CPSC 340: Machine Learning and Data Mining

Nonlinear Regression

Fall 2020

Last Time: Linear Regression

- We discussed linear models:

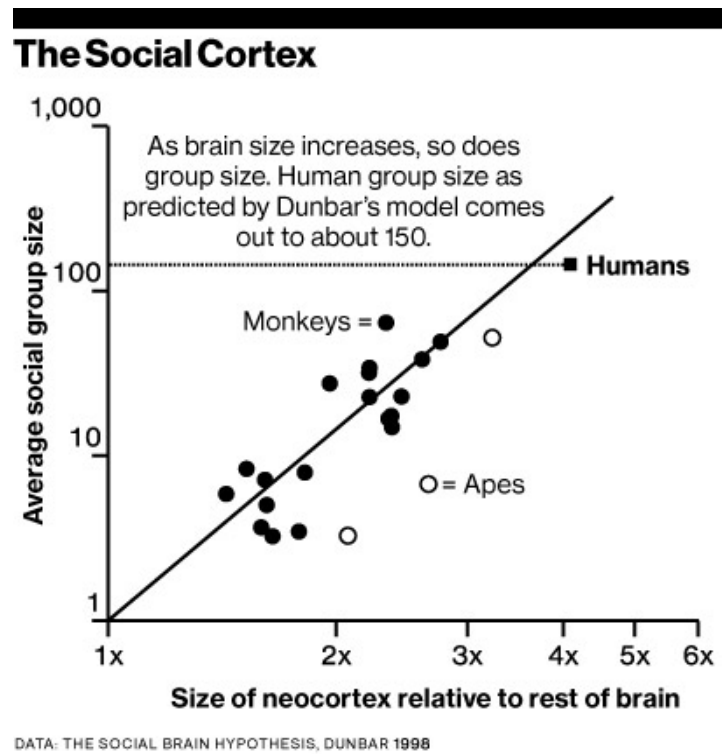
$$y_i = w_1 x_{i1} + w_2 x_{i2} + \cdots + w_d x_{id}$$
$$= \sum_{j=1}^{d} w_j x_{ij} = w^T x_i$$

- "Multiply feature x_{ij} by weight w_j, add them to get y_i".
- We discussed squared error function:

$$f(w) = \frac{1}{2}\sum_{i=1}^{n}(w^T x_i - y_i)^2$$

Predicted value — True value

- Interactive demo:
  - http://setosa.io/ev/ordinary-least-squares-regression



To predict on test case $\tilde{x}_i$
use $\hat{y}_i = w^T \tilde{x}_i$

http://www.bloomberg.com/news/articles/2013-01-10/the-dunbar-number-from-the-guru-of-social-networks

Matrix/Norm Notation (MEMORIZE/STUDY THIS)

- To solve the d-dimensional least squares, we use matrix notation:
  - We use ‘w’ as a “d times 1” vector containing weight ‘w_j’ in position ‘j’.
  - We use ‘y’ as an “n times 1” vector containing target ‘y_i’ in position ‘i’.
  - We use ‘x_i’ as a “d times 1” vector containing features ‘j’ of example ‘i’.
    - We’re now going to be careful to make sure these are column vectors.
  - So ‘X’ is a matrix with x_i^T in row ‘i’.

$$w = \begin{bmatrix} w_1 \\ w_2 \\ \vdots \\ w_d \end{bmatrix} \quad y = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix} \quad x_i = \begin{bmatrix} x_{i1} \\ x_{i2} \\ \vdots \\ x_{id} \end{bmatrix} \quad X = \begin{bmatrix} x_{11} & x_{12} & \cdots & x_{1d} \\ x_{21} & x_{22} & \cdots & x_{2d} \\ \vdots & & \ddots & \vdots \\ x_{n1} & x_{n2} & \cdots & x_{nd} \end{bmatrix} = \begin{bmatrix} \text{---} & x_1^T & \text{---} \\ \text{---} & x_2^T & \text{---} \\ & \vdots & \\ \text{---} & x_n^T & \text{---} \end{bmatrix}$$

Matrix/Norm Notation (MEMORIZE/STUDY THIS)

- To solve the d-dimensional least squares, we use matrix notation:
  - Our prediction for example 'i' is given by the scalar $w^T x_i$.
  - Our predictions for all 'i' (n times 1 vector) is the matrix-vector product Xw.

$$\hat{y}_i = w^T x_i$$

$$Xw = \underbrace{\begin{bmatrix} — x_1^T — \\ — x_2^T — \\ \vdots \\ — x_n^T — \end{bmatrix}}_{X} \underbrace{\begin{bmatrix} | \\ w \\ | \end{bmatrix}}_{w} = \begin{bmatrix} x_1^T w \\ x_2^T w \\ \vdots \\ x_n^T w \end{bmatrix} = \begin{bmatrix} \hat{y}_1 \\ \hat{y}_2 \\ \vdots \\ \hat{y}_n \end{bmatrix} = \hat{y}$$

Prediction for example 'i' in row 'i'

Also, because $w^T x_i$ is a scalar, we have $w^T x_i = x_i^T w$. (e.g., $[5]^T = [5]$)

Matrix/Norm Notation (MEMORIZE/STUDY THIS)

- To solve the d-dimensional least squares, we use matrix notation:
  - Our prediction for example 'i' is given by the scalar w^Tx_i.
  - Our predictions for all 'i' (n times 1 vector) is the matrix-vector product Xw.
  - Residual vector 'r' gives difference between predictions and y_i (n times 1).
  - Least squares can be written as the squared L2-norm of the residual.

$$f(w) = \sum_{i=1}^{n} (\underbrace{w^Tx_i - y_i}_{r_i})^2 = \sum_{i=1}^{n} (r_i)^2$$

$$= \sum_{i=1}^{n} r_i r_i$$

$$= r^Tr$$

$$= \|r\|^2 = \|Xw - y\|^2$$

$$r = \hat{y} - y = \underbrace{Xw}_{\hat{y}} - y = \begin{bmatrix} w^Tx_1 \\ w^Tx_2 \\ \vdots \\ w^Tx_n \end{bmatrix} - \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix} = \begin{bmatrix} w^Tx_1 - y_1 \\ w^Tx_2 - y_2 \\ \vdots \\ w^Tx_n - y_n \end{bmatrix}$$

r_2 is difference for example 2.

