1 Week 3: Linear regression with multiple variables

So far, you've been willing to humour me with trying to predict house prices with just one variable. But there's plenty of information in the Ames dataset that might help us have better predictions.

Imagine we find that there are 20 features that might help us make better predictions. We could go and edit

```
case class SimplePoint(x: Double, y: Double)
```

and start adding our features

```
case class House(
MSZoning: Int,
LotFrontage: Double,
LotArea: Double,
LandContour: Int,
LotConfiguration: Int, ....
)
```

but this will create a cascade where we need to then have a way to calculate mean squared error for a house, and an implementation of Gradient Descent that works for a house, and so on. We need a more general way to represent our data.

1.1 Matrices

If we can generalise our gradient descent model to work in terms of matrices, then our model will be flexible and we can easily change how many features we wish to consider.

First, we need to review matrix operations. Khan academy has great resources for this with interactive exercises to try out, so I'll just link to those resources below (if you look at this with a pdf viewer [not github's web viewer], you should be able to click on the link, otherwise just copy/paste.

1.1.1 Matrix dimensions and transpose of a matrix

Here is an introduction to matrices, including matrix dimensions:

https://www.khanacademy.org/math/precalculus/precalc-matrices/intro-to-matrices/a/intro-to-matrices

You can see a video about the transpose of a matrix here:
https://www.khanacademy.org/math/linear-algebra/matrix-transformations/matrix-transpose/v/linear-algebra-transpose-of-a-matrix?modal=1

1.1.2 Matrix addition

Here is how you can add and subtract matrices:

https://www.khanacademy.org/math/precalculus/precalc-matrices/adding-and-subtracting-matrices/a/adding-and-subtracting-matrices

1.1.3 Scalar multiplication

Here is how you can multiply matrices by a scalar:

https://www.khanacademy.org/math/precalculus/precalc-matrices/multiplying-matrices-by-scalars/a/multiplying-matrices-by-scalars

1.1.4 Matrix multiplication and the dot product

You can read about the dot product and multiplying matrices together here:

https://www.khanacademy.org/math/precalculus/precalc-matrices/multiplying-matrices-by-matrices/a/multiplying-matrices

1.2 Linear regression with multiple features

Now that we have n features $x_1, x_2, ..., x_n$, we must update our hypothesis:

Hypothesis: $h_\theta(x) = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + ... + \theta_n x_n$ (we think that a line will fit our data)

And so our model will have n parameters

Parameters: θ_0, θ_1, ... , θ_n

Cost function: $J(\theta_0, \theta_1, ..., \theta_n) = \frac{1}{2m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)})^2$

From here, we will refer to $J(\theta_0, \theta_1, ..., \theta_n)$ as $J(\theta)$

Goal: minimise $J(\theta)$. Once we have θ_0, θ_1, ..., θ_n we can plug them into our hypothesis and make predictions!

We can also now express our model in terms of matrices. We know that in our hypothesis we have parameter θ_0, so for the convenience of tidier notation we'll define $x_0^{(i)}$ for all i. Let X be the nxm matrix where there is a column for each feature. Let y be a $mx1$ matrix.

So let's go back to our table of sample data:

Table 1: Sample from a dataset of heights and weights

x_1	x_2	x_3	x_4	y
Id	LotArea	LotFrontage	NumberOfBedrooms	SellingPrice
1143	9965.0	440	2	424870.0
1105	2016.0	320	3	106000.0
923	10237.0	800	3	169990.0
499	7800.0	200	5	130000.0
1124	9405.0	180	4	118000.0

We would have $\begin{bmatrix} 1 & 1 & 1 & 1 & 1 \\ 1143 & 1105 & 923 & 499 & 1124 \\ 9965.0 & 2016.0 & 10237.0 & 7800 & 9405.0 \\ 440 & 320 & 800 & 200 & 180 \\ 2 & 3 & 3 & 5 & 4 \end{bmatrix}$

And our y, the value we will be trying to predict as $y = \begin{bmatrix} 424870.0 \\ 106000.0 \\ 169990.0 \\ 130000.0 \\ 118000.0 \end{bmatrix}$

Let's also define our parameters as a matrix $\theta = \begin{bmatrix} \theta_0 \\ \theta_1 \\ \theta_2 \\ \theta_3 \\ \theta_4 \end{bmatrix}$

Recall our hypothesis: $h_\theta(x) = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + ... + \theta_n x_n$ (we think that a line will fit our data)

If we let $x^{(i)}$ represent the ith training example, $x^{(i)} = \begin{bmatrix} x_0^{(i)} \\ x_1^{(i)} \\ x_2^{(i)} \\ x_3^{(i)} \\ x_4^{(i)} \end{bmatrix}$

Then we can express our model as the following:

Hypothesis: $h_\theta(x^{(i)}) = \theta^T x^{(i)}$

Parameters: θ (our vector of thetas)

Cost function:

$$J(\theta_0, \theta_1, ..., \theta_n) = \frac{1}{2m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)})^2$$

We also need to update our algorithm for gradient descent:

```
Data: An $m \times n$ matrix with $n$ features and $m$ training examples,
      $y$, a $m \times 1$ matrix
Result: $\theta$ a $n \times 1$ matrix of parameters for the line that fits the
        input data
set some initial values for $\theta_0, \theta_1, ..., \theta_n$
while we have not reached convergence do
    for each $j$ in range 0 up to $n$ do
        $\theta_j := \theta_j - \alpha \frac{1}{m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)}) \times x^{(i)}$
    end
end
return $\theta_0, \theta_1, ..., \theta_n$
```

Algorithm 1: gradient descent