CPSC 340:
Machine Learning and Data Mining

Nonlinear Regression

Original version of these slides by Mark Schmidt, with modifications by Mike Gelbart.

Admin

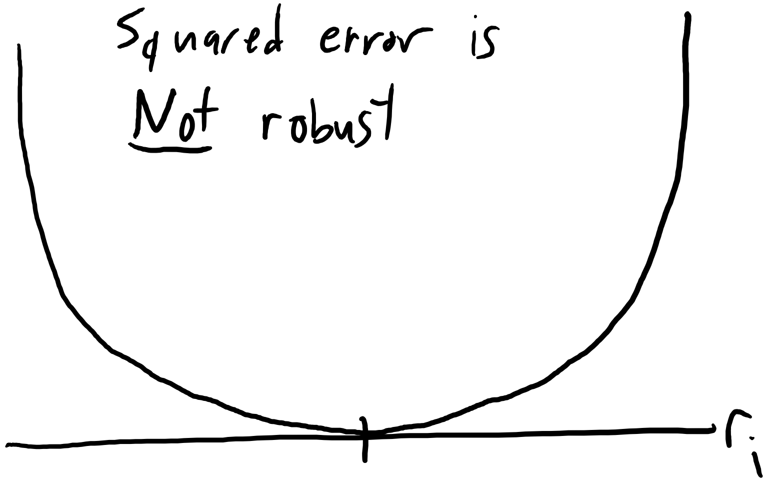
- Midterm on Feb 14 in class.
  - Previous midterms on course homepage.
  - Covers lecture 1-14 (i.e., up to & including today), assignments 1-3.

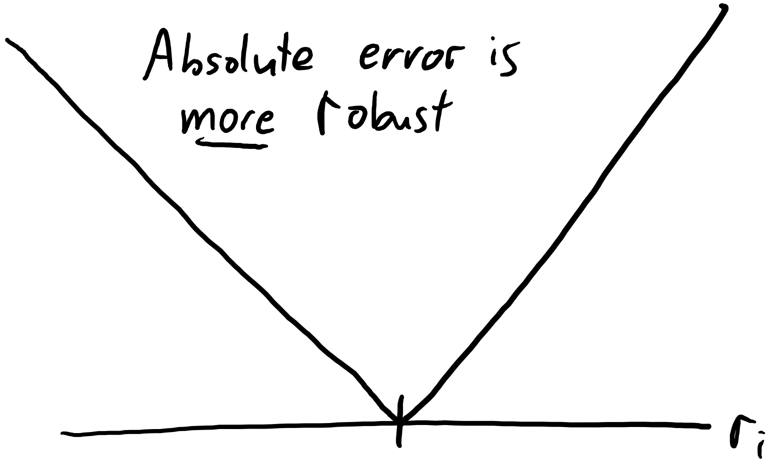
- Note to self: we need to finite the previous lecture before starting.

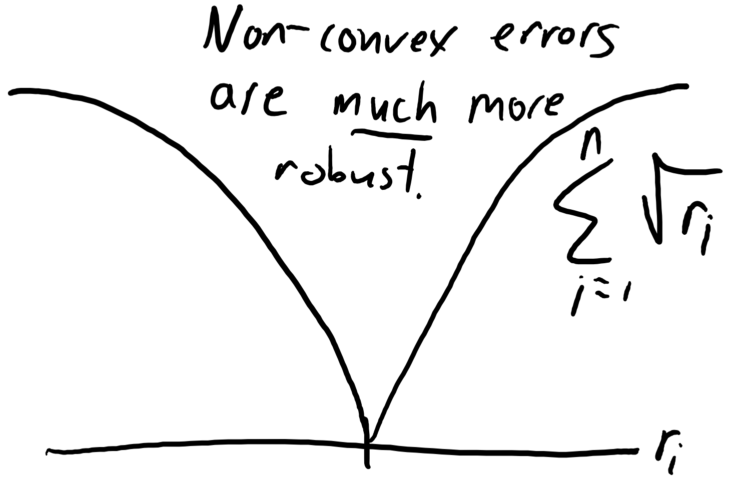
Summary of Last Lecture

1. Error functions:
   - Squared error is sensitive to outliers.
   - Absolute (L_1) error and Huber error are more robust to outliers.
   - Brittle (L_∞) error is more sensitive to outliers.
2. L_1 and L_∞ error functions are convex but non-differentiable:
   - Finding 'w' that minimizes these errors is harder than squared error.
3. We can approximate these with convex differentiable functions:
   - L_1 can be approximated with Huber.
   - L_∞ can be approximated with log-sum-exp.
4. Gradient descent finds stationary point of differentiable function.
   - "Stationary point" == "critical point" == "a value of 'w' where $\nabla f(w) = 0$".
5. For convex functions, any stationary point is a global minimum.
   - So gradient descent finds global minimum.

