

Kerasを用いた 株価騰落予測の試み

2017/11/16

石垣哲郎

本プレゼンは、日経平均の過去データから未来日（翌営業日および5営業日後）における騰落およびその度合いを、Kerasを用いた錬金術的手法によって求めてみた、その結果についてのご紹介です。

発表者自己紹介

氏名: **石垣 哲郎**

1986年4月 日本電気株式会社入社

2015年11月 日本インサイトテクノロジー株式会社入社

TensorflowやKERASは仕事ではなく、もっぱらオフタイムに触っています。

Kerasとは

Kerasは、Pythonで書かれた、TensorFlowまたはCNTK, Theano上で実行可能な高水準のニューラルネットワークライブラリです。Kerasは、迅速な実験を可能にすることに重点を置いて開発されました。 **アイデアから結果に到達するまでのリードタイムをできるだけ小さくすることが、良い研究をするための鍵になります。**

次のような場合で深層学習ライブラリが必要なら、Kerasを使用してください:

- 容易に素早くプロトタイプの実成が可能(ユーザーフレンドリー, モジュール性, および拡張性による)
- CNNとRNNの両方, およびこれらの2つの組み合わせをサポート
- CPUとGPU上でシームレスな動作

(Kerasドキュメンテーションより)

Kerasによるモデル構成

KerasのSequentialモデルを用い、各メソッドを積み上げて構成した。
以下はKerasドキュメンテーションに掲載されていた、LSTMの例。

```
from keras.models import Sequential
from keras.layers import Dense, Dropout
from keras.layers import Embedding
from keras.layers import LSTM
```

```
model = Sequential()
model.add(Embedding(max_features, output_dim=256))
model.add(LSTM(128))
model.add(Dropout(0.5))
model.add(Dense(1, activation='sigmoid'))
```

addメソッドを1行記述するだけで、簡単に層追加が出来る！

```
model.compile(loss='binary_crossentropy',
              optimizer='rmsprop',
              metrics=['accuracy'])
```

```
model.fit(x_train, y_train, batch_size=16, epochs=10)
score = model.evaluate(x_test, y_test, batch_size=16)
```

実際に使ったコード**[1/2]**

```
class Prediction :
    def __init__(self,maxlen,n_hidden,n_in,n_out):
        self.maxlen=maxlen
        self.n_hidden=n_hidden
        self.n_in=n_in
        self.n_out=n_out

    def create_model(self):
        model = Sequential()
        model.add(LSTM(self.n_hidden, batch_input_shape=(None, self.maxlen, self.n_in),
            activation='tanh', recurrent_activation='hard_sigmoid', use_bias=True,
            kernel_initializer=glorot_uniform(seed=20170719),
            recurrent_initializer=orthogonal(gain=1.0, seed=20170719),
            bias_initializer='zeros',
            unit_forget_bias=True,
            kernel_regularizer=None,
            recurrent_regularizer=None,
            bias_regularizer=None,
            activity_regularizer=None,
            kernel_constraint=None,
            recurrent_constraint=None,
            bias_constraint=None,
            dropout=0.5,
            recurrent_dropout=0.5))
```

実際に使ったコード**[2/2]**

```
model.add(Dropout(0.5,noise_shape=None, seed=None))
model.add(Dense(self.n_out,activation=None, use_bias=True,
                kernel_initializer=glorot_uniform(seed=20170719),
                bias_initializer='zeros',
                kernel_regularizer=None,
                bias_regularizer=None,
                activity_regularizer=None,
                kernel_constraint=None,
                bias_constraint=None))
model.add(Activation("softmax"))
model.compile(loss="categorical_crossentropy",optimizer="RMSprop",
              metrics=['categorical_accuracy'])
return model
```

学習

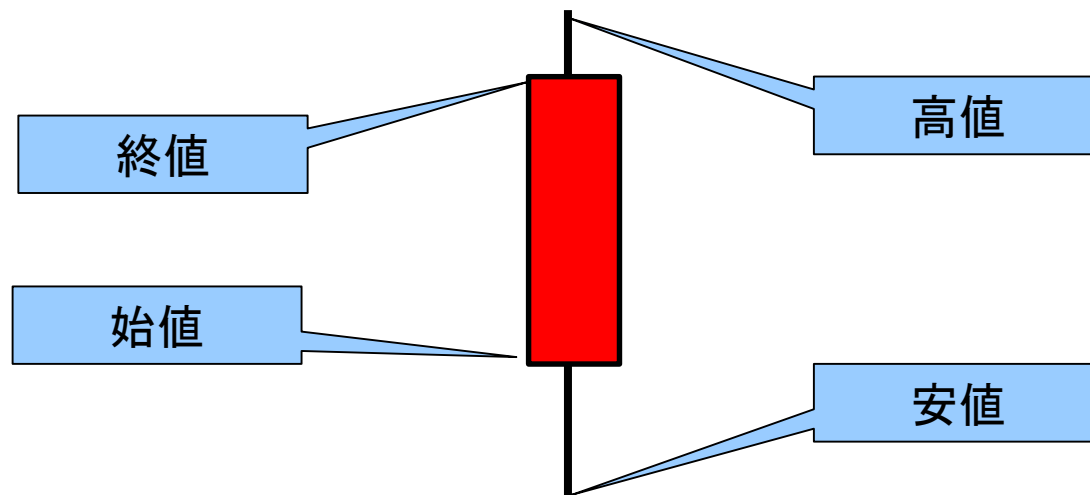
```
def train(self, x_train, t_train,batch_size,epochs) :
    early_stopping = EarlyStopping(patience=10, verbose=1)
    model = self.create_model()
    model.fit(x_train, t_train, batch_size=batch_size, epochs=epochs,verbose=1,
              shuffle=True,callbacks=[early_stopping],validation_split=0.1)
    return model
```

モデルの説明

- 系列データ
 - 日経平均(始値、高値、安値、終値)、NYダウ終値、ドル円相場(6種類)
- ネットワーク:LSTM
- 出力:翌営業日の日経平均終値を、以下の4分類の確率 (softmax)で予測
 - 1%以上上昇、1%未満の上昇、-1%以内の下落、-1%以上の下落

系列データ項目選択の理由

- NYダウおよびドル円相場は、明らかに日経平均と相関がある
 - 日経平均は前日のNYダウの値動きに連動することが多い
 - 円安になると日経平均は上がり、円高になると逆に下がる
- ろうそくチャートの形状(ひげの長さ等)が投資判断に使われることがある



系列データの加工

値をそのまま使うのではなく、**対数**を取った上で**前日との差分**を算出する

※人間は大きさの評価を対数目盛での大小比較で行っている(ex.音程(ドレミ)、音量(dB)、マグニチュード)

※ブラックショールズ方程式は株価の対数値の挙動に対する方程式

元データ

Date	Open	High	Low	Close
1986/9/3	18,694	18,694	18,393	18,505
1986/9/4	18,515	18,624	18,514	18,560

自然対数

Date	Open	High	Low	Close
1986/9/3	9.835958	9.835958	9.819725	9.825796
1986/9/4	9.826336	9.832206	9.826282	9.828764

前日差分

Date	Open	High	Low	Close
1986/9/4	-0.00962	-0.00375	0.006557	0.002968

100倍

Date	Open	High	Low	Close
1986/9/4	-0.96214	-0.37515	0.655705	0.296776

※**100倍**しているのは値を見やすくするため

系列データの範囲

ドル円のデータが2002年までしかさかのぼれなかったため、以下の2種類を用意した。

- パターン1:ドル円データ有。日経平均(始値、高値、安値、終値) + NYダウ終値 + ドル円(6種類)

2002/4/2～2017/7/20

- パターン2:ドル円データなし。日経平均(始値、高値、安値、終値) + NYダウ終値(5種類)

