UNVEILING THE UNEXPECTED: STOCK MARKET OUTLIER DETECTION WITH PYTHON

YASHIT GUPTA (21MC3016)

Data Mining and Outlier Detection

This project tackles a critical challenge within the stock market: identifying outliers. Outliers are data points that deviate significantly from the expected market behavior. By pinpointing these outliers using historical stock data, we aim to achieve two key goals:

- Develop a robust system: We will leverage Python's capabilities to construct a system that effectively identifies outliers within the data. This system will serve as a valuable tool for market analysis, utilizing techniques like:
    - Data Preprocessing: Cleaning and preparing the data for analysis.
    - Feature Engineering: Creating new informative features from existing data, such as moving averages and technical indicators (RSI, Bollinger Bands).
    - Outlier Detection Methods: Employing techniques like Z-scores, Interquartile Range (IQR), and Isolation Forest to pinpoint outliers.
- Gain actionable insights: By analyzing the identified outliers, we hope to gain valuable insights into potential market shifts. These insights can empower investors to make informed decisions based on a deeper understanding of market behavior.

In the following sections, we'll delve into the methodologies employed to achieve these goals. We'll explore data mining techniques, outlier detection methods mentioned above, and the power of machine learning (specifically a Random Forest classifier) to build a comprehensive outlier detection system for the stock market.

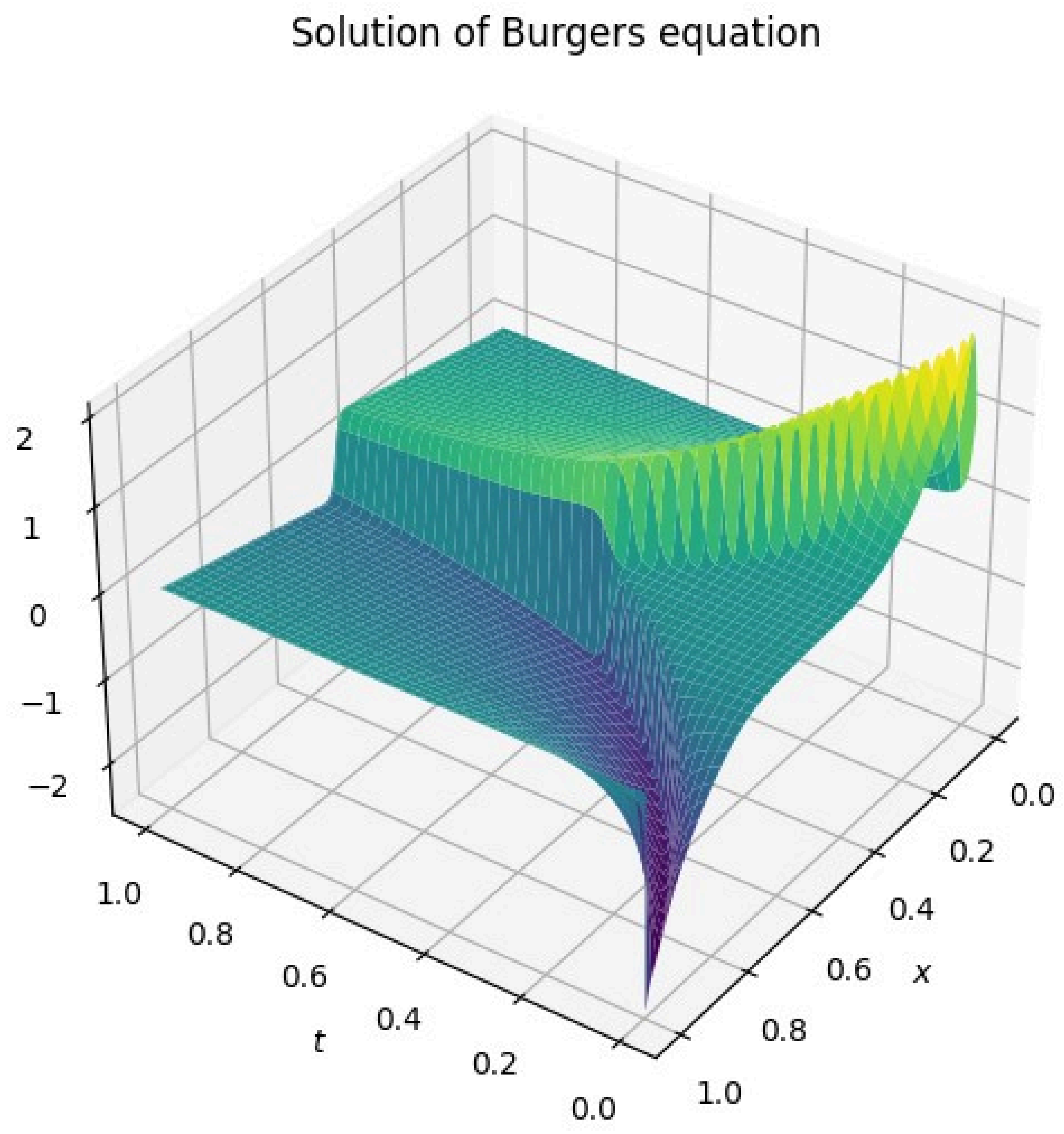
Adam optimizer was used, Activation Function:- tanh

