**Data Mining Algorithms (Analysis Services - Data Mining)**

<https://docs.microsoft.com/en-us/analysis-services/data-mining/data-mining-algorithms-analysis-services-data-mining?view=asallproducts-allversions>

#### Read more on Medium. [Create a free account.](https://medium.com/m/signin?operation=register&redirect=https%3A%2F%2Fmedium.com%2Fanalytics-vidhya%2Fpredict-population-growth-using-linear-regression-machine-learning-d555b1ff8f38&source=-----d555b1ff8f38---------------------metered_view_2-)

[Sign in](https://medium.com/m/signin?operation=login&redirect=https%3A%2F%2Fmedium.com%2Fanalytics-vidhya%2Fpredict-population-growth-using-linear-regression-machine-learning-d555b1ff8f38&source=post_page-----d555b1ff8f38----------------------)

Get started

[](https://medium.com/analytics-vidhya?source=post_page-----d555b1ff8f38----------------------)

* [ABOUT US](https://medium.com/analytics-vidhya/about?source=post_page-----d555b1ff8f38----------------------)
* [MACHINE LEARNING](https://medium.com/analytics-vidhya/tagged/machine-learning?source=post_page-----d555b1ff8f38----------------------)
* [DEEP LEARNING](https://medium.com/analytics-vidhya/tagged/deep-learning?source=post_page-----d555b1ff8f38----------------------)
* [HACKATHONS](https://medium.com/analytics-vidhya/tagged/hackathons?source=post_page-----d555b1ff8f38----------------------)
* [CONTRIBUTE](https://medium.com/analytics-vidhya/contribute/home?source=post_page-----d555b1ff8f38----------------------)
* [FREE COURSES](https://courses.analyticsvidhya.com/collections?category=free&source=post_page-----d555b1ff8f38----------------------)

# Predict Population Growth Using Linear Regression — Machine Learning Easy and Fun

# <https://medium.com/analytics-vidhya/predict-population-growth-using-linear-regression-machine-learning-d555b1ff8f38>

## In Machine Learning one of the simplest prediction models is Linear Regression. It allows you to make predictions on the data and it’s really intuitive and easy to understand



[Gavril Ognjanovski](https://medium.com/%40ognjanovski.gavril?source=post_page-----d555b1ff8f38----------------------)

Follow

[Dec 4, 2018](https://medium.com/analytics-vidhya/predict-population-growth-using-linear-regression-machine-learning-d555b1ff8f38?source=post_page-----d555b1ff8f38----------------------) · 9 min read

Photo by [Jacek Dylag](https://unsplash.com/%40dylu?utm_source=medium&utm_medium=referral" \t "_blank) on [Unsplash](https://unsplash.com/?utm_source=medium&utm_medium=referral" \t "_blank)

In this project, we will make a prediction on the Swedish population growth. There is also a lot of open data available online in Sweden, so that’s an opportunity for making interesting and real projects. I will try to split this project in multiple steps.

# Table of Contents

1. Understand Linear Regression
2. Collect and Prepare the Data
3. Visualize Data (Understand the Data)
4. Normalize the Features
5. Create Linear Regression Model
6. Predict Using Linear Regression Model

*The data and the code are available on my Github account*[*https://github.com/Gago993/PopulationPredictionLinearRegression*](https://github.com/Gago993/PopulationPredictionLinearRegression)

For this solution I used the [GNU Octave](https://www.gnu.org/software/octave/) (compatible with Matlab)and Visual Studio Code.

**So, let’s start…**

## Understand Linear Regression

*Linear — arranged in or extending along a straight or nearly straight line.*

By definition, linear simply means a straight line. So, if we have multiple (x,y) coordinates like on the graph below, then the blue straight line would represent the linear regression equation. Actually it is the straight line that best fits (make the best prediction) for the given data.



