Machine learning notes: week 3

Lars Yencken

Logistic regression

Classification

- When the outcome we're predicting is boolean (or, an enum)
- Examples
  - Is this email spam?
  - Is this tumour malignant?
  - Has the deadly neurotoxin worked?
- Ghetto approach: apply linear regression and use some threshold, but…
  - Outlying examples can really screw up the model
  - Outputs can be way outside the range [0, 1]
- Better approach: use a different family of models which are S shaped curves instead of straight lines

Hypothesis representation

- Let $g(z) = \frac{1}{1+e^{-z}}$ where $z = \theta^T x$
- g is called the {*sigmoid function} or the {*logistic function}
- In practice, this function squishes any z value into something between 0 and 1
  - Large negative values of z become close to zero
  - Large postive values of z become close to one
- Now that $h_\theta(x)$ is between 0 and 1, call it the "probability that $y = 1$"
  - Formally $h_\theta(x) = \Pr(y = 1|x; \theta)$

Decision boundary

- We want to decide if y is 0 or 1, but $h_\theta(x)$ is a floating point number between 0 and 1
- If we pick 0.5 as a threshold, then it means we'll pick
  - $y = 1$ if $\theta^T x > 0$
  - $y = 0$ if $\theta^T x < 0$
- A threshold like this cuts the space for x into two parts, where we always predict $y = 1$ on one side and $y = 0$ on the other
- It's called a *decision boundary*
- Decision boundaries can be non-linear, if we have non-linear features (e.g. polynomial features)

Cost function

- Given our training set, how can we pick θ?
- Use a cost function to determine which is best (i.e. lowest cost), gradient descent to find that θ
- The naive translation of $J(\theta)$ is non-convex (i.e. full of local minima)
- Instead, let $cost(h_\theta(x), y) = \begin{cases} -\log(h_\theta(x)) & \text{if } y = 1 \\ -\log(1 - h_\theta(x)) & \text{if } y = 0 \end{cases}$
- This adds a generous penalty for being wrong in any individual case, and will make $J(\theta)$ convex
- Once again, you don't need to be able to come up with this math, it's enough to understand roughly what it's doing

Gradient descent

- We can add up costs for each case in the training set to get $J(\theta)$:
  - $J(\theta) = \frac{1}{m} \sum_{i=1}^{m} cost(h_\theta(x^{(i)}, y^{(i)}))$
- We can simplify the cost component to one line:
  - $J(\theta) = -\frac{1}{m} [\sum_{i=1}^{m} y^{(i)} \log(h_\theta(x^{(i)}) + (1 - y^{(i)}) \log(1 - h_\theta(x^{(i)}))]$
- For gradient descent, rule looks very similar to linear regression:
  - $\theta_j := \theta_j - \alpha \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)}) x_j^{(i)}$
  - Notice that $h_\theta(x)$ is the only part that's changed
- The same rules as normal apply:
  - Monitor $J(\theta)$ and check that it's decreasing with more iterations
  - Try various values for α and tune appropriately

Advanced optimisation

- Gradient descent calculates $J(\theta)$ and $\frac{\delta}{\delta \theta_j} J(\theta)$ in every loop
- A bunch of advanced algorithms exist which can:
  - Avoid the need to pick α manually
  - Can converge much faster than gradient descent
- They're quite complex to implement – don't roll your own!
- These become quite important for scaling to larger datasets

Multiclass problems

- "multiclass": predicting an enum instead of a boolean
- Examples
  - Email tagging: work, friends, family or hobby
  - Medical diagnosis: not ill, cold, flu
- One vs all
  - Train a model to detect each class, ending up with n models
  - When predicting, run it against all the models and pick the most confident

Regularization

Overfitting

- *Underfitting* or *bias*: when our model doesn't just doesn't fit the data well
- *Overfitting* or *variance*: when the model fits our *training* data very well but works badly on new examples (*generalises* poorly)
- Overfitting's especially a problem when:
  - You have a large number of features
  - You use a high-degree polynomial to fit data
- Approaches to combat it:
  1. Reduce the number of features
  2. *Regularisation*: penalise the use of too many active features in our model, thus ending up with models with less active features

Cost function

- Intiution: smaller values for our parameters means a simpler hypothesis
  - For example, if less important parameters θ_j are set to zero, our model no longer uses those terms
- Achieve this by adding an extra term to our cost function
  - $J(\theta) = \frac{1}{2m}[\sum_{i=1}^{m}(h\theta(x^{(i)}) - y^{(i)})^2 + \lambda \sum_{i=1}^{n} \theta_j^2]$
  - Note that by convention there's no penalty on θ_0
  - λ is called the *regularisation parameter*
- If λ is too large, we will underfit
- If λ is too low, we may overfit

Regularized linear regression

- We have a slightly updated update rule for gradient descent:
  - For θ_0, the rule is the same as before
  - For θ_j where $j > 0$, we have $\theta_j := \theta_j - \alpha[\frac{1}{m}\sum_{i=1}^{m}(h\theta(x^{(i)}) - y^{(i)})x_j^{(i)} + \frac{\lambda}{m}\theta_j]$
    * Refactored: $\theta_j := \theta_j(1 - \alpha\frac{\lambda}{m}) - \alpha\frac{1}{m}\sum_{i=1}^{m}(h\theta(x^{(i)}) - y^{(i)})x_j^{(i)}$
    * The term $(1 - \alpha\frac{\lambda}{m})$ must be < 1, and has the effect of favouring shrinking the parameter, all else being even
- Normal equation updated:
  - $\theta = (X^TX + \lambda D)^{-1}X^Ty$, where D is 0 at $(1, 1)$ and 1 down the rest of the diagonal
  - Advanced: if $\lambda > 0$, the above matrix being inverted will never be singular (i.e. non-invertible)

Regularized logistic regression

- Add the regularization term to our cost function:
  - $J(\theta) = -\frac{1}{m}[\sum_{i=1}^{m} y^{(i)} \log(h_\theta(x^{(i)}) + (1 - y^{(i)}) \log(1 - h_\theta(x^{(i)}))] + \frac{\lambda}{2m}\sum_{j=1}^{n}\theta_j^2$
- The update rule is the same as for linear regression, except that our $h\theta(x)$ uses the sigmoid function