Telecom Churn Prediction Learning Material

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.

Outlier Detection:

Outlier detection is a critical data analysis technique that involves identifying and removing data points that are significantly different from the rest of the data. Outliers are data points that lie far away from the rest of the data, and they can significantly influence the statistical analysis and machine learning models' performance. Therefore, identifying and removing outliers is essential to ensure accurate and reliable data analysis results.

There are two main approaches for outlier detection: parametric and non-parametric.

- Parametric Methods: Parametric methods assume that the data follows a specific distribution, such as a normal distribution. In this approach, outliers are identified by calculating the distance of each data point from the mean of the distribution in terms of the number of standard deviations. Data points that are beyond a certain number of standard deviations (usually three or more) are considered as outliers.

One common parametric method is the Z-score method, which calculates the distance of each data point from the mean in terms of standard deviations. Parametric methods can be useful when the data follows a known distribution, but they may not be effective when the data is not normally distributed.

- Non-Parametric Methods: Non-parametric methods do not assume any specific distribution of the data. Instead, they rely on the rank or order of the data points. In this approach, outliers are identified by comparing the values of each data point with the values of other data points. Data points that are significantly different from other data points are considered as outliers.

Quantiles are an important concept in non-parametric outlier detection methods. They represent values that divide a dataset into equal-sized parts, such as quarters or thirds. The most commonly used quantiles are the median (which divides the data into two equal parts), the first quartile (which divides the data into the lowest 25% and the highest 75%), and the third quartile (which divides the data into the lowest 75% and the highest 25%).

The interquartile range (IQR) is another important concept related to quantiles. It is defined as the difference between the third and first quartiles and represents the middle 50% of the data. The IQR can be used to identify outliers by defining a range (known as the Tukey's fence) beyond which any data points are considered outliers.
Non-parametric methods can be useful when the data is not normally distributed or when the distribution is unknown.

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 performed all 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.

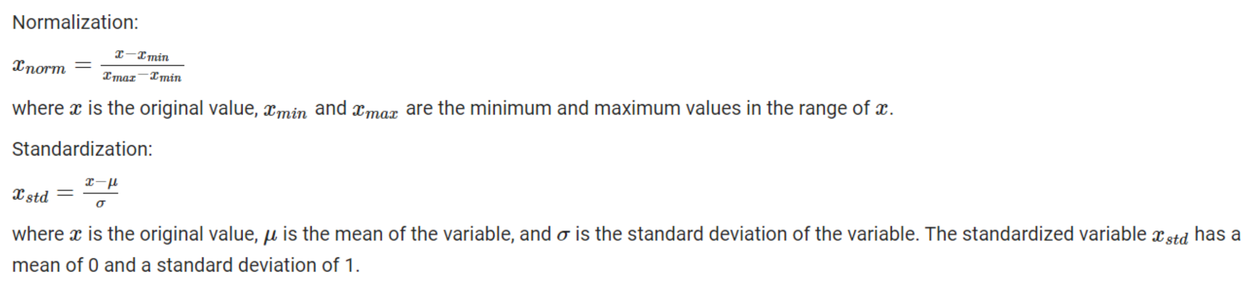
Transforming Variables

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.

Numerical variables, on the other hand, are variables that take on continuous or discrete numerical values, such as age, income, or number of children. One common way to transform numerical variables is through standardization or normalization. Standardization involves scaling the variable to have a mean of 0 and a standard deviation of 1, while normalization involves scaling the variable to have values between 0 and 1.



Normalization:

$$x_{norm} = \frac{x - x_{min}}{x_{max} - x_{min}}$$

where x is the original value, x_{min} and x_{max} are the minimum and maximum values in the range of x.

Standardization:

$$x_{std} = \frac{x - \mu}{\sigma}$$

where x is the original value, μ is the mean of the variable, and σ is the standard deviation of the variable. The standardized variable x_{std} has a mean of 0 and a standard deviation of 1.

