Machine Learning Lecture 4
Linear Regression

Dr. Ioannis Patras
EECS, QMUL 2015

Slide thanks: Dr. Tim Hospedales

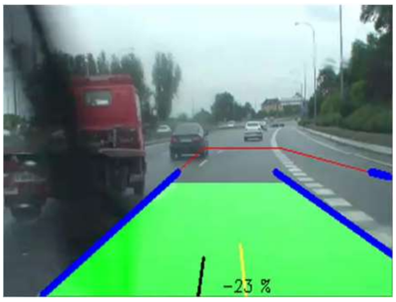
Course Context

- Supervised Learning
  - (Linear) regression
  - Logistic Regression (Classification)
  - Neural Networks
- Unsupervised
  - Clustering
  - Density Estimation
  - HMMs

Supervised Learning

- Applications where the training data comprises examples of input vectors along with corresponding target vectors
- Linear Regression: desired output consists of one or more continuous variables
- Logistic "Regression": Desired output consists of a finite number of discrete categories.
  - (Will explain the confusing name next week)

Regression Applications



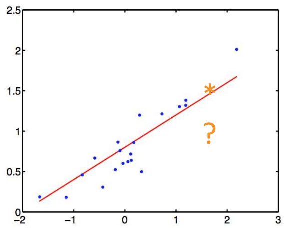
- Autonomous vehicles (what are the outputs?)
- House market prediction

Outline

- Linear Regression
- Non-linear regression
- Overfitting and underfitting
- Regularisation and cross-validation
- Probabilistic Interpretation
- Practical Learning

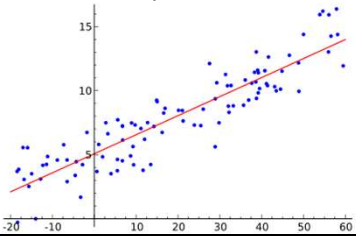
Linear Regression

- Goal:
  - Predict the value of one or more continuous target variables *y* given a D-dimensional vector ***x*** of inputs..
- Assume an unknown continuous function y=f(x)
- Given examples (x_i,y_i), which may be noisy
- Learn f(x), to enable prediction of y* given new point x*. It should generalise well to new x*
  - E.g., x: Temperature, y=Ice Cream Sales



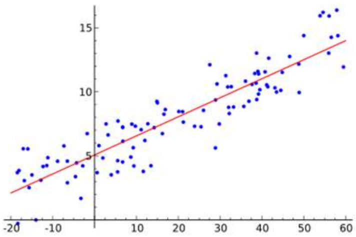
Example Applications

- Car Mileage, Year => Price
- Temperature => Ice Cream Demand
- Voltage => Temperature
- House Postcode, Rooms, Square Meters => Price
- Age, Salary, Past Claims => Insurance Premium
- Robot Arm Reach Target => Arm Motor Torque



Various Settings (1-d linear)

- Given training set of **x**=(**x**$_1$,…**x**$_N$), y=(**y**$_1$,..,**y**$_N$). Learn weights w to predict y=f$_w$(x)
- Single input: $y = f(x) = w_0 + w_1 x = \mathbf{w}^T[1, x]$
  - Fit a line defined by ***w***



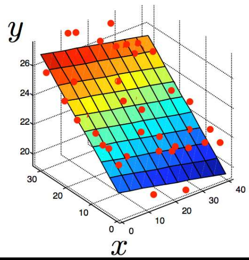
Various Settings (N-d, linear)

- Given training set of $\mathbf{x}=(\mathbf{x}_1,\ldots\mathbf{x}_N)$, $y=(\mathbf{y}_1,..,\mathbf{y}_N)$. Learn weights w to predict $y=f_w(x)$

- Multiple inputs

$$y=f(\mathbf{x})=w_1x_1+w_2x_2+w_3x_3=\mathbf{w}^T\mathbf{x}$$

– Fit a plane (defined by **w**).
(Linear wrt **w**, linear wrt **x**)



Various Settings (N-d, polynomial)

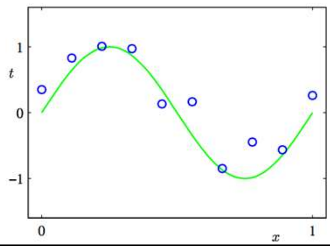
- Given training set of $\mathbf{x}=(\mathbf{x}_1,\ldots\mathbf{x}_N)$, $y=(\mathbf{y}_1,..,\mathbf{y}_N)$. Learn weights w to predict $y=f_w(x)$

- Non-linear (polynomial)-> with respect to x!!!

$$y=f(x)=w_0+w_1x+w_2x^2+w_3x^3\ldots$$

$$y=f(x)=\mathbf{w}^T\phi(x)$$
$$\phi(x)=(1,x,x^2,x^3)^T$$

(Linear wrt **w**, non-linear wrt **x**)



Various Settings

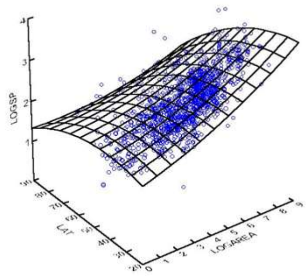
- Given training set of $\mathbf{x}=(\mathbf{x}_1,\ldots\mathbf{x}_N)$, $y=(\mathbf{y}_1,..,\mathbf{y}_N)$. Learn weights w to predict $y=f_w(x)$

- Non-linear and multivariate:

$$f(\mathbf{x}) = w_0 + w_1 x_1 + w_2 x_2 + w_3 x_1 x_2 + w_4 x_1^2 + w_5 x_2^2$$

$$y = f(x) = \mathbf{w}^T \phi(x)$$

(Linear wrt **w**, non-linear wrt **x**)



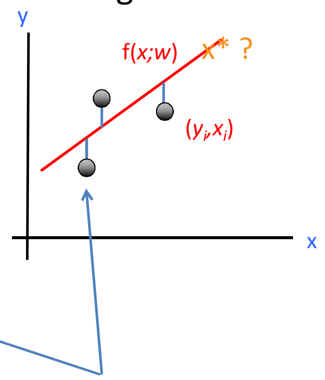
Settings:

- Given training set of $\mathbf{x}=(\mathbf{x}_1,\ldots\mathbf{x}_N)$, $y=(\mathbf{y}_1,..,\mathbf{y}_N)$. Learn weights w to predict $y=f_w(x)$
  - Single input, linear
  - Single input, non-linear
  - Multiple input, linear
  - Multiple input, non-linear
- In each case have to find the weight w so the line predicts the data well. Linear wrt the unknown w
  - How?

Error (Cost) Function

- How to find a good setting for weights w?
  - First need to quantify how well the line goes through the train points
- Most common cost
  - Sum of squared errors

$$E(\mathbf{w}) = \sum_{i=1}^{N} (y_i - f(\mathbf{x}_i;\mathbf{w}))^2$$



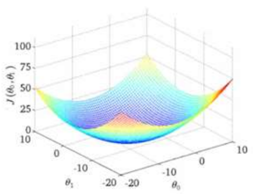
(Focusing on linear for now)

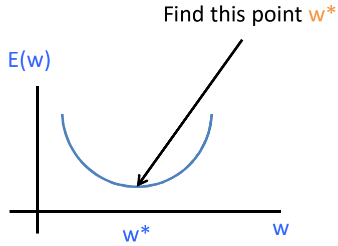
Adds up these errors

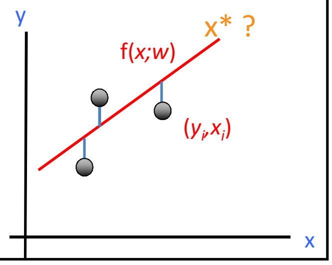
Error (Cost) Function

- Goal: Find the best line by choosing w such that E(w) is as small as possible
  - Line would then be f(x,w*)
- E.g., 1D linear regression has 2 params. So cost like this:

But how to do this without trying all ws?



