



University: Isfahan University - Faculty of Computer Engineering

Course: Fundamentals and Applications of Artificial Intelligence

Flight Price Prediction using Multiple Linear Regression

Course Instructor: Dr. Hossein Karshenas

Student Name: **Sheida Abedpour**

Student ID: 4003623025

2023 Nov

Introduction:

The aim of this project is to develop a predictive model using multiple linear regression to forecast flight prices based on various factors, providing valuable insights for both consumers and industry professionals. By leveraging historical flight data and incorporating features such as departure/arrival time, airlines, flight duration, distance and other relevant parameters, we can build a robust model that estimates flight prices with reasonable accuracy. The application of multiple linear regression allows us to capture the relationships and dependencies between these factors and the corresponding ticket prices.

Project Objective:

The primary objective of this project is to provide travelers with a reliable tool for estimating flight prices before making their travel plans. This can help them make informed decisions regarding their travel budgets and potentially save money by identifying periods of lower fares or choosing alternative routes. Additionally, the model can be beneficial for airline companies in setting competitive pricing strategies, optimizing revenue management, and understanding the impact of various factors on ticket prices.

Problem Overview:

To achieve these objectives, we will follow a systematic approach that involves collecting relevant flight data, preprocessing and cleaning the data, selecting the most significant features, training the multiple linear regression model, and evaluating its performance. We will also explore techniques for model validation and fine-tuning to ensure the accuracy and reliability of our predictions.

In conclusion, this project aims to leverage multiple linear regression to predict flight prices based on various factors. The outcomes of this project can have significant implications for both travelers and airline companies, providing valuable insights and aiding in decision-making processes. By accurately estimating flight prices, we can empower individuals to make informed travel choices and assist industry professionals in optimizing their pricing strategies.

Multiple Linear Regression:

Linear regression is a statistical modeling technique used to understand and predict the relationship between a dependent variable and one or more independent variables. It assumes a linear relationship between the predictor variables and the response variable.

Multiple linear regression extends this concept to multiple independent variables. The equation takes the form:

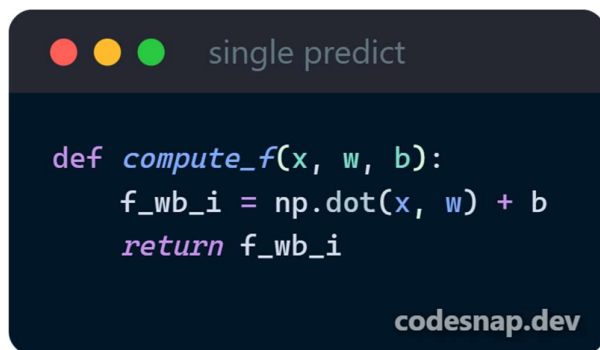
$$f(x_0, x_1, \dots, x_n) = w_0 \cdot x_0 + w_1 \cdot x_1 + \dots + w_n \cdot x_n + b$$

where f is the dependent variable, x_1, x_2, \dots, x_n are the independent variables, b is the f-intercept, and w_0, w_1, \dots, w_n are the coefficients associated with each independent variable.

or in vector notation:

$$f_{w,b}(\mathbf{x}) = \mathbf{w} \cdot \mathbf{x} + b$$

Single Prediction implementation:



```
def compute_f(x, w, b):  
    f_wb_i = np.dot(x, w) + b  
    return f_wb_i
```

codesnap.dev

Compute Cost With Multiple Variables:

In the context of machine learning and optimization, a cost function, also known as a loss function or objective function, is used to quantify the error or mismatch between the predicted values of a model and the actual values in the training dataset. The goal is to minimize this cost function to improve the model's performance.

The choice of a suitable cost function depends on the specific problem and the type of model being used. Here are a few commonly used cost functions:

1. **Mean Squared Error (MSE):** The MSE is a widely used cost function for regression problems. It calculates the average squared difference between the predicted values and the actual values. The formula for MSE is:

$$MSE = \frac{1}{2m} \sum_{i=0}^{m-1} (f_{w,b}(X^{(i)}) - Y^{(i)})^2$$

2. **Mean Absolute Error (MAE):** The MAE is another cost function used for regression problems. It measures the average absolute difference between the predicted values and the actual values. The formula for MAE is:

$$MAE = \frac{1}{2m} \sum_{i=0}^{m-1} |f_{w,b}(X^{(i)}) - Y^{(i)}|$$

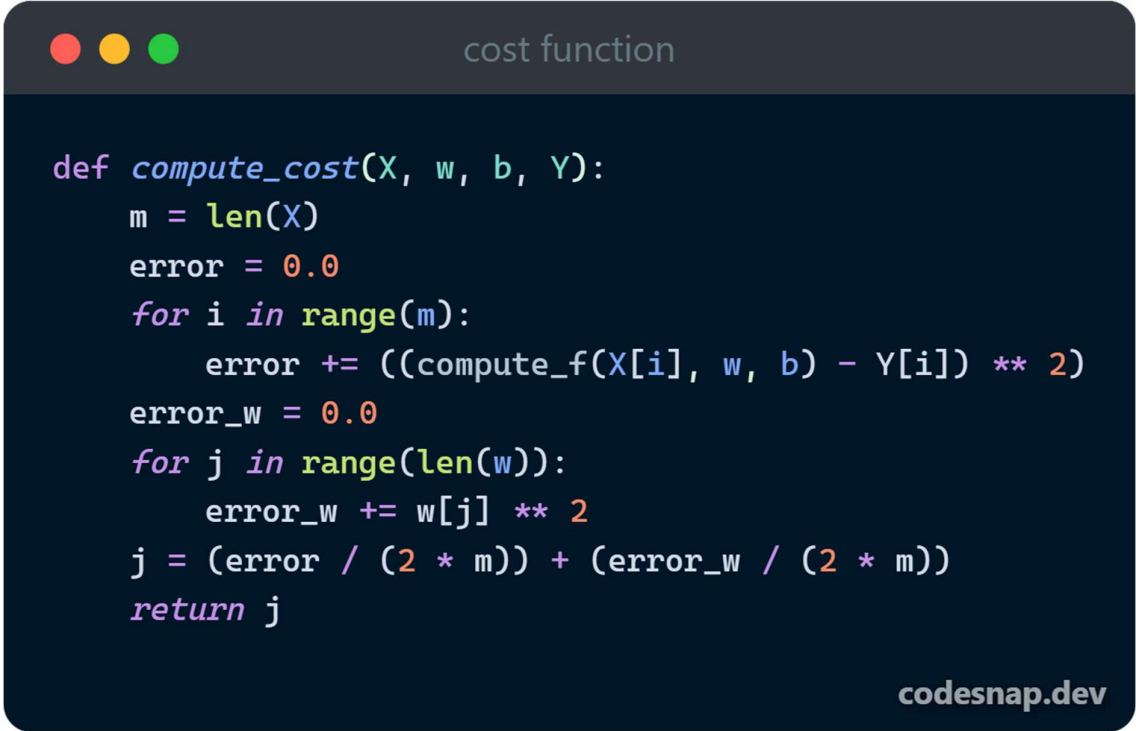
3. **Root Mean Squared Error (RMSE):** It measures the square root of the average squared difference between the predicted values and the actual values. The formula for RMSE is:

$$RMSE = \sqrt{\frac{1}{2m} \sum_{i=0}^{m-1} (f_{w,b}(X^{(i)}) - Y^{(i)})^2}$$

4. **R Squared(R2):** It measures the proportion of the variance in the dependent variable that can be explained by the independent variables in the model. R2 score ranges from 0 to 1, where a higher value indicates a better fit of the model to the data.

$$R^2 = 1 - \frac{\sum_{i=0}^{m-1} (f_{w,b}(X^{(i)}) - Y^{(i)})^2}{\sum_{i=0}^{m-1} (f_{w,b}(X^{(i)}) - \bar{Y})^2}$$

Cost Function implementation:



```
def compute_cost(X, w, b, Y):  
    m = len(X)  
    error = 0.0  
    for i in range(m):  
        error += ((compute_f(X[i], w, b) - Y[i]) ** 2)  
    error_w = 0.0  
    for j in range(len(w)):  
        error_w += w[j] ** 2  
    j = (error / (2 * m)) + (error_w / (2 * m))  
    return j
```

codesnap.dev

Gradient Descent:

Gradient descent is an optimization algorithm commonly used in machine learning and optimization problems to minimize the cost (or loss) function of a model. It is an iterative algorithm that adjusts the model's parameters or coefficients in the direction of steepest descent of the cost function, with the goal of finding the optimal values that minimize the cost.

The general idea behind gradient descent is to start with some initial values for the model's parameters and iteratively update them in the opposite direction of the gradient (slope) of the cost function with respect to those parameters. By repeatedly taking steps in the direction of steepest descent, the algorithm gradually converges towards the minimum of the cost function.

