Vertica Machine Learning V9.0.0 Cheat Sheet

Vertica Machine Learning supports the whole workflow of machine learning via a SQL interface. To learn the full capability of Vertica ML, go to my.vertica.com/documentation. Example data sets used in the cheat sheet are available on github.com/vertica/Machine-Learning-Examples.

Preprocessing the data

Summarize data

```
=> SELECT summarize_numcoL(hits, salary) OVER() FROM baseball WHERE dob > '1975-7-1'::DATE; -- for each column, display 'COUNT, 'AVG', 'STDDEV', 'MIN', 'PERC25', 'MEDIAN', 'PERC75', and 'MAX'
```

Detect outliers

```
=> SELECT detect_outliers('baseball_outliers', 'baseball', 'hr, hits, avg, salary', 'robust_zscore' USING PARAMETERS outlier_threshold=3.0); -- outliers are stored in table 'baseball_outliers'
```

Normalize

```
=> SELECT normalize('baseball_normz', 'baseball', 'hr, hits', 'zscore'); --output normalized result to view 'baseball_normz'
=> SELECT normalize_fit('baseball_normfitrz', 'baseball', 'hr,hits', 'robust_zscore'); --store normalization parameters in a model 'baseball_normfitrz'
=> SELECT apply_normalize(* USING PARAMETERS model_name = 'baseball_normfitrz') FROM baseball; --apply the normalization parameters to 'baseball'
=> SELECT reverse_normalize(* USING PARAMETERS model_name = 'baseball_normfitrz') FROM baseball; --reverse normalization in 'baseball'
```

Encode categorical features

```
=> SELECT one_hot_encoder_fit('bTeamEncoder', 'baseball', 'team' USING PARAMETERS extra_levels='{"team" : ["Red Sox"]}');
   --generate the encoding for column 'team' and store them 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;
    --generate the encoded columns in view 'baseballEncoded'
```

Impute missing values

```
=> SELECT impute ('baseballImputed', 'baseball', 'hits', 'mean' USING PARAMETERS partition_columns='team'); --impute the missing values for 'hits' using the mean value for each team separately
```

Process imbalance data

```
=> SELECT balance ('baseballBalanced', 'baseball', 'team', 'hybrid_sampling'); --make the sample size even across all teams
```

Sample

```
=> CREATE TABLE baseball_sample AS SELECT * FROM baseball TABLESAMPLE(25); --generate a 25% sample set randomly
```

Training and predicting

Regression

Linear Regression

```
=> SELECT linear_reg('myLinearRegModel', 'faithful', 'eruptions', 'waiting' USING PARAMETERS optimizer='cgd', regularization='L1'); -- train a model that uses 'waiting' time to predict the duration of the next eruption
=> SELECT id, predict_linear_reg(waiting USING PARAMETERS model_name='myLinearRegModel') FROM faithful_test; --run the prediction
```

Support Vector Machines (SVM)

```
=> SELECT svm_regressor('mySvmRegModel', 'faithful_train', 'eruptions', 'waiting' USING PARAMETERS error_tolerance=0.1, max_iterations=100);
=> SELECT id, predict_svm_regressor(waiting USING PARAMETERS model_name='mySvmRegModel') FROM faithful_test;
```

Classification

Logistic Regression

```
=> SELECT logistic_reg('myLogisticRegModel', 'mtcars_train', 'am','mpg, cyl, disp, hp, drat, wt, qsec, vs, gear, carb' USING PARAMETERS exclude_columns='hp', optimizer='BFGS', regularization='L2');
    --train a model to predict if a car has automatic or manual transmission
=> SELECT car_model, predict_logistic_reg(mpg, cyl, disp, drat, wt, qsec, vs, gear, carb USING PARAMETERS model_name='myLogisticRegModel') FROM mtcars_test;
```

Support Vector Machines (SVM)

```
=> SELECT svm_classifier('mySvmClassModel', 'mtcars_train', 'am', 'mpg,cyl,disp,hp,drat,wt,qsec,vs,gear,carb' USING PARAMETERS exclude_columns='hp,drat');
=> SELECT car_model, predict_svm_classifier(mpg,cyl,disp,wt,qsec,vs,gear,carb USING PARAMETERS model_name='mySvmClassModel') FROM mtcars_test;
```

