative sample of the minority class in each fold. In such cases, bootstrapping can be used to generate multiple samples that are more representative of the underlying distribution.

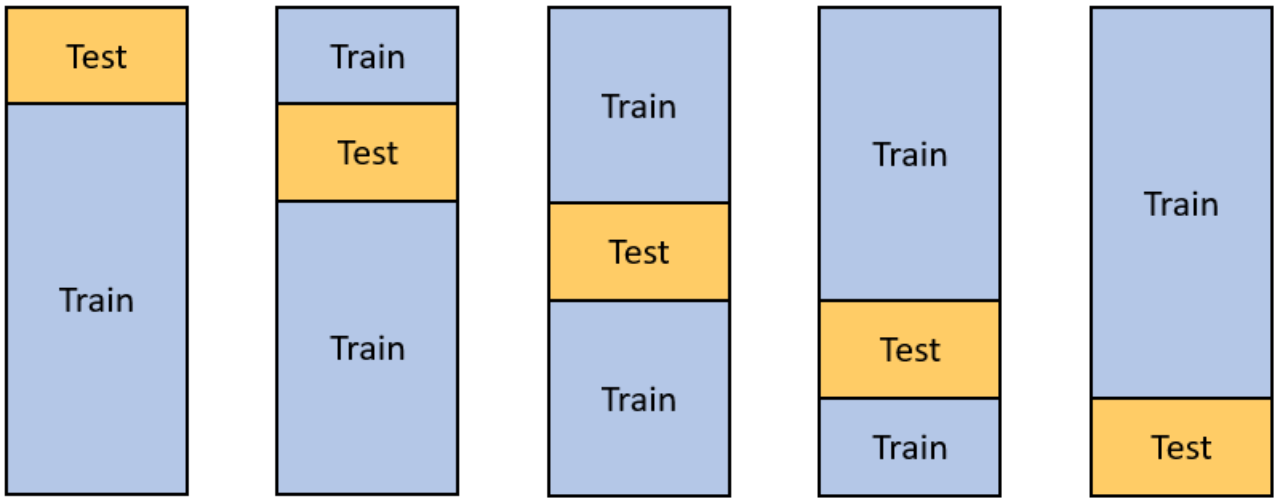
Overall, both bootstrapping and cross-validation are useful techniques for estimating the performance of a model on unseen data, and the choice of which technique to use depends on the size and nature of the dataset, as well as the specific goals of the analysis.

Q8 : Explain nested cross-validation and 5x2 cross-validation in detail and when we should use them.

