Logistic Regression

Logistic regression is a machine learning technique for binary classification, where the output is either 0 or 1. It is based on the idea of finding the best linear combination of features that predicts the probability of an event occurring, such as buying a product, clicking a link, or having a disease.

Logistic regression uses a special function called the logistic function, or the sigmoid function, to map the linear combination of features to a probability value between 0 and 1. The logistic function has the following formula:

$$f(x) = \frac{1}{1 + e^{-x}}$$

where x is the linear combination of features, such as:

$$x = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \ldots + \beta_n x_n$$

where β_0 is the intercept, and $\beta_1, \beta_2, ..., \beta_n$ are the coefficients of the features $x_1, x_2, ..., x_n$.

The logistic function has an S-shaped curve, as shown below:

The logistic function has some nice properties, such as:

- It is always between 0 and 1, so it can be interpreted as a probability.
- It is monotonic, meaning it always increases or decreases, so it can be used to rank the inputs by their probability.
- It is differentiable, meaning it has a derivative, so it can be optimized using gradient-based methods.

To use logistic regression for classification, we need to find the optimal values of the coefficients $\beta_0, \beta_1, ..., \beta_n$ that best fit the data. This can be done by using a method called maximum likelihood estimation, which tries to maximize the likelihood of the data given the model. The likelihood is the probability of the data given the model parameters, and it can be written as:

$$L(\beta) = \prod_{i=1}^{n} p(y_i | x_i, \beta)$$

where n is the number of observations, y_i is the output label (0 or 1), x_i is the input vector, and β is the vector of coefficients.

To simplify the calculation, we can take the logarithm of the likelihood, which is called the log-likelihood, and it can be written as:

$$\ell(\beta) = \sum_{i=1}^{n} y_i \log(p(y_i | x_i, \beta)) + (1 - y_i)\log(1 - p(y_i | x_i, \beta))$$

where $p(y_i \mid x_i, \beta_i)$ is the logistic function applied to the linear combination of features.

To maximize the log-likelihood, we can use an iterative algorithm, such as gradient ascent, which updates the coefficients by moving in the direction of the gradient of the log-likelihood. The gradient is the vector of partial derivatives of the log-likelihood with respect to each coefficient, and it can be written as:

$$\nabla\ell(\beta) = \begin{bmatrix} \frac{\partial \ell}{\partial \beta_0} \\ \frac{\partial \ell}{\partial \beta_1} \\ \vdots \\ \frac{\partial \ell}{\partial \beta_n} \end{bmatrix} = \sum_{i=1}^{n} (p(y_i | x_i, \beta) - y_i) x_i$$

where x_i is the input vector with a leading 1 for the intercept term.

The gradient ascent algorithm starts with an initial guess of the coefficients, and then updates them by adding a fraction of the gradient, where the fraction is called the learning rate. The algorithm repeats this process until the log-likelihood converges to a maximum value, or until a maximum number of iterations is reached. The algorithm can be written as:

$$\beta^{(t+1)} = \beta^{(t)} + \alpha \nabla \ell(\beta^{(t)})$$

where $\beta^{(t)}$ is the vector of coefficients at iteration **t**, **α** is the learning rate, and $\Delta_\ell(\beta^{(t)})$ is the gradient of the log-likelihood at iteration **t**.

Once the optimal coefficients are found, we can use them to make predictions for new inputs. We can apply the logistic function to the linear combination of features, and then compare the probability with a threshold value, usually 0.5, to assign a class label. For example, if the probability is greater than 0.5, we predict 1; otherwise, we predict 0.

To evaluate the performance of the logistic regression model, we can use different metrics, such as accuracy, precision, recall, F1-score, ROC curve, and AUC. These metrics can help us measure how well the model can classify the inputs, and how well it can handle the trade-off between true positives and false positives.