Naive Bayes

```
=> SELECT naive_bayes('naive_house84_model', 'house84_train', 'party', '*' USING PARAMETERS exclude_columns='party, id'); --train a model to predict a person's party association
=> SELECT party, predict_naive_bayes(vote1, vote2, vote3 USING PARAMETERS model_name='naive_house84_model', type='response') AS predicted_party FROM house84_test;
=> SELECT predict_naive_bayes_classes(id, vote1, vote2, vote3 USING PARAMETERS model_name='naive_house84_model', key_columns='id', exclude_columns='id', classes='democrat, republican',
   match_by_pos='false') OVER() FROM house84_test; --return the probability of the predicted class and the specified class 'democrat' and 'republican'
```

Random Forest

```
=> SELECT rf_classifier('myRFModel', 'iris_train', 'species', 'sepal_length, sepal_width, petal_length, petal_width' USING PARAMETERS ntree=100, sampling_size=0.3);
=> SELECT id, predict_rf_classifier(sepal_length, sepal_width, petal_length, petal_width USING PARAMETERS model_name='myRFModel') FROM iris_test;
=> SELECT predict_rf_classifier_classes(id, sepal_length, sepal_width, petal_length, petal_width USING PARAMETERS model_name='myRFModel', key_columns ='id', exclude_columns='id') OVER () FROM iris_test;
   --return the probability of the predicted class
```

Clustering

K-means

```
=> SELECT kmeans('myKmeansModel', 'iris', '*', 5  USING PARAMETERS max_iterations=20, key_columns='id', exclude_columns='species, id'); --cluster iris records into groups
=> SELECT id, apply_kmeans(sepal_length, 2.2, 1.3, petal_width USING PARAMETERS model_name='myKmeansModel', match_by_pos='true') FROM iris;
```

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='myLinearRegModel') AS pred FROM faithful_testing) AS prediction_output;
```

R Squared

```
=>SELECT rsquared(obs, pred) OVER() FROM (SELECT eruptions AS obs, PREDICT_LINEAR_REG (waiting USING PARAMETERS model_name='myLinearRegModel') AS pred FROM faithful_testing) AS prediction_output;
```

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, drat, wt, qsec, vs, gear, carb
   USING PARAMETERS model_name='myLogisticRegModel')::INT AS pred FROM mtcars) AS prediction_output;
```

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, drat, wt, qsec, vs, gear, carb
   USING PARAMETERS model_name='myLogisticRegModel', type='response') AS pred FROM mtcars) AS prediction_output;
```

Lift Table

```
=> SELECT lift_table(obs::int, prob USING PARAMETERS num_bins=2) OVER() FROM (SELECT am AS obs, predict_logistic_reg(mpg, cyl, disp, drat, wt, qsec, vs, gear, carb
   USING PARAMETERS model_name='myLogisticRegModel', type='probability') AS prob FROM mtcars) AS prediction_output;
```

ROC

```
=> SELECT roc(obs::int, prob USING PARAMETERS num_bins=2) OVER() FROM (SELECT am AS obs, predict_logistic_reg(mpg, cyl, disp, drat, wt, qsec, vs, gear, carb
   USING PARAMETERS model_name='myLogisticRegModel', type='probability') AS prob FROM mtcars) AS prediction_output;
```

Cross Validation

```
=> SELECT cross_validate('svm_classifier', 'mtcars', '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');  --15 models (5 folds and 3 values of hyper parameter 'C') will be evaluated, and the result is stored in 'svm_cv'
```

Managing models

List models

```
=> SELECT * FROM models;
```

Delete a model

```
=> DROP MODEL myLinearRegModel;
```

Change model name, owner and schema of a model

```
=> ALTER MODEL myKmeansModel OWNER TO user1;
=> ALTER MODEL myKmeansModel SET SCHEMA public;
=> ALTER MODEL myKmeansModel RENAME to myKmeans;
```

Read model attributes

```
=> SELECT get_model_summary(USING PARAMETERS model_name='myLinearRegModel'); --display a summary about the model
=> SELECT get_model_attribute(USING PARAMETERS model_name='myLinearRegModel'); --list all attributes in the model
=> SELECT get_model_attribute(USING PARAMETERS model_name='myLinearRegModel', attr_name='details'); --return the value for attribute 'details'
```

Import/export models to other Vertica clusters

```
=> SELECT export_models('/home/dbadmin/myModels', 'myKmeansModel'); --export model 'myKmeansModel' to directory 'myModels'
=> SELECT import_models('/home/newDir/myModels/*' USING PARAMETERS new_schema='user1'); --import all models under 'myModels' to schema 'user1'
```