eature is in a pre-release state and might change or have limited support. For more information, see the product launc s (/products/#product-launch-stages).

Training with built-in algorithms on AI Platform allows you to submit your dataset and train a model without writing any training code. This page explains how the built-in XGBoost algorithm works, and how to use it.

The built-in XGBoost algorithm is a wrapper for the XGBoost (https://xgboost.readthedocs.io/en/latest/) algorithm that is compatible to be run on AI Platform.

This document describes a version of the algorithm that runs on a single virtual machine replica. There is also a distributed version of this algorithm (/ml-engine/docs/algorithms/distributed-xgboost) that uses multiple virtual machines for training and requires slightly different usage.

This algorithm has two phases:

1. Preprocessing: AI Platform processes your mix of categorical and numerical data into an all numerical dataset in order to prepare it for training with XGBoost.
2. Training: AI Platform runs training using the XGBoost algorithm based on your dataset and the model parameters you supplied. The current implementation is based on XGBoost's 0.80 version.

The following features are *not* supported for training with the single-replica version of the built-in XGBoost algorithm:

- **Training with GPUs.** To train with GPUs, use the built-in distributed XGBoost algorithm (/ml-engine/docs/algorithms/distributed-xgboost).

- **Distributed training.** To run a distributed training job, use the built-in distributed XGBoost algorithm (/ml-engine/docs/algorithms/distributed-xgboost).

The following AI Platform scale tiers and machine types (/ml-engine/docs/machine-types) are supported:

- BASIC scale tier
- CUSTOM scale tier with any of the Compute Engine machine types supported by AI Platform Training (/ml-engine/docs/machine-types#compute-engine-machine-types).
- CUSTOM scale tier with any of the following legacy machine types (/ml-engine/docs/machine-types#legacy-machine-types):
  - standard
  - large_model
  - complex_model_s
  - complex_model_m
  - complex_model_l

XGBoost works on numerical tabular data. Each row of a dataset represents one *instance*, and each column of a dataset represents a feature value. The *target column* represents the value you want to predict.

Your input data must be a CSV file with UTF-8 encoding. If your training data only consists of categorical and numerical values, then you can use our preprocessing module to convert categorical data to numerical data. Otherwise, you can run training without automatic preprocessing enabled.

You must prepare your input CSV file to meet the following requirements:

- **Remove the header row.** The header row contains the labels for each column. Remove the header row in order to avoid submitting it with the rest of the data instances as part of the training data.

- **Ensure that the target column is the first column.** The target column contains the value that you are trying to predict. For a classification algorithm, all values in the target column are a class or category. For a regression algorithm, all values in the target column are a numerical value.

Columns of integer values are interpreted as categorical columns by default, if there are few enough unique values. For example, if a column in your dataset includes integer values such as {101, 102, 103}, AI Platform interprets these values as categories, such as {'high', 'medium', 'low'}.

To avoid this incorrect analysis, make sure to convert integers to floats when you intend the data to be numerical: {101.0, 102.0, 103.0}. To ensure that integers are interpreted as categorical, append a string before or after each value: {code_101, code_102, code_103}.

For regression training jobs, make sure to normalize your target values so that each value is between 0 and 1.

This section explains how to submit a built-in XGBoost training job.

You can find brief explanations of each hyperparameter within the Google Cloud Console, and a more comprehensive explanation in the reference for the built-in XGBoost algorithm (/ml-engine/docs/algorithms/reference/xgboost).

Automatic preprocessing works for categorical and numerical data. The preprocessing routine first *analyzes* and then *transforms* your data.

First, AI Platform automatically detects the data type of each column, identifies how each column should be treated, and computes some statistics of the data in the column. This information is captured in the `metadata.json` file.

AI Platform analyzes the type of the target column to identify whether the given dataset is for regression or classification. If this analysis conflicts with your selection for the `objective`, it results in an error. Be explicit about how the target column should be treated by formatting your data clearly in ambiguous cases (#handle-integer-values).

