Build a Lead Scoring Model using Machine Learning in Python

Introduction

A company that specializes in B2C sales (product: Data Science and Data Engineering related courses) is struggling to identify and prioritize high-quality leads efficiently. Currently, sales representatives are responsible for manually qualifying and scoring leads based on limited information, which often results in wasted time and resources on low-quality leads.

This project aims to develop a lead scoring model using machine learning algorithms that can accurately predict the likelihood of a lead converting into a paying customer. This model will enable the sales team to prioritize high-quality leads and improve the efficiency of the sales process.

What is Lead Scoring?

Lead scoring is a methodology used by businesses to rank and prioritize their leads based on their level of interest and potential for conversion into customers. The purpose of lead scoring is to identify and focus on the most promising leads, allowing sales and marketing teams to allocate their time and resources more efficiently.

Lead scoring involves assigning a numerical value or score to each lead based on certain criteria and behaviors. These criteria typically include demographic information, such as job title and company size, as well as behavioral data, such as website visits,

email engagement, and social media interactions. The specific criteria and their corresponding weights may vary depending on the business and its target audience.

Business Impact of Lead Scoring

Improve effectiveness of campaigns: Lead scoring can enhance sales and marketing effectiveness by allowing teams to focus their efforts on leads with the highest potential, resulting in better lead conversion rates, increased revenue, and improved overall efficiency.

Increase product conversions: Lead scoring leads to righ user targeting through right channels/assets which leads to better conversions.

Increase revenue: Increased conversions(as mentioned in point above) will lead to more revenue or buyer engagement. For example, if the company is able to target a user who is more active on Instagram, chances are more that he/she will click on the Ad and add the product to cart. So overall probability of an order increases and hence the revenue.

Cost Efficiency: Lead scoring helps optimize resource allocation by focusing on leads that are more likely to convert. By avoiding wasteful spending on low-quality leads or unproductive marketing efforts, businesses can reduce costs and maximize their return on investment.

Why Data Science?

Companies do digital and offline marketing through various channels such as social media, email, and search advertising. They have a fixed budget which they can spend across these channels to get more sales, acquire new customers or regain old customers.

Often they have vast majority of data on how efficient each of these channels have been for them and how much sales it is driving. If you do a simple vanilla analysis of channels

vs revenue correlation you could get meaningful insights. Now imagine if a model is supplied with such rich and crucial data?

Data science enables these companies to build a model which can help them understand the likelihood of a conversion or in other words a customer buying their products if the latter is targeted through a specific channel. Model understands the historical nuances and builds the internal model parameters so as to tell the team where they should focus on spending their marketing budget!

Possible Solutions

Methods for Lead Scoring

There are various ways to build a model for lead scoring, and the choice depends on factors such as the data available, the complexity of the system, and the computational resources available. Some approaches are:

1. Manual scoring:

This approach involves a subjective evaluation of leads by sales or marketing teams based on predefined criteria. Leads are manually assigned scores or labels based on their fit, interest level, and engagement. Although this method can be simple and cost-effective, it is prone to subjectivity and may lack consistency.

2. Rule-based systems:

Rule-based lead scoring involves the creation of predefined rules and thresholds for assigning scores to leads based on specific criteria and behaviors. These rules can be based on demographic information (e.g., job title, company size) and engagement data

(e.g., email opens, website visits). This method offers more consistency than manual scoring but may lack the flexibility and adaptability of automated approaches.

3. Linear optimizatiom:

In this approach, we define the objective function(e.g. increase conversion, reduce total cost) and the constraints to come up with a linear equation. The linear equation is usually solved using a solver such as Gurobi, Pulp, etc. for the right solution which gives the confidence around the lead the company should pursue.

4. Machine learning:

Machine learning systems combines strength of historical data and statistical techniques to explain the right lead for individual customers of the company. For example, a ML system can tell with a high confidence if a potentail customer will click on the ad or not.

Assumptions

- We assume that **Interest Level** is our target variable which refers to the interest of a user for a lead id
- We assume that whenever the target variable is NA or Not called, the corresponding lead id is not meaningful and hence dropped
- If the model's prediction is very close to 1, it means that the user is very likely to engage with the lead id
- Columns with a lot of null values are not meaningful and imputation also won't be helpful

---

Approach

We are treating this problem as a supervised learning problem. So every data point will have a target variable for the model to learn the dependencies and predict on the unknown.

In real life, this model would tell the business whether a user is likely to engage with the ad or not and that would in turn help the company to pursue the lead.

Given our assumptions about the data, we will build a prediction model based on the historical data. Simplifying, here's the logic of what we'll build:

1. We'll build a model to identify if a customer will be interested in the lead;
2. We'll use various models and compare their performance on interest prediction;
3. We will then choose the most successful model to use in production;

- Exploratory Data Analysis (EDA):
    - Understand the features and their relationships with target variables
    - Check for missing or invalid values and their imputation
- Data Preprocessing:
    - Encode the variables using label encoding
    - Split the dataset into training and testing sets
- Model Building and Testing:
    - Linear Regression
    - Naive Bayes
    - Support Vector Machines
    - Random Forest
    - Light Gradient Boosting
    - Extreme Gradient Boosting
    - Neural Network

Supervised Machine Learning:

In supervised machine learning, the algorithm is trained on an labeled dataset with a predefined target variable. The goal is to identify patterns, relationships, and structures of the data with the target variable, such as logistic regression, decision tree or boosting trees

Exploratory Data Analysis

Data Exploration

Data exploration is a critical step in the data analysis process, where you examine the dataset to gain a preliminary understanding of the data, detect patterns, and identify potential issues that may need further investigation. Data exploration is important because it helps to provide a solid foundation for subsequent data analysis tasks, hypothesis testing and data visualization.

Data exploration is also important because it can help you to identify an appropriate approach for analyzing the data.

Here are the various functions that help us explore and understand the data.

