Mrs. Swati Amit Motugade

MACHINE LEARNING

ASSIGNMENT 5

Q.1.

Ans.

Residual Sum of Squares:

The residual sum of squares (RSS) is a statistical technique used to measure the amount of variance in a data set that is not explained by a regression model itself. The residual sum of squares (RSS) measures the level of variance in the error term, or residuals, of a regression model. The smaller the residual sum of squares, the better your model fits your data; the greater the residual sum of squares, the poorer your model fits your data.

In a model with a single explanatory variable, RSS is given by

$RSS = \sum_{i=1}^{n}(y_i - f(x_i))^2$

Where, yi is the ith value of dependent variable and xi is the ith value of independent variable in a linear regression equation $y_i = \alpha + \beta x_i + \epsilon_i$

And α and β are coefficients.

R-Squared:

R-Squared is a statistical measure of fit that indicates how much variation of a dependent variable is explained by the independent variable(s) in a regression model.

The actual calculation of R-squared requires several steps. This includes taking the data points (observations) of dependent and independent variables and finding the line of best fit, often from a regression model. From there you would calculate predicted values, subtract actual values and square the results. This yields a list of errors squared, which is then summed and equals the unexplained variance.

Formula of R-squared is

$$R^2 = 1 - \frac{Unexplained\ Variation}{Total\ variation} = 1 - \frac{RSS}{TSS}$$

The RSS is just the absolute amount of explained variation, the R squared is the (RSS/SST), i.e., the absolute amount of variation as a proportion of total variation.

Q.2. What are TSS (Total Sum of Squares), ESS (Explained Sum of Squares) and RSS (Residual Sum of Squares) in regression. Also mention the equation relating these three metrics with each other.

Ans.

1. Total Sum of Squares:

The total sum of squares is a variation of the values of a dependent variable from the sample mean of the dependent variable. Essentially, the total sum of squares quantifies the total variation in a sample. It can be determined using the following formula:

$$\boldsymbol{TSS = \sum_{i=1}^{n} (y_i - \bar{y})^2}$$

Where, y_i is the value in sample and $\bar{y}$ is the mean value of sample.

2. Explained Sum of Squares:

The explained sum of squares which is also known as Regression Sum of Squares is describes how well a regression model represents the modelled data. A higher regression sum of squares indicates that the model does not fit the data well.

The formula for calculating the regression sum of squares is:

$$ESS = \sum_{i=1}^{n} (\hat{y}_i - \bar{y})^2$$

Where, $\hat{y}_i$ is the value estimated by regression line and $\bar{y}$ is the mean value of sample.

3. Residual Sum of Squares:

The residual sum of squares essentially measures the variation of modelling errors. In other words, it depicts how the variation in the dependent variable in a regression model cannot be explained by the model. Generally, a lower residual sum of squares indicates that the regression model can better explain the data, while a higher residual sum of squares indicates that the model poorly explains the data.

The residual sum of squares can be found using the formula below:

$$RSS = \sum_{i=1}^{n} (y_i - \hat{y}_i)^2$$

Where, $\boldsymbol{y_i}$ is the observed value and $\hat{y}_i$ is the value estimated by regression line.

The relationship between TSS, ESS and RSS can be summarized by the following equation:

$$\boldsymbol{TSS = ESS + RSS}$$

Q. 3. What is the need of regularization in machine learning?

Ans. While training a machine learning model, the model can easily be overfitted or under fitted. To avoid this, we use regularization in machine learning to properly fit a model onto our test set. Regularization techniques help reduce the chance of overfitting and help us get an optimal model.

Regularization refers to techniques that are used to calibrate machine learning models in order to minimize the adjusted loss function and prevent overfitting or underfitting. Using Regularization, we can fit our machine learning model appropriately on a given test set and hence reduce the errors in it.

Q. 4. What is Gini–impurity index?

Ans. Gini Impurity is a measurement of the likelihood of an incorrect classification of a new instance of a random variable, if that new instance were

randomly classified according to the distribution of class labels from the data set.

