Vertica Machine Learning V12.0.1 Cheat Sheet

Vertica Machine Learning supports the entire machine learning workflow via an SQL interface. For more information about the capabilities of Vertica ML, see the Vertica ML documentation or the Vertica ML examples repository on GitHub. To download the datasets used in this cheat sheet, you can follow the guidelines in the Vertica documentation or download the data directly from the Vertica ML GitHub repository. For a scikit-like machine learning library that integrates directly with the data in your Vertica database, see VerticaPy.

Preprocessing data

Summarize data

=> SELECT summarize_numcol(hits, salary) OVER() FROM baseball WHERE dob > '1975-7-1'::DATE; -- for numeric columns, displays 'COUNT, 'MEAN, 'STDDEV', 'MIN', 'PERC25', 'MEDIAN', 'PERC75', and 'MAX'

=> SELECT summarize_catcol(team USING PARAMETERS topk = 10) OVER() FROM baseball; --for a categorical column, displays 'CATEGORY', 'COUNT', and 'PECENT'

Detect outliers

=> SELECT detect_outliers('baseball_outliers', 'baseball', 'hr, hits, avg, salary', 'robust_zscore' USING PARAMETERS outlier_threshold=3.0); -- outputs outliers to the 'baseball_outliers' table

=> SELECT iforest('baseball_outliers', 'baseball', 'team, hr, hits, avg, salary' USING PARAMETERS ntree=75, sampling_size=0.7, max_depth=15); --trains an iforest model on specified columns of the 'baseball' dataset

=> SELECT * FROM (SELECT first_name, last_name, apply_iforest(hr, hits, salary USING PARAMETERS model_name='baseball_outliers', contamination=0.1) AS predictions FROM baseball)
AS outliers WHERE predictions.is_anomaly IS true; --applies 'baseball_outliers' to 'baseball' and list the outliers

Measure correlations

=> SELECT corr_matrix(hr, hits, avg, salary) OVER() FROM baseball; --calculates the Pearson Correlation Coefficient between each pair of input columns

Normalize data

=> SELECT normalize('baseball_normz', 'baseball', 'hr, hits', 'zscore'); --outputs a normalized result to view 'baseball_normz'

=> SELECT normalize_fit('baseball_normfitrz', 'baseball', 'hr,hits', 'robust_zscore'); --computes and stores normalization parameters in model 'baseball_normfitrz'

=> SELECT apply_normalize(* USING PARAMETERS model_name = 'baseball_normfitrz') FROM baseball; --applies the normalization parameters from 'baseball_normfitrz' to 'baseball'

=> SELECT reverse_normalize(* USING PARAMETERS model_name = 'baseball_normfitrz') FROM baseball; --reverses the normalization transformation

Dimensionality reduction

=> SELECT pca('world_pca', 'world', '*' USING PARAMETERS exclude_columns='HDI, country'); --computes the principal components of 'world' and saves them in model 'world_pca'

=> CREATE TABLE worldPCA AS SELECT apply_pca(* USING PARAMETERS model_name='world_pca', exclude_columns='HDI, country', key_columns='HDI, country', cutoff=0.99) OVER() FROM world;
--uses the 'world_pca' model to transform the 'world' data such that the principal components account for 99 percent of the variance

=> SELECT apply_inverse_pca(HDI, country, col1, col2 USING PARAMETERS model_name='world_pca', exclude_columns='HDI, Country', key_columns='HDI, country') OVER() FROM worldPCA;
--reverts the apply_pca data transform

=> SELECT svd('world_svd', 'world', '*' USING PARAMETERS exclude_columns='HDI, country'); --computes the SVD decomposition of 'world' and saves the result in the model 'world_svd'

=> CREATE TABLE worldSVD AS SELECT apply_svd(* USING PARAMETERS model_name='world_svd', exclude_columns='HDI, country', key_columns='HDI, country',cutoff=0.99) OVER() FROM world;
-- computes the U matrix of the SVD decomposition of 'world' and stores the result in table 'worldSVD'

