Tesla LSTM binary classification

December 12, 2020

1 Tesla closing price trend classification using LSTM

[1]:
```
# import libraries
import numpy as np
import pandas as pd
import seaborn as sns
from keras.models import Sequential
from keras.layers import Dense
from keras.layers import LSTM
from keras.layers import Dropout
from keras.callbacks import EarlyStopping, ReduceLROnPlateau
from keras.optimizers import Adam
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import OneHotEncoder
from sklearn.preprocessing import LabelEncoder
from sklearn.preprocessing import MinMaxScaler
from keras.optimizers import SGD
```

[2]:
```
# Read the dataset
df = pd.read_csv('C:/Users/isaac/Documents/Poly/y4s1/data_mining/tesla/
↪Tesla_regression.csv')
```

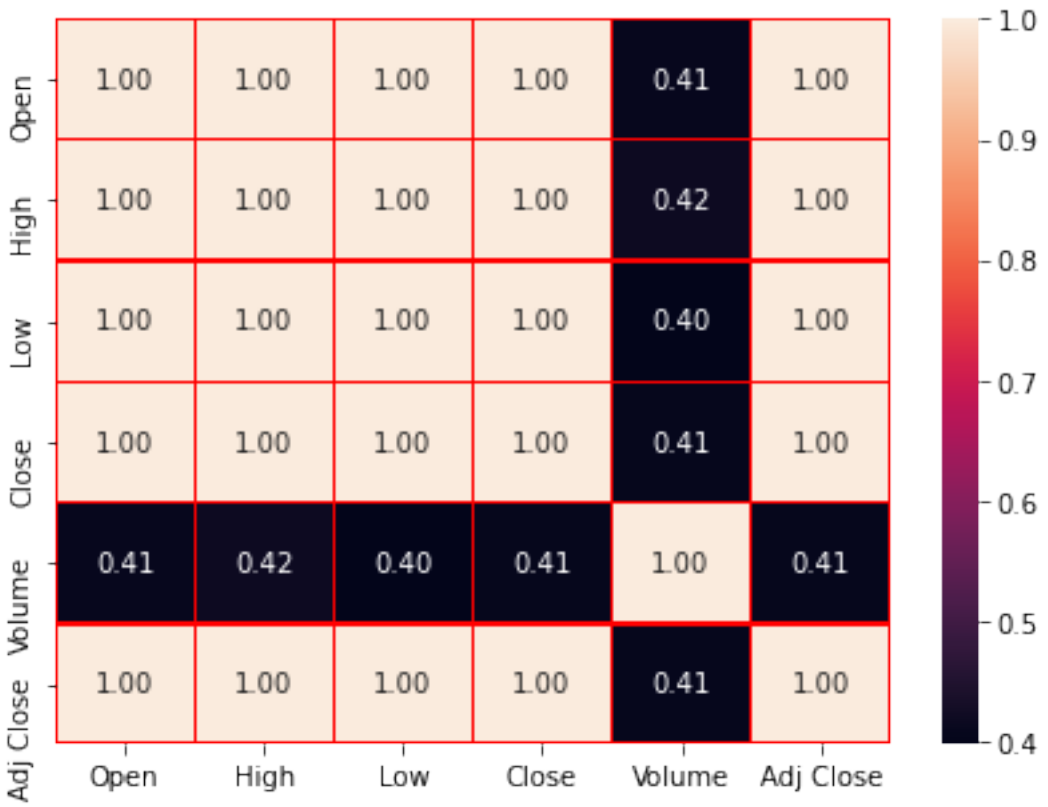
[3]:
```
# Drop data with null values
df = df.dropna()
```

1.0.1 Data Visualization and Analysis

Put the data into Heatmap.

As we can see, High, Open, Low, Close, and Adj clsoe are highly correlated to each other, while volume have low correlation with it.

[4]:
```
f,ax = plt.subplots(figsize = (7,5))
sns.heatmap(df.corr(), annot=True, linewidths=0.5,linecolor="red", fmt= '.
↪2f',ax=ax);
```



As we can see, High, Open, Low, Close, and Adj clsoe are highly correlated to each other, while volume have low correlation with it

Adj Close have the same data of Close price.

[5]:
```
# Comparing adj close and close
adj_close , close = np.array(df['Adj Close']), np.array(df['Close'])
comparison = adj_close == close
equal_arrays = comparison.all()
print(equal_arrays)
```

```
True
```

1.0.2 Feature modification

With the information extracted from above step, and the domain knowledge. I have made some feature modification before the model construction.

Dimensionality reduction Since Open, High, Low, Close, and Adj Close have very high correlation, only close will be keep in the feature set.

[6]:
```
df = df.filter(items=["Date", "Close", "Volume", "Trend"])
```

Feature Creation Year, month, day, dayOfWeek maybe some useful information in the dataset, so it should be included in the feature set.

