**Q What is Machine Learning**

Ans: Machine Learning is about building system that can learn from data. Learning means getting better at some task, given some performance measure.

Machine Learning Types:

* **Regression:** In this algorithm learns to establish a relationship between input features and continuous output values. The goal is to predict numerical outcomes or quantities, making it useful for tasks such as price prediction, sales forecasting, and trend analysis.
* **Classification:** In this algorithm learns to categorize data into predefined classes based on labeled training examples. The goal is to predict the class of new, unseen data accurately.

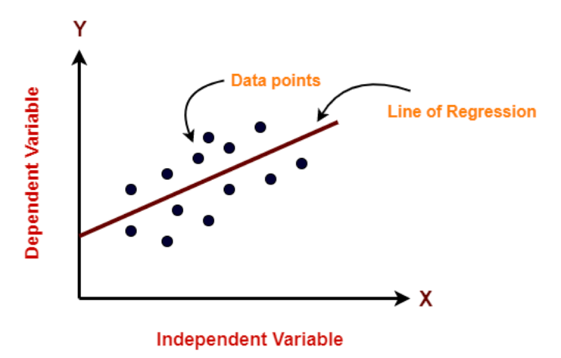
|  |  |
| --- | --- |
| Regression Techniques:   * Simple Linear Regression * Multiple linear Regression * Polynomial Linear Regression * Support Vector for Regression (SVR) * Decision Tree Regression * Random Forest Regression | Classification Techniques:   * Logistic Regression * Support Vector Machine SVM * Decision Tree Classifier * Random Forest Classifier * K-NN K-Nearest Neighbours * Kernel SVM * Naive Bayes * SGDClassifier |

1. **Linear regression**: Linear regression builds a model which establishes a relationship between feature (independent variable) and target (Dependent variable).

e.g House price: Target = house price, Features = [size, location]

f(x) = wX + b 🡪 np.dot(w, X) + b [X is feature value, w is weight, b is bias]

Numpy np function uses parallel hardware to efficiently calculate the dot product.