Back to Deriving Least Squares for d > 2…

- We can write vector of predictions $\hat{y}_i$ as a matrix-vector product:

$$\hat{y} = Xw = \begin{bmatrix} w^T x_1 \\ w^T x_2 \\ \vdots \\ w^T x_n \end{bmatrix}$$

- And we can write linear least squares in matrix notation as:

$$f(w) = \frac{1}{2}\|Xw - y\|^2 = \frac{1}{2}\sum_{i=1}^{n}(w^T x_i - y_i)^2$$

- We'll use this notation to derive d-dimensional least squares 'w'.
  - By setting the gradient $\nabla f(w)$ equal to the zero vector and solving for 'w'.

Digression: Matrix Algebra Review

- Quick review of linear algebra operations we'll use:
  - If 'a' and 'b' be vectors, and 'A' and 'B' be matrices then:

$$a^T b = b^T a$$

$$\|a\|^2 = a^T a$$

$$(A+B)^T = A^T + B^T$$

$$(AB)^T = B^T A^T$$

$$(A+B)(A+B) = AA + BA + AB + BB$$

$$a^T \underbrace{Ab}_{\text{vector}} = \underbrace{b^T A^T}_{\text{vector}} a$$

Sanity check:

ALWAYS CHECK THAT DIMENSIONS MATCH

(if not, you did something wrong)

Linear and Quadratic Gradients

- From these rules we have (see post-lecture slide for steps):

$$f(w) = \frac{1}{2}\sum_{i=1}^{n}(w^Tx_i - y_i)^2 = \frac{1}{2}\|Xw - y\|^2 = \frac{1}{2}w^T\underbrace{X^TX}_{\text{matrix 'A'}}w - w^T\underbrace{X^Ty}_{\text{vector 'b'}} + \underbrace{\frac{1}{2}y^Ty}_{\text{scalar 'c'}}$$

$$= \frac{1}{2}w^TAw + w^Tb + c$$

These are scalars so dimensions match.

- How do we compute gradient?

Let's first do it with $d=1$:

$$f(w) = \frac{1}{2}waw + wb + c$$

$$= \frac{1}{2}aw^2 + wb + c$$

$$f'(w) = aw + b + 0$$

Here are the generalizations to 'd' dimensions:

$\nabla[c] = 0$ (zero vector)

$\nabla[w^Tb] = b$

$\nabla[\frac{1}{2}w^TAw] = Aw$ (if A is symmetric)

Full derivations are on webpage in notes on linear and quadratic gradients.

Linear and Quadratic Gradients

- We've written as a d-dimensional quadratic:

$$f(w) = \frac{1}{2}\sum_{i=1}^{n}(w^T x_i - y_i)^2 = \frac{1}{2}\|Xw - y\|^2 = \frac{1}{2} w^T \underbrace{X^T X}_{\text{matrix 'A'}} w - w^T \underbrace{X^T y}_{\text{vector 'b'}} + \underbrace{\frac{1}{2} y^T y}_{\text{scalar 'c'}}$$

$$= \frac{1}{2} w^T A w + w^T b + c$$

- Gradient is given by: $\nabla f(w) = Aw - b + 0$

- Using definitions of 'A' and 'b': $= \underbrace{X^T X w - X^T y}_{\text{Sanity check: all dimensions match}}$

$$(d \times n)(n \times d)(d \times 1) - (d \times n)(n \times 1)$$

Normal Equations

- Set gradient equal to zero to find the "critical" points:

$$X^T X w - X^T y = 0$$

- We now move terms not involving 'w' to the other side:

$$X^T X w = X^T y$$

- This is a set of 'd' linear equations called the normal equations.
  - This a linear system like "Ax = b" from Math 152.
    - You can use Gaussian elimination to solve for 'w'.
  - In Julia, the "\" command can be used to solve linear systems:

Train: w = (X'X)\(X'y)   Predict: yhat = X_{test} * w

Normal Equations

- Set gradient equal to zero to find the "critical" points:

$$X^T X w - X^T y = 0$$

- We now move terms not involving 'w' to the other side:

$$X^T X w = X^T y$$

- This is a set of 'd' linear equations called the "normal equations".
  - This a linear system like "Ax = b" from Math 152.
    - You can use Gaussian elimination to solve for 'w'.
  - In Python, you solve linear systems in 1 line using numpy.linalg.solve.

Incorrect Solutions to Least Squares Problem

The least squares objective is $f(w) = \frac{1}{2}\|Xw - y\|^2$

The minimizers of this objective are solutions to the linear system:

$$X^TXw = X^Ty$$

The following are not the solutions to the least squares problem:

$w = (X^TX)^{-1}(X^Ty)$ (only true if X^TX is invertible)

$wX^TX = X^Ty$ (matrix multiplication is not commutative, dimensions don't even match)

$w = \frac{X^Ty}{X^TX}$ (you cannot divide by a matrix)

Least Squares Cost

- Cost of solving "normal equations" $X^TXw = X^Ty$?
- Forming X^Ty vector costs $O(nd)$.
  - It has 'd' elements, and each is an inner product between 'n' numbers.
- Forming matrix X^TX costs $O(nd^2)$.
  - It has d^2 elements, and each is an inner product between 'n' numbers.
- Solving a d x d system of equations costs $O(d^3)$.
  - Cost of Gaussian elimination on a d-variable linear system.
  - Other standard methods have the same cost.
- Overall cost is $O(nd^2 + d^3)$.
  - Which term dominates depends on 'n' and 'd'.

