ML Worksheet 5 Solution

1. R-squared or Residual Sum of Squares (RSS) which one of these two is a better measure of goodness of fit model in regression and why?

Ans. R-squared is a statistical measure that represents the proportion of the variance in the dependent variable that is predictable from the independent variable(s). It ranges from 0 to 1, where 1 represents a perfect fit and 0 represents a model that does not explain any of the variability in the dependent variable. R-squared is a relative measure, which means that it only compares the model's performance to a null model that always predicts the mean of the dependent variable.

Residual Sum of Squares (RSS) represents the sum of the squared differences between the predicted values and the actual values. It measures the overall difference between the predicted values and the actual values, regardless of whether the model is good or bad. A lower RSS value indicates a better fit of the model.

R-squared is often preferred as a measure of goodness of fit in regression because it is easily interpretable, it has a defined range between 0 and 1, and it allows for easy comparison between different models. However, in certain scenarios where the sample size is very small or the number of features is large, R-squared can be unreliable as it can be artificially high even if the model is not a good fit. In those cases, it may be more appropriate to rely on other evaluation metrics such as root mean square error (RMSE), mean absolute error (MAE), or cross-validation score.

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. The residual sum of squares is used to help you decide if a statistical model is a good fit for your data. It measures the overall difference between your data and the values predicted by your estimation model (a "residual" is a measure of the distance from a data point to a regression line).

ANOVA, Total SS is related to the total sum and explained sum with the following formula:

Total SS = Explained SS + Residual Sum of Squares. Watch the video for a definition and calculation steps for Total (TSS), Between (BSS), and Within (WSS):

The Total SS (TSS or SST) tells you how much variation there is in the dependent variable.

Total SS = Σ(Yi – mean of Y)2.

Note: Sigma (Σ) is a mathematical term for summation or "adding up." It's telling you to add up all the possible results from the rest of the equation.

Sum of squares is a measure of how a data set varies around a central number (like the mean). You might realize by the phrase that you're summing (adding up) squares—but squares of what? You'll sometimes see this formula:

y = Y-Y bar

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

Ans. Overfitting is a phenomenon that occurs when a Machine Learning model is constraint to training set and not able to perform well on unseen data.
Regularization is a technique used to reduce the errors by fitting the function appropriately on the given training set and avoid overfitting.

The commonly used regularization techniques are:

L1 regularization

L2 regularization

Dropout regularization

A regression model which uses L1 Regularization technique is called LASSO(Least Absolute Shrinkage and Selection Operator) regression.

A regression model that uses L2 regularization technique is called Ridge regression.

Lasso Regression adds "absolute value of magnitude" of coefficient as penalty term to the loss function(L).

Ridge regression adds "squared magnitude" of coefficient as penalty term to the loss function(L).

NOTE that during Regularization the output function(y_hat) does not change. The change is only in the loss function.

We define Loss function in Logistic Regression as :

L(y_hat,y) = y log y_hat + (1 - y)log(1 - y_hat)

Loss function with no regularization :

$$L = y \log (wx + b) + (1 - y)\log(1 - (wx + b))$$

Lets say the data overfits the above function.

Loss function with L1 regularization :

$$L = y \log (wx + b) + (1 - y)\log(1 - (wx + b)) + lambda*||w||1$$

Loss function with L2 regularization :

$$L = y \log (wx + b) + (1 - y)\log(1 - (wx + b)) + lambda*||w||22$$

lambda is a Hyperparameter Known as regularization constant and it is greater than zero.

4. What is Gini–impurity index?

Ans. Gini Index is a metric to measure how often a randomly chosen element would be incorrectly identified.

It means an attribute with a lower Gini index should be preferred.

Sklearn supports “Gini” criteria for Gini Index and by default, it takes “gini” value.

The Formula for the calculation of the Gini Index is given below.

Calculation

The Gini Index or Gini Impurity is calculated by subtracting the sum of the squared probabilities of each class from one. It favors mostly the larger partitions and are very simple to implement. In simple terms, it calculates the probability of a certain randomly selected feature that was classified incorrectly.

