Task 7: Extension

Muhammad Ali, 103960437, COS30018, 06/10/2024, Tutor: Ru Jia.

Requirements / Deliverables –

- Research of the potential approaches for predicting companies' stock prices/trends.
- Implementation of your selected idea and the evaluation results you have obtained when assessing the performance of the selected idea.

Result –

Summary of research findings –

1. Linear Regression

Linear regression is a basic machine learning algorithm that can be implemented on this dataset. The linear regression model returns an equation determining the relationship between the independent and dependent variables.

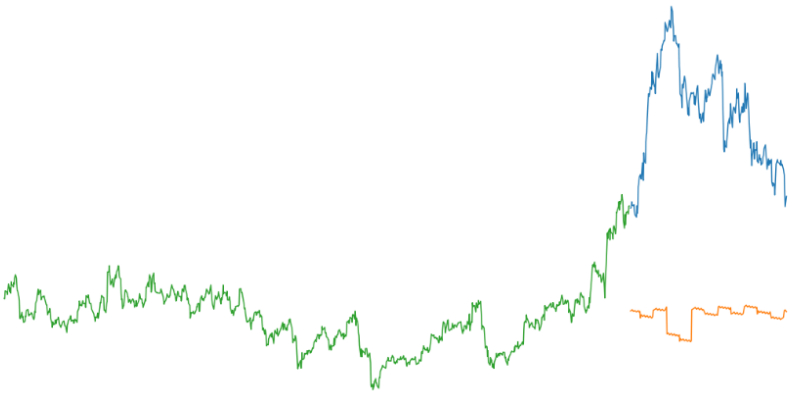
The equation for linear regression can be written as:

$$Y = \theta_1 X_1 + \theta_2 X_2 + \dots \theta_n X_n$$

X1, x2,....Xn represent the independent variables while the coefficients O1, O2,..... On represent the weights.

Disadvantages –

A problem with using regression algorithms in this case is that the model overfits the data from the dataset.



Model prediction – orange.

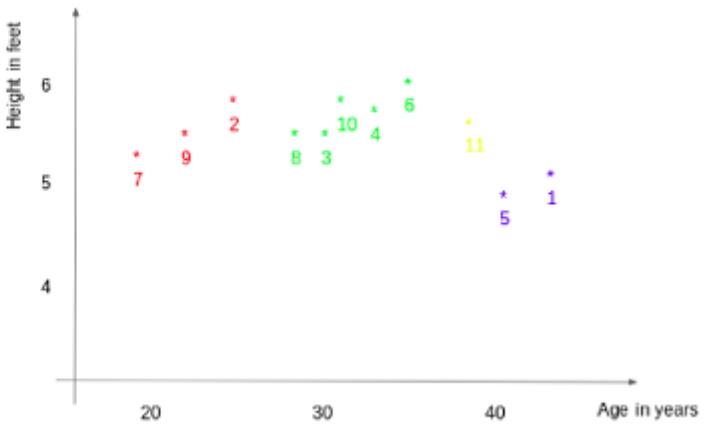
2. K-Nearest Neighbours

kNN finds the similarity between new and old data points, based on the independent variables.

For example:

Consider the height and age of 11 people (Based on the given features 'Age' and 'Height:

ID	Age	Height	Weight
1	45	5	77
2	26	5.11	47
3	30	5.6	55
4	34	5.9	59
5	40	4.8	72
6	36	5.8	60
7	19	5.3	40
8	28	5.8	60
9	23	5.5	45
10	32	5.6	58
11	38	5.5	?



To determine the weight for ID #11. kNN considers the weight of the nearest neighbours of this ID.

The weight of ID #11 is predicted to be the average of its neighbours. If we consider three neighbours (k=3) for now, the weight for ID11 would be:

= (77 + 72 + 60) / 3

= 69.66 kg.