The sklearn library provides a convenient way to implement logistic regression in Python. The LogisticRegression class from the sklearn.linear_model module can be used to create a logistic regression model, fit it to the data, and make predictions. The LogisticRegression class has many arguments that can be used to customize the behavior and performance of the model. Here are some of the most important ones:

- penalty: This argument specifies the type of regularization to apply to the coefficients. Regularization is a technique that adds a penalty term to the log-likelihood function, to prevent overfitting and reduce the variance of the model. The penalty can be either ‘l1’, ‘l2’, ‘elasticnet’, or ‘none’. The default value is ‘l2’, which adds a squared norm of the coefficients to the log-likelihood function. The ‘l1’ penalty adds an absolute norm of the coefficients, which can result in sparse coefficients. The ‘elasticnet’ penalty combines both ‘l1’ and ‘l2’ penalties, with a mixing parameter that controls the balance between them. The ‘none’ penalty does not apply any regularization.
- C: This argument specifies the inverse of the regularization strength. It can be any positive float value. The default value is 1.0. A smaller value of C means a stronger regularization, which can result in smaller and simpler coefficients. A larger value of C means a weaker regularization, which can result in larger and more complex coefficients.
- solver: This argument specifies the algorithm to use for optimizing the log-likelihood function. It can be either ‘newton-cg’, ‘lbfgs’, ‘liblinear’, ‘sag’, ‘saga’, or ‘auto’. The default value is ‘lbfgs’, which is a variant of the gradient ascent algorithm that uses a limited-memory approximation of the Hessian matrix to speed up the convergence. The ‘newton-cg’ solver uses the Newton-Raphson method, which uses the second-order derivative of the log-likelihood function to find the optimal coefficients. The ‘liblinear’ solver is a library that implements a coordinate descent algorithm, which updates one coefficient at a time. The ‘sag’ and ‘saga’ solvers are variants of the stochastic gradient ascent algorithm, which use a random subset of the data at each iteration. The ‘auto’ solver chooses the best solver based on the data and the penalty.
- max_iter: This argument specifies the maximum number of iterations for the solver to converge. It can be any positive integer value. The default value is 100. A higher max_iter may result in a more accurate model, but it may also increase the computation time. A lower max_iter may result in a faster model, but it may also fail to converge.
- multi_class: This argument specifies how to handle multiple classes in the target variable. It can be either ‘auto’, ‘ovr’, or ‘multinomial’. The default value is ‘auto’,

which chooses ‘ovr’ for binary problems and ‘multinomial’ for multiclass problems. The ‘ovr’ option uses the one-vs-rest strategy, which trains a binary classifier for each class and then chooses the class with the highest probability. The ‘multinomial’ option uses the cross-entropy loss function, which directly optimizes the multiclass probability distribution.

Logistic Regression Assumption

Logistic regression is a method that we can use to fit a binary regression model when the response variable is binary. Before fitting a model to a dataset, logistic regression makes the following assumptions:

- **Assumption #1: The Response Variable is Binary**
    - Logistic regression assumes that the response variable only takes on two possible outcomes. Some examples include: Yes or No, Male or Female, Pass or Fail, etc.
    - How to check this assumption: Simply count how many unique outcomes occur in the response variable. If there are more than two possible outcomes, you will need to perform ordinal regression or multinomial regression instead.
- **Assumption #2: The Observations are Independent**
    - Logistic regression assumes that the observations in the dataset are independent of each other. That is, the observations should not come from repeated measurements of the same individual or be related to each other in any way.
    - How to check this assumption: The easiest way to check this assumption is to create a plot of residuals against time (i.e. the order of the observations) and observe whether or not there is a random pattern. If there is not a random pattern, then this assumption may be violated.
- **Assumption #3: There is No Multicollinearity Among Explanatory Variables**
    - Logistic regression assumes that there is no severe multicollinearity among the explanatory variables. Multicollinearity occurs when two or more explanatory variables are highly correlated to each other, such that they do not provide unique or independent information in the regression model. If the degree of correlation is high enough between variables, it can cause problems when fitting and interpreting the model.