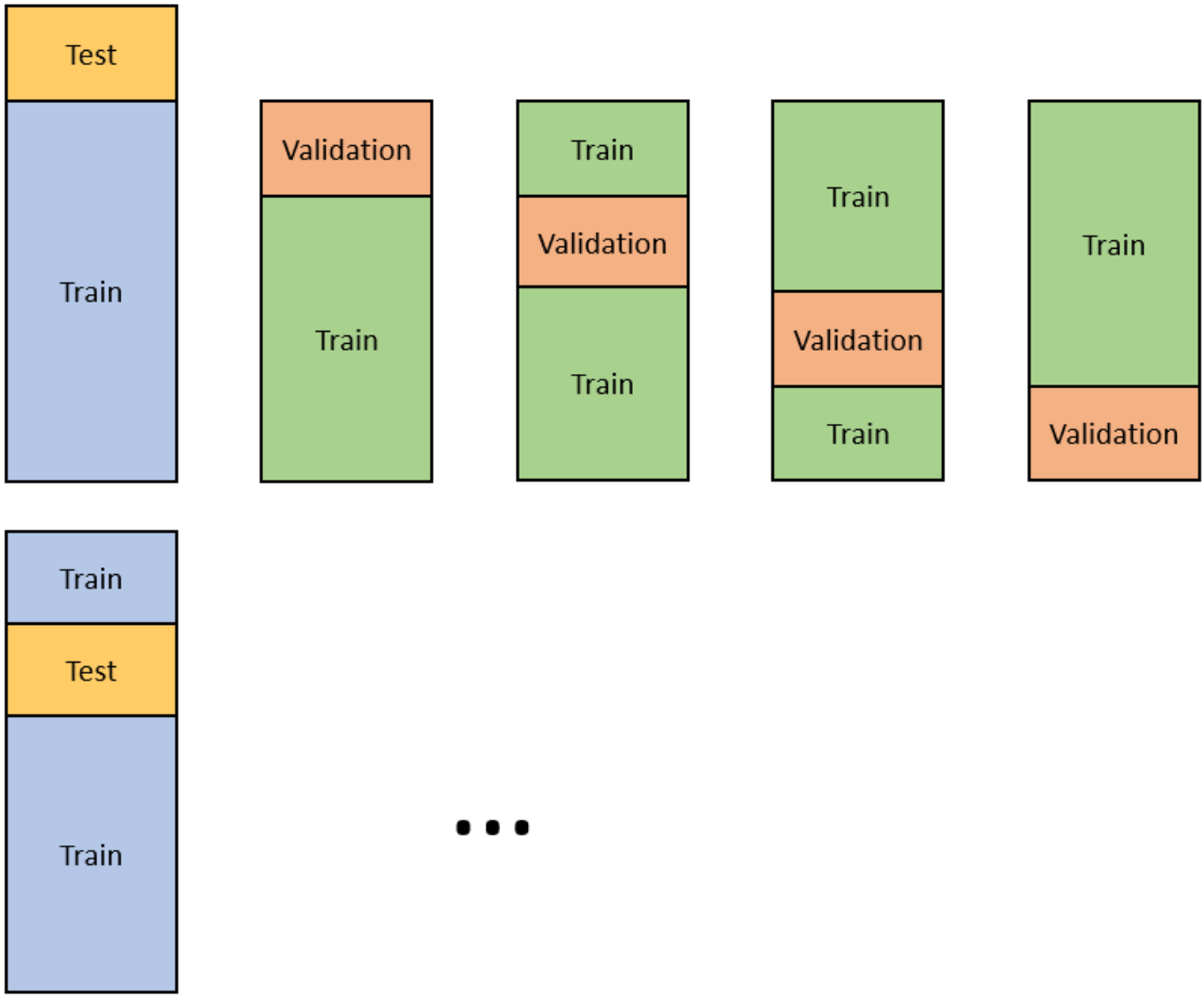
Nested cross-validation and 5x2 cross-validation are two techniques used in machine learning to estimate the performance of a model on unseen data.

Nested cross-validation involves using multiple layers of cross-validation to both tune the hyperparameters of the model and estimate its performance. It is especially useful when the dataset is small and there are a large number of hyperparameters to tune. The advantage of nested cross-validation is that it provides a more accurate estimate of the model's performance, since it uses multiple levels of cross-validation to evaluate the model.

Simple cross-validation



Nested cross-validation



On the other hand, 5x2 cross-validation involves dividing the dataset into 10 non-overlapping subsets, with 5 subsets used for training and 2 subsets used for testing. This process is repeated 5 times, with a different set of 5 training subsets and 2 testing subsets used in each iteration. The results from each iteration are then averaged to provide an estimate of the model's performance. 5x2 cross-validation is particularly useful when the dataset is large and there is a need for a more robust estimate of the model's performance, especially when the data is noisy or the model is sensitive to the choice of training and testing subsets.

The 5x2cv paired t test is a method used to compare the performance of two estimators (such as classifiers) A and B, using a labeled dataset D. This method involves repeated splitting of the dataset into two parts, a training set and a test set.

In each of the 5 iterations, the dataset is split randomly into 50% training and 50% test data. Both A and B are trained on the training split and their performance (measured as pA and pB) is evaluated on the corresponding test split.

