฿の価格変動予測したらLSTMにSVMが勝った話

2019年4月17日 2019年4月20日

SHARE

1. 初めに

こんにちは。機械学習を学び始めて約１ヶ月、出来ること出来ないことがやっと少しずつ見え始めてきて自身のイメージとの違いに日々驚かされています、研修生の浅井寛之です。大学では経営学や金融などを勉強しています。

早速本題に入りますが、これから仮想通貨の価格変動予測をしていきたいと思います。特に今回は　”実際にどれくらい正解しているのか”　というのをわかりやすく伝えるために、”up” “down” “stay” の３種分類問題を扱います。スコアを我々のイメージしやすい正解率という形で出力できるので比較しやすいから、というのがメインの理由ですが、ネットにあまり価格変動の分類問題が上がっていなかったからというのも理由の一つです。

また、実際本気で収入源にしてる方はさておき、小金儲けしようとして仮想通貨を買っている人たちの多くは１ヶ月後までの価格の推移よりも明日価格が上がっているかどうかの方に興味があると思います。自分は一時期仮想通貨でお金を溶かしていた時期があったので、今回はそのリベンジマッチの気分で挑みます。

2.1 今回使うモジュール

まずは時系列データの解析が得意なニューラルネットワークであるRNNを使っていきます。

仮想通貨の価格変動には技術進歩や政府規制などの将来への見通しに基づく長期トレンドと、テクニカル指標の動きや単発のニュースに基づく短期トレンドの２種類があると考えられるので、今回は時系列データの分析が得意なRNNの中でも特に長期依存関係にも対応可能なLSTMを用いていきたいと思います。

```
import random
import pandas as pd
import time
import matplotlib.pyplot as plt
import datetime
import numpy as np
from sklearn import preprocessing
from keras.utils.np_utils import to_categorical
from keras.models import Sequential
from keras.layers import Dense, Activation, Dropout
from keras.layers.recurrent import SimpleRNN, LSTM
from keras.layers.normalization import BatchNormalization
from keras.callbacks import EarlyStopping
plt.style.use('ggplot')
```

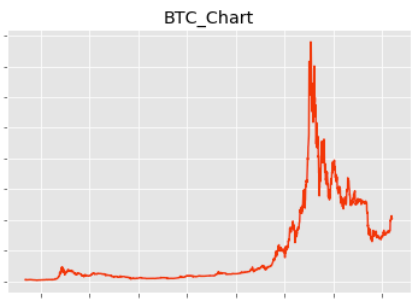
2.2. 価格データの取得

続いて価格データを拾ってきます。取得日のスタートを固定しており実行する日によってデータの取得量は変わってくるので、最初にサイズも見ておきます。また、このサイトでは欠損値がNanの時と”-“の時の２種類ある点に注意が必要です。これらが残っていると後々面倒なのでしっかり処理していきます。

```
bitcoin_market_info = pd.read_html("https://coinmarketcap.com/currencies/bitcoin/historical
bitcoin_market_info = bitcoin_market_info.assign(Date=pd.to_datetime(bitcoin_market_info['D
bitcoin_market_info=bitcoin_market_info.drop(["Date","Volume"],axis=1)
datasize=bitcoin_market_info.shape[0]
```

確認のため今回使っていくデータを可視化してみます。使う説明変数は以下の５つです。

	Open*	High	Low	Close**	Market Cap
0	5061.20	5103.27	4955.85	5089.54	89792633045
1	5325.08	5354.23	5017.30	5064.49	89341168457
2	5204.11	5421.65	5193.38	5324.55	93918439274
3	5289.92	5289.92	5167.42	5204.96	91799330425
4	5199.84	5318.84	5148.21	5289.77	93286365973
5	5062.79	5235.19	5050.41	5198.90	91674230186



特に問題はなさそうなので続けていきます。

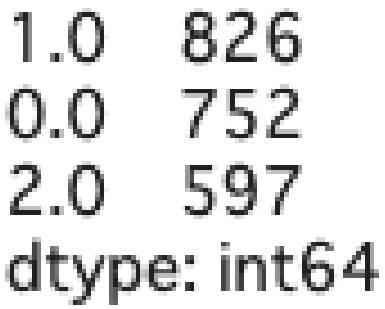
2.3. データの前処理(1)