- Shape: Shape is used to identify the dimensions of the dataset. It gives the number of rows and columns present in the dataset. Knowing the dimensions of the dataset is important to understand the amount of data available for analysis and to determine the feasibility of different methods of analysis.
- Head: The head function is used to display the top five rows of the dataset. It helps us to understand the structure and organization of the dataset. This function gives an idea of what data is present in the dataset, what the column headers are, and how the data is organized.
- Tail: The tail function is used to display the bottom five rows of the dataset. It provides the same information as the head function but for the bottom rows. The tail function is particularly useful when dealing with large datasets, as it can be time-consuming to scroll through all the rows.
- Describe: The describe function provides a summary of the numerical columns in the dataset. It includes the count, mean, standard deviation, minimum, and maximum values, as well as the quartiles. It helps to understand the distribution of the data, the presence of any outliers, and potential issues that can affect the model's accuracy.
- Isnull: The isnull function is used to identify missing values in the dataset. It returns a Boolean value for each cell, indicating whether it is null or not. This function is useful to identify the presence of missing data, which can be problematic for regression analysis.
- Dropna: The dropna function is used to remove rows or columns with missing data. It is used to remove any observations or variables with missing data, which can lead to biased results in the regression analysis. The dropna function is used after identifying the missing data with the isnull function.
- Columns: The .columns method is a built-in function that is used to display the column names of a pandas DataFrame or Series. It returns an array-like object

that contains the names of the columns in the order in which they appear in the original DataFrame or Series. It can be used to obtain a quick overview of the variables in a dataset and their names.

Data Processing & Feature engineering

Data Preprocessing and Leakage

Data leakage is a situation where information from the test or prediction data is inadvertently used during the training process of a machine learning model. This can occur when information from the test or prediction data is leaked into the training data, and the model uses this information to improve its performance during the training process.

Data leakage can occur during the preprocessing phase of machine learning when information from the test or prediction data is used to preprocess the training data, inadvertently leaking information from the test or prediction data into the training data.

For example, consider a scenario where the preprocessing step involves imputing missing values in the dataset. If the missing values are imputed using the mean or median values of the entire dataset, including the test and prediction data, then the imputed values in the training data may be influenced by the values in the test and prediction data. This can lead to data leakage, as the model may learn to recognize patterns in the test and prediction data during the training process, leading to overfitting and poor generalization performance.

To avoid data leakage, it's important to perform the data preprocessing steps on the training data only, and then apply the same preprocessing steps to the test and prediction data separately. This ensures that the test and prediction data remain unseen by the model during the training process, and helps to prevent overfitting and improve the accuracy of the model.

In the context of this problem, we will perform data preprocessing steps together for the sake of simplicity, which could potentially lead to data leakage. However, in real-world scenarios, it's important to treat the test and prediction data separately and apply the necessary preprocessing steps separately, based on the characteristics of the data.

Missing Value Detection and Imputation

Real world datasets are never friendly to data scientists. They always pose great challenges to those who are dealing with them due to many different reasons and one of them is “missing values”

Missing values can be imputed with a provided constant value, or using the statistics (mean, median or most frequent) of each column in which the missing values are located

Label Encoding

Transforming Categorical Variables

Transforming variables is an important step in the data preprocessing pipeline of machine learning, as it helps to convert the data into a format that is suitable for analysis and modeling. There are several ways to transform variables, depending on the type and nature of the data.

Categorical variables, for example, are variables that take on discrete values from a finite set of categories, such as colors, gender, or occupation. One common way to transform categorical variables is through one-hot encoding. One-hot encoding involves creating a new binary variable for each category in the original variable, where the value is 1 if the observation belongs to that category and 0 otherwise. This approach is useful when the categories have no natural order or ranking.

Another way to transform categorical variables is through label encoding. Label encoding involves assigning a unique integer value to each category in the variable. This approach is useful when the categories have a natural order or ranking, such as low, medium, and high. Transforming categorical features into numerical labels:

Note: We are NOT using dummies here to minimize the explosion of columns because of the distance methods we are using.

Supervised learning

Supervised learning uses a training set to teach models to yield the desired output. This training dataset includes inputs and correct outputs, which allow the model to learn over time. The algorithm measures its accuracy through the loss function, adjusting until the error has been sufficiently minimized.

Supervised learning can be separated into two types of problems when data mining—classification and regression:

1. Classification uses an algorithm to accurately assign test data into specific categories. It recognizes specific entities within the dataset and attempts to draw some conclusions on how those entities should be labeled or defined. Common classification algorithms are linear classifiers, support vector machines (SVM), decision trees, k-nearest neighbor, and random forest, which are described in more detail below.
2. Regression is used to understand the relationship between dependent and independent variables. It is commonly used to make projections, such as for sales revenue for a given business. Linear regression, logistical regression, and polynomial regression are popular regression algorithms.

Logistic Regression

Logistic regression is a statistical model used for binary classification tasks. Despite its name, it is primarily used for classification rather than regression. The goal of logistic

regression is to estimate the probability that a given instance belongs to a particular class based on its features.

Here's a step-by-step explanation of the logistic regression algorithm:

1. Data Preparation: Logistic regression requires a labeled dataset for training. Each instance in the dataset consists of a set of features and a binary class label.
2. Linear Combination: Logistic regression assumes a linear relationship between the features and the log-odds (also known as the logit) of the probability of the positive class. The log-odds is defined as the logarithm of the odds, where odds is the ratio of the probability of the positive class to the probability of the negative class.
3. Logistic Function (Sigmoid): To convert the linear combination into a valid probability value between 0 and 1, logistic regression uses the logistic function, also known as the sigmoid function. The sigmoid function transforms any real-valued number into a value between 0 and 1.
4. Model Training (Maximum Likelihood Estimation): The logistic regression model is trained by finding the optimal values for the model parameters that maximize the likelihood of the observed data. This is typically done using an optimization algorithm such as gradient descent.
5. Cost Function (Log Loss): The cost function in logistic regression is often defined as the negative log-likelihood, or log loss. The goal is to minimize the log loss by adjusting the model parameters during training.
6. Classification Decision: Once the model parameters are learned, logistic regression uses a threshold (often 0.5) to classify instances. If the predicted probability is above the threshold, the instance is assigned to the positive class; otherwise, it is assigned to the negative class.
7. Model Evaluation: After training the logistic regression model, its performance is evaluated using a separate test dataset. Common evaluation metrics include accuracy, precision, recall, and F1 score.

