Machine Learning TP1

Félix HENAFF & Alfred DEDUYER, FIPA25

3 Linear regression with one variable

3. 1 Plotting the Data

How many features are involved in this problem?

There is 1 feature involved in the problem: the population in a city.

How much data is involved in this problem?

There are 97 data because there are 97 cities.

What is the problem from a machine learning point-of-view?

– Supervised/Unsupervised? Supervised because we give a y value.

– Regression/Classification? Regression because y is continuous

– Linear/non-linear problem?

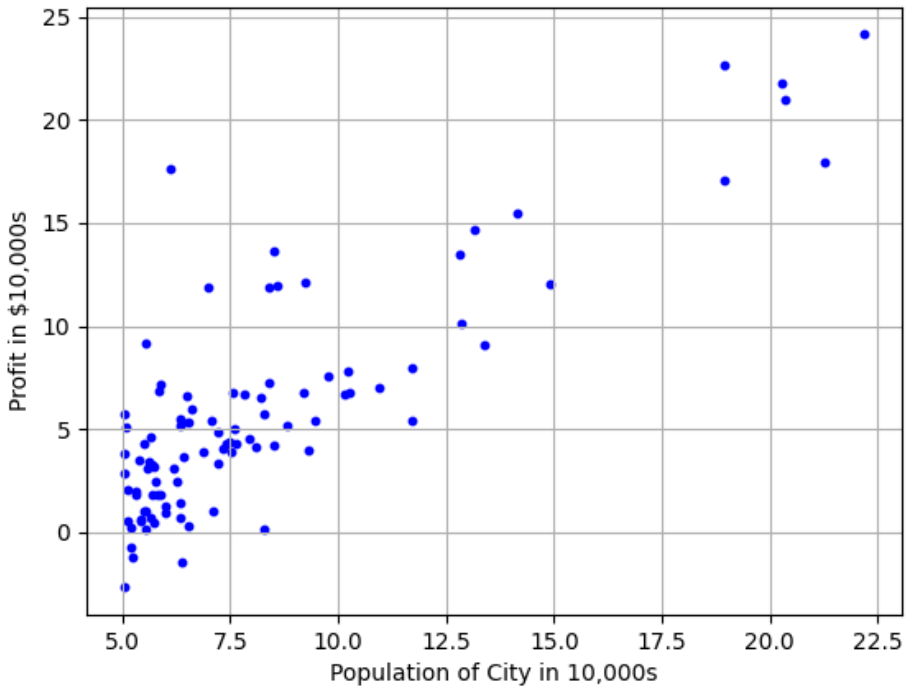

Figure 1 Scatter plot of training data

3.2 Gradient Descent

3.2.3 Computing the cost J(θ)

```
def computeCost(X, y, theta):
    """
       computes the cost of using theta as the parameter for linear
       regression to fit the data points in X and y
    """
    # m = number of training examples
    m = y.size

    # Hypothesis (predicted values)
    h = X.dot(theta)

    # Compute the squared errors
    squared_errors = (h - y) ** 2

    # Calculate the cost function
    J = (1 / (2 * m)) * np.sum(squared_errors)

    return J
```

3.2.4 Gradient descent

```
import numpy as np
from computeCost import computeCost

def gradientDescent(X, y, theta, alpha, num_iters):
    """
    Performs gradient descent to learn theta.
    theta, cost_history, theta_history = gradientDescent(X, y, theta,
alpha, num_iters)
    Updates theta by taking num_iters gradient steps with learning rate
alpha.
    """
    # Initialize some useful values
    m = y.size  # number of training examples
    n = theta.size  # number of parameters
    cost_history = np.zeros(num_iters)  # cost over iterations
    theta_history = np.zeros((n, num_iters))  # theta over iterations

    for i in range(num_iters):
        # Compute the prediction error
        error = X.dot(theta) - y

        # Perform the gradient step for each theta parameter
        gradient = (1 / m) * X.T.dot(error)

        # Update theta
        theta = theta - alpha * gradient

        # Save the cost J in every iteration
        cost_history[i] = computeCost(X, y, theta)

        # Save the values of theta in every iteration
        theta_history[:, i] = theta.flatten()

    return theta, cost_history, theta_history
```

```
--------------------------
Theta found by gradient descent:
-3.6302914394043593 1.1663623503355818
Expected theta values (approx)
-3.6303  1.1664
--------------------------
```