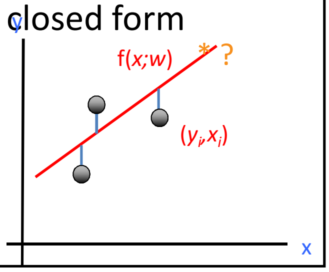




Error (Cost) Function

- Goal: Find the best line by choosing w such that E(w) is as small as possible
- Error function is quadratic in w
  - => Derivatives wrt w will be linear.
  - => Error is (1) convex, and (2) has a closed form solution for its minimum w*.

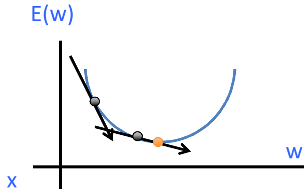
$$E(\mathbf{w}) = \sum_{i=1}^{N} (y_i - f(\mathbf{x}_i;\mathbf{w}))^2$$

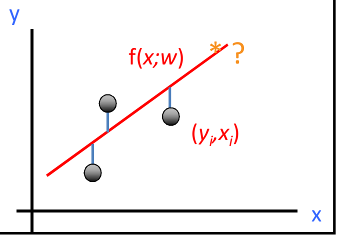


Solving Error (Cost) Function

- Two ways to find the minimum:
  - Gradient
  - Closed form solution

$$E(\mathbf{w}) = \sum_{i=1}^{N} (y_i - f(\mathbf{x}_i;\mathbf{w}))^2$$



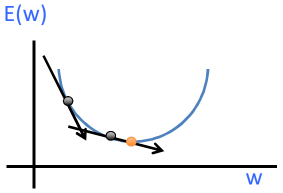


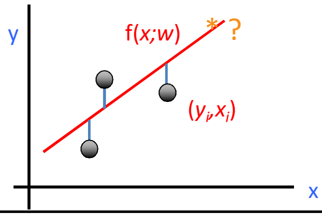
Solving Error (Cost) Function: Gradient

- Cost is: $E(\mathbf{w}) = \sum_{i=1}^{N}(y_i - f(\mathbf{x}_i;\mathbf{w}))^2 = \sum_{i=1}^{N}(y_i - \mathbf{w}^T\mathbf{x}_i)^2$
- Derivatives wrt w: $\frac{dE(\mathbf{w})}{d\mathbf{w}} = 2\sum_i -\mathbf{x}_i(y_i - \mathbf{w}^T\mathbf{x}_i)$
- Algorithm:
  - Get the gradient at any point x
  - Move in direction of gradient
  - Go until converged

    (no change, i.e. gradient very small)

$$\mathbf{w}^{s+1} := \mathbf{w}^s - \alpha\frac{dE(\mathbf{w})}{d\mathbf{w}}$$
$$\mathbf{w}^{s+1} := \mathbf{w}^s + \alpha\sum_i \mathbf{x}_i(y_i - \mathbf{w}^T\mathbf{x}_i)$$

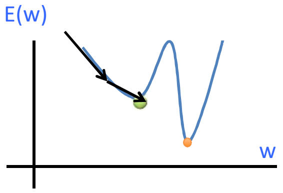




Solving Error (Cost) Function: Gradient and Convexity

- Algorithm: Repeat:
  - Start at random w
  - Repeat: Move in direction of gradient
- How do we know this works?
  - How do we know it converges?
  - How do we know it converges to the global optima?

$$E(\mathbf{w}) = \sum_{i=1}^{N}(y_i - \mathbf{w}^T\mathbf{x}_i)^2$$
$$\mathbf{w}^{s+1} := \mathbf{w}^s + \alpha\sum \mathbf{x}_i(y_i - \mathbf{w}^T\mathbf{x}_i)$$

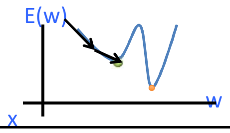


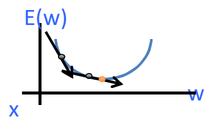
If the error surface is like this, we would converge to local rather than global optima.

How do we know what the error surface is like?

Solving Error (Cost) Function: Gradient and Convexity

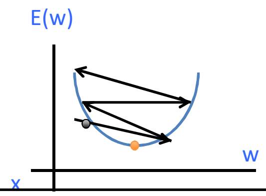
- Convex cost functions: $E(\mathbf{w})=\sum_{i=1}^{N}\left(y_i-\mathbf{w}^T\mathbf{x}_i\right)^2$
  - Have a single minima
  - And advanced gradient-based algorithms to optimise them efficiently
- Convex if Hessian (second derivative matrix) is positive.
  - Least squares cost function is convex! ☺
  - (Proof omitted. Read in Barber Sec 17.4.1)





Solving Error (Cost) Function: Gradient and Convexity

- Least squares has a single minima: Ok!
  - Q: But is gradient descent guaranteed to converge?
- With a big learning rate, it may not ☹
  - Possible to overshoot and diverge
  - Guaranteed only with a “sufficiently” small alpha.
  - Tradeoff: Fast learning vs stability.
    - For quadratic costs like this there are algorithms (conjugate gradient) to determine optimal alpha.



$$E(\mathbf{w})=\sum_{i=1}^{N}\left(y_i-\mathbf{w}^T\mathbf{x}_i\right)^2$$

$$\mathbf{w}^{s+1}:=\mathbf{w}^{s}+\alpha\sum_{i}\mathbf{x}_i(y_i-\mathbf{w}^T\mathbf{x}_i)$$

Algorithm

- Input: Data x, Labels y, Learning Rate alpha.
- w^0=random
- Repeat:

$$\mathbf{w}^{s+1} := \mathbf{w}^{s} + \alpha \sum \mathbf{x}_i (y_i - \mathbf{w}^T \mathbf{x}_i)$$

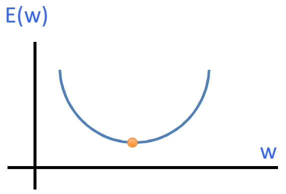
- Until Convergence ($|\mathbf{w}^{s+1} - \mathbf{w}^{s}| < \varepsilon$)
- Output: w^s

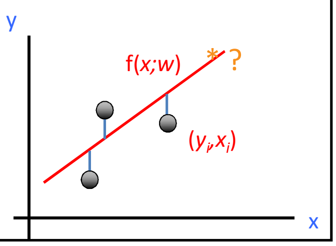
Solving Error (Cost) Function

- Two ways to find the minimum:
  - Gradient
  - Closed form solution
    - (Relies on the convexity)

$$E(\mathbf{w}) = \sum_{i=1}^{N} \left(y_i - f(\mathbf{x}_i; \mathbf{w})\right)^2$$