Logistic regression is a widely used algorithm due to its simplicity and interpretability. It can handle both categorical and continuous features, and it can be extended to handle multiclass classification problems (e.g., using one-vs-rest or softmax regression). However, logistic regression assumes a linear relationship between the features and the log-odds, which may limit its performance on complex, nonlinear datasets. In such

cases, more advanced techniques like decision trees, support vector machines, or neural networks may be more appropriate.

Naive Bayes

Naive Bayes is a simple and popular algorithm for classification tasks. It is based on Bayes' theorem, which calculates the probability of a certain event occurring given prior knowledge or evidence. Naive Bayes assumes independence between features, meaning that the presence or absence of one feature does not affect the presence or absence of another feature.

Here's a step-by-step explanation of the Naive Bayes algorithm:

1. Data Preparation: The algorithm requires a labeled dataset for training. Each instance in the dataset consists of a set of features and a corresponding class label.
2. Feature Independence Assumption: Naive Bayes assumes that all features in the dataset are independent of each other, given the class label. Although this assumption may not hold in real-world scenarios, Naive Bayes can still perform well in practice.
3. Calculating Class Priors: The algorithm begins by calculating the prior probability of each class label. It determines the relative frequency of each class in the training dataset.
4. Calculating Feature Likelihoods: For each feature, the algorithm calculates the likelihood of that feature occurring in each class. It does this by estimating the conditional probability of a feature given a class label.
5. Combining Likelihoods: Using the calculated likelihoods and the prior probabilities, Naive Bayes combines the probabilities using Bayes' theorem to compute the posterior probability of each class given the features of an instance.
6. Classification Decision: Once the posterior probabilities have been calculated for each class, the algorithm selects the class with the highest probability as the predicted class for the given instance.
7. Model Evaluation: After training the Naive Bayes model, its performance is evaluated using a separate test dataset. Common evaluation metrics include accuracy, precision, recall, and F1 score.

Naive Bayes is efficient and performs well even with a small amount of training data. However, it may struggle when faced with features that are not truly independent or when there are missing combinations of feature values in the training data.

There are different variations of Naive Bayes, such as Gaussian Naive Bayes for continuous features, Multinomial Naive Bayes for discrete features, and Bernoulli Naive Bayes for binary features. These variations accommodate different types of data and make appropriate probability assumptions based on the feature distributions.

Overall, Naive Bayes is a straightforward yet effective algorithm for classification tasks, particularly in text classification, spam filtering, and sentiment analysis.

Linear SVM

Linear SVM (Support Vector Machine) is a popular algorithm for binary classification. It aims to find an optimal hyperplane that separates the data into different classes by maximizing the margin between the classes.

Here's a step-by-step explanation of the linear SVM algorithm:

1. Data Preparation: Linear SVM requires a labeled dataset for training. Each instance in the dataset consists of a set of features and a binary class label.
2. Hyperplane Definition: A hyperplane in a feature space is a subspace of one dimension less than the input space. In the case of linear SVM, the hyperplane is defined as a linear combination of the input features.
3. Margin Maximization: The key idea in linear SVM is to find the hyperplane that maximizes the margin between the classes. The margin is the distance between the hyperplane and the nearest data points of each class. SVM aims to find the hyperplane that maximizes this margin, as it provides better generalization and reduces the risk of misclassification.
4. Optimization Objective: To find the optimal hyperplane, SVM formulates an optimization problem. The objective is to minimize the norm of the weight vector (||w||) while satisfying the constraint that all data points are correctly classified, i.e., they lie on the correct side of the hyperplane. This is often referred to as the primal problem.

5. Dual Problem: The primal problem can be transformed into a dual problem that involves maximizing a function subject to constraints. The dual problem is typically solved using optimization techniques, such as the quadratic programming algorithm.
6. Classification Decision: Once the optimal hyperplane is found, the linear SVM classifies new instances based on which side of the hyperplane they fall on. Instances on one side are assigned to one class, while instances on the other side are assigned to the other class.
7. Model Evaluation: After training the linear SVM model, its performance is evaluated using a separate test dataset. Common evaluation metrics include accuracy, precision, recall, and F1 score.

Linear SVM is widely used due to its ability to handle high-dimensional data and its effectiveness in scenarios where the classes are separable by a hyperplane. It is often used in text classification, image classification, and other applications. However, linear SVM may not perform well when the classes are not linearly separable. In such cases, nonlinear SVM kernels, such as polynomial or Gaussian (RBF) kernels, can be used to handle more complex decision boundaries.

Decision Tree

A decision tree is a non-parametric supervised learning algorithm, which is utilized for both classification and regression tasks. It has a hierarchical, tree structure, which consists of a root node, branches, internal nodes and leaf nodes.

Decision tree learning employs a divide and conquer strategy by conducting a greedy search to identify the optimal split points within a tree. This process of splitting is then repeated in a top-down, recursive manner until all, or the majority of records have been classified under specific class labels. Whether or not all data points are classified as homogenous sets is largely dependent on the complexity of the decision tree. Smaller trees are more easily able to attain pure leaf nodes—i.e. data points in a single class.

However, as a tree grows in size, it becomes increasingly difficult to maintain this purity, and it usually results in too little data falling within a given subtree. When this occurs, it is known as data fragmentation, and it can often lead to overfitting. As a result, decision trees have preference for small trees, which is consistent with the principle of parsimony in Occam's Razor; that is, "entities should not be multiplied beyond necessity." Said differently, decision trees should add complexity only if necessary, as the simplest explanation is often the best. To reduce complexity and prevent overfitting, pruning is usually employed; this is a process, which removes branches that split on features with low importance. The model's fit can then be evaluated through the process of cross-validation.