Implementation –

```
#importing libraries
from sklearn import neighbors
from sklearn.model_selection import GridSearchCV
from sklearn.preprocessing import MinMaxScaler
scaler = MinMaxScaler(feature_range=(0, 1))
```

Using scaled data –

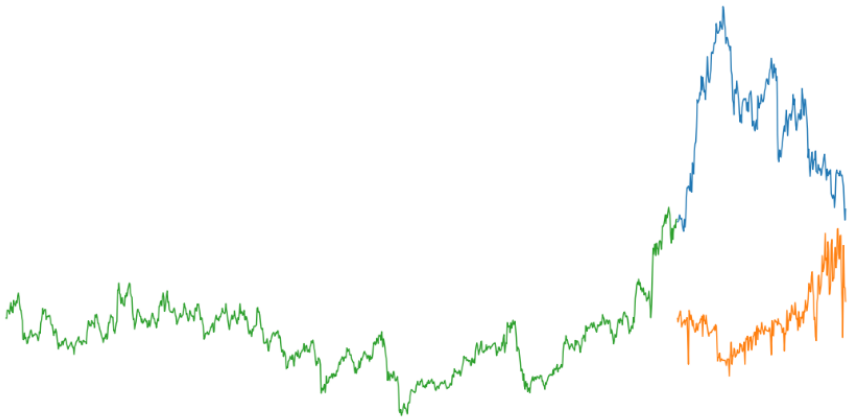
```
x_train_scaled = scaler.fit_transform(x_train)
x_train = pd.DataFrame(x_train_scaled)
x_valid_scaled = scaler.fit_transform(x_valid)
x_valid = pd.DataFrame(x_valid_scaled)
```

Finding the best parameters –

```
params = {'n_neighbors':[2,3,4,5,6,7,8,9]}
knn = neighbors.KNeighborsRegressor()
model = GridSearchCV(knn, params, cv=5)
```

Fit the model and make predictions –

```
model.fit(x_train,y_train)
preds = model.predict(x_valid)
```



kNN RESULT –

k-Nearest Neighbours is a regression technique as well and like linear regression. This might be because instead of considering the previous values from the point of prediction, the model considers the values from the same date a month ago or the same date/month a year ago.

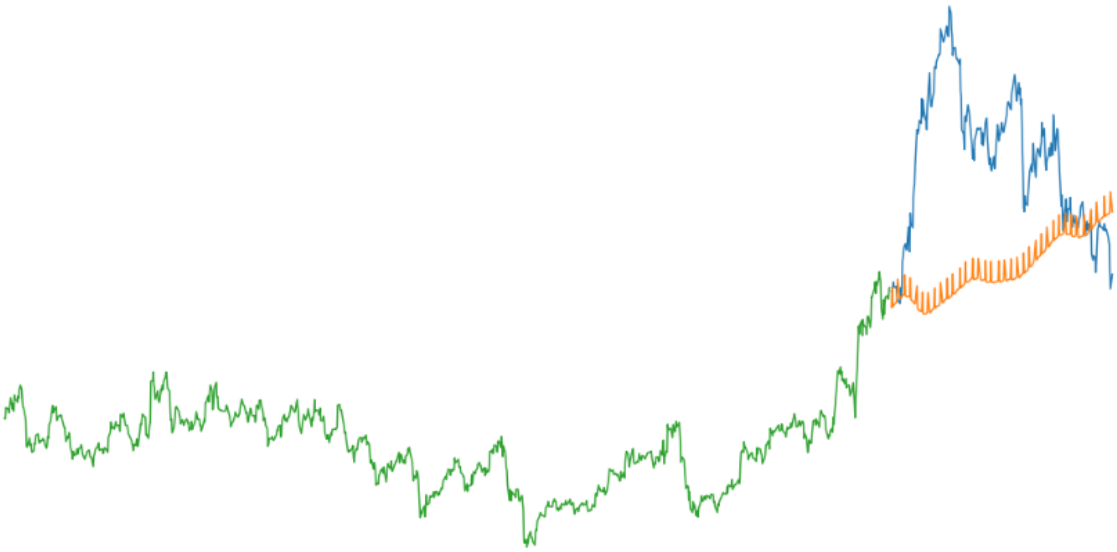
Due to this *regression models* lead to overfitting.

3. Prophet

Prophet is a time series forecasting library that requires no data preprocessing and is extremely simple to implement.

Prophet tries to capture the seasonality in past data and works well when the dataset is large.