After the initial evaluation, the training and test sets are swapped and the performance of both A and B is evaluated again. This results in two performance difference measures for each iteration.

This process is repeated five times, resulting in a total of 10 performance difference measures. These measures are used to compute a paired t-test, which is a statistical test that determines whether there is a significant difference in the performance of A and B.

$$p^{(1)} = p_A^{(1)} - p_B^{(1)}$$

$$p^{(2)} = p_A^{(2)} - p_B^{(2)}.$$

Then, we estimate the estimate mean and variance of the differences:

$$\overline{p} = \frac{p^{(1)} + p^{(2)}}{2}$$

and

$$s^2 = (p^{(1)} - \overline{p})^2 + (p^{(2)} - \overline{p})^2.$$

The variance of the difference is computed for the 5 iterations and then used to compute the *t* statistic as follows:

$$t = \frac{p_1^{(1)}}{\sqrt{(1/5)\sum_{i=1}^{5} s_i^2}},$$

The choice of which technique to use depends on the size and nature of the dataset, as well as the specific goals of the analysis. Nested cross-validation is preferred when the dataset is small and there are many hyperparameters to tune, while 5x2 cross-validation is preferred when the dataset is large and a more robust estimate of the model's performance is required.

Q9 : How can we compare different models using statistical significance tests?

To compare different models using statistical significance tests, you need to follow these general steps:

i. Formulate a null hypothesis, which is the hypothesis that assumes there is no significant difference between the models being compared.

ii. Formulate an alternative hypothesis, which is the hypothesis that assumes there is a significant difference between the models being compared.

iii. Select an appropriate statistical test based on the type of data and hypothesis being tested.

iv. Calculate the test statistic, which measures the difference between the observed data and what would be expected under the null hypothesis.

v. Determine the p-value, which is the probability of obtaining a test statistic as extreme as the observed value, assuming the null hypothesis is true.

vi. Interpret the results by comparing the p-value to a significance level (usually 0.05). If the p-value is less than the significance level, we reject the null hypothesis and conclude that there is a significant difference between the models being compared. If the p-value is greater than the significance level, we fail to reject the null hypothesis and conclude that there is insufficient evidence to support the alternative hypothesis.

There are other solutions for comparing different models beyond statistical significance tests. Some of these include:

i. Model selection criteria: These are quantitative measures that assess the quality of a model, such as accuracy, precision, recall, or F1-score. By comparing these metrics across different models, you can determine which one performs better.

ii. Cross-validation: This is a technique for assessing the performance of a model on an independent dataset. By randomly splitting the data into training and validation sets and testing the model on each subset, you can get a sense of how well it generalizes to new data.

iii. Information criteria: These are statistical measures that balance model fit and complexity, such as Akaike Information Criterion (AIC) or Bayesian Information Criterion (BIC). By comparing these criteria across different models, you can select the one that achieves the best balance between fit and complexity.

iv. Visualizations: Sometimes, it can be helpful to visualize the performance of different models, such as through ROC curves, precision-recall curves, or confusion matrices. This can help you see how the models differ and where they excel or struggle.

Keep in mind that the specifics of the analysis will depend on the data and problem being studied, and it's important to seek the advice of a statistician or data analyst to ensure the analysis is appropriate and the results are correctly interpreted.