The steps involved in gradient descent are as follows:

1. Initialization: Start with initial values for the model's parameters.
2. Compute cost: Calculate the cost or loss function based on the current parameter values.
3. Compute gradients: Calculate the gradients of the cost function with respect to each parameter.
4. Update parameters: Update the parameter values by taking a step in the opposite direction of the gradients, multiplied by a learning rate (step size).
5. Repeat steps 2-4: Iterate the process by recomputing the cost, gradients, and updating the parameters until convergence or a predefined number of iterations is reached.

The learning rate is an important hyperparameter in gradient descent. It controls the step size taken in each iteration. A large learning rate can result in overshooting the minimum, leading to oscillations or divergence. A small learning rate can make the convergence slow. Finding an appropriate learning rate is crucial for the algorithm's effectiveness.

Gradient descent for multiple variables:

repeat until convergencs: {

$$w_j = w_j - \alpha \frac{\partial J(w,b)}{\partial w_j} \text{ for } j = 0..n-1$$

$$b = b - \alpha \frac{\partial J(w,b)}{\partial b}$$

}

where, n is the number of features, parameters w_j , b are updated simultaneously and where

$$\frac{\partial J(w,b)}{\partial w_j} = \frac{1}{m} \sum_{i=0}^{m-1} (f_{w,b}(X^i) - Y^i) X_j^i$$

$$\frac{\partial J(w,b)}{\partial b} = \frac{1}{m} \sum_{i=0}^{m-1} (f_{w,b}(X^i) - Y^i)$$

- m is the number of training examples in the data set
- $f_{w,b}(X(i))$ is the model's prediction, while $Y(i)$ is the target value

Gradient Descent implementation:

```
gradient descent

def compute_gradient_descent(X, Y, w_in, b_in, alpha, num_iterations):
    b = b_in
    w = copy.deepcopy(w_in)
    for i in range(num_iterations):
        dj_dw, dj_db = compute_gradient(X, w, b, Y)
        w = w - (alpha * dj_dw)
        b = b - (alpha * dj_db)
        cost = compute_cost(X, w, b, Y)
    return w, b

def compute_gradient(X, w, b, Y):
    m = len(X)
    n = len(X[0])
    dj_dw = np.zeros((n,))
    dj_db = 0
    for i in range(m):
        err = compute_f(X[i], w, b) - Y[i]
        for j in range(n):
            dj_dw[j] += (err * X[i,j])
        dj_db += err
    dj_dw /= m
    dj_db /= m
    return dj_dw, dj_db
```

codesnap.dev

Result:

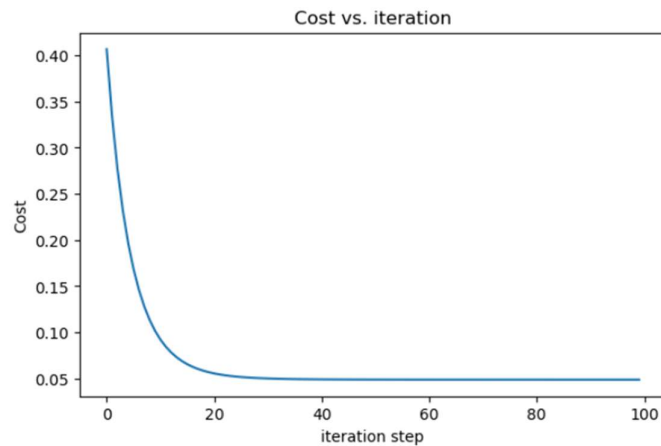
The choice of the number of iterations is a trade-off between achieving convergence and computational efficiency. Too few iterations may result in the algorithm not reaching the optimal solution, while too many iterations can lead to unnecessary computational burden. It is important to find a balance that ensures convergence while considering the available computational resources.

```
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size = 0.2, random_state = 42, shuffle = True)

initial_w = np.zeros(len(x[0]))
initial_b = 0
alpha = 0.1
num_itr = 100

start_time = time.time()
w, b = compute_gradient_descent(x_train, y_train, initial_w, initial_b, alpha, num_itr)
elapsed_time = time.time() - start_time
```

codesnap.dev



```
PRICE = 6.465940347251696e-05 +
-0.0026632754658709383 * departure_time +
0.104308948045109 * stops +
0.018115817746831533 * arrival_time +
0.026040501748192873 * duration +
-0.0780831756354624 * days_left +
-0.9332141739077996 * Economy
```

Traning Time: 1 m 45 s

Logs:

```
MSE: 0.10
RMSE: 0.31
MAE: 0.20
R2: 0.90
```

codesnap.dev

Appendix A:

Encoding is the process of converting data from one representation to another, often in the context of preparing data for machine learning or data analysis tasks. It involves transforming categorical or textual data into numerical representations that can be processed by machine learning algorithms.