=> CREATE TABLE inverse_worldSVD AS SELECT apply_inverse_svd(* USING PARAMETERS model_name='world_svd', exclude_columns='HDI, country', key_columns='HDI, country') OVER() FROM worldSVD;
--transforms the table 'worldSVD' back to the original 'world' data

Encode categorical features

=> SELECT one_hot_encoder_fit('bTeamEncoder', 'baseball', 'team' USING PARAMETERS extra_levels='{"team" : ["Red Sox"]}'); --generates the encoding for column 'team' and stores it in model 'bTeamEncoder'

=> CREATE VIEW baseballEncoded AS SELECT apply_one_hot_encoder(* USING PARAMETERS model_name='bTeamEncoder', drop_first=True, ignore_null=False) FROM baseball;
--generates the encoded columns in view 'baseballEncoded'

Impute missing values

=> SELECT impute ('baseballImputed', 'baseball', 'hits', 'mean' USING PARAMETERS partition_columns='team'); --imputes missing values for 'hits' based on the relevant team's mean 'hits' value

Process imbalanced data

=> SELECT balance ('baseballBalanced', 'baseball', 'team', 'hybrid_sampling'); --returns a view where each team is equally represented

Sample data

=> CREATE TABLE baseball_sample AS SELECT * FROM baseball TABLESAMPLE(25); --creates a new table containing a 25% sample of 'baseball'

Training and predicting

Regression

Linear Regression

=> SELECT linear_reg('linear_reg_faithful', 'faithful_training', 'eruptions', 'waiting' USING PARAMETERS optimizer='bfgs'); --trains a linear regression model that uses 'waiting' time to predict the duration of eruptions

=> SELECT id, predict_linear_reg(waiting USING PARAMETERS model_name='linear_reg_faithful') FROM faithful_testing; --applies model 'linear_reg_faithful' to the 'faithful_testing' returns predicted eruption durations

Support Vector Machines (SVM)

=> SELECT svm_regressor('svm_reg_faithful', 'faithful_training', 'eruptions', 'waiting' USING PARAMETERS error_tolerance=0.1); --trains an SVM regression model to predict duration of eruptions based on 'waiting' time

=> SELECT id, predict_svm_regressor(waiting USING PARAMETERS model_name='svm_reg_faithful') FROM faithful_testing; --applies 'svm_reg_faithful' to the 'faithful_testing' and returns predicted eruption durations

Random Forest

=> SELECT rf_regressor('rf_reg_cars', 'mtcars_train', 'mpg', 'carb, cyl, hp, drat, wt' USING PARAMETERS ntree=100); --trains a random forest model to predict a car's MPG based on a number of the car's attributes

=> SELECT mpg, predict_rf_regressor(carb, cyl, hp, drat, wt USING PARAMETERS model_name='rf_reg_cars') FROM mtcars_test; --applies 'rf_reg_cars' to 'mtcars_test' and returns predicted MPGs

XGBoost

=> SELECT xgb_regressor ('xgb_cars', 'mtcars_train', 'mpg', 'carb, cyl, hp, drat, wt' USING PARAMETERS learning_rate=0.5); --trains an XGBoost model to predict a car's MPG based on several of the car's attributes

=> SELECT mpg, predict_xgb_regressor(carb,cyl,hp,drat,wt USING PARAMETERS model_name='xgb_cars') FROM mtcars_test; --applies 'xgb_cars' to 'mtcars_test' and returns predicted MPGs

Classification

Logistic Regression

=> SELECT logistic_reg('logistic_cars', 'mtcars_train', 'am', 'mpg, cyl, disp, hp, drat, wt, qsec, gear, carb' USING PARAMETERS optimizer='BFGS', regularization='L2');

--trains a logistic regression model to predict whether a car has an automatic or manual transmission

=> SELECT car_model, predict_logistic_reg(mpg, cyl, disp, hp, drat, wt, qsec, gear, carb USING PARAMETERS model_name='logistic_cars') FROM mtcars_test; --returns 'logistic_cars' predictions on the 'mtcars_test' dataset

Support Vector Machines (SVM)

