The process of building machine learning models can be divided into these main stages:

1. **Domain Understanding**

knowing where you are going, what your destination is, and what you want to deliver. Therefore, you need to clearly define the business problem you intend to solve.

1. **Data Collection and Preparation**

the next step is to collect and prepare the data that will feed the model. This involves identifying data sources, gathering the necessary datasets, and performing cleaning and preprocessing tasks to ensure the data is consistent, relevant, and free from errors or unwanted noise.

1. **Data Exploration and Analysis**

With the data collected and prepared, it is essential to explore and analyze the data to understand its structure, distribution, and patterns. This can involve data visualization, calculating descriptive statistics, and identifying potential anomalies or outliers. Here, in this stage, we have various activities, techniques, and procedures where we can explore and detect problems such as missing values, for instance. We then apply a cleaning technique and have to choose the ideal technique according to the dataset, according to what we will do next, and so on.

1. **Feature Selection and Engineering**

Feature selection involves choosing the most informative attributes, while feature engineering involves creating new attributes from existing ones to better represent patterns in the data.

1. **Model Building and Evaluation**

In this stage, machine learning models are built using the prepared data. Once built, it is crucial to evaluate their performance on test datasets to understand their accuracy, robustness, and generalization.We cannot know which algorithm is ideal beforehand. This is where the science comes in again. We need to experiment.And note, for each model with each algorithm, there is still the next stage, which is hyperparameter optimization. In other words, you need to know the possibilities each algorithm offers. Easily, you can create 10, 15, 20 versions of the same model until you reach the near-ideal model.

1. **Hyperparameter Optimization**

Hyperparameter optimization involves experimenting with different values to find the combination that produces the best model performance.

Do you know beforehand what the best hyperparameters are? No, you don’t. So, what do you have to do? You have to experiment.

Hyperparameter optimization can be the job of a data scientist or already be the assignment of a machine learning engineer, including this within the CI/CD pipeline.You can automate a good part of this work, but this is essential for having a well-tuned model, a model with high accuracy, a stable model. This is a fundamental part of the entire process.

1. **Cross-Validation**

You can automate a good part of this work, but this is essential for having a well-tuned model, a model with high accuracy, a stable model. This is a fundamental part of the entire process.We want the model to look at the data pattern and establish a mathematical generalization. If it achieves this, we can use the model to make predictions from new data. If it does not achieve this, we will likely face issues such as underfitting and overfitting.How can the relationship of the data be explained generally? This is what we want. If we achieve this, then you just need to provide new data to the model, and it will be able to make predictions.

So, this is the main goal of performing all the previous work until reaching cross-validation in the model preparation.

Cross-validation helps to check the generalization capability of the model, which is our main goal. And if you create a CI/CD pipeline, this will likely be part of your pipeline.

1. **Model Deployment**

Once the model is finalized and optimized, it is implemented or put into production, which is basically the synonym we use for deployment. This means that the model is made available in an environment that can receive new data, process it, and provide predictions or insights in real-time. And what exactly would this environment be? It depends on how the company wants to offer its model. Will it create a web application? An application for smartphones? Create an API, make a call from another application? Generate predictions in CSV format, then load it into Power BI to create a dashboard. It depends on how the company wants to deliver it. If the company does not know, then we can offer alternatives. Can that model be placed on the company’s web page? Will it be used on the internet or intranet? Is it just an API that we need to feed another existing application? What are the possibilities? Well, this is where we will define how we will do the deployment.

When deploying, we have small details that need to be considered.

Any transformation applied to the training data m

1. **Iteration and Continuous Improvement**

Because the data pattern changed.

So, do I have to keep retraining the model all the time? I wouldn’t say all the time, but with some frequency, certainly.

You might need to retrain the model once a week, or once a month. More than that, it starts to get complicated depending, of course, on the business area, the data volume, etc. But the fact is, the data pattern can change. This is the concept of Data Drift and Model Drift, which is when the pattern changes. Did the pattern change? Well, then you need to go and adjust the model as necessary. And this is part of the iteration and continuous improvement stage.

Building a machine learning model is not a one-time process. As new data becomes available, or the domain needs change, it is essential to revisit, re-evaluate, and, if necessary, refine or rebuild the model to ensure it remains relevant and effective.

**We will cover the following strategies and techniques for model deployment:**

* 1
* Shadow evaluation
* 2
* A/B testing
* 3
* Multi Arm Bandits
* 4
* Blue-green deployment
* 5
* Canary testing
* 6
* Feature flag
* 7
* Rolling deployment
* 8
* Recreate strategy

These strategies can be broken down into two categories:

* **Static deployment strategies**: These are the strategies where the distribution of traffic or request are handled manually. Examples of this are shadow evaluation, A/B testing, Canary testing, Rolling deployment, Blue-green deployment et cetera.
* **Dynamic deployment strategies:** These are the strategies where the distribution of traffic or request are handled automatically. Example of this is Multi Arm Bandits.

Now, another term that you must be familiar with is [**MLOps**](https://neptune.ai/blog/mlops). MLOps is generally a set of practices that enables ML Lifecycle. Its stitches machine learning and software applications together. Simply put, it is a collaboration between data scientists and the operations team that takes care of and orchestrates the whole ML lifecycle. The three key areas that MLOps focuses on are **continuous integration**, **continuous deployment,** and **continuous testing**

### **A/B testing model deployment strategy**

**A/B testing is a data-based strategy method. It is used to evaluate two models namely A and B, to assess which one performs better in a controlled environment. It is primarily used in e-commerce websites and social media platforms. With A/B testing the data scientists can evaluate and choose the best design for the website based on the data received from the users.**

**The two models differ slightly in terms of features and they cater to different sets of users. Based on the interaction and data received from the users such as feedback, data scientists choose one of the models that can be deployed globally into production.**

#### **Methodology**

**In A/B the two models are set up parallelly with different features. The aim is to increase the conversion rate of a given model. To do that data scientist sets up a hypothesis. A hypothesis is an assumption based on an abstract intuition of the data. This assumption is proposed through an experiment, if the assumption passes the test it is accepted as fact and the model is accepted, otherwise, it’s rejected.**

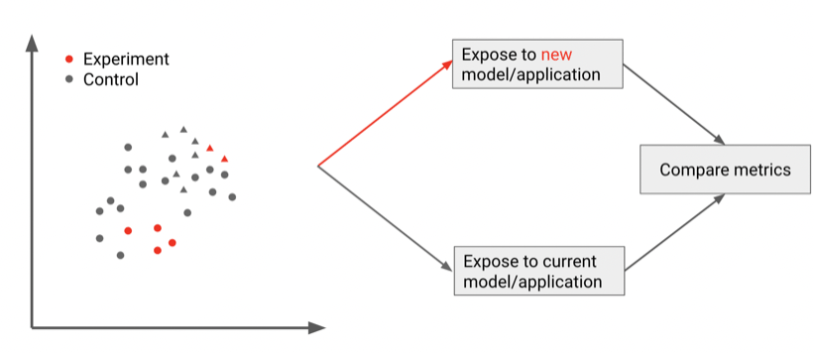
##### **Hypothesis testing**

**In A/B testing there are two types of hypothesis:**

* **1**
* **Null Hypothesis states that the phenomenon occurring in the model is purely out of chance and not because of a certain feature.**
* **2**
* **Alternate Hypothesis challenges the null hypothesis by stating that the phenomenon occurring in the model is because of a certain feature.**

