FA 690: Multiple Machine Learning Models and LSTM to Predict Stock Prices

Zongqi Shen **Instructor: Zachary Feinstein**

Overview:

In this assignment I did two parts to predict General Motors stock price:

1. First part: try to use different *Machine Learning methods* (Supervised Learning) to predict n day's stock price based on n-1 day's 10 factors. These factors including: *VIX; SP500 Index; NASDAQ Index; gold price; oil price; Tuopu; FCX; AA; rolling_10; GM_Volume*. Here I want to mention that Ningbo Top Group Co., Ltd. is a technology-leading auto parts enterprise listed on the Shanghai Stock Exchange (stock code: 601689). manufacture. His clients include dozens of companies including GM; AUDI; BMW; VALEO etc. FCX; AA these two companies are related to the production of non-ferrous metals, because auto parts like Automobile three-way catalytic converter may use non-ferrous metals. These are the reasons why I choose these indexes.
2. Second: Use deep learning LSTM to predict General Motors stock price

First Part: *Machine Learning methods*

- **Data Description and Data cleaning:**

1. Data source: we can get the data from Yahoo finance API in python "pandas_datareader".
2. Data initialization: to predict n day's stock price, I set a target index. If the stock price tomorrow is higher than today, I set the target to 1 and vice versa. Set 2015-04-01 to 2021-12-06 as training set, 2021-12-07 to 2022-08-31 as test set

3. Feature Engineering: I use two ddifferent preprocessing method: StandardScaler and MinMaxScaler for dataset to compare result.

- **Machine Learn Models**: Because The stock movement I want to predict is a binary variable so I need some classification models.

    a) *Random Forest and XGBoost*: *Random Forest* is an ensemble learning method for classification, regression and other tasks that operates by constructing a multitude of decision trees at training time. Random forests generally outperform decision trees, but their accuracy is lower than gradient boosted trees. *XGBoost* is an optimized distributed gradient boosting library designed to be highly efficient, flexible, and portable. It implements machine learning algorithms under the Gradient Boosting framework.

    b) *K-Nearest Neighbors algorithm (k-NN)* is used for classification and regression. In both cases, the input consists of the k closest training examples in a data set. In *k-NN* classification: the output is a class membership. An object is classified by a plurality vote of its neighbors, with the object being assigned to class most common among its k nearest neighbors. If k = 1, then object is simply assigned to the class of that single nearest neighbor.

    c) *Support-Vector Machines* are supervised learning models with associated learning algorithms that analyze data for classification and regression analysis. Given a set of training examples, each marked as belonging to one of two categories, an *SVM* training algorithm builds a model that assigns new examples to one category or the other, making it a non-probabilistic binary linear classifier (although methods such as Platt scaling exist to use *SVM* in a probabilistic classification setting. *SVM* maps training examples to points in space to maximize the width of the gap between the two

categories. New examples are then mapped into that same space and predicted to belong to a category based on which side of the gap they fall.

- **Model results**

a) Accuracy before Hyperparameter tuning

Accuracy (Table 1)

	Random Forest	*XGBoost*	*K-NN*	*SVM*
Standardization	0.52601	0.53179	0.57225	0.51445
Normalization	0.52601	0.53179	0.55491	0.51445

MSE

	Random Forest	*XGBoost*	*K-NN*	*SVM*
Standardization	0. 47398	0.468208	0.42774	0.48554
Normalization	0.47398	0.468208	0.44508	0.48554

From the data above we can see that SVM has lowest accuracy and K-NN using Standardization has best accuracy score.

b) Hyperparameter tuning process ***---- APPENDIX 1***

1. Random forest: The technique of cross validation (CV) is best explained by example using the most common method, K-Fold CV. At the meantime, I set parameter max depth loop from 10 all the way to 100, estimators 120, 200, 300.

   After tuning, the best depth is 30, best estimators is 200.

2. XGBoost: max depth of the tree loop from 3 to 10, step is 2. Min child weight from 1 to 6, step is 2. After tuning, the best max depth is 3, min child weight is 3.