=> SELECT svm_classifier('svm_cars', 'mtcars_train', 'am', 'mpg, cyl, disp, hp, drat, wt, qsec, vs, gear, carb' USING PARAMETERS exclude_columns='hp,drat');

--trains an SVM model to predict whether a car has an automatic or manual transmission

=> SELECT car_model, am, predict_svm_classifier(mpg, cyl, disp, wt, qsec, vs, gear, carb USING PARAMETERS model_name='svm_cars') FROM mtcars_test; --returns 'svm_cars' predictions on the 'mtcars_test' dataset

Naive Bayes

=> SELECT naive_bayes('naive_congress', 'house84_train', 'party', '*' USING PARAMETERS exclude_columns='party, id'); --trains a Naïve Bayes model to predict a Congress member's party association

=> SELECT party, predict_naive_bayes(vote1, vote2, vote3 USING PARAMETERS model_name='naive_congress', type='response') AS predicted_party FROM house84_test; --returns the actual party and the predicted party

=> SELECT predict_naive_bayes_classes(id, vote1, vote2, vote3 USING PARAMETERS model_name='naive_congress', key_columns='id', exclude_columns='id', classes='democrat, republican', match_by_pos='false') OVER() FROM house84_test; --returns the predicted party and the probability for both 'democrat' and 'republican'

Random Forest

=> SELECT rf_classifier('rf_iris', 'iris1', 'species', 'sepal_length, sepal_width, petal_length, petal_width' USING PARAMETERS ntree=100, sampling_size=0.3); --trains a Random Forest model to predict 'species' based on feature measurements

=> SELECT id, predict_rf_classifier(sepal_length, sepal_width, petal_length, petal_width USING PARAMETERS model_name='rf_iris') FROM iris2; --returns the predicted 'species' classifications

=> SELECT predict_rf_classifier_classes(id, sepal_length, sepal_width, petal_length, petal_width USING PARAMETERS model_name='rf_iris', key_columns ='id', exclude_columns='id') OVER () FROM iris2;

--returns the predicted 'species' class and its probability

XGBoost

=> SELECT xgb_classifier('xgb_iris', 'iris1', 'species', 'Sepal_Length, Sepal_Width, Petal_Length, Petal_Width' USING PARAMETERS max_depth=5); --trains an XGBoost model to predict 'species' based on feature measurements

=> SELECT predict_xgb_classifier(Sepal_Length, Sepal_Width, Petal_Length, Petal_Width USING PARAMETERS model_name='xgb_iris') FROM iris2; --returns the predicted 'species' classifications

=> SELECT predict_xgb_classifier_classes(Sepal_Length, Sepal_Width, Petal_Length, Petal_Width USING PARAMETERS model_name='xgb_iris', classes='virginica, versicolor, setosa', probability_normalization='softmax') OVER() FROM iris2; --returns the predicted 'species' and the probability for each 'species' class

Clustering

K-means

=> SELECT kmeans('agar_kmeans', 'agar_dish_1', '*', 5 USING PARAMETERS exclude_columns ='id', output_view='agar_view', key_columns='id'); --trains a K-means model to group microbes into 5 clusters based on x-y coordinates

=> SELECT id, apply_kmeans(x, y USING PARAMETERS model_name='agar_kmeans') FROM agar_dish_2; --applies 'agar_kmeans' to 'agar_dish_2' and assigns each input row to a cluster

Bisecting k-means

=> SELECT bisecting_kmeans('agar_bkmeans', 'agar_dish_1', '*', 5 USING PARAMETERS exclude_columns='id', key_columns='id', output_view='agar_bk_view');
--trains a bisecting k-means model to group microbes into five or fewer clusters based on x-y coordinates
=> SELECT id, apply_bisecting_kmeans(x,y USING PARAMETERS model_name='agar_bkmeans', number_clusters=3) FROM agar_dish_2;--applies 'agar_bkmeans' to 'agar_dish_2' and assigns each row to one of three clusters