次にデータをモデルに入れやすいように整形してくれる関数を作ります。機械学習を行う上で最も大事と言われているのがこの前処理です。LSTMに突っ込むデータの形にするために、wideで何日前まで参照するかを決め、DataFrameオブジェクトの形をいじっています。また、正解ラベルは翌日の値になってくるのでここら辺のindex番号も注意しながら慎重に関数を作っていきます。

なお、今回の正解ラベルは誤差のような変動は邪魔なデータになると判断して1%以下の価格の変動は無視しています。（０以上か０未満かだけに注目して２値分類もやってみましたが、スコアは全体的に0~0.2くらい低かったです。選択肢の数が３÷２になってる中で正解率が下がっているので、やはり小さな

価格変動は無視した方が良さそうです。)また、気をつけないといけないのがここの閾値を大きくしすぎるとほぼ全てのデータが変動無しとなり、全て"stay"と予測するだけで"精度の高い"モデルになってしまう、という点です。

丁度"up","down","stay"の比が1：1：1くらいになるよう適当に調節し行った結果が以下になります。今回は価格の１％以下の変動は無視することにしました。



ではコードを見ていきます。

```
wide=60#何日前のデータまで見るか

#連続値を離散値にする関数（閾値は変化率0.01）
f=lambda x: 2 if x>0.01 else 0 if x<-0.01 else 1 if -0.01<=x<=0.01 else np.nan

def seikei(df):
    random.shuffle([i for i in range(datasize-wide-2)])#RNNでは学習する順番によっても結果が変わ
    test_index=shuffle_index[:datasize//3]
    train_index=shuffle_index[datasize//3:]

    df_train_list=[]
    df_test_list=[]
    df_list=[]
    keys=["{}".format(i) for i in range(wide)]
    columns=df.columns

    #正解ラベルの作成
    close_diff=df.loc[:,"Close**"].pct_change(-1).map(f).rename(columns={'Close**': 'diff'}

    y_train=close_diff[train_index]
    y_test=close_diff[test_index]

    diff_list=[]
#変分からなるデータフレームに書き換える
    for col in columns:
        data=df.loc[:,col]
        diff_data_cleaned=preprocessing.scale(data.pct_change(-1)[:datasize-1])#価格変動をみ
        diff_data_cleaned.index=range(datasize-1)
        diff_list.append(pd.Series(data=diff_data_cleaned, dtype='float'))

    df=pd.concat(diff_list,axis=1)

    for column in columns:
        series_list=[df.loc[:,column]]
        for i in range(wide):
            series_kari=series_list[0].drop(0)
            series_kari.index=range(datasize-(i+2))
            series_list.insert(0,series_kari)

        concat_df=pd.concat(series_list,axis=1,keys=keys).drop(0).dropna()
        concat_df.index=range(datasize-(wide+2))

        concat_df_train=concat_df.iloc[train_index,:]
        concat_df_test=concat_df.iloc[test_index,:]
```

```
        df_train_list.append(concat_df_train)
        df_test_list.append(concat_df_test)
    return df_train_list,df_test_list,y_train,y_test
```

2.4. データの前処理(2)

ここでは完成したDataFrameをLSTMに入れられるようnumpyデータにする関数を作ります。上の関数にくっつけても良いのですが、上の処理は今回得られたデータに特有の処理でこちらの関数は比較的様々なデータに活用できると考えられるので、別々で作ります。

```
def convert_threeDarray_for_nn(df_list):
    array_list = []
    for df in df_list:
        ndarray = np.array(df)
        array_list.append(np.reshape(
            ndarray, (ndarray.shape[0], ndarray.shape[1], 1)))

    return np.concatenate(array_list, axis=2)
```

2.5. データの前処理(3)

では実際に取得した価格情報データを入れて、トレーニングデータを作成していきましょう。今回は分類問題なので、正解ラベルに関しては少し特殊なバイナリーデータという形にしておきます。これで前処理は最後になります。

```
train_df_list,test_df_list,Y_train,Y_test=seikei(bitcoin_market_info)

X_train = convert_threeDarray_for_nn(train_df_list)
X_test = convert_threeDarray_for_nn(test_df_list)

n_classes =3
Y_train = to_categorical(Y_train, n_classes)
Y_test = to_categorical(Y_test, n_classes)

input_size = [X_train.shape[1], X_train.shape[2]]#入力するデータサイズを取得
```