**In** [**hypothesis testing**](https://www.analyticsvidhya.com/blog/2021/09/hypothesis-testing-in-machine-learning-everything-you-need-to-know/)**, the aim is to reject the null hypothesis by setting up experiments like the A/B testing and exposing the new model with a certain feature to a few users. The new model essentially is designed on an alternate hypothesis. If the alternate hypothesis is accepted and the null hypothesis is rejected then that feature is added and the new model is deployed globally.**

**It is important to know that in order to reject the null hypothesis you have to prove the** [**statistical significance**](https://www.investopedia.com/terms/s/statistical-significance.asp#:~:text=Statistical%20significance%20refers%20to%20the,attributable%20to%20a%20specific%20cause.) **of the test.**

****A/B testing model deployment strategy | [Source](https://www.oreilly.com/library/view/building-machine-learning/9781492045106/)

**Advantages:**

* **It is simple.**
* **Yields quick results and helps in the elimination of the low performing model.**

**Disadvantages:**

* **Models can be unreliable if the complexity is increased. One should use A/B testing in the case of simple hypothesis testing.**

**When to use it?**

**As mentioned earlier, A/B testing is predominantly used for e-commerce, social media platforms, and online streaming platforms. In such a setting and if you have two models you can use A/B to evaluate and choose which one to deploy globally.**

## Statistical significance of the Test

**Now, the main question is – Can we conclude from here that the Test group is working better than the control group?**

**The answer to this is a simple No! To reject our null hypothesis, we must prove our test's Statistical significance.**

<https://www.analyticsvidhya.com/blog/2020/10/ab-testing-data-science/>

# target Encoding[¶](https://www.kaggle.com/code/ryanholbrook/target-encoding#Target-Encoding)

A **target encoding** is any kind of encoding that replaces a feature's categories with some number derived from the target.

This kind of target encoding is sometimes called a **mean encoding**. Applied to a binary target, it's also called **bin counting**. (Other names you might come across include: likelihood encoding, impact encoding, and leave-one-out encoding.)

# Smoothing

An encoding like this presents a couple of problems, however. First are unknown categories. Target encodings create a special risk of overfitting, which means they need to be trained on an independent "encoding" split. When you join the encoding to future splits, Pandas will fill in missing values for any categories not present in the encoding split. These missing values you would have to impute somehow.

Second are rare categories. When a category only occurs a few times in the dataset, any statistics calculated on its group are unlikely to be very accurate. In the Automobiles dataset, the mercurcy make only occurs once. The "mean" price we calculated is just the price of that one vehicle, which might not be very representative of any Mercuries we might see in the future. Target encoding rare categories can make overfitting more likely.

**A solution to these problems is to add smoothing. The idea is to blend the in-category average with the overall average. Rare categories get less weight on their category average, while missing categories just get the overall average.**

**In pseudocode:**

**encoding = weight \* in\_category + (1 - weight) \* overall**

**where weight is a value between 0 and 1 calculated from the category frequency.**

**An easy way to determine the value for weight is to compute an m-estimate:**

**weight = n / (n + m)**

**where n is the total number of times that category occurs in the data. The parameter m determines the "smoothing factor". Larger values of m put more weight on the overall estimate.**

**Use Cases for Target Encoding**

**Target encoding is great for:**

* **High-cardinality features: A feature with a large number of categories can be troublesome to encode: a one-hot encoding would generate too many features and alternatives, like a label encoding, might not be appropriate for that feature. A target encoding derives numbers for the categories using the feature's most important property: its relationship with the target.**
* **Domain-motivated features: From prior experience, you might suspect that a categorical feature should be important even if it scored poorly with a feature metric. A target encoding can help reveal a feature's true informativeness.**

### **Precision (Minimize False Positives)**

**Scenario: Finance (Credit Card Fraud Detection)**

* **Context:** In fraud detection systems, a false positive represents a legitimate transaction flagged as fraudulent.
* **Consequence:** High false positives can lead to customer dissatisfaction due to frequent and unnecessary transaction blocks, leading to a poor user experience and potential loss of clients.
* **Why Precision:** High precision ensures that when a transaction is flagged as fraudulent, it is very likely to be truly fraudulent, minimizing inconvenience to customers.
* tp/tp+fp

### **Recall (Minimize False Negatives)**

**Scenario: Healthcare (Cancer Screening)**

* **Context:** In medical screening tests, a false negative result means failing to identify a disease when it is actually present.
* **Consequence:** A high rate of false negatives can be life-threatening, as patients might not receive the necessary treatments in time.
* **Why Recall:** Maximizing recall in this context ensures that nearly all patients with the disease are correctly identified and receive timely treatment, which is critical in life-threatening conditions.
* tp/tp+fn

### **Specificity (Maximize True Negatives)**

**Scenario: Manufacturing (Quality Control)**

* **Context:** In a manufacturing process, specificity refers to the ability of a quality control system to correctly identify products that meet quality standards (true negatives).
* **Consequence:** Poor specificity results in non-defective products being discarded or reworked unnecessarily, increasing costs and wasting resources.
* **Why Specificity:** High specificity ensures that good quality products are not incorrectly rejected, optimizing production costs and resource use.

### **Accuracy**

**Scenario: Marketing (Targeted Advertising)**

* **Context:** In targeted advertising, accuracy reflects the overall correctness of the campaign in identifying who will respond positively (click or buy) and who will not.
* **Consequence:** In a balanced dataset where the target population is evenly split between those who will and will not respond, low accuracy can lead to ineffective ad spend and lower ROI.
* **Why Accuracy:** High accuracy is important when the costs of false positives and false negatives are relatively balanced, ensuring the campaign reaches the most appropriate audience effectively.

### **Balancing These Metrics**

Often, the focus on one metric can affect others due to the trade-offs inherent in statistical modeling. For example:

* **Trade-off Between Recall and Precision:** Increasing recall (reducing false negatives) in a cancer screening scenario might lead to a decrease in precision, resulting in more false positives (e.g., people undergoing unnecessary further testing).
* **Balancing with F1 Score:** In scenarios where both precision and recall are important but there is a need to find a balance (like in document classification or email spam detection), the F1 Score (the harmonic mean of precision and recall) is used as a single metric to balance between the two.

Understanding these trade-offs and choosing the right metric based on real-world needs and consequences is essential for developing effective, efficient, and ethically responsible predictive models.

A confidence interval, in statistics, refers to the probability that a [population](https://www.investopedia.com/terms/p/population.asp) parameter will fall between a set of values for a certain proportion of times. Analysts often use confidence intervals that contain either 95% or 99% of expected observations. Thus, if a point estimate is generated from a statistical model of 10.00 with a 95% confidence interval of 9.50 to 10.50, it means one is 95% confident that the true value falls within that range.

* A confidence interval displays the probability that a parameter will fall between a pair of values around the mean.
* Confidence intervals measure the degree of uncertainty or certainty in a sampling method.
* They are also used in hypothesis testing and regression analysis.
* Statisticians often use p-values in conjunction with confidence intervals to gauge statistical significance.
* They are most often constructed using confidence levels of 95% or 99%.

Regularization is a technique used in machine learning to penalize complex models to protect them from overfitting.