Time Series

Autoregression

=> SELECT autoregressor('AR_temp', 'temp_data', 'Temperature', 'time' USING PARAMETERS p=3); --trains an autoregressive model to predict 'Temperature' based on the previous three values
=> SELECT predict_autoregressor(Temperature USING PARAMETERS model_name='AR_temp', npredictions=20) OVER(ORDER BY time) FROM temp_data; --returns 20 predictions beginning at the end of 'temp_data'

Moving-average

=> SELECT moving_average('MA_temp', 'temp_data', 'temperature', 'time' USING PARAMETERS q=3); --trains a moving-average model to predict 'Temperature' based on the errors of the previous three predictions
=> SELECT predict_moving_average(Temperature USING PARAMETERS model_name='MA_temp', npredictions=20) OVER(ORDER BY time) FROM temp_data; --returns 20 predictions beginning at the end of 'temp_data'

Evaluating model performance

Regression metrics

Mean Squared Error

=> SELECT mse(obs, pred) OVER() FROM (SELECT eruptions AS obs, predict_linear_reg(waiting USING PARAMETERS model_name='linear_reg_faithful') AS pred FROM faithful_testing) AS prediction_output;
--returns the mean squared error between the predictions from 'linear_reg_faithful' and the ground truth values in 'faithful_testing'

R-squared

=> SELECT rsquared(obs, pred) OVER() FROM (SELECT eruptions AS obs, predict_linear_reg(waiting USING PARAMETERS model_name='linear_reg_faithful') AS pred FROM faithful_testing) AS prediction_output;
--returns the R-squared value between the predictions from 'linear_reg_faithful' and the ground truth values in 'faithful_testing'

Classification metrics

Confusion Matrix

=> SELECT confusion_matrix(obs::int, pred::int USING PARAMETERS num_classes=2) OVER() FROM (SELECT am AS obs, predict_logistic_reg(mpg, cyl, disp, hp, drat, wt, qsec, gear, carb USING PARAMETERS
model_name='logistic_cars') AS pred FROM mtcars_test) AS prediction_output; --computes a confusion matrix for the 'logistic_cars' model using the predictions and the observed values in 'mtcars_test'

Error Rate

=> SELECT error_rate(obs::int, pred::int USING PARAMETERS num_classes=2) OVER() FROM (SELECT am AS obs, predict_logistic_reg(mpg, cyl, disp, hp, drat, wt, qsec, gear, carb USING PARAMETERS
model_name='logistic_cars', num_classes=2) AS pred FROM mtcars_test) AS prediction_output; --calculates 'logistic_cars' rate of incorrect classifications for each class as well as the total error rate across all classes

Lift Table

=> SELECT lift_table(obs::int, prob USING PARAMETERS num_bins=50) OVER() FROM (SELECT am AS obs, predict_logistic_reg(mpg, cyl, disp, hp, drat, wt, qsec, gear, carb USING PARAMETERS model_name='logistic_cars',
type='probability') AS prob FROM mtcars_test) AS prediction_output; --returns a lift chart for the 'logistic_cars' model on the 'mtcars_test' dataset

ROC

=> SELECT roc(obs::int, prob USING PARAMETERS num_bins=50) OVER() FROM (SELECT am AS obs, predict_logistic_reg(mpg, cyl, disp, hp, drat, wt, qsec, gear, carb USING PARAMETERS model_name='logistic_cars',
type='probability') AS prob FROM mtcars_test) AS prediction_output; --returns a table containing points on an ROC curve for the 'logistic_cars' model and 'mtcars_test' dataset

Cross Validation

=> SELECT cross_validate('svm_cars', 'mtcars_train', 'am', 'mpg,cyl,disp,hp,drat,wt,qsec,vs,gear,carb' USING PARAMETERS cv_fold_count= 5, cv_hyperparams='{"C":[0.1,1,5]}', cv_model_name='svm_cv',
cv_metrics='error_rate'); --performs k-fold cross validation for an SVM classifier on 'mtcars_train' data -- evaluates 15 models (5 folds and 3 values of hyperparameter 'C') and saves the result in 'svm_cv'

