Stock price estimation of top 5 GPU companies

21781A33C4(R PAVITHRA)

21781A33D0(S SONIYA)

21781A33D1(SANJANA K)

21781A33E9(V NAGASHANTHI)

Team ID : 738312

Guided by

Prof.Ms Rasha

A Dissertation Submitted to SRI VENKATESWARA COLLEGE OF ENGINEERING AND TECHNOLOGY, An Autonomous Institution affiliated to 'JNTU Ananthapur' in Partial Fulfilment of the Requirements for the Bachelor of Technology (Computer Engineering) with Specialization in Artificial Intelligence and Machine Learning.

MAY 2024

SRI VENKATESHWARA COLLEGE OF ENGINEERING AND TECHNOLOGY

R.V.S Nagar Tirupati road, Andhra Pradesh-517127

Compliance certificate

This is to certify that the research work embodied in this dissertation entitled "Stock Price Prediction Using Machine Learning" was carried out by 21781A33C4 (R Pavithra), 21781A33D0 (S Soniya), 21781A33D1 (Sanjana K), 21781A33E9 (V Nagashanthi) at Sri Venkateswara College of Engineering and Technology for partial fulfilment of Bachelor of Technology (Computer Engineering) with Specialization in Artificial Intelligence and Machine Learning degree to be awarded by JNTU Anantapur. We have complied to the comments given by the Dissertation phase – We as well as Mid Semester Dissertation Reviewer to our satisfaction.

Date : 11/05/2024

Place : Chittoor

(Prof .Ms Rasha)

Head ,(CSE(AI&ML))(M Dr Lavanya)

Principal (Dr M Mohan Babu)

SRI VENKATESHWARA COLLEGE OF ENGINEERING AND TECHNOLOGY

R.V.S Nagar Tirupati road, Andhra Pradesh-517127

Declaration of originality

We hereby certify that We were the sole author of this dissertation and that neither any part of this dissertation nor the whole of the dissertation has been submitted for a degree to any other University or Institution.

We certify that, to the best of my knowledge, my dissertation does not infringe upon anyone's copyright nor violate any proprietary rights and that any ideas, techniques, quotations or any other material from the work of other people included in our dissertation, published or otherwise, are fully acknowledged in accordance with the standard referencing practices. Furthermore, to the extent that we have included copyrighted material that surpasses the bounds of fair dealing within the meaning of the Indian Copyright Act, We certify that we have obtained a written permission from the copyright owner(s) to include such material(s) in our dissertation and have included copies of such copyright clearances to our appendix.

We declare that this is a true copy of dissertation, including any final revisions, as approved by our dissertation review committee.

Date: 05/05/24

Place: Chittoor

21781A33C4 (R Pavithra)

21781A33D0 (S Soniya)

21781A33D1 (Sanjana K)

21781A33E9 (V Nagashanthi)

SRI VENKATESHWARA COLLEGE OF ENGINEERING AND TECHNOLOGY

R.V.S Nagar Tirupati Road, Andhra Pradesh-517127

Model Optimization and Tuning Phase Report

Model Optimization Phase:

Gradient Descent Algorithm

Gradient Descent is an optimization algorithm that iteratively adjusts the model's parameters to minimize the cost function. In the context of linear regression, the cost function represents the sum of squared errors.

For a given problem statement, the solution starts with Random Initialization. These initial parameters are then used to generate the predictions i.e. the output. Once we have the predicted values we can calculate the error or the cost i.e. how far the predicted values are from the actual target. In the next step, we update the parameters acordingly and again made the predictions with updated parameters. This process will go iteratively until we reach the optimum solution with minimal cost/error.

Steps involved in Gradient Descent

Step-1: Random Initialization

Initialize the coefficients **β0** and **β1** with arbitrary values.