****

## **What Is Principal Component Analysis?**

Principal component analysis (PCA) is a dimensionality reduction and machine learning method used to simplify a large data set into a smaller set while still maintaining significant patterns and trends.

## **How Do You Do a Principal Component Analysis?**

1. Standardize the range of continuous initial variables
2. Compute the covariance matrix to identify correlations
3. Compute the eigenvectors and eigenvalues of the covariance matrix to identify the principal components
4. Create a feature vector to decide which principal components to keep
5. Recast the data along the axes of the principal component

### **1. Standardize the Range of Continuous Initial Variables**

Standardizing continuous variables is necessary because PCA is affected by the scale of the data. By standardizing, each variable will have a mean of 0 and a standard deviation of 1.

* **How to Standardize:**
  + For each continuous variable, subtract the mean and divide by the standard deviation: Z=X−μσZ = \frac{X - \mu}{\sigma}Z=σX−μ​
* where XXX is the original value, μ\muμ is the mean, and σ\sigmaσ is the standard deviation.
  + In Python, you can use StandardScaler from sklearn.preprocessing to standardize the dataset.

### **2. Compute the Covariance Matrix to Identify Correlations**

The covariance matrix helps to identify the relationships between variables. PCA finds directions (principal components) that maximize the variance in the data, and the covariance matrix is key to understanding this variance.

* **How to Compute the Covariance Matrix:**
  + Let ZZZ be the matrix of standardized data.
  + The covariance matrix CCC can be calculated as: C=1n−1ZTZC = \frac{1}{n-1} Z^T ZC=n−11​ZTZ where nnn is the number of observations, ZTZ^TZT is the transpose of ZZZ.
  + In Python, you can use np.cov() on your standardized data matrix.

### **3. Compute the Eigenvectors and Eigenvalues of the Covariance Matrix**

Eigenvectors and eigenvalues of the covariance matrix help to identify the principal components. Eigenvalues indicate the amount of variance explained by each eigenvector.

* **How to Compute Eigenvectors and Eigenvalues:**
  + Using the covariance matrix CCC, compute the eigenvalues and eigenvectors. This can be done with np.linalg.eig() in Python.
  + Each eigenvalue corresponds to a principal component's variance, and each eigenvector defines the direction of a principal component.

### **4. Create a Feature Vector to Decide Which Principal Components to Keep**

The feature vector consists of the eigenvectors selected based on their eigenvalues. Typically, you keep components with the highest eigenvalues, as they capture the most variance in the data.

* **How to Create a Feature Vector:**
  + Rank the eigenvalues from highest to lowest. Choose the top kkk eigenvalues that capture the most variance (usually until you capture ~95% of the variance).
  + Select the corresponding eigenvectors for these eigenvalues.
  + In Python, you can use PCA from sklearn.decomposition to automatically select components based on variance thresholds.

### **5. Recast the Data Along the Principal Components Axes**

Now, you can project the original data onto the new feature space defined by the principal components.

* **How to Recast the Data:**
  + Multiply the original standardized data by the selected eigenvectors (the feature vector matrix) to get the recast data: Y=Z⋅WY = Z \cdot WY=Z⋅W where ZZZ is the standardized data and WWW is the matrix of selected eigenvectors.
  + The resulting matrix YYY represents the data in terms of principal components.
  + In Python, PCA from sklearn.decomposition can be used to transform data directly onto the principal component axes.

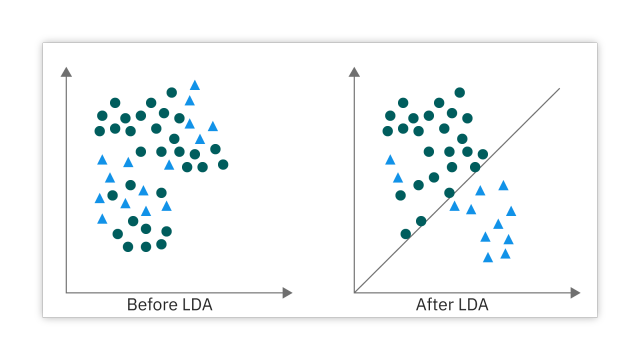
By following these steps, you can effectively reduce the dimensionality of your data while retaining the most important information in terms of variance.

What is LDA?

Linear discriminant analysis (LDA) is an approach used in supervised machine learning to solve multi-class classification problems. LDA separates multiple classes with multiple features through data dimensionality reduction. This technique is important in data science as it helps optimize machine learning models.

LDA algorithms make predictions by using Bayes to calculate the probability of whether an input data set will belong to a particular output.

LDA works by identifying a linear combination of features that separates or characterizes two or more classes of objects or events. LDA does this by projecting data with two or more dimensions into one dimension so that it can be more easily classified. The technique is, therefore, sometimes referred to as dimensionality reduction

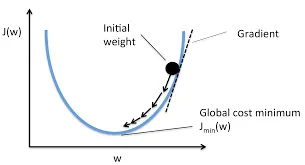


**COST FUCNTION:p-=**

the cost function is a function that measures the error between the predicted and actual values in machine learning. A convex cost function has a single minimum point, while a non-convex cost function has multiple local minima. Optimization algorithms, such as gradient descent, attempt to find the optimal set of model parameters that minimize the cost function. The global minimum represents the best possible fit of the model to the data, but optimization algorithms can get stuck in local minima, especially for non-convex cost functions. To overcome this problem, various techniques can be employed, such as using different optimization algorithms or modifying the cost function to make it more convex.

**Gradient Descent:**

It is an optimization algorithm for finding out the local minima of a differentiable function, it helps us to find the best parameters(Weights and Biases) for the algorithm we are using for our model.



## **Batch Gradient Descent**

It is a technique where all the data is taken into consideration while making a single update to the parameters of our model.

## **Stochastic Gradient Descent**

It is a technique where one row or one input is taken at a time to make a single update to the parameters of our model.

## **Mini Batch Gradient Descent**

It is a technique where a batch is taken in consideration at a time to make a single update to the parameters of our model, we can set the batch size according to us , we set the batch size and (no. of batches = Total Rows in data/batch size), it is a mid way between Batch gradient descent and Stochastic gradient descent.

## **Bootstrapping Statistics Defined**

Bootstrapping statistics is a form of hypothesis testing that involves resampling a single data set to create a multitude of simulated samples. Those samples are used to calculate standard errors, confidence intervals and for hypothesis testing. This approach allows you to generate a more accurate sample from a smaller data set than the traditional method.

### **What is the purpose of bootstrapping statistics?**

The purpose of bootstrapping statistics is to give teams an easy, cost-effective way to test hypotheses and make accurate calculations, even when they have limited data to start.

### **What is the difference between bootstrapping and traditional statistical methods?**

Traditional statistics only takes one sample from a population and uses that sample to draw estimates for the larger population. However, bootstrapping takes one sample and resamples it to generate many samples, making calculations about the larger population based on the sampling distribution of these simulated samples.

### **What is a good sample size for bootstrapping?**

It’s best to aim for resampling a sample to produce at least 1,000 simulated samples.