Bagging

Bagging is an ensemble learning technique that aims to decrease the variance of a single estimator by combining the predictions from multiple learners. The basic idea behind bagging is to generate multiple versions of the training dataset through random sampling with replacement, and then train a separate classifier for each sampled dataset. The predictions from these individual classifiers are then combined using averaging or voting to obtain a final prediction.

Algorithm:

Suppose we have a training set D of size n, and we want to train a classifier using bagging. Here are the steps involved:

- Create k different bootstrap samples from D, each of size n.
- Train a classifier on each bootstrap sample.
- When making predictions on a new data point, take the average or majority vote of the predictions from each of the k classifiers.

Mathematical Explanation:

Suppose we have a binary classification problem with classes -1 and 1. Let's also assume that we have a training set D of size n, and we want to train a decision tree classifier using bagging.

Bootstrap Sample: For each of the k classifiers, we create a bootstrap sample of size n by sampling with replacement from D. This means that each bootstrap sample may contain duplicates of some instances and may also miss some instances from the original dataset. Let's denote the i-th bootstrap sample as D_i.

Train a Classifier: We train a decision tree classifier T_i on each bootstrap sample D_i. This gives us k classifiers T_1, T_2, ..., T_k.

Combine Predictions: To make a prediction on a new data point x, we take the majority vote of the predictions from each of the k classifiers.

The idea behind bagging is that the variance of the prediction error decreases as k increases. This is because each classifier has a chance to explore a different part of the feature space due to the random sampling with replacement, and the final prediction is a combination of these diverse classifiers.

Boosting

Boosting is a machine learning algorithm that works by combining several weak models (also known as base learners) into a strong model. The goal of boosting is to reduce the bias and variance of the base learners by iteratively adding new models to the ensemble that focus on correcting the errors made by the previous models. In other words, the boosting algorithm tries to learn from the mistakes of the previous models and improve the overall accuracy of the ensemble.

Boosting works by assigning higher weights to the data points that the previous models misclassified, and lower weights to the ones that were classified correctly. This ensures that the new model focuses more on the difficult data points that the previous models

struggled with, and less on the ones that were already well-classified. As a result, the new model is more specialized and can improve the accuracy of the ensemble.

There are several types of boosting algorithms, including AdaBoost (Adaptive Boosting), Gradient Boosting, and XGBoost (Extreme Gradient Boosting). Each of these algorithms has its own approach to assigning weights to the data points and building the new models, but they all share the fundamental idea of iteratively improving the accuracy of the ensemble by combining weak models into a strong one. Boosting is a powerful algorithm that has been shown to achieve state-of-the-art results in many machine learning tasks, such as image classification, natural language processing, and recommender systems.

Difference between Bagging and Boosting

It's important to remember that boosting is a generic method, not a specific model, in order to comprehend it. Boosting involves specifying a weak model, such as regression or decision trees, and then improving it. In Ensemble Learning, the primary difference between Bagging and Boosting is that in bagging, weak learners are trained in simultaneously, but in boosting, they are trained sequentially. This means that each new model iteration increases the weights of the prior model's misclassified data. This redistribution of weights aids the algorithm in determining which parameters it should focus on in order to increase its performance.

Both the Ensemble techniques are used in a different way as well. Bagging methods, for example, are often used on poor learners who have large variance and low bias such as decision trees because they tend to overfit, whereas boosting methods are employed when there is low variance and high bias. While bagging can help prevent overfitting, boosting methods are more vulnerable to it because of a simple fact they continue to build on weak learners and continue to minimise error. This can lead to overfitting on the training data but specifying a decent number of models to be generated or hyperparameter tuning, regularization can help in this case, if overfitting encountered.

Random Forest

Another way that decision trees can maintain their accuracy is by forming an ensemble via a random forest algorithm; this classifier predicts more accurate results, particularly when the individual trees are uncorrelated with each other.

Random Forest is an ensemble learning algorithm that builds a large number of decision trees and combines them to make a final prediction. It is a type of bagging method, where multiple decision trees are trained on random subsets of the training data and features. The algorithm then averages the predictions of these individual trees to produce a final prediction. Random Forest is particularly useful for handling high-dimensional data and for avoiding overfitting.

Algorithm of Random Forest

The algorithm of Random Forest can be summarized in the following steps:

- Start by randomly selecting a subset of the training data, with replacement. This subset is called the bootstrap sample.
- Next, randomly select a subset of features from the full feature set.
- Build a decision tree using the bootstrap sample and the selected subset of features. At each node of the tree, select the best feature and split the data based on the selected feature.
- Repeat steps 1-3 to build multiple trees.
- Finally, combine the predictions of all trees to make a final prediction. For classification, this is usually done by taking a majority vote of the predicted classes. For regression, this is usually done by taking the average of the predicted values.

Mathematics Behind Random Forest

The mathematics behind Random Forest involves the use of decision trees and the bootstrap sampling technique. Decision trees are constructed using a recursive binary partitioning algorithm that splits the data based on the values of the selected features. At each node, the algorithm chooses the feature and the split point that maximizes the

information gain. Information gain measures the reduction in entropy or impurity of the target variable after the split. The goal is to minimize the impurity of the subsets after each split.

Bootstrap sampling is a statistical technique that involves randomly sampling the data with replacement to create multiple subsets. These subsets are used to train individual decision trees. By using bootstrap samples, the algorithm can generate multiple versions of the same dataset with slightly different distributions. This introduces randomness into the training process, which helps to reduce overfitting.

Difference between Bagging and Random Forest

Bagging and Random Forest are both ensemble learning algorithms that involve training multiple models on random subsets of the data. The main difference between the two is the way the individual models are trained.