Least Squares Issues

- Issues with least squares model:
  - Solution might not be unique.
  - It is sensitive to outliers.
  - It always uses all features.
  - Data can might so big we can't store X^TX.
    - Or you can't afford the $O(nd^2 + d^3)$ cost.
  - It might predict outside range of y_i values.
  - It assumes a linear relationship between x_i and y_i.

X is $n \times d$
so X^T is $d \times n$
and X^TX is $d \times d$.

Non-Uniqueness of Least Squares Solution

- Why isn't solution unique?
  - Imagine having two features that are identical for all examples.
  - I can increase weight on one feature, and decrease it on the other, without changing predictions.

$$\hat{y}_i = w_1 x_{i1} + w_2 \underbrace{x_{i1}}_{\text{copy}} = (w_1 + w_2) x_{i1} + 0 x_{i1}$$

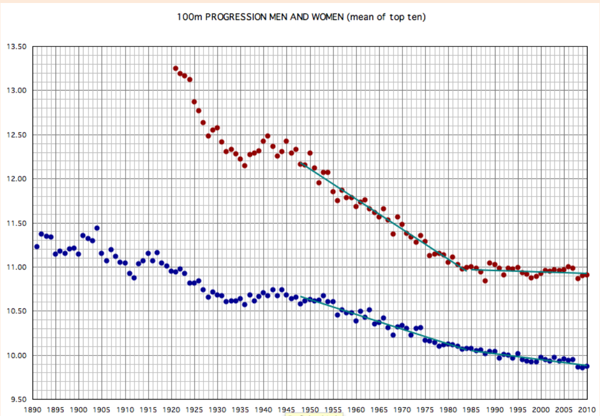
  - Thus, if (w_1,w_2) is a solution then $(w_1+w_2, 0)$ is another solution.
  - This is special case of features being "collinear":
    - One feature is a linear function of the others.

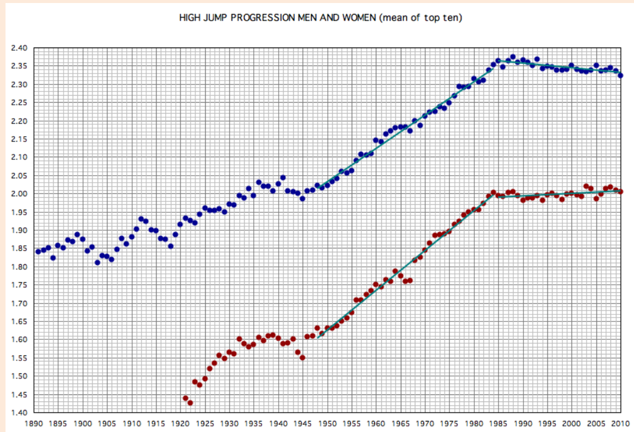
- But, any 'w' where ∇ f(w) = 0 is a global minimizer of 'f'.
  - This is due to convexity of 'f', which we'll discuss later.

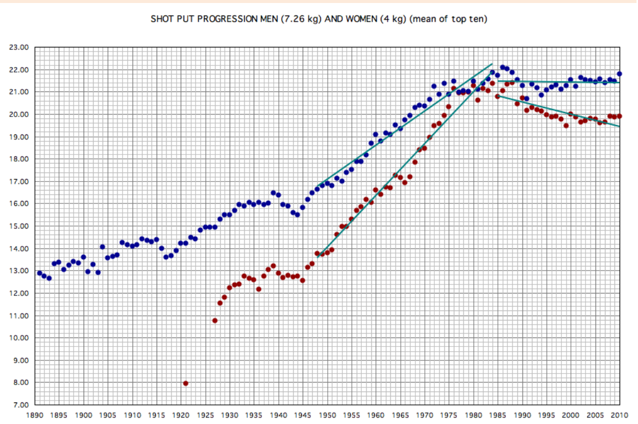
(pause)

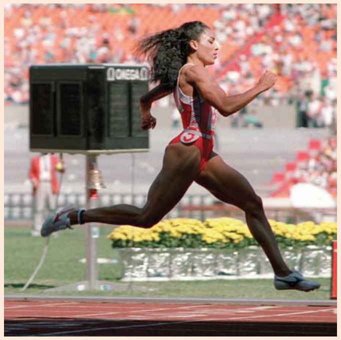
Motivation: Non-Linear Progressions in Athletics

- Are top athletes going faster, higher, and farther?









http://www.at-a-lanta.nl/weia/Progressie.html
https://en.wikipedia.org/wiki/Usain_Bolt
http://www.britannica.com/biography/Florence-Griffith-Joyner