Supervised Machine Learning

Supervised learning is a type of machine learning where the algorithm learns from labeled data. In other words, the data used to train the algorithm includes input variables and corresponding output variables. The algorithm learns to predict the output variable based on the input variables. Supervised learning can be further divided into two categories: regression and classification.

- **Regression** is a type of supervised learning where the algorithm learns to predict a continuous output variable. In other words, the output variable is a numerical value. Examples of regression problems include predicting housing prices, stock prices, or the amount of rainfall in a particular area.
- **Classification**, on the other hand, is a type of supervised learning where the algorithm learns to predict a discrete output variable. In other words, the output variable is a category or label. Examples of classification problems include predicting whether an email is spam or not, whether a tumor is malignant or benign, or whether a customer is likely to churn or not.

Logistic Regression

Logistic regression is a type of machine learning algorithm used for classification problems where we need to predict if something belongs to one category or another. For example, we can use it to predict if a customer will churn or not.

The algorithm works by analyzing the relationship between the input variables (such as customer demographics and usage patterns) and the binary output variable (such as churn or no churn). It then estimates the probability of the output variable using a logistic function, which outputs a value between 0 and 1.

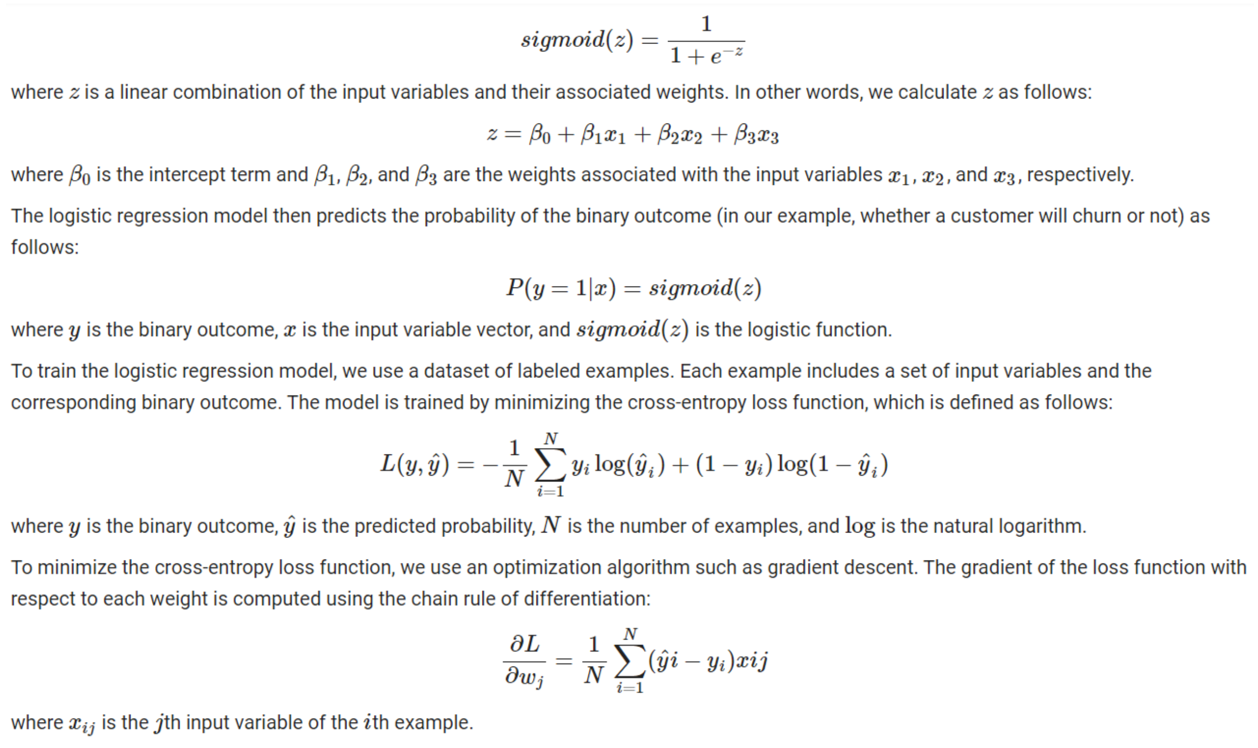
Logistic regression is actually a type of classification algorithm, but it is called "logistic regression" because it uses a logistic function to model the probability of the binary output variable.

