Vertica ML Python Workshop

Exercise 11: ML Classification

Ouali Badr

December 6, 2019

Executive Summary

> *"Science knows no country, because knowledge belongs to humanity, and is the torch which illuminates the world."*
>
> **Louis Pasteur**

VERTICA ML PYTHON allows the users to use Vertica advanced analytics and Machine Learning with a Python front-end Interface. In this exercise, you'll learn some basics to begin your fantastic Data Science Journey with the API. As a summary:

- Split the data
- Build a Logistic Regression model
- Evaluate the model with different metrics
- Compute the model features importance

Contents

1 Presentation

Classification is the problem of identifying to which of a set of categories (sub-populations) a new observation belongs, on the basis of a training set of data containing observations (or instances) whose category membership is known. Vertica ML Python has many classification algorithms already implemented (Logistic Regression, Random Forest, SVM...). We will use a Logistic Regression to classify the survival of the Titanic passengers.

2 Functions used during the Exercise

2.1 train_test_split

Library: vertica_ml_python.learn.model_selection

```
train_test_split(input_relation: str, cursor, test_size: float = 0.33)
```

Build one table and 2 views which can be used to evaluate a model. The table will include all the main relation information with a test column (boolean) which represents if the data belong to the test or train set.

Parameters

- **input_relation:** *<str>*
  The relation used to test the estimator.
- **cursor:** *<object>*
  A DB cursor.
- **test_size:** *<float>*
  Proportion of the test set comparint to the training set.

Returns

A tuple (name of the train view, name of the test view)

2.2 LogisticRegression

Create a LogisticRegression object by using the Vertica Highly Distributed and Scalable Logistic Regression on the data.

initialization

Library: vertica_ml_python.learn.linear_model

```
class LogisticRegression(
    name: str,
    cursor,
    penalty: str = ’L2’,
    tol: float = 1e-4,
    C: int = 1,
    max_iter: int = 100,
    solver: str = ’CGD’,
    l1_ratio: float = 0.5)
```

- **name:** *<str>*
  Name of the model.

- **cursor:** *<object>*
  DB cursor.

- **penalty:** *<str>*, optional
  Determines the method of regularization: {None | L1 | L2 | ENet}

- **tol:** *<float>*, optional
  Determines whether the algorithm has reached the specified accuracy result.

- **C:** *<int>*, optional
  The regularization parameter value. The value must be zero or non-negative.

- **max_iter:** *<int>*, optional
  Determines the maximum number of iterations the algorithm performs before achieving the specified accuracy result.

- **solver:** *<int>*, optional
  The optimizer method used to train the model: {Newton | BFGS | CGD}

- **l1_ratio:** *<float>*, optional
  ENet mixture parameter that defines how much L1 versus L2 regularization to provide.

Methods

The LogisticRegression object has many methods:

```python
# Add the LogisticRegression prediction in a vDataframe
def add_to_vdf(self, vdf, name: str = "", cutoff: float = 0.5)

# Compute different metrics to evaluate the model
def classification_report(self, cutoff: float = 0.5)

# Draw the confusion matrix of the model
def confusion_matrix(self, cutoff: float = 0.5)

# Save a table or a view in the DB corresponding to the model predictions for all the classes
def deploy_to_DB(self, name: str, view: bool = True, cutoff = -1)

# Drop the model from the DB
def drop(self)

# Compute the importance of each feature
def features_importance(self)

# Fit the model with the input columns
def fit(self, input_relation: str, X: list, y: str, test_relation: str = "")

# Draw the Lift Chart
def lift_chart(self)
```

```
# Plot the LogisticRegression if it is possible (The length of X must be lesser of equal to 2)
def plot(self)

# Draw the PRC Curve
def prc_curve(self)

# Draw the ROC Curve
def roc_curve(self)

# Compute the selected metric
def score(self, cutoff: float = 0.5, method: str = "accuracy")
```

Attributes

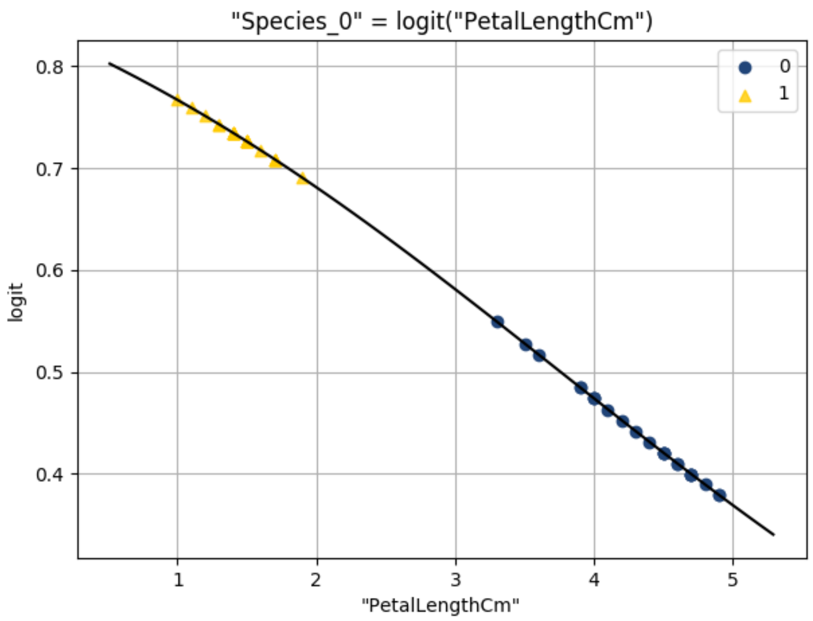
The LogisticRegression object has only one attribute:

```
self.coef # Informations about the model coefficients
```

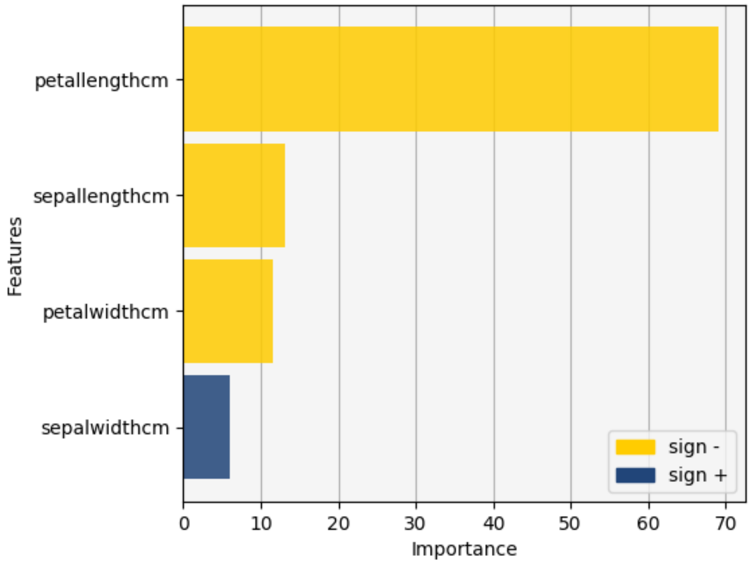
Example

```
from vertica_ml_python import vDataframe
from vertica_ml_python.learn.linear_model import LogisticRegression

# We create dummies
iris = vDataframe("iris", cur)
iris["Species"].get_dummies(use_numbers_as_suffix = True)
iris.to_db("iris_dummy")

# We can build the model
model = LogisticRegression("logit_iris", cur)
model.fit("iris_dummy", ["PetalLengthCm"], "Species_0")

# We can evaluate the model
model.classification_report()

# Output
                               value
auc                              1.0
prc_auc           0.9800000000000001
accuracy           0.953333333333333
log_loss           0.187846851053108
precision                        1.0
recall            0.8771929824561403
f1-score          0.9345794392523363
mcc               0.9032106474595007
informedness      0.8771929824561404
markedness        0.9300000000000002
csi               0.8771929824561403
```

```

# We can also draw the model
model.plot()
```



```
# We can build a new model
model.drop()
model.fit("iris_dummy", ["PetalLengthCm", "PetalWidthCm",  "SepalLengthCm", "SepalWidthCm"], "Species_0")

# We can see the features importance
model.features_importance()
```



3 Questions

Turn on Jupyter with the 'jupyter notebook' command. Start the notebook exercise11.ipynb and answer to the following questions.

- **Question 1:** Split the dataset into a training and a testing.
- **Question 2:** Create a Logistic Regression model.
- **Question 3:** Look at the model coef attribute and see what features you should eliminate if you decide to build another Logistic Regression model.
- **Question 4:** Look at the features importance and confirm the hypothesis.
- **Question 5:** Draw the ROC Curve, PRC Curve and compute a classification report. What can you say about your model ? How can you solve this problem using the same features ?