Regression Model Of Foreign Exchange Sales

Burak Göksu

INTRODUCTION

The problem aims to predict the value of a foreign currency based on the values of other foreign currencies. The dataset provided includes records of 8 different foreign currencies from January 3, 2005, to November 28, 2023. Seven of these currencies serve as input features, and the eighth one is designated as the output variable. The task is to develop a model that can effectively predict the output currency based on the input features.

METHODOLOGY

The proposed methodology involves several phase:
Code Phase 1: Reading and Querying the Dataset
Code Phase 2: Cost Function, Derivative, and Optimizer
Code Phase 3: Preventing Overfitting with Hybrid L2/L1 Penalty
Code TEST Phase: Prediction with 7 Input Currencies

DISCUSSION ABOUT PROBLEM

The problem involves a regression task where the goal is to predict a continuous output variable (foreign currency value) based on multiple input features. The dataset's temporal nature adds a time-series aspect to the problem, which may require careful consideration during model development.

ALTERNATIVES AND CORRECTIONS

Feature Engineering: Consider additional feature engineering techniques to capture temporal patterns or trends in the data.
Model Evaluation: Implement robust model evaluation techniques such as cross-validation to ensure the model's generalization performance.
Hyperparameter Tuning: Explore different hyperparameter values to optimize the model's performance.

EXPLAIN OF CODE

I used only numpy and matplotlib libraries in this project. First, I converted the dataset from Excel type to CSV type and removed the empty rows so that they would not disrupt my model. I imported my csv file to my project with numpy and I shuffled the output to get better results. Then I divided the data into two: train and test. I chose the z score model to normalize. Z-score normalization aims to transform the data set into a standard normal distribution by measuring the mean of a data set around μ and its standard deviation in units of σ. This normalization is used in the data preprocessing step for many machine learning algorithms such as regression models. This normalization process helps algorithms perform better if different features in the data set are at different scales. Additionally, Z-score normalization helps the model generalize better by reducing scale differences that affect the weight coefficients in regression models. There are 6 inputs and one output in our data. i.e. X_train = 0,1,2,3,4,5 and 6th columns, Y_train = 7th column. I performed z score normalization while introducing the data to X and Y. I have 3 different models in this project. L1 penalty(Lasso), L2 penalty(Ridge) and hybrid(L1-L2) model. First of all, I will explain how I calculated the cost for my L1 model. Determines the m value that represents the number of samples in the data set. A column is added to the data set X. This is a column that represents the constant term (bias) and the free variable (intercept) of the model. It makes the predictions of the model. Theta is the vector containing the regression coefficients. The error (residual) between the actual values and the model's predictions is calculated. The error squares are averaged, which expresses the cost function of traditional linear regression. The L1 regularization term is added for Lasso regression. alpha is the regularization parameter, and this term controls the complexity of the model by moving the feature weights closer to zero. The traditional regression cost function and the lasso

regularization term are summed, which represents the total cost for the lasso regression. I preferred the gradient descent model for the optimizer. This algorithm uses an iterative approach to reach the minimum or maximum value of a function. It usually aims at minimization of the function, as it is often used to reach the minimum point. Determines the m value that represents the number of samples in the data set. A column is added to the data set X. This is a column that represents the constant term (bias) and expresses the free variable (intercept) of the model. The weights are initialized randomly. This ensures that the algorithm starts at a point at the beginning. Initially sets the cost to infinity. A list is created to track how the cost changes in each iteration. It continues iteratively as long as a certain maximum number of iterations or a threshold value is not exceeded. Gradient is calculated for Lasso regression. The first term is the derivative of the error term; The second term is the l1 regularization term. The weights are updated. The cost is calculated with the updated weights. By checking the cost variation, it checks whether a certain threshold value has been reached. If the threshold value is reached, the algorithm is stopped. Updates the previous cost for the next iteration. Increases the number of iterations. Returns the final weights. An array is created to determine the various alpha values to use for Lasso regression. The np.logspace function creates values in a logarithmic range. A loop is started for each alpha value. The Lasso regression model is trained using the lasso_gradient_descent function. This function takes the dataset and other hyperparameters, updates the weights of the model using gradient descent and tries to find the best weights. The lasso_cost_function function is used to calculate the cost of the trained model. This evaluates how well the model fits and performs together with the regularization term. The cost value calculated for each alpha value is added to a list. This helps monitor the performance of the model for each alpha value. The np.argmin function is used to find the lowest cost among the calculated cost values. This determines the best performing alpha value. The alpha value with the lowest cost is assigned as the best performing alpha. The lowest cost value is assigned as the cost of the best performing model. As a result, it trains Lasso regression models for various alpha values, evaluates the performance of each model, and determines the regularization parameter and cost of the best-performing model. This type of process is a common practice to adjust the fit of the model using regularization methods to prevent overfitting and improve generalization performance. Lasso regression is used to test the model, evaluate the performance of the model and also make a prediction on a new data point. It trains the Lasso regression model using a separate theta value for the test set. It uses the cost function to evaluate the performance of the trained model on the test set.