The term "regression" comes from the fact that the logistic regression model is based on a linear combination of the input variables and their associated weights, which is similar to linear regression. However, in linear regression, we predict a continuous output variable, while in logistic regression, we predict a probability of belonging to a particular class.

In other words, logistic regression is a regression algorithm that is used for classification problems. The logistic function transforms the output of the regression equation into a probability value between 0 and 1, which can then be used to classify the input variable into one of two categories.

Let's see how!!

The logistic regression model is based on the logistic function, which maps any real-valued input to a value between 0 and 1. The logistic function is defined as follows:



$$sigmoid(z) = \frac{1}{1+e^{-z}}$$

where z is a linear combination of the input variables and their associated weights. In other words, we calculate z as follows:

$$z = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \beta_3 x_3$$

where β_0 is the intercept term and β_1, β_2, and β_3 are the weights associated with the input variables x_1, x_2, and x_3, respectively.

The logistic regression model then predicts the probability of the binary outcome (in our example, whether a customer will churn or not) as follows:

$$P(y=1|x) = sigmoid(z)$$

where y is the binary outcome, x is the input variable vector, and $sigmoid(z)$ is the logistic function.

To train the logistic regression model, we use a dataset of labeled examples. Each example includes a set of input variables and the corresponding binary outcome. The model is trained by minimizing the cross-entropy loss function, which is defined as follows:

$$L(y,\hat{y}) = -\frac{1}{N}\sum_{i=1}^{N} y_i \log(\hat{y}_i) + (1-y_i)\log(1-\hat{y}_i)$$

where y is the binary outcome, $\hat{y}$ is the predicted probability, N is the number of examples, and $\log$ is the natural logarithm.

To minimize the cross-entropy loss function, we use an optimization algorithm such as gradient descent. The gradient of the loss function with respect to each weight is computed using the chain rule of differentiation:

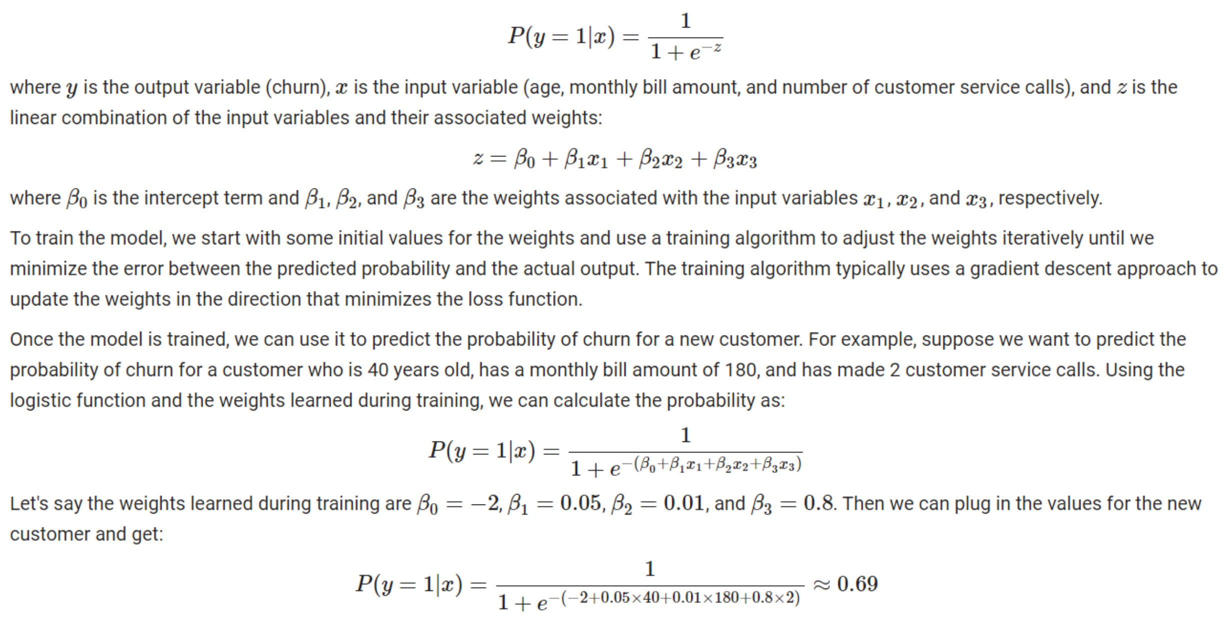
$$\frac{\partial L}{\partial w_j} = \frac{1}{N}\sum_{i=1}^{N}(\hat{y}i - y_i)xij$$