1986/9/4～2017/7/20

系列データのイメージ

Date	Open	High	Low	C bse	NYDOW	JPY
1986/9/3						
1986/9/4	-0.962140	-0.375155	0.655705	0.296776	2.019516	
1986/9/5	0.307386	0.823490	0.312787	1.151747	-1.045181	
1986/9/8	1.097765	0.180899	-0.145486	-1.194860	-0.586530	
1986/9/9	-1.286398	-1.472623	-0.605767	-0.421326	-0.238551	
1986/9/10	-0.324202	0.532646	0.216767	0.781823	-0.246571	
1986/9/11	0.744099	0.171563	0.405176	-0.311996	-4.717693	
1986/9/12	-0.328231	-0.612543	-3.338485	-2.509542	-1.924260	
1986/9/15	0.000000	0.000000	3.370829	2.471821	0.502510	
1986/9/16	-2.549195	-2.549195	-6.083012	-6.060104	0.618147	
1986/9/17	-3.465108	-2.615883	-0.707001	-0.724141	-0.515231	
1986/9/18	-0.827638	-0.569153	-0.051929	0.810000	0.269786	
1986/9/19	0.953482	0.523740	0.953482	0.257136	-0.652000	
1986/9/22	0.194175	0.537955	0.194175	1.038926	1.732274	
1986/9/23	0.000000	-1.021749	0.000000	0.000000	0.000000	
2002/3/14	0.000000	1.739340	-0.597998	1.335818	0.145488	
2002/3/15	1.066830	1.208182	1.666095	0.682176	0.852961	
2002/3/18	1.293378	0.681052	-0.523212	-1.292918	-0.278315	
2002/3/19	-1.267592	0.028499	1.041638	2.528475	0.542122	
2002/3/20	2.015356	0.348329	-0.814408	-2.281785	-1.264916	
2002/3/22	-3.202856	-2.691734	-1.555618	-1.588887	-0.706196	
2002/3/25	-1.069226	-1.230661	-1.416454	-0.743077	-1.410015	
2002/3/26	-1.106807	1.270158	-0.017194	-0.473274	0.694845	
2002/3/27	0.333396	-0.899719	0.227146	1.027542	0.707884	
2002/3/28	0.548272	-0.639374	0.451160	0.083248	-0.220536	
2002/3/29	0.324658	0.363974	-1.940692	-2.756854	0.000000	
2002/4/1	-0.175671	0.120566	0.157126	0.021097	0.207178	
2002/4/2	0.330442	0.614694	0.382816	1.581363	-0.473877	0.037545
2002/4/3	-0.351609	2.288794	-0.068799	1.736102	-1.125401	0.000000
2002/4/4	2.898319	0.523068	2.627234	-0.188849	0.360976	-0.075103
2002/4/5	-0.560168	-1.081832	-0.302587	-0.384861	0.355684	-0.602865
2002/4/8	-0.397750	0.149193	-0.325877	0.153377	-0.219871	-0.568506
2002/4/9	0.143079	-0.586404	-1.359372	-2.122262	-0.395060	0.000000
2002/4/10	-2.214962	-0.618197	-0.544804	0.932166	1.681022	-0.648239
2002/4/11	1.807425	0.243214	0.851982	-0.637676	-2.000770	0.305577
2002/4/12	-1.981531	-1.763593	-2.278781	-1.667040	0.144747	0.342662
2002/4/15	-0.508706	0.143123	0.418442	1.577563	-0.957886	0.492892
2002/4/16	1.328029	1.849789	1.806500	1.862362	2.036359	-0.265101
2002/4/17	2.318947	1.731428	2.328957	1.721725	-0.784914	-0.684934
2002/4/18	0.672095	0.786434	0.710390	0.277001	-0.151767	-0.267635
2002/4/19	-0.098139	-0.950713	-0.860192	-0.551980	0.506500	-0.383583
2017/7/11	0.017038	0.362627	0.237647	0.568572	0.002573	0.087573
2017/7/12	0.315144	-0.237111	-0.047393	-0.481958	0.573205	-0.456221
2017/7/13	0.198731	0.152524	0.005530	0.007113	0.097245	-0.396494
2017/7/14	-0.095992	-0.099787	0.196402	0.094726	0.391984	0.229277
2017/7/18	-0.415138	-0.409398	-0.791968	-0.592987	-0.291628	-0.973631
2017/7/19	-0.517571	-0.243448	0.020652	0.104692	0.305537	-0.392122
2017/7/20	-0.380438	-0.690304	0.405072	0.616106	-0.122861	-0.071460
2017/7/21	0.214063	-0.108106	0.247692	-0.222838	-0.146829	0.116097

パターン2

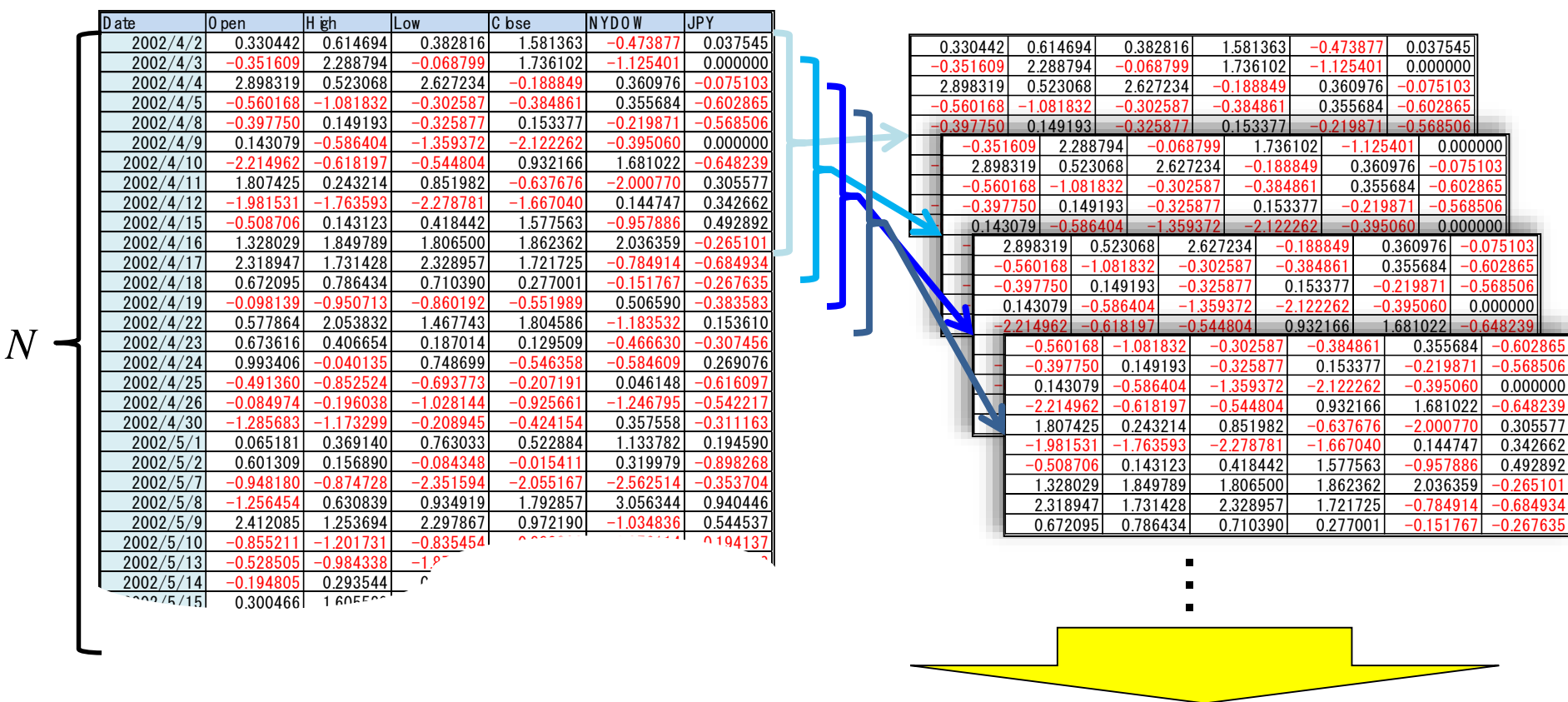
パターン1

1986/9/4~
2017/7/20

2002/4/2~
2017/7/20

学習データの作成

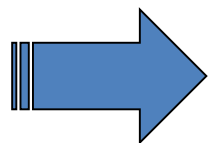
系列データを入力系列数(Input_length)ずつ切り出してテンソルに整形する



(N-Input_length+1, Input_length, 6)
次元テンソル

出力次元について

- 騰落予測だけでは、応用上の有用性はあまりないので、どれくらい増減するかも予測できるようにしたい
- sinカーブの予測と同じやり方で、終値の増減値それ自体を予測することも可能だが、使用に耐える精度が得られる可能性は低い



次ページへつづく

出力次元について

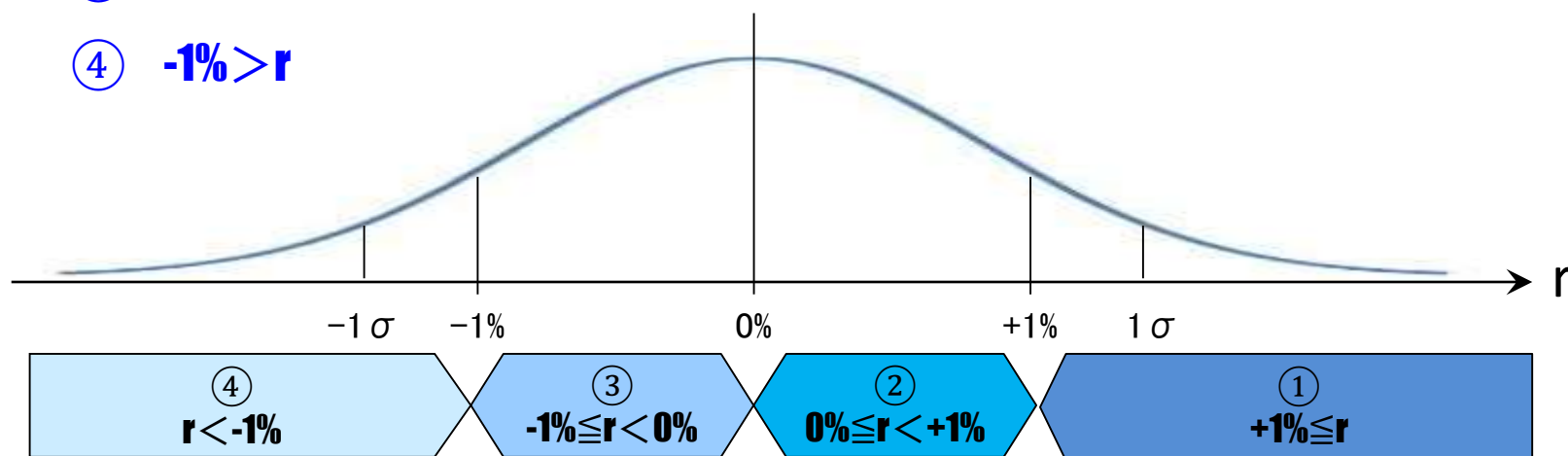
- そこで、終値の増減率を、 $\pm 1\%$ という値を閾値として以下の4カテゴリに分類することによって、前ページの懸案解決を意図した。増減率を r とするとき、