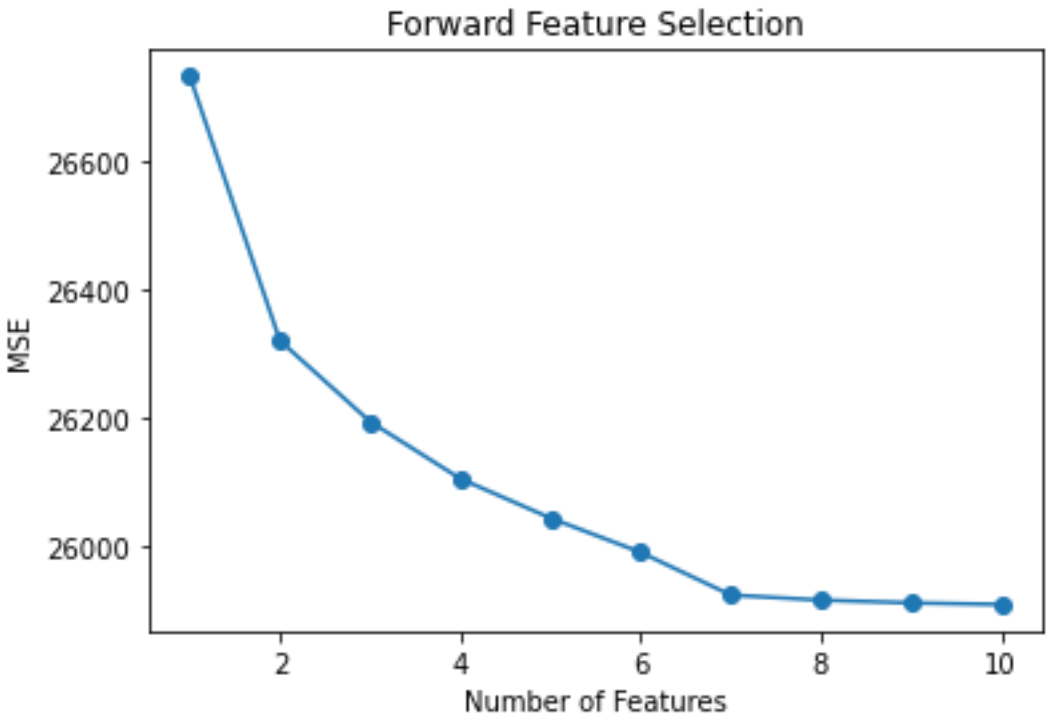
A combination of different methods may be needed to determine the best model for a given problem.

Q10 : Implement Forward and Backward Feature selection algorithms from scratch with MSE as the metric.

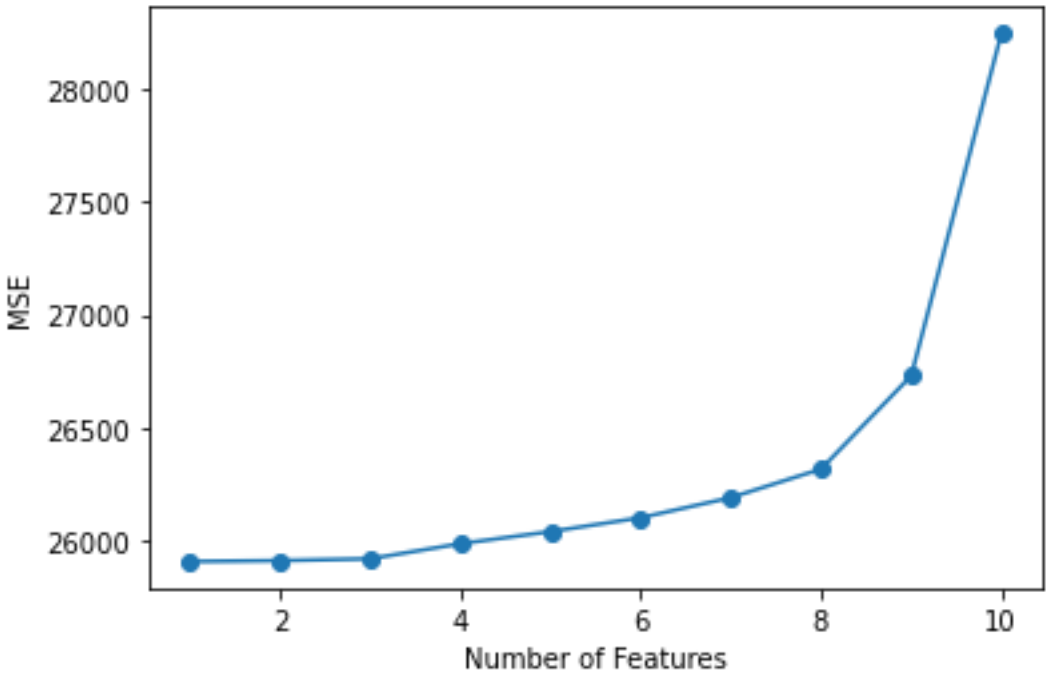
The backward feature selection algorithm starts with all the features and then removes one feature at a time based on which feature's removal results in the lowest MSE. This process is continued until no more features can be removed.

