How to Write A Recipe, and a TimeSeries Recipe?

Automating Feature Engineering Using DriverlessAI

Ashrith Barthur

H_2O.ai

October 8, 2019

Question

1. How many of us have built variables, features, transformers, or feature transformers?
2. What are they?

Answer

1. Variables, features, transformers, feature transformers all refer to the same.
2. Each column in your data is considered a variable(*incoming*) or a feature(*incoming*).
3. Each *new* column created is also referred to as a variable or a feature.
4. The process of creating a new variable, or a feature is called a transformation.
5. The code processing an *existing* column to a *new* column is called a *transformer*.

Example Transformation

1. *height*- Variable
2. New variable after transformation *log2(height)*

Question

1. How many of us are familiar with Custom Transformers in Driverless AI?
2. What are they?

Answer

1. DriverlessAI already has a large, comprehensive set of transformers.
2. But there are always domains that require nuanced features.
3. And for this, DriverlessAI provides us to create custom transformers.
4. This is provided by provisioning an extension class *CustomTransformer*

How Did We Build A Custom Transformer?

Driverless AI provides an extension.
This is a class ‘CustomTransformer‘

```
class ExampleLogTransformer(CustomTransformer):
```

How Did We Build This?

The class has:

1. Parameters that need to be provided.
2. These parameters are specific to the type of feature recipe that you are building.
3. It also has four methods which primarily handle your feature engineering transformation.

Parameters - Basic

```
class ExampleLogTransformer(CustomTransformer):
    _regression = True
    _binary = True
    _multiclass = True
```

Parameters - Advanced

```
class ExampleLogTransformer(CustomTransformer):
    _regression = True
    _binary = True
    _multiclass = True
    _numeric_output = True
    _is_reproducible = True
    _excluded_model_classes = [‘tensorflow’]
    _modules_needed_by_name = ["custom_package==1.0.0"]
```

Acceptance Method

```
class ExampleLogTransformer(CustomTransformer):
    _regression = True
    _binary = True
    _multiclass = True
    _numeric_output = True
    _is_reproducible = True
    _excluded_model_classes = [‘tensorflow’]
    _modules_needed_by_name = ["custom_package==1.0.0"]

    @staticmethod
    def do_acceptance_test():
        return True
```

Input Data

```
...
@staticmethod
def do_acceptance_test():
    return True

@staticmethod
def get_default_properties():
    return dict(col_type = "numeric"
        ,min_cols = 1, max_cols = 1,
        relative_importance = 1)
```

Input Data Types

```
a. "all"    - all column types
b. "any"    - any column types
c. "numeric"    - numeric int/float column
d. "categorical" - string/int/float column considered a categorical
                    for feature engineering
e. "numcat" - allow both numeric or categorical
f. "datetime"    - string or int column with raw datetime such as
                    '%Y/%m/%d %H:%M:%S' or '%Y%m%d%H%M'
```

H2O.ai

Input Data Types

```
g. "date"   - string or int column with raw date such as
              '%Y/%m/%d' or '%Y%m%d'
h. "text"   - string column containing text
               (and hence not treated as categorical)
i. "time_column" - the time column specified at the start of
                   the experiment (unmodified)
```

H2O.ai

Fit Function

```
@staticmethod
def get_default_properties():
    return dict(col_type = "numeric"
        ,min_cols = 1, max_cols = 1,
        relative_importance = 1)

def fit_transform(self, X: dt.Frame, y: np.array = None):
    X_pandas = X.to_pandas()
    X_p_log = np.log10(X_pandas)
    return X_p_log
```

Transform Function

```
def fit_transform(self, X: dt.Frame, y: np.array = None):
    X_pandas = X.to_pandas()
    X_p_log = np.log10(X_pandas)
    return X_p_log

def transform(self, X: dt.Frame):
    X_pandas = X.to_pandas()
    X_p_log = np.log10(X_pandas)
    return X_p_log
```

Library

```
from h2oaicore.systemutils import segfault,
    ,loggerinfo, main_logger
from h2oaicore.transformer_utils
    import CustomTransformer
import datatable as dt
import numpy as np
import pandas as pd
import logging
```

Time Series Introduction Auto Arima

1. In our example we will bring in the *auto_arima* function as a part of the recipe.
2. This is available in the *pmdarima* package available for *Python*.
3. The *auto_arima* function tries different ‘p‘, ‘q‘, and ‘d‘ values for *ARIMA*, automatically.
4. It selects the best values based on the lowest value in the information criterion.

Recap on ARIMA

What is ARIMA?
ARIMA stands for Auto Regressive Integrated Moving Average

Recap on ARIMA

Auto Regressive means the target depends on its own lags:

1. $Y_t = \alpha + \beta_1 Y_{t-1} + ... + \beta_p Y_{t-p} + \epsilon_t$

Integrated means that the target has been differenced to make the time series stationary:

1. First order differencing: $Y_t^{d1} = Y_t - Y_{t-1}$
2. Second order differencing: $Y_t^{d2} = Y_t^{d1} - Y_{t-1}^{d1}$

Moving Average means the target depends on previous prediction errors:

1. $Y_t = \alpha + \phi_1 \epsilon_{t-1} + ... + \beta_q \epsilon_{t-q}$

TimeSeries Recipe Basics

1. There is a custom class for creating TimeSeries recipes *CustomTimeSeriesTransformer*.
2. Similar to *CustomTransformer*, *CustomTimeSeriesTransformer* has pre-defined parameters and functions.