ML Algorithm Implementation:

Linear Regression:

he equation of a linear regression line is:

y=β0+β1x1+β2x2+⋯+βnxn+ϵy = \beta\_0 + \beta\_1x\_1 + \beta\_2x\_2 + \dots + \beta\_nx\_n + \epsilony=β0​+β1​x1​+β2​x2​+⋯+βn​xn​+ϵ

where:

* yyy is the dependent variable.
* β0\beta\_0β0​ is the y-intercept.
* β1,β2,…,βn\beta\_1, \beta\_2, \dots, \beta\_nβ1​,β2​,…,βn​ are the coefficients.
* x1,x2,…,xnx\_1, x\_2, \dots, x\_nx1​,x2​,…,xn​ are the independent variables.
* ϵ\epsilonϵ is the error term, which accounts for the variability in yyy that cannot be explained by the linear model.

### **Assumptions of Linear Regression**

Linear regression makes several key assumptions:

1. **Linear relationship**: There must be a linear relationship between the dependent and independent variables.
2. **Multivariate normality**: The residuals (errors) of the model should be normally distributed.
3. **No or little multicollinearity**: Independent variables should not be too highly correlated.
4. **Homoscedasticity**: The variance of error terms should be similar across the values of the independent variables.
5. **Independence**: Observations should be independent of each other.

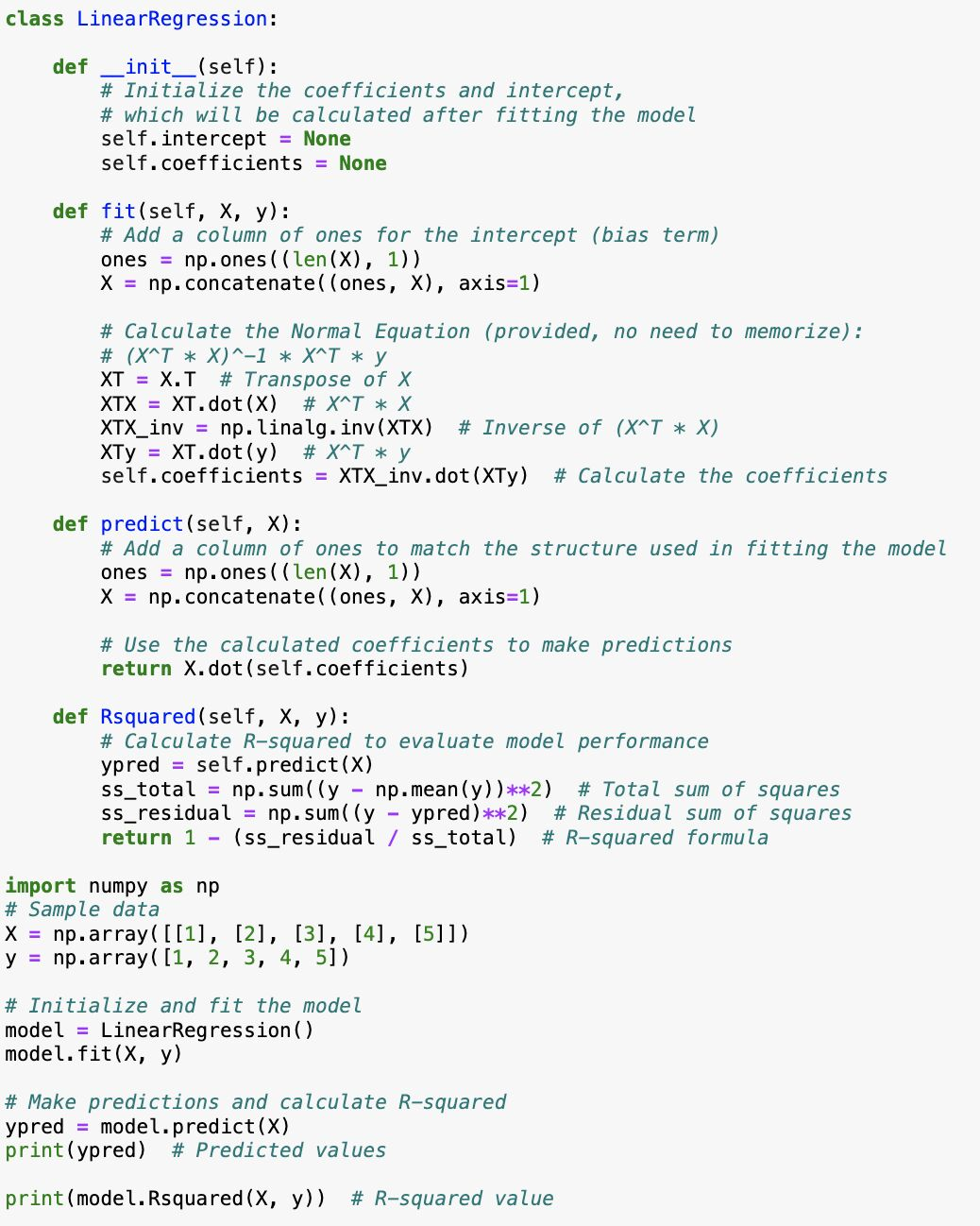
Class Linear Regression:

Def init(self):

Def fit(X,y):

Def predict(X):

Def Rsquared(X,y):



Linear regression is a supervised algorithm[ℹ] that learns to model a dependent variable, y y, as a function of some independent variables (aka "features"),

xi

X by finding a line (or surface) that best "fits" the data.



https://mlu-explain.github.io/linear-regression/

### **What is Support Vector Regression (SVR)?**

Support Vector Regression (SVR) applies the principles of Support Vector Machines (SVM) to solve regression problems. SVR is a type of regression algorithm that aims to fit the best line within a threshold value, where the line can be as far as possible from the nearest points of all the data points.

### **Key Concepts of SVR**

* **Hyperplane**: In SVR, the hyperplane becomes a line that tries to fit the largest possible number of points.
* **Boundary lines**: Also known as margin lines, these are set a distance ϵ\epsilonϵ (epsilon) away from the hyperplane. They create a margin that is tolerated in the fit.
* **Support Vectors**: Points that are on the boundary lines or outside the boundary lines. These points help define and adjust the position of the line.

### **Objective of SVR**

The main objective is to minimize error, fitting the error within a certain threshold. The model produced by SVR depends only on a subset of the training data, because the cost function ignores any training data close to the model prediction (within a threshold ϵ\epsilonϵ).

### **Mathematical Formulation**

The SVR performs linear regression in a higher (possibly infinite) dimensional space. We can think of SVR as if each data point in the training set is plotted in n-dimensional space (where n is number of features), with the value of each feature being the value of a particular coordinate. Then, SVR performs a linear regression in this n-dimensional space.

A linear SVR model can be represented as follows:

y=w⋅x+by = \mathbf{w} \cdot \mathbf{x} + by=w⋅x+b

The goal of SVR is to find the function f(x)f(x)f(x) that has at most ϵ\epsilonϵ deviation from the actually obtained targets yyy for all the training data, and at the same time is as flat as possible.

#### **Constraints:**

∣yi−(w⋅xi+b)∣≤ϵ|y\_i - (\mathbf{w} \cdot \mathbf{x}\_i + b)| \leq \epsilon∣yi​−(w⋅xi​+b)∣≤ϵ

Additionally, for non-linear SVR, the kernel trick is used:

* **Kernel**: Transforms the training set data so that a non-linear separator can be used to transform the data into a higher dimension where it becomes separable.