Bagging involves training multiple models using the bootstrap sampling technique, but each model uses the same set of features. This can lead to correlated predictions, which reduces the variance but not necessarily the bias of the model.

Random Forest, on the other hand, involves training multiple models using the bootstrap sampling technique, but each model uses a randomly selected subset of features. This introduces additional randomness into the model and helps to reduce the correlation between individual predictions. Random Forest can achieve better performance than Bagging, especially when dealing with high-dimensional data or noisy features. In simpler terms it uses subsets of observations as well as features.

Gradient Boosting Trees

Gradient boosting is a machine learning technique used in regression and classification tasks, among others. It gives a prediction model in the form of an ensemble of weak prediction models, which are typically decision trees.

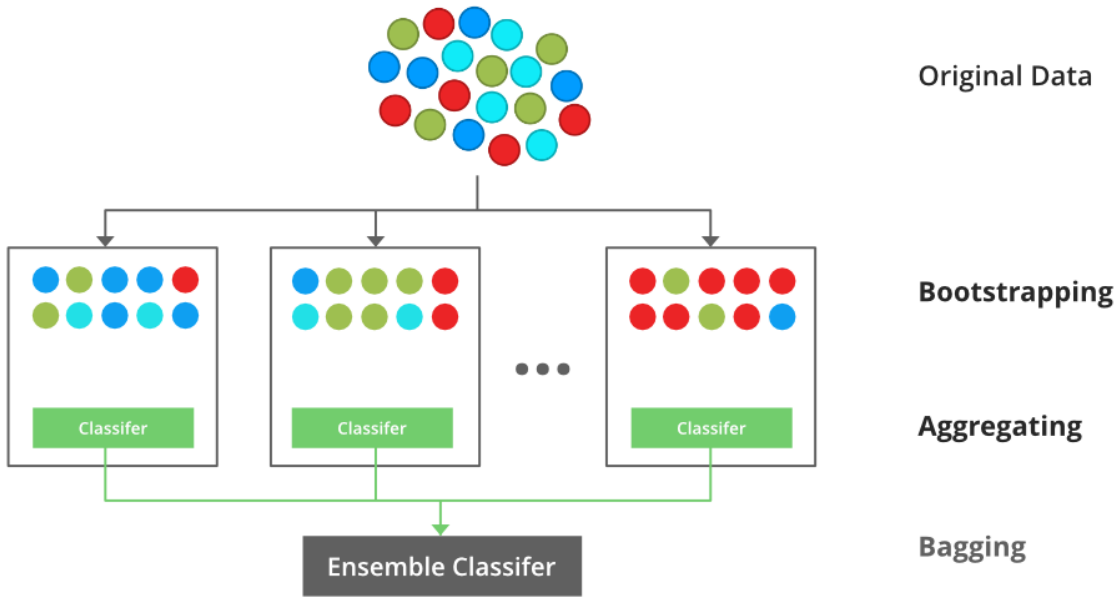
When a decision tree is the weak learner, the resulting algorithm is called gradient-boosted trees; it usually outperforms random forest. A gradient-boosted trees model is built in a stage-wise fashion as in other boosting methods, but it generalizes the other methods by allowing optimization of an arbitrary differentiable loss function.

Few examples of gradient boosting trees are Xgboost, LightGBM, etc

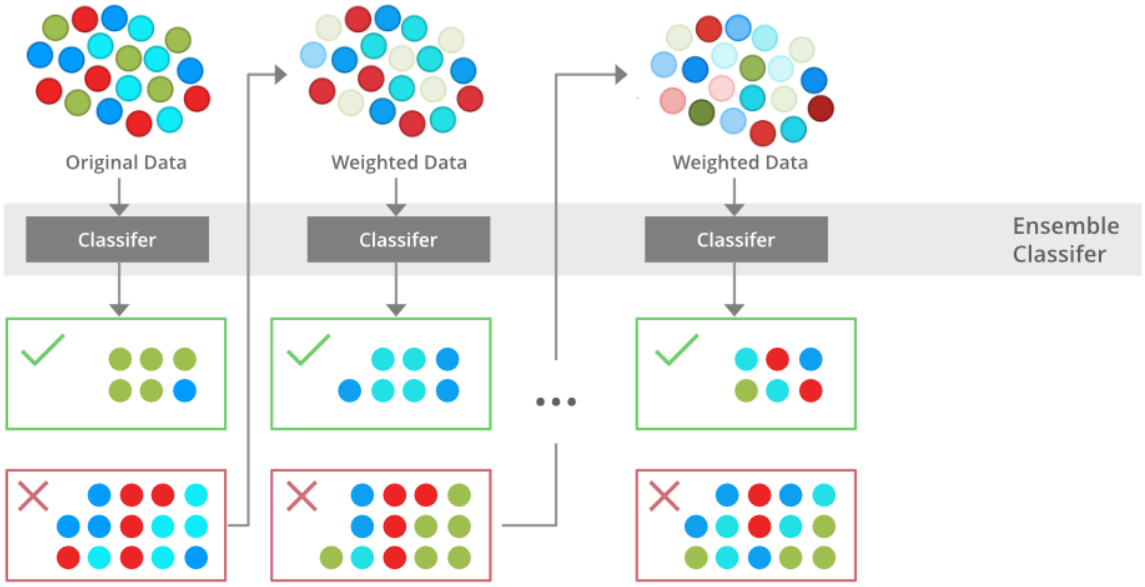
---

Bagging vs Boosting

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



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



Credit: geeksforgeeks

Neural Network Architecture

Neural networks, also known as artificial neural networks or simply neural nets, are computational models inspired by the structure and functionality of the human brain. They are widely used in machine learning and deep learning for solving complex problems across various domains.

Basic Structure

A neural network consists of interconnected layers of artificial neurons, also known as nodes or units. The layers are organized into an input layer, one or more hidden layers, and an output layer. The input layer receives the input data, the hidden layers process the information, and the output layer produces the final predictions or outputs.