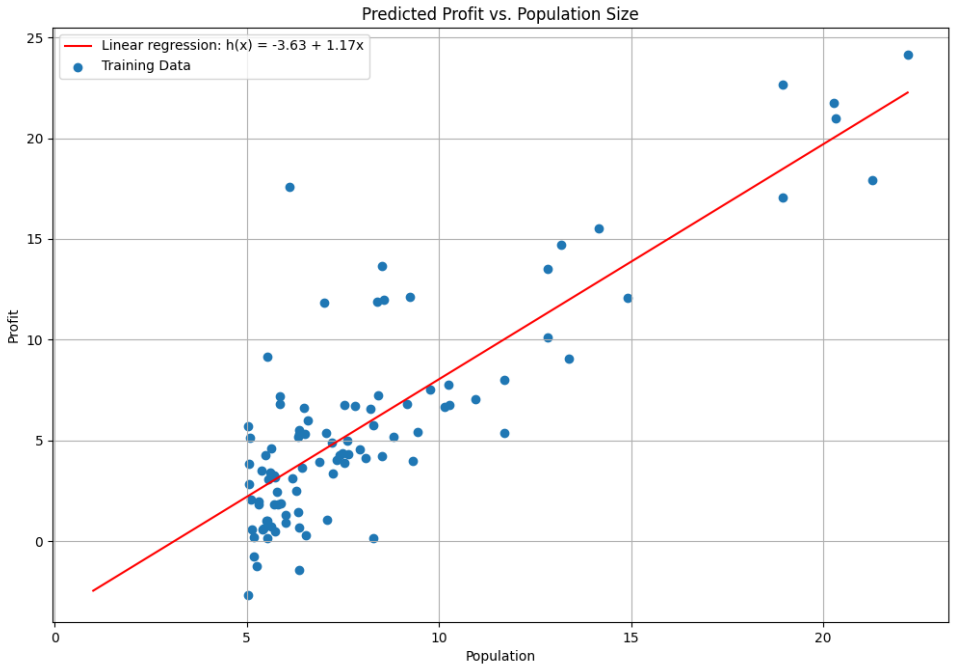

Figure 2 Training data with linear regression fit

3.3 Visualizing J(θ)

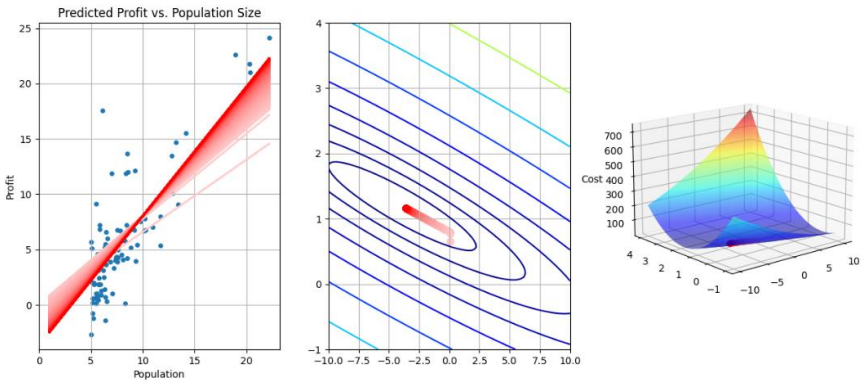

Figure 3 Cost function J(θ)

4 Linear regression with multiple features

4.1 Feature Normalization

```
def featureNormalize(X):
    """
    Returns a normalized version of X where
    the mean value of each feature is 0 and the standard deviation
    is 1. This is often a good preprocessing step to do when
    working with learning algorithms.
    """
    # Compute the mean and standard deviation of each feature
    mu = np.mean(X, axis=0)
    sigma = np.std(X, axis=0)

    # Normalize each feature in X
    X_norm = (X - mu) / sigma

    return X_norm, mu, sigma
```

