Using CNN for financial time series prediction

by **Adrian Tam** on November 20, 2021 in **Machine Learning for Finance** 34

Convolutional neural networks have their roots in image processing. It was first published in LeNet to recognize the MNIST handwritten digits. However, convolutional neural networks are not limited to handling images.

In this tutorial, we are going to look at an example of using CNN for time series prediction with an application from financial markets. By way of this example, we are going to explore some techniques in using Keras for model training as well.

After completing this tutorial, you will know

- What a typical multidimensional financial data series looks like?
- How can CNN applied to time series in a classification problem
- How to use generators to feed data to train a Keras model
- How to provide a custom metric for evaluating a Keras model

Let's get started

Using CNN for financial time series prediction
Photo by Aron Visuals, some rights reserved.

Tutorial overview

This tutorial is divided into 7 parts; they are:

1. Background of the idea
2. Preprocessing of data
3. Data generator
4. The model
5. Training, validation, and test
6. Extensions
7. Does it work?

Background of the idea

In this tutorial we are following the paper titled “CNNpred: CNN-based stock market prediction using a iverse set of variables” by Ehsan Hoseinzade and Saman Haratizadeh. The data file and sample code from the author are available in github:

- https://github.com/hoseinzadeehsan/CNNpred-Keras

The goal of the paper is simple: To predict the next day’s direction of the stock market (i.e., up or down compared to today), hence it is a binary classification problem. However, it is interesting to see how this problem are formulated and solved.

We have seen the examples on using CNN for sequence prediction. If we consider Dow Jones Industrial Average (DJIA) as an example, we may build a CNN with 1D convolution for prediction. This makes sense because a 1D convolution on a time series is roughly computing its moving average or using digital signal processing terms, applying a filter to the time series. It should provide some clues about the trend.

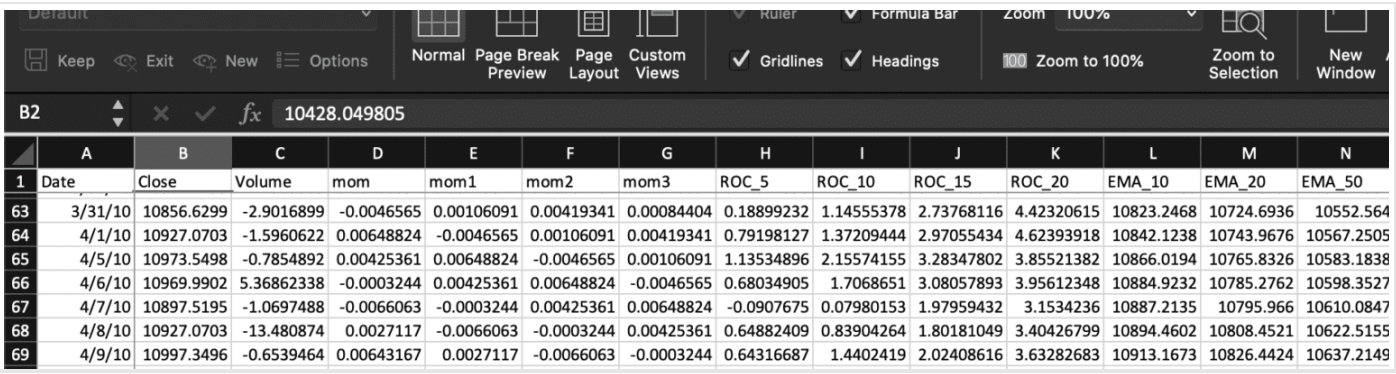
However, when we look at financial time series, it is quite a common sense that some derived signals are useful for predictions too. For example, price and volume together can provide a better clue. Also some other technical indicators such as the moving average of different window size are useful too. If we put all these align together, we will have a table of data, which each time instance has multiple **features**, and the goal is still to predict the direction of **one** time series.

1	Day	which day of the week	Primitive	Pandas
2	Close	Close price	Primitive	Yahoo Finance
3	Vol	Relative change of volume	Technical Indicator	TA-Lib
4	MOM-1	Return of 2 days before	Technical Indicator	TA-Lib
5	MOM-2	Return of 3 days before	Technical Indicator	TA-Lib
6	MOM-3	Return of 4 days before	Technical Indicator	TA-Lib
7	ROC-5	5 days Rate of Change	Technical Indicator	TA-Lib
8	ROC-10	10 days Rate of Change	Technical Indicator	TA-Lib
9	ROC-15	15 days Rate of Change	Technical Indicator	TA-Lib
10	ROC-20	20 days Rate of Change	Technical Indicator	TA-Lib
11	EMA-10	10 days Exponential Moving Average	Technical Indicator	TA-Lib
12	EMA-20	20 days Exponential Moving Average	Technical Indicator	TA-Lib
13	EMA-50	50 days Exponential Moving Average	Technical Indicator	TA-Lib
14	EMA-200	200 days Exponential Moving Average	Technical Indicator	TA-Lib
15	DTB4WK	4-Week Treasury Bill: Secondary Market Rate	Other	FRED
16	DTB3	3-Month Treasury Bill: Secondary Market Rate	Other	FRED
17	DTB6	6-Month Treasury Bill: Secondary Market Rate	Other	FRED
18	DGS5	5-Year Treasury Constant Maturity Rate	Other	FRED
19	DGS10	10-Year Treasury Constant Maturity Rate	Other	FRED
20	DAAA	Moody's Seasoned Aaa Corporate Bond Yield	Other	FRED
21	DBAA	Moody's Seasoned Baa Corporate Bond Yield	Other	FRED
22	TE1	DGS10-DTB4WK	Other	FRED
23	TE2	DGS10-DTB3	Other	FRED
24	TE3	DGS10-DTB6	Other	FRED
25	TE5	DTB3-DTB4WK	Other	FRED
26	TE6	DTB6-DTB4WK	Other	FRED
27	DE1	DBAA-BAAA	Other	FRED
28	DE2	DBAA-DGS10	Other	FRED
29	DE4	DBAA-DTB6	Other	FRED
30	DE5	DBAA-DTB3	Other	FRED
31	DE6	DBAA-DTB4WK	Other	FRED
32	CTB3M	Change in the market yield on U.S. Treasury securities at 3-month constant maturity, quoted on investment basis	Other	FRED
33	CTB6M	Change in the market yield on U.S. Treasury securities at 6-month constant maturity, quoted on investment basis	Other	FRED
34	CTB1Y	Change in the market yield on U.S. Treasury securities at 1-year constant maturity, quoted on investment basis	Other	FRED
35	Oil	Relative change of oil price (WTI), Oklahoma	Commodity	FRED
36	Oil	Relative change of oil price (Brent)	Commodity	Investing.com
37	Oil	Relative change of oil price (WTI)	Commodity	Investing.com
38	Gold	Relative change of gold price (London market)	Commodity	FRED
39	Gold-F	Relative change of gold price futures	Commodity	Investing.com
40	XAU-USD	Relative change of gold spot U.S. dollar	Commodity	Investing.com
41	XAG-USD	Relative change of silver spot U.S. dollar	Commodity	Investing.com

(*continued on next page*)

Excerpt from the CNNpred paper showing the list of features used.