2.6. 指標の設定

今回は実際の売買を想定して、通常の正解率に加え”投資失敗率”という指標を導入していきます。というのも実際に売買を行うタイミングは予測値がupになったときであることを考えると、*“予測値がupだったのに実際はdownだった”*という誤りは、*“予測値がstayだったのに実際はupだった”*という間違えより明らかに重要度が高いからです。今回はこのなんとしてでも避けたい間違いの割合を投資失敗率と定義して同時に出力していきたいと思います。

分類問題で使うsoftmaxという活性化関数は離散値でなく連続値を出力するという点や（それぞれのラベルの予測確率を出力します。）、正解ラベルがバイナリーデータであるという点に注意しながら処理していきます。

```
def to_array(y)
  array=[]
  for i in range(y.shape[0]):
    array.append(y[i].argmax())
  return(array)
```

```
def kentei(predict_y,test_y):
  count=0
  for i in range(len(predict_y)):
    if predict_y[i]==2 and test_y[i]==0:
      count+=1
  return count/predict_y.count(2)
```

2.7.モデルの作成

続いてモデルの作成に入ります。後でチューニングできるようにこちらも関数にしていきます。今回絶対外せないのは活性化関数と損失関数を３種以上の分類問題用のものを使うという点だと思います。ここを２値分類用の関数とかにしても回ってしまうので気をつけないといけません。

```
def pred_activity_lstm(input_dim,
                       activate_method='softmax',  # 活性化関数
                       loss_method='categorical_crossentropy',  # 損失関数
                       optimizer_method='adam',  # パラメータの更新方法
                       kernel_init_method='glorot_normal',  # 重みの初期化方法
                       batch_normalization=False,  # バッチ正規化
                       dropout_rate=None  # ドロップアウト率
                       ):

    model = Sequential()
    model.add(
        LSTM(
            input_shape=(input_dim[0], input_dim[1]),
            units=60,
            kernel_initializer=kernel_init_method,
            return_sequences=True
        ))

    if batch_normalization:
        model.add(BatchNormalization())

    if dropout_rate:
        model.add(Dropout(dropout_rate))

    model.add(
        LSTM(
            units=30,
            kernel_initializer=kernel_init_method,
            return_sequences=False
        ))

    if batch_normalization:
        model.add(BatchNormalization())

    if dropout_rate:
        model.add(Dropout(dropout_rate))

    model.add(Dense(units=n_classes, activation=activate_method))
    model.compile(loss=loss_method, optimizer=optimizer_method,
                  metrics=['accuracy'])

    return model

turned_model = pred_activity_lstm(
    input_dim=input_size,
    activate_method='softmax',
    loss_method='categorical_crossentropy',
    optimizer_method='adam',
    kernel_init_method='glorot_normal',
    batch_normalization=True
```

```
)

early_stopping = EarlyStopping(monitor='val_loss', patience=10, verbose=1)
```

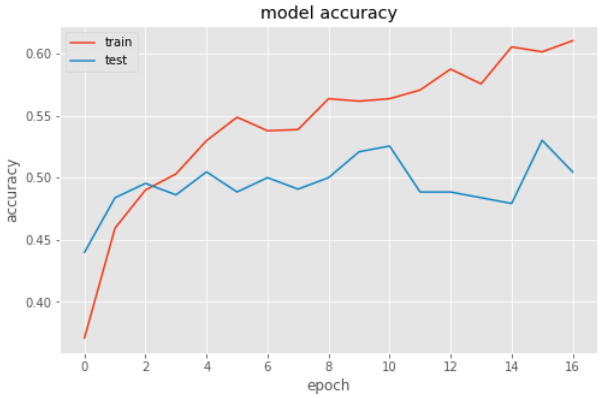
2.8.学習

いよいよ学習に入っていきます。ここもチューニングが絡んできますね。機械学習ではバッチサイズを2の累乗にする習慣があるらしいです。epochに関してはEarlyStoppingを組み込んでる以上、増やす分には問題ないと思うので１００で行きます。EarlyStoppingは、学習を進めているのにval_accの値が下がり始めた場合、過学習と判断して自動で止めてくれます。