3. SVM: Because of the bad tuning result, I'll not describe here.

d) Accuracy after Hyperparameter tuning:

The MSE is a measure of the quality of an estimator. As it is derived from the square of Euclidean distance, it is always a positive value that decreases as the error approaches zero. The higher the accuracy the better. The lower the Mean Squared Error, the better. Here we can see XGBoost has the best result

Accuracy (Table 2)

	Random Forest	*XGBoost*	*K-NN*	*SVM*
Standardization	0.51445	0.59537	0.57225	0.46820
Normalization	0. 51445	0. 59537	0.55491	0.46820

MSE

	Random Forest	*XGBoost*	*K-NN*	*SVM*
Standardization	0. 47398	0.40462	0.42774	0.53179
Normalization	0.47398	0.40462	0.44508	0.53179

- **Next steps:**

Try to use different Hyperparameter tuning methods.

Second Part: *Long short-term memory* (*LSTM*):

- **Data Description and Data cleaning:**

1. Feature Engineering: normalization: normalize to (0, 1)
2. Data initialization: The opening price training set of the previous (505-180=325) days, test set of the 180 days. Use the for loop to traverse the entire training set, extract the opening price of the training set for 60 consecutive days as the input feature x_train, and the data on

the 61st day as the label. The for loop constructs a total of 505-180-60=265 sets of data. Use the for loop to traverse the entire test set, extract the opening prices of the test set for 60 consecutive days as the input feature x_train, and the data on the 61st day as the label. The for loop constructs a total of 180-60=120 sets of data. ***---- APPENDIX 2***

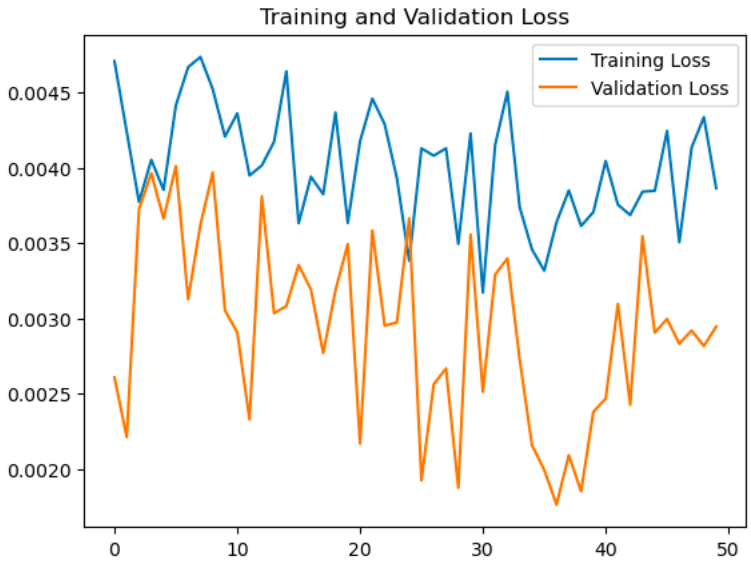
- **Build LSTM model: *---- APPENDIX 3***

Long Short-Term Memory (LSTM) Networks typically have better of time series data than normal RNN. LSTM networks have “Cell” or “Gate” with an internal recurrence outer recurrence of the RNN. Here I use 80 units; Fraction of the units to drop for the linear transformation of the inputs I use 0.2.

- **Model results:**

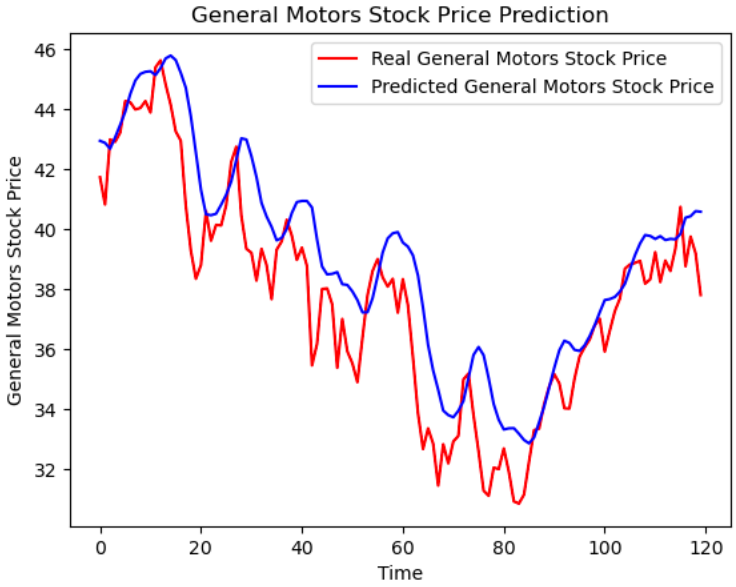
1. Training and Validation Loss:

In the model, we define Loss function with mean squared error. The application only observes the loss value, not the accuracy rate, so delete the metrics option, and only display the loss value when each epoch iteration is displayed.