① $r \geq +1\%$

② $+1\% > r \geq 0\%$

③ $0\% > r \geq -1\%$

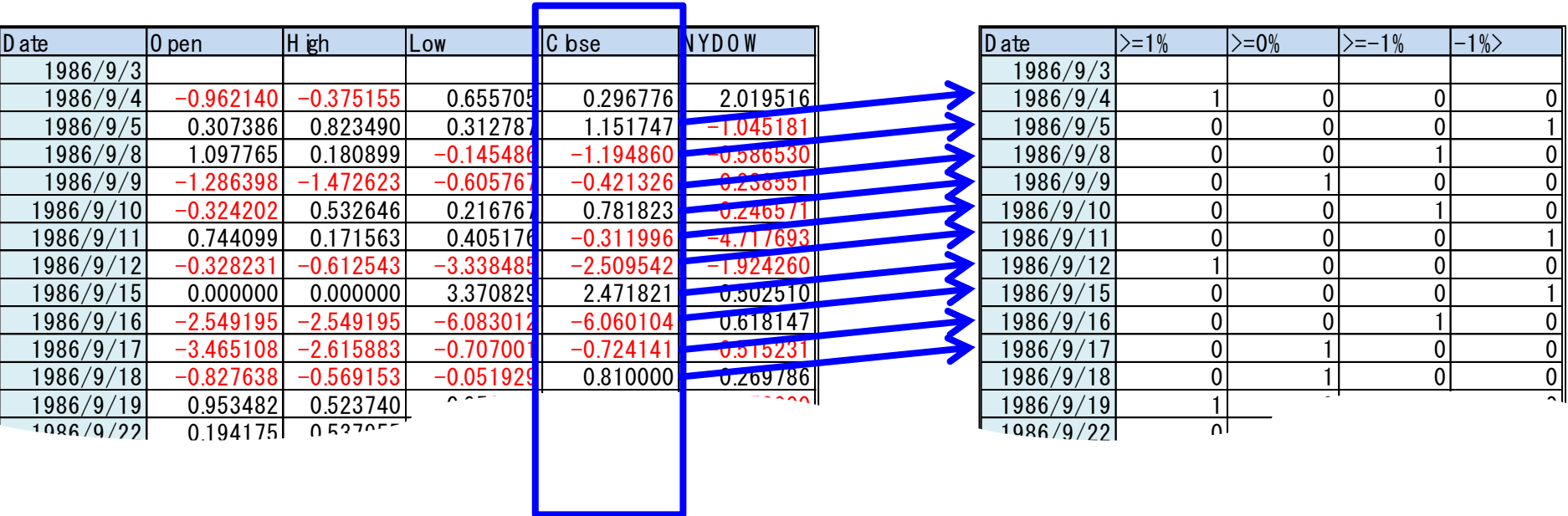
④ $-1\% > r$



- $\pm 1\%$ という値は、切れの良さと発生頻度の多さ(0.66σ)から選択した。なお、増減率 r の平均はほぼ0である

ラベルデータの作成

「翌日」の終値を閾値 ($\log(101/100) \times 100 = 0.995033085$) に従って分類し、ラベル付けする。

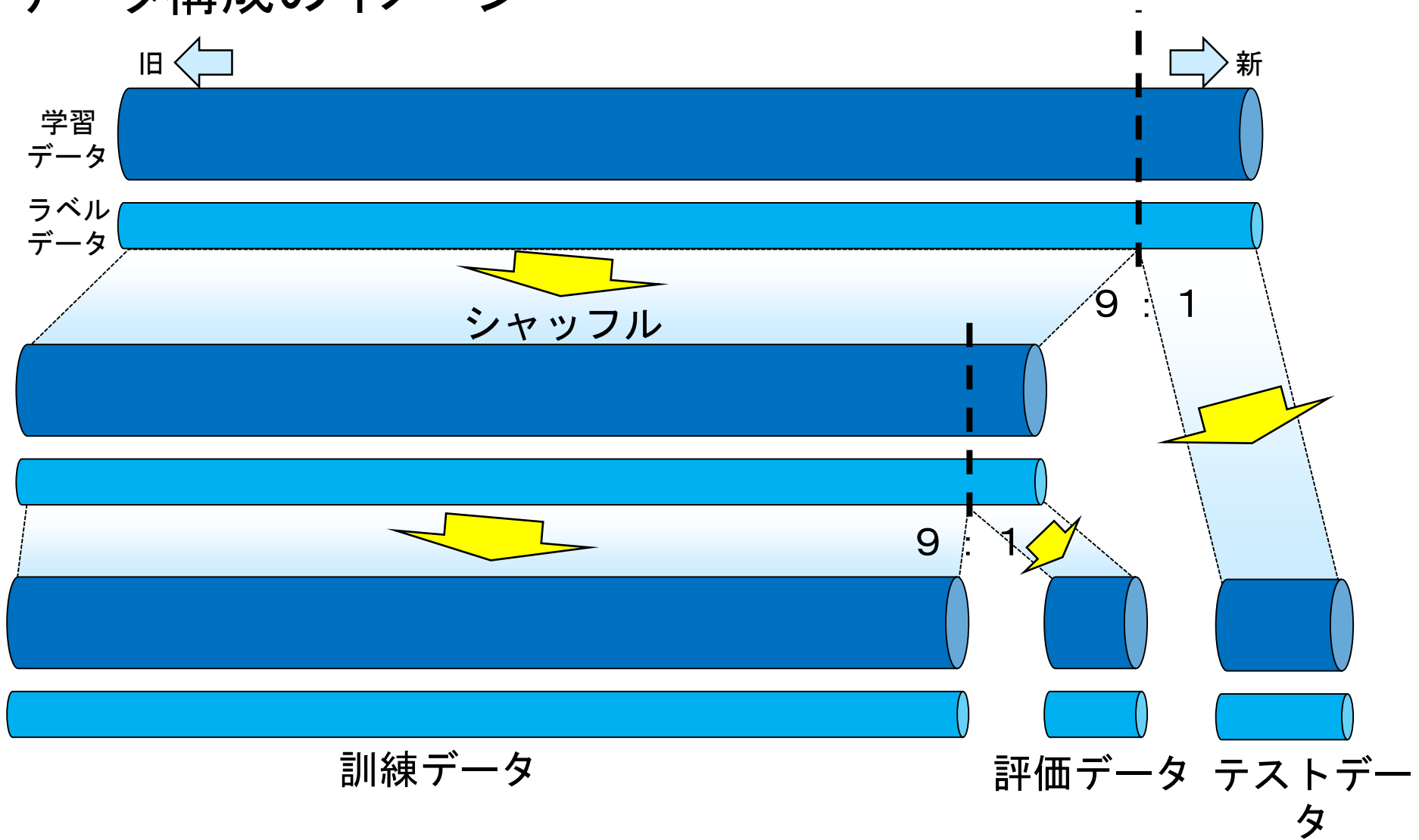


訓練データ、評価データ、テストデータの構成

- 初期状態では、学習データおよびラベルデータは時間の昇順(つまり古い順)に並べてある
- これを9:1に分割し、新しいほうの1割をテストデータに使用する
- 古いほうの9割はシャッフルし、その1割を評価データに充て、残りを訓練データに使用した

訓練データ、評価データ、テストデータの構成

データ構成のイメージ



乱数生成に対する**seed**指定

- 重み行列の初期値用の乱数には、seed指定により、毎回同じ値が生成されるようにした
- Dropoutに対しては、seedを指定すると学習性能が悪くなったので、seed指定はしていない。


学習の打ち切り方法

過学習を防ぐため、fitメソッドのcallbacks引数にEarlyStoppingを指定して、学習を停止するようにした

EarlyStoppingとは

学習に伴い損失関数は減少するが、評価データを使って算出した損失 val_loss は却って増加することがある。このような場合に学習を止める仕組み。

fitメソッドで以下のように設定する

```
earlystopping=keras.callbacks.EarlyStopping( patience=0, verbose=1)
fit(self, x, y, batch_size=32,
    epochs=10, verbose=1,
    callbacks=[earlystopping], 
    validation_split=0.0, validation_data=None,
    shuffle=True, class_weight=None,
    sample_weight=None, initial_epoch=0)
```

パターン1データ（JPYあり、レコード量が少ないほう）での結果

評価対象パラメータ

パラメータ	説明	デフォルト値
units	出力の次元数	—
Input_length	入力系列の長さ	—
<ul style="list-style-type: none">• kernel_regularizer• recurrent_regularizer• bias_regularizer• activity_regularizer	出力テンソルに対する正則化の有無	無し
<ul style="list-style-type: none">• kernel_constraint• recurrent_constraint• recurrent_constraint	出力テンソルに対する制約の有無	無し
<ul style="list-style-type: none">• Dropout• recurrent_dropout	ドロップアウトの比率	0
lr	Optimizer(RMSprop) の学習率	0.001