To encode ordinal data in a dataframe using pandas in Python, you can utilize the `map()` function along with a dictionary that defines the encoding scheme.

```
time_mapping = {'Early_Morning': 1, 'Morning': 2,
                'Afternoon': 3, 'Evening': 4,
                'Night': 5, 'Late_Night': 6}

df['departure_time'] = df['departure_time'].map(time_mapping)
df['arrival_time'] = df['arrival_time'].map(time_mapping)

stop_mapping = {'zero': 0, 'one': 1, 'two_or_more': 2}
df['stops'] = df['stops'].map(stop_mapping)
```

codesnap.dev

One-Hot Encoding: One-hot encoding creates a binary vector for each category in a categorical variable. Each category is represented by a binary column, where a value of 1 indicates the presence of that category and 0 indicates its absence. This encoding is useful when there is no inherent order among the categories. One-hot encoding avoids the assumption of ordinality and prevents misinterpretation by algorithms. However, it can lead to high-dimensional data when applied to variables with many categories.

```
dummies_columns = ['class']
for dummy in dummies_columns:
    dummies = pd.get_dummies(df[dummy], drop_first=True).astype(int)
    df = pd.concat([df, dummies], axis='columns')
    df = df.drop([dummy], axis='columns')
```

codesnap.dev

Before encoding:

| | departure_time | stops | arrival_time | class | duration | days_left | price |
|--------|----------------|-------|---------------|----------|----------|-----------|-------|
| 0 | Evening | zero | Night | Economy | 2.17 | 1 | 5953 |
| 1 | Early_Morning | zero | Morning | Economy | 2.33 | 1 | 5953 |
| 2 | Early_Morning | zero | Early_Morning | Economy | 2.17 | 1 | 5956 |
| 3 | Morning | zero | Afternoon | Economy | 2.25 | 1 | 5955 |
| 4 | Morning | zero | Morning | Economy | 2.33 | 1 | 5955 |
| ... | ... | ... | ... | ... | ... | ... | ... |
| 270133 | Early_Morning | one | Night | Business | 17.25 | 49 | 68739 |
| 270134 | Morning | one | Evening | Business | 10.08 | 49 | 69265 |
| 270135 | Afternoon | one | Night | Business | 10.42 | 49 | 77105 |
| 270136 | Early_Morning | one | Evening | Business | 10.00 | 49 | 81585 |
| 270137 | Morning | one | Evening | Business | 10.08 | 49 | 81585 |

270138 rows × 7 columns

After encoding:

| | departure_time | stops | arrival_time | duration | days_left | price | Economy |
|--------|----------------|-------|--------------|----------|-----------|-------|---------|
| 0 | 4 | 0 | 5 | 2.17 | 1 | 5953 | 1 |
| 1 | 1 | 0 | 2 | 2.33 | 1 | 5953 | 1 |
| 2 | 1 | 0 | 1 | 2.17 | 1 | 5956 | 1 |
| 3 | 2 | 0 | 3 | 2.25 | 1 | 5955 | 1 |
| 4 | 2 | 0 | 2 | 2.33 | 1 | 5955 | 1 |
| ... | ... | ... | ... | ... | ... | ... | ... |
| 270133 | 1 | 1 | 5 | 17.25 | 49 | 68739 | 0 |
| 270134 | 2 | 1 | 4 | 10.08 | 49 | 69265 | 0 |
| 270135 | 3 | 1 | 5 | 10.42 | 49 | 77105 | 0 |
| 270136 | 1 | 1 | 4 | 10.00 | 49 | 81585 | 0 |
| 270137 | 2 | 1 | 4 | 10.08 | 49 | 81585 | 0 |

270138 rows × 7 columns

Appendix B:

Scaling data is an important preprocessing step in machine learning for several reasons:

1. **Feature Normalization:** Scaling data helps to normalize the features to a similar scale. Many machine learning algorithms use distance-based calculations or optimization techniques, such as gradient descent. If the features have different scales, variables with larger magnitudes can dominate the learning process and have a disproportionate impact on the results. By scaling the features, we ensure that each feature contributes proportionally to the learning process.
2. **Improved Model Performance:** Scaling the data can improve the performance of many machine learning algorithms. Algorithms such as K-nearest neighbors (KNN), support vector machines (SVM), and neural networks often perform better when the features are on a similar scale. Scaling can lead to faster convergence, better accuracy, and more stable models.
3. **Regularization:** Scaling data can also help with regularization techniques. Regularization methods, such as L1 or L2 regularization, introduce penalty terms based on the magnitude of the weights or coefficients. If the features have different scales, the regularization may be biased towards features with larger scales. Scaling the data ensures that the regularization is applied uniformly across all features.
4. **Gradient Descent Convergence:** In optimization algorithms like gradient descent, scaling the data can help the algorithm converge faster. When features have significantly different scales, the learning rate may need to be set differently for each feature to balance the convergence speed. By scaling the data, we can achieve a similar learning rate for all features and improve convergence speed.

StandardScaler is a preprocessing technique commonly used in machine learning to standardize or normalize the features of a dataset. It transforms the features so that they have zero mean and unit variance, making them more suitable for certain algorithms and improving the performance and interpretability of the models.

```

scaler = StandardScaler()
df = pd.DataFrame(scaler.fit_transform(df), columns=df.columns)

```

codesnap.dev

Before Scalling:

| | departure_time | stops | arrival_time | duration | days_left | price | Economy |
|--------|----------------|-------|--------------|----------|-----------|-------|---------|
| 0 | 4 | 0 | 5 | 2.17 | 1 | 5953 | 1 |
| 1 | 1 | 0 | 2 | 2.33 | 1 | 5953 | 1 |
| 2 | 1 | 0 | 1 | 2.17 | 1 | 5956 | 1 |
| 3 | 2 | 0 | 3 | 2.25 | 1 | 5955 | 1 |
| 4 | 2 | 0 | 2 | 2.33 | 1 | 5955 | 1 |
| ... | ... | ... | ... | ... | ... | ... | ... |
| 270133 | 1 | 1 | 5 | 17.25 | 49 | 68739 | 0 |
| 270134 | 2 | 1 | 4 | 10.08 | 49 | 69265 | 0 |
| 270135 | 3 | 1 | 5 | 10.42 | 49 | 77105 | 0 |
| 270136 | 1 | 1 | 4 | 10.00 | 49 | 81585 | 0 |
| 270137 | 2 | 1 | 4 | 10.08 | 49 | 81585 | 0 |

270138 rows × 7 columns

After Scalling:

| | departure_time | stops | arrival_time | duration | days_left | price | Economy |
|--------|----------------|-----------|--------------|-----------|-----------|-----------|-----------|
| 0 | 0.798879 | -2.320915 | 0.962015 | -1.397233 | -1.843752 | -0.658203 | 0.672585 |
| 1 | -1.318820 | -2.320915 | -1.257371 | -1.374997 | -1.843752 | -0.658203 | 0.672585 |
| 2 | -1.318820 | -2.320915 | -1.997166 | -1.397233 | -1.843752 | -0.658071 | 0.672585 |
| 3 | -0.612920 | -2.320915 | -0.517576 | -1.386115 | -1.843752 | -0.658115 | 0.672585 |
| 4 | -0.612920 | -2.320915 | -1.257371 | -1.374997 | -1.843752 | -0.658115 | 0.672585 |
| ... | ... | ... | ... | ... | ... | ... | ... |
| 270133 | -1.318820 | 0.191205 | 0.962015 | 0.698512 | 1.696330 | 2.108062 | -1.486801 |
| 270134 | -0.612920 | 0.191205 | 0.222219 | -0.297940 | 1.696330 | 2.131237 | -1.486801 |
| 270135 | 0.092980 | 0.191205 | 0.962015 | -0.250688 | 1.696330 | 2.476657 | -1.486801 |
| 270136 | -1.318820 | 0.191205 | 0.222219 | -0.309058 | 1.696330 | 2.674039 | -1.486801 |
| 270137 | -0.612920 | 0.191205 | 0.222219 | -0.297940 | 1.696330 | 2.674039 | -1.486801 |

270138 rows × 7 columns

References:

The resources I used in this project include:

- Chat GPT
- [youtube.com/@codebasics](https://www.youtube.com/@codebasics)
- coursera.org/learn/machine-learning

Python Libraries used in project:

- Numpy
- Pandas
- sklearn