Gini impurity is lower bounded by 0, with 0 occurring if the data set contains only one class.

The formula for calculating the gini impurity of a data set or feature is as follows:

$$G(k) = \sum_{i=1}^{J} P(i) * (1 - P(i))$$

Where P(i) is the probability of a certain classification i, per the training data set.

In other words, Gini Impurity is a measurement used to build Decision Trees to determine how the features of a dataset should split nodes to form the tree. More precisely, the Gini Impurity of a dataset is a number between 0-0.5, which indicates the likelihood of new, random data being misclassified if it were given a random class label according to the class distribution in the dataset.

Q. 5. Are unregularized decision-trees prone to overfitting? If yes, why?

Ans. Decision trees are a type of model used for both classification and regression. Trees answer sequential questions which send us down a certain route of the tree given the answer. The model behaves with “if this than that” conditions ultimately yielding a specific result.

Decision trees are prone to overfitting, especially when a tree is particularly deep. This is due to the amount of specificity we look at leading to smaller sample of events that meet the previous assumptions. This small sample could lead to unsound conclusions.

Decision Trees are a non-parametric supervised machine learning approach for classification and regression tasks. Overfitting is a common problem; a data scientist needs to handle while training decision tree models. Comparing to other machine learning algorithms, decision trees can easily overfit.

Q. 6. What is an ensemble technique in machine learning?

Ans. Ensemble methods are techniques that aims at improving the accuracy of results in models by combining multiple models instead of using a single model. The combined models increase the accuracy of the results significantly. This has boosted the popularity of ensemble methods in machine learning.

Ensemble methods fall into two broad categories, i.e., sequential ensemble techniques and parallel ensemble techniques.

Sequential ensemble techniques generate base learners in a sequence, e.g., Adaptive Boosting (AdaBoost). The sequential generation of base learners promotes the dependence between the base learners. The performance of the model is then improved by assigning higher weights to previously misrepresented learners.

In parallel ensemble techniques**,** base learners are generated in a parallel format, e.g., random forest. Parallel methods utilize the parallel generation of base learners to encourage independence between the base learners. The independence of base learners significantly reduces the error due to the application of averages.

Q. 7. What is the difference between Bagging and Boosting techniques?

Ans. Ensemble methods combine different decision trees to deliver better predictive results, afterward utilizing a single decision tree. The primary principle behind the ensemble model is that a group of weak learners come together to form an active learner.

There are two techniques that are used to perform ensemble decision tree.

Bagging is used when our objective is to reduce the variance of a decision tree. Here the concept is to create a few subsets of data from the training sample, which is chosen randomly with replacement. Now each collection of subset data is used to prepare their decision trees thus, we end up with an ensemble of various models. The average of all the assumptions from numerous tress is used, which is more powerful than a single decision tree.

Boosting is another ensemble procedure to make a collection of predictors. In other words, we fit consecutive trees, usually random samples, and at each step, the objective is to solve net error from the prior trees.

If a given input is misclassified by theory, then its weight is increased so that the upcoming hypothesis is more likely to classify it correctly by consolidating the entire set at last converts weak learners into better performing models.

Difference between Bagging and Boosting

Bagging	Boosting
Various training data subsets are randomly drawn with replacement from the whole training dataset.	Each new subset contains the components that were misclassified by previous models.
Bagging attempts to tackle the over-fitting issue.	Boosting tries to reduce bias.
If the classifier is unstable (high variance), then we need to apply bagging.	If the classifier is steady and straightforward (high bias), then we need to apply boosting.
Every model receives an equal weight.	Models are weighted by their performance.
Objective to decrease variance, not bias.	Objective to decrease bias, not variance.
It is the easiest way of connecting predictions that belong to the same type.	It is a way of connecting predictions that belong to the different types.
Every model is constructed independently.	New models are affected by the performance of the previously developed model.