Prophet Results –



Disadvantages –

Unfortunately, stock prices are hardly ever seasonal in nature and are often affected by outside factors like world events, so seasonality is often hard to capture and use for stock predictions. Hence, forecasting techniques like Prophet, ARIMA, and SARIMA would not show good results for stock prediction.

Implementation of selected idea –

Backpropagation –

Backpropagation is a multilayer feed-forward network. Backpropagation is given a function to model; the neural network modifies the *internal weightings* of input to produce an expected output. The system is trained, where the *error* between the system's output and the expected output is used to modify its *internal state.*

A backpropagation model can be used for both *regression* and *classification* problems but judging by the results of the regression models I'm using classification to predict.

Implementation –

To implement backpropagation using TensorFlow we need to implement a feedforward neural network, backpropagation will then update the weights during training using the specified *loss function* and *optimiser*.

Steps –

1. **Add a function to create a feedforward model.**

In data_processing.py, we add a new function to create a feedforward model (utilises the Dense, and Flatten functions from TensorFlow library):

```
 def create_feedforward_model(input_shape, hidden_layers, hidden_units,
activation_functions, output_size,
                             loss_function='mse', optimizer='adam'):
    """
    Creates a feedforward neural network model.

    Parameters:
    - input_shape (tuple): Shape of the input data (timesteps, features).
    - hidden_layers (int): Number of hidden layers.
    - hidden_units (list of int): Number of units in each hidden layer.
    - activation_functions (list of str): Activation functions for each
hidden layer.
    - output_size (int): Number of units in the output layer.
    - loss_function (str): Loss function to use for training.
    - optimizer (str): Optimizer to use for training.

    Returns:
    - model (Sequential): Compiled Keras feedforward model.
    """
    model = Sequential()
    model.add(Flatten(input_shape=input_shape))

    for i in range(hidden_layers):
        model.add(Dense(hidden_units[i],
activation=activation_functions[i]))

    model.add(Dense(output_size))  # Output layer
    model.compile(optimizer=optimizer, loss=loss_function)

    return model
```

2. **Integrate FF model into main program.**

The feedforward model is implemented into the previous program that uses LSTM and SARIMA in an ensemble approach to provide predictions.

1. **The data input shape is adjusted for the backpropagation model:**

```
input_shape_ff = (PREDICTION_DAYS, X_train_scaled.shape[2])
hidden_layers = 3
hidden_units = [100, 50, 25]
activation_functions_ff = ['relu', 'relu', 'relu']
```

2. **I create and compile the backpropagation model; I save its best model output after model training:**

```
3. # Create and compile the backpropagation model
   backpropagation_model = create_backpropagation_model(
       input_shape=input_shape_ff,
       hidden_layers=hidden_layers,
       hidden_units=hidden_units,
       activation_functions=activation_functions_ff,
       output_size=FUTURE_STEPS,
       loss_function='huber',
       optimizer='adam'
   )
4. # Define early stopping and checkpoint for the backpropagation model
   checkpoint_ff = ModelCheckpoint('best_ff_model.h5',
   save_best_only=True, monitor='val_loss')

   # Train the backpropagation model
   history_ff = backpropagation_model.fit(
       X_train_scaled, y_train_scaled,
       epochs=100, batch_size=32, verbose=1,
       validation_split=0.2, callbacks=[early_stopping, checkpoint_ff]
5. # Predict with the backpropagation model
   backpropagation_model.load_weights('best_ff_model.h5')
   predicted_ff = backpropagation_model.predict(X_test_scaled)
```

I implemented the feedforward network and adjusted the weights according to each model's input to the predictions output.

```
# Combine DL, SARIMA, and Feedforward predictions using weighted average
weight_dl = 0.5
weight_sarima = 0.2
weight_ff = 0.3
```

The LSTM model has a 50% weighting to the output, the SARIMA model has 20% due to its seasonal predictive output, and the new backpropagation model has a 30% weight to the prediction output.

Results –

Backpropagation provides a positive result to the stock prediction using the stock price, due to its feedforward network whose output is backpropagated into the model by accounting for its error/loss, therefore improving on the model's previous predictions and outputting an improved result. Backpropagation combined with the LSTM/SARIMA ensemble method produces a complete ensemble approach for stock prediction.