    - How to check this assumption: The most common way to detect multicollinearity is by using the variance inflation factor (VIF), which measures the correlation and strength of correlation between the predictor variables in a regression model. Check out this tutorial for an in-depth explanation of how to calculate and interpret VIF values.
- **Assumption #4: There are No Extreme Outliers**
    - Logistic regression assumes that there are no extreme outliers or influential observations in the dataset.
    - How to check this assumption: The most common way to test for extreme outliers and influential observations in a dataset is to calculate Cook's distance for each observation. If there are indeed outliers, you can choose to (1) remove them, (2) replace them with a value like the mean or median, or (3) simply keep them in the model but make a note about this when reporting the regression results.
- **Assumption #5: There is a Linear Relationship Between the Logit of the Outcome and Each Predictor Variables**
    - Logistic regression assumes that there is a linear relationship between the logit of the outcome and each predictor variables. The logit is the natural logarithm of the odds, which is the ratio of the probability of success to the probability of failure. A linear relationship means that the logit of the outcome changes by a constant amount for a unit change in the predictor variable.
    - How to check this assumption: The easiest way to check this assumption is to create a scatter plot of the logit of the outcome against each predictor variable and observe whether or not there is a linear pattern. If there is not a linear pattern, then this assumption may be violated.
- **Assumption #6: The Sample Size is Large Enough**
    - Logistic regression assumes that the sample size is large enough to provide reliable estimates of the model parameters. There is no definitive rule for how large the sample size should be, but a common guideline is that the number of observations should be at least 10 times the number of predictor variables in the model.
    - How to check this assumption: The easiest way to check this assumption is to compare the number of observations with the number of predictor variables in the model and see if they satisfy the guideline. If the sample size is too small, then the model may suffer from overfitting, underfitting, or high variance.

Logistic Regression const function

The cost function of logistic regression is a measure of how well the model fits the data. It quantifies the difference between the predicted probability and the actual output of the model. The goal of logistic regression is to minimize the cost function, which can be done by adjusting the model parameters, such as the coefficients.

The cost function of logistic regression is based on the concept of maximum likelihood estimation, which tries to find the parameter values that make the observed data most likely to occur, under the assumed model. The likelihood is the probability of the data given the model parameters, and it can be written as:

$$L(\beta) = \prod_{i=1}^{n} p(y_i | x_i, \beta)$$

where n is the number of observations, y_i is the output label (0 or 1), x_i is the input vector, and β is the vector of coefficients.

To simplify the calculation, we can take the logarithm of the likelihood, which is called the log-likelihood, and it can be written as:

$$\ell(\beta) = \sum_{i=1}^{n} y_i \log(p(y_i | x_i, \beta)) + (1 - y_i)\log(1 - p(y_i | x_i, \beta))$$

where $p(y_i \mid x_i, \beta)$ is the logistic function applied to the linear combination of features.

The cost function of logistic regression is the negative of the log-likelihood, which can be written as:

$$J(\beta) = -\ell(\beta) = -\sum_{i=1}^{n} y_i \log(p(y_i | x_i, \beta)) + (1 - y_i)\log(1 - p(y_i | x_i, \beta))$$

The cost function of logistic regression has some nice properties, such as:

- It is convex, meaning it has a single global minimum, which can be found using gradient-based methods.
- It is differentiable, meaning it has a derivative, which can be used to calculate the gradient and the direction of the steepest descent.
- It is related to the cross-entropy loss, which measures the divergence between the predicted and the actual probability distributions.

To minimize the cost function of logistic regression, we can use an iterative algorithm, such as gradient descent, which updates the coefficients by moving in the opposite direction of the gradient of the cost function. The gradient is the vector of partial derivatives of the cost function with respect to each coefficient, and it can be written as:

$$\nabla J(\beta) = \begin{bmatrix} \frac{\partial J}{\partial \beta_0} \\ \frac{\partial J}{\partial \beta_1} \\ \vdots \\ \frac{\partial J}{\partial \beta_n} \end{bmatrix} = \sum_{i=1}^{n} (p(y_i|x_i, \beta) - y_i)x_i$$

where x_i is the input vector with a leading 1 for the intercept term.