where x_{ij} is the jth input variable of the ith example.

Logistic Regression: Mathematical Example

Suppose we have the following dataset with three input variables (customer age, monthly bill amount, and number of customer service calls) and a binary output variable (1 for churn and 0 for no churn):

We can use logistic regression to build a model that predicts the probability of churn based on these input variables. The logistic function that we use is:



$$P(y=1|x) = \frac{1}{1+e^{-z}}$$

where y is the output variable (churn), x is the input variable (age, monthly bill amount, and number of customer service calls), and z is the linear combination of the input variables and their associated weights:

$$z = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \beta_3 x_3$$

where β_0 is the intercept term and β_1, β_2, and β_3 are the weights associated with the input variables x_1, x_2, and x_3, respectively.

To train the model, we start with some initial values for the weights and use a training algorithm to adjust the weights iteratively until we minimize the error between the predicted probability and the actual output. The training algorithm typically uses a gradient descent approach to update the weights in the direction that minimizes the loss function.

Once the model is trained, we can use it to predict the probability of churn for a new customer. For example, suppose we want to predict the probability of churn for a customer who is 40 years old, has a monthly bill amount of 180, and has made 2 customer service calls. Using the logistic function and the weights learned during training, we can calculate the probability as:

$$P(y=1|x) = \frac{1}{1+e^{-(\beta_0+\beta_1 x_1+\beta_2 x_2+\beta_3 x_3)}}$$

Let's say the weights learned during training are $\beta_0 = -2$, $\beta_1 = 0.05$, $\beta_2 = 0.01$, and $\beta_3 = 0.8$. Then we can plug in the values for the new customer and get:

$$P(y=1|x) = \frac{1}{1+e^{-(-2+0.05\times 40+0.01\times 180+0.8\times 2)}} \approx 0.69$$

So, the model predicts that there is a 69% probability that this customer will churn. We can use this probability to classify the customer as churn or no churn, depending on a threshold that we set (e.g., if the probability is above 0.5, we classify the customer as churn).

This is a simple example, but it illustrates how logistic regression uses the logistic function and linear combination of input variables to predict the probability of a binary output variable.

Decision Trees

Decision Trees in Classification

Decision trees are a type of supervised learning algorithm that can be used for classification as well as regression problems. They are widely used in machine learning because they are easy to understand and interpret, and can handle both categorical and numerical data. The idea behind decision trees is to create a model that predicts the value of a target variable by learning simple decision rules inferred from the data features.

The decision tree starts with a single node, called the root node, which represents the entire dataset. The root node is then split into several child nodes based on the value of a chosen feature. The process of selecting the best feature and splitting the nodes is repeated recursively for each child node until a stopping criterion is reached. This results in a tree-like structure that represents the decision rules learned from the data.

Each node in the decision tree represents a decision or a test of a feature value, and each branch represents the possible outcomes of that decision. The leaves of the tree represent the final decision or the class label assigned to the input data.