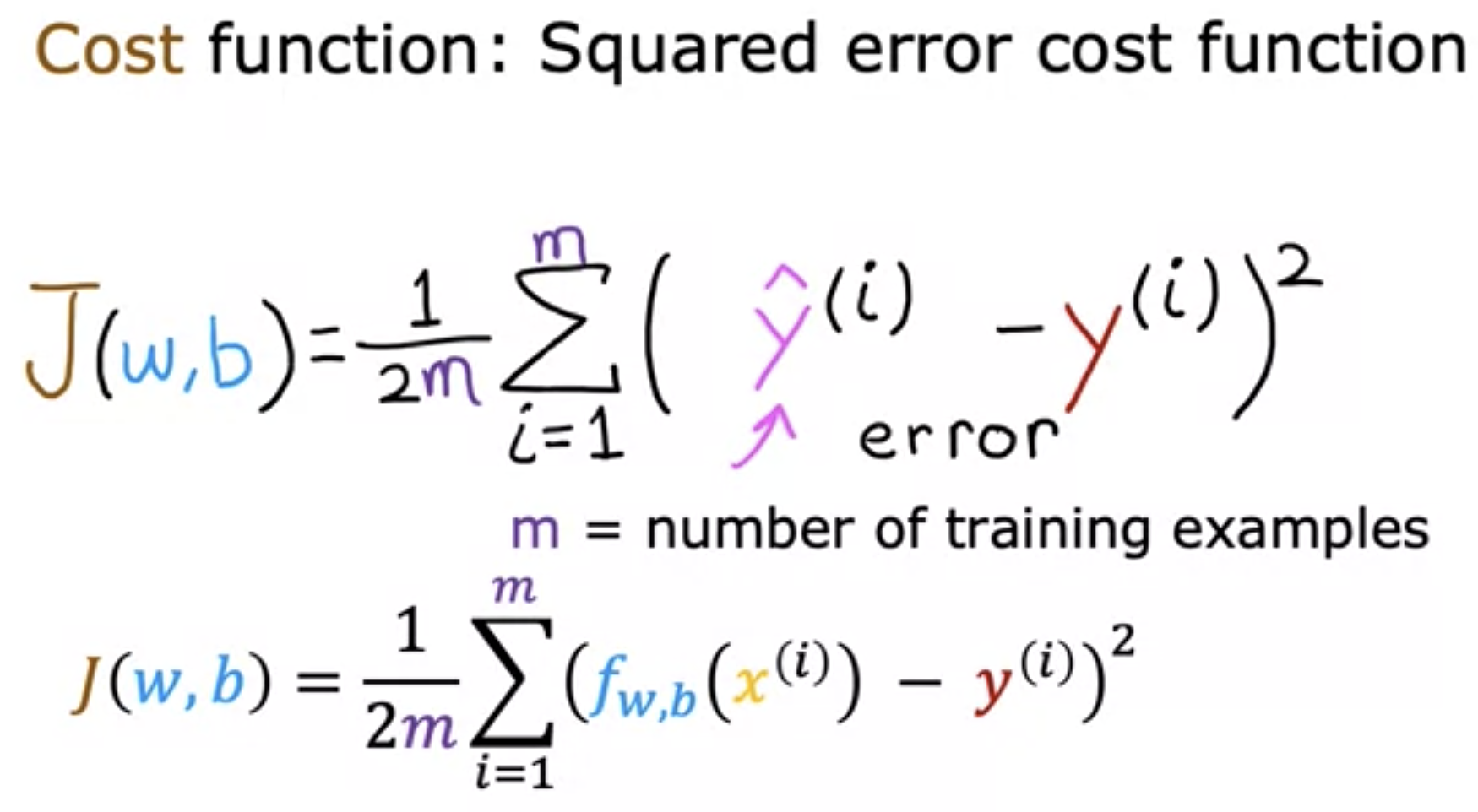


**Loss**: It is a measure of the difference of a single example to its target value.

(prediction – target 🡺 y\_hat - y)

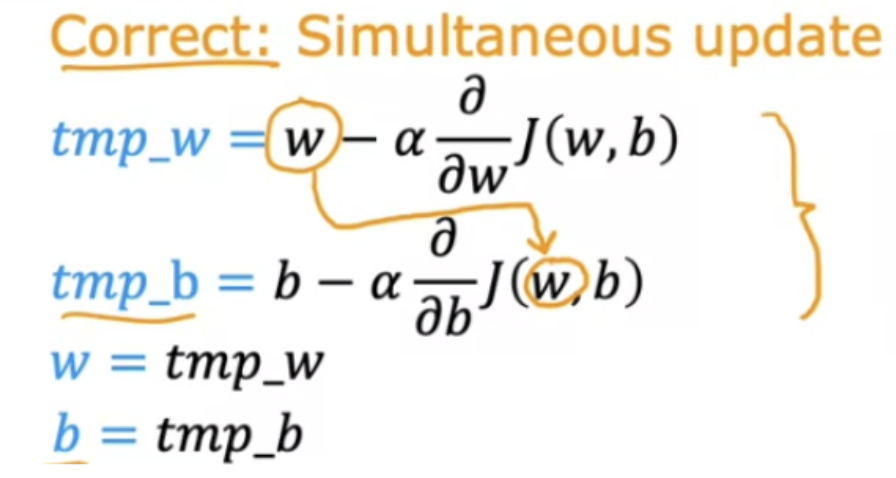
**Cost**: It is a measure how well our model is predicting the target.

Cost Function: Squared Error Cost Function



Gradient Decent: Gradient Descent is an iterative optimization algorithm used to minimize the loss function in machine learning models. It aims to find the optimal set of parameters (weights and biases) that result in the best performance of the model on the training data.

Repeat until convergence.

 here alpha is learning rate.

The learning rate is a crucial hyperparameter in Gradient Descent.

|  |  |
| --- | --- |
| If alpha/learning-rate is too small, Gradient Decent may be slow. | If alpha/learning-rate is too large, Gradient Decent may   * overshoot, never reach minimum. * Fail to converge, diverge   Diagram  Description automatically generated |

There are variations of Gradient Descent, such as Stochastic Gradient Descent (SGD), Mini-batch Gradient Descent, and Adam (Adaptive Moment Estimation), which use different strategies to update the parameters and address various issues like computational efficiency and adaptability to varying learning rates.

What is cross-validation?

Cross-validation is a technique used to assess a model's performance and generalization ability. The data is split into multiple subsets (folds), and the model is trained and evaluated on different combinations of these subsets to get a more reliable performance estimate.

**Bias & Variance Trade off**

**What is Bias (Underfitting)?**

Model with high bias pays very little attention to the training data and oversimplifies the model. It always leads to high error on training and test data.