また、機械学習分野では可視化することで問題点が見えやすくなることが多いので、今回も精度の推移図を出力していきます。ここを見ながらチューニングしていくことになります。

```
# 学習スタート
history = turned_model.fit(
    X_train,
    Y_train,
    batch_size=64,
    epochs=100
    validation_split=0.3,
    callbacks=[early_stopping],
    verbose=2
)

score = lstm_model.evaluate(X_test, Y_test, verbose=1)


# 精度の推移図を出力
plt.figure(figsize=(8, 5))
plt.plot(history.history['acc'])
plt.plot(history.history['val_acc'])
plt.title('model accuracy')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='upper left')
plt.show()

# 損失関数の推移図を出力
plt.figure(figsize=(8, 5))
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('model loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='upper left')
plt.show()

score = lstm_model.evaluate(X_test, Y_test, verbose=0)
print('Test loss:', score[0])
print('Test accuracy:', score[1])

test_y=to_array(Y_test)
pred_y=to_array(turend_model.predict(X_test)

print("投資失敗率:{}".format(kentei(pred_y,test_y)))
```

2.9.結果



Test loss: 1.1031700715525397
Test accuracy: 0.44551724211923005

投資失敗率:0.32286995515695066

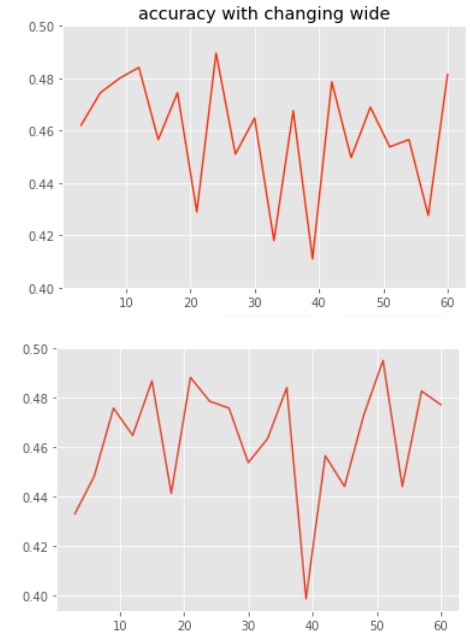
2.10.レビュー

ランダムウォークが基本と言われている価格変動に対してスコア0.445、正直言うともう少し高い精度を求めていたとのは事実ですが、まぁ３択問題であることを考慮に入れれば及第点でしょうか。しかし投資失敗率が0.3というのは少し実戦には不安が残ります。５割以下であることを考えれば理論上は継続して行えばトータルで勝てることになりますが、そうはうまくいきそうにないです。

グラフを見てやると、epochが早い段階で打ち切られているのが分かります(16/100)。やはり価格変動自体ランダムウォークが基盤となっているので、データ数を増やしても過学習に繋がるリスクの方が大きくなるのかなと思います。そもそもLSTMモデル自体少ないデータ数でも学習が進むというのが強みでもあるので、ここも関係しそうです。

データの必要数が少なくて良いなら、例えばここ1年だけに絞ったデータを使ってみればより今のトレンドに合わせた予測ができるのではと判断して取得日時の範囲をチューニングしてみましたが、変化は特にありませんでした。価格変動はある程度普遍的なルールに基づいており、学習データの範囲を広げることでリスク分散ができていると解釈できます。

また、意外だったのがwide(過去何日分のパラメータを参照するか）パラメータのチューニングで、ここをfor文で回してみたところ、wideを大きくしてもスコアに変化はありませんでした。wideがおよそ１０以上であれば、下のグラフから分かる通りおそらく実行中の乱数によってでる精度の違いしか得られませんでした。（バッチ数やエポック数などのパラメータチューニングをその都度している訳ではないので、適切なチューニングができていないのも原因の一つと考えられます。）



（上の二枚のグラフは全て同じパラメータで実行し得られたグラフなので、テストデータやトレインデータの分類など乱数が絡むことでスコアが変動していることがわかります。）

