# 内部仕様書

１．メソッド名 (R)

cor.test

２．メソッドの説明

時系列のピアソン相関係数とp値を計算する

３．引数

時系列aと時系列b

４．返り値

時系列のピアソン相関係数，p値，t値などの情報

５．使い方の例

a = c(1, 2, 3, 4)

b = c(5, 6, 7, 8)

cor.test(a,b)

１．メソッド名 (R)

cor.lag

２．メソッドの説明

時系列のLag and Lead相関係数を計算する

３．引数

時系列a，時系列b，Lag時間，Lead時間

４．返り値

時系列のLag and Lead相関係数の配列

５．使い方の例

a = c(1, 2, 3, 4)

b = c(5, 6, 7, 8)

cor.lag(a,b,16,16)

１．メソッド名 (Python)

xgboost.XGBRegressor

xgboost.XGBRegressor.fit

xgboost.XGBRegressor.predict

２．メソッドの説明

xgboost.XGBRegressor:

XGBoostの回帰モデルを実装する

xgboost.XGBRegressor.fit:

XGBoostの回帰モデルを訓練する

xgboost.XGBRegressor.predict:

訓練されたXGBoostの回帰モデルより予測を行う

３．引数

xgboost.XGBRegressor:

n\_estimators: 決定木の数，もしくはBoostingの回数

max\_depth: 決定技の最大深度，オーバーフィッティングを防止

learning\_rate: Boostingの学習率

min\_child\_weight: 決定木の葉の重みの下限，オーバーフィッティングを防止

gamma: 決定技の葉の追加による損失減少の下限

subsample: 各決定木においてランダムに抽出されるデータの割合

colsample\_bytree: 各決定木においてランダムに抽出される列の割合

reg\_alpha: L1正則化の重み

reg\_lambda: L2正則化の重み

n\_jobs: 並列化スレッドの数

scale\_pos\_weight: 正と負の重みのバランスを調整

xgboost.XGBRegressor.fit:

X: 特徴量配列

y: ラベル

eval\_metric: 評価メトリックを設定

eval\_set: 検証用データの入力

xgboost.XGBRegressor.predict:

X: 予測用の特徴量配列

４．返り値

xgboost.XGBRegressor: None

xgboost.XGBRegressor.fit: XGBoost回帰モデル対象

xgboost.XGBRegressor.predict: XGBoost回帰モデルより予測した対象結果

５．使い方の例

param\_dist = {‘max\_depth’: 8, ‘min\_child\_weight’: 1, ‘gamma’: 0, ‘subsample’: 0.7, ‘colsample\_bytree’: 0.8, ‘reg\_alpha’: 0.1, ‘n\_estimators’: 1000, ‘reg\_lambda’: 0.1,

‘learning\_rate’: 0.2 }

model = xgboost.XGBRegressor(\*\*param\_dist)

#x\_trainは訓練用特徴量データ，y\_trainは訓練用ラベル

#x\_testは検証用特徴量データ，y\_testは検証用ラベル

model.fit(x\_train, y\_train, eval\_set=[(x\_test, y\_test)], eval\_metric=’rmse’)

pred = model.predict(x\_test)

１．メソッド名 (Python)

xgboost.plot\_importance

２．メソッドの説明

モデルに各特徴量の重要度を図に示す

３．引数

booster: BoostまたはXGBModel対象

importance\_type: 重要度の計算方法を設定する．“weight”, “gain”, “cover”が設定されられる．

show\_value: 図に数値を表す．

４．返り値

matplotlibの図対象

５．使い方の例

#modelはXGB Modelの対象

xgboost.plot\_importance(model, importance\_type = ‘gain’, show\_value = False)

1. メソッド名 (Python)

from sklearn.model\_selection import RandomizedSearchCV

RamdomizedSearchCV

RamdomizedSearchCV.fit

２．メソッドの説明

モデルのパラメーターをランダムサーチより最適化を行う

３．引数

RamdomizedSearchCV:

estimator: 最適化を行うモデル

param\_distributions: 各パラメーターのサーチ範囲

n\_iter: 繰り返す回数

cv: k-分割交差検証の設定

n\_jobs: 並列化で実行するスレッドの数

RamdomizedSearchCV.fit:

X: 訓練用特徴量

y: 訓練用ラベル

early\_stopping\_rounds: ラウンド経過しても性能が向上しないときはサーチを打ち切る

eval\_set: 検証用データの入力

eval\_metric: 評価メトリックを設定

４．返り値

RamdomizedSearchCV: None

RamdomizedSearchCV.fit: ランダムサーチより適合されたモデル

５．使い方の例

cv\_params ={

'max\_depth':[3,4,5,6,7,8,9,10],

'min\_child\_weight':[1,2,3,4,5],

'gamma':[i/10.0 for i in range(0,6)],

'subsample':[i/10.0 for i in range(6,11)],

'colsample\_bytree':[i/10.0 for i in range(6,11)],

'reg\_alpha':[1e-5, 1e-2, 0.1, 1, 100],

'n\_estimators':[1000,1200,1500,1700,2000],

'reg\_lambda':[1e-5, 1e-2, 0.1, 1],

'learning\_rate':[0.1, 0.2, 0.3]

}

model = xgboost.XGBRegressor()

model\_random = RandomizedSearchCV(model, cv\_params, n\_iter=100, cv=5)

model\_random.fit(x\_train. y\_train, early\_stopping\_rounds=50, eval\_set[(x\_test, y\_test)], eval\_metric=’rmse’)

１．メソッド名 (Python)

from keras.models import Sequential

from tensorflow import keras

Sequential

keras.layers.LSTM

keras.layers.Dropout

keras.layers.Dense

Sequential.compile

Sequential.fit

Sequential.summary

predict

1. メソッドの説明

Sequential:

モデルの層を積み重ねたものを実装

keras.layers.LSTM:

長短期記憶ユニットを実装

keras.layers.Dropout:

入力にドロップアウトを適用

keras.layers.Dense:

通常の前結合ニューラルネットワークレイヤー

Sequential.compile

学習のためのモデルを設定

Sequential.fit

固定回数（データセットの反復）の試行でモデルを学習させる

Sequential.summary

モデルの要約を出力

predict

入力に対する予測の出力を生成する

３．引数

keras.models.Sequential:

モデルの層

keras.layers.LSTM:

units: 出力の次元数

input\_shape: 入力のshapeを設置

return\_sequences: 出力系列の最後の出力を返すか，完全な系列を返すか

keras.layers.Dropout:

rate: 0と１の間の浮動小数点数，入力ユニットをドロップする割合

keras.layers.Dense:

unit: 出力空間の次元数

activation: 使用する活性化関数名．”sofmax”, ”tanh”, “sigmoid”などが使える

Sequential.compile:

optimizer: optimizerのオブジェクトを設定する．”adam”,”nadam”,”adgrad”など設定できる．

loss: 目的関数．“mean\_squared\_error”,”mean\_absolute\_error”,

”mean\_absolute\_percentage\_error”など設定できる

metrics: 訓練時とテスト時にモデルにより評価される評価関数のリスト

Sequential.fit

x: 訓練データの特徴量配列

y: 訓練データのラベル配列（教師データ）

batch\_size: 勾配更新のサンプル数を示す整数．デフォルト値は32

epochs: 訓練データ配列の反復回数を示す整数

verbose: 進行状況の表示モード

validation\_data: 各試行の最後に損失とモデル評価関数を評価するために用いられるテストデータのタプル

Sequential.summary:

None

predict:

x\_test: テストデータの特徴量配列

４．返り値

Sequential:

モデルの対象

keras.layers.LSTM:

LSTM層の対象

keras.layers.Dropout:

ドロップアウト層の対象

keras.layers.Dense:

前結合ニューラルネットワーク層の対象

Sequential.compile:

None

Sequential.fit:

Historyオブジェクト．訓練損失と評価関数値の記録，検証における損失と評価関数値も記録されている

Sequential.summary:

モデルの要約

predict:

予測結果の配列

５．使い方の例

num\_units = 20

output\_dimension = 1

model =Sequential(

[keras.layers.LSTM(num\_units,

input\_shape=(x.train.shape[1], x\_train.shape[2]), return\_sequences=True),

keras.layers.Dropout(0.1),

keras.layers.LSTM(num\_units),

kerase.layers.Dropout(0.1),

keras.layers.Dense(output\_dimension, activation=’linear’)

