

# stack-rnn-arima-xgb

September 29, 2021

```
[1]: import tensorflow as tf
from sklearn.model_selection import KFold, cross_val_score, train_test_split
from sklearn.metrics import mean_squared_error
import xgboost as xgb
import numpy as np
import matplotlib.pyplot as plt
from sklearn.preprocessing import MinMaxScaler
import seaborn as sns
import pandas as pd
import autoencoder
import model
from datetime import datetime
from datetime import timedelta
sns.set()
```

## 0.1 Deep Feed-forward Auto-Encoder Neural Network to reduce dimension + Deep Recurrent Neural Network + ARIMA + Extreme Boosting Gradient Regressor

### 0.1.1 Our target is Close market

```
[2]: google = pd.read_csv('GOOG.csv')
eur_myr = pd.read_csv('eur-myr.csv')
usd_myr = pd.read_csv('usd-myr.csv')
oil = pd.read_csv('oil.csv')
```

```
[3]: google['oil_price'] = oil['Price']
google['oil_open'] = oil['Open']
google['oil_high'] = oil['High']
google['oil_low'] = oil['Low']
google['eur_myr'] = eur_myr['Unnamed: 1']
google['usd_myr'] = usd_myr['Unnamed: 1']
```

```
[4]: date_ori = pd.to_datetime(google.iloc[:, 0]).tolist()
google.head()
```

```
[4]:
```

	Date	Open	High	Low	Close	Adj Close	\
0	2017-10-02	959.979980	962.539978	947.840027	953.270020	953.270020	
1	2017-10-03	954.000000	958.000000	949.140015	957.789978	957.789978	
2	2017-10-04	957.000000	960.390015	950.690002	951.679993	951.679993	
3	2017-10-05	955.489990	970.909973	955.179993	969.960022	969.960022	
4	2017-10-06	966.700012	979.460022	963.359985	978.890015	978.890015	

	Volume	oil_price	oil_open	oil_high	oil_low	eur_my	usd_my
0	1283400	54.27	54.26	54.39	54.22	4.9260	4.226
1	888300	54.24	54.59	55.22	53.89	4.9232	4.232
2	952400	54.38	54.08	54.85	53.93	4.9255	4.231
3	1213800	54.15	54.16	54.46	53.75	4.9239	4.238
4	1173900	53.90	52.80	54.20	52.25	4.9251	4.241

```
[5]: minmax = MinMaxScaler().fit(google.iloc[:, 4].values.reshape((-1,1)))
df_log = MinMaxScaler().fit_transform(google.iloc[:, 1:].astype('float32'))
df_log = pd.DataFrame(df_log)
df_log.head()
```

```
[5]:
```

	0	1	2	3	4	5	6	\
0	0.094605	0.050227	0.000000	0.021539	0.021539	0.092326	0.978389	
1	0.000000	0.000000	0.018810	0.082769	0.082769	0.000000	0.972495	
2	0.047461	0.026441	0.041238	0.000000	0.000000	0.014979	1.000000	
3	0.023572	0.142825	0.106207	0.247630	0.247630	0.076062	0.954813	
4	0.200918	0.237416	0.224569	0.368600	0.368600	0.066738	0.905698	

	7	8	9	10	11
0	0.938202	0.847145	1.000000	0.033373	0.523804
1	1.000000	1.000000	0.935547	0.000000	0.714279
2	0.904495	0.931860	0.943359	0.027411	0.682536
3	0.919476	0.860036	0.908203	0.008343	0.904755
4	0.664794	0.812155	0.615234	0.022643	1.000000

```
[6]: thought_vector = autoencoder.reducedimension(df_log.values, 4, 0.001, 128, 100)
```

```
epoch: 10 loss: 0.272533 time: 0.0006597042083740234
epoch: 20 loss: 0.272347 time: 0.0007002353668212891
epoch: 30 loss: 0.272032 time: 0.0006601810455322266
epoch: 40 loss: 0.271498 time: 0.0006575584411621094
epoch: 50 loss: 0.270591 time: 0.0006284713745117188
epoch: 60 loss: 0.26905 time: 0.0006418228149414062
epoch: 70 loss: 0.266411 time: 0.0006747245788574219
epoch: 80 loss: 0.261816 time: 0.0007426738739013672
epoch: 90 loss: 0.253563 time: 0.0006310939788818359
epoch: 100 loss: 0.238662 time: 0.0006124973297119141
```

```
[7]: thought_vector.shape
```