Q. 8. What is out-of-bag error in random forests?

Ans. The out-of-bag error is the average error for each predicted outcome calculated using predictions from the trees that do not contain that data point in their respective bootstrap sample. This way, the Random Forest model is constantly being validated while being trained. Let us consider the j^{th} decision tree that has been fitted on a subset of the sample data. For every training

observation or sample $z_i = (x_i, y_i)$ not in the sample subset of DT_j where x_i is the set of features and y_i is the target, we use DT_j to predict the outcome o_i for x_i. The error can easily be computed as $|o_i - y_i|$.
The out-of-bag error is thus the average value of this error across all decision trees.

Q. 9. What is K-fold cross-validation?

Ans. K-fold cross-validation is a data splitting technique that can be implemented with k > 1 folds. K-Fold Cross Validation is also known as k-cross, k-fold cross-validation, k-fold CV, and k-folds. The k-fold cross-validation technique can be implemented easily using Python with scikit learn (Sklearn) package which provides an easy way to calculate k-fold cross-validation models. It is important to learn the concepts of cross-validation concepts in order to perform model tuning with the end goal to choose a model which has a high generalization performance**.**

K-fold cross-validation is defined as a method for estimating the performance of a model on unseen data. This technique is recommended to be used when the data is scarce and there is an ask to get a good estimate of training and generalization error thereby understanding the aspects such as underfitting and overfitting. This technique is used for hyperparameter tuning such that the model with the most optimal value of hyperparameters can be trained. It is a resampling technique without replacement. The advantage of this approach is that each example is used for training and validation (as part of a test fold) exactly once. This yields a lower-variance estimate of the model performance than the holdout method. As mentioned earlier, this technique is used because it helps to avoid overfitting, which can occur when a model is trained using all of the data. By using k-fold cross-validation, we are able to "test" the model on k different data sets, which helps to ensure that the model is generalizable.

Q. 10. What is hyper parameter tuning in machine learning and why it is done?

Ans. Hyperparameters are parameters whose values control the learning process and determine the values of model parameters that a learning algorithm ends up learning. The prefix 'hyper_' suggests that they are 'top-

level' parameters that control the learning process and the model parameters that result from it.

Hyperparameter tuning takes advantage of the processing infrastructure of Google Cloud to test different hyperparameter configurations when training your model. It can give you optimized values for hyperparameters, which maximizes your model's predictive accuracy. Hyperparameters are important because they can have a direct impact on the behaviour of the training algorithm and have a significant impact on the performance of the model being trained. Choosing appropriate hyperparameters plays a critical role in the success of neural network architecture and has a huge impact on the learned model. For example, if the learning rate is too low, the model will miss important patterns in the data. If it is high, it may have collisions.

Q. 11. What issues can occur if we have a large learning rate in Gradient Descent?

Ans. The learning rate controls how quickly the model is adapted to the problem. Smaller learning rates require more training epochs given the smaller changes made to the weights each update, whereas larger learning rates result in rapid changes and require fewer training epochs.

A learning rate that is too large can cause the model to converge too quickly to a suboptimal solution.

In order for Gradient Descent to work, we must set the learning rate to an appropriate value. This parameter determines how fast or slow we will move towards the optimal weights. If the learning rate is very large, we will skip the optimal solution. So, using a good learning rate is crucial.

Q. 12. Can we use Logistic Regression for classification of Non-Linear Data? If not, why?

Ans. Logistic regression is a classification algorithm used to find the probability of event success and event failure. It is used when the dependent variable is binary (0/1, True/False, Yes/No) in nature. It supports categorizing data into discrete classes by studying the relationship from a given set of labelled data. It learns a linear relationship from the given dataset and then introduces a non-linearity in the form of the Sigmoid function.

The major limitation of Logistic Regression is the assumption of linearity between the dependent variable and the independent variables. Non-linear problems can't be solved with logistic regression because it has a linear decision surface. Linearly separable data is rarely found in real-world scenarios.