It is possible that backward feature selection is returning a higher MSE compared to forward feature selection because it may be removing important features too early in the process. This can happen if the order of removal of features is not optimal. In some cases, removing certain features may not have a large impact on the MSE, but removing other features may have a bigger impact, and backward feature selection may not be able to identify the latter as important until later stages.

In general, both forward and backward feature selection algorithms may not always find the optimal set of features for a given problem. It is often a good idea to try multiple feature selection methods and compare their results to determine the best set of features.



Forward selection



Backward Selection

```
Forward selection: Best features = [2, 8, 4, 3, 1, 5, 6, 0, 7, 9], MSE = 25907.579899
784774
Backward selection: Best features = [], MSE = 28245.176666666666
```

If the initial model with all features has the lowest MSE, then the algorithm will not remove any feature, and the best feature subset will be the full set of features. However, if the initial model has a high MSE, then the algorithm will remove one feature at a time until all features are removed, resulting in an empty set of selected features.

Q11 : Suppose the features in your training set have very different scales. Which algorithms (Gradient Descent, Normal Equation, SVD) might suffer from this, and how? What can you do about it?

When the features in a training set have very different scales, some machine learning algorithms may suffer from this in different ways:

i. Gradient Descent: Gradient Descent is an optimization algorithm that tries to find the minimum of the cost function by iteratively updating the model's parameters. In cases where the features are not properly scaled, the cost function can be elongated and have a lot of oscillations. This leads to slow convergence or even the model getting stuck in a suboptimal solution. One way to handle this is to scale the features before training the model. Standardization (scaling to have zero mean and unit variance) or normalization (scaling to the range [0, 1]) are common techniques used to scale features. This will ensure that each feature has a similar impact on the cost function and that Gradient Descent can converge faster.

ii. Normal Equation: Normal Equation is a closed-form solution to linear regression that computes the optimal parameters for the model in one step. However, if the features are not scaled, the matrix inverse operation can be numerically unstable and lead to incorrect results. This is because large differences in the scales of the features can lead to large eigenvalues in the covariance matrix, which can make the inverse computation inaccurate. One way to handle this is to use the SVD method instead, which is more numerically stable and can handle large differences in scales.

iii. Singular Value Decomposition (SVD): SVD is a matrix factorization technique used in many machine learning algorithms, including Principal Component Analysis (PCA) and Linear Discriminant Analysis (LDA). SVD is also affected by different scales of features, as it involves computing the covariance matrix of the data. This can lead to eigenvectors with very different magnitudes and directions, which can affect the quality of the decomposition. One way to handle this is to scale the features before performing SVD, or to use PCA instead, which automatically scales the features to have zero mean and unit variance.

In general, scaling the features is an important step in most machine learning algorithms. It ensures that each feature has a similar impact on the model, avoids numerical instability issues, and can improve the model's performance.

aim for about $-1 \leq x_j \leq 1$ for each feature x_j

$-3 \leq x_j \leq 3$

$-0.3 \leq x_j \leq 0.3$

} acceptable ranges

$0 \leq x_1 \leq 3$ okay, no rescaling

$-2 \leq x_2 \leq 0.5$ okay, no rescaling

$-100 \leq x_3 \leq 100$ too large → rescale

$-0.001 \leq x_4 \leq 0.001$ too small → rescale

$98.6 \leq x_5 \leq 105$ too large → rescale