Unlike LSTM, which there is an explicit concept of time steps applied, we present data as a matrix in CNN models. As shown in the table below, the features across multiple time steps are presented as a 2D array.



	A	B	C	D	E	F	G	H	I	J	K	L	M	N
1	Date	Close	Volume	mom	mom1	mom2	mom3	ROC_5	ROC_10	ROC_15	ROC_20	EMA_10	EMA_20	EMA_50
63	3/31/10	10856.6299	-2.9016899	-0.0046565	0.00106091	0.00419341	0.00084404	0.18899232	1.14555378	2.73768116	4.42320615	10823.2468	10724.6936	10552.564
64	4/1/10	10927.0703	-1.5960622	0.00648824	-0.0046565	0.00106091	0.00419341	0.79198127	1.37209444	2.97055434	4.62393918	10842.1238	10743.9676	10567.2505
65	4/5/10	10973.5498	-0.7854892	0.00425361	0.00648824	-0.0046565	0.00106091	1.13534896	2.15574155	3.28347802	3.85521382	10866.0194	10765.8326	10583.1838
66	4/6/10	10969.9902	5.36862338	-0.0003244	0.00425361	0.00648824	-0.0046565	0.68034905	1.7068651	3.08057893	3.95612348	10884.9232	10785.2762	10598.3527
67	4/7/10	10897.5195	-1.0697488	-0.0066063	-0.0003244	0.00425361	0.00648824	-0.0907675	0.07980153	1.97959432	3.1534236	10887.2135	10795.966	10610.0847
68	4/8/10	10927.0703	-13.480874	0.0027117	-0.0066063	-0.0003244	0.00425361	0.64882409	0.83904264	1.80181049	3.40426799	10894.4602	10808.4521	10622.5155
69	4/9/10	10997.3496	-0.6539464	0.00643167	0.0027117	-0.0066063	-0.0003244	0.64316687	1.4402419	2.02408616	3.63282683	10913.1673	10826.4424	10637.2149

78	4/22/10	11134.29	-1.5243758	0.00084226	0.00070705	0.00225475	0.00666049	-0.0922447	1.89638869	2.55751701	2.75134284	11071.1041	10985.8055	10774.126
79	4/23/10	11204.2803	-2.6734707	0.00628601	0.00084226	0.00070705	0.00225475	1.68459789	1.88164123	2.53691019	3.34898331	11095.318	11006.6126	10790.9948
80	4/26/10	11205.0303	-1.9497073	6.69E-05	0.00628601	0.00084226	0.00070705	1.01857159	1.80865976	2.10944017	3.26873864	11115.2656	11025.5095	10807.2315
81	4/27/10	10991.9902	-6.0532092	-0.0190129	6.69E-05	0.00628601	0.00084226	-1.1250217	-0.2489213	0.20054712	0.88226059	11092.8519	11022.3172	10814.4769
82	4/28/10	11045.2695	-0.7879329	0.0048471	-0.0190129	6.69E-05	0.00628601	-0.7159637	-0.6998116	1.35581312	1.263815	11084.2006	11024.5031	10823.5276
83	4/29/10	11167.3203	-1.6077325	0.01105005	0.0048471	-0.0190129	6.69E-05	0.29665362	0.20413528	2.19866802	2.86175759	11099.3133	11038.1048	10837.0097
84	4/30/10	11008.6104	-4.5531541	-0.014212	0.01105005	0.0048471	-0.0190129	-1.7463855	-0.0912071	0.10239506	0.7462205	11082.8218	11035.2958	10843.7391
85	5/3/10	11151.8301	-0.284403	0.01300979	-0.014212	0.01105005	0.0048471	-0.4747885	0.53894703	1.32528396	1.6246363	11095.3688	11046.3943	10855.8211
86	5/4/10	10926.7695	0.11478752	-0.0201815	0.01300979	-0.014212	0.01105005	-0.5933475	-1.711694	-0.8407919	-0.3939904	11064.7144	11035.0014	10858.6034
87	5/5/10	10866.8301	-0.4213986	-0.0054856	-0.0201815	0.01300979	-0.014212	-1.6155283	-2.3199254	-2.3040343	-0.2816187	11028.7354	11018.9851	10858.926
88	5/6/10	10520.3203	8.83210649	-0.0318869	-0.0054856	-0.0201815	0.01300979	-5.7936907	-5.5142243	-5.6013824	-3.7224067	10936.2963	10971.4932	10845.6474
89	5/7/10	10380.4297	-0.9148126	-0.0132972	-0.0318869	-0.0054856	-0.0201815	-5.7062667	-7.3529987	-5.7922693	-5.6097145	10835.2296	10915.2015	10827.4035
90	5/10/10	10785.1396	-3.7997407	0.03898778	-0.0132972	-0.0318869	-0.0054856	-3.2881637	-3.7473404	-2.7669381	-2.0064573	10826.1224	10902.8146	10825.7461
91	5/11/10	10748.2598	-0.9797658	-0.0034195	0.03898778	-0.0132972	-0.0318869	-1.633692	-2.2173461	-3.3174222	-2.460748	10811.9655	10888.0951	10822.7075
92	5/12/10	10896.9102	10.518849	0.01383018	0.0034195	0.03898778	0.0132972	0.27680637	1.3431938	2.0495407	2.0336056	10827.41	10888.9346	10825.6174

Preprocessing of data

In the following, we try to implement the idea of the CNNpred from scratch using Tensorflow's keras API. While there is a reference implementation from the author in the github link above, we reimplement it differently to illustrate some Keras techniques.

Firstly the data are five CSV files, each for a different market index, under the `Dataset` directory from github repository above, or we can also get a copy here:

- CNNpred-data.zip

The input data has a date column and a name column to identify the ticker symbol for the market index. We can leave the date column as time index and remove the name column. The rest are all numerical.

As we are going to predict the market direction, we first try to create the classification label. The market

```
...
X["Target"] = (X["Close"].pct_change().shift(-1) > 0).astype(int)
```

The line of code above is to compute the percentage change of the closing index and align the data with the previous day. Then convert the data into either 1 or 0 for whether the percentage change is positive.

