Machine learning notes: week 1

Lars Yencken

Introduction

Welcome

- Machine learning is already pervasive in our lives
  - E.g. web search, photo tagging
- This course is especially meant to be practical
- Old AI: teach computers to solve a problem
- New AI: teach computers to teach themselves
- Applications
  - Data minining: we intensively measure so much now, making sense of it is a machine learning problem
  - Things too difficult to program by hand: flying helicopters, handwriting recognition, language, vision
  - Self-customisation: recommendation a la Netflix, Amazon

What is machine learning?

- Tom Mitchell's definition:
  - Learn from experience E (e.g. user spam ratings)
  - On task T (e.g. spam detection)
  - So as to improve with respect to some performance measure P (e.g. correctly labelling spam/not spam) with increasing experience E
- Three main families:
  - Supervised learning
  - Unsupervised learning
  - Reinforcement learning

Supervised learning

- The most common type of learning problem
- *Supervised*: the dataset contains the correct answers for each case
- Examples:
  - Predict house price given size in square feet (trained with a dataset with both measurements)
  - Predict whether a tumour is malignant or benign given the tumour size (trained with a dataset with both measurements)
- *Classification problem*: when each prediction is one of a fixed number of choices (e.g. cancer vs no cancer)
- *Regression problem*: when a prediction is numeric (e.g. house price)
- *Features*: the input variables used to make a prediction (e.g. land size, number of bedrooms, post code)

Unsupervised learning

- Unsupervised learning: no correct answers in the dataset
- Often looking for groups in the dataset which share a lot in common (*clusters*)
- Examples:
  - Detecting trending news topics just based on article text
  - Detecting market segments based on discovered customer groups
  - Detecting patterns in social networks
  - Cocktail party problem: given multiple microphones in different places, separate the audio from multiple simultaneous speakers into one stream per speaker
- We're using Octave in this course because it lets you deal with vectors and matrices at a very high level (other reasonable languages: Matlab, R, Python, Julia, IDL)

Linear regression with one variable

Model representation

- Problem: predict house prices in Portland based on size in square feet
  - Supervised: since our dataset has actual house prices
  - Regression: since we're predicting price, a number
- Terminology:
  - m: the number of training examples
  - x: an input variable or row ("features")
  - y: an output variable or row ("target")

- h: our trained learning algorithm ("hypothesis", turns x's into predicted y's)
- θ: a parameter of the learning algorithm
    * Example: in univariate linear regression, our model will be a straight line defined by two numbers a and b so that $y = a + bx$. We just write $h_\theta(x) = \theta_0 + \theta_1 x$

Cost function

- We have a training set, and a family of hypotheses $h_\theta(x) = \theta_0 + \theta_1 x$
- Which values for θ_0 and θ_1 are best?
- We want $h_\theta(x)$ to be close to our y's on our training data
- Measuring closeness with a cost function
    - For each data point (i), distance is the squared difference between what we predicted and what the actual answer was: $(h_\theta(x^{(i)}) - y^{(i)})^2$
    - Distance summed for the whole data set: $J(\theta_0, \theta_1) = \frac{1}{2m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)})^2$
        * The $\frac{1}{2m}$ just makes the math come out more cleanly
    - The values of θ_0 and θ_1 which give us the smallest $J(\theta_0, \theta_1)$ will give us the closest fit to our data
    - So, if we can find these values and minimise the cost function, we will have "trained" our algorithm

Cost function intuition

- See lectures for worked example of cost function
- What does the cost function look like generally?
    - Think of a landscape with lots of mountains, hills and valleys
- What does it look like for linear regression and other "nice" problems?
    - For one variable, the cost function is a U-shaped curve, with the best answer at the bottom
    - For two variables, the cost function is a 2d bowl, with the best answer at the bottom
    - For more variables, it's hard to visualise. Pretend it's a 2d bowl and carry on.
- *How will we code this stuff?* We'll write the code to calculate the cost function, and let automatic solvers find the parameters which minimize it for us. Imagining a bowl, the solver starts somewhere random then heads downhill from there.

Gradient descent

- A technique to find input values which minimise function which minimise it
- For examples, it finds values of θ_0 and θ_1 which minimize $J(\theta_0, \theta_1)$
- Visualising it

  - Imagine you're dropped somewhere randomly in the world, and you walk downhill from there
  - Will you reach the lowest point in the world? Probably not, most likely just the lowest point in your local area
  - A "convex" function is like a volcano crater: no matter where in it you're dropped, walking downhill gets you to the true bottom
  - Nice cost functions, like the ones for linear regression, are convex
- Algorithm loop: $\theta_j := \theta_j - \alpha \frac{\delta}{\delta \theta_j} J(\theta_0, \theta_1)$
  - Don't get too caught up on the partial derivative $\frac{\delta}{\delta \theta_j} J(\theta_0, \theta_1)$, just think of it as how J changes as you change θ_j (e.g. will it become bigger as θ_j becomes bigger?)
  - α is the "learning rate"
    * When it's too small, it will take forever to reach the answer
    * When it's too large, it keeps jumping over the correct answer without settling

Gradient descent for linear regression

- To apply it, we need to calculate a partial derivative of the cost function $J(\theta_0, \theta_1)$
- They endup being:
  - $\frac{\delta}{\delta \theta_0} J(\theta_0, \theta_1) = \frac{1}{m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)})$
  - $\frac{\delta}{\delta \theta_1} J(\theta_0, \theta_1) = \frac{1}{m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)}) \cdot x^{(i)}$
- You don't need to be able to work these out yourself
- The $\cdot$ operator just means mean multiplying the values of two vectors together, then adding the results.
- Technically this is "batch" gradient descent, since we use all the training data in every step

Next week

- Solving linear regression in one step, without gradient descent
  - Cleaner math, but doesn't scale as well
- Linear regression with larger number of features
- A lil' more linear algebra, thinking about our problems as using vectors and matrices