The Gini Index varies between 0 and 1, where 0 represents purity of the classification and 1 denotes random distribution of elements among various classes. A Gini Index of 0.5 shows that there is equal distribution of elements across some classes.

The Gini Index works on categorical variables and gives the results in terms of “success” or “failure” and hence performs only binary split. It isn’t computationally intensive as its counterpart – Information Gain. From the Gini Index, the value of another parameter named Gini Gain is calculated whose value is maximized with each iteration by the Decision Tree to get the perfect CART

Let us understand the calculation of the Gini Index with a simple example. In this, we have a total of 10 data points with two variables, the reds and the blues. The X and Y axes are numbered with spaces of 100 between each term. From the given example, we shall calculate the Gini Index and the Gini Gain.

For a decision tree, we need to split the dataset into two branches. Consider the following data points with 5 Reds and 5 Blues marked on the X-Y plane. Suppose we make a binary split at X=200, then we will have a perfect split as shown below.

It is seen that the split is correctly performed and we are left with two branches each with 5 reds (left branch) and 5 blues (right branch).

Basic Mechanism

To calculate the Gini Impurity, let us first understand it's basic mechanism.

First, we shall randomly pick up any data point from the dataset

Then, we will classify it randomly according to the class distribution in the given dataset. In our dataset, we shall give a data point chosen with a probability of 5/10 for red and 5/10 for blue as there are five data points of each color and hence the probability.

Now, in order to calculate the Gini Index, the formula is given by

G = Submission from I to C (p(i)*(1-p(I)))

Where, C is the total number of classes and p(i) is the probability of picking the data point with the class i.

In the above example, we have C=2 and p(1) = p(2) = 0.5, Hence the Gini Index can be calculated as,

$G = p(1) * (1-p(1)) + p(2) * (1-p(2))$

$= 0.5 * (1-0.5) + 0.5 * (1-0.5)$

$= 0.5$

Where 0.5 is the total probability of classifying a data point imperfectly and hence is exactly 50%.

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

Ans. Yes, unregularized decision trees are prone to overfitting. This is because they can continue to split on a feature until the leaves contain only a single class or a very small number of samples. This can lead to a tree with a large number of levels and a complex structure that perfectly fits the training data, but may not generalize well to new data.

To avoid this, different regularization techniques such as pruning, setting a maximum depth for the tree, and setting a minimum number of samples required at a leaf node can be used to control the growth of the tree and prevent overfitting.

6. What is an ensemble technique in machine learning?

Ans**.** Ensemble methods is a machine learning technique that combines several base models in order to produce one optimal predictive model. To better understand this definition lets take a step back into ultimate goal of machine learning and model building. This is going to make more sense as I dive into specific examples and why Ensemble methods are used.

I will largely utilize Decision Trees to outline the definition and practicality of Ensemble Methods (however it is important to note that Ensemble Methods do not only pertain to Decision Trees).

A Decision Tree determines the predictive value based on series of questions and conditions. For instance, this simple Decision Tree determining on whether an individual should play outside or not. The tree takes several weather factors into account, and given each factor either makes a decision or asks another question. In this example, every time it is overcast, we will play outside. However, if it is raining, we must ask if it is windy or not? If windy, we will not play. But given no wind, tie those shoelaces tight because were going outside to play.

Decision Trees can also solve quantitative problems as well with the same format. In the Tree to the left, we want to know whether or not to invest in a commercial real estate property. Is it an office building? A Warehouse? An Apartment building? Good economic conditions? Poor Economic Conditions? How much will an investment return? These questions are answered and solved using this decision tree.

When making Decision Trees, there are several factors we must take into consideration: On what features do we make our decisions on? What is the threshold for classifying each question into a yes or no answer? In the first Decision Tree, what if we wanted to ask ourselves if we had friends to play with or not. If we have friends, we will play every time. If not, we might continue to ask ourselves questions about the weather. By adding an additional question, we hope to greater define the Yes and No classes.

This is where Ensemble Methods come in handy! Rather than just relying on one Decision Tree and hoping we made the right decision at each split, Ensemble Methods allow us to take a sample of Decision Trees into account, calculate which features to use or questions to ask at each split, and make a final predictor based on the aggregated results of the sampled Decision Trees.