For five data file in the directory, we read each of them as a separate pandas DataFrame and keep them in a Python dictionary:

```
...
data = {}
for filename in os.listdir(DATADIR):
    if not filename.lower().endswith(".csv"):
        continue # read only the CSV files
    filepath = os.path.join(DATADIR, filename)
    X = pd.read_csv(filepath, index_col="Date", parse_dates=True)
    # basic preprocessing: get the name, the classification
    # Save the target variable as a column in dataframe for easier dropna()
    name = X["Name"][0]
    del X["Name"]
    cols = X.columns
    X["Target"] = (X["Close"].pct_change().shift(-1) > 0).astype(int)
    X.dropna(inplace=True)
    # Fit the standard scaler using the training dataset
    index = X.index[X.index > TRAIN_TEST_CUTOFF]
    index = index[:int(len(index) * TRAIN_VALID_RATIO)]
    scaler = StandardScaler().fit(X.loc[index, cols])
    # Save scale transformed dataframe
    X[cols] = scaler.transform(X[cols])
    data[name] = X
```

The result of the above code is a DataFrame for each index, which the classification label is the column “Target” while all other columns are input features. We also normalize the data with a standard scaler.

In time series problems, it is generally reasonable not to split the data into training and test sets randomly, but to set up a cutoff point in which the data before the cutoff is training set while that afterwards is the test set. The scaling above are based on the training set but applied to the entire dataset.

Data generator

We are not going to use all time steps at once, but instead, we use a fixed length of N time steps to predict the market direction at step N+1. In this design, the window of N time steps can start from anywhere. We can just create a large number of DataFrames with large amount of overlaps with one another. To save memory, we are going to build a data generator for training and validation, as follows:

```
...
TRAIN_TEST_CUTOFF = '2016-04-21'
TRAIN_VALID_RATIO = 0.75

def datagen(data, seq_len, batch_size, targetcol, kind):
    "As a generator to produce samples for Keras model"
    batch = []
    while True:
        # Pick one dataframe from the pool
        key = random.choice(list(data.keys()))
        df = data[key]
        input_cols = [c for c in df.columns if c != targetcol]
        index = df.index[df.index < TRAIN_TEST_CUTOFF]
        split = int(len(index) * TRAIN_VALID_RATIO)
        if kind == 'train':
            index = index[:split]   # range for the training set
        elif kind == 'valid':
            index = index[split:]   # range for the validation set
        # Pick one position, then clip a sequence length
        while True:
            t = random.choice(index)      # pick one time step
            n = (df.index == t).argmax()  # find its position in the dataframe
            if n-seq_len+1 < 0:
                continue # can't get enough data for one sequence length
            frame = df.iloc[n-seq_len+1:n+1]
```

```
            X, y = np.expand_dims(np.array(X), 3), np.array(y)
            yield X, y
            batch = []
```

Generator is a special function in Python that does not `return` a value but to `yield` in iterations, such that a sequence of data are produced from it. For a generator to be used in Keras training, it is expected to `yield` a batch of input data and target. This generator supposed to run indefinitely. Hence the generator function above is created with an infinite loop starts with `while True`.

In each iteration, it randomly pick one DataFrame from the Python dictionary, then within the range of time steps of the training set (i.e., the beginning portion), we start from a random point and take N time steps using the pandas `iloc[start:end]` syntax to create a input under the variable `frame`. This DataFrame will be a 2D array. The target label is that of the last time step. The input data and the label are then appended to the list `batch`. Until we accumulated for one batch's size, we dispatch it from the generator.

The last four lines at the code snippet above is to dispatch a batch for training or validation. We collect the list of input data (each a 2D array) as well as a list of target label into variables `X` and `y`, then convert them into numpy array so it can work with our Keras model. We need to add one more dimension to the numpy array `X` using `np.expand_dims()` because of the design of the network model, as explained below.

The Model

The 2D CNN model presented in the original paper accepts an input tensor of shape $\mathrm{N} \times \mathrm{m} \times 1$ for N the number of time steps and m the number of features in each time step. The paper assumes $\mathrm{N} = 60$ and $\mathrm{m} = 82$.

The model comprises of three convolutional layers, as described as follows:

```
        Input(shape=(seq_len, n_features, 1)),
        Conv2D(n_filters[0], kernel_size=(1, n_features), activation="relu"),
        Conv2D(n_filters[1], kernel_size=(3,1), activation="relu"),
        MaxPool2D(pool_size=(2,1)),
        Conv2D(n_filters[2], kernel_size=(3,1), activation="relu"),
        MaxPool2D(pool_size=(2,1)),
        Flatten(),
        Dropout(droprate),
        Dense(1, activation="sigmoid")
    ])
    return model
```

and the model is presented by the following:

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 60, 1, 8)          664
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 58, 1, 8)          200
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 29, 1, 8)          0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 27, 1, 8)          200
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 13, 1, 8)          0
_________________________________________________________________
flatten (Flatten)            (None, 104)               0
_________________________________________________________________
dropout (Dropout)            (None, 104)               0
_________________________________________________________________
dense (Dense)                (None, 1)                 105
=================================================================
Total params: 1,169
Trainable params: 1,169
Non-trainable params: 0
```

The first convolutional layer has 8 units, and is applied across all features in each time step. It is followed by a second convolutional layer to consider three consecutive days at once, for it is a common belief that three days can make a trend in the stock market. It is then applied to a max pooling layer and another convolutional layer before it is flattened into a one-dimensional array and applied to a fully-connected layer with sigmoid activation for binary classification.

Training, validation, and test

That's it for the model. The paper used MAE as the loss metric and also monitor for accuracy and F1 score to determine the quality of the model. We should point out that F1 score depends on precision and recall ratios, which are both considering the positive classification. The paper, however, consider the average of the F1 from positive and negative classification. Explicitly, it is the F1-macro metric:

$$F_1 = \frac{1}{2}\left(\frac{2 \cdot \frac{TP}{TP+FP} \cdot \frac{TP}{TP+FN}}{\frac{TP}{TP+FP} + \frac{TP}{TP+FN}} + \frac{2 \cdot \frac{TN}{TN+FN} \cdot \frac{TN}{TN+FP}}{\frac{TN}{TN+FN} + \frac{TN}{TN+FP}} \right)$$

The fraction $\frac{TP}{TP+FP}$ is the precision with TP and FP the number of true positive and false positive. Similarly $\frac{TP}{TP+FN}$ is the recall. The first term in the big parenthesis above is the normal F1 metric that considered positive classifications. And the second term is the reverse, which considered the negative classifications.

While this metric is available in scikit-learn as `sklearn.metrics.f1_score()` there is no equivalent in Keras. Hence we would create our own by borrowing code from this stackexchange question:

```
from tensorflow.keras import backend as K

def recall_m(y_true, y_pred):
    true_positives = K.sum(K.round(K.clip(y_true * y_pred, 0, 1)))
    possible_positives = K.sum(K.round(K.clip(y_true, 0, 1)))
    recall = true_positives / (possible_positives + K.epsilon())
    return recall

def precision_m(y_true, y_pred):
    true_positives = K.sum(K.round(K.clip(y_true * y_pred, 0, 1)))
    predicted_positives = K.sum(K.round(K.clip(y_pred, 0, 1)))
    precision = true_positives / (predicted_positives + K.epsilon())
    return precision

def f1_m(y_true, y_pred):
    precision = precision_m(y_true, y_pred)
    recall = recall_m(y_true, y_pred)
    return 2*((precision*recall)/(precision+recall+K.epsilon()))

def f1macro(y_true, y_pred):
    f_pos = f1_m(y_true, y_pred)
    # negative version of the data and prediction
    f_neg = f1_m(1-y_true, 1-K.clip(y_pred,0,1))
    return (f_pos + f_neg)/2
```

The training process can take hours to complete. Hence we want to save the model in the middle of the training so that we may interrupt and resume it. We can make use of checkpoint features in Keras:

```
                    save_best_only=True, save_weights_only=False, save_freq="epoch")
]
```

We set up a filename template `checkpoint_path` and ask Keras to fill in the epoch number as well as validation F1 score into the filename. We save it by monitoring the validation's F1 metric, and this metric is supposed to increase when the model gets better. Hence we pass in the `mode="max"` to it.

It should now be trivial to train our model, as follows:

```
seq_len    = 60
batch_size = 128
n_epochs   = 20
n_features = 82

model = cnnpred_2d(seq_len, n_features)
model.compile(optimizer="adam", loss="mae", metrics=["acc", f1macro])
model.fit(datagen(data, seq_len, batch_size, "Target", "train"),
          validation_data=datagen(data, seq_len, batch_size, "Target", "valid"),
          epochs=n_epochs, steps_per_epoch=400, validation_steps=10, verbose=1,
          callbacks=callbacks)
```

Two points to note in the above snippets. We supplied `"acc"` as the accuracy as well as the function `f1macro` defined above as the `metrics` parameter to the `compile()` function. Hence these two metrics will be monitored during training. Because the function is named `f1macro`, we refer to this metric in the checkpoint's `monitor` parameter as `val_f1macro`.

Separately, in the `fit()` function, we provided the input data through the `datagen()` generator as defined above. Calling this function will produce a generator, which during the training loop, batches are fetched from it one after another. Similarly, validation data are also provided by the generator.

Because the nature of a generator is to dispatch data indefinitely. We need to tell the training process on how to define a epoch. Recall that in Keras terms, a batch is one iteration of doing gradient descent update. An epoch is supposed to be one cycle through all data in the dataset. At the end of an epoch is the time to run validation. It is also the opportunity for running the checkpoint we defined above. As Keras has no way to infer the size of the dataset from a generator, we need to tell how many batch it should process in one epoch using the `steps_per_epoch` parameter. Similarly, it is the `validation_steps` parameter to tell how many batch are used in each validation step. The validation does not affect the training, but it will report to us the metrics we are interested. Below is a screenshot of what we will see in the middle of training, which we will see that the metric for training set are updated on each batch but that for validation set is provided only at the end of epoch:

```
Epoch 1/20
400/400 [==============================] - 43s 106ms/step - loss: 0.4062 - acc: 0.6184 - f1ma
Epoch 2/20
400/400 [==============================] - 44s 111ms/step - loss: 0.2760 - acc: 0.7489 - f1ma
Epoch 3/20
 60/400 [===&gt;                          ] - ETA: 39s - loss: 0.2399 - acc: 0.7783 - f1macro
```

```
def testgen(data, seq_len, targetcol):
    "Return array of all test samples"
    batch = []
    for key, df in data.items():
        input_cols = [c for c in df.columns if c != targetcol]
        # find the start of test sample
        t = df.index[df.index &gt;= TRAIN_TEST_CUTOFF][0]
        n = (df.index == t).argmax()
        for i in range(n+1, len(df)+1):
            frame = df.iloc[i-seq_len:i]
            batch.append([frame[input_cols].values, frame[targetcol][-1]])
    X, y = zip(*batch)
    return np.expand_dims(np.array(X),3), np.array(y)

# Prepare test data
test_data, test_target = testgen(data, seq_len, "Target")

# Test the model
test_out = model.predict(test_data)
test_pred = (test_out &gt; 0.5).astype(int)
print("accuracy:", accuracy_score(test_pred, test_target))
print("MAE:", mean_absolute_error(test_pred, test_target))
print("F1:", f1_score(test_pred, test_target))
```

The structure of the function `testgen()` is resembling that of `datagen()` we defined above. Except in `datagen()` the output data's first dimension is the number of samples in a batch but in `testgen()` is the the entire test samples.

Using the model for prediction will produce a floating point between 0 and 1 as we are using the sigmoid activation function. We will convert this into 0 or 1 by using the threshold at 0.5. Then we use the functions from scikit-learn to compute the accuracy, mean absolute error and F1 score (which accuracy is just one minus the MAE).

Tying all these together, the complete code is as follows:

```
import os
import random

import numpy as np
import pandas as pd
import tensorflow as tf
from tensorflow.keras import backend as K
from tensorflow.keras.layers import Dense, Dropout, Flatten, Conv2D, MaxPool2D, Input
from tensorflow.keras.models import Sequential, load_model
from tensorflow.keras.callbacks import ModelCheckpoint
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import accuracy_score, f1_score, mean_absolute_error