The gradient descent algorithm starts with an initial guess of the coefficients, and then updates them by subtracting a fraction of the gradient, where the fraction is called the learning rate. The algorithm repeats this process until the cost function converges to a minimum value, or until a maximum number of iterations is reached. The algorithm can be written as:

$$\beta^{(t+1)} = \beta^{(t)} - \alpha \nabla J(\beta^{(t)})$$

where $\beta^{(t)}$ is the vector of coefficients at iteration t, α is the learning rate, and $\Delta J(\beta^{(t)})$ is the gradient of the cost function at iteration t.

Once the optimal coefficients are found, we can use them to make predictions for new inputs. We can apply the logistic function to the linear combination of features, and then compare the probability with a threshold value, usually 0.5, to assign a class label. For example, if the probability is greater than 0.5, we predict 1; otherwise, we predict 0.

To evaluate the performance of the logistic regression model, we can use different metrics, such as accuracy, precision, recall, F1-score, ROC curve, and AUC. These metrics can help us measure how well the model can classify the inputs, and how well it can handle the trade-off between true positives and false positives.

For more information and examples, you can refer to these sources:

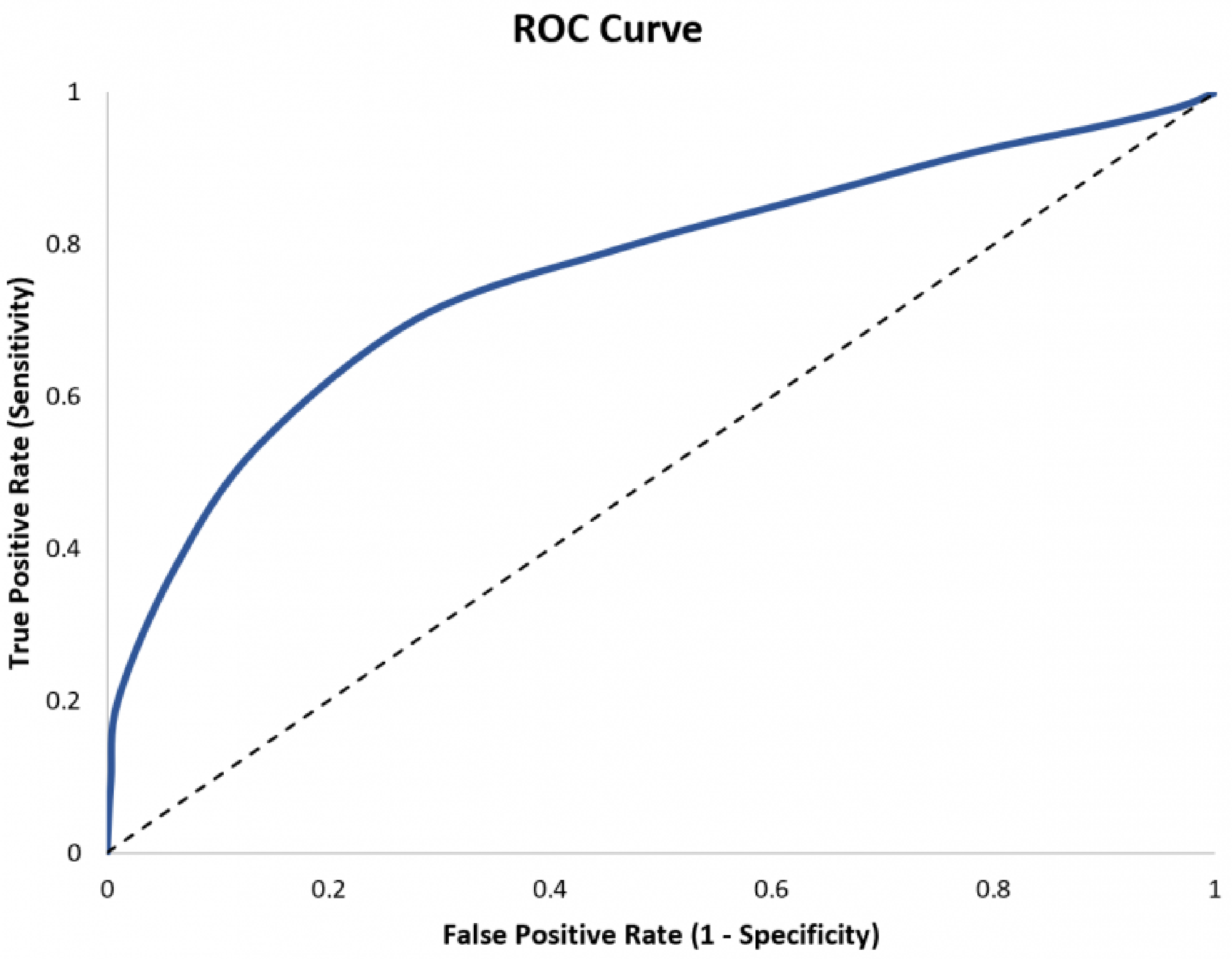
- Cost function in Logistic Regression in Machine Learning: This is a comprehensive article that explains the cost function in logistic regression, with examples and code snippets.
- The cost function in logistic regression: This is a detailed article that covers the cost function and its derivation, with examples and diagrams.
- [Logistic Regression in Machine Learning]: This is a thorough article that covers the logistic regression algorithm and its implementation, with examples and code snippets.