Types of Ensemble Methods

Bagging, or Bootstrap Aggregating.

Random forest model

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

Ans. Bagging is a homogeneous weak learners' model that learns from each other independently in parallel and combines them for determining the model average.

Boosting is also a homogeneous weak learners' model but works differently from Bagging. In this model, learners learn sequentially and adaptively to improve model predictions of a learning algorithm.

Bagging is a homogeneous weak learners' model that learns from each other independently in parallel and combines them for determining the model average.

Boosting is also a homogeneous weak learners' model but works differently from Bagging. In this model, learners learn sequentially and adaptively to improve model predictions of a learning algorithm.

That was Bagging and Boosting at a glimpse. Let's look at both of them in detail. Some of the factors that cause errors in learning are noise, bias, and variance. The ensemble method is applied to reduce these factors resulting in the stability and accuracy of the result.

Bagging

Bagging is an acronym for 'Bootstrap Aggregation' and is used to decrease the variance in the prediction model. Bagging is a parallel method that fits different, considered learners independently from each other, making it possible to train them simultaneously.

Bagging generates additional data for training from the dataset. This is achieved by random sampling with replacement from the original dataset. Sampling with replacement may repeat some observations in each new training data set. Every element in Bagging is equally probable for appearing in a new dataset.

These multi datasets are used to train multiple models in parallel. The average of all the predictions from different ensemble models is calculated. The majority vote gained from the voting mechanism is considered when classification is made. Bagging decreases the variance and tunes the prediction to an expected outcome.

Suppose you have a set D of d tuples. At every iteration i, a training set Di of the d tuples is chosen through row sampling using a replacement method from D. Subsequently, a classifier model Mi is learned for every training set D < i. Every classifier Mi provides its class prediction. Also, the bagged classifier M* calculates the votes and allocates the class with the highest votes to X (unidentified sample). This example of bagging in machine learning gives you an idea of how bagging work.

Implementation steps:

You can implement bagging in machine learning by following these steps.

Multiple subsets are prepared from the original data set with equal tuples. The observations with replacement are selected.

A base model is prepared on every subset.

iii. Every model is learned in parallel with the training set. These models are independent of each other.

The final predictions are done by merging the predictions from all the models.

Example of Bagging:

The Random Forest model uses Bagging, where decision tree models with higher variance are present. It makes random feature selection to grow trees. Several random trees make a Random Forest.

Boosting

Boosting is a sequential ensemble method that iteratively adjusts the weight of observation as per the last classification. If an observation is incorrectly classified, it increases the weight of that observation. The term 'Boosting' in a layman language, refers to algorithms that convert a weak learner to a stronger one. It decreases the bias error and builds strong predictive models.

Data points mis predicted in each iteration are spotted, and their weights are increased. The Boosting algorithm allocates weights to each resulting model during training. A learner with good training data prediction results will be assigned a higher weight. When evaluating a new learner, Boosting keeps track of learner's errors.

If a provided input is inappropriate, its weight is increased. The purpose behind this is that the forthcoming hypothesis is more likely to properly categorize it by combining the entire set, at last, to transform weak learners into superior performing models.

It involves several boosting algorithms. The original algorithms invented by Yoav Freund and Robert Schapire were not adaptive. They couldn't make the most of the weak learners. These people then invented AdaBoost, which is an adaptive boosting algorithm. It received the esteemed Gödel Prize and was the first successful boosting algorithm created for binary classification. AdaBoost stands for Adaptive Boosting. It merges multiple "weak classifiers" into a "strong classifier".

Gradient Boosting represents an extension of the boosting procedure. It equates to the combination of Gradient Descent and Boosting. It uses a gradient descent algorithm capable of optimizing any differentiable loss function. Its working involves the construction of an ensemble of trees, and individual trees are summed sequentially. The subsequent tree restores the loss (the difference between real and predicted values).

Just like the algorithm of bagging in machine learning, Boosting involves an algorithm with the following steps.