この理由として、価格変動は基本的に正弦波を描いており、長期のトレンドが影響するのは正弦波の中心軸方向の傾きである以上、翌日の価格変動という短期スパンのみに注目した場合影響はかなり小さくなってしまうのではないか、と推測しました。勿論影響が０な訳ではないのですが、wideを大きくするとそれ以上に余分な情報が入ってきてしまい学習が効率的に進まなくなることも考えられるので、それと効果が打ち消しあっているのではないでしょうか。

色々試してみて思ったんですがLSTMは本当に理解がないとパラメータチューニングで時間が無限に溶けます。ここに関してはより理論を学んでから再挑戦しないと、現状ではこれ以上どうしようもないという印象です。

3.1. 非線形SVMでやってみた

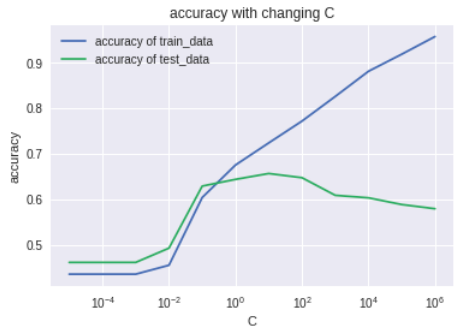
LSTMを用いた分析の結果長期トレンドはかなり無視して情報を絞って良さそうなので、通常の複数の特徴量と正解ラベルをもつ非線形SVMのモデルでもいけるのではと推測しました。ちなみにこちらもチューニングをしたところ、実際wide=3で最大値をとっていました。SVMが分離超平面を引くというモデルである以上特徴量の次元が大きくなりすぎると精度が低くなるのでしょう。

LSTMの時とは渡すデータの形が違うのでコード自体は少し変わっていますが、やっていることの流れとしてはほぼ一緒です。スコアはどうなるんでしょうか。

```
import pandas as pd
import time
import matplotlib.pyplot as plt
import datetime
import numpy as np
from sklearn import preprocessing
from sklearn.svm import SVC
from sklearn.ensemble import RandomForestClassifier as RFC
from sklearn.model_selection import train_test_split
from tqdm import tqdm
%matplotlib inline

bitcoin_market_info = pd.read_html("https://coinmarketcap.com/currencies/bitcoin/historical
datasize=bitcoin_market_info.shape[0]

#データの前処理
##欠損データの処理
bitcoin_market_info = bitcoin_market_info.replace("-", np.nan).fillna(method="ffill")
bitcoin_market_info.index=range(datasize)
```

```
#Date,Volumeの削除
bitcoin_market_info=bitcoin_market_info.drop(["Date","Volume"],axis=1)
cols=bitcoin_market_info.columns

diff_list=[]
for col in cols:
    diff_data=bitcoin_market_info.loc[:,col].pct_change()[1:]
    diff_data.index=range(datasize-1)
    series = pd.Series(data=diff_data, dtype='float')
    diff_list.append(series)

df=pd.concat(diff_list,axis=1)

#時間方向を横軸に組み込んだDataFrameの作成
dataframe_list=[df]
wide=3
keys=["{}".format(i) for i in range(wide)]
for i in range(wide):
    data_kari=dataframe_list[i].drop(i)
    data_kari.index=range(datasize-(i+2))
    dataframe_list.append(data_kari)
concat_df=pd.concat(dataframe_list,axis=1,keys=keys).dropna()

#学習用データの作成
f=lambda x: 2 if x>0.01 else 0 if x<-0.01 else 1
y=concat_df_1.iloc[:,1].map(f).values.astype(np.int64)[:y.shape[0]-1]
X=preprocessing.scale(concat_df_1).astype(np.float64)[1:,1]

train_X,test_X,train_y,test_y=train_test_split(X,y,random_state=0)

def kentei(predict_y,test_y):
  count=0
  for i in range(len(predict_y)):
    if predict_y[i]==2 and test_y[i]==0:
      count+=1
  return count/predict_y.tolist().count(2)

C_list = [10 ** i for i in range(-5,7)]

# グラフ描画用の空リストを用意
train_accuracy = []
test_accuracy = []

for C in tqdm(C_list):
    model = SVC(C=C)
    model.fit(train_X, train_y)
    train_accuracy.append(model.score(train_X, train_y))
    test_accuracy.append(model.score(test_X, test_y))

predict_y=model.predict(test_X)

# グラフの準備
plt.semilogx(C_list, train_accuracy, label="accuracy of train_data")
plt.semilogx(C_list, test_accuracy, label="accuracy of test_data")
plt.title("accuracy with changing C")
plt.xlabel("C")
plt.ylabel("accuracy")
plt.legend()
plt.show()

print("Average score is {}".format(np.mean(test_accuracy)))
print("Max score is {}".format(np.max(test_accuracy)))

print("投資失敗率:{}".format(kentei(predict_y,test_y)))
```