[7]: (23, 4)

```
[8]: num_layers = 1
size_layer = 128
timestamp = 5
epoch = 500
dropout_rate = 0.1
```

```
[9]: tf.reset_default_graph()
modelnn = model.Model(0.01, num_layers, thought_vector.shape[1], size_layer, 1,
    ↳ dropout_rate)
sess = tf.InteractiveSession()
sess.run(tf.global_variables_initializer())
for i in range(epoch):
    init_value = np.zeros((1, num_layers * 2 * size_layer))
    total_loss = 0
    for k in range(0, (thought_vector.shape[0] // timestamp) * timestamp,
    ↳ timestamp):
        batch_x = np.expand_dims(thought_vector[k: k + timestamp, :], axis = 0)
        batch_y = df_log.values[k + 1: k + timestamp + 1, 3].reshape([-1, 1])
        last_state, _, loss = sess.run([modelnn.last_state,
            modelnn.optimizer,
            modelnn.cost], feed_dict={modelnn.X:
    ↳ batch_x,
                                                                    modelnn.Y:
    ↳ batch_y,
                                                                    modelnn.
    ↳ hidden_layer: init_value})
        init_value = last_state
        total_loss += loss
    total_loss /= (thought_vector.shape[0] // timestamp)
    if (i + 1) % 100 == 0:
        print('epoch:', i + 1, 'avg loss:', total_loss)
```

WARNING:tensorflow:<tensorflow.python.ops.rnn\_cell\_impl.LSTMCell object at 0x7ff23c502128>: Using a concatenated state is slower and will soon be deprecated. Use state\_is\_tuple=True.

```
epoch: 100 avg loss: 0.226264208555
epoch: 200 avg loss: 0.0964816752821
epoch: 300 avg loss: 0.0767136435024
epoch: 400 avg loss: 0.0496228779666
epoch: 500 avg loss: 0.0471770029981
```

```
[10]: output_predict = np.zeros(((thought_vector.shape[0] // timestamp) * timestamp,
    ↳ 1))
init_value = np.zeros((1, num_layers * 2 * size_layer))
```

```

for k in range(0, (thought_vector.shape[0] // timestamp) * timestamp,
    ↳timestamp):
    out_logits, last_state = sess.run([modelnn.logits, modelnn.last_state],
    ↳feed_dict = {modelnn.X:np.expand_dims(thought_vector[k: k + timestamp, :],
    ↳axis = 0),
                                modelnn.hidden_layer: init_value})

    init_value = last_state
    output_predict[k: k + timestamp, :] = out_logits

```

```

[11]: print('Mean Square Error:', np.mean(np.square(output_predict[:, 0] - df_log.
    ↳iloc[1: (thought_vector.shape[0] // timestamp) * timestamp + 1, 0].values)))

```

Mean Square Error: 0.0510127100734

Import ARIMA model using stats model

```

[12]: import statsmodels.api as sm
from itertools import product
from scipy import stats

Qs = range(0, 1)
qs = range(0, 2)
Ps = range(0, 2)
ps = range(0, 2)
D=1
parameters = product(ps, qs, Ps, Qs)
parameters_list = list(parameters)
best_aic = float("inf")
for param in parameters_list:
    try:
        arima=sm.tsa.statespace.SARIMAX(df_log.iloc[:,3].values,
    ↳order=(param[0], D, param[1]), seasonal_order=(param[2], D, param[3], 1)).
    ↳fit(dispatch=-1)
    except:
        continue
    aic = arima.aic
    if aic < best_aic and aic:
        best_arima = arima
        best_aic = aic

best_aic