Implementation steps of a Boosting algorithm:

Initialize the dataset and allocate equal weight to every data point.

Offer this as input to the model and detect the incorrectly classified data points.

iii. Increase the incorrectly classified data points’ weights and decrease the correctly classified data points’ weights.

Normalize the weights of each data point.

Understanding the working of boosting and bagging in ML helps you to effectively carry out comparison between them. So, let’s understand how Boosting works.

The following steps are involved in the boosting technique:

A subset wherein every data point is provided equal weights is prepared from the training dataset.

This step prepares a based model is created for the initial dataset. This model helps you to perform predictions on the whole dataset.

iii. Errors are counted using actual and predicted values. The observation that was incorrectly predicted is provided a higher weight.

Boosting algorithm tries to correct the previous model’s errors.

The process is iterated for multiple models and each of them corrects the previous model’s errors.

The final model works as a strong learner and shows the weighted mean of all the models.

Example of Boosting:

The AdaBoost uses Boosting techniques, where a 50% less error is required to maintain the model. Here, Boosting can keep or discard a single learner. Otherwise, the iteration is repeated until achieving a better learner.

Similarities and Differences between Bagging and Boosting

Bagging and boosting, both being the popularly used methods, have a universal similarity of being classified as ensemble methods. Here we will highlight more similarities between them, followed by the differences they have from each other. Let us first start with similarities as understanding these will make understanding the differences easier.

Bagging and Boosting: Similarities

Bagging and Boosting are ensemble methods focused on getting N learners from a single learner.

Bagging and boosting make random sampling and generate several training data sets

Bagging and boosting arrive upon the end decision by making an average of N learners or taking the voting rank done by most of them.

Bagging and boosting reduce variance and provide higher stability with minimizing errors.

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

Ans. Out-of-bag (OOB) error, also called out-of-bag estimate, is a method of measuring the prediction error of random forests, boosted decision trees, and other machine learning models utilizing bootstrap aggregating (bagging). Bagging uses subsampling with replacement to create training samples for the model to learn from. OOB error is the mean prediction error on each training sample xi, using only the trees that did not have xi in their bootstrap sample.

Bootstrap aggregating allows one to define an out-of-bag estimate of the prediction performance improvement by evaluating predictions on those observations that were not used in the building of the next base learner.

Out-of-bag dataset

When bootstrap aggregating is performed, two independent sets are created. One set, the bootstrap sample, is the data chosen to be "in-the-bag" by sampling with replacement. The out-of-bag set is all data not chosen in the sampling process.

When this process is repeated, such as when building a random forest, many bootstrap samples and OOB sets are created. The OOB sets can be aggregated into one dataset, but each sample is only considered out-of-bag for the trees that do not include it in their bootstrap sample. The picture below shows that for each bag sampled, the data is separated into two groups.

9. What is K-fold cross-validation?

Ans. In this method, we split the data-set into k number of subsets(known as folds) then we perform training on the all the subsets but leave one(k-1) subset for the evaluation of the trained model. In this method, we iterate k times with a different subset reserved for testing purpose each time.

Note:

It is always suggested that the value of k should be 10 as the lower value

of k is takes towards validation and higher value of k leads to LOOCV method.

Example

Let's take an example of the training subsets and evaluation subsets generated in k-fold cross-validation. Here, we have total 25 instances. In first iteration we use the first 20 percent of data for evaluation, and the remaining 80 percent for training ([1-5] testing and [5-25] training) while in the second iteration we use the second subset of 20 percent for evaluation, and the remaining three subsets of the data for training ([5-10] testing and [1-5 and 10-25] training), and so on.

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

Ans. A Machine Learning model is defined as a mathematical model with a number of parameters that need to be learned from the data. By training a model with existing data, we are able to fit the model parameters.

However, there is another kind of parameter, known as Hyperparameters, that cannot be directly learned from the regular training process. They are usually fixed before the actual training process begins. These parameters express important properties of the model such as its complexity or how fast it should learn.

Some examples of model hyperparameters include:

The penalty in Logistic Regression Classifier i.e. L1 or L2 regularization

The learning rate for training a neural network.