パターン1データでの結果

LSTMの固定パラメータ

- input_dim(入力次元数): 6(日経平均(始値、高値、安値、終値)、NYダウ終値、ドル円)
- use_bias: True
- bias_initializer: (biasベクトルのInitializer): 'zeros'
- unit_forget_bias: True

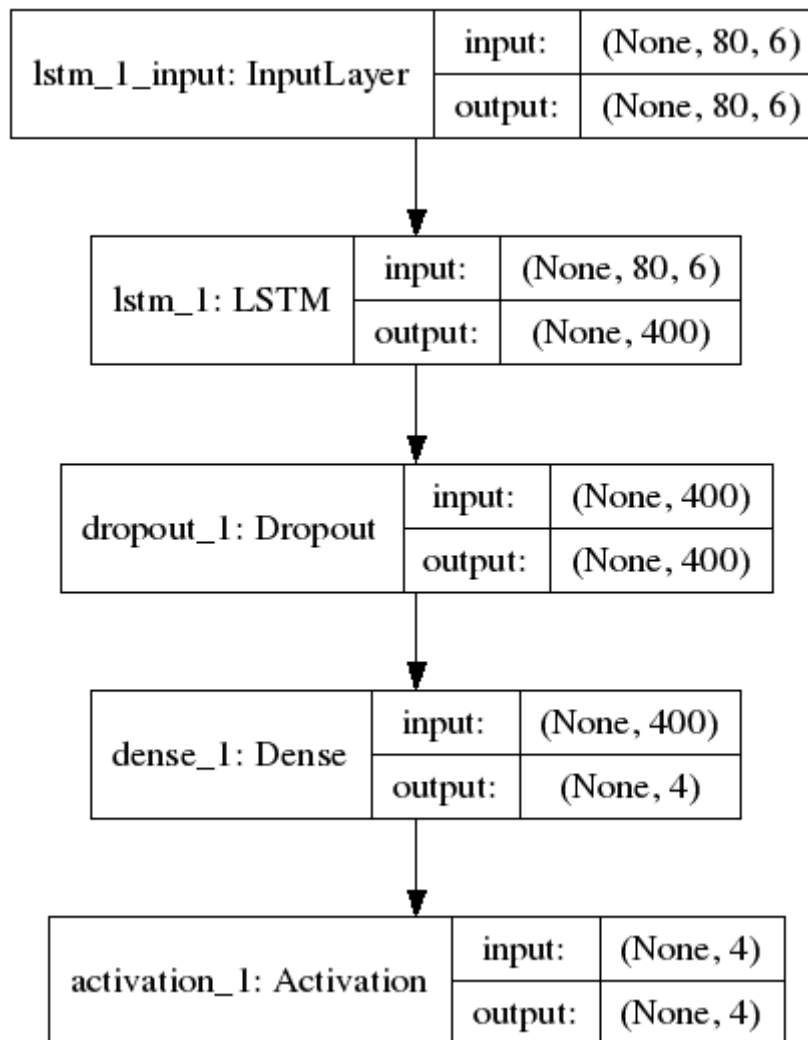
その他の固定パラメータ

- optimizer: 'RMSprop'
- ミニバッチのサイズ: 10
- patience(EarlyStopping): 0

上記以外のパラメータはすべてデフォルト

パターン1データでの結果

入力系列数80、出力次元400の場合のグラフ



パターン1データでの結果

units(出力次元)



units	input_length	test loss	test accuracy	準正解率(騰落)
200	80	1.28646299	0.32520326	0.56097561
400	80	1.28687992	0.39837399	0.60162602

- 入力系列数は80固定で評価した。
- 80という数字にfeatureしたのは、「75日移動平均」という株価指標があるため。
- 出力次元数が多いほうが良い結果が出る。
- 結果は載せていないが、出力次元が400以上の場合も実験してみた。しかし、400を超えて大きくしても、計算コストの割に精度は良くならなかった。

※準正解率:単純な騰落予想(上がるか／下がるか)の正解率。正解が「1%超値上がり」の場合において、推測結果「1%未満の値上がり」は不正解だが、騰落予測の観点からは正解なので、これをカウントした正解率

パターン1データでの結果

入力系列数と出力次元



Input_ length	units	test loss	test accuracy	準正解率(騰 落)
20	100	1.29075358	0.29457365	0.51162791
40	200	1.30860280	0.35433072	0.60629921
80	400	1.28687992	0.39837399	0.60162602

- 入力系列数と出力次元を連動して変化させた。
- 入力系列が長すぎるとかく乱要因になるのではと考えたが、値が大きいほうが良い結果が得られた。

パターン1データでの結果

ドロップアウト



Dropout recurrent_dr opout	test loss	test accuracy	準正解率(騰 落)
0.7	1.29998663	0.34959351	0.50406504
0.5	1.28687992	0.39837399	0.60162602
0.3	1.26966589	0.34146343	0.51219512

ドロップアウトは、大きすぎても小さすぎても良い結果は得られなかった

パターン1データでの結果

lr(学習率)



lr	test loss	test accuracy	準正解率(騰落)
0.0005	1.30964265	0.30894310	0.52032520
0.0001	1.29191087	0.34959351	0.51219512
0.001	1.28687992	0.39837399	0.60162602

学習率0.001はoptimizer 'RMSprop'のデフォルト値である。

学習率を小さくすれば、より高い精度に追い込めると考えたが、そうではなかった。

パターン1データでの結果

Regularizer

- 過学習を防ぐために、損失関数に重み行列のノルムを付け加えて学習を実施する、「正則化」という手法があるが、Regularizerとはそのノルム項目
- 重み行列のノルムには、以下の3種類がある
 - L2:いわゆるL2ノルム
 - L1:いわゆるL1ノルム。各要素の絶対値の和
 - L0:非零要素の数
- Regularizerの指定値は、ノルムにかける係数
- 今回の試行では、L2ノルム決め打ちで評価を行った

パターン1データでの結果

Regularizer (L2正則)



regularizers	test loss	test accuracy	準正解率(騰落)
0.1	1.59390535	0.09756098	0.49593496
0.01	1.40690100	0.38211383	0.50406504
0.001	1.34301223	0.34959351	0.52845528
無し	1.28687992	0.39837399	0.60162602

今回の問題に関してはRegularizerを使用しないほうが、良い結果が得られた

パターン1データでの結果

制約

- Kerasには、学習の過程で重み行列の要素の値が野放図に大きくなることを避けるための、「制約 (constraints)」というパラメータがある。
- 制約には以下の3種類がある。
 - `maxnorm(max_value=2, axis=0)`: 最大値ノルム制約
 - `non_neg()`: 非負値制約
 - `unit_norm()`: ノルム正規化制約, 行列の最後のaxisのノルムで正規化
- 今回の試行では、`maxnorm`決め打ちで、`max_value`の値をいろいろ変化させて、その効果を見た。

パターン1データでの結果

制約(maxnorm)



max_value (maxnorm)	test loss	test accuracy	準正解率(騰 落)
2	1.29055099	0.40650407	0.60162602
1	1.28116846	0.41463416	0.54471545
0.5	1.27496684	0.40650408	0.49593496
無し	1.28687992	0.39837399	0.60162602

正解率がわずかに改善しているが、準正解率は同等か、かえって悪くなった。損失関数と正解率の双方が改善しているパターンは無かったので、制約は使用しないことにした。

パターン1データでの結果

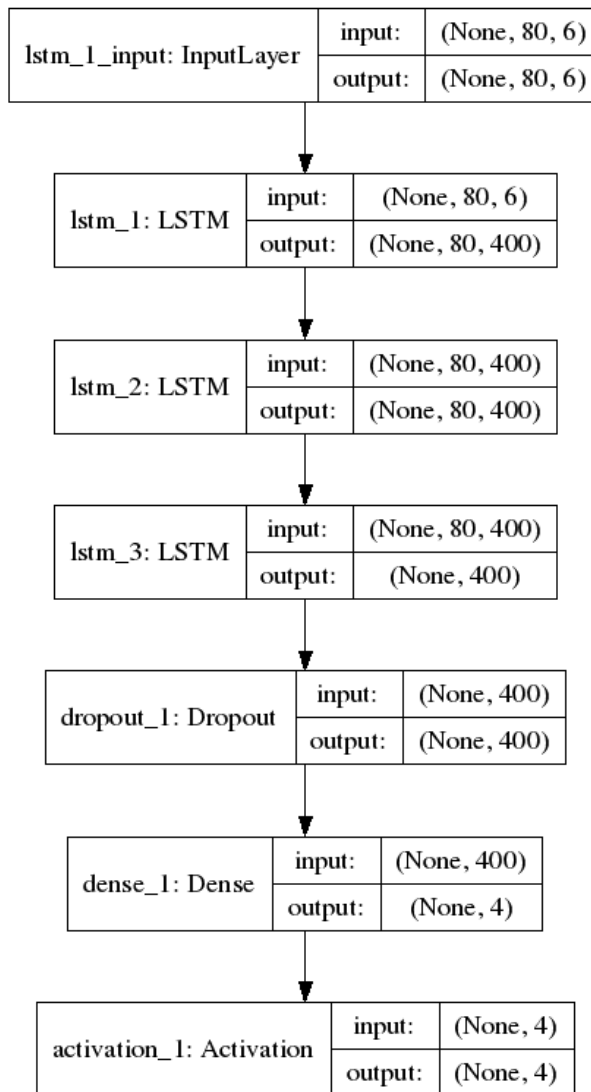
Stacked LSTM

KerasではLSTMを簡単に多段に積むことができるので、効果を試してみた。

パターン1データでの結果

Stacked LSTM

Stacked LSTMのグラフ(スタック数=3)