```

/usr/local/lib/python3.5/dist-packages/statsmodels/compat/pandas.py:56:  
FutureWarning: The pandas.core.datetools module is deprecated and will be  
removed in a future version. Please use the pandas.tseries module instead.  
from pandas.core import datetools

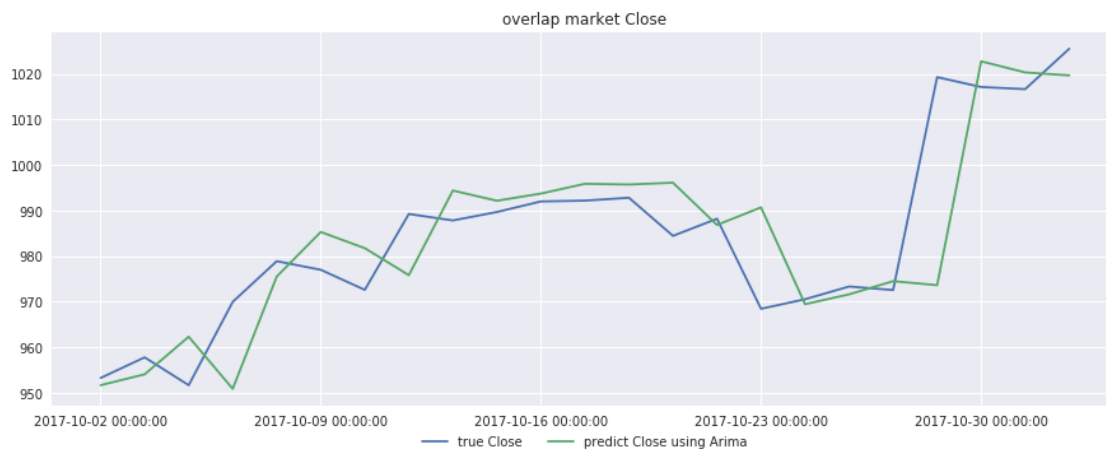
```

[12]: -7.7935465732797873

```

```
[13]: def reverse_close(array):
        return minmax.inverse_transform(array.reshape((-1,1))).reshape((-1))
```

```
[14]: pred_arima = best_arima.predict()
x_range = np.arange(df_log.shape[0])
fig = plt.figure(figsize = (15,6))
ax = plt.subplot(111)
ax.plot(x_range, reverse_close(df_log.iloc[:,3].values), label = 'true Close')
ax.plot(x_range, reverse_close(pred_arima), label = 'predict Close using Arima')
box = ax.get_position()
ax.set_position([box.x0, box.y0 + box.height * 0.1, box.width, box.height * 0.
↪9])
ax.legend(loc = 'upper center', bbox_to_anchor= (0.5, -0.05), fancybox = True,
↪shadow = True, ncol = 5)
plt.xticks(x_range[::5], date_ori[::5])
plt.title('overlap market Close')
plt.show()
```