### **Implementation in Python using Scikit-Learn**

Here is how to implement SVR using the scikit-learn library in Python:

python

Copy code

import numpy as np

from sklearn.svm import SVR

from sklearn.model\_selection import train\_test\_split

from sklearn.metrics import mean\_squared\_error, r2\_score

import matplotlib.pyplot as plt

# Generating random data:

X = np.sort(5 \* np.random.rand(40, 1), axis=0)

y = np.sin(X).ravel()

# Add noise to targets

y[::5] += 3 \* (0.5 - np.random.rand(8))

# Split the data into training/testing sets

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

# Create SVR model

svr\_rbf = SVR(kernel='rbf', C=100, gamma=0.1, epsilon=.1)

# Train the model on training data

svr\_rbf.fit(X\_train, y\_train)

# Make predictions using the testing set

y\_pred = svr\_rbf.predict(X\_test)

# The coefficients

print('Mean squared error: %.2f' % mean\_squared\_error(y\_test, y\_pred))

print('Coefficient of determination: %.2f' % r2\_score(y\_test, y\_pred))

# Plot outputs

plt.scatter(X\_test, y\_test, color='darkorange', label='data')

plt.plot(X\_test, y\_pred, color='navy', lw=2, label='RBF model')

plt.xlabel('Data')

plt.ylabel('Target')

plt.title('Support Vector Regression')

plt.legend()

plt.show()

### **Applications of SVR**

* **Economics**: Predicting economic events, such as GDP growth, inflation rates, stock prices, and more.
* **Climate Science**: Forecasting temperatures, rainfall levels, and other climate variables.
* **Energy Production**: Predicting electricity demand, solar panel output, and other energy-related queries.
* **Finance**: Used in the prediction of market trends and the behavior of financial indices.
* **Healthcare**: Estimating future medical conditions or the expected length of hospital stays.

SVR is especially notable for its ability to handle non-linear data through the use of kernel methods and its robustness against overfitting, particularly in high-dimensional space. This makes SVR an effective tool for many predictive modeling tasks where linear models fall short.

The goal of the SVM algorithm is to find a hyperplane in an N-dimensional space (N — the number of features) that distinctly classifies the data points. The SVM finds the hyperplane which has the maximum margin, meaning the maximum distance between data points of both classes. Maximizing the margin distance provides some reinforcement so that future data points can be classified with more confidence

### **What is a Decision Tree?**

A Decision Tree is a flowchart-like structure in which each internal node represents a "test" on an attribute (e.g., whether a coin flip comes up heads or tails), each branch represents the outcome of the test, and each leaf node represents a class label (decision taken after computing all attributes). The paths from root to leaf represent classification rules.

### **Key Concepts of Decision Trees**

* **Root Node**: This represents the entire dataset, which further gets divided into two or more homogeneous sets.
* **Splitting**: It is a process of dividing a node into two or more sub-nodes based on if-then conditions.
* **Decision Node**: When a sub-node splits into further sub-nodes, then it is called the decision node.
* **Leaf/Terminal Node**: Nodes that do not split further, which means that these nodes represent the decision.
* **Pruning**: Removing the sub-nodes of a decision node can be called pruning. You can say it is the opposite process of splitting.
* **Branch/Sub-Tree**: A subsection of the entire tree is called branch or sub-tree.
* **Parent and Child Node**: A node, which is divided into sub-nodes, is called the parent node of sub-nodes, whereas sub-nodes are the children of the parent node.

### **How Does a Decision Tree Work?**

The decision of making strategic splits heavily affects a tree’s accuracy. The decision criteria are different for classification and regression trees. Decision trees use multiple algorithms to decide to split a node into two or more sub-nodes. The creation of sub-nodes increases the homogeneity of resultant sub-nodes. In other words, we can say that purity of the node increases with respect to the target variable.

#### **Common algorithms used in the construction of decision trees include:**

1. **ID3 (Iterative Dichotomiser 3)**: Uses Entropy function and Information gain as metrics.
2. **C4.5 (Successor of ID3)**: Uses the concept of Gain Ratio.
3. **CART (Classification and Regression Trees)**: Uses Gini Index(Classification) as a metric for creating the splits.

### **Gini Index and Entropy**

* **Gini Index**: 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.

Gini(D)=1−∑i=1npi2\text{Gini}(D) = 1 - \sum\_{i=1}^n p\_i^2Gini(D)=1−∑i=1n​pi2​ Where pip\_ipi​ is the probability of an object being classified to a particular class.

* **Entropy**: A measure of the randomness in the information being processed. The higher the entropy, the harder it is to draw any conclusions from that information.

Entropy(S)=−∑i=1npilog⁡2pi\text{Entropy}(S) = - \sum\_{i=1}^n p\_i \log\_2 p\_iEntropy(S)=−∑i=1n​pi​log2​pi​

* **Information Gain**: The measure of the difference in entropy from before to after the set SSS is split on an attribute AAA.

Gain(S,A)=Entropy(S)−∑v∈Values(A)∣Sv∣∣S∣Entropy(Sv)\text{Gain}(S, A) = \text{Entropy}(S) - \sum\_{v \in Values(A)} \frac{|S\_v|}{|S|} \text{Entropy}(S\_v)Gain(S,A)=Entropy(S)−∑v∈Values(A)​∣S∣∣Sv​∣​Entropy(Sv​)

### **Implementation in Python**

Using Python's scikit-learn library, here is how you can implement a Decision Tree Classifier:

python

Copy code

from sklearn.datasets import load\_iris

from sklearn.tree import DecisionTreeClassifier

from sklearn.model\_selection import train\_test\_split

from sklearn.metrics import accuracy\_score

from sklearn import tree

import matplotlib.pyplot as plt

# Load data

iris = load\_iris()

X = iris.data

y = iris.target

# Split data

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

# Initialize and train classifier

clf = DecisionTreeClassifier()

clf.fit(X\_train, y\_train)

# Predict

y\_pred = clf.predict(X\_test)

# Evaluate classifier

print(f"Accuracy: {accuracy\_score(y\_test, y\_pred)}")

# Plot tree

plt.figure(figsize=(12,12))

tree.plot\_tree(clf, filled=True, feature\_names=iris.feature\_names, class\_names=iris.target\_names, rounded=True)

plt.show()

### **Applications of Decision Trees**

* **Business Management**: Decision trees are helpful in making strategic decisions about business operations and policy planning.
* **Healthcare**: Used for diagnosis processes, predicting illness progression, and customizing treatment plans.
* **Finance**: Credit scoring, claim prediction, and other financial analysis.
* **Manufacturing**: Quality control and predictive maintenance.

Decision Trees are popular due to their ease of interpretation, handling of both numerical and categorical data, and ability to handle multi-output problems. However, they are prone to overfitting, especially with complex datasets and without proper pruning. To mitigate this, ensemble methods like Random Forests or boosting algorithms (such as XGBoost) are often used, which involve building multiple trees to improve the model’s robustness and accuracy.

### **What is Random Forest?**

Random Forest is an ensemble learning method used for both classification and regression that operates by constructing a multitude of decision trees at training time. For classification tasks, the output of the random forest is the class selected by most trees. For regression tasks, it predicts the average or mean of the outputs of individual trees.