パターン1データでの結果

Stacked LSTM



スタック数	test loss	test accuracy	準正解率(騰落)
4	1.34761956	0.30081302	0.47967480
3	1.32311327	0.37398375	0.51219512
2	1.31537145	0.34959351	0.48780488
1	1.28687992	0.39837399	0.60162602

実は効果を期待していたが、スタックをたくさん積みばいいというわけではなかった

パターン1データでの結果

ここまでの結果をまとめると以下の通り。以下のパラメータの場合に、最も良い結果を得られた。

パラメータ	説明	設定値
units	出力の次元数	400
Input_length	入力系列の長さ	80
<ul style="list-style-type: none">• kernel_regularizer• recurrent_regularizer• bias_regularizer• activity_regularizer	出力テンソルに対する正則化の有無	無し【デフォルト値】
<ul style="list-style-type: none">• kernel_constraint• recurrent_constraint• recurrent_constraint	出力テンソルに対する制約の有無	無し【デフォルト値】
<ul style="list-style-type: none">• Dropout• recurrent_dropout	ドロップアウトの比率	0.5
lr	Optimizer(RMSprop) の学習率	0.001 【デフォルト値】
Stacked LSTM	LSTM の多段構成	無し

パターン2 データ (**JPY**無し、レコード数の多いほう) の結果

使用したデータ以外のパラメータは同一条件で、比較を実施した



データ	test loss	test accuracy	準正解率(騰落)
パターン2	1.26204092	0.34840426	0.53324468
パターン1	1.28687992	0.39837399	0.60162602

パターン2はレコード数がパターン1の2倍あるにもかかわらず、精度はパターン1のほうが良い。

入力データの拡充

ここまで評価したところで、ネット上の成果を検索したところ、株価の対数値を使用した事例が既に存在した。

<http://qiita.com/akiraak/items/b27a5616a94cd64a8653> 0 - Google の
サンプルコードを動かしてみる

その事例では、NYダウ、FTSE、DAX指数、ハンセン指数、日経平均、NASDAQ、SP500、上海総合指数の8指数の終値をもとに翌営業日のSP500騰落を予測していた。(MLP(多層パーセプトロン)、入力系列数は3日分)

入力データの拡充

そこで、この事例に倣って、入力データに以下の指数を追加した。

DAX指数、ハンセン指数、TOPIX、NASDAQ、SP500、上海総合指数

(FTSEは過去データを手に入できなかったなので、代わりにTOPIXを追加した)

これにより、入力次元数input_dimは以下の12となる

- | | |
|------------|----------|
| ① 日経平均(始値) | ⑦ ハンセン指数 |
| ② 日経平均(高値) | ⑧ TOPIX |
| ③ 日経平均(安値) | ⑨ NASDAQ |
| ④ 日経平均(終値) | ⑩ SP500 |
| ⑤ NYダウ | ⑪ 上海総合指数 |
| ⑥ DAX指数 | ⑫ ドル円 |

入力データの拡充

最適パラメータの選択方法

各パラメータの評価結果として得られる以下の各数字を尺度として評価する。一番良い結果が出た尺度が最も多いパラメータを最適パラメータとして選択する。

尺度	意味	評価軸
test loss	テストデータに対する損失	小さいほうが良い
test accuracy	テストデータに対する正解率	大きいほうが良い
準正解率(騰落)	テストデータに対する騰落予測の正解率	大きいほうが良い
正解数	テストデータに対する予測の正解数	大きいほうが良い
的中率	テストデータに対して正解を予測したもののうち、実際に正解だった数	大きいほうが良い

※正解率、的中率は、 $\pm 1\%$ 越え選択枝の結果を評価対象とする

※最良尺度数が同数の場合は、「的中率」の優先度を下げて評価する

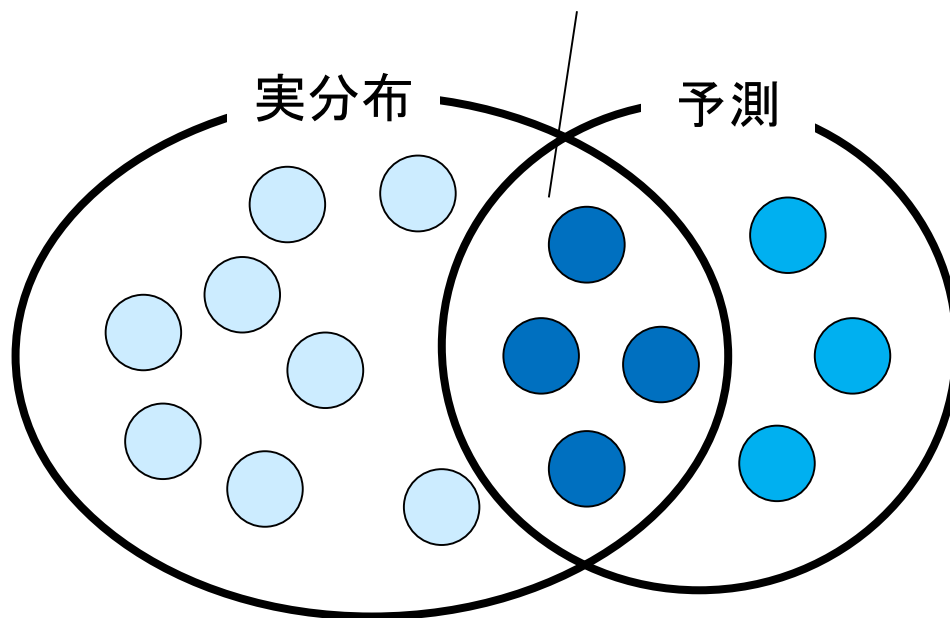
入力データの拡充

正解率と的中率

正解率 = 正解数 / 実分布数

的中率 = 正解数 / 予測数

正解 = 実分布 \cap 予測



上図の例の場合、正解率 = $4/12 = 0.33$ 、的中率 = $4/7 = 0.57$

拡充データでの結果

入力系列数と出力次元(1)

- 入力系列数(input_length)が80、60、40、20、10、5の場合について、評価を実施した。
- 出力次元(units)は入力系列数 \times 入力次元(=12) \times 0.9で算出

拡充データでの結果

入力系列数と出力次元(2)

input_length	units	test loss	test accuracy	準正解率 (騰落)	分類別正解				
						1%以上	0%以上	-1%以上	-1%未満
80	864	1.28486	0.36957	0.60326	実分布	61	133	119	55
					予測結果	94	32	237	5
					正解数	30	13	92	1
					的中率	0.31915	0.40625	0.38819	0.20000
60	648	1.32041	0.37568	0.53243	実分布	62	133	119	56
					予測結果	5	355	0	10
					正解数	4	130	0	5
					的中率	0.80000	0.36620	0.00000	0.50000
40	432	1.26887	0.41935	0.61290	実分布	62	134	119	57
					予測結果	28	218	121	5
					正解数	14	92	49	1
					的中率	0.50000	0.42202	0.40496	0.20000
20	216	1.25308	0.39572	0.57487	実分布	62	134	120	58
					予測結果	50	23	289	12
					正解数	25	13	106	4
					的中率	0.50000	0.56522	0.36678	0.33333
10	108	1.25481	0.43733	0.60000	実分布	62	134	120	59
					予測結果	65	133	145	32
					正解数	26	61	62	15
					的中率	0.40000	0.45865	0.42759	0.46875
5	54	1.24710	0.42133	0.61067	実分布	62	134	120	59
					予測結果	47	189	104	35
					正解数	18	80	44	16
					的中率	0.38298	0.42328	0.42308	0.45714
80	400	1.28688	0.39837	0.60163					



※各列ごとに、最も結果の良いセルを着色してある

データパターン1の結果

拡充データでの結果

入力系列数と出力次元(3)

- データの種類が多いと、精度が若干改善する(でも思ったほどではない)
- 入力系列数が40以下のほうが良い結果が出ている。一番結果が良かったのは入力系列数=5の場合