3.2 結果



Average score is 0.5692401960784313
Max score is 0.65625

投資失敗率:0.1643835616438356

3.3. レビュー

Average scoreはあくまで様々なチューニングの平均ですので、このモデルのスコア自体はMaxscore である0.656と判断して良いでしょう。LSTMよりはるかに良いスコアをだすとは思いませんでした。ただc＝１０辺りから明らかに過学習に入って行っています。ランダムウォークが基本的な値動きの原理と言われている以上過学習は避けられないので、これ以上のスコアの上昇を見込むのは厳しそうです。

また、投資失敗率が0.16とかなり低いです。実際にこのモデルを運用してupのタイミングで購入し翌日売るというアルゴリズムで動いた場合、翌日に逆に1%以上価格が減少してしまう確率は約16%しかないということになります。シンプルなモデルにしては十分じゃないでしょうか。少し不安になり、何らかの予測の偏りがあり、実際は使えないようなモデルになっているのではと思って確認してみましたが、以下の写真の通り特に偏りはありませんでした。

```
print(predict_y)
```

```
[0 1 1 2 2 0 1 2 0 0 2 0 2 1 1 1 0 0 1 0 1 1 1 1 1 1 1 0 2 1 2 1 1 2 1 1 2
 1 1 1 0 1 0 1 2 1 1 1 0 1 1 2 1 2 2 1 2 1 2 0 0 0 1 1 1 0 0 0 1 2 1 1 0 1
 1 0 0 0 2 0 1 2 0 1 0 1 0 2 2 0 1 1 2 0 1 2 0 0 2 1 0 0 2 1 0 1 1 0 2 1 1
 1 0 1 2 0 1 1 1 0 0 2 2 2 2 2 1 0 1 2 0 1 1 1 0 2 1 1 1 1 2 0 0 0 0 2 1 1
 0 1 2 1 1 2 2 2 0 1 0 2 1 1 1 1 2 1 2 0 1 2 1 0 1 1 1 1 1 1 0 1 1 1 2 1 1 1
 0 1 1 0 1 2 0 2 1 1 1 2 0 2 1 0 0 1 1 0 0 1 1 0 2 0 0 0 2 1 0 1 2 1 2 0 2
 0 2 0 0 1 2 2 0 1 0 0 2 1 2 2 2 1 1 2 1 2 2 1 0 0 1 2 1 1 2 2 2 1 0 1 0 0 1
 0 0 1 0 1 1 0 1 2 2 2 1 0 1 2 0 2 2 0 1 0 1 2 1 1 1 0 0 1 2 1 0 0 1 1 1 2
 1 2 1 2 1 2 1 2 1 1 1 0 0 2 0 2 1 0 2 1 2 1 1 2 0 1 0 2 1 0 0 1 1 1 0 0 1
 1 2 1 1 1 1 0 2 0 0 0 1 1 2 2 1 0 1 1 0 2 1 0 0 2 2 1 1 1 1 1 0 1 2 1 1 0
 2 1 1 2 1 1 2 0 1 2 1 1 0 2 2 1 2 1 2 1 1 0 0 2 2 0 0 1 1 1 2 0 1 1 1 0 2
 1 1 0 1 1 1 1 2 2 1 1 1 2 0 2 1 0 1 0 2 2 1 0 1 1 1 1 2 2 2 2 2 1 0 1 2 2
 1 0 1 2 1 2 1 1 2 1 1 0 1 0 2 0 2 0 2 0 1 1 2 1 2 1 2 0 2 1 0 2 1 1 2 1 2 2
 2 2 2 1 2 1 2 1 2 1 1 1 1 1 0 0 1 0 2 1 0 1 0 1 0 1 0 1 2 0 1 2 1 2 1 2 2
 0 0 0 0 0 1 1 1 0 0 1 1 1 0 0 1 0 1 2 1 0 2 1 0 2 0]
```

ちなみに参考のためこの後にランダムフォレストも試してみましたが、こちらもSVMよりやや低いものの0.6前後の高いスコアを出しました。考察はほとんど同じになるので特にいたしませんが、技術的なことに関していうとランダムフォレスト の.futureというメソッドを用いることで特徴量毎の正解ラベルへの寄与を見ることができます。実はLSTMモデルを使う前にこの方法で選定を多少行なっていたのですが、殆ど寄与率が同じだったため今回は特に役に立ちませんでした。もともと必要最低限の価格データしか取り入れていないので当然かもしれません。

4.考察

一般的に非常に強力だと言われており最近よく耳にするLSTMが、かなり古くから使われているモデルであるSVMやランダムフォレストと比べてこのようなスコアになってしまうのは非常に驚きでした。少し調べてみたところ、

https://qiita.com/koreyou/items/f86a2dd223d4a7d26f64

の記事がわかりやすいかなと思いました。簡単に説明すると、SVMではある程度パラメータのチューニングが自動的に行われているため、モデル自体がシンプルでそこの処理をSVMとLSTMで差別化できないなら、チューニングをうまくできない場合はSVMの方がいいスコアがでることもあるという事です。

自分みたいにまだまだニューラルネットワークの理解が浅くてパラメータチューニングを上手くできない、もしくはPCがチューニングに耐える処理能力を持っていない、という方は大人しくSVMを使った方が良さそうです。

もっと言ってしまうと、今回このブログを書くにあたってSVMのパラメータチューニングにかけた時間は10分程度、LSTMに関しては20時間ほどかかりました。というのもLSTMでは一つのパラメータをいじることで他のパラメータの最適値が変わる上、計算の実行自体も恐ろしく時間がかかるからです。１２０倍の労力に見合った結果かと問われるとこれは明らかに否です。（調べてみると、実際のマーケ部門とかのデータサイエンティストの方の多くもちょっとした依頼に対してはいまだにランダムフォレスト やSVMを使うのが現状らしいです。）そのため知識がない中で闇雲にニューラルネットワークなどを使うよりは、シンプルな機械学習モデルを扱った方が良いのかなと印象です。

また、今回自分が使った指標は”open”,”close”,”high”,”low”,”market cap”というかなり基礎的な値だけで、参照する過去の期間もかなり短かったという事がSVMの強みを活かしたのは確かです。短期トレンドが比較的短めな仮想通貨チャートでは共通しているのではと推測できます。

ではLSTMは仮想通貨の価格変動予測の分類問題に向かないかというと恐らくそれは誤りであると思います。上の議論は扱う値を上記５種に絞っているから成立するのであり、本来は他にも様々な景気のトレンドに影響を受けています。例えばドルや世界の株式指数などを説明変数に持ってきたり、もしかすると仮想通貨の基盤となっているブロックチェーン技術に関する論文の本数なども使える値かもしれません。

こういった様々なデータを引っ張り出して指標として用いてより精度の高いモデルを作っていくためには、確かなチューニングの知識を用いたLSTMモデルの運用が必要になってくると思います。今回は参照先をマーケットプライスのみに絞りましたが、今後は上述したように様々なデータを持ってきて、より複雑なモデルで実験してみたいと思います。

5.終わりに

初めにも書きましたが、実際にプログラミングを学習していると自身のイメージとの違いに日々驚かされます。自分は大学でIT×企業戦略に関する授業をいくらか取っていますが、その中で耳にするいわゆる”AI”の能力と実際にプログラムを動かしてみてわかることには大きな差があります。今回も自分の中の漠然とした”LSTMは最新で凄い、SVMはそれに劣る”というステレオタイプなイメージが崩されました。

今後AIによって社会が大きく変わることは間違いないだろうし、技術はより複雑化し想像もできないものになっているかもしれません。そんな中であっても上っ面の解釈やイメージだけに頼るのではなく実際にコードを動かしてみることで見えてくるものは、参考書やビジネス書を読んで得られる知識に新たな見方を与えてくれるんじゃないかと思います。