Splitting Criteria

To build a decision tree, we need a measure that determines how to split the data at each node. The splitting criterion is chosen based on the type of data and the nature of the problem. The most common splitting criteria are:

- Gini index: measures the impurity of a set of labels. It calculates the probability of misclassifying a randomly chosen element from the set, and is used to minimize misclassification errors.
- Information gain: measures the reduction in entropy (uncertainty) after a split. It is used to maximize the information gain in each split.
- Chi-square: measures the difference between observed and expected frequencies of the classes. It is used to minimize the deviation between the observed and expected class distribution.

Overfitting in Decision Trees

One of the main challenges in building decision trees is overfitting. Overfitting occurs when the tree is too complex and fits the training data too well, resulting in poor performance on new and unseen data. This can be addressed by pruning the tree or limiting its depth, or by using ensemble methods such as bagging and boosting.

Ensemble Methods

Ensemble methods are techniques that combine multiple models to improve performance and reduce overfitting. The two most common ensemble methods used with decision trees are:

- Bagging (Bootstrap Aggregating): involves training multiple decision trees on different subsets of the training data and then combining their predictions by averaging or voting. This reduces the variance and improves the stability of the model.
- Boosting: involves training multiple decision trees sequentially, where each subsequent tree focuses on the misclassified examples of the previous tree. This reduces the bias and improves the accuracy of the model.

Deecision trees are powerful tools for classification problems that provide a clear and interpretable representation of the decision rules learned from the data. The choice of splitting criterion, stopping criterion, and ensemble method can have a significant impact on the performance and generalization of the model.

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.

Random Forest

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.

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.

Gradient Boosting

The primary idea behind this technique is to develop models in a sequential manner, with each model attempting to reduce the mistakes of the previous model.The additive model, loss function, and a weak learner are the three fundamental components of Gradient Boosting.

The method provides a direct interpretation of boosting in terms of numerical optimization of the loss function using Gradient Descent. We employ Gradient Boosting Regressor when the target column is continuous, and Gradient Boosting Classifier when the task is a classification problem. The "Loss function" is the only difference between the two. The goal is to use gradient descent to reduce this loss function by adding weak learners. Because it is based on loss functions, for regression problems, Mean squared error (MSE) will be used, and for classification problems, log-likelihood.

XG Boost

XGBoost is a variant of gradient boosting, which is a popular ensemble learning technique that works by iteratively adding new models to an ensemble, each model attempting to correct the errors made by the previous models. In each iteration, the algorithm calculates the negative gradient of the loss function with respect to the current prediction, and fits a new model to the residual errors. The new model is then added to the ensemble, and the algorithm repeats this process until the desired number of models is reached.

In XGBoost, the objective function is used to measure the difference between the predicted values and the true labels. The objective function is a sum of the loss function and the regularization term, where the latter prevents overfitting and encourages the model to be simple.

Suppose we have a dataset with three features, x1, x2, and x3, and we want to predict a binary outcome, y. We decide to use decision trees as our weak learners. We start by training a decision tree on the entire dataset. However, this decision tree may not be able to capture the complex relationships between the features and the outcome, and it may be overfitting the training data.

To improve upon the first decision tree, we can use XGBoost. Here's how:

- Initialize the model: We start by initializing the XGBoost model with default hyperparameters. This model will be a simple decision tree with a single split.
- Make predictions: We use this model to make predictions on the training data. We compare these predictions to the true labels and calculate the residuals, which are the differences between the predicted values and the true labels.
- Fit a new tree: We then fit a new decision tree to the residuals. This tree will be a weak learner, as it is only modeling the errors of the previous model.
- Combine the models: We add the new tree to the previous model to create a new ensemble. This new ensemble consists of the previous model plus the new tree.
- Repeat: We repeat steps 2-4 for a specified number of iterations, adding a new tree to the ensemble each time.
- Predictions: To make predictions on new data, we combine the predictions of all the trees in the ensemble.

The key idea behind XGBoost is that it improves upon the predictions of the weak learners by focusing on the misclassified data points. By fitting a new tree to the residuals, XGBoost can correct the errors of the previous model and improve its overall accuracy. Additionally, XGBoost uses regularization to prevent overfitting and to improve generalization 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.