DATADIR = "./Dataset"
TRAIN_TEST_CUTOFF = '2016-04-21'
TRAIN_VALID_RATIO = 0.75

# https://datascience.stackexchange.com/questions/45165/how-to-get-accuracy-f1-precision-an
```

```
25
26  def precision_m(y_true, y_pred):
27      true_positives = K.sum(K.round(K.clip(y_true * y_pred, 0, 1)))
28      predicted_positives = K.sum(K.round(K.clip(y_pred, 0, 1)))
29      precision = true_positives / (predicted_positives + K.epsilon())
30      return precision
31
32  def f1_m(y_true, y_pred):
33      precision = precision_m(y_true, y_pred)
34      recall = recall_m(y_true, y_pred)
35      return 2*((precision*recall)/(precision+recall+K.epsilon()))
36
37  def f1macro(y_true, y_pred):
38      f_pos = f1_m(y_true, y_pred)
39      # negative version of the data and prediction
40      f_neg = f1_m(1-y_true, 1-K.clip(y_pred,0,1))
41      return (f_pos + f_neg)/2
42
43  def cnnpred_2d(seq_len=60, n_features=82, n_filters=(8,8,8), droprate=0.1):
44      "2D-CNNpred model according to the paper"
45      model = Sequential([
46          Input(shape=(seq_len, n_features, 1)),
47          Conv2D(n_filters[0], kernel_size=(1, n_features), activation="relu"),
48          Conv2D(n_filters[1], kernel_size=(3,1), activation="relu"),
49          MaxPool2D(pool_size=(2,1)),
50          Conv2D(n_filters[2], kernel_size=(3,1), activation="relu"),
51          MaxPool2D(pool_size=(2,1)),
52          Flatten(),
53          Dropout(droprate),
54          Dense(1, activation="sigmoid")
55      ])
56      return model
57
58  def datagen(data, seq_len, batch_size, targetcol, kind):
59      "As a generator to produce samples for Keras model"
60      batch = []
61      while True:
62          # Pick one dataframe from the pool
63          key = random.choice(list(data.keys()))
64          df = data[key]
65          input_cols = [c for c in df.columns if c != targetcol]
66          index = df.index[df.index < TRAIN_TEST_CUTOFF]
67          split = int(len(index) * TRAIN_VALID_RATIO)
68          assert split > seq_len, "Training data too small for sequence length {}".format(seq
69          if kind == 'train':
70              index = index[:split]   # range for the training set
71          elif kind == 'valid':
72              index = index[split:]   # range for the validation set
73          else:
74              raise NotImplementedError
75          # Pick one position, then clip a sequence length
76          while True:
77              t = random.choice(index)     # pick one time step
78              n = (df.index == t).argmax() # find its position in the dataframe
79              if n-seq_len+1 < 0:
80                  continue # this sample is not enough for one sequence length
81              frame = df.iloc[n-seq_len+1:n+1]
82              batch.append([frame[input_cols].values, df.loc[t, targetcol]])
```

```
            batch = []

def testgen(data, seq_len, targetcol):
    "Return array of all test samples"
    batch = []
    for key, df in data.items():
        input_cols = [c for c in df.columns if c != targetcol]
        # find the start of test sample
        t = df.index[df.index >= TRAIN_TEST_CUTOFF][0]
        n = (df.index == t).argmax()
        # extract sample using a sliding window
        for i in range(n+1, len(df)+1):
            frame = df.iloc[i-seq_len:i]
            batch.append([frame[input_cols].values, frame[targetcol][-1]])
    X, y = zip(*batch)
    return np.expand_dims(np.array(X),3), np.array(y)

# Read data into pandas DataFrames
data = {}
for filename in os.listdir(DATADIR):
    if not filename.lower().endswith(".csv"):
        continue # read only the CSV files
    filepath = os.path.join(DATADIR, filename)
    X = pd.read_csv(filepath, index_col="Date", parse_dates=True)
    # basic preprocessing: get the name, the classification
    # Save the target variable as a column in dataframe for easier dropna()
    name = X["Name"][0]
    del X["Name"]
    cols = X.columns
    X["Target"] = (X["Close"].pct_change().shift(-1) > 0).astype(int)
    X.dropna(inplace=True)
    # Fit the standard scaler using the training dataset
    index = X.index[X.index < TRAIN_TEST_CUTOFF]
    index = index[:int(len(index) * TRAIN_VALID_RATIO)]
    scaler = StandardScaler().fit(X.loc[index, cols])
    # Save scale transformed dataframe
    X[cols] = scaler.transform(X[cols])
    data[name] = X

seq_len = 60
batch_size = 128
n_epochs = 20
n_features = 82

# Produce CNNpred as a binary classification problem
model = cnnpred_2d(seq_len, n_features)
model.compile(optimizer="adam", loss="mae", metrics=["acc", f1macro])
model.summary()  # print model structure to console

# Set up callbacks and fit the model
# We use custom validation score f1macro() and hence monitor for "val_f1macro"
checkpoint_path = "./cp2d-{epoch}-{val_f1macro:.2f}.h5"
callbacks = [
    ModelCheckpoint(checkpoint_path,
                    monitor='val_f1macro', mode="max",
                    verbose=0, save_best_only=True, save_weights_only=False, save_freq="epo
]
model.fit(datagen(data, seq_len, batch_size, "Target", "train"),
```

```
# Test the model
test_out = model.predict(test_data)
test_pred = (test_out > 0.5).astype(int)
print("accuracy:", accuracy_score(test_pred, test_target))
print("MAE:", mean_absolute_error(test_pred, test_target))
print("F1:", f1_score(test_pred, test_target))
```

Extensions

The original paper called the above model “2D-CNNpred” and there is a version called “3D-CNNpred”. The idea is not only consider the many features of one stock market index but cross compare with many market indices to help prediction on **one index**. Refer to the table of features and time steps above, the data for one market index is presented as 2D array. If we stack up multiple such data from different indices, we constructed a 3D array. While the target label is the same, but allowing us to look at a different market may provide some additional information to help prediction.

Because the shape of the data changed, the convolutional network also defined slightly different, and the data generators need some modification accordingly as well. Below is the complete code of the 3D version, which the change from the previous 2d version should be self-explanatory:

```
import os
import random

import numpy as np
import pandas as pd
import tensorflow as tf
from tensorflow.keras import backend as K
from tensorflow.keras.layers import Dense, Dropout, Flatten, Conv2D, MaxPool2D, Input
from tensorflow.keras.models import Sequential, load_model
from tensorflow.keras.callbacks import ModelCheckpoint
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import accuracy_score, f1_score, mean_absolute_error