ここまで読んでいただき、ありがとうございました。以下、参考にした記事です。

https://blog.aidemy.net/entry/2018/08/23/195247

https://qiita.com/kazuimotn/items/96e2f067c4c0f3788d9d

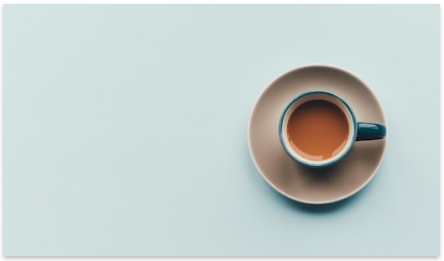
https://arakan-pgm-ai.hatenablog.com/entry/2017/09/03/080000

SHARE

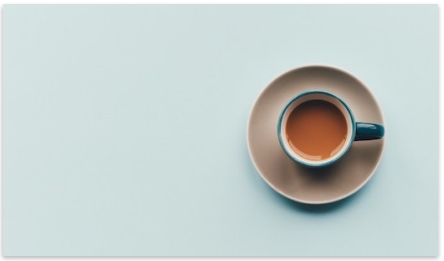
ツイート シェア はてブ LINE Pocket

この記事が気に入ったらフォローしよう

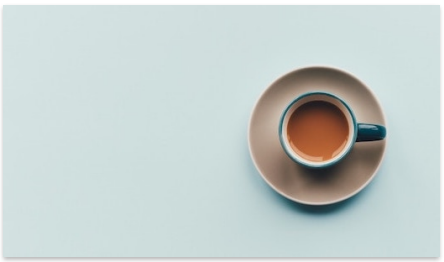
CATEGORY : 未分類



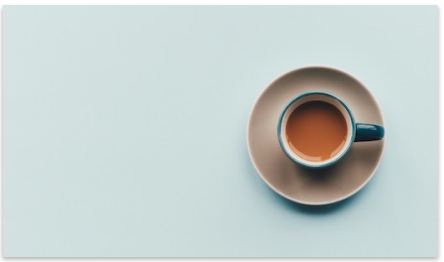
facebookのイベント通知から自然言語処理を学ぶ



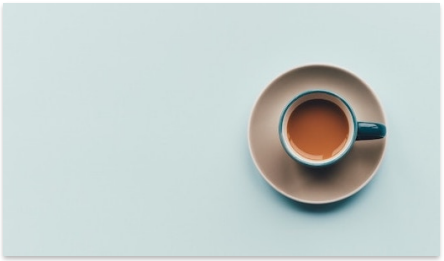
ラーメンを本気で分類してみた©



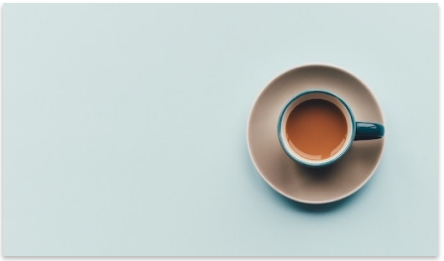
西野カナに「恋」とは何か聞いてみた



『グラスを分類するAI』を作ってみた



MySQLをjupyter notebookのkernelに入れる方法



弊社のコンテンツチェック/修正スケジュールならびに体制について

コメントを残す

メールアドレスが公開されることはありません。 * が付いている欄は必須項目です

コメント

名前 *

メールアドレス *

サイト

コメントを送信

前の記事

機械学習による日本の人口推移予想

次の記事

5月の新コース「統計学標準」リリース

サイト内検索

検索

最近の投稿

Aidemyの演習画面が大幅リニューアル！スマホ、タブレットでもコードが書けるようになります！
2019年8月27日

8月の新コース『ソラコム流、ラズパイで始めるIoT』リリース
2019年8月9日

6月の新コース「マイクロソフトのAIプラットフォーム講座」リリース
2019年6月28日

6月の新コース「オープンイノベーション実践のためのAIリテラシー」リリース
2019年6月25日

6月の新コース「AIマーケター育成講座」リリース
2019年6月10日

カテゴリー

カテゴリーを選択

HOME

© 2020 Aidemy, inc. All rights reserved.