Step-2: Calculating Gradients

Calculate the gradients of the cost function w.r.t **β0** and **β1**

$$\beta_0 = \beta_1 - \alpha \cdot \frac{\delta \varepsilon_{mse}}{\delta \beta_0}$$

$$\beta_1 = \beta_1 - \alpha \cdot \frac{\delta \varepsilon_{mse}}{\delta \beta_1}$$

"Gradient Descent is a first-order iterative optimization algorithm for finding a local minimum of a differential function"

Step-3: Updating Coefficients

Update the coefficients using the gradients and learning rate (α)

$$w_{new} = w - \alpha * \frac{\delta L}{\delta w}$$

$$b_{new} = b - \alpha * \frac{\delta L}{\delta b}$$

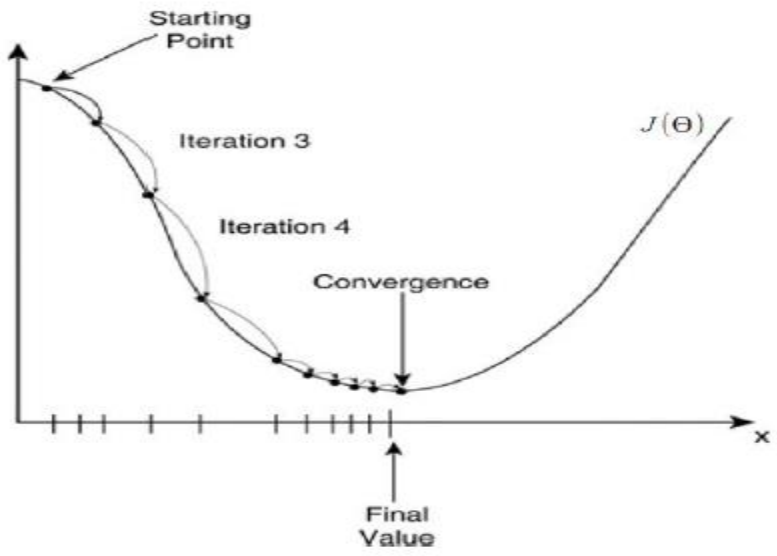
- **Learning Rate-** It is tuning parameter in an optimization algorithm that determine the step size at each iteration while moving toward minimum of loss function

Step-4: Iteration

Repeat steps 2 and 3 until **convergence (when the change in coefficient becomes negligible)** or for a **specified number of iterations.**

Step-5:Making Prediction

After convergence , use the final coefficients to make predictions



Ordinary Least Square (OLS)

OLS serves as a method for estimating Linear Regression model parameters. It strives to identify coefficients that minimize the sum of squared residuals where residuals represent the disparities between observed and predicted dependent variable values. The OLS algorithm assumes normally distributed errors with zero mean and constant variance along with no multicollinearity(high correlation) among independent variables which are assumption for Linear Regression modelling. In scenarios where these assumptions are not met, techniques like Generalized Least Squares or Weighted Least Squares should be considered.

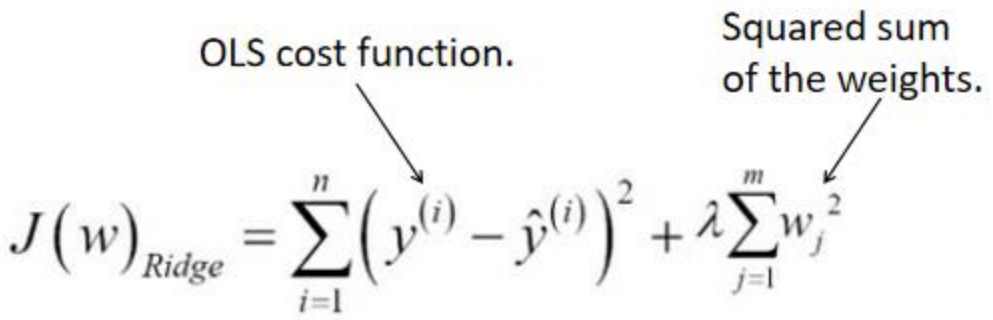
Steps involved in OLS

Step-1: Data Collection and Visualization

Collect the dataset containing the dependent and independent variables. Visualize the data to understand the relationship between the variables.