DATADIR = "./Dataset"
```

```
    true_positives = K.sum(K.round(K.clip(y_true * y_pred, 0, 1)))
    possible_positives = K.sum(K.round(K.clip(y_true, 0, 1)))
    recall = true_positives / (possible_positives + K.epsilon())
    return recall

def precision_m(y_true, y_pred):
    true_positives = K.sum(K.round(K.clip(y_true * y_pred, 0, 1)))
    predicted_positives = K.sum(K.round(K.clip(y_pred, 0, 1)))
    precision = true_positives / (predicted_positives + K.epsilon())
    return precision

def f1_m(y_true, y_pred):
    precision = precision_m(y_true, y_pred)
    recall = recall_m(y_true, y_pred)
    return 2*((precision*recall)/(precision+recall+K.epsilon()))

def f1macro(y_true, y_pred):
    f_pos = f1_m(y_true, y_pred)
    # negative version of the data and prediction
    f_neg = f1_m(1-y_true, 1-K.clip(y_pred,0,1))
    return (f_pos + f_neg)/2

def cnnpred_3d(seq_len=60, n_stocks=5, n_features=82, n_filters=(8,8,8), droprate=0.1):
    "3D-CNNpred model according to the paper"
    model = Sequential([
        Input(shape=(n_stocks, seq_len, n_features)),
        Conv2D(n_filters[0], kernel_size=(1,1), activation="relu", data_format="channels_la
        Conv2D(n_filters[1], kernel_size=(n_stocks,3), activation="relu"),
        MaxPool2D(pool_size=(1,2)),
        Conv2D(n_filters[2], kernel_size=(1,3), activation="relu"),
        MaxPool2D(pool_size=(1,2)),
        Flatten(),
        Dropout(droprate),
        Dense(1, activation="sigmoid")
    ])
    return model

def datagen(data, seq_len, batch_size, target_index, targetcol, kind):
    "As a generator to produce samples for Keras model"
    # Learn about the data's features and time axis
    input_cols = [c for c in data.columns if c[0] != targetcol]
    tickers = sorted(set(c for _,c in input_cols))
    n_features = len(input_cols) // len(tickers)
    index = data.index[data.index < TRAIN_TEST_CUTOFF]
    split = int(len(index) * TRAIN_VALID_RATIO)
    assert split > seq_len, "Training data too small for sequence length {}".format(seq_len
    if kind == "train":
        index = index[:split]   # range for the training set
    elif kind == 'valid':
        index = index[split:]   # range for the validation set
    else:
        raise NotImplementedError
    # Infinite loop to generate a batch
    batch = []
    while True:
        # Pick one position, then clip a sequence length
        while True:
            t = random.choice(index)
```

```
            X = np.full(shape, np.nan)
            for i,ticker in enumerate(tickers):
                X[i] = frame.xs(ticker, axis=1, level=1).values
            batch.append([X, data[targetcol][target_index][t]])
            break
        # if we get enough for a batch, dispatch
        if len(batch) == batch_size:
            X, y = zip(*batch)
            yield np.array(X), np.array(y)
            batch = []

def testgen(data, seq_len, target_index, targetcol):
    "Return array of all test samples"
    input_cols = [c for c in data.columns if c[0] != targetcol]
    tickers = sorted(set(c for _,c in input_cols))
    n_features = len(input_cols) // len(tickers)
    t = data.index[data.index >= TRAIN_TEST_CUTOFF][0]
    n = (data.index == t).argmax()
    batch = []
    for i in range(n+1, len(data)+1):
        # Clip a window of seq_len ends at row position i-1
        frame = data.iloc[i-seq_len:i]
        target = frame[targetcol][target_index][-1]
        frame = frame[input_cols]
        # convert frame with two level of indices into 3D array
        shape = (len(tickers), len(frame), n_features)
        X = np.full(shape, np.nan)
        for i,ticker in enumerate(tickers):
            X[i] = frame.xs(ticker, axis=1, level=1).values
        batch.append([X, target])
    X, y = zip(*batch)
    return np.array(X), np.array(y)

# Read data into pandas DataFrames
data = {}
for filename in os.listdir(DATADIR):
    if not filename.lower().endswith(".csv"):
        continue # read only the CSV files
    filepath = os.path.join(DATADIR, filename)
    X = pd.read_csv(filepath, index_col="Date", parse_dates=True)
    # basic preprocessing: get the name, the classification
    # Save the target variable as a column in dataframe for easier dropna()
    name = X["Name"][0]
    del X["Name"]
    cols = X.columns
    X["Target"] = (X["Close"].pct_change().shift(-1) > 0).astype(int)
    X.dropna(inplace=True)
    # Fit the standard scaler using the training dataset
    index = X.index[X.index < TRAIN_TEST_CUTOFF]
    index = index[:int(len(index) * TRAIN_VALID_RATIO)]
    scaler = StandardScaler().fit(X.loc[index, cols])
    # Save scale transformed dataframe
    X[cols] = scaler.transform(X[cols])
    data[name] = X

# Transform data into 3D dataframe (multilevel columns)
for key, df in data.items():
    df.columns = pd.MultiIndex.from_product([df.columns, [key]])
```

```
n_stocks = 5

# Produce CNNpred as a binary classification problem
model = cnnpred_3d(seq_len, n_stocks, n_features)
model.compile(optimizer="adam", loss="mae", metrics=["acc", f1macro])
model.summary() # print model structure to console

# Set up callbacks and fit the model
# We use custom validation score f1macro() and hence monitor for "val_f1macro"
checkpoint_path = "./cp3d-{epoch}-{val_f1macro:.2f}.h5"
callbacks = [
    ModelCheckpoint(checkpoint_path,
                    monitor='val_f1macro', mode="max",
                    verbose=0, save_best_only=True, save_weights_only=False, save_freq="epo
]

model.fit(datagen(data, seq_len, batch_size, "DJI", "Target", "train"),
          validation_data=datagen(data, seq_len, batch_size, "DJI", "Target", "valid"),
          epochs=n_epochs, steps_per_epoch=400, validation_steps=10, verbose=1, callbacks=c