No of layers=8, No of neuron in each layer=20

```
def init_model(num_hidden_layers=8, num_neurons_per_layer=20):
    # Initialize a feedforward neural network
    model = tf.keras.Sequential()

    # Input is two-dimensional (time + one spatial dimension)
    model.add(tf.keras.Input(2))

    # Introduce a scaling layer to map input to [lb, ub]
    scaling_layer = tf.keras.layers.Lambda(
                lambda x: 2.0*(x - lb)/(ub - lb) - 1.0)
    model.add(scaling_layer)

    # Append hidden layers
    for _ in range(num_hidden_layers):
        model.add(tf.keras.layers.Dense(num_neurons_per_layer,
            activation=tf.keras.activations.get('tanh'),
            kernel_initializer='glorot_normal'))

    # Output is one-dimensional
    model.add(tf.keras.layers.Dense(1))

    return model
```

```
# Function to compute the loss
def compute_loss(model, X_r, X_data, u_data):
    r = get_r(model, X_r)
    phi_r = tf.reduce_mean(tf.square(r))
    loss = phi_r

#neuwman boundary
    ab=tf.expand_dims(X_r[:,0],axis=1)
    min_value=tf.math.reduce_min(X_r[:,1])
    cd=tf.ones((500,1),dtype=DTYPE)*min_value
    abcd=tf.concat([ab,cd],axis=1)
    u_pred_low = model(abcd)
    loss += tf.reduce_mean(tf.square(u_data[1] - u_pred_low))

    ab=tf.expand_dims(X_r[:,0],axis=1)
    max_value=tf.math.reduce_max(X_r[:,1])
    cd=tf.ones((500,1),dtype=DTYPE)*max_value
    abcd=tf.concat([ab,cd],axis=1)
    u_pred_high = model(abcd)
    loss += tf.reduce_mean(tf.square(u_data[1] - u_pred_high))



    u_pred = model(X_data[0])
    loss += tf.reduce_mean(tf.square(u_data[0] - u_pred))
    return loss

# Function to compute gradients
def get_grad(model, X_r, X_data, u_data):
    with tf.GradientTape(persistent=True) as tape:
        tape.watch(model.trainable_variables)
        loss = compute_loss(model, X_r, X_data, u_data)
    g = tape.gradient(loss, model.trainable_variables)
    del tape
    return loss, g
```

Loss Function



SOLVED USING NEURAL NETWORK

```
It 38800: loss = 2.43962905e-03
It 38850: loss = 2.38862191e-03
It 38900: loss = 3.10197985e-03
It 38950: loss = 2.37775943e-03
It 39000: loss = 2.55254447e-03
It 39050: loss = 2.33828533e-03
It 39100: loss = 2.47856416e-03
It 39150: loss = 3.17686889e-03
It 39200: loss = 2.29859492e-03
It 39250: loss = 2.98059359e-03
It 39300: loss = 2.26472015e-03
It 39350: loss = 2.52369419e-03
It 39400: loss = 2.25270074e-03
It 39450: loss = 2.39847507e-03
It 39500: loss = 2.22351309e-03
It 39550: loss = 2.19581253e-03
It 39600: loss = 4.07151226e-03
It 39650: loss = 2.18665018e-03
It 39700: loss = 2.18407507e-03
It 39750: loss = 2.28683301e-03
It 39800: loss = 2.19064066e-03
It 39850: loss = 2.17124098e-03
It 39900: loss = 2.11950601e-03
It 39950: loss = 6.25137836e-02
It 40000: loss = 2.45129205e-02

Computation time: 2813.8123185634613 seconds
```