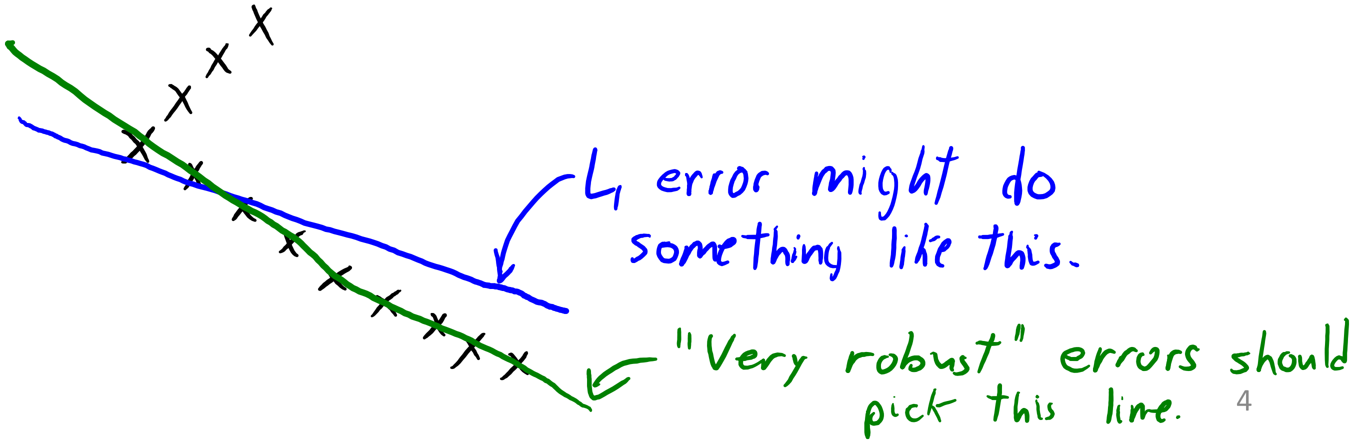
Very Robust Regression



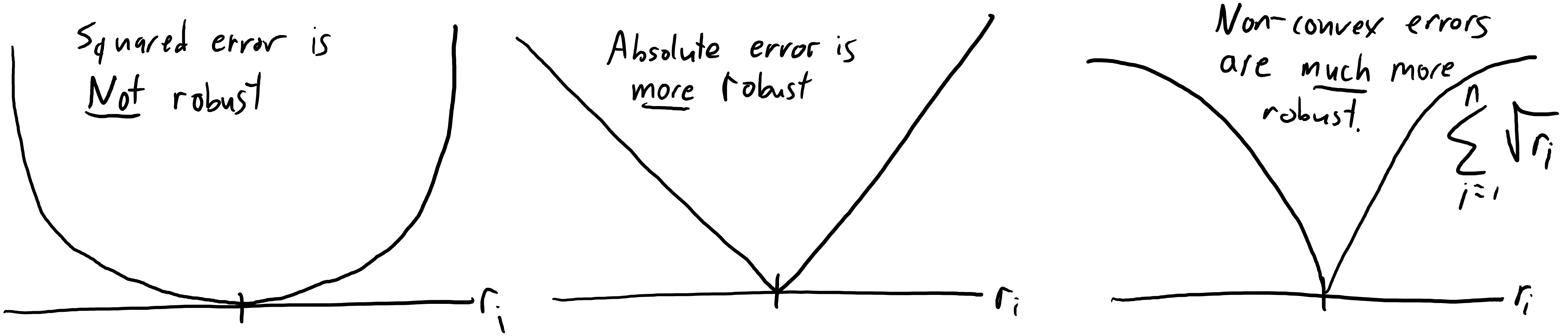




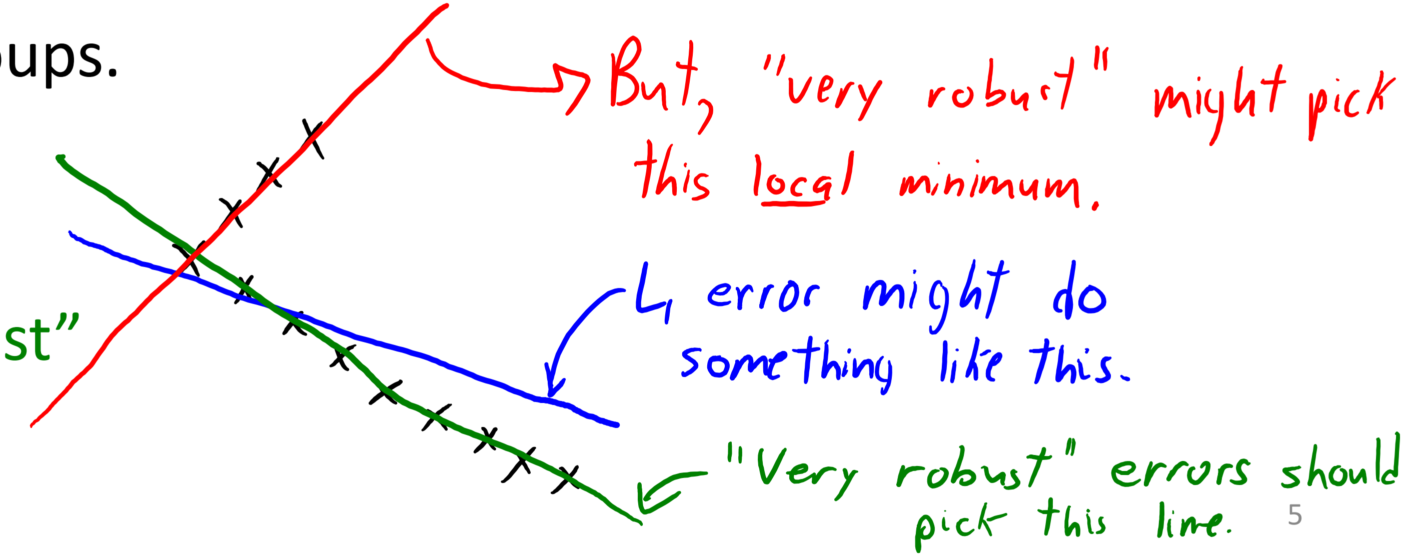
- Non-convex errors can be very robust:
  - Not influenced by outlier groups.



Very Robust Regression



- Non-convex errors can be very robust:
  - Not influenced by outlier groups.
  - But non-convex, so finding global minimum is hard.
  - Absolute value is "most robust" convex loss function.



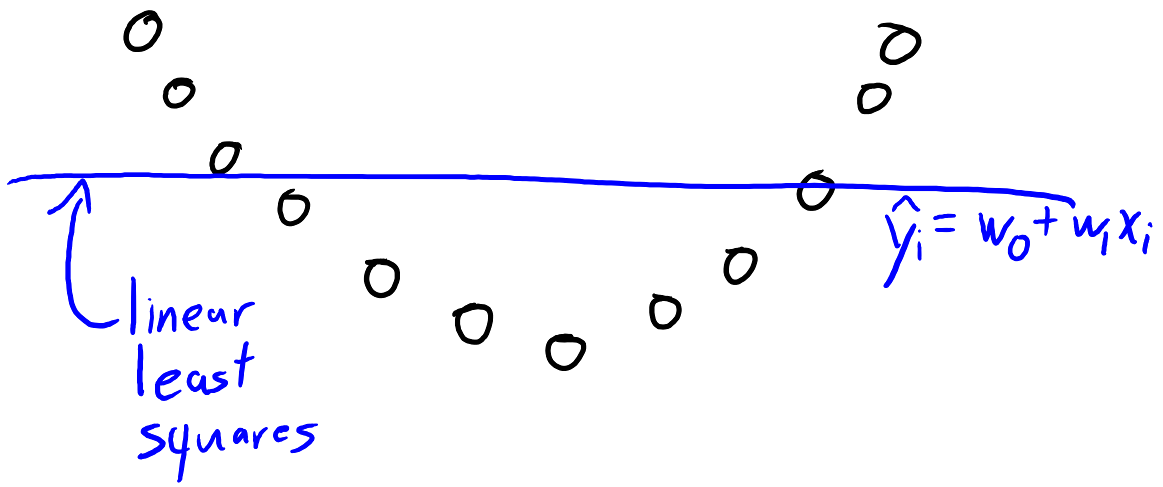
(pause)

Nonlinear regression

- We can adapt classification methods to perform regression.
- E.g. decision tree regression, KNN regression.
- See bonus slides for more details.
- We will focus on direct extensions of linear regression.

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?

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

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

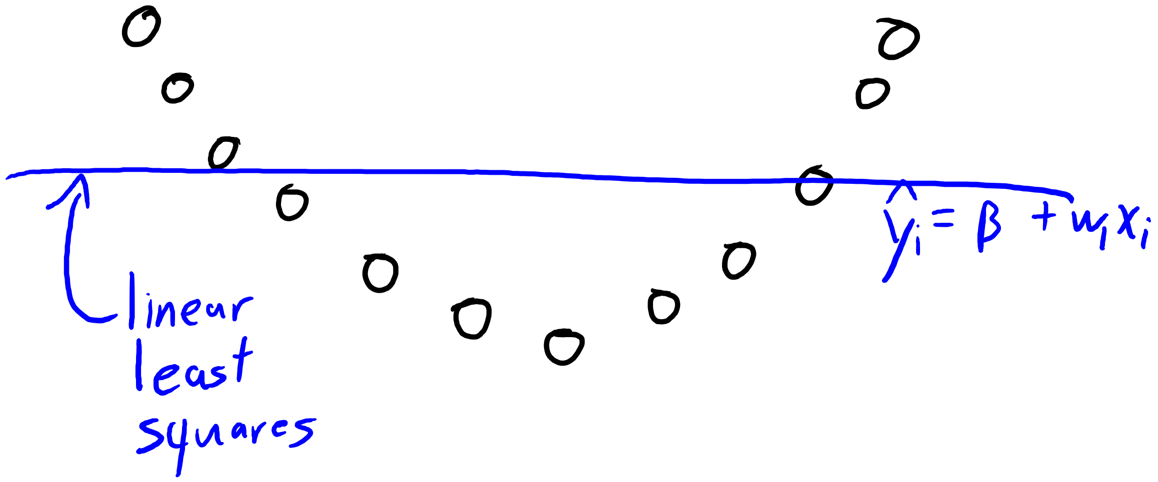
$$X = \begin{bmatrix} 0.2 \\ -0.5 \\ 1 \\ 4 \end{bmatrix} \qquad Z = \begin{array}{c} \begin{bmatrix} 1 & 0.2 & (0.2)^2 \\ 1 & -0.5 & (-0.5)^2 \\ 1 & 1 & (1)^2 \\ 1 & 4 & (4)^2 \end{bmatrix} \\ \begin{array}{ccc} \text{y-int} & x & x^2 \end{array} \end{array}$$