# Prepare test data
test_data, test_target = testgen(data, seq_len, "DJI", "Target")

# Test the model
test_out = model.predict(test_data)
test_pred = (test_out > 0.5).astype(int)
print("accuracy:", accuracy_score(test_pred, test_target))
print("MAE:", mean_absolute_error(test_pred, test_target))
print("F1:", f1_score(test_pred, test_target))
```

While the model above is for next-step prediction, it does not stop you from making prediction for k steps ahead if you replace the target label to a different calculation. This may be an exercise for you.

Does it work?

As in all prediction projects in the financial market, it is always unrealistic to expect a high accuracy. The training parameter in the code above can produce slightly more than 50% accuracy in the testing set. While the number of epochs and batch size are deliberately set smaller to save time, there should not be much room for improvement.

In the original paper, it is reported that the 3D-CNNpred performed better than 2D-CNNpred but only attaining the F1 score of less than 0.6. This is already doing better than three baseline models mentioned in the paper. It may be of some use, but not a magic that can help you make money quick.

From machine learning technique perspective, here we classify a panel of data into whether the market

Further readings

The original paper is available at:

- “CNNPred: CNN-based stock market prediction using several data sources”, by Ehsan Hoseinzade, Saman Haratizadeh, 2019. (https://arxiv.org/abs/1810.08923)

If you are new to finance application and want to build the connection between machine learning techniques and finance, you may find this book useful:

- *Machine Learning in Finance: From Theory to Practice*, by Matthew F. Dixon, Igor Halperin, and Paul Bilokon. 2000. (https://www.amazon.com/dp/3030410676/)

On the similar topic, we have a previous post on using CNN for time series, but using 1D convolutional layers;

- How to develop convolutional neural network models for time series forecasting

You may also find the following documentation helpful to explain some syntax we used above:

- Panads user guide: https://pandas.pydata.org/pandas-docs/stable/user_guide/index.html
- Keras model training API: https://keras.io/api/models/model_training_apis/
- Keras callbacks API: https://keras.io/api/callbacks/

Summary

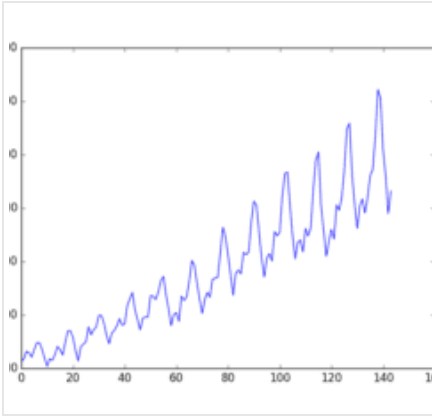
In this tutorial, you discovered how a CNN model can be built for prediction in financial time series.

Specifically, you learned:

- How to create 2D convolutional layers to process the time series
- How to present the time series data in a multidimensional array so that the convolutional layers can be applied
- What is a data generator for Keras model training and how to use it
- How to monitor the performance of model training with a custom metric
- What to expect in predicting financial market

More On This Topic

Understand Time Series Forecast Uncertainty Using…



Time Series Prediction with Deep Learning in Keras

Time Series Prediction with LSTM Recurrent Neural…

LSTM for Time Series Prediction in PyTorch

CNN Long Short-Term Memory Networks

How to Develop a Multichannel CNN Model for Text…

About Adrian Tam

Adrian Tam, PhD is a data scientist and software engineer.

View all posts by Adrian Tam →

‹ Face Recognition using Principal Component Analysis

Visualizing the vanishing gradient problem ›

REPLY

Paul Kornreich November 19, 2021 at 2:40 am #

This has promise,, especially using multiple parameters, but, in general, CNNs are losing out in accuracy compared with Transformers.

Adrian Tam November 19, 2021 at 10:31 am #

REPLY

Correct. But this shows how simple it is to get something not too far away.

tanunchai November 19, 2021 at 5:56 am #

REPLY

There is error at a line in def datagen() on the following:

index = data.index[data.index < TRAIN_TEST_CUTOFF]

compliler said

“File “C:\Users\TANUNC~1.J\AppData\Local\Temp/ipykernel_10252/1129467454.py”, line 66
index = data.index[data.index < TRAIN_TEST_CUTOFF]
^
SyntaxError: invalid syntax
”

How to solve it ?

waiting your answer, Thanks in advance

Adrian Tam November 19, 2021 at 10:39 am #

REPLY

The line seems OK but maybe the line before it in your code caused some problem. Can you check if you copied the code correctly?

William Smith November 19, 2021 at 9:44 am #

REPLY

@tanunchai
The code above seems to say

index = data.index[data.index < TRAIN_TEST_CUTOFF]

Try changing to

index = data.index[data.index < TRAIN_TEST_CUTOFF]

William Smith November 19, 2021 at 9:45 am #

REPLY

Argh. The code actually says [data.index & l t ; TRAIN_TEST_CUTOFF]

Change to

index = data.index[data.index < TRAIN_TEST_CUTOFF]

Adrian Tam November 19, 2021 at 10:57 am #

REPLY

Thanks William. There were some hassles with the HTML. I fixed that and the copy should work now.

tanunchai November 19, 2021 at 10:09 pm #

REPLY

Thanks you William Smith

tanunchai November 19, 2021 at 10:17 pm #

REPLY

How to solve this bug ?

Now facing new problem, in def datagen() at nest loop while true
has the problem on the following:

if n-seq_len+1 &; 0:

while True:

if n-seq_len+1 &; 0: # ****error said invalid syntax
continue # this sample is not enough for one sequence length
frame = data.iloc[n-seq_len+1:n+1][input_cols]
# convert frame with two l

tanunchai November 19, 2021 at 10:20 pm #

REPLY

Now facing new problem, in def datagen() at nest loop while true
has the problem on the following:

if n-seq_len+1 < 0:

while True:
# Pick one position, then clip a sequence length
while True:
t = random.choice(index)
n = (data.index == t).argmax()
#index = data.index[data.index < TRAIN_TEST_CUTOFF] , the line below has the problem
if n-seq_len+1 < 0:
continue # this sample is not enough for one sequence length
frame = data.iloc[n-seq_len+1:n+1][input_cols]
# convert frame with two level of indices into 3D array
shape = (len(tickers), len(frame), n_features)
X = np.full(shape, np.nan)
for i,ticker in enumerate(tickers):

tanunchai November 19, 2021 at 10:26 pm #

REPLY

at def datagen() still has problem on the following;

if n-seq_len+1 < 0: # Error line , say “invalid syntax”

&lt doing what ? I do not understand and it made error also.

_____

while True:
# Pick one position, then clip a sequence length
while True:
t = random.choice(index)
n = (data.index == t).argmax()