Performance Measurment

ROC Cureve

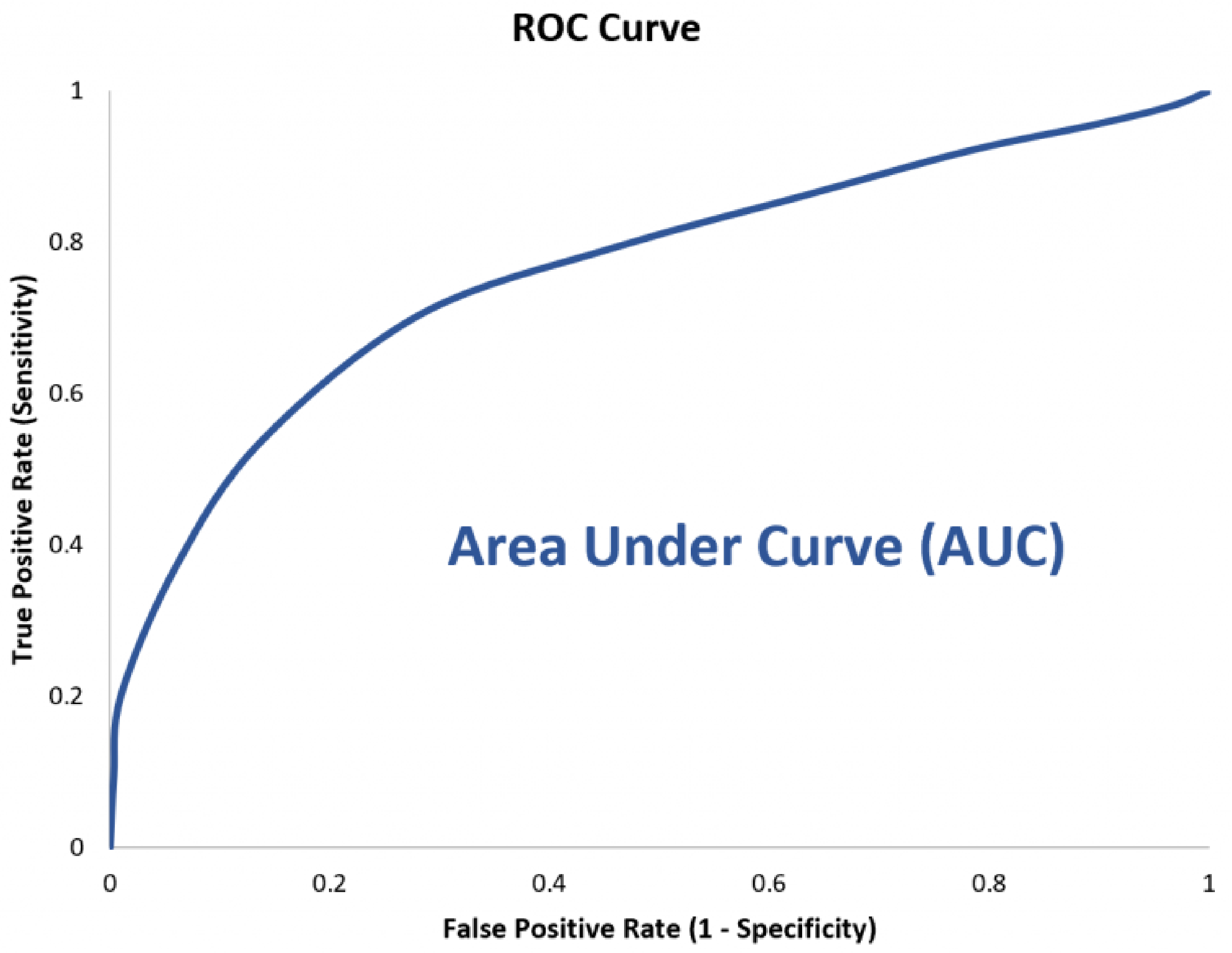
ROC stands for Receiver Operating Characteristic, and the ROC curve is a graphical representation of the performance of a binary classification model at different classification thresholds. A binary classification model is a model that predicts whether an input belongs to one of two classes, such as spam or not spam, positive or negative, etc. A classification threshold is a value that determines how the model assigns a class label based on the predicted probability. For example, if the threshold is 0.5, then the model predicts 1 if the probability is greater than 0.5, and 0 otherwise.