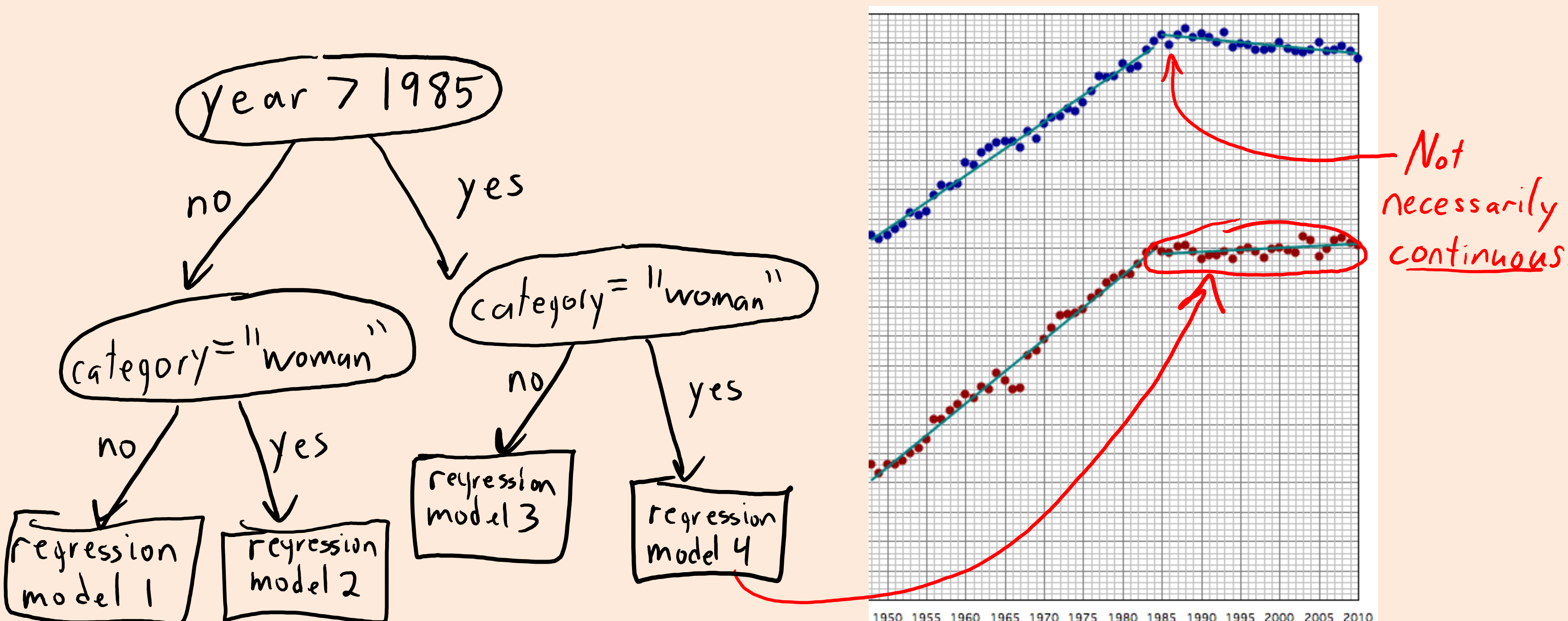
Adapting Counting/Distance-Based Methods

- We can adapt our classification methods to perform regression:

http://www.at-a-lanta.nl/weia/Progressie.html

Adapting Counting/Distance-Based Methods

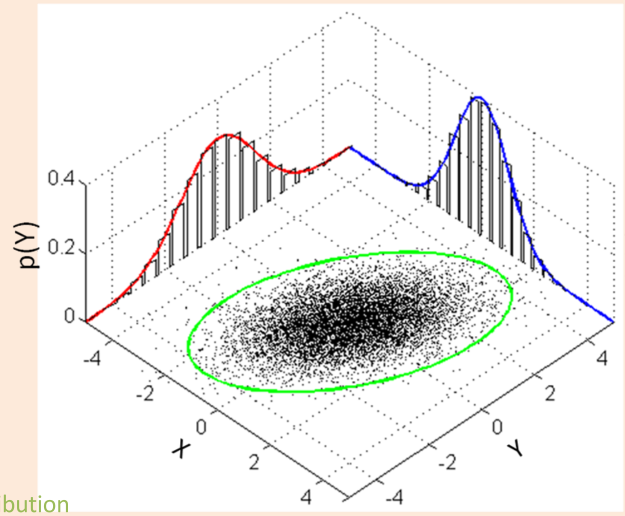
- We can adapt our classification methods to perform regression:
  - Regression tree: tree with mean value or linear regression at leaves.



http://www.at-a-lanta.nl/weia/Progressie.html

Adapting Counting/Distance-Based Methods

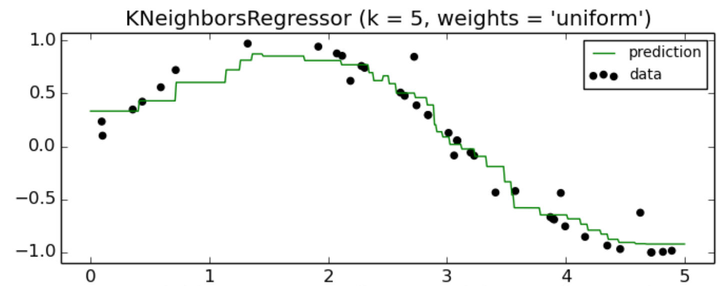
- We can adapt our classification methods to perform regression:
  - Regression tree: tree with mean value or linear regression at leaves.
  - Probabilistic models: fit $p(x_i \mid y_i)$ and $p(y_i)$ with Gaussian or other model.
    - CPSC 540.



https://en.wikipedia.org/wiki/Multivariate_normal_distribution

Adapting Counting/Distance-Based Methods

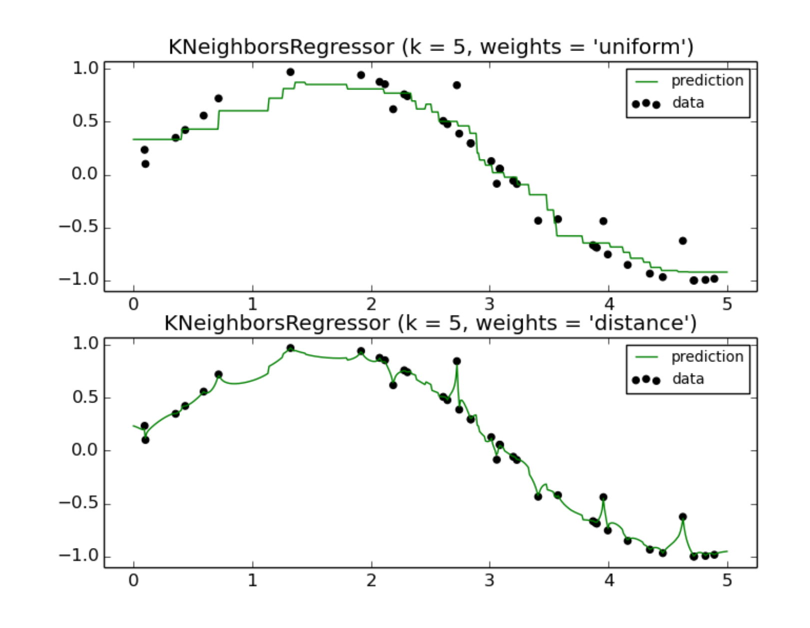
- We can adapt our classification methods to perform regression:
  - Regression tree: tree with mean value or linear regression at leaves.
  - Probabilistic models: fit $p(x_i \mid y_i)$ and $p(y_i)$ with Gaussian or other model.
  - Non-parametric models:
    - KNN regression:
      - Find 'k' nearest neighbours of $\tilde{x}_i$.
      - Return the mean of the corresponding y_i.