4.2 Learning with gradient decent

4.2.1 Compute cost and gradients

```
def computeCostMulti(X, y, theta):
    """
    Computes the cost of using theta as the parameter for linear
    regression to fit the data points in X and y.
    """
    m = y.size  # Number of training examples

    # Hypothesis (predicted values)
    h = X.dot(theta)

    # Compute the squared errors
    squared_errors = (h - y) ** 2

    # Calculate the cost function
    J = (1 / (2 * m)) * np.sum(squared_errors)

    return J
def gradientDescentMulti(X, y, theta, alpha, num_iters):
    """
    Performs gradient descent to learn theta.
    theta, cost_history, theta_history = gradientDescentMulti(X, y, theta,
alpha, num_iters)
    Updates theta by taking num_iters gradient steps with learning rate
alpha.
    """
    m = y.size  # Number of training examples
    n = theta.size  # Number of parameters
    cost_history = np.zeros(num_iters)
    theta_history = np.zeros((n, num_iters))

    for i in range(num_iters):
        # Compute the error (difference between prediction and actual
values)
        error = X.dot(theta) - y

        # Update each parameter theta_j
```

```
        gradient = (1 / m) * X.T.dot(error)
        theta = theta - alpha * gradient

        # Save the cost J in every iteration
        cost_history[i] = computeCostMulti(X, y, theta)
        theta_history[:, i] = theta.flatten()

    return theta, cost_history, theta_history
```

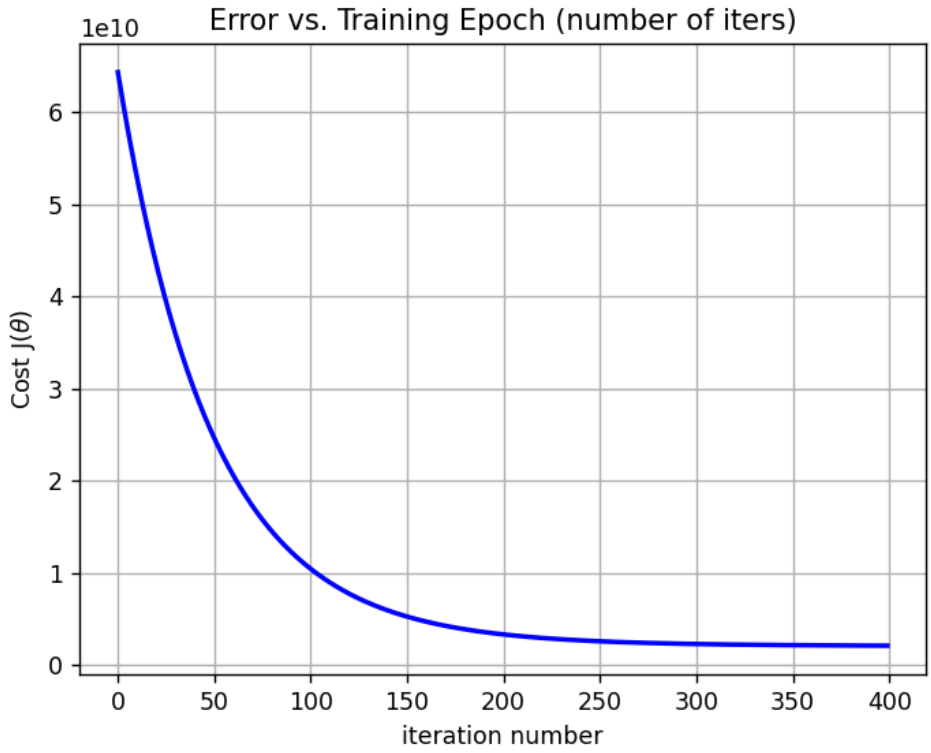

Figure 4 Convergence of gradient descent with an appropriate learning rate

4.2.3 Predict the price for a new house

we used a learning rate (αα) of 0.05 and 600 iterations to adjust the model's parameters during gradient descent.

Predicted price of a 1650 sq-ft, 3 br house

(using gradient descent):

[[293081.48469249]]

In this case, after 600 iterations and with an alpha of 0.05, the model predicted a price of approximately $293,081 for a 1650 sq-ft, 3-bedroom house.

4.3 Normal equations

```
def normalEqn(X,y):
    """ Computes the closed-form solution to linear regression
       normalEqn(X,y) computes the closed-form solution to linear
       regression using the normal equations.
    """
    theta = np.linalg.inv(X.T @ X) @ X.T @ y
    return theta
```

Predicted price of a 1650 sq-ft, 3 br house

(using normal equations):

[[293081.4643349]]