The mean square error (MSE) on the test set is calculated and printed on the screen. Then, I did the following to make a prediction based on manually entered data. I created a manual data point and normalized it. I extrapolated on this new data point using the trained Lasso regression model (theta_test_lasso).

I converted the prediction result from normalized form to real value and printed it on the screen. I did these steps for prediction and plotting on the test set. Predictions are made on the test set using the trained Lasso regression model. Predictions and actual values are converted from their normalized form to actual values. A scatter plot is drawn showing the relationship between actual values and predictions. The line learned by the Lasso regression model is plotted as a red line on the graph. The L2 regression model is almost the same as what I did in the L1 model, but the cost function and gradient descent function are different. It is used to calculate the cost function for Ridge regression. Ridge regression is used by adding l2 regularization to traditional regression models. L2 regularization allows feature selection by adding squared terms to the weight coefficients, moving the weights closer to zero. Determines the m value that represents the number of samples in the data set. A column is added to the data set X. This is a column that represents the constant term (bias) and expresses the free variable (intercept) of the model. theta: Weight vector for Ridge regression. Makes predictions of the model. The error (residual) between the actual values and the model's predictions is calculated. The error squares are averaged, which represents the traditional regression cost function. For ridge regression, the l2 regularization term is added. alpha is the regularization parameter,

and this term contains the squared form of the sum of feature weights. The regularization term l2 is summed with the traditional regression cost function, which represents the total cost for Ridge regression. The gradient descent algorithm for the L2 model was like this.
Determines the m value that represents the number of samples in the data set. A column is added to the data set X. This is a column that represents the constant term (bias) and expresses the free variable (intercept) of the model.mThe weights are initialized randomly. This ensures that the algorithm starts at a point at the beginning. It initially sets the cost as infinite. It continues iteratively as long as a certain maximum number of iterations or a threshold value is not exceeded. Gradient is calculated for ridge regression. The first term is the derivative of the error term; The second term is the l2 regularization term. The weights are updated. The cost is calculated with the updated weights. By checking the cost variation, it checks whether a certain threshold value has been reached. If the threshold value is reached, the algorithm is stopped. Updates the previous cost for the next iteration. It increases the number of iterations. Returns the final weights. The hybrid_model function combines the predictions of the Lasso and Ridge models. Predictions of the Lasso model (predictions_lasso) and predictions of the Ridge model (predictions_ridge) are taken. The hybrid estimate is created using a specific weight (lasso_weight). By default, this weight puts more emphasis on the Lasso model. Hybrid predictions are used to evaluate performance on the test set. The performance of the hybrid model on the test set is measured using Lasso's cost function (lasso_cost_function).
Additionally, this performance value is visualized by comparing it with the performances of the Lasso and Ridge models.

CONCLUSION WITH FUTURE ADVISE

In conclusion, the proposed model development process involves reading and preprocessing the dataset, implementing key machine learning components, preventing overfitting, and testing the model's predictions. For future work, it is advised to: Explore advanced time-series modeling techniques if temporal patterns are crucial. Experiment with different regularization techniques and hyperparameter values for model optimization. Consider ensemble methods or more complex architectures for potential performance improvement. Overall, continuous monitoring and refinement of the model will be essential to adapt to potential changes in the data distribution over time. Additionally, collaboration with domain experts may provide valuable insights for feature selection and model interpretation. The provided code snippets will be tailored according to the dataset's specific structure, and additional adjustments may be needed based on the exploratory data analysis.