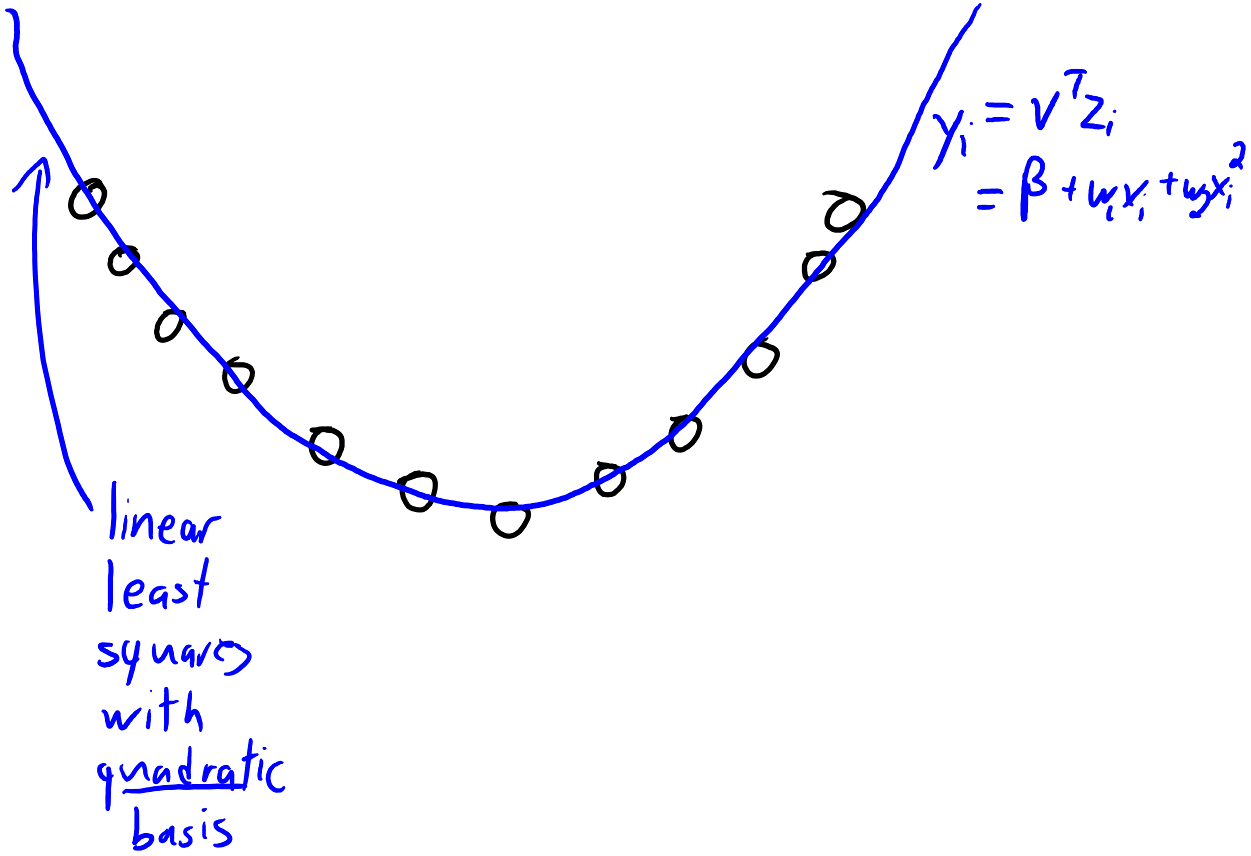
$$\begin{aligned} \hat{y}_i &= v^T z_i \\ &= v_1 z_{i0} + v_2 z_{i1} + v_3 z_{i2} \\ &= \beta + w_1 x_i + w_2 x_i^2 \end{aligned}$$

- It's a linear function of w, but a quadratic function of x_i.
- Fit using normal equations with Z instead of X: $v = (Z^TZ)^{-1}(Z^Ty)$

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

Non-Linear Feature Transforms





General Polynomial Features (d=1)

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

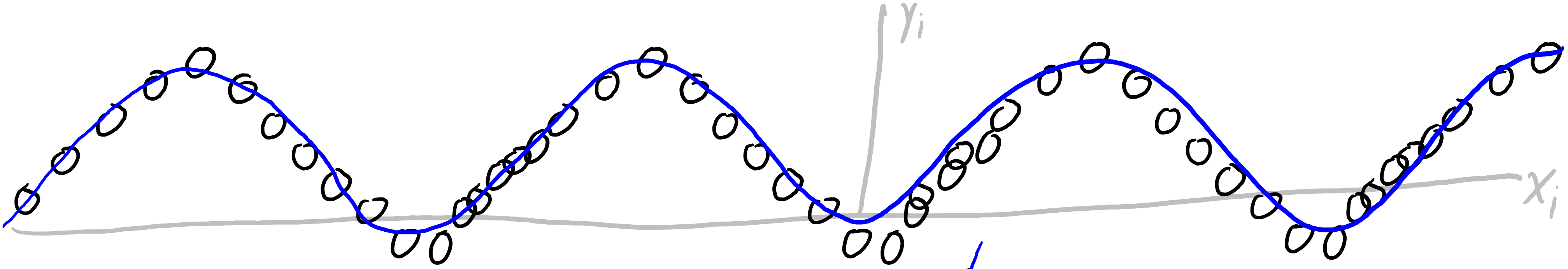
$$Z = \begin{bmatrix} 1 & x_1 & (x_1)^2 & \cdots & (x_1)^p \\ 1 & x_2 & (x_2)^2 & \cdots & (x_2)^p \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & x_n & (x_n)^2 & \cdots & (x_n)^p \end{bmatrix}$$

- There are polynomial basis functions that are numerically nicer:
  - E.g., Lagrange polynomials (see CPSC 303).
- If you have more than one feature, you include interactions:
  - With p=2, in addition to $(x_{i1})^2$ and $(x_{i2})^2$ you would include $x_{i1}x_{i2}$.

Jupyter notebook demo

Beyond Polynomial Transformations

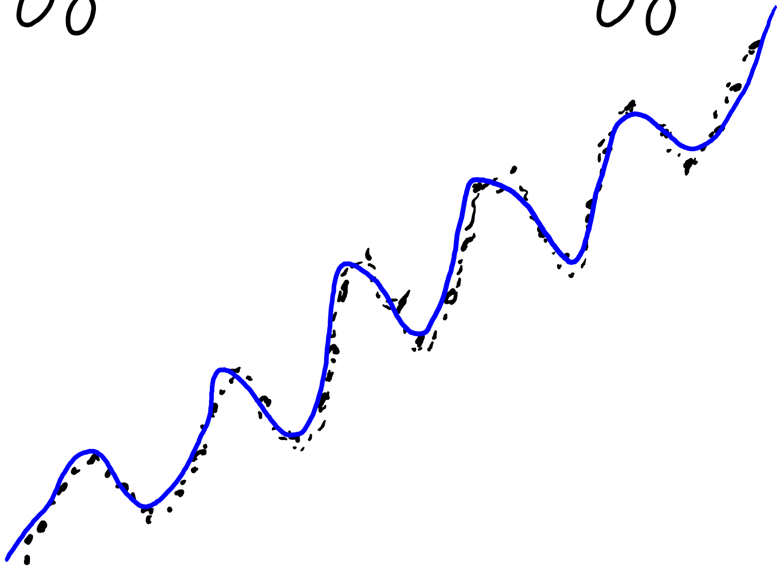
- Polynomials are not the only possible transformation:
  - Exponentials, logarithms, trigonometric functions, etc.
  - The right non-linear transform will vastly improve performance.