### **Key Concepts of Random Forest**

1. **Ensemble Learning**: Random Forest combines multiple decision trees to improve the overall result. The main principle behind ensemble learning is that a group of weak learners come together to form a strong learner.
2. **Bagging (Bootstrap Aggregating)**: Random Forest uses the bagging technique. It involves training each decision tree on a different data sample where sampling is done with replacement (bootstrap sample).
3. **Feature Randomness**: When splitting a node during the construction of the tree, the split that is chosen is no longer the best split among all features. Instead, the split that is picked is the best among a random subset of features. This results in a diversity that generally results in a better model.

### **How Does a Random Forest Work?**

* **Creation of Subsets**: Each decision tree is built on a different sample of the data. This sampling is done with replacement, known as bootstrapping.
* **Tree Building**: Each tree grows during the learning process without pruning (cutting back of branches). It fully grows to the largest extent possible on the bootstrap data.
* **Prediction**:
  + For a classification problem, each tree in the forest votes for a class, and the class receiving the most votes by a simple majority becomes the model’s prediction.
  + For a regression problem, the forecasted outputs of all trees are averaged to provide a single prediction.

### **Mathematical Model**

Given a training set X=x1,x2,...,xnX = x\_1, x\_2, ..., x\_nX=x1​,x2​,...,xn​ with responses Y=y1,y2,...,ynY = y\_1, y\_2, ..., y\_nY=y1​,y2​,...,yn​, bagging repeatedly (B times) selects a random sample with replacement of the training set and fits trees to these samples:

For b=1,2,...,Bb = 1, 2, ..., Bb=1,2,...,B:

* Sample, with replacement, nnn training examples from X,YX, YX,Y; call these Xb,YbX\_b, Y\_bXb​,Yb​.
* Train a decision tree fbf\_bfb​ on Xb,YbX\_b, Y\_bXb​,Yb​.

After training, predictions for unseen samples x′x'x′ can be made by averaging the predictions from all the individual regression trees on x′x'x′ or by taking the majority vote in the case of classification trees.

### **Implementation in Python**

Here’s how you can implement a Random Forest using Python's scikit-learn library:

python

Copy code

from sklearn.datasets import load\_iris

from sklearn.ensemble import RandomForestClassifier

from sklearn.model\_selection import train\_test\_split

from sklearn.metrics import accuracy\_score

# Load dataset

iris = load\_iris()

X = iris.data

y = iris.target

# Split dataset into training and testing set

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.3, random\_state=42)

# Initialize Random Forest Classifier

clf = RandomForestClassifier(n\_estimators=100)

# Fit Random Forest on data

clf.fit(X\_train, y\_train)

# Predict on test data

y\_pred = clf.predict(X\_test)

# Evaluate model

print(f"Accuracy: {accuracy\_score(y\_test, y\_pred)}")

### **Applications of Random Forest**

* **Banking**: Used for credit scoring and predicting loan defaults.
* **Medicine**: Predicting diseases and diagnosis, understanding disease traits and correlations.
* **Stock Market**: Stock market analysis, including predicting stock behavior and trends.
* **E-commerce**: Recommendation systems that suggest products to customers based on their historical usage.
* **Fraud Detection**: Identifying potential fraudulent cases in banking, insurance, and finance.

Random Forest is a powerful method due to its robustness, ease of use, and the ability to run in parallel. It handles overfitting well, provides a good indicator of the feature importance, and is effective on large databases. However, its complexity can make the model difficult to interpret compared to a single decision tree, and

What is Gradient Boosting?

Gradient Boosting is a powerful machine learning technique that builds on decision trees and uses a gradient descent algorithm to minimize errors in sequential models. It is a type of ensemble learning method where new models are added to correct the errors made by existing models. The models are added sequentially until no further improvements can be made, and it is commonly used for both regression and classification problems.

Key Concepts of Gradient Boosting

Boosting: This is the principle behind building an ensemble of models where each new model incrementally decreases the error of the whole system. Unlike bagging in random forests, boosting changes the weight of an observation based on the last classification.

Weak Learner: Gradient boosting involves using high bias/low variance base learners. Most commonly, these are decision trees with a small depth, known as "stumps".

Loss Function: An important aspect of gradient boosting is the use of a loss function that needs to be optimized, which can vary depending on the type of problem (binary classification, multi-class classification, or regression).

Gradient Descent: The algorithm uses a numerical optimization method to minimize the loss function, specifically adapting the gradient descent algorithm to optimize the predictions of decision trees on all fronts.

How Gradient Boosting Works

Model Initialization: Starts with a single model, usually a decision tree, which makes a set of predictions on the dataset. These predictions are usually quite naive — often just a simple average of the target variable.

Compute Loss: Calculate the loss by comparing the predictions with the actual target values using a loss function appropriate to the type of problem being solved.

Build a New Model: A new model is built focusing on the errors committed by the previous model. This is done by calculating the negative gradient of the loss function used for training. This new model then makes predictions on the residual errors made by the previous model.

Combine Models: Combine the predictions of the new model with the predictions of the existing ensemble in a manner that minimizes the overall prediction error. Typically, a fraction of the output of the new model, controlled by a parameter called the learning rate, is added to the predictions of the existing ensemble.

Iterate: Repeat steps 2-4 until a pre-specified number of models have been added or no further improvements can be made.

Output Final Model: The final model is an ensemble of weaker prediction models that together deliver a powerful composite model.

Implementation in Python

You can implement gradient boosting using scikit-learn with the GradientBoostingClassifier or GradientBoostingRegressor classes. Here's a simple example using the classifier:

python

Copy code

from sklearn.datasets import load\_iris

from sklearn.ensemble import GradientBoostingClassifier

from sklearn.model\_selection import train\_test\_split

from sklearn.metrics import accuracy\_score

# Load dataset

iris = load\_iris()

X = iris.data

y = iris.target

# Split dataset into training and testing set

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

# Initialize Gradient Boosting Classifier

gb\_clf = GradientBoostingClassifier(n\_estimators=100, learning\_rate=0.1, max\_depth=3, random\_state=42)

# Fit model

gb\_clf.fit(X\_train, y\_train)

# Predict

y\_pred = gb\_clf.predict(X\_test)

# Evaluate model

print(f"Accuracy: {accuracy\_score(y\_test, y\_pred):.2f}")

Applications of Gradient Boosting

Financial Modeling: Used for credit scoring and risk assessment.

Biological Sciences: Used for modeling the effects of mutations in DNA or simulating ecological processes.

Internet Search Engines: Ranking pages based on a variety of attributes.

Anomaly Detection: Identifying unusual data points in financial transactions, manufacturing, or network traffic.

Gradient Boosting is favored due to its effectiveness in handling different types of data and variable relationships, along with its high performance on many problems. However, it can be computationally expensive and prone to overfitting if not tuned properly. The learning rate and the number of estimators are crucial parameters that need careful adjustment to balance model complexity and training resource consumption.

### **What is XGBoost?**

XGBoost, which stands for eXtreme Gradient Boosting, is an advanced implementation of gradient boosting that is much more efficient, flexible, and powerful. It has become a leading machine learning algorithm in Kaggle competitions and is widely used in industry for structured data.

### **Key Concepts of XGBoost**

* **Gradient Boosting**: XGBoost is based on the principle of gradient boosting, an ensemble technique where new models are added to correct the errors made by existing models. Models are added sequentially until no further improvements can be made.
* **Decision Trees as Base Learners**: It uses decision tree algorithms as the base learner, which are binary trees that split the training data and assign a fixed value at each leaf.
* **Regularization**: XGBoost incorporates both L1 and L2 regularization, which helps to reduce overfitting and improve model performance.
* **Handling Missing Values**: XGBoost has an in-built routine to handle missing values. When XGBoost encounters a missing value at a node, it learns the direction to take for missing values for optimal results.
* **Tree Pruning**: Unlike GBM (Gradient Boosting Machine), which stops splitting a node when it encounters a negative loss, XGBoost splits up to the max\_depth specified and then starts pruning the tree backwards and removes splits beyond which there is no positive gain.
* **Learning Rate**: Also known as "eta," it makes the model more robust by shrinking the weights on each step.
* **Built-in Cross-Validation**: XGBoost allows user to run a cross-validation at each iteration of the boosting process.

### **Mathematical Model**

XGBoost involves many complex mathematical functions, primarily optimizing the following objective function:

Obj(Θ)=∑i=1nl(yi,y^i(t))+∑k=1tΩ(fk)\text{Obj}(\Theta) = \sum\_{i=1}^n l(y\_i, \hat{y}\_i^{(t)}) + \sum\_{k=1}^t \Omega(f\_k)Obj(Θ)=∑i=1n​l(yi​,y^​i(t)​)+∑k=1t​Ω(fk​)

Where:

* y^i(t)\hat{y}\_i^{(t)}y^​i(t)​ is the prediction at the ttt-th iteration.
* lll is a differentiable convex loss function that measures the difference between the predicted y^i\hat{y}\_iy^​i​ and the actual yiy\_iyi​.
* Ω\OmegaΩ represents the regularization term which is used to penalize the complexity of the model (typically the number of leaves in the tree, and the sum of the scores of each model's leaf nodes).

### **Implementation in Python**

XGBoost can be implemented using its Python library. Here's a basic example using the Iris dataset:

python

Copy code

import xgboost as xgb

from sklearn.datasets import load\_iris

from sklearn.model\_selection import train\_test\_split

from sklearn.metrics import accuracy\_score

# Load data

iris = load\_iris()

X = iris.data

y = iris.target

# Split dataset into training and testing set

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

# Specify parameters via map

params = {

'objective': 'multi:softprob', # Multiclass classification using the softmax objective

'num\_class': 3, # Number of possible output classes

'learning\_rate': 0.1, # Learning rate

'max\_depth': 3, # Depth of the tree

'n\_estimators': 100 # Number of trees

}

# Training

model = xgb.XGBClassifier(\*\*params)

model.fit(X\_train, y\_train)

# Predicting

y\_pred = model.predict(X\_test)

# Evaluate

accuracy = accuracy\_score(y\_test, y\_pred)

print(f"Accuracy: {accuracy:.2f}")

### **Applications of XGBoost**

* **Finance**: Credit scoring, risk management, and default prediction.
* **Healthcare**: Predicting medical events such as patient readmission and critical illnesses.
* **Retail and E-commerce**: Customer behavior prediction, sales forecasting, and inventory management.
* **Telecommunications**: Churn prediction, customer segmentation, and fraud detection.

XGBoost's ability to handle various types of data, its robustness to different distributions and relationships in data, along with its high performance on many problems, makes it a preferred choice for many predictive modeling tasks.

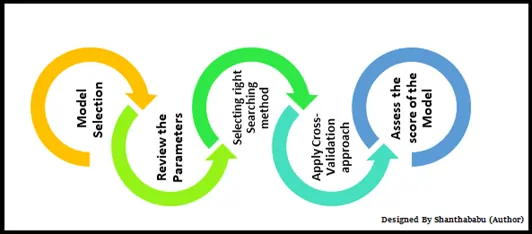
## What is Hyperparameter Tuning?

Hyperparameter tuning is the process of selecting the optimal set of hyperparameters for a machine learning model. It is an important step in the model development process, as the choice of hyperparameters can have a significant impact on the model's performance.

## Hyperparameter Optimization Techniques

In the ML world, there are many Hyperparameter optimization techniques are available.

* Manual Search
* Random Search
* Grid Search
* Halving
  + Grid Search
  + Randomized Search
* Automated Hyperparameter tuning
  + Bayesian Optimization
  + Genetic Algorithms
* Artificial Neural Networks Tuning
* HyperOpt-Sklearn
* Bayes Search



In machine learning (ML), convergence is when a model's predictions stop improving and the error rate becomes constant. This means that the model has reached a stable state where it can make accurate predictions and additional training will not improve it.

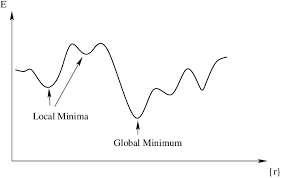
If a model is not converging, it could be due to a number of reasons, including:

* **Data quality**: The data may be dirty, poorly formatted, or not relevant to the problem.
* **Model complexity**: The model may be too complex or too simple for the data.
* **Model parameters**: The model parameters may be inappropriate.

To troubleshoot a model that isn't converging, you can try:

* Checking the data to make sure it's clean and properly formatted
* Normalizing the data to scale individual samples to have unit norm
* Checking the model's complexity

There are also convergence algorithms that can be used to improve the convergence of a model. For example, Root Mean Square Propagation (RMSprop) adjusts the learning rate based on the average of the squared gradients



While training a logistic regression model, you encounter convergence issues where the algorithm fails to reach an optimal solution within the given iterations. How do you solve this?

Common reasons for convergence problems are -

𝗟𝗮𝗰𝗸 𝗼𝗳 𝗙𝗲𝗮𝘁𝘂𝗿𝗲 𝗦𝗰𝗮𝗹𝗶𝗻𝗴

Logistic regression uses gradient-based optimization methods like Gradient Descent or Newton-Raphson (in solvers like liblinear or lbfgs). These algorithms are sensitive to feature scales, and if the features are on vastly different scales (e.g., income in thousands and age in years), the optimization process may struggle to converge.

Solution - Standardization or normalization

𝗟𝗲𝗮𝗿𝗻𝗶𝗻𝗴 𝗥𝗮𝘁𝗲

The learning rate controls how much the model updates its weights at each iteration during optimization. If the learning rate is too high, the algorithm might overshoot the optimal solution, leading to instability and failure to converge. If the learning rate is too low, the algorithm may take too long to converge or stop making significant progress.

Solution - Adjust the learning rate

𝗥𝗲𝗴𝘂𝗹𝗮𝗿𝗶𝘇𝗮𝘁𝗶𝗼𝗻

Logistic regression models can include L1 or L2 regularization to prevent overfitting. However, if the regularization parameter (e.g., CCC in scikit-learn or lambda) is too high, the model penalizes large coefficients too aggressively, which may cause convergence issues. In extreme cases, this forces the model coefficients to approach zero, and the optimization may not converge.

Solution - Tune the regularization strength, use the appropriate solver

𝗖𝗼𝗹𝗹𝗶𝗻𝗲𝗮𝗿𝗶𝘁𝘆 (𝗠𝘂𝗹𝘁𝗶𝗰𝗼𝗹𝗹𝗶𝗻𝗲𝗮𝗿𝗶𝘁𝘆)

Multicollinearity occurs when two or more features in the dataset are highly correlated. This makes it difficult for the optimization algorithm to determine which feature contributes to the change in the target variable, leading to unstable coefficients and poor convergence.