http://scikit-learn.org/stable/modules/neighbors.html

Adapting Counting/Distance-Based Methods

- We can adapt our classification methods to perform regression:
  - Regression tree: tree with mean value or linear regression at leaves.
  - Probabilistic models: fit $p(x_i \mid y_i)$ and $p(y_i)$ with Gaussian or other model.
  - Non-parametric models:
    - KNN regression.
    - Could be weighted by distance.
      - Close points 'j' get more "weight" w_{ij}.

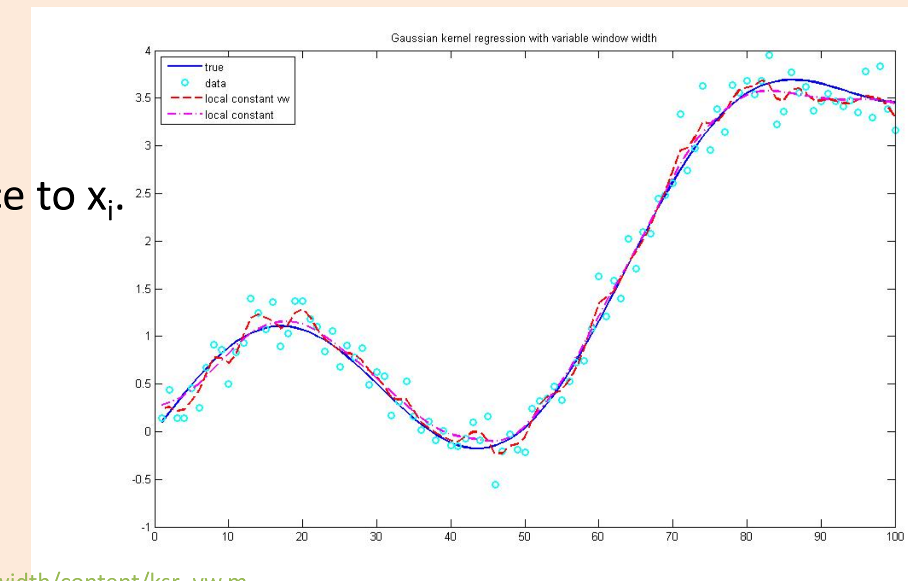


http://scikit-learn.org/stable/modules/neighbors.html

Adapting Counting/Distance-Based Methods

- We can adapt our classification methods to perform regression:
  - Regression tree: tree with mean value or linear regression at leaves.
  - Probabilistic models: fit $p(x_i \mid y_i)$ and $p(y_i)$ with Gaussian or other model.
  - Non-parametric models:
    - KNN regression.
    - Could be weighted by distance.
    - 'Nadaraya-Waston': weight *all* y_i by distance to x_i.

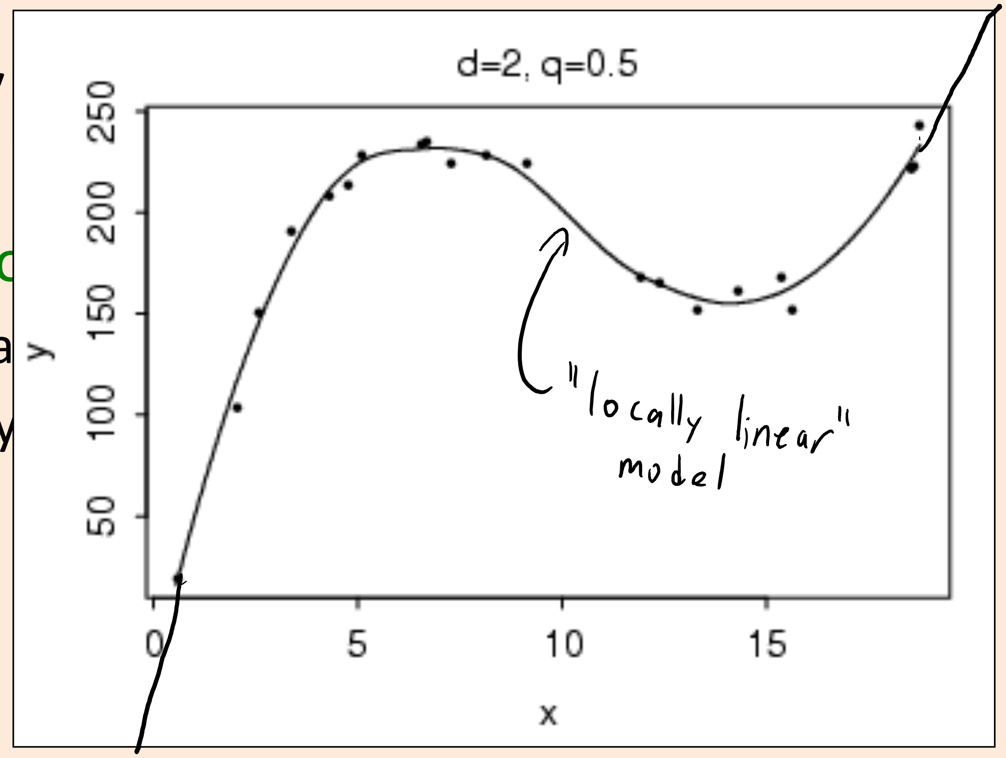
$$\hat{y}_i = \frac{\sum_{j=1}^{n} v_{ij} y_j}{\sum_{j=1}^{n} v_{ij}}$$



http://www.mathworks.com/matlabcentral/fileexchange/35316-kernel-regression-with-variable-window-width/content/ksr_vw.m