Importance of Evaluation Metrics in Churn

Churn is a critical business problem for many companies because losing customers can have a significant impact on revenue and profitability. Therefore, it is essential to have accurate and reliable churn prediction models that can identify customers who are at risk of leaving.

Evaluation metrics such as F1-score, recall, and ROC-AUC score provide insight into the performance of churn prediction models. By analyzing these metrics, businesses can determine how well their models are performing and make informed decisions about how to improve them. For example, if the F1-score is low, it may indicate that the model is not accurately identifying customers who are likely to churn. This may prompt businesses to re-evaluate their feature selection, hyperparameter tuning, or even their data collection processes.

Moreover, evaluation metrics also help businesses make trade-offs between different prediction models. For example, in some cases, a company may prioritize recall over precision because they want to identify as many at-risk customers as possible, even if it means some false positives. On the other hand, in other cases, a company may prioritize precision over recall because they want to avoid incorrectly flagging customers as at-risk and taking unnecessary retention actions.

Therefore, evaluation metrics play a crucial role in helping businesses optimize their churn prediction models to minimize churn and retain valuable customers.

Data Drift Monitoring

So Why Is Drift So Important?

Machine learning models are meant to predict on unseen data, based on previous known data. If the data or the relationships between features and the target label have changed, our model's performance may degrade.

Detecting drift is an important warning sign that our model may be not as accurate on newer data (compared to the training data), and that it should be adjusted or retrained on different data. In production environments, detecting drift (and other measures derived from drift, such as model confidence) is often the only way to know that our model performance is deteriorating, as in many cases the label is unknown for some time after the prediction is made.

It is important to note that not all changes in data represent drift. For example, periodic changes in data due to daily, weekly or seasonal changes are usually not considered drift, as they are often present in the training data.

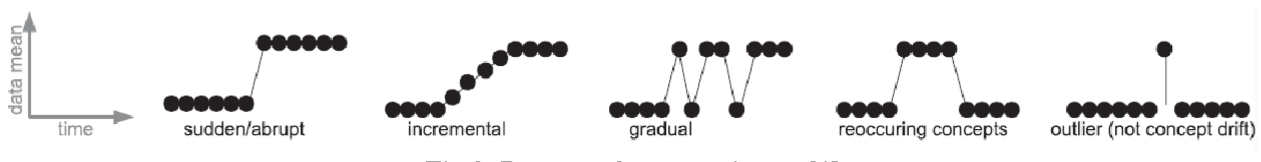


Fig 2: Patterns of concept change [4]

Which Types of Drift Are There?

In machine learning, we usually refer to 2 types of drift:

Data Drift

Data drift is any change in the distribution of the data.

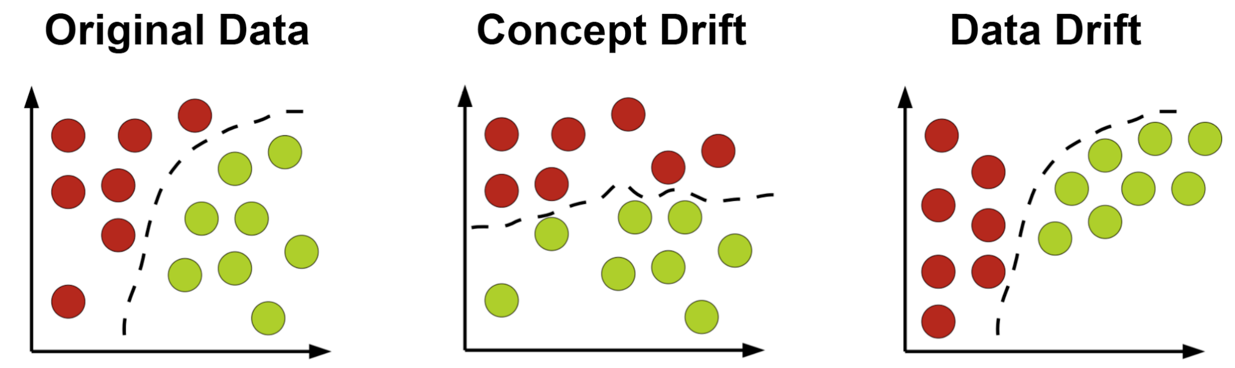
For example, in a dataset predicting a person's income, the target (income) is highly correlated with high level of education (advanced academic degrees). A government plan to help people of lower social-economic status to get higher education, would create a data drift that changes how the data distributes. However, this will not change the relation between a person's salary and their level of education, as these new graduates will be able to work in better paying professions.