拡充データでの結果

Patience(EarlyStopping)



input_length	units	Patience	test loss	test accuracy	準正解率 (騰落)	分類別正解				
							1%以上	0%以上	-1%以上	-1%未満
5	54	0	1.24710	0.42133	0.61067	実分布	62	134	120	59
						予測結果	47	189	104	35
						正解数	18	80	44	16
						的中率	0.38298	0.42328	0.42308	0.45714
5	54	5	1.23869	0.42400	0.62933	実分布	62	134	120	59
						予測結果	36	167	128	34
						正解数	20	71	55	13
						的中率	0.55556	0.42515	0.42969	0.38235
5	54	10	1.24373	0.44800	0.63467	実分布	62	134	120	59
						予測結果	54	163	120	38
						正解数	24	73	54	17
						的中率	0.44444	0.44785	0.45000	0.44737
5	54	15	1.23979	0.42667	0.61867	実分布	62	134	120	59
						予測結果	42	191	111	31
						正解数	20	81	47	12
						的中率	0.47619	0.42408	0.42342	0.38710
5	54	20	1.22939	0.42133	0.60533	実分布	62	134	120	59
						予測結果	54	102	186	33
						正解数	26	46	72	14
						的中率	0.48148	0.45098	0.38710	0.42424

Patience=10の 때가最も結果が良い。

拡充データでの結果

入力系列数再評価 (Patience=10)

input_length	units	Patience	test loss	test accuracy	準正解率 (騰落)	分類別正解				
							1%以上	0%以上	-1%以上	-1%未満
5	54	10	1.24373	0.44800	0.63467	実分布	62	134	120	59
						予測結果	54	163	120	38
						正解数	24	73	54	17
						的中率	0.44444	0.44785	0.45	0.44737
10	108	10	1.22348	0.44267	0.61067	実分布	62	134	120	59
						予測結果	47	187	95	46
						正解数	24	81	43	18
						的中率	0.51064	0.43316	0.45263	0.39130
20	216	10	1.22429	0.47059	0.63102	実分布	62	134	120	58
						予測結果	63	167	101	43
						正解数	29	81	49	17
						的中率	0.46032	0.48503	0.48515	0.39535
40	432	10	1.24741	0.43280	0.65323	実分布	62	134	119	57
						予測結果	74	153	80	65
						正解数	30	74	37	20
						的中率	0.40541	0.48366	0.46250	0.30769



入力系列数=40のときが最も結果が良い。

拡充データでの結果

最も結果が良かったパラメータ値は以下の通り。これ以外のパラメータはパターン1と同じ

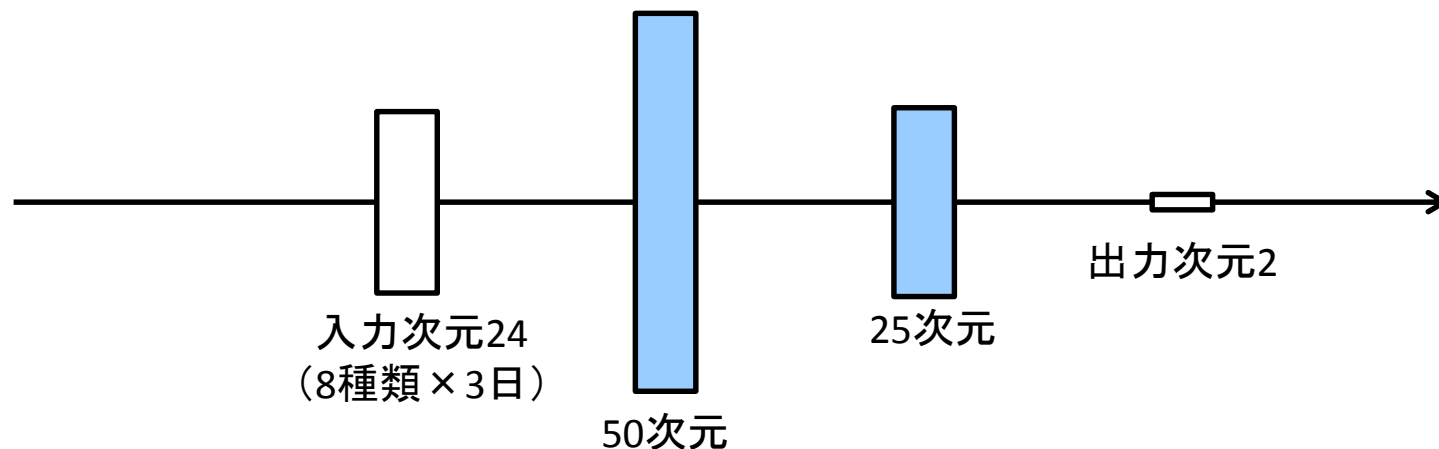
パラメータ	説明	設定値
units	出力の次元数	432
Input_length	入力系列の長さ	40
patience	val_loss の値が改善しなくなったときの、学習打ち切りの閾値	10

MLP との比較

ネット上の事例のMLPモデル

ネット上の事例に倣って、MLPでのやり方を今回のデータパターンに適用し、MLPとLSTMの比較を行った。

ネット上の事例のモデルは以下の通り。



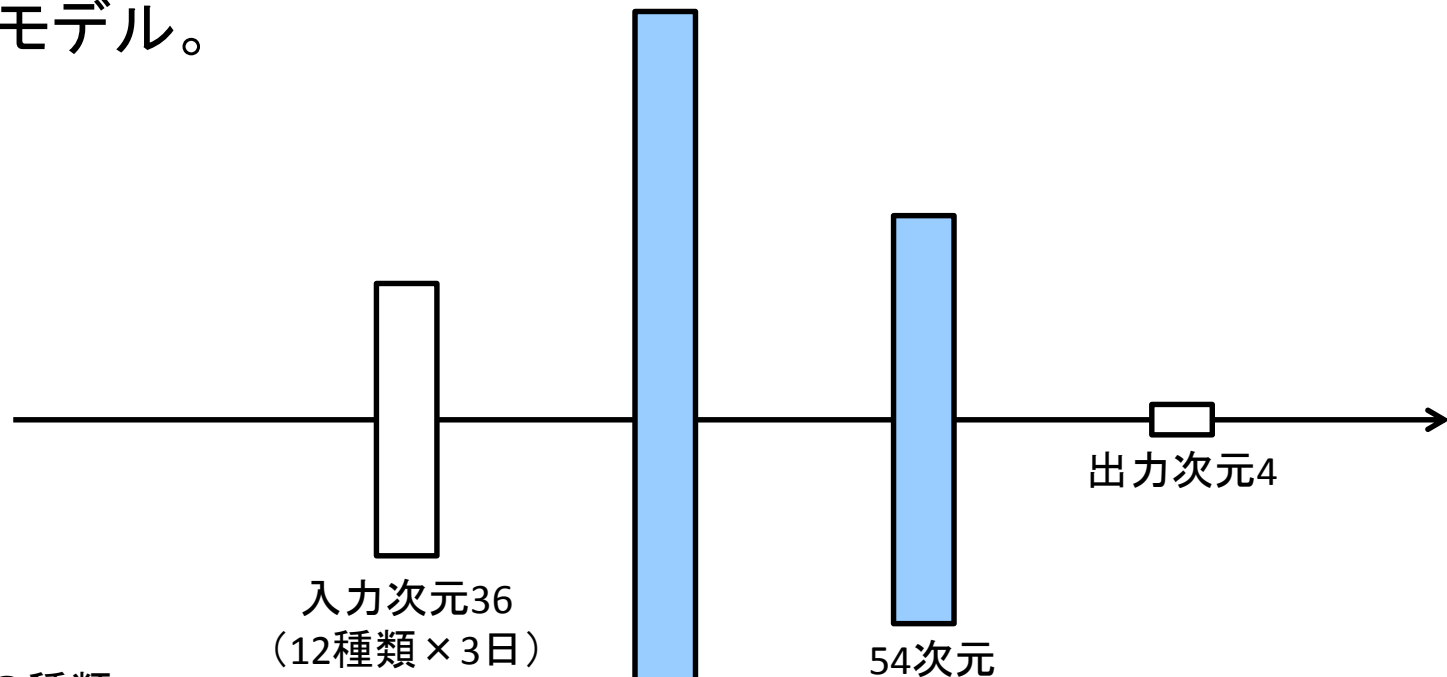
入力データの種類

- | | |
|----------|----------|
| ① NYダウ | ⑤ 日経平均 |
| ② FTSE | ⑥ NASDAQ |
| ③ DAX指数 | ⑦ SP500 |
| ④ ハンセン指数 | ⑧ 上海総合指数 |

MLP との比較

今回実施のMLPモデル

入力対象日数=3、5、10の場合について評価した。図は入力対象日数3のモデル。



入力データの種類

- | | |
|------------|----------|
| ① 日経平均(始値) | ⑦ ハンセン指数 |
| ② 日経平均(高値) | ⑧ TOPIX |
| ③ 日経平均(安値) | ⑨ NASDAQ |
| ④ 日経平均(終値) | ⑩ SP500 |
| ⑤ NYダウ | ⑪ 上海総合指数 |
| ⑥ DAX指数 | ⑫ ドル円 |

M L P との比較

パラメータ設定

LSTM固有パラメータ以外の以下のパラメータは、LSTM試行時の値に合わせた

- dropout:0.5
- patience:10

MLP との比較

グラフ

dense_1_input: InputLayer	input:	(None, 36)
	output:	(None, 36)



dense_1: Dense	input:	(None, 36)
	output:	(None, 108)



activation_1: Activation	input:	(None, 108)
	output:	(None, 108)



dropout_1: Dropout	input:	(None, 108)
	output:	(None, 108)



dense_2: Dense	input:	(None, 108)
	output:	(None, 54)



activation_2: Activation	input:	(None, 54)
	output:	(None, 54)



dropout_2: Dropout	input:	(None, 54)
	output:	(None, 54)



dense_3: Dense	input:	(None, 54)
	output:	(None, 4)



activation_3: Activation	input:	(None, 4)
	output:	(None, 4)