Neurons and Connections

Each neuron in a neural network performs a computation on its inputs and produces an output. The neurons in one layer are connected to the neurons in the subsequent layer through weighted connections. These weights determine the strength of the connections and are adjusted during the training process to optimize the network's performance.

Mathematics of a Basic Neural Network

A basic neural network consists of multiple layers of neurons connected by weighted connections. Let's consider a neural network with one input layer, one hidden layer, and one output layer.

Notation:

- Input layer: $X = [x_1, x_2, \ldots, x_n]$, where x_i represents the i-th input feature.
- Hidden layer: $H = [h_1, h_2, \ldots, h_m]$, where h_i represents the i-th neuron in the hidden layer.
- Output layer: $Y = [y_1, y_2, \ldots, y_k]$, where y_i represents the i-th output neuron.

Forward Propagation:

The weighted sum of inputs for a neuron in the hidden layer is calculated as:

$$z_j = \sum_{i=1}^{n} w_{ji}^{(1)} x_i + b_j^{(1)}$$

where $w_{ji}^{(1)}$ represents the weight connecting the i-th input to the j-th neuron in the hidden layer, and $b_j^{(1)}$ is the bias term for the j-th neuron.

The output of each neuron in the hidden layer is obtained by applying an activation function σ:

$$h_j = \sigma(z_j)$$

Similarly, the weighted sum of inputs for a neuron in the output layer is calculated as:

$$z_k = \sum_{j=1}^{m} w_{kj}^{(2)} h_j + b_k^{(2)}$$

where $w_{kj}^{(2)}$ represents the weight connecting the j-th neuron in the hidden layer to the k-th neuron in the output layer, and $b_k^{(2)}$ is the bias term for the k-th neuron.

The output of each neuron in the output layer is obtained by applying an activation function σ:

$$y_k = \sigma(z_k)$$

Activation Functions:

Activation functions are a vital component of neural networks. They introduce non-linearity and enable the neural network to learn complex relationships in the data. Here are a few reasons why activation functions are necessary:

1. **Non-Linearity**: Without activation functions, the neural network would only be able to approximate linear functions, limiting its learning capacity. Activation functions allow the network to model non-linear relationships between inputs and outputs.
2. **Normalization**: Activation functions can normalize the output of a neuron, ensuring that the values fall within a desired range. This can help in stabilizing the learning process and improving the convergence of the network.
3. **Differentiability**: Activation functions are differentiable, which is essential for training neural networks using gradient-based optimization algorithms like backpropagation. The gradients of the activation functions help in updating the weights and biases during the training process.

Common activation functions used in neural networks include the sigmoid function, tanh (hyperbolic tangent) function, and Rectified Linear Unit (ReLU) function.

By applying activation functions after the weighted sum of inputs, neural networks can model complex relationships and make non-linear predictions, enabling them to solve a wide range of problems.

Feedforward Propagation

In feedforward propagation, the information flows through the network in the forward direction, starting from the input layer and passing through the hidden layers until it reaches the output layer. Each neuron receives inputs from the previous layer, computes its weighted sum, applies the activation function, and passes the output to the next layer.

Backpropagation

Backpropagation is an algorithm used to train neural networks by adjusting the weights based on the calculated gradients of the loss function with respect to the weights. It involves computing the error between the predicted outputs and the actual targets and propagating this error backward through the network to update the weights.

Loss Functions

Loss functions quantify the difference between the predicted outputs of the neural network and the actual targets. Common loss functions include mean squared error (MSE) for regression tasks and cross-entropy for classification tasks. The choice of the loss function depends on the nature of the problem being solved.

Optimization Algorithms

Optimization algorithms, such as stochastic gradient descent (SGD) and its variants (e.g., Adam, RMSprop), are used to minimize the loss function and update the weights of the neural network during training. These algorithms adjust the weights iteratively based on the gradients computed through backpropagation.

Hidden Layers and Network Depth

The hidden layers in a neural network perform the computations necessary for feature extraction and representation learning. The number of hidden layers and the number of neurons in each layer, referred to as the network's depth, are hyperparameters that can be adjusted based on the complexity of the problem and the available computational resources.

Deep Neural Networks

Deep neural networks refer to neural networks with multiple hidden layers. Deep learning has gained significant attention due to the ability of deep neural networks to

learn hierarchical representations and solve complex problems in areas such as computer vision, natural language processing, and speech recognition.

Neural network architecture plays a crucial role in the performance and capabilities of the model. Choosing the right architecture, including the number of layers, the number of neurons, and the activation functions, is essential for achieving optimal results in various machine learning and deep learning tasks.

Neural Network Prediction for Regression and Classification

Neural networks are versatile models that can be used for both regression and classification tasks. The prediction process differs slightly depending on the type of problem being addressed.

Regression Prediction

In regression tasks, the goal is to predict a continuous numerical value as the output. Here's how neural networks make predictions for regression:

1. **Feedforward Propagation**: The input data is passed through the neural network in the forward direction. Each neuron in the network receives inputs from the previous layer, computes the weighted sum of inputs, applies an activation function, and passes the output to the next layer. This process continues until the output layer is reached.
2. **Output Layer**: In regression, the output layer typically consists of a single neuron that produces a continuous numerical value. The activation function used in the output layer depends on the specific requirements of the problem. For example, a ReLU function is commonly used for regression tasks.
3. **Final Prediction**: The output value of the neural network's output neuron represents the predicted value for the regression task. It can be interpreted as the model's estimation or approximation of the target value based on the given input.

Classification Prediction

In classification tasks, the goal is to assign input data to specific categories or classes. Neural networks can perform multi-class or binary classification. Here's how neural networks make predictions for classification:

1. **Feedforward Propagation**: Similar to regression, the input data is propagated through the network in the forward direction. Each neuron computes the weighted sum of inputs, applies an activation function, and passes the output to the next layer.
2. **Output Layer**: In classification, the output layer depends on the number of classes in the problem. For binary classification, the output layer typically consists of a single neuron using a sigmoid activation function, which produces a value between 0 and 1 representing the probability of belonging to the positive class. For multi-class classification, the output layer may consist of multiple neurons using softmax activation, where each neuron represents the probability of belonging to a specific class.
3. **Final Prediction**: In binary classification, the predicted class can be determined based on a threshold value (e.g., 0.5). If the output probability is above the threshold, the sample is classified as the positive class; otherwise, it is classified as the negative class. In multi-class classification, the class with the highest predicted probability is assigned as the predicted class.