For periodic data we might use

$$Z = \begin{bmatrix} \sin(x_1) \\ \sin(x_2) \\ \vdots \\ \sin(x_n) \end{bmatrix}$$

$$\hat{y}_i = v^T z_i = w_1 \sin(x_i)$$

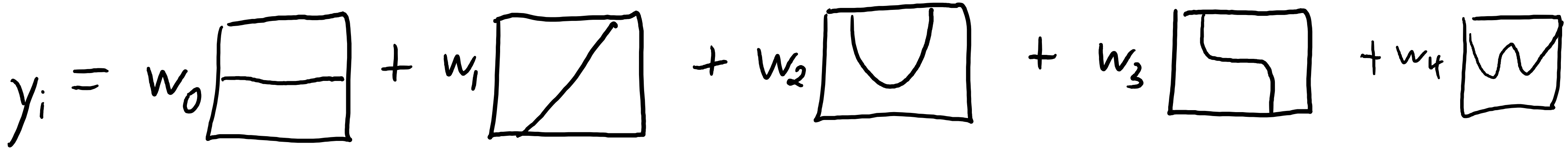


You can have different types of bases

$$Z = \begin{bmatrix} x_1 & \sin(6x_1) \\ x_2 & \sin(6x_2) \\ \vdots & \vdots \\ \underbrace{x_n}_{\text{linear}} & \underbrace{\sin(6x_n)}_{\text{periodic}} \end{bmatrix}$$

Parametric vs. Non-Parametric Transforms

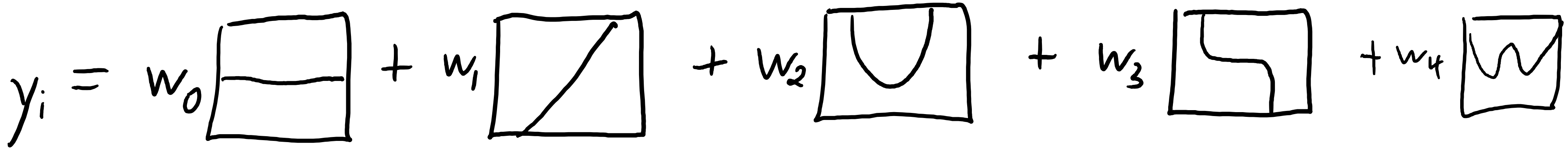
- We've been using linear models with polynomial bases:



- But polynomials are not the only possible bases:
  - Exponentials, logarithms, trigonometric functions, etc.
  - The right basis will vastly improve performance.
  - If we use the wrong basis, our accuracy is limited even with lots of data.
  - But the right basis may not be obvious.

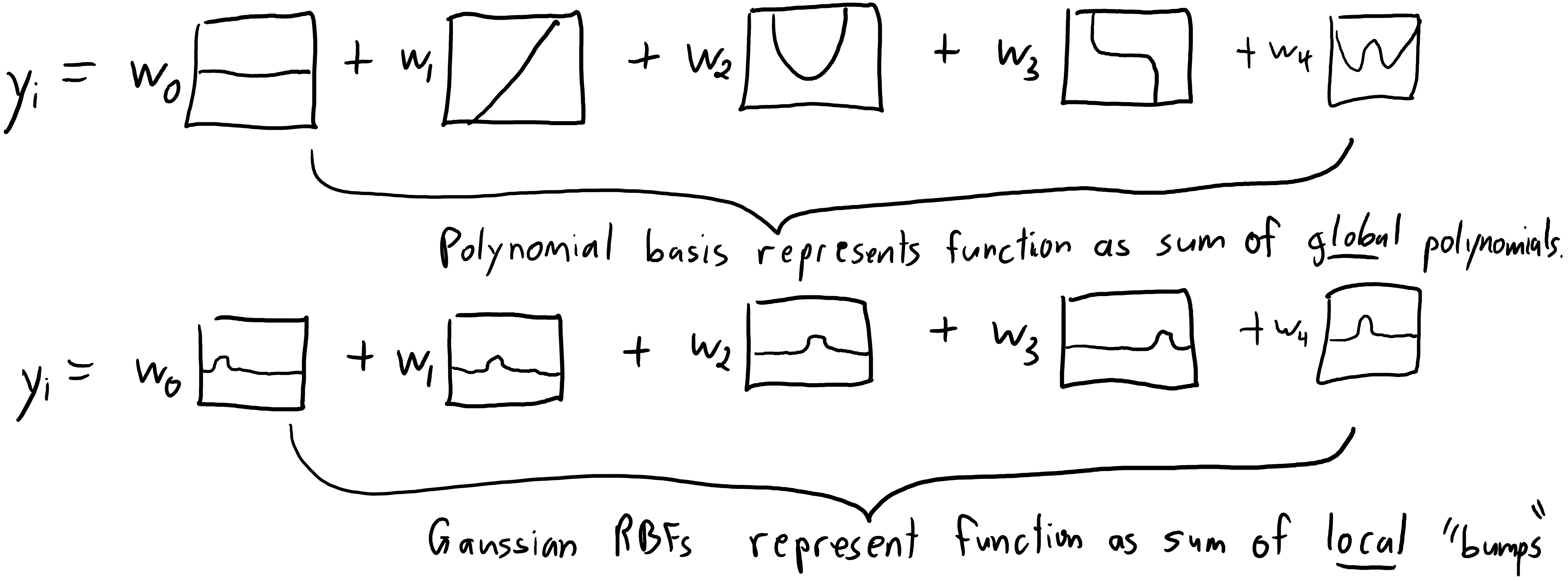
Parametric vs. Non-Parametric Transforms

- We've been using linear models with polynomial bases:



- Alternative is non-parametric bases:
  - Size of basis (number of features) grows with 'n'.
  - Model gets more complicated as you get more data.
  - Can model complicated functions where you don't know the right basis.
    - With enough data.
  - Classic example is "Gaussian RBFs".

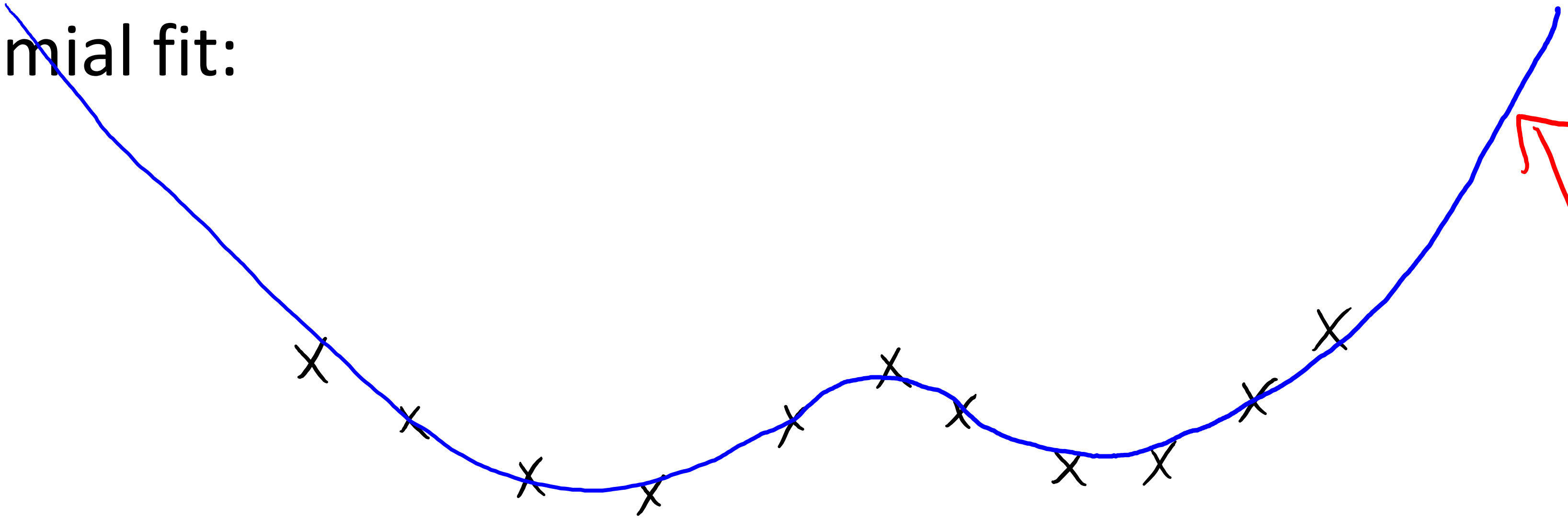
Gaussian RBFs: A Sum of “bumps”



- Gaussian RBFs are universal approximators (compact subets of $\mathbb{R}^d$)
  - Enough bumps can approximate any continuous function to arbitrary precision.
  - Achieve optimal test error as ‘n’ goes to infinity.