LOSS AFTER 38000 EPOCH

USING NEWMAN CONTROL ON EQUATION

We will approach the problem using nonlinear Neuman Bounday Control:-

$$w_x(0,t) = \frac{1}{\epsilon}\left(c_0 + \frac{W_d}{2} + \frac{1}{9c_0}w^2(0,t)\right)w(0,t),$$

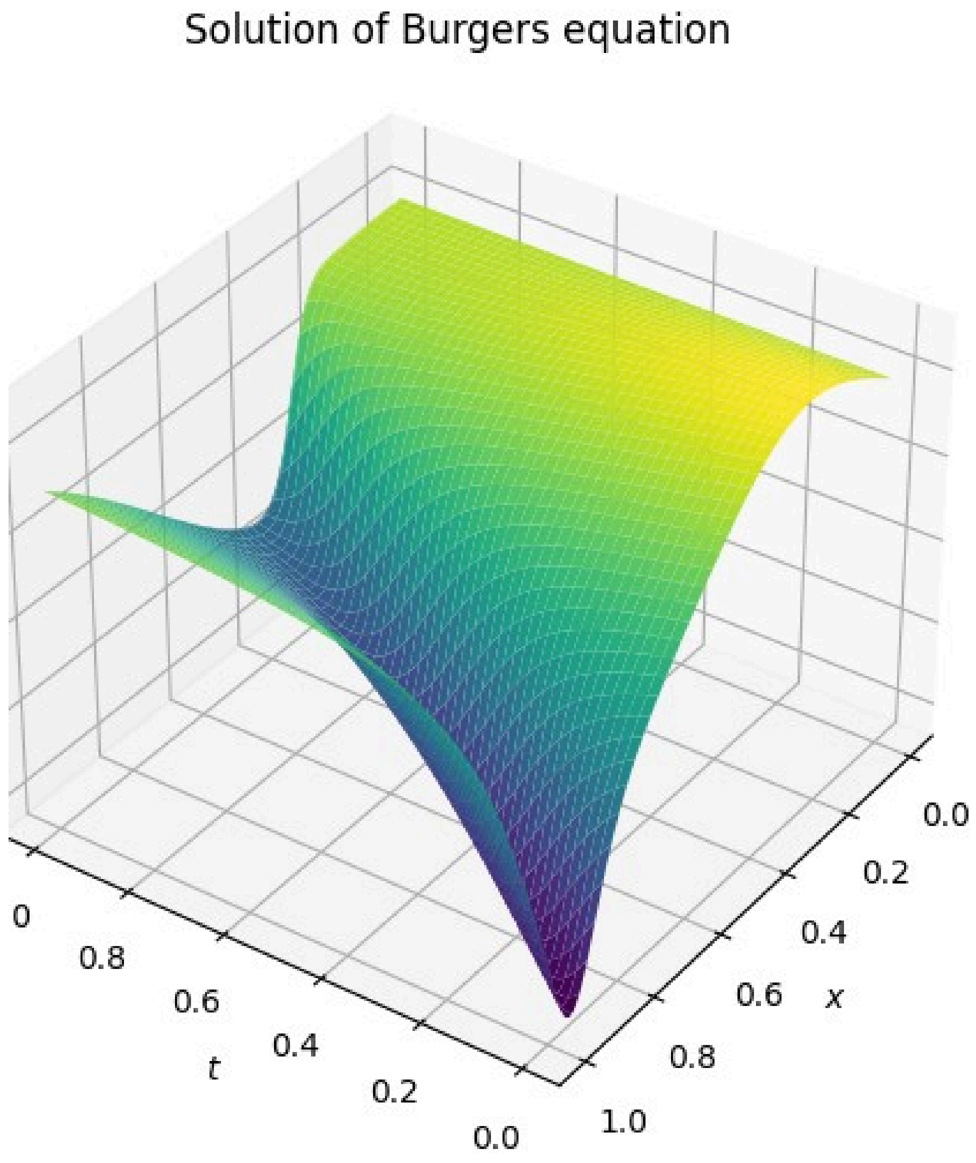
$$w_x(1,t) = -\frac{1}{\epsilon}\left(c_1 + \frac{1}{9c_1}w^2(1,t)\right)w(1,t),$$

```
# Function to compute the loss
def compute_loss(model, X_r, X_data, u_data):
    r = get_r(model, X_r)
    phi_r = tf.reduce_mean(tf.square(r))
    loss = phi_r

#neuwman boundary
    ab=tf.expand_dims(X_r[:,0],axis=1)
    min_value=tf.math.reduce_min(X_r[:,1])
    cd=tf.ones((5000,1),dtype=DTYPE)*min_value
    abcd=tf.concat([ab,cd],axis=1)
    u_pred_low = model(abcd)
    eqn1=(1.51+((u_pred_low/0.09)*u_pred_low))/0.1
    loss += tf.reduce_mean(tf.square((u_data[1] - u_pred_low)-eqn1))

    ab=tf.expand_dims(X_r[:,0],axis=1)
    max_value=tf.math.reduce_max(X_r[:,1])
    cd=tf.ones((5000,1),dtype=DTYPE)*max_value
    abcd=tf.concat([ab,cd],axis=1)
    u_pred_high = model(abcd)
    eqn1=(0.01+((u_pred_low/0.09)*u_pred_high))/0.1
    loss += tf.reduce_mean(tf.square((u_data[1] - u_pred_high)-eqn1))



    u_pred = model(X_data[0])
    loss += tf.reduce_mean(tf.square(u_data[0] - u_pred))
    return loss

# Function to compute gradients
def get_grad(model, X_r, X_data, u_data):
    with tf.GradientTape(persistent=True) as tape:
        tape.watch(model.trainable_variables)
        loss = compute_loss(model, X_r, X_data, u_data)
    g = tape.gradient(loss, model.trainable_variables)
    del tape
    return loss, g
```