PRC

=> SELECT prc(obs::int, prob::float USING PARAMETERS num_bins=50, f1_score=true) OVER() FROM (SELECT am AS obs, predict_logistic_reg(mpg, cyl, disp, hp, drat, wt, qsec, gear, carb USING PARAMETERS
model_name='logistic_cars', type='probability') AS prob FROM mtcars_test) AS prediction_output; --returns a table containing points on a precision recall curve (PRC) for the 'logistic_cars' model and 'mtcars_test' dataset

Decision tree metrics

=> SELECT read_tree(USING PARAMETERS model_name='rf_iris', format='tabular'); --returns information for decision trees in the 'rf_iris' model, such as 'is_leaf' and 'node_depth'
=> SELECT rf_predictor_importance(USING PARAMETERS model_name = 'rf_iris'); --measures the importance of each predictor in the 'rf_iris' model using the Mean Decrease Impurity (MDI) method

Managing models

List models

=> SELECT * FROM models; --returns information about all models in the database

Delete a model

=> DROP MODEL xgb_iris; --removes the model from the database

Upgrade a model

=> SELECT upgrade_model(USING PARAMETERS model_name = 'xgb_cars'); --upgrades 'xgb_cars' to the Vertica version currently in use by the database

Change model owner, schema, and name

=> ALTER MODEL xgb_cars OWNER TO analyst;

=> ALTER MODEL xgb_cars SET SCHEMA public;

=> ALTER MODEL xgb_cars RENAME to xgb_autos;

Change model privileges

=> GRANT USAGE ON MODEL linear_reg_faithful TO analyst; --allows the user 'analyst' to run the 'linear_reg_faithful' model

=> REVOKE USAGE ON MODEL linear_reg_faithful FROM analyst; --removes the usage privileges for 'analyst' on the 'linear_reg_faithful' model

Read model attributes

=> SELECT get_model_summary(USING PARAMETERS model_name='linear_reg_faithful'); --displays a summary of the model 'linear_reg_faithful'

=> SELECT get_model_attribute(USING PARAMETERS model_name='linear_reg_faithful'); --lists all attributes of the model (e.g. the type of regularization used, number of iterations, etc.)

=> SELECT get_model_attribute(USING PARAMETERS model_name='linear_reg_faithful', attr_name='data'); --returns the value of attribute 'data' in the model 'linear_reg_faithful'

Import/export models to other Vertica clusters

=> SELECT export_models('/home/dbadmin/myModels', 'AR_temp'); --exports model 'AR_temp' to directory 'myModels'

=> SELECT import_models('/home/newDir/myModels/*' USING PARAMETERS new_schema='analyst'); --imports all models under 'myModels' to schema 'analyst'

Using external models

--As the following examples are dependent on external models, they will not work out of the box. For more information on using external models in Vertica, see the Vertica documentation or github TensorFlow example.

PMML models

=> SELECT import_models('/data/username/temp/spark_logistic_reg' USING PARAMETERS category='pmml'); --imports a PMML model named 'spark_logistic_reg'

=> SELECT SELECT predict_pmml(* USING PARAMETERS model_name='spark_logistic_reg') AS prediction FROM test_data; --makes predictions on the 'test_data' using the 'spark_logistic_reg' model

=> SELECT export_models('/path/to/export/to', 'spark_logistic_reg' USING PARAMETERS category='pmml'); --exports 'spark_logistic_reg' in PMML format

TensorFlow models

=> SELECT import_models('/path/tf_models/tf_mnist_keras' USING PARAMETERS category='tensorflow'); --imports a TensorFlow model named 'tf_mnist_keras'

=> SELECT prdict_tensorflow(* USING PARAMETERS model_name='tf_mnist_keras') OVER(PARTITION BEST) FROM tf_mnist_test_images; --makes predictions on the 'tf_mnist_test_images' data with the 'tf_mnist_keras' model

=> SELECT export_models('/path/to/export/to', 'tf_mnist_keras' USING PARAMETERS category='tensorflow'); --exports 'tf_mnist_keras' as a frozen graph