* Error between average model prediction and ground truth
* The bias of the estimated function tells us the capacity of the underlying model to predict the values

**What is Variance (Overfitting)?**

Model with high variance pays a lot of attention to training data and does not generalise 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.

* Average variability in the model prediction for the given dataset
* The variance of the estimated function tells you how much the function can adjust to the change in the dataset.

Chart, scatter chart

Description automatically generated

**Why is Bias Variance Tradeoff?**

If our model is too simple and has very few parameters then it may have high bias and low variance. On the other hand if our model has large number of parameters then it’s going to have high variance and low bias. So we need to find the right/good balance without overfitting and underfitting the data.

**Underfitting**, Underfitting occurs when a machine learning model is too simple to capture the underlying patterns in the data, resulting in poor performance on both the training and test data. It often happens when the model lacks complexity or is trained on insufficient data.

How to prevent Underfitting:

* Use a more complex model with a higher capacity (e.g., increase the number of layers in a neural network).
* Increase the number of features or use more informative features.
* Train the model for a longer time or with more iterations.
* Reduce regularization to allow the model to fit the data better.
* Ensure you have enough diverse and representative training data.

**Overfitting**, Overfitting occurs when a machine learning model performs extremely well on the training data but poorly on unseen data. It happens when the model learns noise or specific patterns that are present only in the training data and fail to generalize to new data.

How to prevent **Overfitting**:

* To simplify the model by selecting one with fewer parameters (e.g., a linear model rather than a high-degree polynomial model), by reducing the number of attributes in the training data or by constraining the model
* Regularization (L1 and L2)
* Early Stopping
* To gather more training data
* To reduce the noise in the training data (e.g., fix data errors and remove outliers)
* Using Ensemble methods.

Q 9: What is an online learning System?

Ans: An online learning system can learn incrementally, as opposed to a batch learning system. This makes it capable of adopting rapidly to both changing data and autonomous system, and training on very large quantity of data.

Q 10: What is out-of-core learning?

Ans: Out-of-core algorithms can handle vast quantities of data that can’t fit in a computer main memory. An out-of-core learning algorithm chops the data into mini-batches and uses online learning techniques to learn from these mini batches.

Q 12: What is the difference between a model parameter and a learning algorithm Hyperparameter

Ans: A model has one or more model parameters that determine what it will predict given a new instance (e.g. the slope of a linear model). A learning algorithm tries to find optimal values for these parameters such that the model generalizes well to new instances. A Hyperparameter is a parameter of a learning algorithm itself, not of the model. (e.g. the amount of regularisation is to apply).

Why do we need a Train set, Validation set and Test set?

When training a model, we divide the available data into three separate sets:

* The training dataset is used for fitting the model’s parameters. However, the accuracy that we achieve on the training set is not reliable for predicting if the model will be accurate on new samples.
* The validation dataset is used to measure how well the model does on examples that weren’t part of the training dataset. The metrics computed on the validation data can be used to tune the hyperparameters of the model. However, every time we evaluate the validation data and we make decisions based on those scores, we are leaking information from the validation data into our model. The more evaluations, the more information is leaked. So we can end up overfitting to the validation data, and once again the validation score won’t be reliable for predicting the behaviour of the model in the real world.
* The test dataset is used to measure how well the model does on previously unseen examples. It should only be used once we have tuned the parameters using the validation set.

So if we omit the test set and only use a validation set, the validation score won’t be a good estimate of the generalization of the model.

**Q: What is an imbalanced dataset? Can you list some ways to deal with it?**

**Ans:** An imbalanced dataset is one that has different proportions of target categories. For example, a dataset with medical images where we have to detect some illness will typically have many more negative samples than positive samples—say, 98% of images are without the illness and 2% of images are with the illness.

There are different options to deal with imbalanced datasets:

* Oversampling or undersampling. Instead of sampling with a uniform distribution from the training dataset, we can use other distributions so the model sees a more balanced dataset.
* Data augmentation. We can add data in the less frequent categories by modifying existing data in a controlled way. In the example dataset, we could flip the images with illnesses, or add noise to copies of the images in such a way that the illness remains visible.
* Using appropriate metrics. In the example dataset, if we had a model that always made negative predictions, it would achieve a precision of 98%. There are other metrics such as precision, recall, and F1-score that describe the accuracy of the model better when using an imbalanced dataset.