2. General Motors Stock Price Movement Prediction:

We use test set input model to predict, Restore the predicted data---de-normalize from (0, 1) to the original range. Restoration to real data --- Deformalize from (0, 1) to the original range. Draw a comparison curve between the actual data and the predicted data.



3. Conclusion:

The MSE is decreasing as the model is trained, and the predicted stock movement chart is also about the same as the real movement

- **Next steps:**

If I had more time, I would like to combine machine learning stock price ups and downs predictions; LSTM forecast stock price movements and portfolios together to make a quantitative trading strategy

APPENDIX

APPENDIX 1：Hyperparameter tuning process

1. RandomForest:

```
def RandomForest(X_train,X_test, y_train, y_test):
    estimator = RandomForestClassifier(random_state=42)
    para_dict = {"max_depth": [10, 20, 30, 40, 50, 60, 70, 80, 90, 100, None], "n_estimators":[120,200,300]}
    estimator = GridSearchCV(estimator, para_dict, cv = 3)
    estimator.fit(X_train,y_train)
    predicted = estimator.predict(X_test)

    #5:
    #print("confusion_matrix: \n",pd.DataFrame(confusion_matrix(y_test, predicted)))
    print("accuracy_score: \n", accuracy_score(y_test, predicted))
    print("MSE:\n", mean_squared_error(y_test,predicted))
    print("\n")
    #6:
    print("best_parameters\n", estimator.best_params_)
    print("best_estimator_\n", estimator.best_estimator_)
    print("mean_squared_error",np.sqrt(mean_squared_error(y_test, predicted)))
    #print("best_score\n", estimator.best_score_)
    #print("cv_results_\n", estimator.cv_results_)
    return None
```

✓ 0.4s

```
RandomForest(X_train_StandardScaler,X_test_StandardScaler, y_train, y_test)
```

✓ 47.1s

```
accuracy_score:
 0.48554913294797686
MSE:
 0.5144508670520231


best_parameters
 {'max_depth': 30, 'n_estimators': 200}
best_estimator_
 RandomForestClassifier(max_depth=30, n_estimators=200, random_state=42)
mean_squared_error 0.7172523036226675
```

```
RandomForest(X_train_minmax,X_test_minmax, y_train, y_test)
```

✓ 46.5s

```
accuracy_score:
 0.48554913294797686
MSE:
 0.5144508670520231


best_parameters
 {'max_depth': 30, 'n_estimators': 200}
best_estimator_
 RandomForestClassifier(max_depth=30, n_estimators=200, random_state=42)
mean_squared_error 0.7172523036226675
```

XGBoost:

```
def xgb_2(X_train,X_test,y_train,y_test):
    param_test1 = {
    'max_depth':[i for i in range(3,10,2)],
    'min_child_weight':[i for i in range(1,6,2)]}
    gsearch = GridSearchCV(
    estimator = XGBClassifier(
    learning_rate =0.1,
    n_estimators=140, max_depth=5,
    min_child_weight=1,
    gamma=0,
    subsample=0.8,
    colsample_bytree=0.8,
    objective= 'binary:logistic',
    nthread=4,
    scale_pos_weight=1,
    seed=27),
    param_grid = param_test1,
    scoring='roc_auc',
    n_jobs=4,
    cv=5)
    gsearch.fit(X_train,y_train)
    predicted = gsearch.predict(X_test)
    #print("confusion_matrix: \n",pd.DataFrame(confusion_matrix(y_test, predicted)))
    print("accuracy_score: \n", accuracy_score(y_test, predicted))
    print("\n")
    print("best_parameters\n", gsearch.best_params_)
    print("mean_squared_error",mean_squared_error(y_test, predicted))
    #print("best_score\n", gsearch.best_score_)
    #print("best_estimator_\n", gsearch.best_estimator_)
    #print("cv_results_\n", estimator.cv_results_)
    return None
```

✓ 0.2s

```
xgb_2(X_train_StandardScaler,X_test_StandardScaler, y_train, y_test)
```