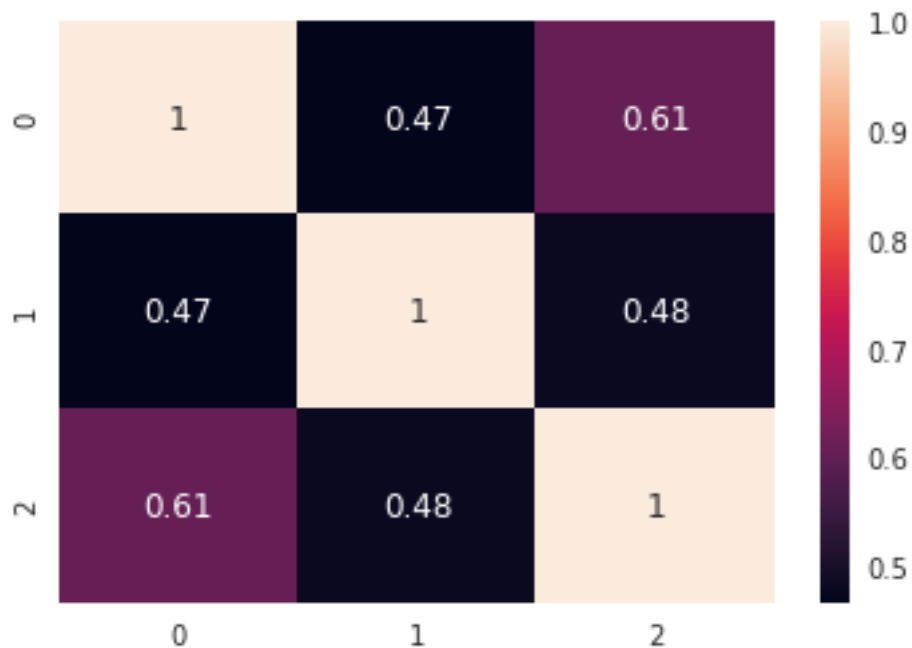


```
[15]: boundary = (thought_vector.shape[0] // timestamp) * timestamp
stack_predict = np.vstack([pred_arima[:boundary], output_predict.
↪reshape((-1))] ).T
```

```
[16]: where_below_0 = np.where(stack_predict < 0)
where_higher_1 = np.where(stack_predict > 1)
stack_predict[where_below_0[0], where_below_0[1]] = 0
stack_predict[where_higher_1[0], where_higher_1[1]] = 1
```

```
[17]: corr_df = pd.DataFrame(np.hstack([stack_predict, df_log.values[:boundary, 3].
↪reshape((-1,1))]))
```

```
[18]: sns.heatmap(corr_df.corr(), annot= True)
plt.show()
```



ARIMA able to predict data that correlate 0.61 originally from original Close

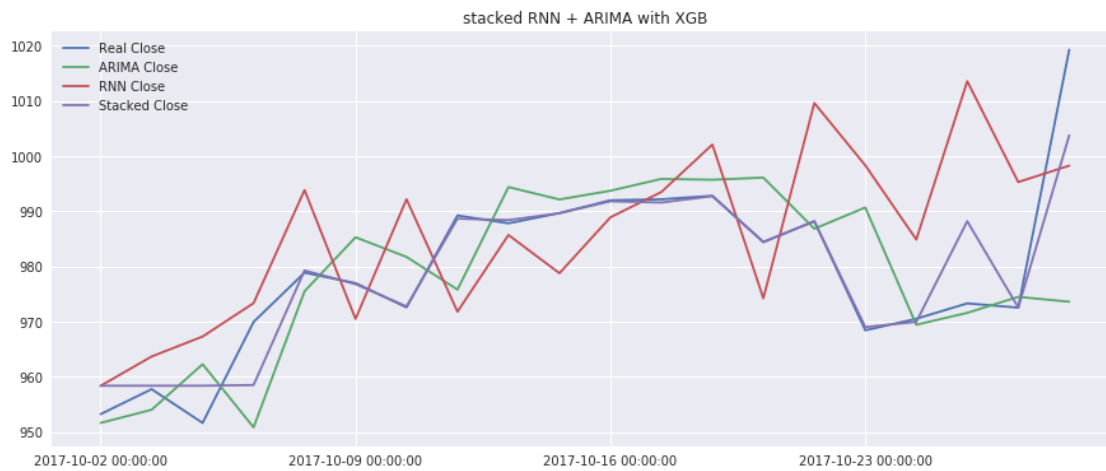
Deep Recurrent Neural Network able to predict data that correlate 0.48 originally from original Close

```
[19]: params_xgd = {
    'max_depth': 7,
    'objective': 'reg:logistic',
    'learning_rate': 0.05,
    'n_estimators': 10000
}
train_Y = df_log.values[:,boundary, 3]
clf = xgb.XGBRegressor(**params_xgd)
clf.fit(stack_predict,train_Y, eval_set=[(stack_predict,train_Y)],
        eval_metric='rmse', early_stopping_rounds=20, verbose=False)
```

```
[19]: XGBRegressor(base_score=0.5, colsample_bylevel=1, colsample_bytree=1, gamma=0,
    learning_rate=0.05, max_delta_step=0, max_depth=7,
    min_child_weight=1, missing=None, n_estimators=10000, nthread=-1,
    objective='reg:logistic', reg_alpha=0, reg_lambda=1,
    scale_pos_weight=1, seed=0, silent=True, subsample=1)
```