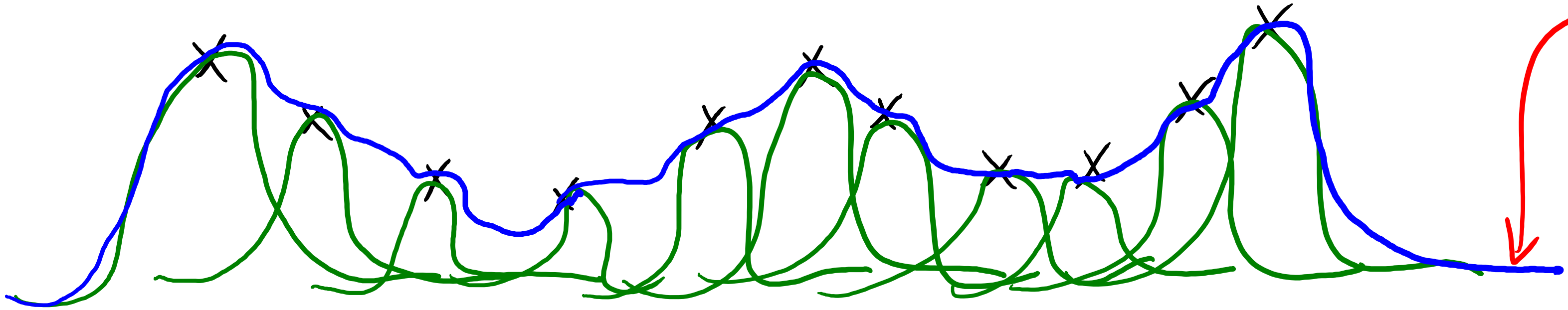
Gaussian RBFs: A Sum of “Bumps”

- Polynomial fit:



polynomial basis becomes polynomial away from data

- Constructing a function from bumps:

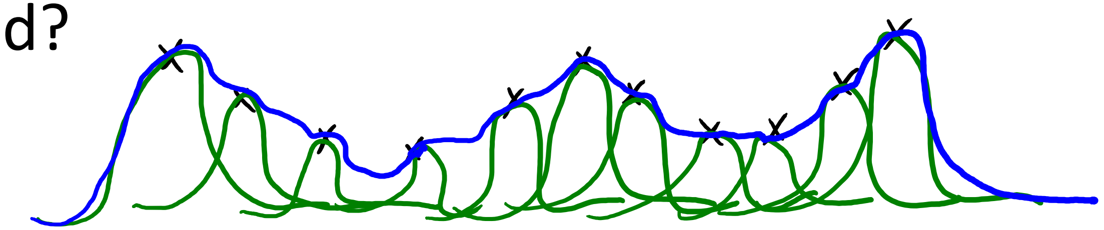


Gaussian RBFs go to zero away from data.

- Bonus slides: challenges of “far from data” (and future) predictions.

Gaussian RBF Parameters

- Some obvious questions:
  1. How many bumps should we use?
  2. Where should the bumps be centered?
  3. How high should the bumps go?
  4. How wide should the bumps be?



- The usual answers:
  1. We use ‘n’ bumps (non-parametric basis).
  2. Each bump is centered on one training example x_i.
  3. Fitting regression weights ‘w’ gives us the heights (and signs).
  4. The width is a hyper-parameter (narrow bumps == complicated model).

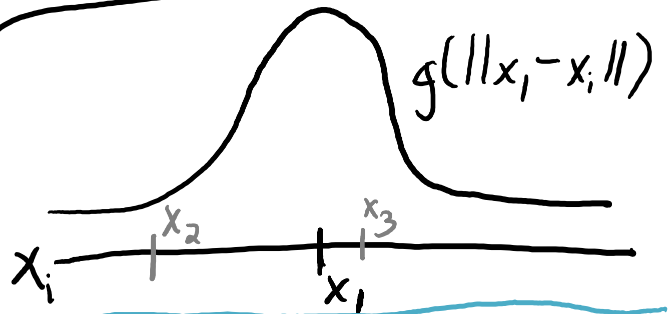
Gaussian RBFs: Formal Details

- What is a radial basis functions (RBFs)?
  - A set of non-parametric bases that depend on distances to training points.

Replace $X = \underbrace{\begin{bmatrix} \\ \\ \\ \end{bmatrix}}_{d} \Big\} n$ by $Z = \underbrace{\begin{bmatrix} g(\|x_1 - x_1\|) & g(\|x_1 - x_2\|) & \cdots & g(\|x_1 - x_n\|) \\ g(\|x_2 - x_1\|) & g(\|x_2 - x_2\|) & \cdots & g(\|x_2 - x_n\|) \\ \vdots & & \ddots & \vdots \\ g(\|x_n - x_1\|) & g(\|x_n - x_2\|) & \cdots & g(\|x_n - x_n\|) \end{bmatrix}}_{n} \Big\} n$

  - Most common 'g' is Gaussian RBF:

$$g(\varepsilon) = \exp\left(-\frac{\varepsilon^2}{2\sigma^2}\right)$$



- Variance σ^2 is a hyper-parameter controlling "width".
  - This affects fundamental trade-off (set it using a validation set).

Do we need $\sigma\sqrt{2\pi}$?
-No because $v^T z_i = (\frac{1}{\beta} v)^T (\beta z_i)$

Gaussian RBFs: Formal Details

- What is a radial basis functions (RBFs)?
  - A set of non-parametric bases that depend on distances to training points.

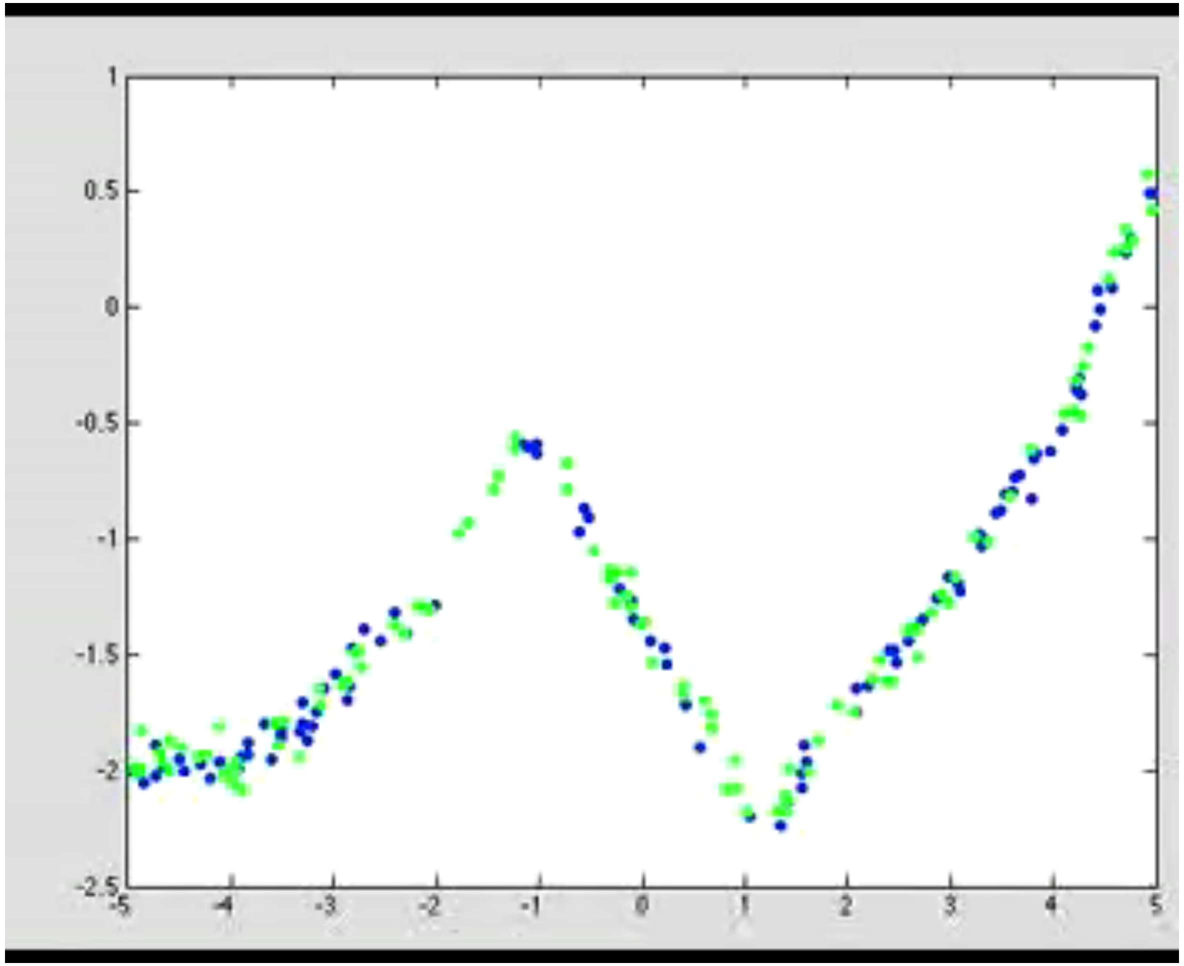
Replace $X = \underbrace{\begin{bmatrix} & & \\ & & \\ & & \end{bmatrix}}_{d} \Big\}n$ by $Z = \underbrace{\begin{bmatrix} g(\|x_1 - x_1\|) & g(\|x_1 - x_2\|) & \cdots & g(\|x_1 - x_n\|) \\ g(\|x_2 - x_1\|) & g(\|x_2 - x_2\|) & \cdots & g(\|x_2 - x_n\|) \\ \vdots & & \ddots & \vdots \\ g(\|x_n - x_1\|) & g(\|x_n - x_2\|) & \cdots & g(\|x_n - x_n\|) \end{bmatrix}}_{n} \Bigg\}n$