✓ 12.4s

```
accuracy_score:
 0.5953757225433526


best_parameters
 {'max_depth': 3, 'min_child_weight': 3}
mean_squared_error 0.4046242774566474
```

```
xgb_2(X_train_minmax,X_test_minmax, y_train, y_test)
```

✓ 8.7s

```
accuracy_score:
 0.5953757225433526


best_parameters
 {'max_depth': 3, 'min_child_weight': 3}
mean_squared_error 0.4046242774566474
```

Support Vector Machine:

```
def svm_cross_validation(X_train,X_test,y_train,y_test):
    from sklearn.model_selection import GridSearchCV
    model = SVC(kernel='rbf', probability=True)
    param_grid = {'C': [1e-3, 1e-2, 1e-1, 1, 10, 100, 1000], 'gamma': [0.001, 0.0001]}
    grid_search = GridSearchCV(model, param_grid, n_jobs = 8, verbose=1)
    grid_search.fit(X_train, y_train)
    best_parameters = grid_search.best_estimator_.get_params()
    model = SVC(kernel='rbf', C=best_parameters['C'], gamma=best_parameters['gamma'], probability=True)
    model.fit(X_train, y_train)
    predicted = model.predict(X_test)
    #print("confusion_matrix: \n",pd.DataFrame(confusion_matrix(y_test, predicted)))
    print("accuracy_score: \n", accuracy_score(y_test, predicted))
    print("mean_squared_error",mean_squared_error(y_test, predicted))
    return None
```

✓ 0.2s

```
svm_cross_validation(X_train_StandardScaler,X_test_StandardScaler, y_train, y_test)
```

✓ 12.7s

```
Fitting 5 folds for each of 14 candidates, totalling 70 fits
```

```
svm_cross_validation(X_train_minmax,X_test_minmax, y_train, y_test)
```

✓ 8.6s

```
Fitting 5 folds for each of 14 candidates, totalling 70 fits
accuracy_score:
 0.4682080924855491
mean_squared_error 0.5317919075144508
```

APPENDIX 2：Data initialization and train test dataset split.

```
model = tf.keras.Sequential([
    LSTM(80, return_sequences=False),
    Dropout(0.2),
    Dense(1)
])

model.compile(optimizer=tf.keras.optimizers.Adam(0.001),
              loss='mean_squared_error')  # Loss function with mean squared error

# The application only observes the loss value, not the accuracy rate, so delete the metrics option, and only display the loss value when each e
checkpoint_save_path = "./checkpoint/rnn_stock.ckpt"

if os.path.exists(checkpoint_save_path + '.index'):
    print('-------------load the model-----------------')
    model.load_weights(checkpoint_save_path)

cp_callback = tf.keras.callbacks.ModelCheckpoint(filepath=checkpoint_save_path,
                                                 save_weights_only=True,
                                                 save_best_only=True,
                                                 monitor='val_loss')

history = model.fit(x_train, y_train, batch_size=64, epochs=50, validation_data=(x_test, y_test), validation_freq=1,
                    callbacks=[cp_callback])

model.summary()
```

APPENDIX 3：Build LSTM model:

```
# Use the for loop to traverse the entire training set, extract the opening price of the training set for 60 consecutive days as the input feature
for i in range(60, len(training_set_scaled)):
        x_train.append(training_set_scaled[i - 60:i, 0])
        y_train.append(training_set_scaled[i, 0])
# Shuffle the training set
np.random.seed(7)
np.random.shuffle(x_train)
np.random.seed(7)
np.random.shuffle(y_train)
tf.random.set_seed(7)
# Change the training set from list format to array format
x_train, y_train = np.array(x_train), np.array(y_train)

# Make x_train meet the RNN input requirements: [the number of samples sent, the number of time expansion steps of the loop kernel, the number of
# The entire data set is sent here, and the number of samples sent is x_train.shape[0], which is 265 sets of data; 60 opening prices are input, an
x_train = np.reshape(x_train, (x_train.shape[0], 60, 1))
#do the same thing to x_test
# Use the for loop to traverse the entire test set, extract the opening prices of the test set for 60 consecutive days as the input feature x_trai
for i in range(60, len(test_set)):
    x_test.append(test_set[i - 60:i, 0])
    y_test.append(test_set[i, 0])
# The test set is changed to array and reshaped to meet the RNN input requirements: [number of input samples, number of time expansion steps of lo
x_test, y_test = np.array(x_test), np.array(y_test)
x_test = np.reshape(x_test, (x_test.shape[0], 60, 1))
```