- Solve for $\frac{dE}{d\mathbf{w}} = 0$





Solving Error (Cost) Function: Closed form Solution

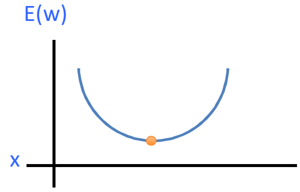
- Solve for zero derivative

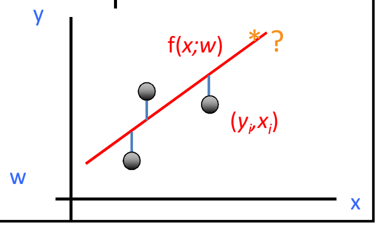
$$E(\mathbf{w}) = \sum_{i=1}^{N} \left(y_i - f(\mathbf{x}_i;\mathbf{w})\right)^2 \quad = (\mathbf{y} - \mathbf{X}\mathbf{w})^T(\mathbf{y} - \mathbf{X}\mathbf{w})$$

$$\frac{dE}{d\mathbf{w}} = \mathbf{X}^T(\mathbf{y} - \mathbf{X}\mathbf{w})$$

$$0 = \mathbf{X}^T(\mathbf{y} - \mathbf{X}\mathbf{w})$$

$$\hat{\mathbf{w}}_{ols} = (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{y}$$





Exact, non-iterative solution.
But in-memory requirement and matrix inversion means not big data

Generalization to Multiple Outputs

- So far we looked at a single output
  - Sometimes want to predict multiple outputs
    - (e.g., control for every joint of a robot)
- Option 1:
  - Do a linear regression for each output independently
- Option 2:
  - Learn a single multi output regression
  - Vector **w** => matrix **W**

$$y = \mathbf{w}^T\mathbf{x} \quad \Rightarrow \quad \mathbf{y} = \mathbf{W}^T\mathbf{x}$$

Generalization to Multiple Outputs

- So far we looked at a single output
  - Sometimes want to predict multiple outputs
    - (e.g., control for every joint of a robot)
- Learn a single multi output regression
  - Still amenable to calculus + linear algebra solution

$$y = \mathbf{w}^T \mathbf{x} \quad \Rightarrow \quad \mathbf{y} = \mathbf{W}^T \mathbf{x}$$

$$\hat{\mathbf{w}}_{ols} = (\mathbf{X}^T \mathbf{X})^{-1} \mathbf{X}^T \mathbf{y} \qquad \hat{\mathbf{W}}_{ols} = (\mathbf{X}^T \mathbf{X})^{-1} \mathbf{X}^T \mathbf{Y}$$

Summary

- Linear Regression aims to learn a line of best fit through data.
  - "Goodness" of a line commonly quantified by SSE
- Minimizing SSE has two solutions:
  - Closed form linear algebra
  - Gradient descent
    - Quite fast and reliable since SSE is convex.
- Generalizations to multiple inputs and outputs

Outline

- Linear Regression
- Non-linear regression
- Overfitting and underfitting
- Regularisation and cross-validation
- Probabilistic Interpretation
- Practical Learning

Non-linear regression

- So far focused on linear regression

$$y = f(\mathbf{x}) = \mathbf{w}^T \mathbf{x}$$

- Non-linear regression is often also of interest

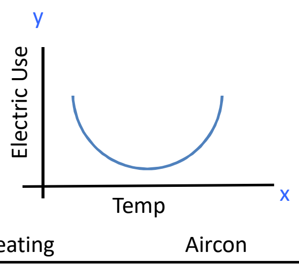
$$f_2(x) = w_0 + w_1 x^1 + w_2 x^2$$

Complicated?
Will we need a different algorithm each time?

$$f_3(x) = w_0 + w_1 x^1 + w_2 x^2 + w_3 x^3$$

$$f_{ss}(x) = w_0 + w_1 \sin(x)$$

$$f_g(x) = w_1 \exp - \frac{(x-u_1)^2}{2\sigma_1^2} + w_2 \exp - \frac{(x-u_2)^2}{2\sigma_2^2}$$



Non-linear regression

- Actually, we can use any (fixed) basis function.

$$y = \mathbf{w}^T \phi(\mathbf{x})$$

- Linear wrt. w!!

$$f(x) = w_0 + w_1 x \longrightarrow \phi(x) = (1, x)^T$$

$$f_3(x) = w_0 + w_1 x^1 + w_2 x^2 + w_3 x^3 \longrightarrow \phi(x) = (1, x, x^2, x^3)^T$$

$$f_{ss}(x) = w_0 + w_1 \sin(x) \longrightarrow \phi(x) = (1, \sin(x))^T$$

$$f_g(x) = w_1 \exp - \frac{(x-u_1)^2}{2\sigma_1^2} + w_2 \exp - \frac{(x-u_2)^2}{2\sigma_2^2}$$

$$\phi(x) = (N(x; \mu_1, \sigma_1), N(x; \mu_2, \sigma_2))^T$$

Non-linear regression

- Cost is now:

$$E(\mathbf{w}) = \sum_{i=1}^{N} \left(y_i - \mathbf{w}^T \mathbf{x}_i) \right)^2 \Rightarrow E(\mathbf{w}) = \sum_{i=1}^{N} \left(y_i - \mathbf{w}^T \phi(\mathbf{x}_i)) \right)^2$$

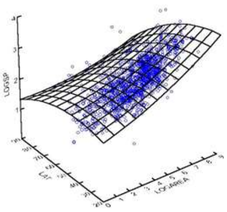
- Solution is now either:

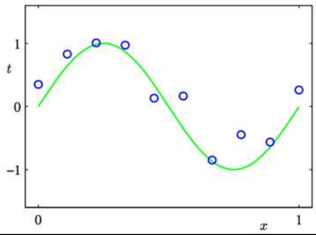
$$\hat{\mathbf{w}}_{ols} = (\Phi^T \Phi)^{-1} \Phi^T \mathbf{y} \qquad \mathbf{w}^{s+1} := \mathbf{w}^s + \alpha \sum_i \phi(\mathbf{x}_i)^T (y_i - \mathbf{w}^T \phi(\mathbf{x}_i))$$

- Model is non-linear in x.
- But cost is linear in w. So we don't need to worry about non-linearity phi.
  - Always easy to optimize, with either algorithm.

Summary

- We can fit non-linear curves to data
  - Use basis functions to express the category of curve
- Solution is easy and efficient, ~ independent of the specific basis functions
  - Gets more complicated if also want to learn the basis functions (see neural net class)





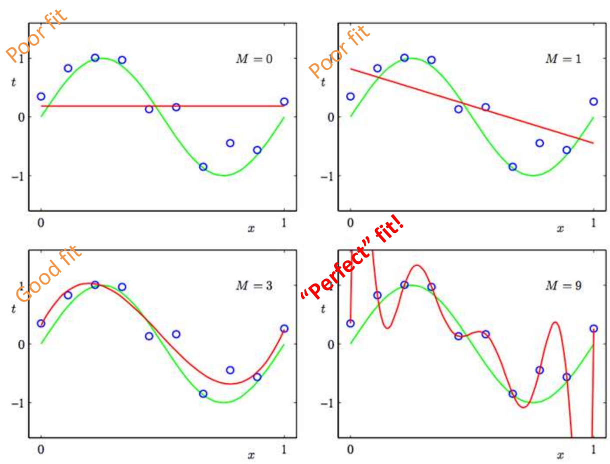
Outline

- Linear Regression
- Non-linear regression
- Overfitting and underfitting
- Regularisation and cross-validation
- Probabilistic Interpretation
- Practical Learning

Consider Polynomial regression…

$$f_M(x) = w_0 + w_1 x^1 + w_2 x^2 + .. + w_M x^M = \sum_{j=0}^{M} w_j x^j$$

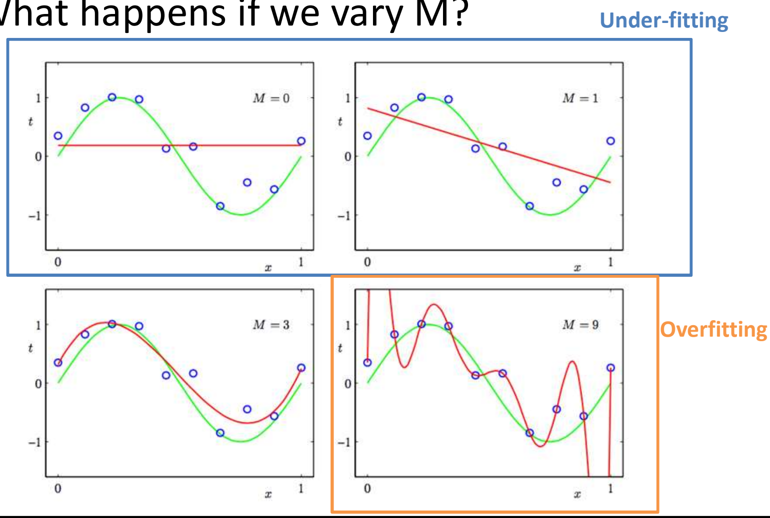
- What happens if we vary M?



Consider Polynomial regression…

$$f_M(x) = w_0 + w_1 x^1 + w_2 x^2 + .. + w_M x^M = \sum_{j=0}^{M} w_j x^j$$

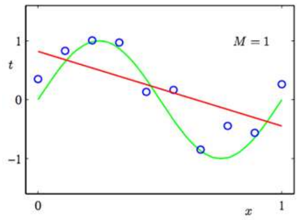
- What happens if we vary M?

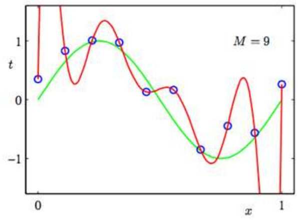


Under and overfitting

$$f_M(x) = w_0 + w_1 x^1 + w_2 x^2 + .. + w_M x^M = \sum_{j=0}^{M} w_j x^j$$

- What happens if we vary M?
- Underfitting
  - Inflexible polynomials can't explain the data
  - Poor train fit
  - Poor generalisation
- Overfitting
  - Too-flexible polynomials tune into noise
  - Perfect train fit
  - Terrible generalisation





Overfitting

- Key is that we care about (performance in the unseen test data!):

$$E_{X^{test}}(\mathbf{w}) = \sum_{i=1}^{Test} \left(y_i - \mathbf{w}^T \mathbf{x}_i)\right)^2$$

Not completely related

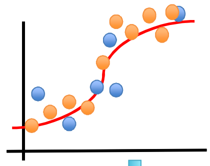
- But what we can actually optimise is:

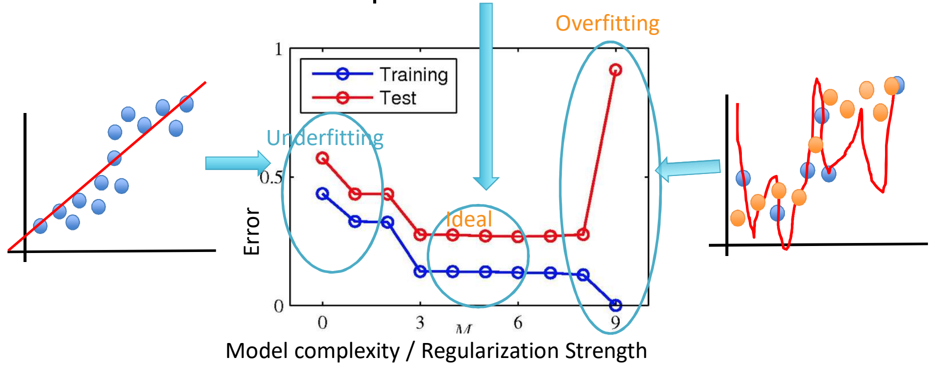
$$E_{X^{train}}(\mathbf{w}) = \sum_{i=1}^{Train} \left(y_i - \mathbf{w}^T \mathbf{x}_i)\right)^2$$

Non-linear regression: From under to over-fitting

$$E_{MSE}(\mathbf{w}, \mathbf{X}_{train}) = \sum_i (y_i - f(\mathbf{x}_i^{tr}))^2$$

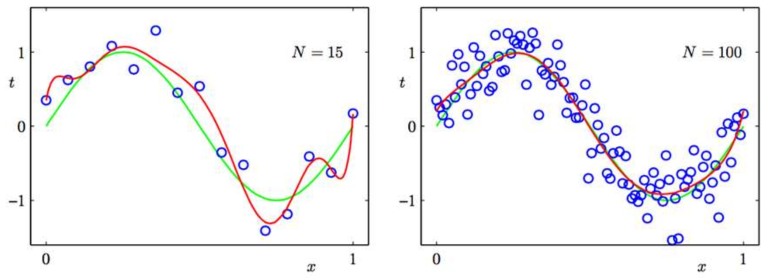
$$E_{MSE}(\mathbf{w}, \mathbf{X}_{test}) = \sum_i (y_i - f(\mathbf{x}_i^{te}))^2$$





Effect of Dataset Size

- For a given model complexity, overfitting becomes less severe as dataset size increases
- The larger dataset, the more complex (flexible) model we can fit (without suffering overfitting)



Effect of Dataset Size

- For a given model complexity, overfitting becomes less severe as dataset size increases
- The larger dataset, the more complex (flexible) model we can fit (without suffering overfitting)
- But many contemporary models of interest are hugely complex, or infinitely complex
  - Neural nets, RBF Kernel SVMs, etc
  - So overfitting is a pervasive issue.

Summary

- Overfitting occurs when the model is complex or the amount of data is small
  - => Good train, poor test performance
- Underfitting occurs when the model is too simple
  - Poor train, poor test performance

Outline

- Linear Regression
- Non-linear regression
- Overfitting and underfitting
- Regularisation and cross-validation
- Probabilistic Interpretation
- Practical Learning

Regularization

- Overfitting is controlled by regularization
  - Add a penalty to our cost to discourage heavy use of all the weights.
  - Commonly l2-norm, or sum-squared value of weights.
  - Also known as "shrinkage" or "weight decay" techniques, because they reduce the values of coefficients

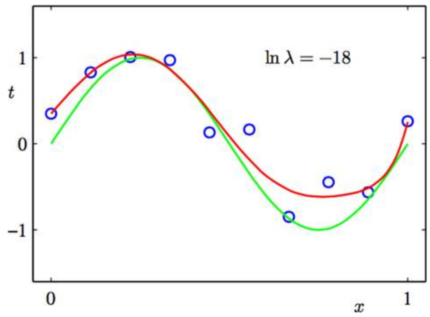
$$E(\mathbf{w}) = \sum_{i=1}^{N} \left(y_i - \mathbf{w}^T \phi(\mathbf{x}_i) \right)^2 \Rightarrow E(\mathbf{w}) = \sum_{i=1}^{N} \left(y_i - \mathbf{w}^T \phi(\mathbf{x}_i) \right)^2 \boxed{+ \lambda \mathbf{w}^T \mathbf{w}}$$

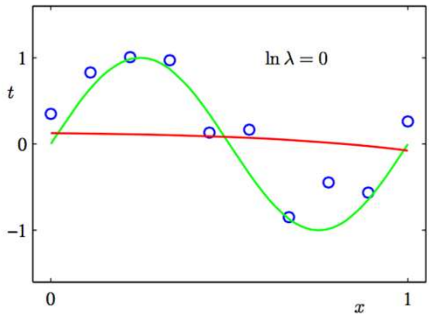
This specific (l2 norm) model is called Ridge Regression in statistics.

Regularization

- Regularisation parameter λ controls complexity
  - By defining a tradeoff between fit and weight decay.

$$E(\mathbf{w}) = \sum_{i=1}^{N} \left(y_i - \mathbf{w}^T \phi(\mathbf{x}_i)\right)^2 + \lambda \mathbf{w}^T \mathbf{w}$$

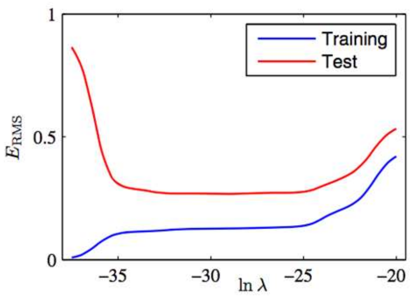




Regularization

- Regularisation parameter λ controls the complexity… and hence degree of over/under fitting

$$E(\mathbf{w}) = \sum_{i=1}^{N} \left(y_i - \mathbf{w}^T \phi(\mathbf{x}_i)\right)^2 + \lambda \mathbf{w}^T \mathbf{w}$$



Learning Regularized Regression

- Learning with regularization.
  - Usual procedure: Differentiate E(w) wrt w, and get gradient or closed form solution

$$E(\mathbf{w}) = \frac{1}{2}\sum_{i=1}^{N}\left(y_i - \mathbf{w}^T\phi(\mathbf{x}_i)\right)^2 + \lambda\frac{1}{2}\|\mathbf{w}\|^2$$

$$\begin{cases} \hat{\mathbf{w}}_{ols} = (\Phi^T\Phi + \lambda\mathbf{I})^{-1}\Phi^T\mathbf{y} \\ \mathbf{w}^{s+1} := \mathbf{w}^s + \alpha\sum_i \phi(\mathbf{x}_i)^T(y_i - \mathbf{w}^T\phi(\mathbf{x}_i))\boxed{-\lambda\mathbf{w}^s} \end{cases}$$

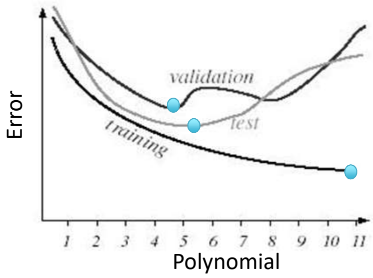
Always pushing large w down toward zero.

Determining Regularisation Strength

- Simple strategy: Use a validation set
  - Split your data X into:
    - X^{tr} Training set: determine w
    - X^{val} Validation set: tune lambda
  - We hope that performance on X^{val} reflects performance on (the unknown) X^{test}.
  - Because the data for training w excludes X^{val}:
    - If w is overfit, it will perform badly on X^{val}
    - Thus we can "safely" use performance on Xval to determine the right complexity
  - Pay attention to how you split the data in train/validation

Determining Regularisation Strength

- Validation set performance should reflect test performance better than train performance

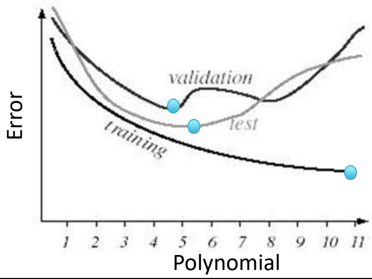


Picking Model Complexity with a Validation Set

- Validation error should approximate test error better than train set

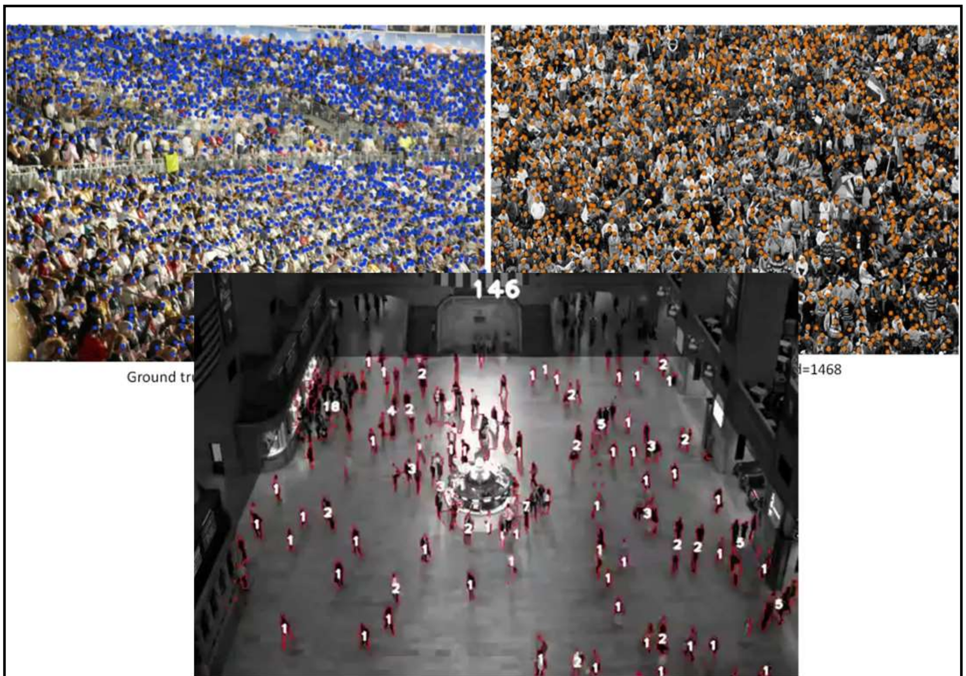
Pseudocode:

- Split data into Train and Validation subsets
- For λ = 0,0.1,1,10…
  - w=MakeRegularizedModel(Xt,Yt, λ)
  - EstimatedError(λ) = EvaluateModel(w,Xv,Yv)
- Pick λ with minimum estimated error



Case Study: Crowd Counting
EECS Research ☺

- "Crowd Counting": Regression problem from x: image pixels to a y: number of people in the scene.
  - x: millions of dimensions
  - y: highly non-linear function of x
- Highly desired by:
  - Retail, Security, Airports, Urban Planners, etc
  - (Airports want to reduce queues!)



Outline

- Linear Regression
- Non-linear regression
- Overfitting and underfitting
- Regularisation and cross-validation
- Probabilistic Interpretation
- Practical Learning

A Probabilistic Interpretation

- Our SSE cost is conveniently convex

$$E(\mathbf{w}) = \sum_{i=1}^{N} \left(y_i - \mathbf{w}^T \mathbf{x}_i \right)^2$$

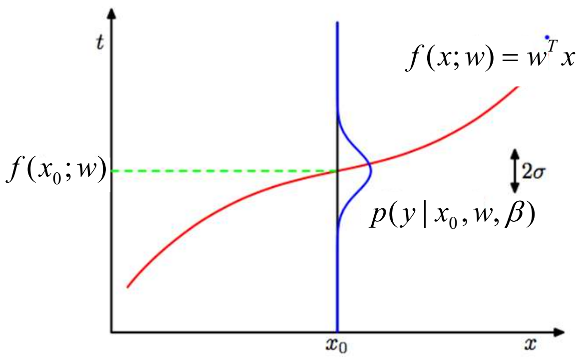
- …but does it relate to anything probabilistic we studied in the previous class?
  - We've said $y = \mathbf{w}^T \mathbf{x}$
  - But suppose $p(y \mid \mathbf{x}, \mathbf{w}, \beta) = N(y \mid \mathbf{w}^T \mathbf{x}, \beta^{-1})$
- i.e., target variable is fit by a Gaussian probability with mean $\mathbf{w}^T\mathbf{x}$ and variance β.

Illustration

- Schematic Gaussian distribution for y given x:
  - Before a single line => Now a distribution at each point x

$$p(y \mid \mathbf{x}, \mathbf{w}, \beta) = N(y \mid \mathbf{w}^T \mathbf{x}, \beta^{-1})$$

β encodes expected dispersion around the mean



Likelihood Function

- If you are modeling a probability of output y.
  - What's the best single prediction, given x?

$$p(y \mid \mathbf{x}, \mathbf{w}, \beta) = N(y \mid \mathbf{w}^T \mathbf{x}, \beta^{-1})$$

- Optimal prediction would be given by the mean of the target variable's distribution (*)
  - (This is exactly what we did before: $y=w^Tx$!)
  - (*) In the case of Gaussian

Likelihood Function

- Means that for IID X={x_1…x_n}. => y distributed:

$$p(\mathbf{y} \mid \mathbf{x}, \mathbf{w}, \beta) = \prod_{n=1}^{N} N(y_n \mid \mathbf{w}^T \mathbf{x}_n, \beta^{-1})$$

- How would we learn **w** or β in this case?
- Recall maximum likelihood strategy:

$$\hat{\theta} = \operatorname{argmax} p(X \mid \theta)$$

- Here:

$$\mathbf{y} \quad \mathbf{w} \qquad \mathbf{x}, \beta$$

Additional conditioning

Likelihood Function

- Means that for IID X={x_1…x_n}. => y distributed:

$$p(\mathbf{y} \mid \mathbf{x}, \mathbf{w}, \beta) = \prod_{n=1}^{N} N(y_n \mid \mathbf{w}^T \mathbf{x}_n, \beta^{-1})$$

- Solve **w** with MLE:

SSE Loss from before!

$$\log p(\mathbf{y} \mid \mathbf{x}, \mathbf{w}, \beta) = \frac{N}{2}\log\beta - \frac{N}{2}\log 2\pi - \frac{\beta}{2}\boxed{\sum_{n=1}^{N}(\mathbf{y}_n - \mathbf{w}^T\mathbf{x}_n)^2}$$

$$\nabla_{\mathbf{w}} \log p(\mathbf{y} \mid \mathbf{x}, \mathbf{w}, \beta) = \beta \sum_{n=1}^{N} (\mathbf{y}_n - \mathbf{w}^T \mathbf{x}_n)\mathbf{x}_n^T$$

SEE Gradient from before is special case when β=1

Could also solve dispersion β with MLE.

Regularization and Priors

- Assumption: $p(y \mid \mathbf{x}, \mathbf{w}, \beta) = N(y \mid \mathbf{w}^T\mathbf{x}, \beta^{-1})$
  - … leads to SSE cost: $E(\mathbf{w}) = \sum_{i=1}^{N}\left(y_i - \mathbf{w}^T\mathbf{x}_i\right)^2$
- How do regularizers fit in? $p(\mathbf{w}) = N(\mathbf{w} \mid 0, \lambda^{-1})$
  - Regularizers are like priors on w
- Recall MAP learning
- Now we have:

$$\hat{\theta} = \underset{\theta}{\operatorname{argmax}}\ p(\theta \mid d)$$

$$\hat{\theta} = \underset{\theta}{\operatorname{argmax}}\ p(d \mid \theta)p(\theta)$$

$$\operatorname{argmax}_{\mathbf{w}} N(y \mid \mathbf{w}^T\mathbf{x}, \beta^{-1}) N(\mathbf{w} \mid 0, \lambda^{-1})$$

Regularization and Priors

- Regularizers are like priors on w $p(\mathbf{w}) = N(\mathbf{w} \mid 0, \lambda^{-1})$
- Now we have:

$$\operatorname{argmax}_{\mathbf{w}} N(y \mid \mathbf{w}^T\mathbf{x}, \beta^{-1}) N(\mathbf{w} \mid 0, \lambda^{-1})$$

- Posterior distribution:

$$p(\mathbf{w} \mid \mathbf{x}, y, \beta) \propto \exp -\frac{\beta}{2}\sum_{n=1}^{N}(\mathbf{y}_n - \mathbf{w}^T\mathbf{x}_n)^2 \exp -\frac{\lambda}{2}\mathbf{w}^T\mathbf{w}$$

- MLE solution with this prior?

$$\nabla_{\mathbf{w}} \log p(\mathbf{w} \mid \mathbf{y}, \mathbf{x}, \mathbf{w}, \beta) = \underbrace{\beta\sum_{n=1}^{N}(\mathbf{y}_n - \mathbf{w}^T\mathbf{x}_n)\mathbf{x}_n^T - \lambda\mathbf{w}}_{\text{Regularized SSE Gradient from before!}}$$

$$\hat{\theta} = \underset{\theta}{\operatorname{argmax}}\ p(\theta \mid d)$$

$$\hat{\theta} = \underset{\theta}{\operatorname{argmax}}\ p(d \mid \theta)p(\theta)$$

Regularization and Priors

- Regularizers are like priors on w $p(\mathbf{w}) = N(\mathbf{w} \mid 0, \lambda^{-1})$
- Posterior distribution of weights:

$$p(\mathbf{w} \mid \mathbf{x}, y, \beta) \propto \exp -\frac{\beta}{2}\sum_{n=1}^{N}(\mathbf{y}_n - \mathbf{w}^T\mathbf{x}_n)^2 \exp -\frac{\lambda}{2}\mathbf{w}^T\mathbf{w}$$

- MAP estimation:

$$\nabla_{\mathbf{w}} \log p(\mathbf{w} \mid \mathbf{y}, \mathbf{x}, \mathbf{w}, \beta) = \beta\sum_{n=1}^{N}(\mathbf{y}_n - \mathbf{w}^T\mathbf{x}_n)\mathbf{x}_n^T - \lambda\mathbf{w}$$

- Implications:
  - "arbitrary" l2 regularizer from before is in fact encoding the prior belief that:
  - weights **w** are normally distributed with zero mean and lambda variance!

Understanding Regularizers as Priors

$$\text{argmax}_{\mathbf{w}}\ p(y \mid \mathbf{w}^T\mathbf{x}, \beta^{-1})p(\mathbf{w} \mid 0, \lambda^{-1})$$

- Previous regularizer : prior belief that **w** is normally distributed with zero mean and lambda variance.
- Leads to (1):
  - If we have some prior weight knowledge, we can plugin a non-zero mean! (used in multi-task learning)
  - E.g., predicting customer's restaurant satisfaction.
    - If we have a model for a previous restaurant. Can leverage it to help learn new restaurant.
    - Regularize toward $\mathbf{w}_{\text{old}}$ rather than toward 0.

$$N(\mathbf{w} \mid 0, \lambda^{-1}) \Rightarrow N(\mathbf{w} \mid \mathbf{w}_{old}, \lambda^{-1})$$

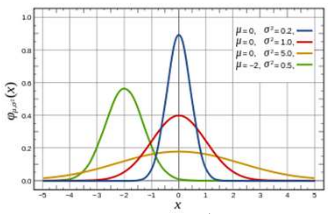
Understanding Regularizers as Priors

$$\text{argmax}_{\mathbf{w}}\, p(y \mid \mathbf{w}^T\mathbf{x}, \beta^{-1})\, p(\mathbf{w} \mid 0, \lambda^{-1})$$

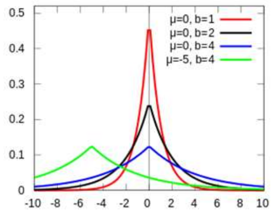
Sparse := Mostly Zero

- Leads to (2):
  - Sometimes non-Gaussian weight priors may be suitable. E.g., Laplace distribution prior:
  - Useful for getting sparse weights w.

$$p(\mathbf{w}) = Lap(\mathbf{w} \mid 0, \lambda^{-1}) = \exp(-\lambda|\mathbf{w}|)$$



Gaussian Distribution



Laplacian Distribution

Understanding Regularizers as Priors

$$\text{argmax}_{\mathbf{w}}\, p(y \mid \mathbf{w}^T\mathbf{x}, \beta^{-1})\, p(\mathbf{w} \mid 0, \lambda^{-1})$$

Laplace Prior:

- Aka L1 regularizer. Because leads to regularizer that subtract l1 norm of weights.
- CF: Gaussian prior => l2 regularizer.

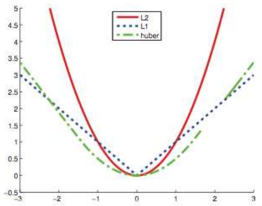
$$p(\mathbf{w} \mid \mathbf{x}, y, \beta) \propto \exp -\frac{\beta}{2}\sum_{n=1}^{N}(\mathbf{y}_n - \mathbf{w}^T\mathbf{x}_n)^2 \exp(-\lambda|\mathbf{w}|)$$

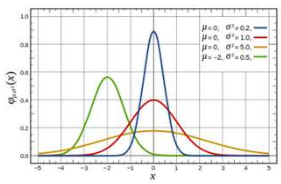
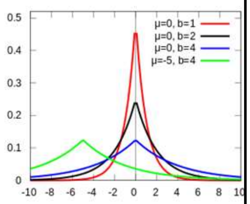
$$\log p(\mathbf{w} \mid \mathbf{y}, \mathbf{x}, \mathbf{w}, \beta) = -\frac{\beta}{2}\sum_{n=1}^{N}(\mathbf{y}_n - \mathbf{w}^T\mathbf{x}_n)^2 - \lambda|\mathbf{w}| + K$$

Aside: Sparse Weights

$$\text{argmax}_{\mathbf{w}}\, p(y \mid \mathbf{w}^T\mathbf{x}, \beta^{-1})p(\mathbf{w} \mid 0, \lambda^{-1}) \qquad p(\mathbf{w}) = \exp(-\lambda|\mathbf{w}|)$$

- Laplace prior => sparse weights w. Why?
- Why might sparse weights be good?
  - Some dimensions are unrelated noise. Kill them.
  - Domain knowledge by finding non-zero weight dims
  - Save memory by finding ignorable dims/columns



Understanding Cost Functions as Likelihoods

$$\text{argmax}_{\mathbf{w}}\, p(y \mid \mathbf{w}^T\mathbf{x}, \beta^{-1})p(\mathbf{w} \mid 0, \lambda^{-1})$$

- Previous likelihood: y is Gaussian distributed with mean of $\boldsymbol{w}^T\boldsymbol{x}$.
- Leads to:
  - Sometimes non-Gaussian likelihoods may be suitable. E.g., laplace likelihood:
  - Useful for robust regression.

$$p(y \mid \mathbf{w}^T\mathbf{x}, \beta^{-1}) = \exp-\beta|y - \mathbf{w}^T\mathbf{x}|$$

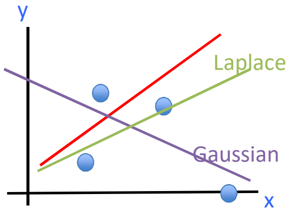
Aside: Robust Regression

$$\text{argmax}_{\mathbf{w}}\, p(y \mid \mathbf{w}^T\mathbf{x}, \beta^{-1})\, p(\mathbf{w} \mid 0, \lambda^{-1})$$

- Leads to:
  - Sometimes non-Gaussian likelihoods may be suitable. E.g., laplace likelihood:
  - Useful for robust regression.

$$p(y \mid \mathbf{w}^T\mathbf{x}, \beta^{-1}) = \exp - \beta\left|y - \mathbf{w}^T\mathbf{x}\right|$$

  - Why?
  - Cost: absolute value, not squared.
    - => More robust as outliers not magnified
  - (But now only gradient solution!)



Summary

- Regression can be formalised as optimising a cost, with penalty to prevent overfitting.
  - Obscure: How to choose cost? prevent overfitting?
- Deeper probabilistic understanding:
  - Corresponds to MAP learning:
    - Likelihood defines cost.
    - Prior defines regularizer.
  - Finding what probability distribution suits your problem/data requires tells cost and regularizer.
    - Lots of choices of distributions (wikipedia ☺)

Outline

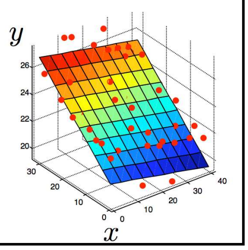
- Linear Regression
- Non-linear regression
- Overfitting and underfitting
- Regularisation and cross-validation
- Probabilistic Interpretation
- Practical Learning

Multivariate Linear Regression: Interpreting Results

- For multi-input linear regression, you can interpret w parameters
  - => Discover the importance of the factors.
- E.g.s,
  - What is the premium of each GB of memory for a PC?
  - How much does each mile you drive cost your car's value?
- w_is will have units of slopes:
- E.g., Price = w_1*rooms + w_2*meters^2+w_3*postcode
  - w_1: price per room. w_2: price per square meter.
  - w_3: how much is this postcode alone worth

$$y = \mathbf{w}^T \mathbf{x}$$

$$y = w_1 x_1 + w_2 x_2 + w_3 x_3 + \ldots$$



Multivariate Linear Regression: Interpreting Results

- For multi-input linear regression, you can interpret w parameters
  - => Discover the importance of the factors.

- Sometimes there are many potential factors, and a goal is to findout which ones influence the target?
  - Use Laplacian prior / l1 regularizer, and sparsify the weights!
  - Read off the non-zero weights.

Scalability (Big Data!)

- We learned closed form and iterative solutions
- Closed form: $\mathbf{w} = (X^T X)^{-1} X^T \mathbf{y}$
  - Matrix multiplies and inversions: O(d^2N) and O(d^3)
  - Needs all O(Nd) memory
  - Simple and convergence/learn rate issues, but not scalable CPU or memory for huge rows/dimensions
- Iterative (batch): $\mathbf{w}^{s+1} := \mathbf{w}^s + \alpha \sum_i^N \mathbf{x}_i (y_i - \mathbf{w}^T \mathbf{x}_i)$
  - O(Nd) cost per iteration (but needs multiple iterations)
  - … and tuning of alpha
  - Needs O(Nd) memory always

Scalability (Big Data!)

- Online Gradient Descent => Iterate over the dataset, following each point *i*'s gradient:

$$\mathbf{w}^{s+1} := \mathbf{w}^{s} + \alpha \mathbf{x}_i (y_i - \mathbf{w}^T \mathbf{x}_i)$$

  - Needs O(d) memory per iteration. Can learn from a stream. But no good otherwise since disk read is slow.
- Stochastic Gradient Descent (SGD):
  - Outer: Sample a random row subset D_{batch} into memory.
    - Inner:

$$\mathbf{w}^{s+1} := \mathbf{w}^{s} + \alpha \sum_{i \in D_{batch}} \mathbf{x}_i (y_i - \mathbf{w}^T \mathbf{x}_i)$$

  - Only needs $O(N^{batch}d)$ memory. +Infrequent reads.
  - Very common state of the art! (But still alpha tuning…)

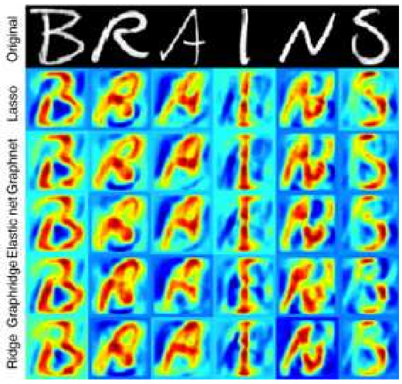
Distributed Computing (Big Data!)

- So your regression pipeline is taking 48 hours to compute…. ☹
- Key bottleneck is usually your cross-validation
  - Remember, it requires you to re-train the model for each of many possible regularizers λ.
- Easiest way to distribute on a cluster:
  - Set each compute node to train the model for one λ.
  - Instant linear speedup parallelization!
- (Within node multi-threading: Likely Out of Memory ☹)

Case Study: fMRI Brain Imaging

Linear regression from Brain Voxels => Pixels

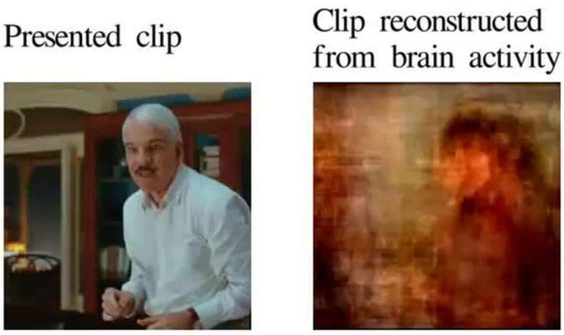
- Very good results with L1 regularization (laplacian prior), i.e. Lasso
- Best results with the a combination of L1 and second order prior.



“Linear reconstruction of perceived images from Human brain activity”, Schoenmakers et al, NeuroImage 2013

Case Study: fMRI Brain Imaging

Brain Voxel => Pixel Regression. Use L1 feature selection.



Reconstructing visual experiences from brain activity evoked by natural movies. Nishimoto et al, Curent Biology, 2011

Summary

- Linear regression parameters may be interpretable.
  - Big chunk of of “data science” roles are about running linear regressions and reading off the weights.
- Considerations for big data scalability.

Learning Outcomes

- You should:
  - Understand regression as building predictive models for continuous quantities
  - Be able to derive gradient and closed form solutions for multi-variate linear regression
  - Appreciate under and over-fitting in the context of non-linear regression
  - Be able to control under and over-fitting via regularisation and (cross)validation.
  - Appreciate the MAP interpretation of regularized regression, and the flexibility of likelihood/prior choice

LR Training Details: For Lab

- Before we said learn linear regression by

$$E(\mathbf{w}) = \frac{1}{2N}\sum_{i=1}^{N}\left(\mathbf{w}^T\mathbf{x}_i - y_i\right)^2 \Rightarrow \mathbf{w}^{s+1} := \mathbf{w}^s - \alpha\frac{1}{N}\sum_i \mathbf{x}_i(\mathbf{w}^T\mathbf{x}_i - y_i)$$

- Or regularized linear regression by

$$E(\mathbf{w}) = \frac{1}{2N}\left[\sum_{i=1}^{N}\left(\mathbf{w}^T\mathbf{x}_i - y_i\right)^2 + \lambda\mathbf{w}^T\mathbf{w}\right]$$

$$\Rightarrow \mathbf{w}^{s+1} := \mathbf{w}^s - \alpha\frac{1}{N}\left[\sum_i \mathbf{x}_i(\mathbf{w}^T\mathbf{x}_i - y_i) + \lambda\mathbf{w}\right]$$

LR Training Details: For Lab

- But remember, we may have used the "concatenate 1" trick to deal with the offset

$$E(\mathbf{w}) = \frac{1}{2N}\sum_{i=1}^{N}\left(\mathbf{w}^T\mathbf{x}_i - y_i\right)^2 \Rightarrow E(\mathbf{w}) = \frac{1}{2N}\sum_{i=1}^{N}\left((\mathbf{w}^T\mathbf{x}_i + w_0) - y_i\right)^2$$

- In this case we may only want to regularize w_k, for k>0, and not w_k for k=0.
  - Why? Because don't want to penalize a model for representing data with mean >> 0.

LR Training Details: For Lab

- How to regularize only $w_{k>0}$?

$$E(\mathbf{w}) = \frac{1}{2N}\left[\sum_{i=1}^{N}\left((\mathbf{w}^T\mathbf{x}_i + w_0) - y_i\right)^2 + \lambda \mathbf{w}^T\mathbf{w}\right]$$

$$\frac{dE(\mathbf{w})}{dw_0} = \frac{1}{N}\left[\sum_i ((\mathbf{w}^T\mathbf{x}_i + w_0) - y_i)1 + 0\right]$$

$$\frac{dE(\mathbf{w})}{dw_k} = \frac{1}{N}\left[\sum_i ((\mathbf{w}^T\mathbf{x}_i + w_0) - y_i)x_{ik} + \lambda w_k\right]$$

LR Training Details: For Lab

- So gradient updates would be

$$w_k := w_k - \alpha \frac{dE(\mathbf{w})}{dw_k}$$

$$w_0 = w_0 - \alpha \frac{1}{N}\left[\sum_i ((\mathbf{w}^T\mathbf{x}_i + w_0) - y_i)1 + 0\right]$$

$$w_k = w_k - \alpha \frac{1}{N}\left[\sum_i ((\mathbf{w}^T\mathbf{x}_i + w_0) - y_i)x_{ik} + \lambda w_k\right]$$

$$w_k = w_k(1 - \alpha \frac{\lambda}{N}) - \alpha \frac{1}{N}\sum_i ((\mathbf{w}^T\mathbf{x}_i + w_0) - y_i)x_{ik}$$