To make predictions on $\tilde{X} = \underbrace{\begin{bmatrix} & & \\ & & \\ & & \end{bmatrix}}_{d} \Big\}t$ use $\tilde{Z} = \underbrace{\overset{n}{\begin{bmatrix} & & \\ & g(\|\tilde{x}_i - x_j\|) & \\ & & \end{bmatrix}}}_{n} \Big\}t$

Number of "features" is number of training examples.

Non-Parametric Basis: RBFs

- Least squares with Gaussian RBFs for different σ values:



Could add bias and linear basis:

$$Z = \left[\begin{array}{ccccc} 1 & — x_1 — & g(\|x_1 - x_1\|) & \cdots & g(\|x_1 - x_n\|) \\ 1 & — x_2 — & \vdots & \ddots & \vdots \\ 1 & — x_3 — & & & \\ \vdots & \vdots & & & \\ 1 & — x_n — & g(\|x_1 - x_n\|) & \cdots & g(\|x_n - x_n\|) \end{array}\right]$$

$$\underbrace{\quad}_{1} \qquad \underbrace{\qquad}_{d} \qquad \underbrace{\qquad\qquad\qquad\qquad}_{n}$$

This reverts to linear regression instead of 0 away from data.

Summary

- Tree/probabilistic/non-parametric/ensemble regression methods.
- Non-linear transforms:
  - Allow us to model non-linear relationships with linear models.
  - Polynomial features are a parametric example
  - RBF features are a non-parametric example

Complexity Penalties

- The next 3 slides are a previous of next week’s topics.

Finding the “True” Model

- What if our goal is find the “true” model?
  - We believe that y_i really is a polynomial function of x_i.
  - We want to find the degree of the polynomial ‘p’.

- Should we choose the ‘p’ with the lowest training error?
  - No, this will pick a ‘p’ that is way too large.
    (training error always decreases as you increase ‘p’)

Finding the "True" Model

- What if our goal is find the "true" model?
  - We believe that y_i really is a polynomial function of x_i.
  - We want to find the degree of the polynomial 'p'.

- Should we choose the 'p' with the lowest validation error?
  - This will also often choose a 'p' that is too large.

  - Even if true model has p=2, this is a special case of a degree-3 polynomial.
  - If 'p' is too big then we overfit, but might still get a lower validation error.
    - Another example of optimization bias.

Complexity Penalties

- There are a lot of "scores" people use to find the "true" model.
- Basic idea behind them: put a penalty on the model complexity.
  - Want to **fit the data and have a simple model**.
- For example, minimize training error plus the degree of polynomial.

Let $$Z_p = \begin{bmatrix} 1 & x_1 & (x_1)^2 & \cdots & (x_1)^p \\ 1 & x_2 & (x_2)^2 & \cdots & (x_2)^p \\ 1 & x_3 & (x_3)^2 & \cdots & (x_3)^p \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & x_n & (x_n)^2 & \cdots & (x_n)^p \end{bmatrix}$$

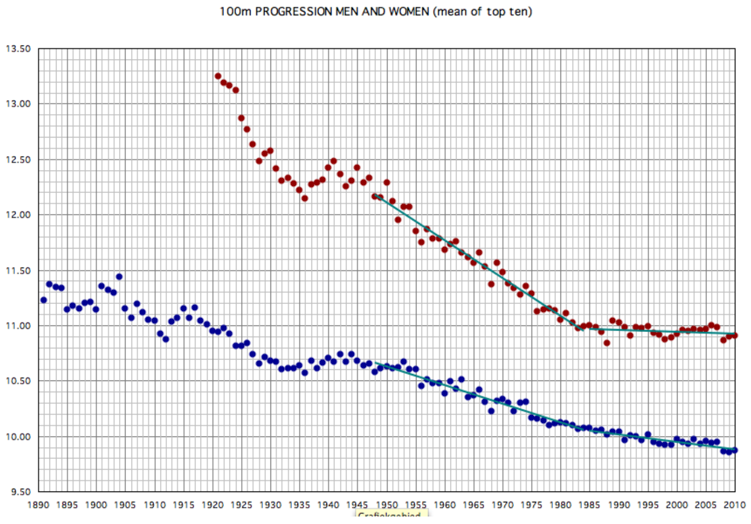
Find 'p' that minimizes:

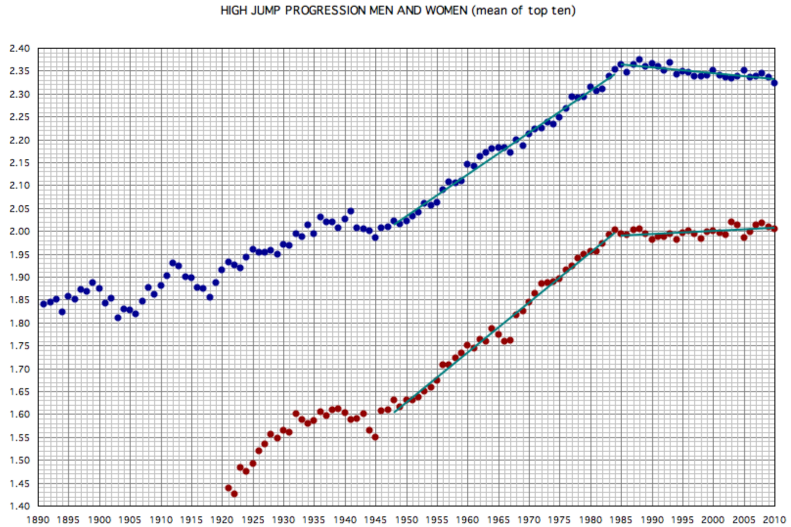
$$\text{score}(p) = \underbrace{\frac{1}{2}\|Z_p v - y\|^2}_{\text{train error for best 'v' with this basis.}} + \underbrace{p}_{\text{degree of polynomial}}$$

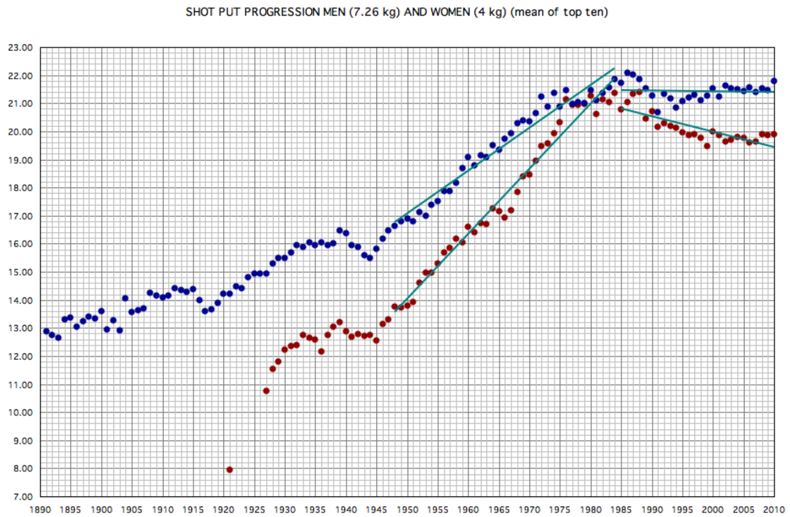
  - If we use p=4, use "training error plus 4" as error.
- If two 'p' values have similar error, this prefers the smaller 'p'.