Image 1: Linear Equation

This blue straight line can be mathematically expressed by the following equation



Linear regression equation

Where h is called hypothesis and θ is called theta. Don’t be confused by the letters, because this equation is same as



Linear equation

## Collect and Prepare the Data

The data that we are using is available on the [Statistics Sweden Agency](https://www.scb.se/en/finding-statistics/statistics-by-subject-area/population/population-composition/population-statistics/pong/tables-and-graphs/yearly-statistics--the-whole-country/population-and-population-changes/).

What would you get is Excel file with multiple columns (Year, Population, Live Births, …)

We need only the first two columns (Year and Population) so you can export them on your own from Excel as .csv file or you can download the prepared data from [here](https://gist.github.com/Gago993/6e6122e6e260437dcb736873e76ff12e). In the prepared data we use only statistics starting from 2007–2017.

At the end the data should look like this

Year;Population
2007;9182927
2008;9256347
...
2017;10120242

Now the data is ready and saved in **swe\_pop\_2007\_2017.csv**, so we can continue with the next step where we will load this data and visualize it so we could better understand it.

## ****Visualize Data****

In order to visualize the data we need to load it. The main function for loading the csv data is **getCsvData.m** file where we send the filename as only function argument and it returns cell array where the first column is the Year and the second column is the number of Population for that particular year. Below it is the code for loading the csv data.

After the data is loaded, we need to visualize it.

In this solution there are two main files **main\_gradient\_descent.m** and **main\_normal\_equation.m** where all the visualization code is the same. The only difference is that the first file is using Gradient Descent and the second one **Normal Equation** to compute the value of theta. I will explain both of them in the next section.

It starts by loading the data by using **getCsvData**function. Next we need to convert returned cell array to matrix for easier computation, also converting the matrix values to double precision. Then we use the build in **plot**function to visualize the data.

After plotting the data we get the following graph



Image 2: Visualize Population data of Sweden

What we can notice there is that the growth of the population is in nearly straight line. After we get the theta values in the “Create Linear Regression Model” section we will be able to plot the equation (line) and check if it fits this data. So, let’s keep on going.

## Normalize the Features

This step is only required if you are going to use the Gradient Descent algorithm. Before we are able to compute the theta values we need to normalize the features so it will optimize the calculations.

*Normalization— standardize the range (values) of independent variables or features of data*

That is done by subtracting the [mean](https://en.wikipedia.org/wiki/Mean) (average) and divide by [standard deviation](https://en.wikipedia.org/wiki/Standard_deviation) for all the features.



Feature Normalization Formula

So there is a simple function in **featureNormalize.m** where the above formula is implemented.

After performing the feature normalization on the Year feature in our project they will be scaled and would look like the values below. This will make the computation more efficient and faster.

I put a comments after every code line so it’s easier to understand. If there is any questions feel free to reach me.

## Create Linear Regression Model

Now we need to obtain the theta values for the equation that best fits the data we visualized in the previous step.

I will explain two options for computing theta values.

* Computing Normal Equation (one-step algorithm)
* Using Gradient Descent algorithm

**Normal Equation (First Option)**

This is analytical solution for Linear Regression computation. It’s formula it’s simple and includes multiplying matrices. When using this formula any feature normalization is not required.



Normal Equation Formula

The code below is part of the **main\_normal\_equation.m** file where first we add column of 1’s to the data in order to include and compute θ₀ in the equation. Then we compute the theta vector with the formula.

...
m = length(y);% Add ones column
X = [ones(m, 1) X];% Gradient Descent with Normal Equation
theta = (pinv(X'\*X))\*X'\*y
...

After computing the theta we get the following values

theta = [-173280098.50770, 90894.36059]

Which actually represent the linear equation

h = 90894.36059 \* x + (-173280098.50770)

In order to verify if the given equation fits the data well we can plot it on the same graph with the data. We can do that by simply calling

...
% Plot linear regression line
plot(X(:,2), X\*theta, '-')
...

What would we get is this graph. Where we can see that if fits well with the data.



Image 3: Linear Regression Normal Equation

**Gradient Descent (Second Option)**

Using Gradient Descent algorithm requires feature normalization in some cases. In this example I will implement normalization since the values for the population are too big so the computation will exceed the maximum value for the number that can be represented and therefor will return “Inf”.

...
% Normalize features
[X\_norm mu sigma] = featureNormalize(X);
...

So this algorithm is trying to optimize the cost function shown below or in other words to find the θ₀ and θ₁ values that return the lowest value for the J function.

The idea is pretty simply, find the minimum error from difference between predicted value and actual value for each training example.



Cost Function

Gradient Descent algorithm intuition can be compared with going down the hill and trying to find the lowest place possible. This is visually represented on the following graph.



Image 4: Gradient Descent steps 3D. Source: [Coursera](https://www.coursera.org/learn/machine-learning%22%20%5Ct%20%22_blank) — Andrew Ng

We can see that at the beginning we start from some random place on the graph (hill) and the algorithm is responsible for deciding in which step it should move.

The general**Gradient Descent formula**is



It finds the partial derivative in respect to θ. Using calculus, we know that the slope of a function is the derivative of the function with respect to a value and that’s how the algorithm would know if it should increase or decrease the value of θ₀ and θ₁ in each step.

*The slope shows in which side we should take the next step towards the bottom of the hill!*

To find the minimum cost we got to change the θ values until convergence (reach the limit value).



In case you are interested the general formula for Gradient Descent will be



The intuition behind Gradient Descent is on the image below where we can see the Gradient slope is defining the direction. The little black arrows represent the α. Where α is called the learning rate and it shows how big the change (step down the hill) should be. The variable w represent the θ value.

*Note: This is simplified case respect to one θ variable. In our case this graph would be 3D (see*Image 4*) as there is θ₀ and θ₁ variables.*



Image 4: Gradient Descent steps towards minimum cost value. Credits: [Machine Learning FAQ](https://sebastianraschka.com/faq/docs/closed-form-vs-gd.html)

So now let’s continue with the code implementation of the Gradient Descent algorithm. The code is available in **gradientDescent.m** file. Where matrices are used to compute the theta values.

Next we use this function in **main\_gradient\_descent.m** file to get the theta values.

...
% Choose some alpha value
alpha = 0.1;
% Set number of iterations
num\_iters = 400;
% Initialize theta
theta = zeros(2, 1);
...% Init Theta and Run Gradient Descent
theta = gradientDescent(X\_norm, y, theta, alpha, num\_iters)
...

What we get for theta is

theta = [9599355.00000, 301462.59019]

And this actually represent the linear equation

h = 301462.59019 \* x + (9599355)

And we will plot this equation on the graph along with our data.

% Plot linear regression line
plot(X, X\_norm\*theta, '-')

Where by looking at the graph we can see that the blue line fits well our data.



Image 5: Linear Equation Gradient Descent

## Predict Using Linear Regression Model

Now that we got the theta values for the equation we should do population prediction for some of the next years. So let’s calculate the expected number of people living in Sweden in 2020.

Using the theta values that we got using the **Normal Equation** we can predict the expected population by multiplying with the theta values as shown on the code below.

% Predict population for 2020
pred\_year = 2020;
pred\_year\_val = [1 2020];% Calculate predicted value
pred\_value = pred\_year\_val \* theta;

What we get is estimation of 10 326 510 people in 2020.

In case of using the**Gradient Descent**, since we have done feature normalization we must not forget to normalize the feature before we do the prediction by using the **mu**and **sigma**variables.

% Predict population for 2020
pred\_year = 2020;% Dont forget to normalize the feature before prediction
pred\_year\_val = (pred\_year .- mu)./sigma;% Add first column
pred\_year\_norm = [1 pred\_year\_val];% Calculate predicted value
pred\_value = pred\_year\_norm \* theta;

So here the predicted population in 2020 is also 10 326 510 people.

At the end we just plot this value on the graph marked with blue cross.



Image 6: Final Graph

# Conclusion

This model is very simple to build and use so it can be used on a lot of other project ides like predicting stock price, average km or miles per liter/galon of petrol, football player salary and scores per game, etc … I’m looking forward to hear your ideas and projects. For any questions or suggestions please post a comment or contact me.

Hope you enjoyed it!