Build marketing budget optmization model for B2C sales

Introduction

A company specializing in B2C sales (product: Data Science and Data Engineering related courses) spends significant money on marketing campaigns across various channels such as social media, email, and search advertising. More information on marketing sources can be found in the data column “Marketing Source”. The marketing team faces challenges in optimizing the marketing budget allocation across these channels to maximize revenue and return on investment (ROI).

This project aims to develop a data-driven approach to optimize the marketing budget allocation across various channels to maximize revenue and ROI. By using machine learning algorithms, the marketing team can make informed decisions about allocating the marketing budget based on predicted revenue and ROI.

Business Impact of Marketing Budget Optimization

Increase product conversions: Marketing Budget Optimization 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.

Improve budget allocation: Over budgeting on non-efficient channels lead to waste of marketing money without getting enough revenue.

Improve Customer Acquisition Cost: Customer Acquisition Cost(CAC) improves if right targeting channels are used for a customer often leading to better repeat rates as well.

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 Marketing Budget Optimization

There are various ways to build a model for marketing budget optimization, 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. Rule-based systems:

In this approach, we typically look at budget proportion allocation based on predefined rules. Rule-based systems are easy to implement and interpret, but they may not be very accurate or adaptive. It makes sense to use them when you have a specific business mandate (example - ease of managing ads or segment of users you want to target).

2. Statistical techniques:

Statistical techniques such as correlation, probability are very helpful to find the righ budget mix for each marketing channel. In this approach, we use historical data to come up with priors and correlations to do budget allocation.

3. Linear optimizatiom:

In this approach, we define the objective function(e.g. increase conversion, reduce total cost) and the constraints(e.g. we should not spend more than 1000$ on Instagram) 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 amount of money company needs to spend on each channel.

4. Machine learning:

Machine learning systems combines strength of historical data and statistical techniques to explain the right channel 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 an 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 allocate budgets accordingly.

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 tree based model 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:
    - Random Forest
    - Light Gradient Boosting
    - Extreme Gradient Boosting

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

---

Learning Outcomes

- Understanding the importance of data-driven decision-making in marketing budget allocation.
- Leveraging machine learning algorithms to optimize marketing budget allocation and maximize revenue and ROI.
- Identifying the significance of targeting the right users through the appropriate channels and assets for improved conversions.
- Data preprocessing, encoding, and splitting into training and testing sets
- Recognizing the impact of targeted marketing on customer acquisition costs and improving customer retention.
- Understanding and Implementing Tree-based models like Random Forest, Light Gradient Boosting, and Extreme Gradient Boosting to effectively predict interest.
- What are PR and AUC curves? And how do they help in chosing the right threshold?
- Improving overall business performance by aligning marketing efforts with customer preferences and behavior.

Prerequisites

- Familiarity with Python programming language
- Familiarity with Pandas, sklearn, numpy libraries in Python, along with concepts like loops, lists, arrays and dataframe
- Basic knowledge of machine learning concepts such as supervised learning, tree models
- Understanding of data preprocessing techniques such as handling missing values, outliers, and categorical variables

- Knowledge of Jupyter Notebook or any other Python IDE.
- Understanding metrics such as precision, recall, PR curve, AUC curve

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

Facts

Binarization is the process of dividing data into two groups and assigning one out. of two values to all the members of the same group. This is usually accomplished. by defining a threshold t and assigning the value 0 to all the data points below. the threshold and 1 to those above it.

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.

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.

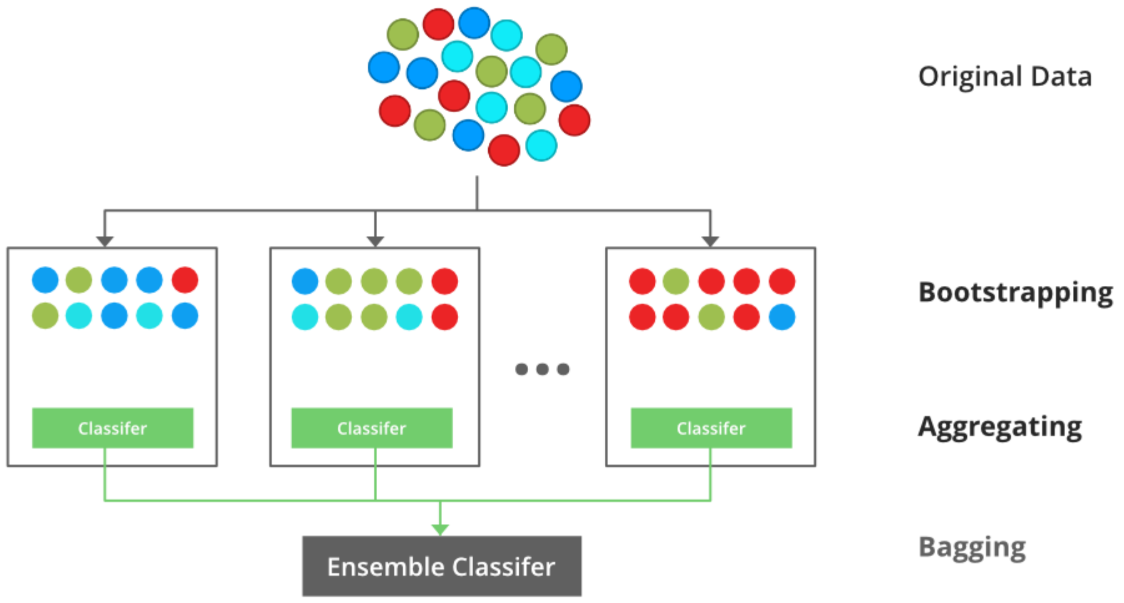
When a decision tree is the weak learner, the resulting algorithm is called gradient-boosted trees; it usually outperforms random forest. A gradient-boosted trees model is built in a stage-wise fashion as in other boosting methods, but it generalizes the other methods by allowing optimization of an arbitrary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

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

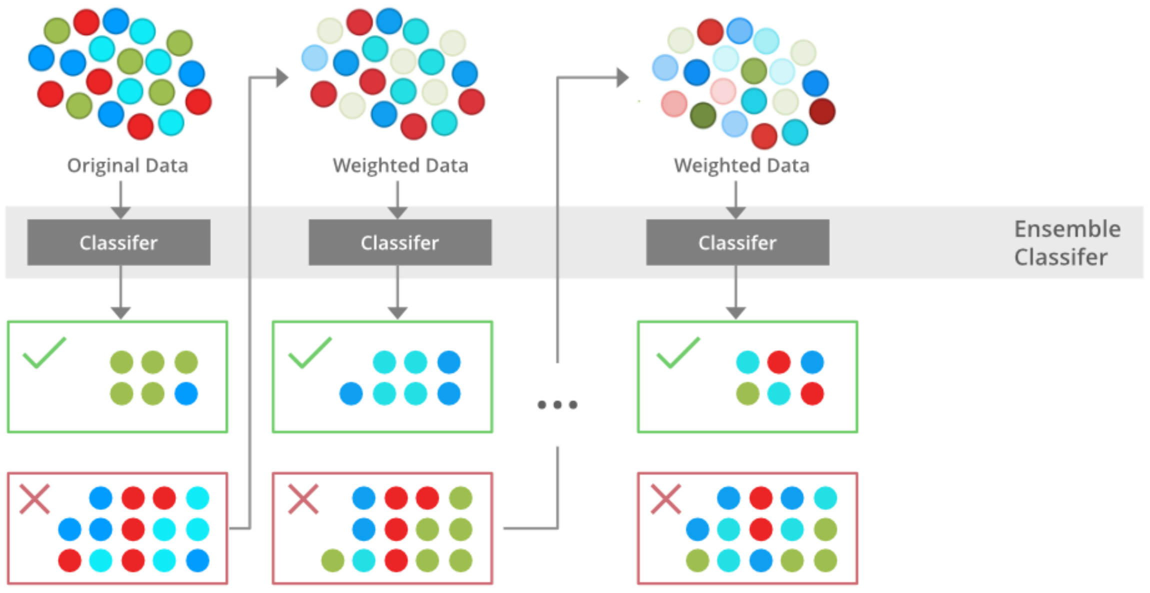
---

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

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:

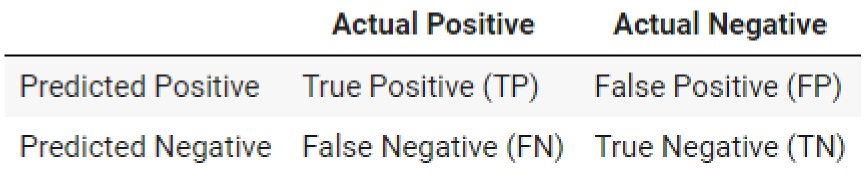
$$Recall = \frac{TP}{TP + FN}$$

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:

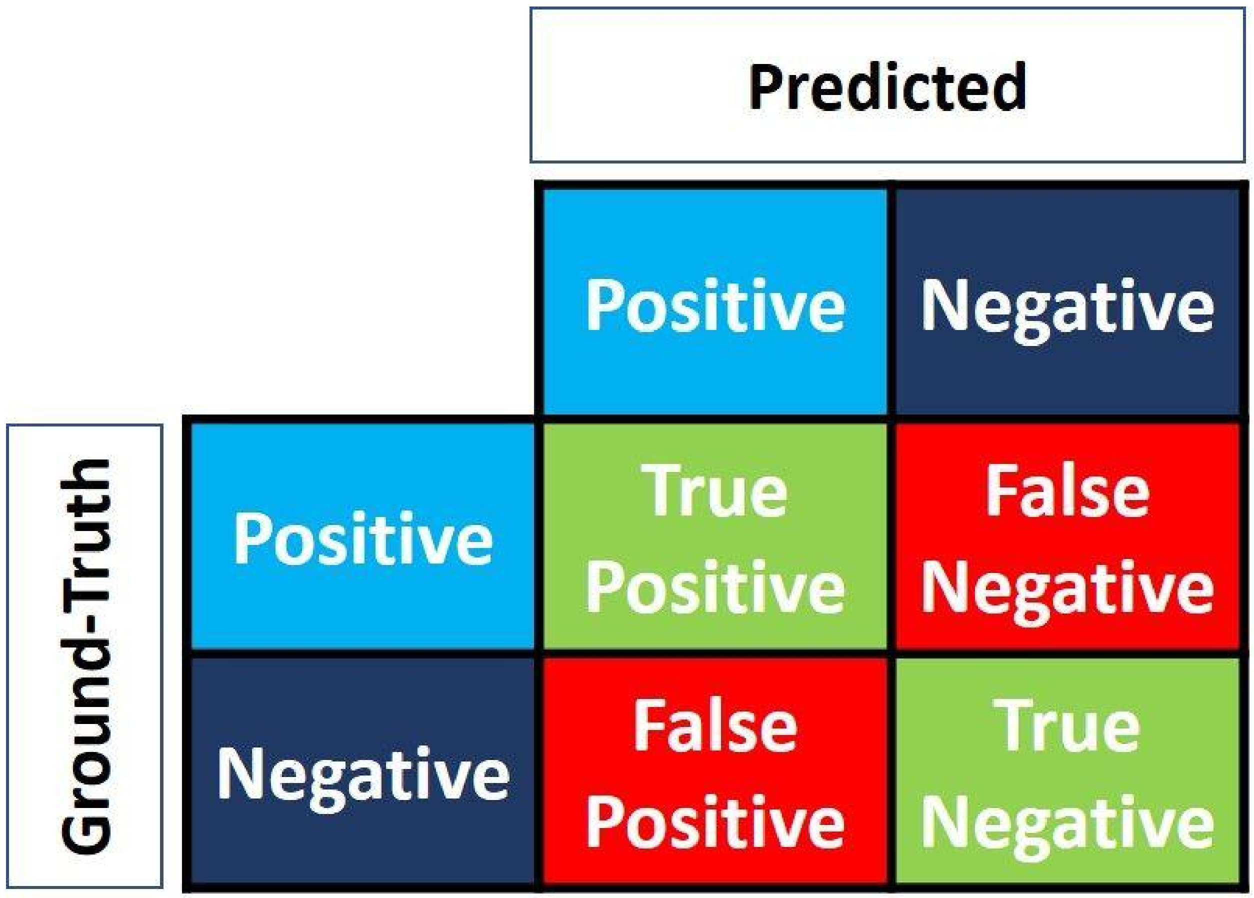
The choice of evaluation metric depends on the specific requirements of the business problem. Here are some general guidelines:

- 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



		Predicted	
		Positive	Negative
Ground-Truth	Positive	True Positive	False Negative
	Negative	False Positive	True Negative

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.

The problem could have been formulated as a multi-class classification problem but we instead formulated this as a binary classification problem and the confidence on the predictions would enable the stakeholders to chase the 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?

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?