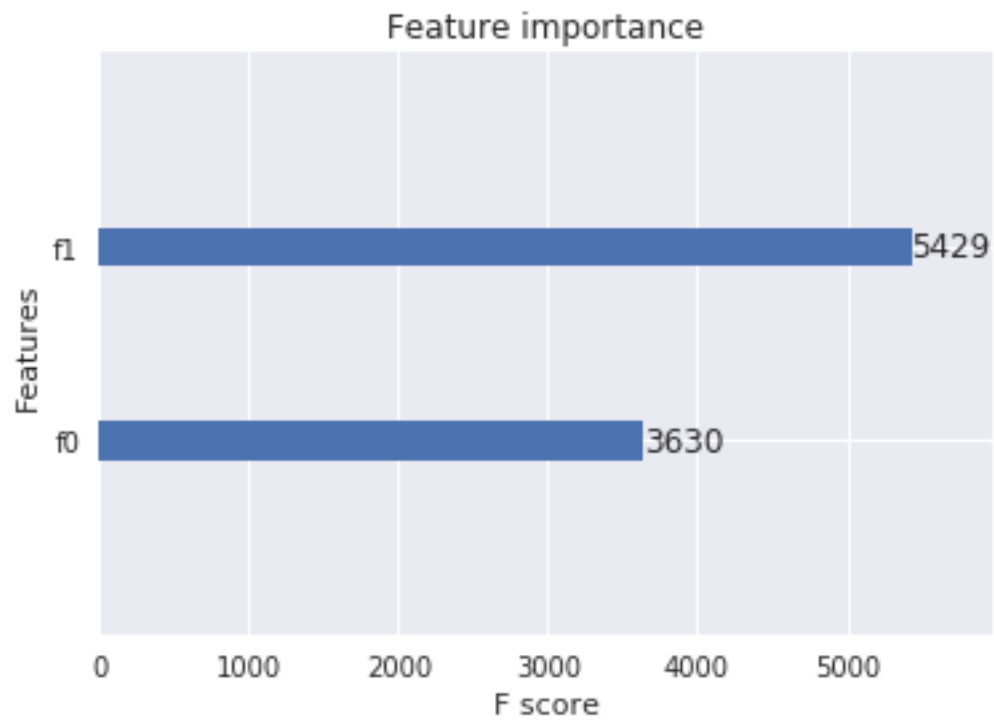
```
[20]: stacked = clf.predict(stack_predict)
```

```
[25]: plt.figure(figsize = (15,6))
x_range = np.arange(boundary)
plt.plot(x_range, reverse_close(train_Y), label = 'Real Close')
plt.plot(x_range, reverse_close(pred_arima[:boundary])), label = 'ARIMA Close')
plt.plot(x_range, reverse_close(output_predict), label = 'RNN Close')
plt.plot(x_range, reverse_close(stacked), label = 'Stacked Close')
plt.legend()
plt.xticks(x_range[::5], date_ori[:boundary][::5])
plt.title('stacked RNN + ARIMA with XGB')
plt.show()
```



## 1 Pretty insane i can say!

```
[26]: from xgboost import plot_importance
plot_importance(clf)
plt.show()
```



1.1 Arima is more important than RNN