The ROC curve plots the True Positive Rate (TPR) versus the False Positive Rate (FPR) at various classification thresholds. TPR is the proportion of actual positive instances that are correctly predicted as positive, and FPR is the proportion of actual negative instances that are incorrectly predicted as positive. The ROC curve shows how the model balances the trade-off between TPR and FPR as the threshold changes. A perfect model would have a TPR of 1 and an FPR of 0, which means it correctly classifies all the positive and negative instances without any errors. A random model would have a TPR and FPR that are equal, which means it predicts the class label randomly without any information. A typical ROC curve looks like this:



AUC Curve

AUC stands for Area Under the Curve, and the AUC curve is the area under the ROC curve. It measures the overall performance of the binary classification model across all possible classification thresholds. AUC ranges from 0 to 1, where a higher value indicates a better model. AUC can be interpreted as the probability that the model ranks a randomly chosen positive instance higher than a randomly chosen negative instance. For example, if the AUC is 0.8, then it means that the model has an 80% chance of assigning a higher probability to a positive instance than to a negative instance. A perfect model would have an AUC of 1, which means it always ranks the positive instances higher than the negative instances. A random model would have an AUC of 0.5, which means it ranks the positive and negative instances equally. A typical AUC curve looks like this:



ROC and AUC curves are useful for evaluating and comparing the performance of binary classification models. They can help us answer questions such as:

- How well does the model discriminate between the two classes?
- How sensitive is the model to the choice of the classification threshold?
- How does the model handle the trade-off between TPR and FPR?
- How does the model compare to other models or baselines?

Some use cases of ROC and AUC curves are:

- **Medical diagnosis**: ROC and AUC curves can be used to assess the accuracy and reliability of a diagnostic test, such as a blood test, a urine test, or an imaging test, that can detect the presence or absence of a disease or a condition. For example, a ROC curve can show how the test results vary with different cut-off values, and an AUC curve can show how well the test can distinguish between healthy and diseased individuals.
- **Spam detection**: ROC and AUC curves can be used to evaluate the effectiveness of a spam filter, such as an email spam filter, a phone spam filter, or a web spam filter, that can classify messages or calls as spam or not spam. For example, a

ROC curve can show how the filter performs with different spam probabilities, and an AUC curve can show how well the filter can separate spam from legitimate messages or calls.

- **Credit scoring**: ROC and AUC curves can be used to measure the quality and consistency of a credit scoring model, such as a credit rating model, a credit risk model, or a credit default model, that can predict the likelihood of a borrower to repay a loan or a debt. For example, a ROC curve can show how the model predicts the default rate with different score thresholds, and an AUC curve can show how well the model can rank the borrowers by their creditworthiness.