Machine Learning Worksheet 1

Q1. → B). O(n)

Q2. → C). Polynomial Regression

Q3. → B). Gradient Descent

Q4. → A).Extrapolation

Q5. → B).Mini-Batch Gradient Descent

Q6. → B).False

Q7. → A). scaling cost function by half makes gradient descent converge faster

Q8. → C).Both of them

Q9. → (A) , (B)

Q10. → (C)

Q11. → (D)

Q12. →

We could use Batch Gradient Descent,Stochastic Gradient descent or Mini-Batch Gradient Descent Algorithms. MBGD and SGD would work the best because neither of them need to load the entire dataset into memory in order to take 1 step of gradient descent.Here, MBGD is a type of gradient descent which works faster than both batch gradient descent and stochastic gradient descent. Here *b* examples where *b<m* are processed per

iteration. So even if the number of training examples is large, it is processed in batches of b training examples in one go. Thus, it works for larger training examples and that too with lesser number of iterations.

SGD is a type of gradient descent which processes 1 training example per iteration. Hence, the parameters are being updated even after one iteration in which only a single example has been processed.

Batch Gradient Descent is a type of gradient descent which processes all the training examples for each iteration of gradient descent. But if the number of training examples is large, then batch gradient descent is computationally very expensive.

Normal equations are not good choice because it is computationally inefficient. The closed-form solution may (should) be preferred for "smaller" datasets – if computing (a "costly") matrix inverse is not a concern. For very large datasets, or datasets where the inverse of $\mathbf{X}^T\mathbf{X}$ may not exist (the matrix is non-invertible or singular, e.g., in case of perfect multicollinearity), the GD or SGD approaches are to be preferred.

Q13. →

The ML algorithm is sensitive to the "**relative scales of features,**" which usually happens when it uses the numeric values of the features rather than say their rank.

Feature scaling is essential for machine learning algorithms that calculate **distances between data**. If not scale, the feature with a higher value range starts dominating when calculating distances.

The underlying algorithms to distance-based models make them the most vulnerable to unscaled data.

Distance based algorithms:

KNN, K-means, and SVM are most affected by the range of features. This is because behind the scenes they are using distances between data points to determine their similarity. Hence, features with a greater magnitude will be assigned a higher weightage by the model. This is not an ideal scenario as we do not want our model to be heavily biased towards a single feature. Evidently, it is crucial that we implement feature scaling to our data before fitting them to distance-based algorithms to ensure that all features contribute equally to the result of the predictions.

1.**K-nearest neighbors** (KNN) with a Euclidean distance measure is sensitive to magnitudes and hence should be scaled for all features to weigh in equally. . **KNN algorithm** is seriously affected because you choose the K closest samples for your predictions

2.K-Means uses the Euclidean distance measure here feature scaling matters.

3. Scaling is critical while performing **Principal Component Analysis(PCA)**. PCA tries to get the features with maximum variance, and the variance is high for high magnitude features and skews the PCA towards high magnitude features.

4.We can speed up **gradient descent** by scaling because θ descends quickly on small ranges and slowly on large ranges, and oscillates inefficiently down to the optimum when the variables are very uneven. Gradient descent is an iterative optimisation algorithm that takes us to the minimum of a function.Machine learning algorithms like linear regression and logistic regression rely on gradient descent to minimise their loss functions or in other words, to reduce the error between the predicted values and the actual values.

Having features with varying degrees of magnitude and range will cause different step sizes for each feature. Therefore, to ensure that gradient descent converges more smoothly and quickly, we need to scale our features so that they share a similar scale.

Tree-Based Algorithms:

Naive Bayes, Linear Discriminant Analysis, and Tree-Based models are not affected by feature scaling. In Short, any Algorithm which is Not Distance based is Not affected by Feature Scaling are **Tree-based algorithms.**For that reason, we can deduce that decision trees are invariant to the scale of the features and thus do not require feature scaling. This also includes other ensemble models that tree-based, for example, random forest and gradient boosting

Algorithms that do not require normalization/scaling are the ones that **rely on rules**. They would not be affected by any monotonic transformations of the variables. Scaling is a monotonic transformation. Examples of algorithms in this category are all the tree-based algorithms — **CART, Random Forests, Gradient Boosted Decision Trees**. These algorithms utilize rules (series of inequalities) and **do not require normalization**.

- Each node in a classification and regression trees (CART) model, otherwise known as decision trees represents a single feature in a dataset.

- The tree splits each node in such a way that it increases the homogeneity of that node. This split is not affected by the other features in the dataset.

- For that reason, we can deduce that decision trees are invariant to the scale of the features and thus do not require feature scaling.

- This also includes other ensemble models that tree-based, for example, random forest and gradient boosting.