Adapting Counting/

- We can adapt our classificatio
  - Regression tree: tree with mea
  - Probabilistic models: fit p(x_i | y
  - Non-parametric models:
    - KNN regression.
    - Could be weighted by distance.
    - 'Nadaraya-Waston': weight *all* y_i
    - 'Locally linear regression': for each x_i, fit a linear model weighted by distance.
      (Better than KNN and NW at boundaries.)



http://www.itl.nist.gov/div898/handbook/pmd/section4/pmd423.htm

Adapting Counting/Distance-Based Methods

- We can adapt our classification methods to perform regression:
  - Regression tree: tree with mean value or linear regression at leaves.
  - Probabilistic models: fit $p(x_i \mid y_i)$ and $p(y_i)$ with Gaussian or other model.
  - Non-parametric models:
    - KNN regression.
    - Could be weighted by distance.
    - 'Nadaraya-Waston': weight *all* y_i by distance to x_i.
    - 'Locally linear regression': for each x_i, fit a linear model weighted by distance.
      (Better than KNN and NW at boundaries.)
  - Ensemble methods:
    - Can improve performance by averaging across regression models.

Adapting Counting/Distance-Based Methods

- We can adapt our classification methods to perform regression.

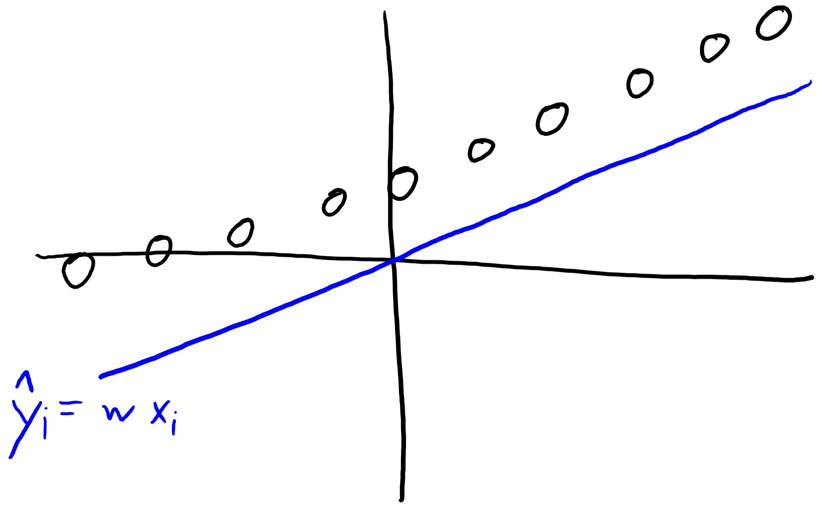
- Applications:
  - Regression forests for fluid simulation:
    - https://www.youtube.com/watch?v=kGB7Wd9CudA
  - KNN for image completion:
    - http://graphics.cs.cmu.edu/projects/scene-completion
    - Combined with “graph cuts” and “Poisson blending”.
  - KNN regression for “voice photoshop”:
    - https://www.youtube.com/watch?v=I3l4XLZ59iw
    - Combined with “dynamic time warping” and “Poisson blending”.

- But we’ll focus on linear models with non-linear transforms.
  - These are the building blocks for more advanced methods.

http://www.itl.nist.gov/div898/handbook/pmd/section4/pmd423.htm

Why don't we have a y-intercept?

– Linear model is $\hat{y}_i = wx_i$ instead of $\hat{y}_i = wx_i + w_0$ with y-intercept w_0.
– Without an intercept, if $x_i = 0$ then we must predict $\hat{y}_i = 0$.



Even "least squares" solution must go through origin.

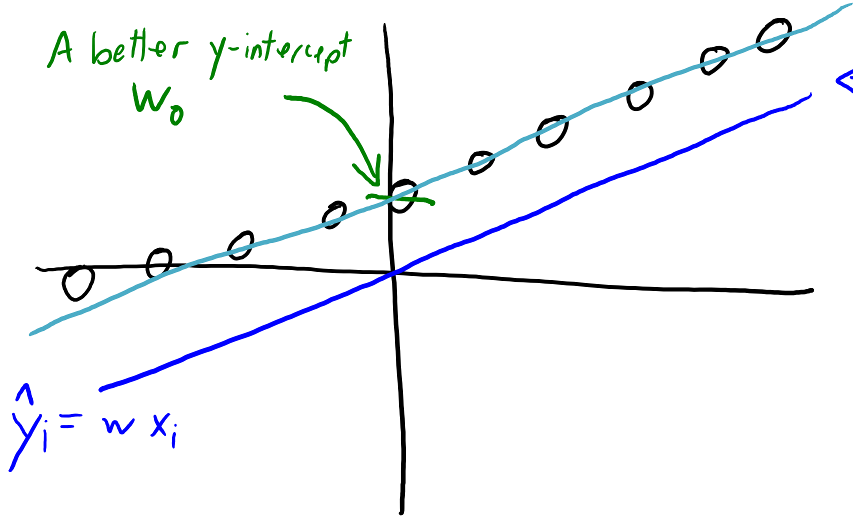
Why don't we have a y-intercept?

- Linear model is $\hat{y}_i = wx_i$ instead of $\hat{y}_i = wx_i + w_0$ with y-intercept w_0.
- Without an intercept, if $x_i = 0$ then we must predict $\hat{y}_i = 0$.

Adding y-intercept fixes this.

$\hat{y}_i = w^T x_i + w_0$

Even "least squares" solution must go through origin.