**Fine Tune Your Model / Hyperparameter Tuning:**

Hyperparameter tuning, also known as hyperparameter optimization, is the process of finding the best combination of hyperparameters for a machine learning model. Hyperparameters are configuration settings that are not learned during the training process but are set before training begins. They significantly influence the model's performance and generalization ability.

The goal of hyperparameter tuning is to find the hyperparameter values that result in the best model performance on the validation or cross-validation set. The process typically involves trying different combinations of hyperparameter values and evaluating the model's performance using a chosen metric (e.g., accuracy, F1 score, or mean squared error).

There are several techniques for hyperparameter tuning, including:

* Grid Search: Exhaustively searching all possible combinations of hyperparameter values within specified ranges.
* Random Search: Sampling random combinations of hyperparameter values within specified ranges.
* Bayesian Optimization: An intelligent search algorithm that models the performance function and selects the next hyperparameter combination based on previous results.
* Genetic Algorithms: Using principles inspired by natural selection to evolve and improve hyperparameter combinations over successive iterations.
* Automated Machine Learning (AutoML): Utilizing automated tools or libraries that internally perform hyperparameter tuning along with other aspects of the machine learning pipeline.

Hyperparameter tuning is crucial to achieve optimal model performance and avoid issues like underfitting or overfitting. It helps fine-tune the model to better generalize and make accurate predictions on unseen data.

--------------- Code ---------------

from sklearn.model\_selection import GridSearchCV

param\_grid = [

{ n\_estimators=[3, 10, 30], max\_features=[2, 4, 6, 8] },

{ n\_estimators=[3, 10], max\_features=[2, 3, 4], bootstrap=False },

]

model = RandomForestRegressor()

grid\_search = GridSearchCV( model,

param\_grid,

cv=5,

scoring=‘negative\_mean\_squared\_error’,

return\_train\_score=True )

Numeric Imputer Strategy: mean, median, most\_frequent

Best combination of Hyperparameters:

grid\_search.best\_params\_

{ max\_features=8, n\_estimators=30}

Find best estimators:

estimator = grid\_search.best\_estimator\_

--------------- Code end ---------------

RandomSearchCV is best as compare to GridSearchCV

. It picks random hyper parameter values, e.g. 1000 iterations , this approach will explore 1000 different value of each hyper parameters.

. We are easily control over the computing budget by controlling one value iteration=500

Model Performance metrics:

Regression Metrics:

* Mean Absolute Error (MAE): It calculates the average absolute difference between the predicted and actual values, providing a measure of the model's average prediction error.
* Mean Squared Error (MSE): It calculates the average squared difference between the predicted and actual values, giving higher weight to larger errors compared to MAE.
* Root Mean Squared Error (RMSE): It is the square root of MSE, providing a measure of the average prediction error with the same unit as the target variable.
* R-squared (Coefficient of Determination): It measures the proportion of the variance in the target variable that can be explained by the model. It ranges from 0 to 1, with higher values indicating a better fit.

Classification Metrics:

* **Accuracy**: It measures the overall correctness of the model's predictions by calculating the ratio of correct predictions to the total number of predictions.
* **Precision**: Exactness of Model. It quantifies the proportion of correctly predicted positive instances out of all instances predicted as positive. It is particularly useful when the goal is to minimize false positives e.g (Health care, medical screening, drug testing).
* **Recall** (Sensitivity or True Positive Rate): Completeness of Model. It measures the proportion of actual positive instances correctly predicted by the model. It is useful when the goal is to minimize false negatives e.g. (Fraud Detection).
* **F1-Score**: It combines precision and recall into a single metric, providing a balanced measure of the model's performance. It is the harmonic mean of precision and recall.
* **Specificity** (True Negative Rate): It measures the proportion of actual negative instances correctly predicted by the model. It is particularly useful when the goal is to minimize false negatives.