- *Type*: The column can be either numerical or categorical.
- *Treatment*: AI Platform identifies how to treat each column as follows:
  - If the column includes a single value in all the rows, it is treated as a **constant**.
  - If the column is categorical, and includes unique values in all the rows, it is treated as a **row_identifier.**
  - If the column is numerical with float values, or if it's numerical with integer values and it contains many unique values, then the column is treated as **numerical**.
  - If the column is numerical with integer values, and it contains few enough unique values, then the column is treated as a categorical column where the integer values are the **identity** or the **vocabulary**.
    - A column is considered to have *few unique values* if the number of unique values in the column is less than 20% of the number of rows in the input dataset.

    - If the column is categorical with high cardinality, then the column is treated with **hashing**, where the number of hash buckets equals to the square root of the number of unique values in the column.
        - A categorical column is considered to have *high cardinality* if the number of unique values is greater than the square root of the number of rows in the dataset.
    - If the column is categorical, and the number of unique values is less than or equal to the square root of the number of rows in the dataset, then the column is treated as a normal categorical column with **vocabulary.**
- *Statistics*: AI Platform computes the following statistics, based on the identified column type and treatment, to be used for transforming the column in a later stage.
    - If the column is numeric, the mean and variance values are computed.
    - If the column is categorical, and the treatment is identity or vocabulary, the distinct values are extracted from the column.
    - If the column is categorical, and the treatment is hashing, the number of hash buckets is computed with respect to the cardinality of the column.

After the initial analysis of the dataset is complete, AI Platform transforms your data based on the types, treatments and statistics applied to your dataset. AI Platform does transformations in the following order:

1. Splits the training dataset into validation and test datasets if you specify the amount of training data to use in each (as a percentage).
2. Removes any rows that have more than 10% of features missing.
3. Fills up missing values. The mean is used for numerical columns, and zeroes are used for categorical columns. See an example below (#examples).
4. For each categorical column with **vocabulary** and **identity** treatment, AI Platform does one-hot encoding on the column values. See an example below (#examples).
5. For each categorical column with **hashing** treatment, AI Platform uses scikit-learn's FeatureHasher (https://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.FeatureHasher.html) to do feature hashing. The number of features counted earlier determines the number of hash buckets.

6. Each column designated with a **row_key** or **constant** treatment is removed.

Rows with 10% of missing values are removed. In the following examples, assume the row has 10 values. Each example row is truncated for simplicity.

Row issue	Original values	Transformed values	Explanation
Example row with no missing values	[3, 0.45, ..., ***'fruits'***, 0, 1]	[3, 0.45, ..., ***1, 0, 0,*** 0, 1]	The string 'fruits' is transformed to the values "1, 0, 0" in one-hot encoding.
Too many missing values	[3, 0.45, ..., 'fruits', __, __]	Row is removed	More than 10% of values in the row are missing.
Missing numerical value	[3, 0.45, ..., ***'fruits'***, 0, __]	[3, 0.45, ..., ***1, 0, 0,*** 0, ***0.54***]	• The mean value for the column replaces the missing numerical value. In this example, the mean is 0.54. • The string 'fruits' is transformed to the values "1, 0, 0" in one-hot encoding.
Missing categorical value	[3, 0.45, ..., __, 0, 1]	[3, 0.45, ..., ***0, 0, 0,*** 0, 1]	• The missing categorical value is transformed to the values "0, 0, 0" in one-hot encoding.

The removal of the rows is applied only to the training data.

After automatic preprocessing is complete, AI Platform uploads your processed dataset back to your Cloud Storage bucket at the directory you specified in the job request.

- Learn more about XGBoost (https://github.com/dmlc/xgboost/tree/master/demo).
- Refer to the built-in XGBoost reference (/ml-engine/docs/algorithms/reference/xgboost) to learn about all the different parameters.