Adding a Bias Variable

- Simple trick to add a y-intercept ("bias") variable:
  - Make a new matrix "Z" with an extra feature that is always "1".

$$X = \begin{bmatrix} -0.1 \\ 0.3 \\ 0.2 \end{bmatrix} \qquad Z = \begin{bmatrix} \underbrace{\begin{matrix} 1 \\ 1 \\ 1 \end{matrix}}_{\text{"always 1"}} & \underbrace{\begin{matrix} -0.1 \\ 0.3 \\ 0.2 \end{matrix}}_{X} \end{bmatrix}$$

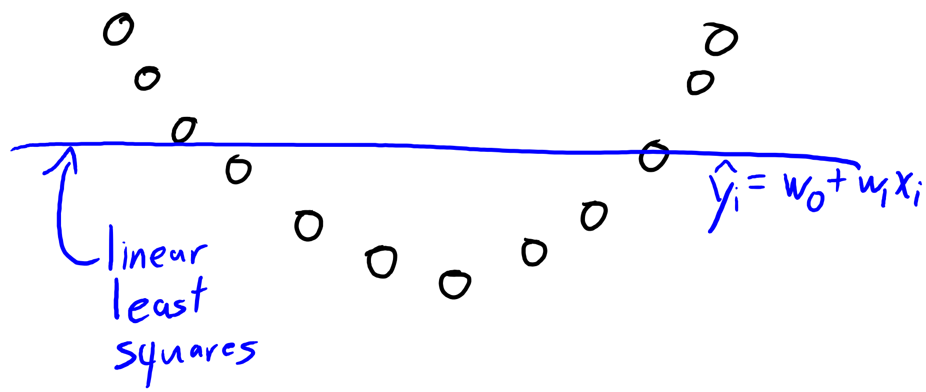
- Now use "Z" as your features in linear regression.
  - We'll use 'v' instead of 'w' as regression weights when we use features 'Z'.

$$\hat{y}_i = \underset{\downarrow\, w_0}{v_1}\, \underset{\downarrow\, 1}{z_{i1}} + \underset{\downarrow\, w_1}{v_2}\, \underset{\downarrow\, x_{i1}}{z_{i2}} = w_0 + w_1 x_{i1}$$

- So we can have a non-zero y-intercept by changing features.
  - This means we can ignore the y-intercept in our derivations, which is cleaner.

Motivation: Limitations of Linear Models

- On many datasets, y_i is not a linear function of x_i.



- Can we use least square to fit non-linear models?

Non-Linear Feature Transforms

- Can we use linear least squares to fit a quadratic model?

$$\hat{y}_i = w_0 + w_1 x_i + w_2 x_i^2$$

- You can do this by changing the features (change of basis):

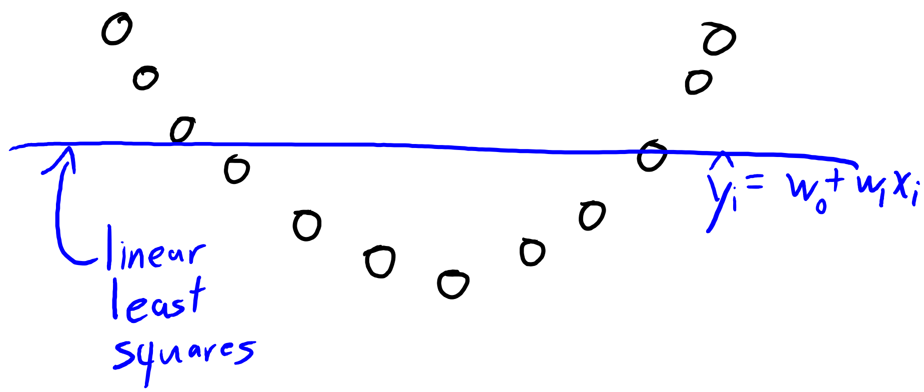
$$X = \begin{bmatrix} 0.2 \\ -0.5 \\ 1 \\ 4 \end{bmatrix} \qquad Z = \begin{bmatrix} 1 & 0.2 & (0.2)^2 \\ 1 & -0.5 & (-0.5)^2 \\ 1 & 1 & (1)^2 \\ 1 & 4 & (4)^2 \end{bmatrix}$$

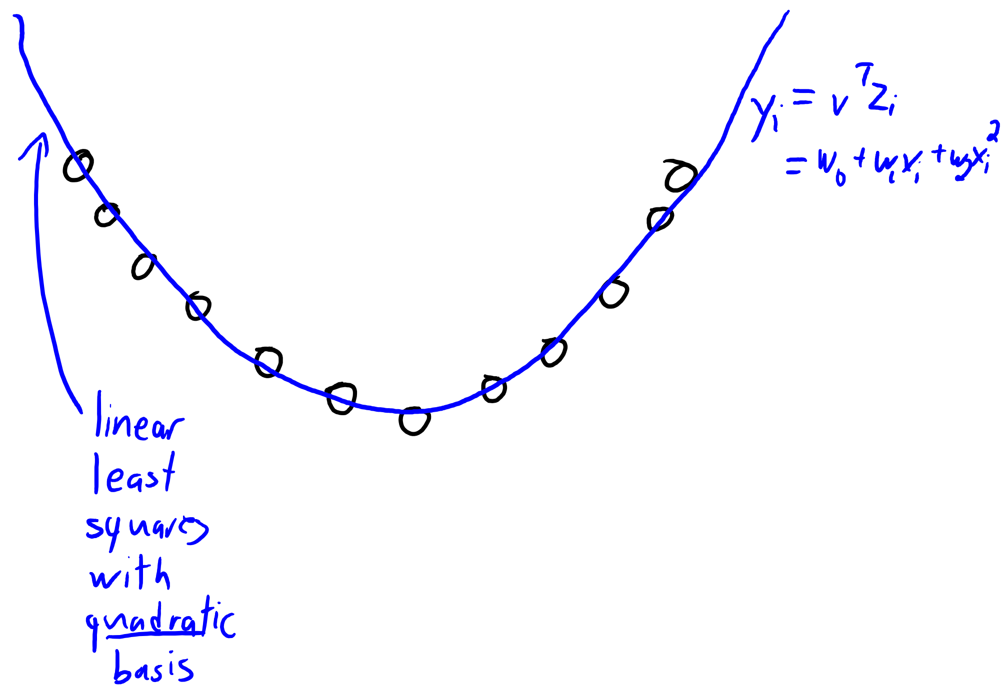
$$\begin{array}{ccc} \text{y-int} & x & x^2 \end{array}$$

