Anomaly Detection Documentation

windows.py

class Window

Object	object object
Methods	None; to be updated
Attributes	None; to be updated

class DataFrameWin

Object	Window object
Methods	__init__: *df* or *csv_file*: only 1 is optional, if both are provided then *df* is used *df: Pandas DataFrame* *csv_file: Directory to csv file to use as dataframe* *drop_columns*: optional list, only compatible with *csv_file,* drops columns in list *parse_dates*: optional string, name of column of timestamps to parse modify: • *mode*: string, either *“drop”* or *“format_dates”* ○ *drop: value* parameter can be either string or list of strings of column name(s) to drop ○ *format_dates: value* parameter should be string, list of timestamp column to format ▪ timestamp column should be any comprehensible time format ▪ the function modifies by splitting the column into three other columns for time/day/month ▪ time: time past midnight, day: day of week from Monday (0) to Sunday (6), month: (1-12) • *value*: this parameter’s expectation depends on *mode* (see above) create_lag: • *n*: number of lags to create; a lag is simply a column shifted downwards, each lag will be shifted incrementaly • *col*: the column to create the lags with; this will be left intact interpolate_missing_rows: • No parameters, interpolates missing dataframe rows split_by_col: • *col_name:* The name of the column to split by (DataFrameWin objects for each different value in the column) • Returns: Dictionary; keys are the unique values in *col_name*; values are DataFrameWin objects
Attributes	self.dates_formatted: True is dates are formatted using *modify(“format_dates”,…)* and false otherwise self.show_all: If set to true, the next time (and only next time) you print the object, it will show all the rows/columns self.nlags: Keeps track of number of lags in function, DO NOT change this

class GraphWin

Object	Window object
Methods	__init__: • DataFrameWin: DataFrameWin object to connect graphing window to export_DF: • Returns: DataFrameWin object the graph window is connected to create_graph: • mode: string, either *“missing-values”* or *“pacf”* or *“heatmap”* ○ *missing_values:* Creates missing value histogram with 50 bins ▪ *x* should be the x axis ▪ *y* should be the frequency column (how many missing values in y for each x?) ○ *pacf:* Creates partial autocorrelation plot ▪ *x* should be the x axis

<table>
<tr><td></td><td>▪ *y* should be an integer in this case; the x-scale
○ *heatmap*: Creates a heatmap with time of day/day of week; DataFrameWin MUST be time formatted
▪ *x* should be the value column for heatmap values
▪ *y* is not used, set to None
• x: depends on mode, see above
• y: depends on mode, see above</td></tr>
<tr><td>Attributes</td><td>self.DataFrameWin: DataFrameWin object graphing window is connected to</td></tr>
</table>

class TrainingWin

<table>
<tr><td>Object</td><td>Window object</td></tr>
<tr><td>Methods</td><td>__init__
• DataFrameWin: DataFrameWin object to connect training window to

export_DF:
• Returns: DataFrameWin object the graph window is connected to

split:
• x_attr: The X attributes (feature columns)
• y_attr: The y attributes (label columns)
• Creates X/y train/test split based on the above (saved within object)

tune_hps:
• param_grid: A dictionary with the keys being the possible tuning parameters and keys being dictionary with values to consider (hyper parameters saved within object)
○ Possible parameters (keys):
'bootstrap', 'max_depth', 'max_features', 'min_samples_leaf', 'min_samples_split', 'n_estimators'

generate_model:
• Creates model based on saved hyper_parameters, if no hyper parameters are saved then use default
• Model saved within object

save_model:
• dir_: directory to save model as “.sav” file in; “.sav” must be included within the directory

load_model:
• dir_: file directory to load “.sav” file from; model stored within object

predict_csv:
• Predicts csv file using model and exports as a separate csv
• dir1: Directory of first csv file (one to predict out of)
• timeCol: String; name of timestamp column
• valueCol: String; name of value column
• dir2: Directory of second csv file (one to export to)
• anomaly: bool or int; optional, if set as int then create an anomaly column, if set as False then don’t
○ The int represents how strict anomaly detection is (larger = less rows are categorized as anomalies)
▪ |forecasted - ground truth| > *anomaly* * Standard Deviation ==> Anomaly

predict_list:
• Returns prediction given list of features
• lis: The list of feature columns’ values (must be same order given in split method)
• Returns: the prediction given the feature columns’ values</td></tr>
<tr><td>Attributes</td><td>• self.DataFrameWin: DataFrameWin object graphing window is connected to
• self.X_train, self.X_test, self.y_train, self.y_test: The traint/test/x/y splits
• self.hyper_param: The optimal hyper parameters
• self.model: The model that drives everything</td></tr>
</table>

Example Code

```
from windows import Window
from windows import DataFrameWin
from windows import GraphWin
from windows import TrainingWin


df = DataFrameWin(csv_file=r"C:\path\to\file.csv")  # create dataframe from CSV

df.modify("format_dates", "timestamp")  # format "timestamp" column as dates
df.modify("drop", "useless column")  # drops "useless column"
df.interpolate_missing_rows()  # interpolate missing rows
df.create_lag(3, "value")  # create 3 lag columns from "value"

df.show_all=True  # print every row/column of data frame
print(df)

training = TrainingWin(df)  # create training window
training.split(["hourOfDay", "dayOfWeek", "monthOfYear", "lag1", "lag2", "lag3"], "value", 0.2)
# split 80/20%

# Tune the hyper-parameters
param_grid = {
    'bootstrap': [True, False],
    'max_depth': [110, None],
    'max_features': [1.0, "sqrt", "log2"],
    'min_samples_leaf': [1, 2],
    'min_samples_split': [2, 6, 8],
    'n_estimators': [1600, 2000]
}
training.tune_hps(param_grid)

# You can also set them manually
# training.hyper_param = {'bootstrap': False, 'max_depth': None, 'max_features': 'sqrt',
'min_samples_leaf': 1, 'min_samples_split': 6, 'n_estimators': 1600}

# Generate and save model
training.generate_model()
training.save_model(r"path\to\file.sav")


# You can also load a pre-existing .sav file as a model
# training.load_model(r"C:\Users\omaro\Desktop\Data\Riyadh2-210123971-FebToMay-Model.sav")

# Predict "value" column with "timestamp" column and create an anomaly column with 3 * std
training.predict_csv(r"path\to\file.csv", "timestamp", "value", r"path\to\file.csv, 3)
print(training.predict_list([12, 0, 6, 30, 40, 50]))  # prints the prediction for this particular
set of features

# Creates graphing window and graphs heatmap using “value” column
graphing_window = GraphWin(df)
graphing_window.create_graph("heatmap", "value", None)
```