Generic Recipe Parameters

```
class MyAutoArimaTransformer(CustomTimeSeriesTransformer):
    _binary = False
    _multiclass = False
    _modules_needed_by_name = [‘pmdarima’]
    _included_model_classes = None
```

TimeSeries Recipe Specific Parameters

```
self.tgc = kwargs['tgc']
self.target = kwargs['target']
if isinstance(kwargs['time_column'],list):
    self.time_column = kwargs['time_column'][0]
else:
    self.time_column kwargs['time_column']
```

TimeSeries Recipe Specific Parameters

There are three parameters primarily required by CustomTimeSeries class.

1. `self.tgc` Time series groups
2. `self.target` The target column
3. `self.time_column` The column that holds time.

TimeSeries Recipe Class

The *CustomTimeSeriesTransformer* class shares the basic, four methods of *CustomTransformer* Class. These are methods that DriverlessAI invokes while running custom recipes.

1. `do_acceptance_test`
2. `get_default_properties`
3. `fit_transform`
4. `transform`

TimeSeries Recipe Class

Additionally, there are two other functions that are invokable in *CustomTimeSeriesTransformer* class. They are:

1. `fit` builds the model to which the data will fit.
2. `update_history` updates the model fit with additional data.

Acceptance and Properties Methods

```
@staticmethod
def do_acceptance_test():
return False

@staticmethod
def get_default_properties():
    return dict(col_type="time_column"
        ,min_cols=1, max_cols=1,
        relative_importance=1)
```

Building the Fit Function

```
def fit(self, X: dt.Frame, y: np.array = None):
    pm = importlib.import_module('pmdarima')
    self.models = {}
    X = X.to_pandas()
    XX = X[self.tgc].copy()
    XX['y'] = np.array(y)
    self.nan_value = np.mean(y)
    elf.ntrain = X.shape[0]
```

Building the Fit Function

```
tgc_wo_time = list(np.setdiff1d(self.tgc, self.time_column))
if len(tgc_wo_time) > 0:
    XX_grp = XX.groupby(tgc_wo_time)
else:
    XX_grp = [([None], XX)]
```

Building the Fit Function

```
nb_groups = len(XX_grp)
for _i_g, (key, X) in enumerate(XX_grp):
    key = key if isinstance(key, list) else [key]
    grp_hash = ‘_’.join(map(str, key))
    order = np.argsort(X[self.time_column])
    try:
        model = pm.auto_arima(X[‘y’].values[order]
                ,error_action=‘ignore’)
        except:
        model = None
    self.models[grp_hash] = model
    return self
```

Building the Transform Function

```
nb_groups = len(XX_grp)
preds = []
for _i_g, (key, X) in enumerate(XX_grp):
    key = key if isinstance(key, list) else [key]
    grp_hash = '_'.join(map(str, key))
    order = np.argsort(X[self.time_column])
```

Building the Transform Function

```
if grp_hash in self.models:
model = self.models[grp_hash]
if model is not None:
    if hasattr(self, ‘is_train’):
        yhat = model.predict_in_sample()
    else:
        model.predict(n_periods=X.shape[0])
    yhat = yhat[order]
    XX = pd.DataFrame(yhat, columns=[‘yhat’])
 else:
    XX = pd.DataFrame(np.full((X.shape[0], 1), self.nan_value)
         ,columns=[‘yhat’])  # invalid model
           ...
```

Building the Transform Function

```
                    ...
else:
    XX = pd.DataFrame(np.full((X.shape[0], 1), self.nan_value),
    columns=['yhat'])  # unseen groups
    XX.index = X.index
    preds.append(XX)
XX = pd.concat(tuple(preds), axis=0).sort_index()
return XX
```

Building the Fit Transform Function

```
def fit_transform(self, X: dt.Frame, y: np.array = None):
    self.is_train = True
    ret = self.fit(X, y).transform(X)
    del self.is_train
    return ret
```

Building the Update History Function

```
def update_history(self, X: dt.Frame, y: np.array = None):
    X = X.to_pandas()
    XX = X[self.tgc].copy()
    XX['y'] = np.array(y)
    tgc_wo_time = list(np.setdiff1d(self.tgc, self.time_column))
    if len(tgc_wo_time) > 0:
       XX_grp = XX.groupby(tgc_wo_time)
    else:
       XX_grp = [([None], XX)]
```

Building the Update History Function

```
for key, X in XX_grp:
    key = key if isinstance(key, list) else [key]
    grp_hash = '_'.join(map(str, key))
    order = np.argsort(X[self.time_column])
    if grp_hash in self.models:
        model = self.models[grp_hash]
         if model is not None:
            model.update(X['y'].values[order])
return self
```

Library

```
from h2oaicore.systemutils import segfault,
    ,loggerinfo, main_logger
from h2oaicore.transformer_utils
    import CustomTimeSeriesTransformer
import datatable as dt
import numpy as np
import pandas as pd
import logging
```

Advantages

1. Feature engineering process standardised by:
   1.1 preset parameters
   1.2 preset methods
2. Effort minimisation leads to minimisation in time spent.
3. Build only once - Feature engineering is carried over from training/testing to production.
4. DAI automatically, runs multiple models on various sets of features to get the best model.
5. All the requirements are handled internally by DAI.

References

How to build a recipe

https://github.com/ashrith/how_to_write_a_recipe

Thanks & Questions

- Olivier Grellier, Ph.D, Data Scientist, Kaggle Grandmaster