MLP との比較

モデル	input_length	input	units1	units2	test loss	test accuracy	準正解率 (騰落)	分類別正解				
									1%以上	0%以上	-1%以上	-1%未満
MLP	-	12 × 3	108	54	1.28752	0.42819	0.61968	実分布	63	134	120	59
								予測結果	38	238	75	25
								正解数	19	102	30	10
								的中率	0.50000	0.42857	0.40000	0.40000
MLP	-	12 × 5	180	90	1.28784	0.41867	0.59467	実分布	62	134	120	59
								予測結果	30	208	116	21
								正解数	16	84	49	8
								的中率	0.53333	0.40385	0.42241	0.38095
MLP	-	12 × 10	360	180	1.36357	0.39200	0.57600	実分布	62	134	120	59
								予測結果	30	249	69	27
								正解数	14	97	26	10
								的中率	0.46667	0.38956	0.37681	0.37037
LSTM	40	12	432	-	1.24741	0.43280	0.65323	実分布	62	134	119	57
								予測結果	74	153	80	65
								正解数	30	74	37	20
								的中率	0.40541	0.48366	0.46250	0.30769

※units1は1段目denseのunits、units2は2段目denseのunitsである。LSTMのunitsはunits1の欄に記載してある。

意外なことに、入力対象日数が大きいほど精度が下がる。また、LSTMのほうが良い結果を得られる。

5営業日後の株価騰落予測

- 翌営業日の騰落予想では、応用上の意義はあまり大きくない。そこで、より未来の予測が可能かどうか、評価を実施する。
- 手法は翌日予測と同じ。正解データを5営業日後の値に入れ替えて、学習を実施する。使用データは拡充データ(入力次元12、期間は2002/4/2～2017/7/20)。
- ただし、5日後だと増減の振れ幅が大きくなるので、出力分類を±3%以上の増減とそれ以外の4種類とした。±3%というのは、約 0.93σ である。
- 5日という期間設定にあまり意味はない。あまりに先の未来にすると予測精度が出ないだろうと考えたのと、切れの良さから選択した。

5 営業日後の株価騰落予測

パラメータ設定は、以下のこれまでの結果を使用する。Input_lengthおよびunitsを変化させて、推定精度の変化を見る。これら以外のパラメータはデフォルトのままとする。

パラメータ	説明	設定値
<ul style="list-style-type: none">• Dropout• recurrent_dropout	ドロップアウトの比率	0.5
patience	val_loss の値が改善しなくなったときの、学習打ち切りの閾値	10
Stacked LSTM	LSTM の多段構成	無し

5営業日後の株価騰落予測



input_length	units	test loss	test accuracy	準正解率 (騰落)	分類別正解				
						3%以上	0%以上	-3%以上	-3%未満
80	864	1.18140	0.44142	0.56948	実分布	49	156	130	32
					予測結果	7	174	174	12
					正解数	5	83	69	5
					的中率	0.71429	0.47701	0.39655	0.41667
60	648	1.16467	0.44173	0.55014	実分布	49	157	131	32
					予測結果	19	305	37	8
					正解数	11	137	12	3
					的中率	0.57895	0.44918	0.32432	0.37500
40	432	1.17119	0.46900	0.57412	実分布	49	157	131	34
					予測結果	13	317	34	7
					正解数	9	146	17	2
					的中率	0.69231	0.46057	0.50000	0.28571
20	216	1.17978	0.44504	0.56568	実分布	49	157	132	35
					予測結果	8	336	23	6
					正解数	5	149	10	2
					的中率	0.62500	0.44345	0.43478	0.33333
10	108	1.15433	0.44920	0.56417	実分布	49	157	132	36
					予測結果	7	308	51	8
					正解数	4	137	20	7
					的中率	0.57143	0.44481	0.39216	0.87500
5	54	1.18270	0.42400	0.54133	実分布	49	157	132	37
					予測結果	5	347	17	6
					正解数	4	148	5	2
					的中率	0.80000	0.42651	0.29412	0.33333

5営業日後の株価騰落予測

考察

- lossおよびaccuracyは翌日予想よりやや良い数字が出ているが、よく見ると、「 $+1\% > r \geq 0\%$ (r は増減率)」への1点張りで数字を稼いでいることがわかる。
- 準正解率(騰落)は翌日予測より悪い結果が出た。
- 一番良い結果が出たのは、入力系列=10のパターン。

5営業日後の株価騰落予測

±3%超予測時の正解率

- 予測が±3%超の時の的中率が結構よい。6割を超えているケースもある。
- ±3%超の予測が出たとき、6割の確率でそれが実現するというのだから、5日後の予測であることを考えると、結構驚きの結果である。
- 次ページに、正解率が6割を超えたケースをマーキングした表を示す。

5営業日後の株価騰落予測

±3%超予測時の的中率が6割を超えたケース

input_length	units	val_loss	正解率	準正解率 (騰落)	分類別正解				
						3%以上	0%以上	-3%以上	-3%未満
80	864	1.18140	0.44142	0.56948	実分布	49	156	130	32
					予測結果	7	174	174	12
					正解数	5	83	69	5
					的中率	0.71429	0.47701	0.39655	0.41667
60	648	1.16467	0.44173	0.55014	実分布	49	157	131	32
					予測結果	19	305	37	8
					正解数	11	137	12	3
					的中率	0.57895	0.44918	0.32432	0.37500
40	432	1.17119	0.46900	0.57412	実分布	49	157	131	34
					予測結果	13	317	34	7
					正解数	9	146	17	2
					的中率	0.69231	0.46057	0.50000	0.28571
20	216	1.17978	0.44504	0.56568	実分布	49	157	132	35
					予測結果	8	336	23	6
					正解数	5	149	10	2
					的中率	0.62500	0.44345	0.43478	0.33333
10	108	1.15433	0.44920	0.56417	実分布	49	157	132	36
					予測結果	7	308	51	8
					正解数	4	137	20	7
					的中率	0.57143	0.44481	0.39216	0.87500
5	54	1.18270	0.42400	0.54133	実分布	49	157	132	37
					予測結果	5	347	17	6
					正解数	4	148	5	2
					的中率	0.80000	0.42651	0.29412	0.33333

5営業日後の株価騰落予測

input_length別の的中状況

date	正解	input_length						的中数	当日の出来事
		80	60	40	20	10	5		
2016/1/21	+3%超増	○	○	○		○	○	5	
2016/1/22	+3%超増	○	○					2	
2016/1/26	+3%超増		○		○	○	○	4	
2016/1/29	-3%超減					○		1	マイナス金利政策決定
2016/2/1	-3%超減	○		○		○		3	
2016/2/2	-3%超減	○	○	○	○	○	○	6	
2016/2/3	-3%超減	○				○		2	
2016/2/4	-3%超減	○				○		2	
2016/2/5	-3%超減	○	○		○	○	○	5	
2016/2/8	-3%超減		○			○		2	
2016/2/10	+3%超増		○	○				2	
2016/2/12	+3%超増	○	○	○			○	4	
2016/3/1	+3%超増		○	○	○	○		4	
2016/4/8	+3%超増		○					1	
2016/4/12	+3%超増			○				1	
2016/4/18	+3%超増				○			1	
2016/5/2	+3%超増	○	○	○	○	○	○	6	
2016/6/16	+3%超増		○	○	○			3	
2016/7/8	+3%超増	○	○	○				3	
2016/11/9	+3%超増		○	○				2	米国大統領選挙

同じロジック、同じデータを使っているのに、入力系列数が違うだけで正解日の予測にこれほどばらつきが出るのは意外であった

5営業日後の株価騰落予測

±3%超予測時の正解率

①複数モデルの予測が一致した場合の正解率

モデルごとの予測がばらついたが、複数モデルの予測が一致した場合には正解率が高まるか、検証した。

予測が一致したモデルの数ごとの的中率は、以下の通り。予想通り、複数のモデルの予測結果が一致した場合は的中する可能性が高い。

予測一致数	予測数	正解数	不正解数	的中率
1	17	4	13	23.5%
2	11	6	5	54.5%
3	5	3	2	60.0%
4	5	3	2	60.0%
5	4	2	2	50.0%
6	2	2	0	100.0%