The C and sigma hyperparameters for support vector machines.

The k in k-nearest neighbors.

The aim of this article is to explore various strategies to tune hyperparameters for Machine learning models.

Models can have many hyperparameters and finding the best combination of parameters can be treated as a search problem. The two best strategies for Hyperparameter tuning are

Hyperparameter tuning

A Machine Learning model is defined as a mathematical model with a number of parameters that need to be learned from the data. By training a model with existing data, we are able to fit the model parameters.

However, there is another kind of parameter, known as Hyperparameters, that cannot be directly learned from the regular training process. They are usually fixed before the actual training process begins. These parameters express important properties of the model such as its complexity or how fast it should learn.

Some examples of model hyperparameters include:

The penalty in Logistic Regression Classifier i.e. L1 or L2 regularization

The learning rate for training a neural network.

The C and sigma hyperparameters for support vector machines.

The k in k-nearest neighbors.

The aim of this article is to explore various strategies to tune hyperparameters for Machine learning models.

Models can have many hyperparameters and finding the best combination of parameters can be treated as a search problem. The two best strategies for Hyperparameter tuning are:

- GridSearchCV
- RandomizedSearchCV
- GridSearchCV

In GridSearchCV approach, the machine learning model is evaluated for a range of hyperparameter values. This approach is called GridSearchCV, because it searches for the best set of hyperparameters from a grid of hyperparameters values.

For example, if we want to set two hyperparameters C and Alpha of the Logistic Regression Classifier model, with different sets of values. The grid search technique will construct many versions of the model with all possible combinations of hyperparameters and will return the best one.

As in the image, for C = [0.1, 0.2, 0.3, 0.4, 0.5] and Alpha = [0.1, 0.2, 0.3, 0.4]. For a combination of C=0.3 and Alpha=0.2, the performance score comes out to be 0.726(Highest), therefore it is selected.

Drawback: GridSearchCV will go through all the intermediate combinations of hyperparameters which makes grid search computationally very expensive.

RandomizedSearchCV

RandomizedSearchCV solves the drawbacks of GridSearchCV, as it goes through only a fixed number of hyperparameter settings. It moves within the grid in a random fashion to find the best set of hyperparameters. This approach reduces unnecessary computation.

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

Ans. A large learning rate in gradient descent can cause a few issues:

1. Overshooting: With a high learning rate, the model's parameters may update too quickly and overshoot the optimal solution. This can cause the cost function to oscillate and converge to a suboptimal solution.
2. Divergence: If the learning rate is too large, the model's parameters may continue to increase or decrease indefinitely, causing the cost function to diverge.
3. Slow Convergence: Even if the cost function doesn't diverge, a large learning rate can cause the model to converge slowly, as the step size of the update is so large that the cost function oscillates around the minimum.
4. Instability: A large learning rate can cause the model to be unstable, resulting in fluctuating prediction.
5. Noise amplification: A large learning rate amplifies the noise in the data, which in turn affects the model's parameters, making the model overfitting.

To avoid these issues, it is usually recommended to use a smaller learning rate and to monitor the cost function during training to ensure that it is decreasing.

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

Ans. Logistic regression is a linear model, which means that it makes the assumption that the relationship between the input features and the output variable is linear. Therefore, logistic regression is not well suited for classification of non-linear data. The model will not be able to capture complex, non-linear relationships between the input features and the output variable, leading to poor performance on non-linear classification tasks.

In case of non-linear data, other techniques such as decision tree, random forest, k-NN, Neural networks, Support Vector Machines (SVMs) with a non-linear kernel, or a combination of these methods can be used. These models are capable of fitting non-linear decision boundaries and can capture complex, non-linear relationships between the input features and the output variable.

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.

In practice, this boosting technique is used with simple classification trees or stumps as base-learners, which resulted in improved performance compared to the classification by one tree or another single base-learner.

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.

The technique yields a direct interpretation of boosting methods from the perspective of numerical optimization in a function space and generalizes them by allowing optimization of an arbitrary loss function.