- Fit new parameters 'v' under "change of basis": solve $Z^TZv = Z^Ty$.
- It's a linear function of w, but a quadratic function of x_i.

$$\hat{y}_i = v^T z_i = \underbrace{v_1}_{w_0} \underbrace{z_{i1}}_{1} + \underbrace{v_2}_{w_1} \underbrace{z_{i2}}_{x_i} + \underbrace{v_3}_{w_2} \underbrace{z_{i3}}_{x_i^2}$$

Non-Linear Feature Transforms





To predict on new data $\tilde{X}$, form $\tilde{Z}$ from $\tilde{X}$ and take $y = \tilde{Z}v$

General Polynomial Features (d=1)

- We can have a polynomial of degree 'p' by using these features:

$$Z = \begin{bmatrix} 1 & x_1 & (x_1)^2 & \cdots & (x_1)^p \\ 1 & x_2 & (x_2)^2 & \cdots & (x_2)^p \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & x_n & (x_n)^2 & \cdots & (x_n)^p \end{bmatrix}$$

- There are polynomial basis functions that are numerically nicer:
  - E.g., Lagrange polynomials (see CPSC 303).

Summary

- Matrix notation for expressing least squares problem.
- Normal equations: solution of least squares as a linear system.
  - Solve $(X^TX)w = (X^Ty)$.
- Solution might not be unique because of collinearity.
  - But any solution is optimal because of "convexity".
- Tree/probabilistic/non-parametric/ensemble regression methods.
- Non-linear transforms:
  - Allow us to model non-linear relationships with linear models.

- Next time: how to do least squares with a million features.

Linear Least Squares: Expansion Step

Want 'w' that minimizes

$$f(w) = \frac{1}{2}\sum_{i=1}^{n}(w^Tx_i - y_i)^2 = \underbrace{\frac{1}{2}\|Xw - y\|_2^2}_{\text{Let's expand then compute gradient.}} = \frac{1}{2}(Xw-y)^T(Xw-y)$$

$$= \frac{1}{2}((Xw)^T - y^T)(Xw-y)$$

$$= \frac{1}{2}(w^TX^T - y^T)(Xw-y)$$

$$= \frac{1}{2}(w^TX^T(Xw-y) - y^T(Xw-y))$$

$$= \frac{1}{2}(w^TX^TXw - w^TX^Ty - y^TXw + y^Ty)$$

$$= \underbrace{\frac{1}{2}w^TX^TXw - w^TX^Ty + \frac{1}{2}y^Ty}_{\text{Sanity check: all of these are scalars.}}$$

Rule:

$\|a\|^2 = a^Ta$

$(A+B^T) = (A^T + B^T)$

$(AB)^T = B^TA^T$

$(A+B)C = AC+BC$

$A(B+C) = AB+BC$

$a^T\underbrace{Ab}_{\text{vector}} = \underbrace{b^TA^T}_{\text{vector}}a$

Vector View of Least Squares

- We showed that least squares minimizes:

$$f(w) = \frac{1}{2}\|Xw - y\|^2$$

- The ½ and the squaring don't change solution, so equivalent to:

$$f(w) = \|Xw - y\|$$

- From this viewpoint, least square minimizes Euclidean distance between vector of labels 'y' and vector of predictions Xw.

Bonus Slide: Householder(-ish) Notation

- Househoulder notation: set of (fairly-logical) conventions for math.

Use greek letters for scalars: $\alpha = 1$, $\beta = 3.5$, $\gamma = \pi$

Use first/last lowercase letters for vectors: $w = \begin{bmatrix} 0.1 \\ 0.2 \end{bmatrix}$, $x = \begin{bmatrix} 0 \\ 1 \end{bmatrix}$, $y = \begin{bmatrix} 2 \\ -1 \end{bmatrix}$, $a = \begin{bmatrix} 1 \\ 0 \\ 1 \end{bmatrix}$, $b = \begin{bmatrix} 0.5 \\ 0.5 \end{bmatrix}$

Assumed to be column-vectors.

Use first/last uppercase letters for matrices: X, Y, W, A, B

Indices use i, j, k.

Sizes use m, n, d, p, and k

hopefully meaning of 'k' is obvious from context

Sets use S, T, U, V

Functions use f, g, and h.

When I write x_i I mean "grab row 'i' of X and make a column-vector with its values."

Bonus Slide: Householder(-ish) Notation

- Househoulder notation: set of (fairly-logical) conventions for math:

Our ultimate least squares notation:

$$f(w) = \frac{1}{2}\|Xw - y\|^2$$

But if we agree on notation we can quickly understand:

$$g(x) = \frac{1}{2}\|Ax - b\|^2$$

If we use random notation we get things like:

$$H(\beta) = \frac{1}{2}\|R\beta - P_n\|^2$$

Is this the same model?

When does least squares have a unique solution?

- We said that least squares solution is not unique if we have repeated columns.
- But there are other ways it could be non-unique:
  - One column is a scaled version of another column.
  - One column could be the sum of 2 other columns.
  - One column could be three times one column minus four times another.
- Least squares solution is unique if and only if all columns of X are "linearly independent".
  - No column can be written as a "linear combination" of the others.
  - Many equivalent conditions (see Strang's linear algebra book):
    - X has "full column rank", X^TX is invertible, X^TX has non-zero eigenvalues, $\det(X^TX) > 0$.
  - Note that we cannot have independent columns if $d > n$.