Motivation: Non-Linear Progressions in Athletics

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







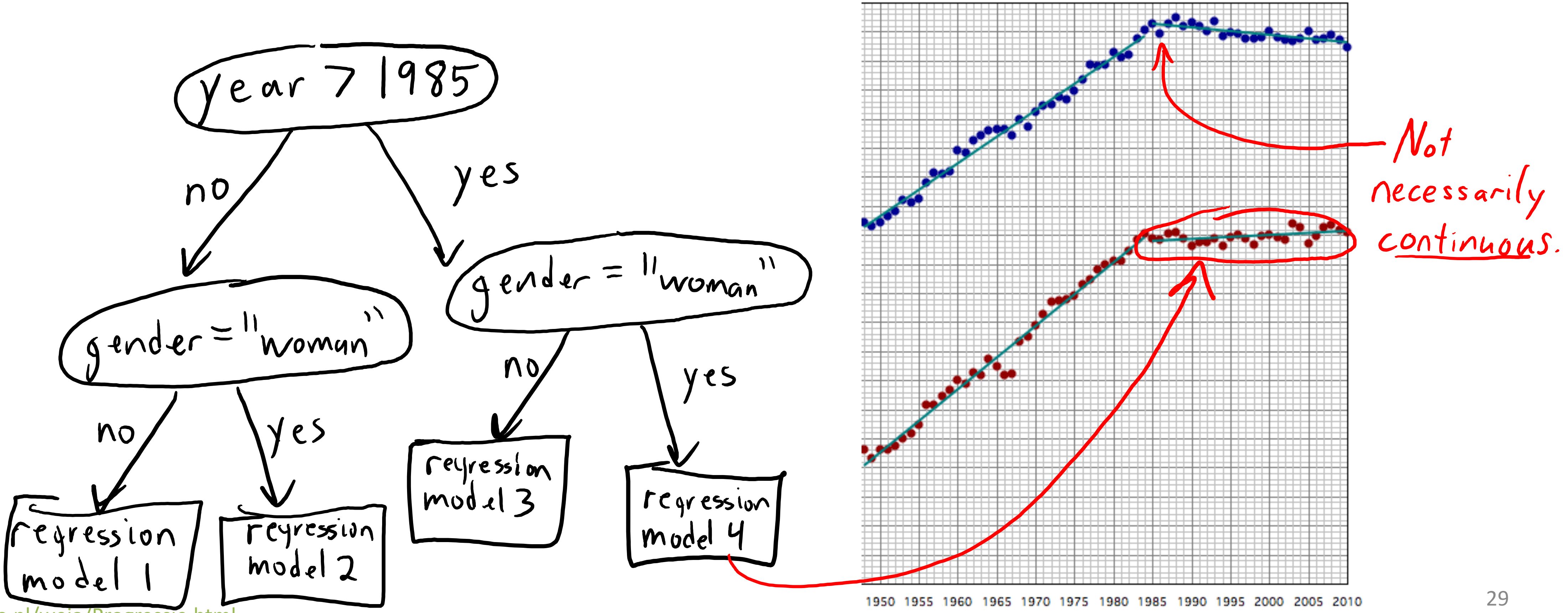
http://www.at-a-lanta.nl/weia/Progressie.html
https://en.wikipedia.org/wiki/Usain_Bolt
http://www.britannica.com/biography/Florence-Griffith-Joyner

Adapting Counting/Distance-Based Methods

- We can adapt our classification methods to perform regression:

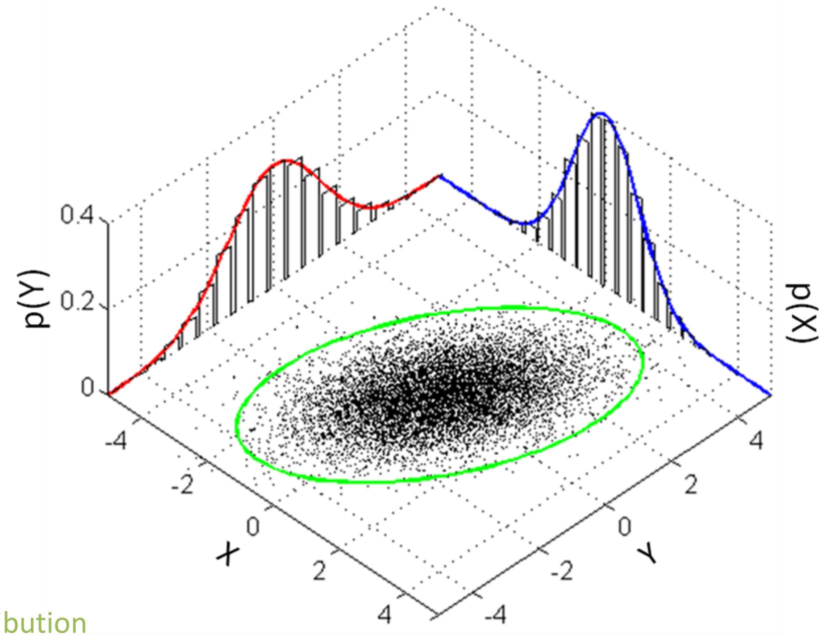
Adapting Counting/Distance-Based Methods

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



Adapting Counting/Distance-Based Methods

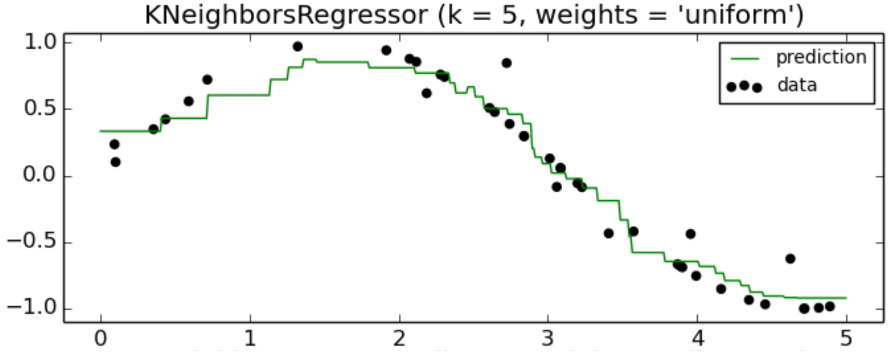
- We can adapt our classification methods to perform regression:
  - Regression tree: tree with mean value or linear regression at leaves.
  - Probabilistic models: fit $p(x_i \mid y_i)$ and $p(y_i)$ with Gaussian or other model.
    - CPSC 540.



