VerticaPy Machine Learning V0.10.1 Cheat Sheet

VerticaPy Machine Learning supports the entire machine learning workflow via a Python interface. For more information about the capabilities of VerticaPy ML, see the VerticaPy ML documentation or check out the VerticaPy examples.

Legend: Grey text describes the function. Highlighted text represents some (and not all) of the optional parameters. Parameters are in purple. Strings are orange.

Preprocessing data

Load data (link)

=> from verticapy.utilities import *

=> import verticapy as vp

=> VDataFrame=vp.read_csv("filename.csv") – Creates a VDataFrame from a csv file.

Summarize data (link)

=> VDataFrame.describe() – Aggregates the vDataFrame using multiple statistical aggregations.

=> VDataFrame.describe(columns=["column_1", "column_2", "column_3", method="categorical"]) – Aggregates the selected columns using categorical statistical aggregations.

Detect Outliers (link) and (link)

=> VDataFrame.outliers_plot(["col1", "col2"]) – A 2D plot to visualize outliers based on the given two columns

=> VDataFrame.outliers(columns=["col1", "col2"], name="name of the outlier columns") – Create a new column which indicates whether a datapoint is an outlier.

Measure correlations (link)

=> VDataFrame.corr(method="pearson") – Calculates and displays the Pearson correlation matrix.

=> VDataFrame.corr(["column_1", "column_2"], method="spearman") – Calculates and displays the Pearson correlation between two columns.

Normalize Data (link)

=> VDataFrame.normalize() – Normalizes all the columns in the dataset using zscore method as default.

=> VDataFrame.normalize(columns=["col1", "col2"], method="minmax") – Normalizes selective columns in the dataset using minimax method as default.

Dimensionality Reduction (link)

=> from verticapy.learn.decomposition import PCA – Importing PCA function.

=> model = PCA("PCA_name") – Make a PCA object.

=> model.fit(VDataFrame) – Apply the PCA on the VDataFrame and display the results of PCA.

=> model.transform(n_components=2) – Create a VDataFrame with columns as the principal components.

Encode Categorical features (label encode link) and (get dummies link)

=> VDataFrame.label_encode() – Encodes a categorical column into numerical values.

=> VDataFrame["column_name"].one_hot_encode() – One Hot Encoding for the desired column.

=> VDataFrame["column_name"].mean_encode() – Mean Encoding for the desired column.

Impute missing values (link)

=> VDataFrame.count_percent() – Counts the percentage of missing values for reach column.

=> VDataFrame["col_to_fill"].fillna(method="auto") – Fills missing values by selecting mean of numeric values and mode for categorical.

=> VDataFrame["col_to_fill"].fillna(method="avg", by =["columns_used_in_parition"]) – Fills missing values using the columns for prediction. This replaces the original column.

Process imbalanced data (link)

=> VDataFrame.balance(column=["column_to_balance"]) – Creates a view with an equal distribution of the input data based on response column. Default method is hybrid.

=> VDataFrame.balance(column=["column_to_balance"], method="under", x=0.5) – Creates a view with a custom distribution of the input data based on response column. Ratio(x) can be changed.

=> VDataFrame["column_to_balance"].topk(k=3) – Returns the count for the values in a column.

Sample data (link)

=> VDataFrame.sample(x=0.2) – The entire table is randomly sampled using the given ratio(x).

=> VDataFrame.sample(n=100) – The entire table is randomly sampled using the number of elements required(n).

=> VDataFrame.sample(x=0.3, method="stratified") – The entire table is randomly sampled using the given ratio(x) and method (random, stratified or systematic).

Training and predicting

Regression – Model Building

Linear Regression (link)

=> from verticapy.learn.linear_model import LinearRegression – Import the Linear Regression function.

=> model = LinearRegression(name="public.Name_of_Model") – Build a Linear Regression model.

Support Vector Machines (SVM) (link)

=> from verticapy.learn.svm import LinearSVR

=> model = LinearSVR(name="Name_of_Model", acceptable_error_margin=0.5) – Build a LinearSVR object using the Vertica SVM (Support Vector Machine) algorithm.

Random Forest (link)

=> from verticapy.learn.ensemble import RandomForestRegressor

=> model = RandomForestRegressor(name="Name_of_Model", n_estimators=20, max_features="auto", max_leaf_nodes=32, sample=0.7, max_depth=3, min_sample_leaf=5, min_info_gain=0.0, nbins=32) – Creates a RandomForestRegressor object using the Vertica Random Forest function on the data.

XGBoost (link)

=> from verticapy.learn.ensemble import XGBoostRegressor

=> model = XGBoostRegressor(name="Name_of_Model", max_ntree=10, max_depth=5, nbins=32, objective="squarederror", split_proposal_method="global", tot=0.001, learning_rate=0.1, min_split_loss=0, weight_reg=0, sample=1) – Creates a XGBoostRegressor object using the Vertica XGBoost algorithm. From all the available options, only name is mandatory.