Neural networks learn the mapping between the input data and the desired output through the training process, where the weights and biases are adjusted to minimize the error between the predicted output and the actual target values. Once trained, the neural network can be used to make predictions on new, unseen data.

Understanding how neural networks make predictions in regression and classification tasks is crucial for interpreting the model's outputs and evaluating its performance.

Classification Evaluation Metrics

Classification evaluation metrics are used to evaluate the performance of a machine learning model that is trained for classification tasks. Some of the commonly used classification evaluation metrics are F1 score, recall score, confusion matrix, and ROC AUC score. Here's an overview of each of these metrics:

F1 score: The F1 score is a metric that combines the precision and recall of a model into a single value. It is calculated as the harmonic mean of precision and recall, and is

expressed as a value between 0 and 1, where 1 indicates perfect precision and recall. F1 score is the harmonic mean of precision and recall. It is calculated as follows:

$$F1 = \frac{2}{\frac{1}{precision} + \frac{1}{recall}}$$

where precision is the number of true positives divided by the sum of true positives and false positives, and recall is the number of true positives divided by the sum of true positives and false negatives.

Recall: Use the recall score when the cost of false negatives (i.e., missing instances of a class) is high. For example, in a medical diagnosis problem, the cost of missing a positive case may be high, so recall would be a more appropriate metric. Recall score (also known as sensitivity) is the number of true positives divided by the sum of true positives and false negatives. It is given by the following formula:

$$Recall = \frac{TP}{TP + FN}$$

Precision: Precision is another important classification evaluation metric, which is defined as the ratio of true positives to the total predicted positives. It measures the accuracy of positive predictions made by the classifier, i.e., the proportion of positive identifications that were actually correct. The formula for precision is:

$$precision = \frac{true\ positive}{true\ positive + false\ positive}$$

where true positive refers to the cases where the model correctly predicted the positive class, and false positive refers to the cases where the model incorrectly predicted the positive class. Precision is useful when the cost of false positives is high, such as in medical diagnosis or fraud detection, where a false positive can have serious consequences. In such cases, a higher precision indicates that the model is better at identifying true positives and minimizing false positives.

Confusion Matrix: A confusion matrix is a table that is often used to describe the performance of a classification model. It compares the predicted labels with the true labels and counts the number of true positives, false positives, true negatives, and false negatives. Here is an example of a confusion matrix:

	Actual Positive	Actual Negative
Predicted Positive	True Positive (TP)	False Positive (FP)
Predicted Negative	False Negative (FN)	True Negative (TN)

ROC AUC Score: ROC AUC (Receiver Operating Characteristic Area Under the Curve) score is a measure of how well a classifier is able to distinguish between positive and negative classes. It is calculated as the area under the ROC curve. The ROC curve is created by plotting the true positive rate (TPR) against the false positive rate (FPR) at various threshold settings. TPR is the number of true positives divided by the sum of true positives and false negatives, and FPR is the number of false positives divided by the sum of false positives and true negatives.

$$ROC\ AUC\ Score = \int_0^1 TPR(FPR^{-1}(t))dt$$

where FPR^{-1} is the inverse of the FPR function.

When to use which:

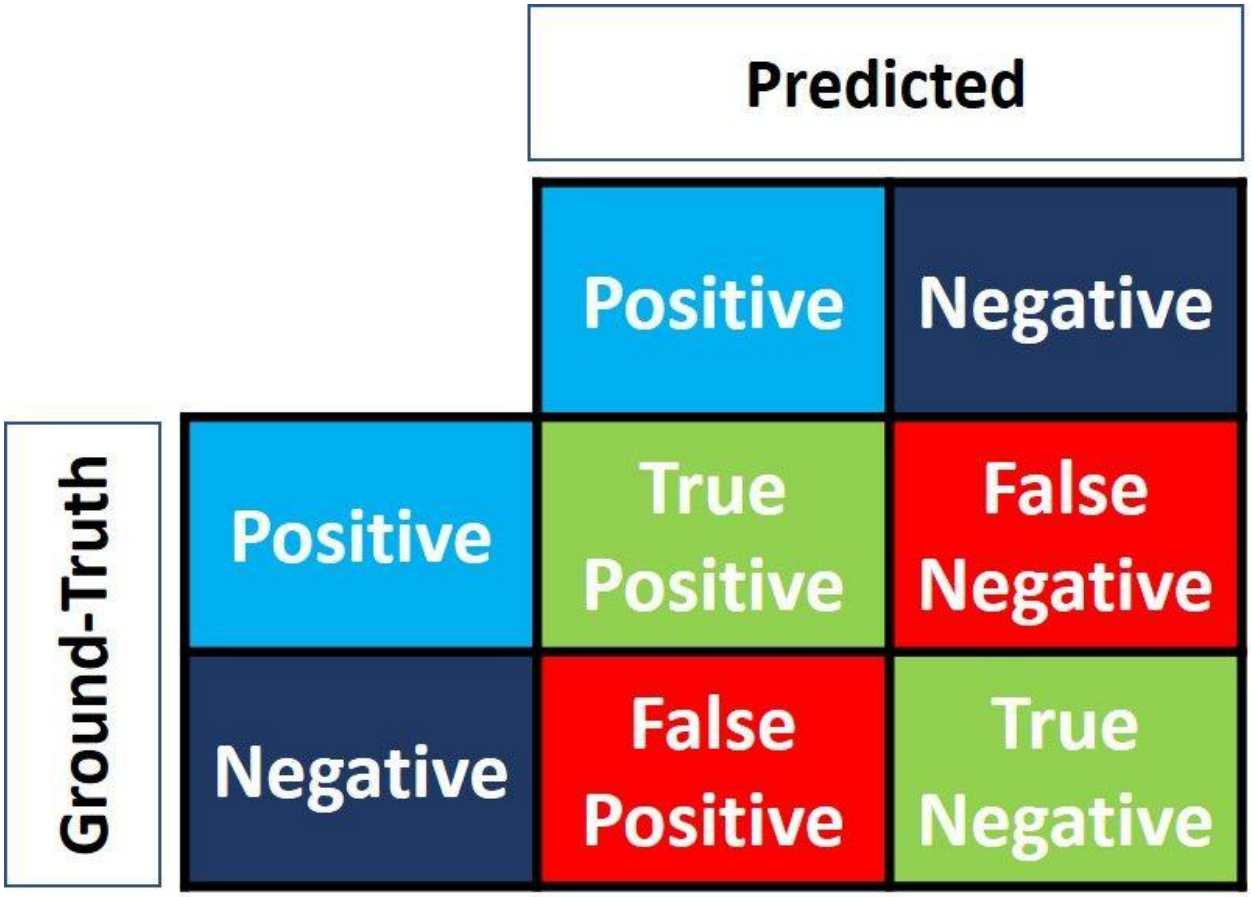
The choice of evaluation metric depends on the specific requirements of the business problem. Here are some general guidelines:

- F1 score: Use the F1 score when the class distribution is imbalanced, and when both precision and recall are equally important.
- Recall score: Use the recall score when the cost of false negatives (i.e., missing instances of a class) is high. For example, in a medical diagnosis problem, the cost of missing a positive case may be high, so recall would be a more appropriate metric.
- Precision: Precision is useful when the cost of false positives is high, such as in medical diagnosis or fraud detection, where a false positive can have serious consequences. In such cases, a higher precision indicates that the model is better at identifying true positives and minimizing false positives.
- Confusion matrix: The confusion matrix is a versatile tool that can be used to visualize the performance of a model across different classes. It can be useful for identifying specific areas of the model that need improvement.
- ROC AUC score: Use the ROC AUC score when the ability to distinguish between positive and negative classes is important. For example, in a credit scoring problem, the ability to distinguish between good and bad credit risks is crucial.

Importance with respect to the business problem:

The importance of each evaluation metric varies depending on the business problem. For example, in a spam detection problem, precision may be more important than recall, since false positives (i.e., classifying a non-spam email as spam) may annoy users, while false negatives (i.e., missing a spam email) may not be as harmful. On the other hand, in a disease diagnosis problem, recall may be more important than precision, since missing a positive case (i.e., a false negative) could have serious consequences. Therefore, it is important to choose the evaluation metric that is most relevant to the specific business problem at hand.

Evaluation metrics



recision-Recall (PR) curve and Area Under the Curve (AUC) curve are evaluation metrics commonly used in binary classification problems

to assess the performance of a model and determine an appropriate threshold for decision making.

The Precision-Recall (PR) curve is a graphical representation of the trade-off between precision and recall for different classification thresholds. Precision is the ratio of true positive predictions to the total number of positive predictions, while recall (also known as sensitivity or true positive rate) is the ratio of true positive predictions to the total number of actual positive instances in the data. The PR curve plots precision on the y-axis and recall on the x-axis, with each point on the curve representing a different classification threshold. A higher precision and recall indicate better model performance.

The Area Under the Curve (AUC) is a single scalar value that summarizes the PR curve. It measures the overall performance of the model across all possible classification thresholds. The AUC value ranges from 0 to 1, where a higher value indicates better model performance. An AUC of 1 represents a perfect model that achieves maximum precision and recall across all thresholds.

Choosing the right threshold depends on the specific requirements of your problem. The PR curve can help you visualize the precision-recall

trade-off at different thresholds. If your problem prioritizes precision (minimizing false positives), you may want to choose a threshold that maximizes precision while maintaining a reasonable level of recall. On the other hand, if recall is more important (minimizing false negatives), you would choose a threshold that maximizes recall while still maintaining an acceptable level of precision.

The selection of the threshold ultimately depends on the cost or impact of false positives and false negatives in your specific problem domain. By analyzing the PR curve and considering the specific requirements and trade-offs of your problem, you can make an informed decision about the threshold that best balances precision and recall for your particular use case.

Conclusion

In this project we used a bunch of supervised models to predict if the customer would be interested in a lead.

A successful data science project requires a clear understanding of the business problem and the data available, as well as the ability to select and apply appropriate data preprocessing techniques, feature engineering methods, and machine learning algorithms. It is also important to assess and optimize the performance of the model and communicate the results effectively to stakeholders.

After looking at the PR and ROC curves above, we can conclude that **LightGBM** is giving us the best possible results.

Interview Questions

Supervised Learning:

- What are decision trees? How does the model decide on the split?
- What is boostrap aggregation?
- Explain bias and variance in context of boosting and bagging?
- How can you use decision tree for missing value imputation?
- How does regularization work in gradient boosting models?
- Can you describe the working principles of Logistic Regression, Naive Bayes, Support Vector Machines, Random Forest, Light Gradient Boosting, Extreme Gradient Boosting, and Neural Network algorithms?
- How do these algorithms handle different types of data and decision boundaries?
- What are the key hyperparameters that need to be tuned for each algorithm?
- How would you evaluate the performance of each algorithm in a classification problem?
- How would you interpret the results of the evaluation metrics and draw conclusions about the effectiveness of each approach?
- Can you explain the concept of overfitting and how it can impact model performance?

Code Implementation:

- Is bagging or boosting computationally faster?
- Can you write your own code to calculate precision and recall?
- How would you handle dataset if the target variable would have been imbalanced?
- Can you do feature importance and interpret the results of feature importance analysis and communicate the findings to stakeholders?