One hot encoding are used to encode the day of week value. (Note that 0 = Monday, 4 = Friday)

[ ]:
```
# adding year, month, date, dayOfWeek data into the dataset
df['Date'] = pd.to_datetime(df['Date'])
df['Year'] = df['Date'].dt.year
df['Month'] = df['Date'].dt.month
df['Day'] = df['Date'].dt.day
df['Day of week'] = df['Date'].dt.dayofweek

# dummy encode the day of week
df = pd.get_dummies(df, columns = ['Day of week'])

# remove date in the dataset
df.set_index('Date',inplace=True)
```

Feature normalization Since I will use LSTM as the model, the feature set should be normalized. In this case, Min-max normalization would be used. Note that the training set(78%), validation set(8%), and test set(20%) should be using the same estimator (using the training set min max value) to avoid data leakage.

[8]:
```
# Extract training set, validation set and test set
X, y = df.filter(regex="[^Trend]"), df.Trend

# Label encode all y
lbe = LabelEncoder()
y = lbe.fit_transform(y)

X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, shuffle=False
)
X_train, X_val, y_train, y_val = train_test_split(
    X_train, y_train, test_size=0.1, shuffle=False
)

X_train_size = len(X_train)
X_test_size = len(X_test)
X_val_size = len(X_val)

# Using MinMaxScaler (feature_range could be 0.1 to 0.9)
scaler = MinMaxScaler()
X_train = scaler.fit_transform(X_train)
X_val = scaler.transform(X_val)
X_test = scaler.transform(X_test)
```

1.0.3 Construct dataset

For each trend prediction, the model exmamines all the data (i.e., close, volume…) in the last 60 days.

[9]:
```
feature_days = 60

all_X = np.concatenate([X_train, X_val, X_test])
merged_X = []
y_train = y_train[feature_days:]
merged_y = np.concatenate([y_train, y_val, y_test])

for i in range(feature_days, len(all_X)):
    merged_X.append(all_X[i-feature_days:i])

merged_X = np.array(merged_X)

val_index_start = X_train_size-feature_days
test_index_start = val_index_start + X_val_size

X_train, y_train = merged_X[:val_index_start], merged_y[:val_index_start]
X_val, y_val = merged_X[val_index_start:test_index_start],␣
↪merged_y[val_index_start:test_index_start]
X_test, y_test = merged_X[test_index_start:], merged_y[test_index_start:]
```

Build the LSTM model (many to one) A LSTM model

[ ]:
```
model = Sequential()
model.add(LSTM(128, input_length=X_train.shape[1], input_dim=X_train.shape[2],␣
↪return_sequences = True))
model.add(Dropout(0.4))
model.add(LSTM(128, return_sequences = True))
model.add(Dropout(0.4))
model.add(LSTM(128, return_sequences = True))
model.add(Dropout(0.4))
model.add(LSTM(128, return_sequences = False))
model.add(Dropout(0.4))
model.add(Dense(20, activation="relu"))
model.add(Dense(1, activation='relu'))

opt = Adam(lr = 0.0001)
reduce_lr = ReduceLROnPlateau(monitor= "loss", factor = 0.5, patience =␣
↪10,min_lr = 0.000001, verbose = 1)
monitor_es = EarlyStopping(monitor= "loss", patience = 25,␣
↪restore_best_weights= False, verbose = True)
model.compile(loss='binary_crossentropy', optimizer=opt, metrics=["accuracy"])
callback = EarlyStopping(monitor="loss", patience=10, verbose=1, mode="auto")
```

```
model.fit(X_train, y_train, validation_data=(X_val, y_val), epochs=300,␣
↪batch_size=16, callbacks = [reduce_lr, monitor_es])
```

[11]:
```
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
lstm (LSTM)                  (None, 60, 128)           73216
_________________________________________________________________
dropout (Dropout)            (None, 60, 128)           0
_________________________________________________________________
lstm_1 (LSTM)                (None, 60, 128)           131584
_________________________________________________________________
dropout_1 (Dropout)          (None, 60, 128)           0
_________________________________________________________________
lstm_2 (LSTM)                (None, 60, 128)           131584
_________________________________________________________________
dropout_2 (Dropout)          (None, 60, 128)           0
_________________________________________________________________
lstm_3 (LSTM)                (None, 128)               131584
_________________________________________________________________
dropout_3 (Dropout)          (None, 128)               0
_________________________________________________________________
dense (Dense)                (None, 20)                2580
_________________________________________________________________
dense_1 (Dense)              (None, 1)                 21
=================================================================
Total params: 470,569
Trainable params: 470,569
Non-trainable params: 0
_________________________________________________________________
```

[12]:
```
predictions = model.predict(X_test)
for i in range(0, len(predictions)):
    if predictions[i][0] > 0.5:
        predictions[i][0] = 1
    else:
        predictions[i][0] = 0
```

[13]:
```
from sklearn.metrics import accuracy_score
accuracy_score(predictions, y_test)
```

[13]:
```
0.5132743362831859
```

Although the score is not accurate enough, some analysis could be made.

1. Maybe it is easier to predict the actual price of the stock I have did another model using

LSTM to predict the price of Tesla closing price and it have a pretty good result (RMSE = 7.6). Maybe it is easier to predict a similar price, but harder the predict the trend of the next day price.

2. Adding the additional information of date (day, month, day of week, year) seems to improve the score. There maybe some correlation about the date and the trend of the price.
3. The open, close, adj close, high, and low are correlated with each other, so only the close is needed in the feature set. The result is similar with or without the extra data.

[ ]: