Vertica ML Python Workshop

Exercise 5: Encoding

Ouali Badr

December 5, 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:

- Encode data using different methods
- Learn when to use the suitable encoding method

Contents

1 Presentation

When dealing with categorical features, many Machine Learning Algorithms will fail because they can not process non-numerical features. Besides, even the ones which could possibly handle them will fail when a feature has too many categories. Non-numerical features are a real problem as we can not also apply many different mathematical functions on them. We are loosing useful information. Encoding is the solution to strike this problem. We can identify 4 main methods to encode the data:

- **User-Defined Encoding:** The most flexible encoding. The user choose to encode the different categories the way he/she wants. It can be used when we deal with too many categories and some are more important than others. Merging the rare categories together and keeping the relevant ones is then a suitable solution.
- **Label Encoding:** Each category is converted to an integer using a bijection to $[0; n-1]$ where n is the feature number of unique values. It can be used to not loose any information on the feature and to not create new features. It can not be used in case of linear ML algorithms.
- **One Hot Encoding:** It creates dummies (values in $0, 1$) of each categories. All the categories are then separated into n features. It can be used when the number of categories is small and you want to proceed ML on it. It will explain each category individually to help you excluding the useless ones.
- **Mean Encoding:** It uses the frequencies of each category regarding a specific response column. It can be used when the number of categories is very big and you want to process ML on all the categories using a specific response column. Be careful, this type of encoding can lead to over-fitting.

During this exercise, you'll use the Titanic dataset to apply in the non-numerical columns different types of encoding.

2 Functions used during the Exercise

2.1 get_dummies

Library: vertica_ml_python.vDataframe[]

```
vDataframe[].get_dummies(
            self,
            prefix: str = "",
            prefix_sep: str = "_",
            drop_first: bool = False,
            use_numbers_as_suffix: bool = False)
```

Compute the different columns dummies and add them to the vDataframe.

Parameters

- **prefix:** *<str>*, optional
  Prefix of the dummies.
- **prefix_sep:** *<str>*, optional
  Prefix delimitor of the dummies.
- **drop_first:** *<bool>*, optional
  Drop the first dummy to avoid the creation of correlated features.

- **use_numbers_as_suffix:** *<bool>*, optional
  Use numbers as suffix instead of the different categories.

Returns

The parent vDataframe.

Example

```
from vertica_ml_python.learn.datasets import load_titanic
titanic = load_titanic(cur)
titanic = titanic.select(["embarked"])

#Output
     embarked
0           S
1           S
2           S
3           S
4           C
...       ...
Name: embarked, Number of rows: 1234, dtype: varchar(20)

titanic["embarked"].get_dummies()

#Output
     embarked    embarked_C    embarked_Q    embarked_S
0           S             0             0             1
1           S             0             0             1
2           S             0             0             1
3           S             0             0             1
4           C             1             0             0
...       ...           ...           ...           ...
Name: titanic, Number of rows: 1234, Number of columns: 4
```

2.2 decode

Library: vertica_ml_python.vDataframe[]

```
vDataframe[].decode(self, values: dict, others = None)
```

Encode the data using the input bijection.

Parameters

- **values:** *<dict>*
  Dictionary of values representing the bijection used to encode the data.

- **others:** *<float>*, optional
  How to encode the values which are not in the dictionary of values.

Returns

The parent vDataframe.

Example

```
from vertica_ml_python.learn.datasets import load_titanic
titanic = load_titanic(cur)
titanic["sex"]

#Output
         sex
0     female
1       male
2     female
3       male
4       male
...      ...
Name: sex, Number of rows: 1234, dtype: varchar(20)

titanic["sex"].decode(values = {"male": 1, "female": 0})

#Output
      sex
0       0
1       1
2       0
3       1
4       1
...   ...
Name: sex, Number of rows: 1234, dtype: int
```

2.3 label_encode

Library: vertica_ml_python.vDataframe[]

```
vDataframe[].label_encode(self)
```

Encode the column using a bijection from the different categories to [0, n - 1] (n being the number of elements).

Returns

The parent vDataframe.

Example

```
from vertica_ml_python.learn.datasets import load_titanic
titanic = load_titanic(cur)
titanic = titanic.select(["embarked"])

#Output
     embarked
0            S
1            S
2            S
3            S
4            C
...        ...
Name: embarked, Number of rows: 1234, dtype: varchar(20)

titanic["embarked"].label_encode()

#Output
     embarked
0            2
1            2
2            2
3            2
4            0
...        ...
Name: embarked, Number of rows: 1234, dtype: int
```

2.4 mean_encode

Library: vertica_ml_python.vDataframe[]

```
vDataframe[].mean_encode(self, response_column: str)
```

Encode the column using the average of the response column for the different categories.

Parameters

- **response_column:** *<str>*
  The response column.

Returns

The parent vDataframe.

Example

```
from vertica_ml_python.learn.datasets import load_titanic
titanic = load_titanic(cur)
titanic = titanic.select(["embarked", "survived"])

#Output
     survived    embarked
0            0           S
1            0           S
2            0           S
3            0           S
4            0           C
...        ...         ...
Name: titanic, Number of rows: 1234, Number of columns: 2

titanic["embarked"].mean_encode(response_column = "survived")

#Output
     survived             embarked
0            1    0.537549407114625
1            1    0.537549407114625
2            1    0.537549407114625
3            1    0.537549407114625
4            1    0.537549407114625
...        ...                  ...
Name: titanic, Number of rows: 1234, Number of columns: 2
```

3 Questions

Turn on Jupyter with the 'jupyter notebook' command. Start the notebook exercise5.ipynb and answer to the following questions.

- **Question 1:** The feature 'sex' has two non-numerical categories. Use a label encoding to encode it.
- **Question 2:** The feature 'embarked' has 3 categories. Why is it more judicious to use a One Hot Encoding to encode it? Encode this feature.
- **Question 3:** The feature 'name' which now represents the title of the passengers. Describe it and find the most occurrent categories.
- **Question 4:** Machine Learning doesn't like too many categories. Encode the data by combining all the rare categories together.
- **Question 5:** As we need numerical values, mean encoding can be a way to encode the result. We want to predict the passengers survival. The response is then the feature 'survived'. Use a mean encoding to encode the feature obtained in the previous question.