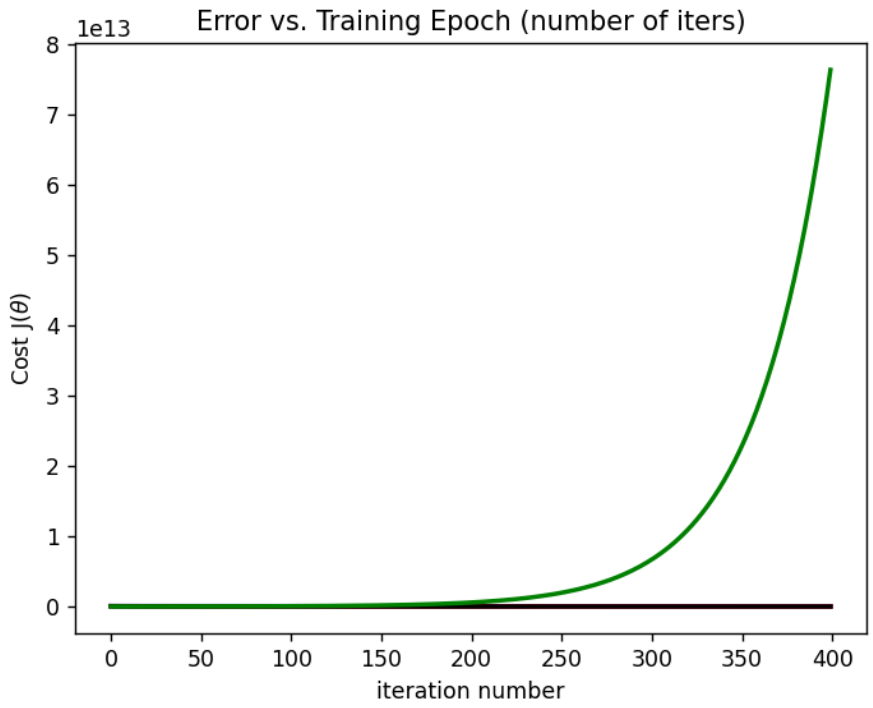

Figure 5 : Error vs training

4.4 Some questions, you have to answer

Define terms:

Supervised/Unsupervised:

- Supervised: The algorithm learns from labeled data (with known responses).
- Unsupervised: The algorithm learns from unlabeled data to identify hidden structures.

Regression/Classification:

- Regression: Prediction of continuous values (e.g., house price).
- Classification: Prediction of categories or classes (e.g., spam/non-spam).

Descriptors/Features: Input variables that explain the data (e.g., surface area, number of bedrooms for a house).

Targets: Output values to be predicted (e.g., house price, object class).

Linear regression model: A model that predicts a linear relationship between the features and a continuous target, in the form of an equation.

$$h_\theta(x) = \theta_0 + \theta_1 x_1 + \cdots + \theta_n x_n$$

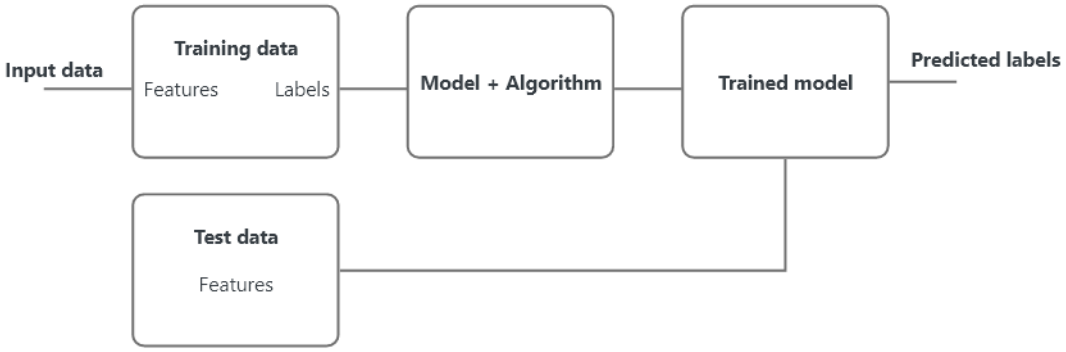

Figure 6 Schematic learning and prediction

How does learning work? By what means? What is the purpose of the cost function? How

is the problem solved? Do you know of other ways of solving it?

Learning works by adjusting model parameters to minimize the difference between predictions and actual values, using a process called optimization.

- How: Through algorithms like gradient descent, which iteratively updates parameters based on the error.
- Cost function: Measures the error between predictions and actual values; its minimization guides the learning process.
- How it's solved: By adjusting model parameters to minimize the cost function, usually with gradient descent.

Why do we sometimes need to standardize features?

Normalization of features is important to ensure that all features have a similar scale. This prevents features with larger ranges from dominating the learning process and helps optimization algorithms like gradient descent converge faster and more efficiently.