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

Adapting Counting/Distance-Based Methods

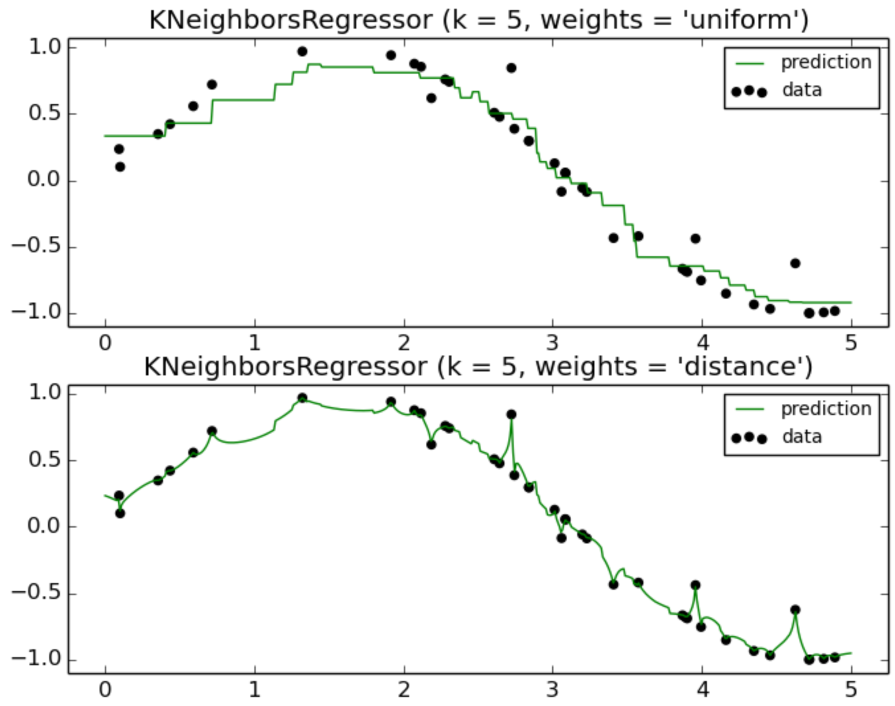
- We can adapt our classification methods to perform regression:
  - Regression tree: tree with mean value or linear regression at leaves.
  - Probabilistic models: fit $p(x_i \mid y_i)$ and $p(y_i)$ with Gaussian or other model.
  - Non-parametric models:
    - KNN regression:
      - Find 'k' nearest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

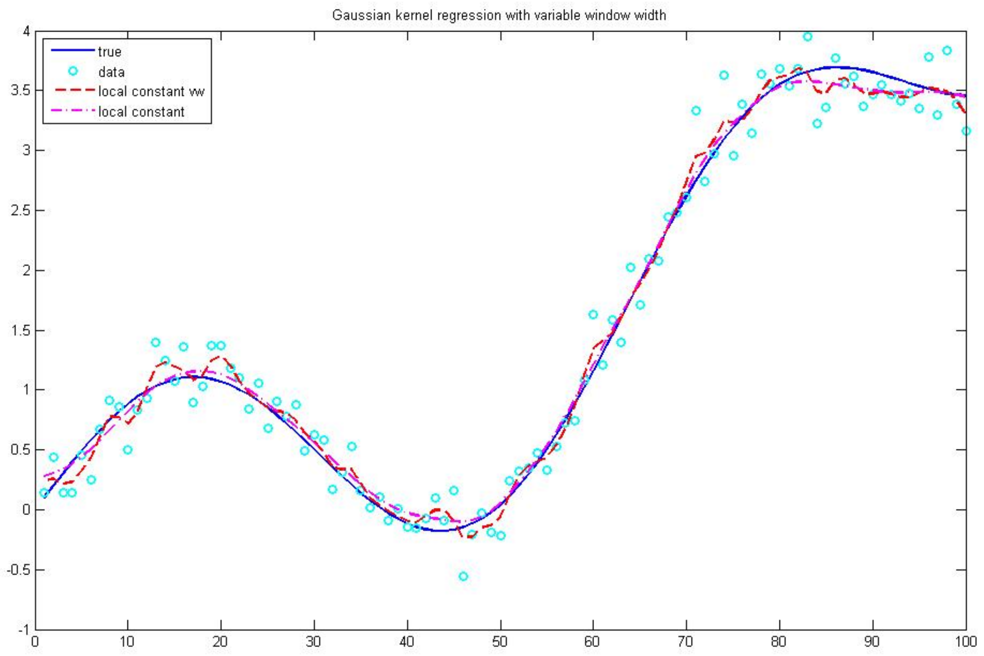


http://scikit-learn.org/stable/modules/neighbors.html

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.

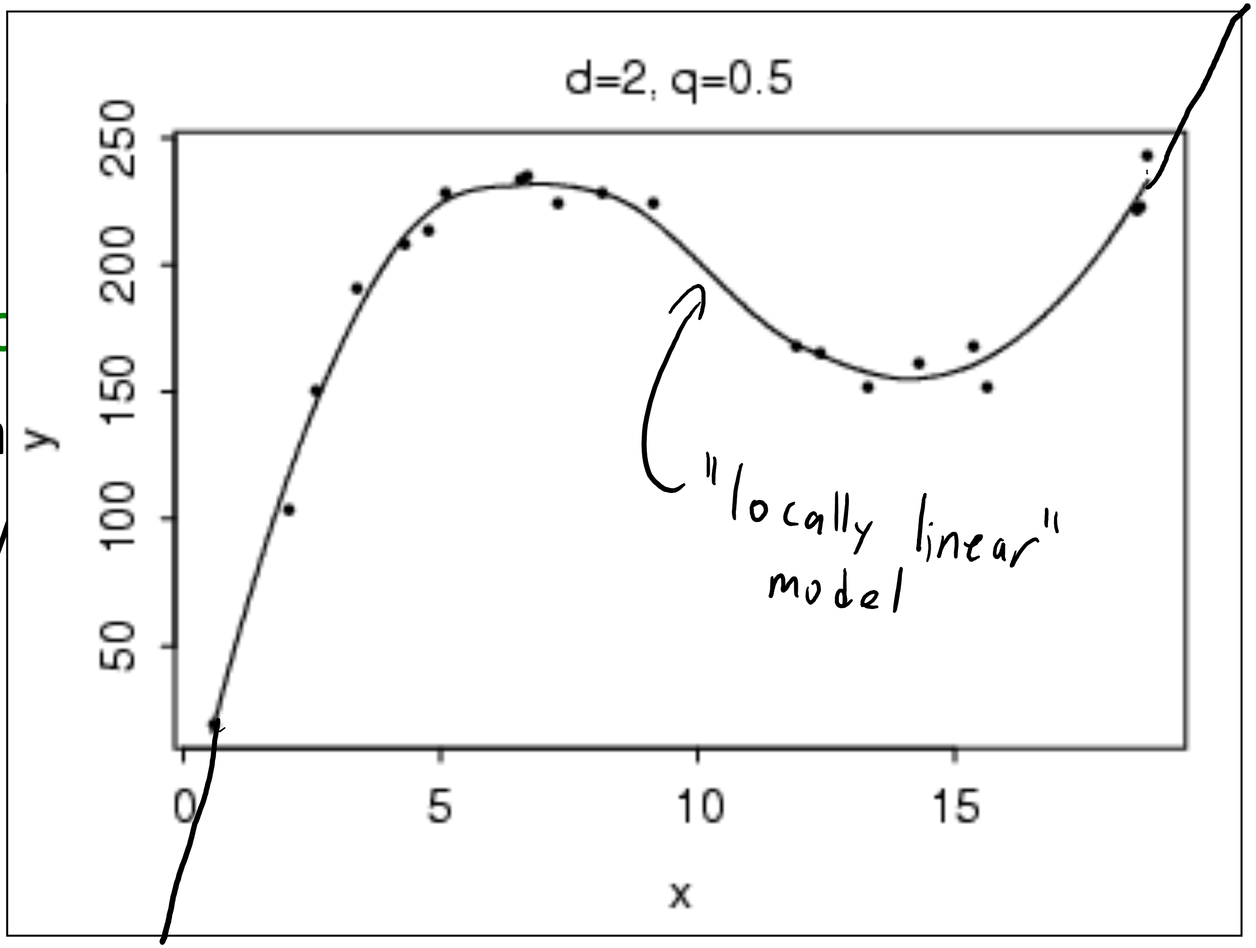
$$\hat{y}_i = \frac{\sum_{j=1}^{n} v_{ij} y_j}{\sum_{j=1}^{n} v_{ij}}$$



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.)



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