Confusion Metric: Row represents actual classes, Column represents predictions

|  |  |
| --- | --- |
| \_\_ \_\_  | TP FP |  | 53057, 1522 |  | |  | 1325, 4096 |  | FN TN |  |\_\_ \_\_| | Precision : TP / (TP + FP) —> 53057 / (53057 + 1522)  Recall: TP / (TP + FN) —> 53057 / (53057+1325)  F1 Score: TP / ( TP + (FN + FP) / 2) |

Gradient Descent (GD): Gradient Descent is an iterative optimization algorithm used to minimize a cost function by updating model parameters in the direction of the steepest descent of the gradient. The key steps in GD are as follows:

* Compute the gradient of the cost function with respect to the model parameters.
* Update the parameters by taking a step proportional to the negative gradient multiplied by the learning rate.
* Repeat these steps until convergence or a stopping criterion is met.
* GD considers the entire training dataset in each iteration, making it computationally expensive for large datasets.

Stochastic Gradient Descent (SGD): Stochastic Gradient Descent is an extension of GD that uses a single randomly chosen training instance (or a small subset) to estimate the gradient and update the parameters. The main steps in SGD are as follows:

* Randomly select a training instance (or a small subset, known as a mini batch).
* Compute the gradient of the cost function using only the selected instance(s).
* Update the parameters using the gradient and the learning rate.
* Repeat these steps for a fixed number of iterations or until convergence.
* SGD is computationally more efficient than GD since it processes one instance (or a mini batch) at a time. However, its convergence can be noisy due to the high variance in gradient estimates.

Mini-Batch Gradient Descent (MBGD): Mini-Batch Gradient Descent is a compromise between GD and SGD. It performs parameter updates based on a small randomly selected subset of the training data (mini batch) instead of using the entire dataset (as in GD) or a single instance (as in SGD). The key steps in MBGD are:

* Randomly select a mini batch of training instances.
* Compute the gradient of the cost function using the selected mini batch.
* Update the parameters using the gradient and the learning rate.
* Repeat these steps for a fixed number of iterations or until convergence.
* MBGD offers a balance between efficiency and stability. It reduces the noise compared to SGD since it uses a mini batch, leading to smoother convergence, and is less computationally expensive than GD.

Comparison between Gradient Descent, Stochastic Gradient Descent and Batch Gradient Descent:

Gradient Descent (GD):

* Advantages: GD guarantees convergence to the global minimum of the cost function for convex problems. It provides a stable and deterministic convergence path.
* Considerations: GD can be computationally expensive, especially for large datasets, as it requires processing the entire dataset in each iteration. It may also get stuck in local minima for non-convex problems.

Stochastic Gradient Descent (SGD):

* Advantages: SGD is computationally efficient as it updates parameters based on a single randomly selected instance or a small subset of instances. It works well for large datasets and non-convex problems.
* Considerations: SGD's convergence can be noisy and fluctuate due to the high variance in gradient estimates. It may require careful tuning of the learning rate to ensure convergence.

Mini-Batch Gradient Descent (MBGD):

* Advantages: MBGD offers a balance between GD and SGD. It processes a mini batch of instances, providing a smoother convergence compared to SGD. It is more computationally efficient than GD.
* Considerations: The mini-batch size in MBGD needs to be chosen carefully. A very small size may introduce noise, while a very large size may slow down convergence. It requires tuning the learning rate and mini-batch size.

In practice, the choice between GD, SGD, or MBGD depends on various factors:

* Dataset size: For large datasets, SGD or MBGD is usually preferred due to computational efficiency.
* Convergence stability: GD provides a stable convergence path, but SGD and MBGD offer faster convergence, although with some noise.
* Computational resources: If computational resources are limited, SGD or MBGD may be preferable due to their efficiency.
* Problem type: For convex problems, GD is generally sufficient. For non-convex problems or deep learning models, SGD or MBGD is commonly used.

**Normalization**: Data normalization is very important pre-processing step, used to rescale values to fit in a specific range to assure better convergence during backpropagation. In general, it boils down to subtracting the mean of each data point and dividing by its standard deviation. If we don't do this then some of the features (those with high magnitude) will be weighted more in the cost function (if a higher-magnitude feature changes by 1%, then that change is pretty big, but for smaller features it's quite insignificant). The data normalization makes all features weighted equally.