Autoregression (link)

=> from verticapy.learn.delphi import AutoML

=> model=AutoML(name="Name_of_Model", estimator_type="regressor", cv=3, stepwise=True) – Tests multiple models to find which the ones which maximize the input score.

Classification

Logistic Regression (link)

=> from verticapy.learn.linear_model import LogisitcRegression – Import the Logistic Regression function.

=> mode = LogisticRegression(name="Name_of_Model", penalty= "L2", tol=1e-4, C=1, max_iter=100, solver= "CGD") – Creates a LogisticRegression object using Vertica LOGISTIC_REG function.

Support Vector Machines (SVM) (link)

=> from verticapy.learn.svm import LinearSVC

=> model = LinearSVC(name="Name_of_Model", tol=1e-4, C=1.0, fit_intercept= True, intercept_model="regularized", max_iter=100) – Build a LinearSVC object using the Vertica SVM (Support Vector Machine) algorithm.

Random Forest (link)

=> from verticapy.learn.ensemble import RandomForestClassifier

=> model = RandomForestClassifier(name="Name_of_Model", n_estimators=20, max_features="auto", max_leaf_nodes=32, sample=0.7, max_depth=3, min_sample_leaf=5, min_info_gain=0.0, nbins=32) – Creates a RandomForestRegressor object using the Vertica Random Forest function on the data.

XGBoost (link)

=> from verticapy.learn.ensemble import XGBoostClassifier

=> model = XGBoostClassifier(name="Name_of_Model", max_ntree=10, max_depth=5, nbins=32, objective="squarederror", split_proposal_method="global", tot=0.001, learning_rate=0.1, min_split_loss=0, weight_reg=0, sample=1) – Creates a XGBoostRegressor object using the Vertica XGBoost algorithm. From all the available options, only name is mandatory.

Autoregression (link)

=> from verticapy.learn.delphi import AutoML

=> model=AutoML(name="Name_of_Model", estimator_type="multi", cv=3, stepwise=True) – Tests multiple models to find which the ones which maximize the input score.

Clustering

K-neighbors (link)

=> from verticapy.learn.neighbors import KNeighborsClassifier

=> model= KNeighborsClassifier(name="Name_of_Model", n_neighbors=5, p=2) – Creates a KNeighborsClassifier object by using the k-nearest neighbors algorithm.

K-nearest centroid (link)

=> from verticapy.learn.neighbors import NearestCentroid

=> model= NearestCentroid(name="Name_of_Model", p=2) – Creates a NearestCentroid object by using the k-nearest centroid algorithm.

Fitting, Predicting and Evaluating models

Regression/Classification – Model Prediction

Fitting

=> model.fit("public.Name_of_Model" , ["independent_col_1", "independent_col_2"], "dependent_col") – Fit the model to the given independent inputs and dependent outputs.

Prediction

=> model.predict(VDataFrame, X=["independent_col_1", "independent_col_2"], name="name_of_pred_column") – Predicts and adds those values inside the VDataFrame using the new name of prediction columns.

General Metrics

Link to all (link)

Mean Squared Error	**R-squared**	**aic**	**bic**	**Explained Variance**
=> model.score("mse")	=> model.score("r2")	=> model.score("aic")	=> model.score("bic")	=> model.score("var")
Max error	**R-squared adjusted**	**RMSE**	**Median Absolute Error**	**Mean Absolute Error**
=> model.score("max")	=> model.score("r2a")	=> model.score("rmse")	=> model.score("mae")	=> model.score("mae")

Classification-specific Metrics

Confusion Matrix (link)

=> model.confusion_matrix(pos_label="Label", cutoff=0.33) – Fit the model to the given independent inputs and dependent outputs.

Lift Chart (link)

=> from verticapy.learn.model_selection import lift_chart

=> lift_chart("Response_Column", "Prediction_Probability", VDataFrame) – Draws a lift chart.

ROC Curve (link)

=> model.roc_curve(nbins=12) – Plots the ROC curve.

Managing models

memModel To build models using their attributes (link)

For Linear Regression

```
=> from verticapy.learn.memmodel import memModel
=> model=memModel (model_type="LinearRegression", attributes={ "coefficients": [0.5, 1.2], "intercept": 2}) – Builds a Linear Regression model from its attributes.
=> model.predict_sql (["x1", "x2"]) – Generates the SQL code for deploying the model in Vertica.
```

Generate SQL code (link)

For Linear Regression

```
=> model.to_sql() – Generates the SQL code for deploying the model in Vertica.
```