Solution - Remove or combine correlated features

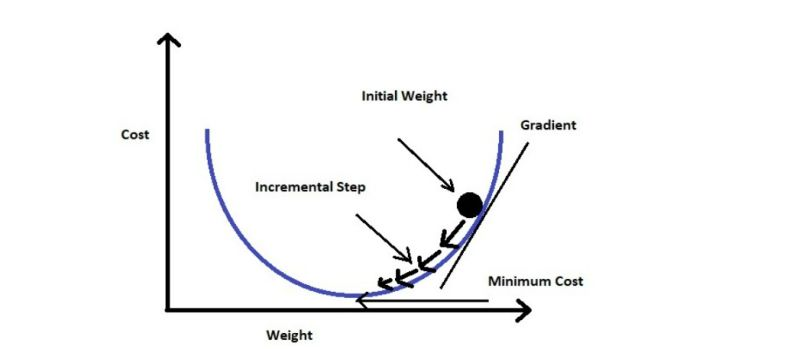
𝗗𝗮𝘁𝗮 𝗜𝗺𝗯𝗮𝗹𝗮𝗻𝗰𝗲

If the dataset is imbalanced (i.e., one class significantly outnumbers the other), the logistic regression model may have difficulty finding an optimal solution that separates the classes, causing convergence problems.

Solution - resampling, class weights

𝗢𝘂𝘁𝗹𝗶𝗲𝗿𝘀

Outliers can have an outsized influence on the logistic regression model, especially if they are present in the feature space or the target variable. These extreme values can distort the optimization process, making convergence difficult.

Solutions - remove/transform outliers

Data Science Interview Question:

If you are working with a dataset that includes categorical variables alongside continuous variables, how would you approach calculating distances for kNN?

Traditional distance metrics like Euclidean won't work. So here a few you can consider instead -

𝗠𝗮𝗵𝗮𝗹𝗮𝗻𝗼𝗯𝗶𝘀 𝗱𝗶𝘀𝘁𝗮𝗻𝗰𝗲 takes into account the correlations of the dataset and scales the continuous variables accordingly. It can be used if the categorical features are encoded as binary variables (0 or 1).

𝗛𝗮𝗺𝗺𝗶𝗻𝗴 𝗱𝗶𝘀𝘁𝗮𝗻𝗰𝗲 can be used for categorical variables, counting the number of positions at which the corresponding elements are different.

𝗚𝗼𝘄𝗲𝗿 𝗱𝗶𝘀𝘁𝗮𝗻𝗰𝗲 is a popular choice for mixed data types. It computes the distance between two instances by calculating the distance for each feature and then combining them. For continuous features, the distance is calculated using the range and the absolute difference between the two values. For 𝗰𝗮𝘁𝗲𝗴𝗼𝗿𝗶𝗰𝗮𝗹 𝗳𝗲𝗮𝘁𝘂𝗿𝗲𝘀, the distance is binary (0 if they are the same, 1 if they are different).

To effectively calculate the overall distance when you have both categorical and continuous variables, you can -

𝗡𝗼𝗿𝗺𝗮𝗹𝗶𝘇𝗲 𝗖𝗼𝗻𝘁𝗶𝗻𝘂𝗼𝘂𝘀 𝗩𝗮𝗿𝗶𝗮𝗯𝗹𝗲𝘀: Scale continuous variables using standardization or min-max normalization to ensure they are on a similar scale.

𝗪𝗲𝗶𝗴𝗵𝘁𝗲𝗱 𝗗𝗶𝘀𝘁𝗮𝗻𝗰𝗲 𝗖𝗮𝗹𝗰𝘂𝗹𝗮𝘁𝗶𝗼𝗻: Combine different distance metrics (like Euclidean for continuous Hamming for categorical) and using weights based on the importance of features to decide the final distance.

**MAXIMUM LIKELIHOOD ESTIMATION**

**A common modeling problem involves how to estimate a joint probability distribution for a dataset.**

**For example, given a sample of observation (*X*) from a domain (*x1, x2, x3, …, xn*), where each observation is drawn independently from the domain with the same probability distribution (so-called independent and identically distributed, i.i.d., or close to it).**

**Density estimation involves selecting a probability distribution function and the parameters of that distribution that best explain the joint probability distribution of the observed data (*X*).**

* **How do you choose the probability distribution function?**
* **How do you choose the parameters for the probability distribution function?**

**This problem is made more challenging as sample (*X*) drawn from the population is small and has noise, meaning that any evaluation of an estimated probability density function and its parameters will have some error.**

**There are many techniques for solving this problem, although two common approaches are:**

* **Maximum a Posteriori (MAP), a Bayesian method.**
* **Maximum Likelihood Estimation (MLE), frequentist method.**

**The main difference is that MLE assumes that all solutions are equally likely beforehand, whereas MAP allows prior information about the form of the solution to be harnessed.**

**In this post, we will take a closer look at the MLE method and its relationship to applied machine learning.**

## **Maximum Likelihood Estimation**

**One solution to probability density estimation is referred to as Maximum Likelihood Estimation, or MLE for short.**

[**Maximum Likelihood Estimation**](https://en.wikipedia.org/wiki/Maximum_likelihood_estimation) **involves treating the problem as an optimization or search problem, where we seek a set of parameters that results in the best fit for the joint probability of the data sample (*X*).**

**First, it involves defining a parameter called *theta* that defines both the choice of the probability density function and the parameters of that distribution. It may be a vector of numerical values whose values change smoothly and map to different probability distributions and their parameters.**

**In Maximum Likelihood Estimation, we wish to maximize the probability of observing the data from the joint probability distribution given a specific probability distribution and its parameters, stated formally as:**

* **P(X | theta)**

**This conditional probability is often stated using the semicolon (;) notation instead of the bar notation (|) because *theta* is not a random variable, but instead an unknown parameter. For example:**

* **P(X ; theta)**

**or**

* **P(x1, x2, x3, …, xn ; theta)**

**This resulting conditional probability is referred to as the likelihood of observing the data given the model parameters and written using the notation *L()* to denote the** [**likelihood function**](https://en.wikipedia.org/wiki/Likelihood_function)**. For example:**

* **L(X ; theta)**

**The objective of Maximum Likelihood Estimation is to find the set of parameters (*theta*) that maximize the likelihood function, e.g. result in the largest likelihood value.**

* **maximize L(X ; theta)**

**We can unpack the conditional probability calculated by the likelihood function.**

**Given that the sample is comprised of n examples, we can frame this as the joint probability of the observed data samples *x1, x2, x3, …, xn* in *X* given the probability distribution parameters (*theta*).**

* **L(x1, x2, x3, …, xn ; theta)**

**The joint probability distribution can be restated as the multiplication of the conditional probability for observing each example given the distribution parameters.**

* **product i to n P(xi ; theta)**

**Multiplying many small probabilities together can be numerically unstable in practice, therefore, it is common to restate this problem as the sum of the log conditional probabilities of observing each example given the model parameters.**

* **sum i to n log(P(xi ; theta))**

**https://machinelearningmastery.com/what-is-maximum-likelihood-estimation-in-machine-learning/**