**Regularization** is a technique used in machine learning to prevent overfitting, which occurs when a model performs well on the training data but fails to generalize to new, unseen data. Regularization introduces a penalty term to the model's objective function, discouraging complex or overly flexible models. The aim is to find a balance between model complexity and generalization performance. Here are two commonly used regularization techniques:

* L1 Regularization (Lasso): L1 regularization adds a penalty term to the loss function that is proportional to the absolute value of the model's parameter values. This encourages sparse solutions by driving some of the parameter values to zero, effectively performing feature selection. L1 regularization can help identify and emphasize the most important features in the model.
* L2 Regularization (Ridge): L2 regularization adds a penalty term to the loss function that is proportional to the squared magnitude of the model's parameter values. This encourages the model to distribute the weights across all features more evenly, reducing the impact of any individual feature. L2 regularization can help improve the model's generalization performance by reducing the model's sensitivity to small fluctuations in the input data.
* Elastic Net regularization is a combination of L1 (Lasso) and L2 (Ridge) regularization techniques. It addresses some limitations of individual regularization methods and provides a more flexible regularization approach. Elastic Net introduces a mixing parameter, α, that controls the balance between the L1 and L2 regularization terms in the model's objective function.

lesso\_regressor = SGDRegressor(penality=‘’L1’)

ridge\_regressor = SGDRegressor(penality=‘’L2’)

elastic\_net\_regressor = SGDRegressor(penality=”L1”, l1\_ration=0.5)

Both L1 and L2 regularization techniques help to prevent overfitting by controlling the complexity of the model. By adding the regularization term to the loss function, the model is incentivized to minimize both the data fitting error and the regularization penalty. The regularization term is typically controlled by a hyperparameter, lambda (λ), which determines the strength of the regularization effect. A higher value of λ leads to stronger regularization, which can reduce model complexity and enhance generalization.

Regularization techniques offer several benefits, including:

* Improved generalization: Regularization reduces overfitting, allowing the model to perform better on unseen data.
* Feature selection: L1 regularization can effectively select relevant features by driving some weights to zero.
* Robustness to noise: L2 regularization can mitigate the impact of noisy or irrelevant features on the model's predictions.
* More stable solutions: Regularization can lead to more stable and reproducible model training by reducing sensitivity to small changes in the input data.

Which Regularization Ridge, Lasso or Elastic Net is best?

* Ridge (l2) is a good default.
* Lesso (l1) is good, If you suspect that only a few features are useful you should prefer Lasso or Elastic net because they tend to reduce the useless features, weight down to zero.
* Elastic net is preferred over Lasso because Lasso may behave erratically when the number of features is greater than number of training instance or when several features are strongly correlated.

# Support Vector Machine

What is fundamental idea behind Support Vector Machine?

|  |  |
| --- | --- |
| The fundamental idea behind Support Vector Machine is to fit the widest possible “Street” between the classes. In other words, the goal is to have the largest possible margin between the decision boundary that separates the two classes and the training instances. |  |

When performing the soft margin classification, the SVM searches for a compromise between perfectly separating the two classes and having the widest possible street (i.e., few instances may end up on the street).

What is a Support Vector?

After training a support vector machine SVM, a support vector is any instance located on the street, including its border.

Explain the concept of the hyperplane in SVM.

In SVM, the hyperplane is a decision boundary that separates data points of different classes. For a binary classification problem, it is a flat (for linear SVM) or curved (for non-linear SVM) surface that maximizes the margin between the closest data points of each class.

Why it is important to scale the input when using SVMs?

SVMs try to fit the largest possible “street” between the classes, so if the training set is not scaled, the SVM will tend to neglect small features.

Can an SVM classifier output a confidence score when it classifies an instance? What about a probability?

An SVM classifier can output the distance between the test instance and the decision boundary, and you can use this as a confidence score. However, this confidence score cannot be directly converted into an estimation of the class probability.

Should you use the primal or the dual form of the SVM problem to train a model on a training set with millions of instances or hundreds of features?

This question is applying only to the linear SVMs, since kernelized SVM can only use the dual form. The computational complexity of the primal form of the SVM problem is proportional to the number of training instances m, while the computational complexity of the dual for is proportional to a number between n2 or n3. So, if there are millions of instances you should definitely use the primal form, because the dual form will be much too slow.

If an SVM classifier trained with an RBF kernel underfits the training set, there might be too much regularization. To decrease it you need to increase gamma or C both values.

How does SVM handle non-linearly separable data?

SVM handles non-linearly separable data by using kernel functions, which implicitly map the data into a higher-dimensional space where it becomes linearly separable. This allows SVM to find a separating hyperplane in that transformed space.

What are the different types of kernels used in SVM? Provide examples.