Step-2: Formulating the Objective Function

The objective is to minimize the sum of squared residuals



Where λ controls the strength of the shrinkage

Where n is the number of data points

Step-3: Calculating Coefficients

Calculate the coefficients using partial derivatives where the equation of the line is,

$$\beta_1 = \frac{\sum_{i=1}^{m}(x_i - \bar{x})(y_i - \bar{y})}{\sum_{i=1}^{m}(x_i - \bar{x})^2}$$

$$\beta_0 = \bar{y} - \beta_1 \bar{x}$$

Where **β0** is the y-intercept and **β1** is the slope**.** Calculating these coefficients involves some mathematics.

Step-4: Making Predictions

Once the coefficients are calculated, use the linear equation to make predictions for new data points.

Model Tuning:

Evaluate the model:

Evaluating the model requires that you first choose a holdout dataset used to evaluate the model. This should be data not used in the training process i.e. the X_test.

The speed of model evaluation is proportional to the amount of data you want to use for the evaluation, although it is much faster than training as the model is not changed.

From an API perspective, this involves calling a function with the holdout dataset and getting a loss and perhaps other metrics that can be reported.

```
In [ ]: model.evaluate(X_test, y_test)
```

Implementing hyperparameter tuning with Sklearn:

Well, we can automate the hyperparameter tunning using **GridSearCV**. GridSearchCV is a hyperparameter search procedure that is done over a defined grid of hyperparameters. Each one of the hyperparameter combinations is used for training a new model, while a cross-validation process is executed to measure the performance of the provisional models. Once the process is done, the hyperparameters and the model with the best performance are chosen.

Let's first take a look at the implementation of GridSearchCV with Sklearn, following the steps:

1. Define the general architecture of the model
2. Define the hyperparameters grid to be validated

3. Run the GridSearchCV process
4. Print the results of the best model

In [ ]:

```
# Import the GridSearchCV class
from sklearn.model_selection import GridSearchCV

# 1. Define the model's architecture
model = Sequential()
model.add(Dense(10, activation='relu', input_shape=(n_feature
s,)))
model.add(Dense(8, activation='relu'))
model.add(Dense(1))
optimizer = RMSprop(0.1)     # 0.1 is the learning rate
model.compile(loss='mean_squared_error',optimizer=optimizer)
# compile the model

# 2. Define the hyperparameters grid to be validated
batch_size = [10, 20, 40, 60, 80, 100]
epochs = [10, 50, 100]
param_grid = dict(batch_size=batch_size, epochs=epochs)
```

```
grid = GridSearchCV(estimator=model, param_grid=param_grid, scor
ing='neg_mean_squared_error', n_jobs=-1)

# 3. Run the GridSearchCV process
grid_result = grid.fit(X_train, y_train)

# 4. Print the results of the best model
print("Best: %f using %s" % (grid_result.best_score_, grid_resul
t.best_params_))
```

Hyperparameters directly control model structure, function, and performance. Hyperparameter tuning allows data scientists to tweak model performance for optimal results. This process is an essential part of machine learning, and choosing appropriate hyperparameter values is crucial for success.

For example, assume you're using the learning rate of the model as a hyperparameter. If the value is too high, the model may converge too quickly with suboptimal results. Whereas if the rate is too low, training takes too long and results may not converge. A good and balanced

choice of hyperparameters results in accurate models and excellent model performance.

Other Hyper Parameter Tuning Techniques:

Bayesian optimization

Bayesian optimization is a technique based on Bayes' theorem, which describes the probability of an event occurring related to current knowledge. When this is applied to hyperparameter optimization, the algorithm builds a probabilistic model from a set of hyperparameters that optimizes a specific metric. It uses regression analysis to iteratively choose the best set of hyperparameters.

Grid search

With grid search, you specify a list of hyperparameters and a performance metric, and the algorithm works through all possible combinations to determine the best fit. Grid search works well, but it's relatively tedious and computationally intensive, especially with large numbers of hyperparameters.

Random search

Although based on similar principles as grid search, random search selects groups of hyperparameters randomly on each iteration. It works well when a relatively small number of the hyperparameters primarily determine the model outcome