Q. 13. Differentiate between Adaboost and Gradient Boosting.

Ans.

Adaboost: AdaBoost or Adaptive Boosting is the first Boosting ensemble model. The method automatically adjusts its parameters to the data based on the actual performance in the current iteration. Meaning, both the weights for re-weighting the data and the weights for the final aggregation are re-computed iteratively.

Gradient Boosting: Gradient Boost is a robust machine learning algorithm made up of Gradient descent and boosting. The word 'gradient' implies that you can have two or more derivatives of the same function. Gradient Boosting has three main components: additive model, loss function and a weak learner.

Adaboost	Gradient Boosting
In case of Adaptive Boosting or AdaBoost, it minimises the exponential loss function that can make the algorithm sensitive to the outliers.	With Gradient Boosting, any differentiable loss function can be utilised. Gradient Boosting algorithm is more robust to outliers than AdaBoost.
AdaBoost is the first designed boosting algorithm with a particular loss function.	Gradient Boosting is a generic algorithm that assists in searching the approximate solutions to the additive modelling problem. This makes Gradient Boosting more flexible than AdaBoost.
AdaBoost minimises loss function related to any classification error and is best used with weak learners. The method was mainly designed for binary classification problems and can be utilised to boost the performance of decision trees.	Gradient Boosting is used to solve the differentiable loss function problem. The technique can be used for both classification and regression problems.
with AdaBoost, it can be identified by high-weight data points.	In the case of Gradient Boosting, the shortcomings of the existing weak

	learners can be identified by gradients
In the case of AdaBoost, the shifting is done by up-weighting observations that were misclassified before	Gradient Boosting identifies the difficult observations by large residuals computed in the previous iterations.

Q. 14. What is bias-variance trade off in machine learning?

Ans. Bias Variance Trade-off is a design consideration when training the machine learning model. Certain algorithms inherently have a high bias and low variance and vice-versa.

1. If a model uses a simple machine learning algorithm like in the case of a linear model in the above code, the model will have high bias and low variance (underfitting the data).
2. If a model follows a complex machine learning model, then it will have high variance and low bias (overfitting the data).
3. We need to find a good balance between the bias and variance of the model we have used. This trade-off in complexity is what is referred to as bias and variance trade-off. An optimal balance of bias and variance should never overfit or underfit the model.
4. This trade-off applies to all forms of supervised learning: classification, regression, and structured output learning.

Q. 15. Give short description each of Linear, RBF, Polynomial kernels used in SVM.

Ans. Different SVM algorithms use differing kinds of kernel functions. These functions are of different kinds—for instance, linear, nonlinear, polynomial, radial basis function (RBF), and sigmoid.

The most preferred kind of kernel function is RBF. Because it's localized and has a finite response along the complete x-axis.

1. Linear Kernel

It is the most basic type of kernel, usually one dimensional in nature. It proves to be the best function when there are lots of features. The linear kernel is mostly preferred for **text-classification problems** as most of these kinds of classification problems can be linearly separated.

Linear kernel functions are **faster** than other functions.

The formula for Linear Kernel Formula is

$$F(X, Xj) = sum(X.Xj)$$

Here, **X,Xj** represents the data we are trying to classify.

2. Polynomial Kernel

It is a more generalized representation of the linear kernel. It **is not** as preferred as other kernel functions as it is less efficient and accurate.

$$F(X, Xj) = (X.Xj + 1)\text{^}d$$

Here '.' shows the **dot product** of both the values, and **d** denotes the degree.

F(x, xj) representing the **decision boundary** to separate the given classes.

3. RBF

It is one of the most preferred and used kernel functions in svm. It is usually chosen for non-linear data. It helps to make proper separation when there is no prior knowledge of data. It is also known as Gaussian Radial Basis Formula

$$F(X, Xj) = \exp(-gamma * \left||X - Xj|\right|^2)$$

The value of gamma varies from 0 to 1. You have to manually provide the value of gamma in the code. The most preferred value for gamma is 0.1.