* 1. Different types of kernels used in SVM include:

1. Linear Kernel: K(x, y) = x \* y (No explicit transformation)
2. Polynomial Kernel: K(x, y) = (x \* y + c)^d
3. Radial Basis Function (RBF) Kernel: K(x, y) = exp(-γ \* ||x - y||^2)
4. Sigmoid Kernel.

What is the difference between linear SVM and logistic regression?

Linear SVM finds the optimal hyperplane that best separates data points, while logistic regression estimates the probability of a data point belonging to a specific class.

How do you handle imbalanced datasets when using SVM?

To handle imbalanced datasets in SVM, you can use techniques like class weighting, resampling (over-sampling or under-sampling), or using different performance metrics like F1-score or area under the Receiver Operating Characteristic (ROC) curve.

Discuss the advantages and disadvantages of SVM.

* 1. Advantages:
  + Effective in high-dimensional spaces.
  + Works well with limited data.
  + Versatile due to the kernel trick.
  + Effective in handling non-linear data.
  1. Disadvantages:
  + Computationally expensive for large datasets.
  + Prone to overfitting if not properly tuned.
  + Difficult to interpret the model.

Can SVM be used for regression tasks? If yes, how?

Yes, SVM can be used for regression tasks. It is called Support Vector Regression (SVR), where instead of finding a hyperplane, it finds a hyperplane that approximates the data points within a certain margin (epsilon) around the target values.

What are some real-world applications of SVM?

Some real-world applications of SVM include text classification, image recognition, bioinformatics, sentiment analysis, and medical diagnosis.

How do you tune the hyperparameters of an SVM model?

Hyperparameter tuning in SVM can be done using techniques like cross-validation, grid search, or random search to find the best combination of parameters like the kernel type, C value, and kernel-specific parameters (e.g., gamma for RBF).

How does SVM compare to other classification algorithms like Random Forest or Neural Networks?

SVM can perform well in high-dimensional spaces and is effective for small to medium-sized datasets. Random Forest is an ensemble method suitable for handling complex features and large datasets. Neural Networks excel in deep learning tasks and can handle very large datasets but may require more data and computational resources.

# Decision Tree

|  |  |
| --- | --- |
| What is a Decision Tree?  A Decision Tree is a supervised machine learning algorithm used for both classification and regression tasks. It recursively splits the data into subsets based on the most significant attribute, forming a tree-like structure that facilitates decision-making.  How does a Decision Tree make decisions?  A Decision Tree makes decisions by evaluating feature attributes at each node and splitting the data based on the attribute that best separates the data into homogeneous subsets with respect to the target variable (for classification) or minimizes the variance (for regression). |  |

What is the difference between Gini impurity and Entropy as impurity measures?

Gini impurity measures the probability of incorrectly classifying a randomly chosen element, while Entropy measures the average amount of information needed to identify the class label of a randomly chosen element. In practice, Gini impurity tends to be slightly faster to compute.

What is Entropy in the context of Decision Trees?

Entropy is a measure of the impurity or randomness in a dataset. In Decision Trees, it is used as a criterion to determine the best attribute for splitting the data at each node. Lower entropy means a more homogeneous subset, making it a preferred split.

What is Information Gain, and how is it related to Entropy?

Information Gain is the reduction in entropy achieved by splitting the data on a particular attribute. It measures the effectiveness of an attribute in classifying the data. Decision Trees choose the attribute with the highest Information Gain as the best split.

How do Decision Trees handle categorical variables?

Decision Trees handle categorical variables by creating branches for each category and splitting the data accordingly. If a categorical attribute has 'n' categories, it will have 'n' branches at that node.

What is Pruning in Decision Trees?

Pruning is a technique used to prevent overfitting in Decision Trees. It involves removing branches that do not contribute significantly to improving accuracy on the validation set, simplifying the tree and improving generalization.

Can Decision Trees handle missing values in data?

Yes, Decision Trees can handle missing values in data. Various strategies can be used to handle missing values, such as imputation or assigning a separate branch for missing values.

How do Decision Trees handle outliers?

Decision Trees are generally robust to outliers as they make decisions based on majority voting within leaf nodes. However, outliers might lead to the creation of unnecessary branches, potentially affecting the interpretability and performance of the tree.

What are the advantages of Decision Trees?

Advantages of Decision Trees include:

* Simple to understand and interpret.
* Handle both numerical and categorical data.
* Require little data preprocessing (e.g., feature scaling).
* Can be visualized easily.