Concept Drift

Concept drift is a change in the underlying relation between the data and the label.

Continuing the example of predicting income using the level of education, let's assume that a change in the job market (for example, the rise of high-tech companies) caused drift in the data: suddenly, job experience became more significant for programming jobs than a degree in computer science. Now, the relation between the level of education and the income has changed - and a person's salary can't be predicted from their level of education as accurately as it was on previous data.

Concept drift will almost always require some changes to the model, usually by retraining of the model on newer data.

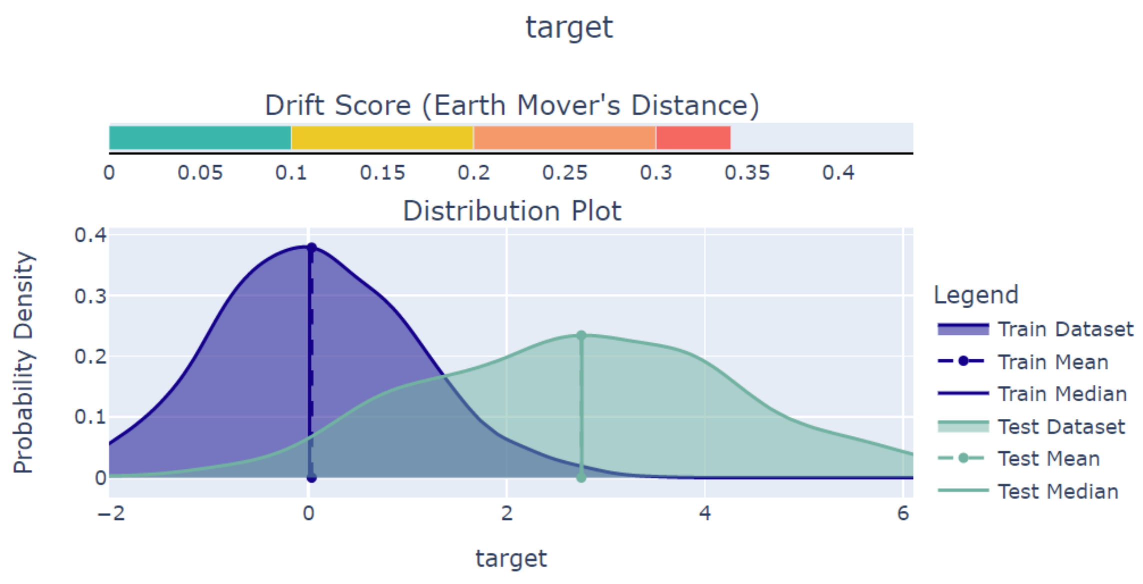


Model Drift and Concept Drift are essentially the same thing. They both refer to the phenomenon where the underlying data generating process changes over time, leading to a degradation in the performance of a machine learning model.

When a model is trained on a dataset and then deployed to make predictions in the real world, it assumes that the statistical properties of the data remain constant over time. However, this is often not the case in practice. For example, in the context of natural language processing, the meanings of words can shift over time, rendering a language model trained on historical data less effective on current data.

When the statistical properties of the data change over time, it can lead to a situation where the model's predictions become less accurate, even if the model itself remains unchanged. This is what is known as model drift or concept drift. To mitigate this issue, techniques such as online learning or retraining the model on fresh data can be used.

What Can You Do in Case of Drift?