```
# convert frame with two level of indices into 3D array
shape = (len(tickers), len(frame), n_features)
```

tanunchai November 19, 2021 at 10:34 pm #

REPLY

ModuleNotFoundError: No module named ‘f1metrics’

How to solve this bug ?

_____

ModuleNotFoundError Traceback (most recent call last)
C:\Users\TANUNC~1.J\AppData\Local\Temp/ipykernel_2580/1760727365.py in
12 from sklearn.metrics import accuracy_score, f1_score, mean_absolute_error
13
—> 14 from f1metrics import f1macro
15
16 DATADIR = “./Dataset”

ModuleNotFoundError: No module named ‘f1metrics’

_____

Adrian Tam November 20, 2021 at 2:33 am #

REPLY

Sorry for all these hassles. Something wrong with the plugin for rendering the code here caused all the mess. I fixed it for now, so please copy over the code and try again.

Jack November 20, 2021 at 5:39 am #

REPLY

Interesting! What is the difference between using:

– conv1d with kernel_size=n_features and input size N x m
– conv2d with kernel_size=(1, n_features) and input size N x m x 1

Are these two equivalent?

Adrian Tam November 20, 2021 at 1:36 pm #

REPLY

Doni November 21, 2021 at 2:52 pm #

REPLY

how to deploy this code on the web app. ?

Adrian Tam November 23, 2021 at 1:06 pm #

REPLY

That's a vague question – you need to think about what the web app expects and how to wrap the model into a function to talk to the web app

kaplan November 29, 2021 at 7:36 am #

REPLY

Is FFDNet (fast and flexible denoising convolutional neural networks) suitable for financial time series?

Adrian Tam November 29, 2021 at 8:56 am #

REPLY

Haven't tried that.

Priya December 1, 2021 at 9:14 pm #

REPLY

My question is from the blog https://machinelearningmastery.com/feature-selection-machine-learning-python/
Sorry for asking this question here.

My work is to develop a model for multioutput prediction (i.e., predicting five outputs by a single model). When I applied Kbest and recursive feature elimination methods to select the best features, I got an error ' bad input shape (x, 5)' (5 is output vectors here). However, PCA works well as it doesn't depend upon the output vector.
Does it mean that we can apply these feature selection algorithms (Kbest and RFE) only for a single output prediction problem?

Adrian Tam December 2, 2021 at 2:55 am #

REPLY

REPLY

Pete March 28, 2022 at 4:32 am #

I think there is too much information in the csv file. Instead of looking too much data and try to make a fundamental analysis it can also be tried to find a ‘pattern’. With pattern I mean looking only to the open price, close price, max price and min price information of each candlestick.

For example:

Let’s consider the candlestick chart of D1 (1 candlestick = 1 day info). Objective: predict the fifth candlestick (up or down) with the info of the 4 formed before. The less quantity of candlestick the easier a pattern can be found

I will try modifying the input data with this model and let you know the results

James Carmichael March 28, 2022 at 7:05 am #

REPLY

Thanks for the feedback Pete! Please share your findings.

Bojie May 10, 2022 at 8:28 am #

REPLY

Many thanks for your tutorial, which is very helpful.

Can you please let me know how to write a data generator to read all different classes of .txt files (likes the function of “datagen.flow_from_directory” which read all different classes of image files)?

Yu June 15, 2022 at 4:10 am #

REPLY

Thanks for this useful article!

I was trying to run your codes a couple of times but there are certain instability issues happened. Sometimes the prediction rate is about 51%, which is good result, but there are also times the prediction rate will go down to 48%. My guess is that this is related to the initialization step. Do you have any insights on this issue?

James Carmichael June 15, 2022 at 7:16 am #

REPLY

Hi Yu…The following resource should add clarity:

REPLY

Yu June 16, 2022 at 7:10 am #

Hi, James,

Thanks for your suggestion! I do consider the multi-restart idea and similarly using ensemble of results. But I didn't have any luck for that yet either.

Do you think for a larger dataset this issue will be alleviated?

rk November 26, 2022 at 5:55 am #

REPLY

Hi, Could you help me i am having an issue with the code

InvalidArgumentError: Graph execution error

It is associated with [[model.fit(datagen(data, seq_len, batch_size,"Target","train"), validation_data=datagen(data, seq_len, batch_size, "Target", "valid"), epochs=n_epochs, steps_per_epoch=400, validation_steps=10, verbose=1, callbacks=callbacks)]]

I was able to trace it back to this line within the code in datagen (2DConv):

input_cols = [c for c in df.columns if c != targetcol]

Im using google colab and it says that when i try to single it out and run it

ValueError: The truth value of a Series is ambiguous. Use a.empty, a.bool(), a.item(), a.any() or a.all().

After trying: input_cols = [c for c in df.columns if c != targetcol.all()]

The issue persists, i would appreciate your help on it

James Carmichael November 26, 2022 at 8:22 am #

REPLY

Hi rk…did you copy and paste the code or type it in? Also, you may want to try to execute your code in Google Colab.

de santos March 25, 2023 at 8:03 pm #

REPLY

Hello, Can you tell me what the following lines of code do? They are:

name = X["Name"][0]

Since I get a bug here on google colab that return the error :

KeyError : “name”

This only happen when I use my own .csv data files. When I ran the codes using your dataset .csv files, I did not encounter any problem though. My csv files only have 7 columns : Date,Close,Open, HIgh, Low, Vol, Change %.
From what I understood, you are trying to drop the other columns but the Date and Close columns, right?
Thanks in advance!

James Carmichael March 26, 2023 at 10:33 am #

REPLY

Hi de santos…You are correct. Can you provide any more detail about the error? Is that the entire error message?

de santos June 27, 2023 at 6:57 pm #

REPLY

Thanks for replying, James! I thought my response was not approved,hence the late reply 4 months later, I’m so sorry.There was another error about Integer data type since originally the Vol column had a word “k” as substitute for “thousand”, that can be easily fixed by Find/ Replace function of Excel to uniform the data into integer numbers. Other than that, the “Name” error still persists, here is the full error message:

KeyError Traceback (most recent call last)

/usr/local/lib/python3.10/dist-packages/pandas/core/indexes/base.py in get_loc(self, key, method, tolerance)
3801 try:
-> 3802 return self._engine.get_loc(casted_key)
3803 except KeyError as err:

4 frames

pandas/_libs/hashtable_class_helper.pxi in pandas._libs.hashtable.PyObjectHashTable.get_item()

pandas/_libs/hashtable_class_helper.pxi in pandas._libs.hashtable.PyObjectHashTable.get_item()

KeyError: ‘Name’

The above exception was the direct cause of the following exception:

3803 except KeyError as err:
-> 3804 raise KeyError(key) from err
3805 except TypeError:
3806 # If we have a listlike key, _check_indexing_error will raise

KeyError: 'Name'

the broad, jeff October 16, 2023 at 2:38 am #

REPLY

I am confused with the below code and would like to seek some clarification:

while True:
t = random.choice(index) # pick one time step

Won't this line of code, always pick up from an index randomly, thus breaking sequence of time series data. As I understand in time series problems, the sequence of time series should be preserved?

What am I missing here? Can you please help me understand this better?

James Carmichael October 16, 2023 at 6:05 am #

REPLY

Hi…This line simply picks a random starting point in the sequence. From there, the sequence is preserved.

Leave a Reply

Name (required)

SUBMIT COMMENT

Welcome!
I'm *Jason Brownlee* PhD
and I **help developers** get results with **machine learning**.
Read more

Never miss a tutorial:

Picked for you:


Your First Deep Learning Project in Python with Keras Step-by-Step


Your First Machine Learning Project in Python Step-By-Step


How to Develop LSTM Models for Time Series Forecasting

Machine Learning for Developers