5営業日後の株価騰落予測

±3%超予測時の正解率

②input_length=10モデルと他のモデルの予測が一致した場合の正解率

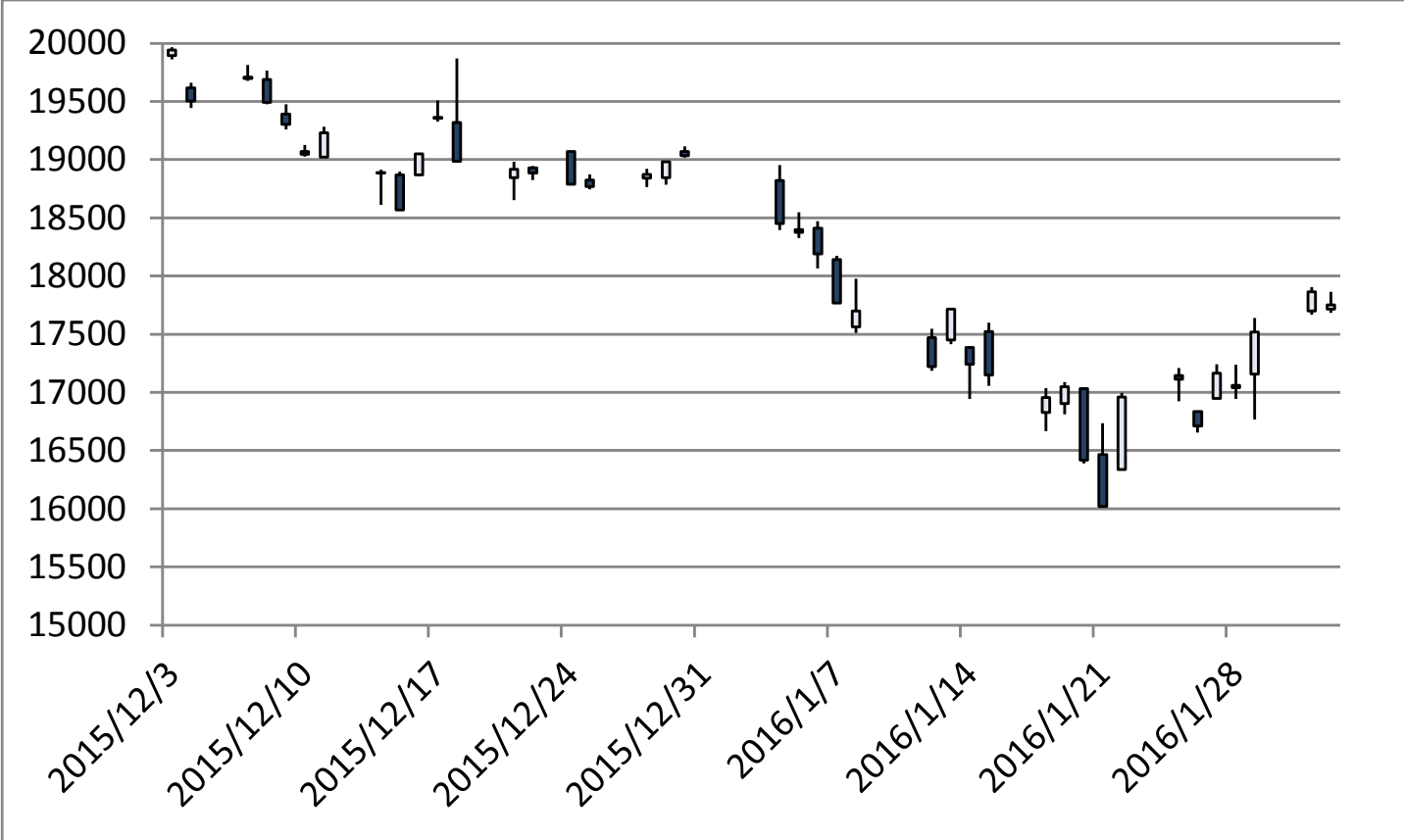
最も成績が良かった、input_length=10のモデルと、他のモデルの予測が一致した場合の正解率は、以下の通り。7割以上の高率になっている。

モデル	一致数	正解数	不正解数	正解率
input_length=80	9	7	2	0.777778
input_length=60	9	7	2	0.777778
input_length=40	6	5	1	0.833333
input_length=20	7	5	2	0.714286
input_length=5	6	5	1	0.833333
上記のいずれか	12	10	2	0.833333

5営業日後の株価騰落予測

±3%超予測「全会一致」時のチャート

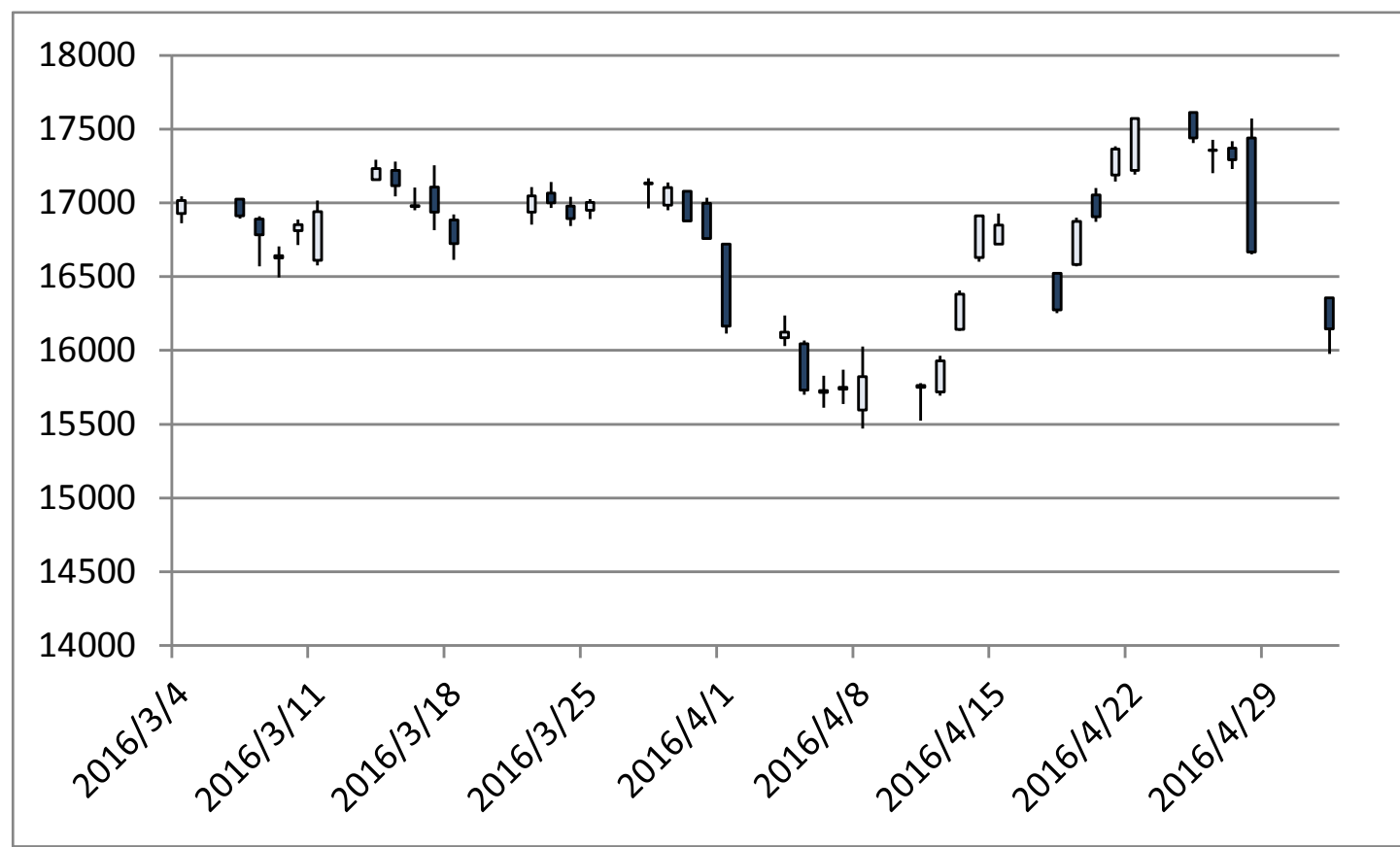
①2016/2/2 -3%減
(2016/2/2終値17750.68、2016/2/9終値16085.44、下落率9.4%)



5営業日後の株価騰落予測

±3%超予測「全会一致」時のチャート

②2016/5/2 +3%増
(2016/5/2終値16147.38、2016/5/12終値16646.34、上昇率3.1%)



5営業日後の株価騰落予測

考察

- 正解率自体は、翌日予測に比べて遜色ない。しかし、一番確率が高そうな選択肢に集中的にbetして数字を稼いでいるだけで、応用上の意味はあまり高くない。
- 準正解率(騰落)は、翌日予測の結果より悪い。
- $\pm 3\%$ 超増減の予測が出ることは少ないが、その予測が出たときの的中率は高い。
- $\pm 3\%$ 超増減予測を複数モデルが一致して出した時の的中率は、さらに高くなる。

まとめ

- LSTMを使って翌営業日の株価騰落予想を実施する場合、入力系列数=40、出力次元=432、dropout=0.5、patience=10と設定するときに、最も良い予測ができた。このときの正解率は約43%、単純騰落予想正解率は約65%。
- 学習データは時系列方向にデータを増やすよりも、データの種類を増やしたほうが効果があった。
- 5営業日先の騰落予想では、入力系列数=10、出力次元=108、dropout=0.5、patience=10と設定するときに、最も良い予測ができた。そのときの正解率は翌営業日予測と同程度。ただし、特定の選択肢に集中的にbetして数字を稼いでいるだけなので、応用上の意味はあまり高くない。
- 5営業日先の騰落予想において、±3%超増減の予測が出ることは少ないが、これが出たときの的中率は高い。複数のモデルが一致して予測した場合は、的中率はさらに高くなる。

ご清聴
ありがとうございました