Loss function

RESULT AND CONCLUSION

```
It 18200: loss = 2.28068680e+02
It 18250: loss = 2.28053146e+02
It 18300: loss = 2.28084946e+02
It 18350: loss = 2.28052551e+02
It 18400: loss = 2.28087891e+02
It 18450: loss = 2.28052963e+02
It 18500: loss = 2.28090027e+02
It 18550: loss = 2.28052551e+02
It 18600: loss = 2.28051758e+02
It 18650: loss = 2.28054123e+02
It 18700: loss = 2.28050720e+02
It 18750: loss = 2.28051727e+02
It 18800: loss = 2.28050461e+02
It 18850: loss = 2.28095093e+02
It 18900: loss = 2.28050079e+02
It 18950: loss = 2.28078537e+02
It 19000: loss = 2.28051743e+02
It 19050: loss = 2.28049500e+02
It 19100: loss = 2.28049210e+02
It 19150: loss = 2.28049896e+02
It 19200: loss = 2.28048813e+02
It 19250: loss = 2.28080368e+02
It 19300: loss = 2.28048508e+02
It 19350: loss = 2.28054871e+02
It 19400: loss = 2.28049057e+02
It 19450: loss = 2.28055161e+02
It 19500: loss = 2.28048599e+02
It 19550: loss = 2.28047241e+02
It 19600: loss = 2.28053116e+02
It 19650: loss = 2.28047073e+02
It 19700: loss = 2.28073334e+02
It 19750: loss = 2.28047745e+02
It 19800: loss = 2.28046295e+02
It 19850: loss = 2.28053238e+02
It 19900: loss = 2.28046371e+02
It 19950: loss = 2.28045761e+02
It 20000: loss = 2.28054932e+02
```



SINCE THE ERROR WAS TOO HIGH IT MEANS BOUNDARY CONTROL WAS NOT IMPLEMENTED PROPERLY. IN FUTURE WE WILL TRY TO REDUCE ERROR USING SOME OTHER LOSS FUNCTION AND OPTIMISER

ERROR AFTER 18200 EPOCH