I'm attaching some code snippets below as photos.

```python
# Veriyi yükleme
data = np.genfromtxt("Doviz_Satislari.csv", delimiter=',', skip_header=1)
```

```python
def z_score_normalize(data):
    mean_val = np.mean(data)
    std_val = np.std(data)
    normalized_data = (data - mean_val) / std_val
    return normalized_data
```

```python
# Lasso regresyonu için cost function
3 usages
def lasso_cost_function(X, y, theta, alpha):
    m = len(X)
    X_b = np.c_[np.ones((m, 1)), X]
    predictions = X_b.dot(theta)
    error = predictions - y
    cost = np.mean(np.square(error)) + alpha * np.sum(np.abs(theta[1:]))
    return cost
```

```python
# Lasso regresyonu için gradient descent fonksiyonu
2 usages
def lasso_gradient_descent(X, y, alpha, learning_rate, threshold, max_iterations=10000000):
    m = len(X)
    X_b = np.c_[np.ones((m, 1)), X]  # Bias terimini ekleyelim
    theta = np.random.randn(8, 1)  # Başlangıçta rastgele ağırlıklar
    prev_cost = float('inf')  # Başlangıçta maliyeti sonsuz olarak ayarlayalım

    iteration = 0
    while iteration < max_iterations:
        gradients = 2/m * X_b.T.dot(X_b.dot(theta) - y) + alpha * np.sign(theta)
        theta = theta - learning_rate * gradients

        # Maliyet kontrolü
        cost = lasso_cost_function(X, y, theta, alpha)

        if abs(prev_cost - cost) < threshold:
            print(f"Iterasyon {iteration}: Cost change ({abs(prev_cost - cost)}) below threshold ({threshold}). Durduruluyor.")
            break

        prev_cost = cost
        iteration += 1
    return theta
```

```python
# Ridge regresyonu için cost function
3 usages
def ridge_cost_function(X, y, theta, alpha):
    m = len(X)
    X_b = np.c_[np.ones((m, 1)), X]
    predictions = X_b.dot(theta)
    error = predictions - y
    cost = np.mean(np.square(error)) + alpha * np.sum(np.square(theta[1:]))
    return cost
```

```python
# Ridge regresyonu için gradient descent fonksiyonu
2 usages
def ridge_gradient_descent(X, y, alpha, learning_rate, threshold,max_iterations=10000000):
    m = len(X)
    X_b = np.c_[np.ones((m, 1)), X]  # Bias terimini ekleyelim
    theta = np.random.randn(8, 1)  # Başlangıçta rastgele ağırlıklar
    prev_cost = float('inf')  # Başlangıçta maliyeti sonsuz olarak ayarlayalım

    iteration = 0
    while iteration < max_iterations:
        gradients = 2 / m * X_b.T.dot(X_b.dot(theta) - y) + 2 * alpha * theta
        theta = theta - learning_rate * gradients

        # Maliyet kontrolü
        cost = ridge_cost_function(X, y, theta, alpha)
        if abs(prev_cost - cost) < threshold:
            print(
                f"Iterasyon {iteration}: Cost change ({abs(prev_cost - cost)}) below threshold ({threshold}). Durduruluyor.")
            break

        prev_cost = cost
        iteration += 1

    return theta
```

```python
# Lasso ve Ridge modellerinin tahminlerini birleştiren hibrit model
2 usages
def hybrid_model(X, theta_lasso, theta_ridge, lasso_weight=0.5):
    predictions_lasso = X.dot(theta_lasso)
    predictions_ridge = X.dot(theta_ridge)

    # Hibrit tahmin: Ağırlıklı ortalamayı kullanabilirsiniz
    hybrid_predictions = lasso_weight * predictions_lasso + (1 - lasso_weight) * predictions_ridge

    return hybrid_predictions
```