What are the disadvantages of Decision Trees?

Disadvantages of Decision Trees include:

* Prone to overfitting, especially on complex datasets.
* Sensitive to small changes in data, leading to different tree structures.
* May not generalize well to unseen data without proper regularization.

Can Decision Trees be used for regression tasks?

Yes, Decision Trees can be used for regression tasks. The algorithm is called Decision Tree Regression, where the target variable is predicted by the average (for mean squared error) or median (for mean absolute error) of the target values within each leaf node.

How do you handle overfitting in Decision Trees?

Overfitting in Decision Trees can be handled through techniques like pruning, setting a maximum depth for the tree, using a minimum number of samples required for a node to be split, and adjusting other hyperparameters.

When would you choose Decision Trees over other machine learning algorithms like SVM or Neural Networks?

Decision Trees are a good choice when interpretability and ease of understanding are important. They work well with a mix of numerical and categorical features and are suitable for small to medium-sized datasets. SVM and Neural Networks may outperform Decision Trees on very large and complex datasets.

How do you deal with a dataset that has high dimensionality when using Decision Trees?

High dimensionality can lead to increased computational complexity and overfitting in Decision Trees. Techniques like feature selection, dimensionality reduction (e.g., PCA), and proper tuning of hyperparameters can help handle this issue.

Can Decision Trees be used for multi-class classification tasks?

Yes, Decision Trees can handle multi-class classification tasks using techniques like One-vs-One or One-vs-All.

What is the role of the CART algorithm in Decision Trees?

CART (Classification and Regression Trees) is a popular algorithm used to build Decision Trees. It uses binary splits to recursively partition the data based on the attribute that optimally separates the target variable.

Can Decision Trees handle continuous (numeric) target variables in regression tasks?

Yes, Decision Trees can handle continuous target variables in regression tasks. The algorithm is known as Decision Tree Regression, and the predicted value at each leaf node is usually the average or median of the target values in that node.

Hyperparametr of Decision Tree:

1. criterion: The function used to measure the quality of a split. It can take two values:

* "gini" for Gini impurity (used for classification tasks).
* "mse" for mean squared error (used for regression tasks).

1. splitter: The strategy used to choose the attribute to split the data. It can take two values:

"best" to choose the best attribute for the split based on the criterion.

"random" to choose a random attribute for the split.

1. max\_depth: The maximum depth of the tree. It limits the number of nodes in the tree. Setting this can help prevent overfitting.
2. min\_samples\_split: The minimum number of samples required to split an internal node. It helps control tree growth by setting a threshold on the number of samples in a node before attempting to split it.
3. min\_samples\_leaf: The minimum number of samples required to be at a leaf node. It sets a threshold on the number of samples in a leaf node.
4. min\_weight\_fraction\_leaf: The minimum weighted fraction of the total samples required to be at a leaf node. It is similar to min\_samples\_leaf but expressed as a fraction of the sum of instance weights.
5. max\_features: The number of features to consider when looking for the best split. It can take several values:

* "auto" (default) - considers all features for the split.
* "sqrt" - considers the square root of the number of features.
* "log2" - considers the log base 2 of the number of features.
* None - considers all features for the split.

1. random\_state: The seed value used by the random number generator. It ensures reproducibility of results.
2. ccp\_alpha: The complexity parameter used for Minimal Cost-Complexity Pruning. It controls the amount of regularization applied to the tree.

# Ensemble Learning and Random Forest

|  |  |
| --- | --- |
| What are Ensemble methods in the context of Decision Trees?  Ensemble methods combine multiple Decision Trees to improve prediction accuracy and generalization. Examples include Random Forest (bagging) and Gradient Boosting (boosting). |  |

A screenshot of a computer program

Description automatically generated

Explain the Random Forest algorithm.

Random Forest is an ensemble learning method that builds multiple Decision Trees using random subsets of the data (bootstrapping) and random subsets of features at each split. The final prediction is made by averaging or voting among the individual tree predictions.

Reference: [Hands-On Machine Learning with Scikit-Learn and TensorFlow](http://powerunit-ju.com/wp-content/uploads/2021/04/Aurelien-Geron-Hands-On-Machine-Learning-with-Scikit-Learn-Keras-and-Tensorflow_-Concepts-Tools-and-Techniques-to-Build-Intelligent-Systems-OReilly-Media-2019.pdf)