The Comparison

Loss Function:

The technique of Boosting uses various loss functions. In case of Adaptive Boosting or AdaBoost, it minimizes the exponential loss function that can make the algorithm sensitive to the outliers. With Gradient Boosting, any differentiable loss function can be utilized. Gradient Boosting algorithm is more robust to outliers than AdaBoost.

Flexibility

AdaBoost is the first designed boosting algorithm with a particular loss function. On the other hand, 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.

Benefits

AdaBoost minimizes 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 utilized 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.

Shortcomings

In the case of Gradient Boosting, the shortcomings of the existing weak learners can be identified by gradients and with AdaBoost, it can be identified by high-weight data points.

Wrapping Up

Though there are several differences between the two boosting methods, both the algorithms follow the same path and share similar historic roots. Both the algorithms work for boosting the performance of a simple base-learner by iteratively shifting the focus towards problematic observations that are challenging to predict.

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

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

Ans. Bias-Variance Trade off – Machine Learning

It is important to understand prediction errors (bias and variance) when it comes to accuracy in any machine learning algorithm. There is a tradeoff between a model’s ability to minimize bias and variance which is referred to as the best solution for selecting a value of Regularization constant. Proper understanding of these errors would help to avoid the overfitting and underfitting of a data set while training the algorithm.

Bias

The bias is known as the difference between the prediction of the values by the ML model and the correct value. Being high in biasing gives a large error in training as well as testing data. It's recommended that an algorithm should always be low biased to avoid the problem of underfitting.

By high bias, the data predicted is in a straight-line format, thus not fitting accurately in the data in the data set. Such fitting is known as Underfitting of Data. This happens when the hypothesis is too simple or linear in nature.

Variance

The variability of model prediction for a given data point which tells us spread of our data is called the variance of the model. The model with high variance has a very complex fit to the training data and thus is not able to fit accurately on the data which it hasn’t seen before. As a result, such models perform very well on training data but has high error rates on test data.

When a model is high on variance, it is then said to as Overfitting of Data. Overfitting is fitting the training set accurately via complex curve and high order hypothesis but is not the solution as the error with unseen data is high.

While training a data model variance should be kept low.

Bias Variance Tradeoff

If the algorithm is too simple (hypothesis with linear eq.) then it may be on high bias and low variance condition and thus is error-prone. If algorithms fit too complex (hypothesis with high degree eq.) then it may be on high variance and low bias. In the latter condition, the new entries will not perform well. Well, there is something between both of these conditions, known as Trade-off or Bias Variance Trade-off.

This tradeoff in complexity is why there is a tradeoff between bias and variance. An algorithm can't be more complex and less complex at the same time. For the graph, the perfect tradeoff will be like.

The best fit will be given by hypothesis on the tradeoff point.

This is referred to as the best point chosen for the training of the algorithm which gives low error in training as well as testing data.

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

Ans. Linear Kernel is used when the data is Linearly separable, that is, it can be separated using a single Line. It is one of the most common kernels to be used. It is mostly used when there are a large number of Features in a particular Data Set. One of the examples where there are a lot of features, is Text Classification, as each alphabet is a new feature. So, we mostly use Linear Kernel in Text Classification.

Advantages of using Linear Kernel:

1. Training a SVM with a Linear Kernel is Faster than with any other Kernel.

2. When training a SVM with a Linear Kernel, only the optimization of the C Regularization parameter is required. On the other hand, when training with other kernels, there is a need to optimize the γ parameter which means that performing a grid search will usually take more time.

RBF kernels are the most generalized form of kernelization and is one of the most widely used kernels due to its similarity to the Gaussian distribution. The RBF kernel function for two points X_1 and X_2 computes the similarity or how close they are to each other.

In a polynomial kernel for SVM, the data is mapped into a higher-dimensional space using a polynomial function. The dot product of the data points in the original space and the polynomial function in the new space is then taken. The polynomial kernel is often used in SVM classification problems where the data is not linearly separable. By mapping the data into a higher-dimensional space, the polynomial kernel can sometimes find a hyperplane that separates the classes.

The polynomial kernel has a number of parameters that can be tuned to improve its performance, including the degree of the polynomial and the coefficient of the polynomial.