When suspecting drift in your data, you must first understand what changed in the data - were it the features, the labels, or maybe just the predictions. In deepchecks, we show a drift score for each feature, starting with your most important features, giving you an idea of the severity of your drift, even if you're not still sure of its source.

It is recommended to manually explore your data and try to understand the root cause of your changes, in order to estimate the effect of the change on your model's performance.

Retrain Your Model

If you have either kind of drift, retraining your model on new data that better represents the current distribution, is the most straight-forward solution. However, this solution may require additional resources such as manual labeling of new data, or might not be possible if labels on the newer data are not available yet.

Retraining is usually necessary in cases of concept drift. However, retraining may still be of use even for other cases, such as data drift that caused a change in the label's distribution, but not in the ability to predict the label from the data. In this cas, retraining the model with the correct distribution of the label can improve the model's performance (this is not relevant when the training dataset is sampled so labels are evenly distributed).

Reference: [Deepchecks Documentation]()

Inference Pipeline with and without Label Availability

In real-world scenarios, we often need to deploy machine learning models to make predictions on new data. In order to make accurate predictions, the input data must be preprocessed and checked for any data drift.

However, there are two cases of inference pipeline, depending on whether the label is available or not.

Case 1: Label is not available In this case, we cannot check the model drift and retrain the model, as we do not have the true label values for the new data. Therefore, we need to use the previously trained model to make predictions on the new data.

In this case, the inference pipeline consists of two steps:

- Preprocessing the new data using the same preprocessing steps that were used for the training data.
- Using the previously trained model to make predictions on the preprocessed new data.
- Even though we cannot retrain the model in this case, we can still check for any data drift in the new data compared to the training data. This can help us identify if there are any significant changes in the data distribution that may affect the accuracy of the predictions.

Case 2: Label is available In this case, we can not only preprocess the new data and check for any data drift, but we can also check for model drift and decide whether to retrain the model.

In this case, the inference pipeline consists of three steps:

- Preprocessing the new data using the same preprocessing steps that were used for the training data.
- Checking for any data drift between the new data and the training data. If the data drift is significant, we may need to retrain the model.
- Using the previously trained or retrained model to make predictions on the preprocessed new data.

The reason why we check for data drift and model drift is that the performance of a machine learning model depends on the assumption that the training data and the new data are drawn from the same distribution. If there is significant data drift or model drift, then the model may not perform well on the new data, and we may need to retrain the model.

The inference pipeline for machine learning models is a crucial step in deploying the models in real-world scenarios. By pre-processing the new data, checking for data drift and model drift, and retraining the model when necessary, we can ensure that the model is accurate and performs well on the new data.

Model Retraining

Model retraining is a process of updating a machine learning model using newly available data. This is done to ensure that the model remains accurate and relevant to the changing data distribution. In the given code, the model retraining is performed when there is a mismatch between the actual target value and the predicted value. In such cases, the misclassified data points are collected and added back to the training data, and the model is retrained on the combined data.

In the next section, when there is data drift, the misclassified observations are used as feedback for the model to update its parameters. The feedback data is concatenated with the original training data, preprocessed, and used to train a new model. The new model is trained for a specific number of rounds and then used for prediction.

To judge if the retrained model should go ahead in production, we need to evaluate its performance on a validation set or a holdout test set. We can calculate various performance metrics such as accuracy, precision, recall, F1 score, and area under the ROC curve (AUC-ROC) to compare the performance of the new model with the old model.

If the performance of the new model is better than the old model, we can deploy it in production. However, we need to be cautious of overfitting to the feedback data. To avoid overfitting, we can use techniques such as regularization, early stopping, and cross-validation while training the new model. We can also monitor the model's performance on the production data and retrain it periodically to ensure its accuracy and robustness.

Finally, it is crucial to monitor the model's performance over time and update it as needed. This is because the data distribution may change over time, leading to a degradation in model performance. Therefore, regular retraining and monitoring of the model's performance are essential to ensure its continued relevance and accuracy.