])

model.compile(loss='mean\_squared\_error',optimizer='adam')

history = model.fit(x\_train, y\_train, epochs=150,

batch\_size=40, validation\_data=(x\_test, y\_test))

model.summary()

pred = model.predict(x\_test)

１．メソッド名 (Python)

Prophet

Prophet.add\_country\_holidays

Prophet.add\_seasonality

Prophet.add\_regressor

Prophet.fit

Prophet.predict

２．メソッドの説明

Prophet:

Prophetモデルの実装

Prophet.add\_country\_holidays:

休日の影響を追加する

Prophet.add\_seasonality:

周期性の影響を追加する

Prophet.add\_regressor:

モデルに説明変数を追加する

Prophet.fit:

Prophetモデルを訓練する

Prophet.predict:

入力に対する予測の出力を生成する

３．引数

Prophet:

Changepoint\_prior\_scale: 最も重要なパラメーターである。非周期性g(t)のトレンドを決めるパラメータであり、トレントがchange pointのところでどのくらい変化をするのかことも決める。トレントがunderfittingまたはoverfittingが起こる場合，このパラメーターより調整する．デフォルトは0.05

seasonality\_prior\_scale: 季節性モデルのflexibilityを決める。値が大きいと季節性が大きな変動に適合し、値が小さいと季節性の大きさが小さくなる。デフォルトは10

holidays\_prior\_scale: 休日モデルのflexibilityを決める．seasonality\_prior\_scaleと似ている意味がある．デフォルトは10

Seasonality\_mode: 時系列により，加法または乗法的季節性を使う．デフォルトは加法，乗法は”multiplicative”

Prophet.add\_country\_holidays:

Country\_name: 国の略称．’JP’（日本），’KR’（韓国），’US’（米国）

Prophet.add\_seasonality

name: どの周期性（週単位，月単位，または年単位）

period: 周期性をフィッティングする時間（週:7，月:30.5，年:365.25）

fourier\_order: フーリエ級数の項数は3に対し週単位の周期性，項数10に対し年単位の周期性を使用している．

prior\_scale: seasonality\_prior\_scaleと同じ意味

Prophet.add\_regressor

x: 説明変数

Prophet.fit:

df: 訓練データ（特徴量とラベル）

Prophet.predict:

df: テストデータの特徴量

４．返り値

Prophet:

Prophetモデルの対象

Prophet.add\_country\_holidays:

None

Prophet.add\_seasonality:

None

Prophet.add\_regressor:

None

Prophet.fit:

None

Prophet.predict:

入力に対する予測結果

５．使い方の例

model = Prophet(seasonality\_mode=’multiplicative’, weekly\_seasonality=False, changepoint\_prior\_scale=0.05,

seasonality\_prior\_scale=10, holidays\_prior\_scale=10)

model.add\_country\_holidays(country\_name=’JP’)

model.add\_seasonality(name=’monthly’, period=30.5,

fourier\_order=5, prior\_scale=0.1)

model.add\_seasonality(name=’yearly’, period=365.25,

fourier\_order=10, prior\_scale=0.15)

model.add\_regressor(‘feature1’)

model.fit(df) #dfは訓練データであるfeature1とラベルを含むDataFrame

test = model.make\_future\_dataframe(period=30)

test[‘feature1’] = df[‘feature1’]

forecast = model.predict(test)

１．メソッド名 (Python)

From sklearn.linear\_model import LinearRegression

LinearRegression

LinearRegression.fit

LinearRegression.predict

２．メソッドの説明

LinearRegression

最小二乗法による線形回帰モデルを実装する

LinearRegression.fit

線形回帰モデルを訓練する

LinearRegression.predict

入力に対する予測結果を生成する

３．引数

LinearRegression:

normalize:正規化を実行するのか（True or False）

n\_jobs: 並列化で実行するスレッドの数

LinearRegression.fit

x: 訓練データ（特徴量）配列

y: 訓練データのラベル配列

LinearRegression.predict

x: テストデータの特徴量配列

４．返り値

LinearRegression

最小二乗法による線形回帰モデルの対象

LinearRegression.fit

None

LinearRegression.predict

入力に対する予測結果

５．使い方の例

LRModel = LinearRegression()

LRModel.fit(x\_train, y\_train)

predict = LRModel.predict(x\_test)