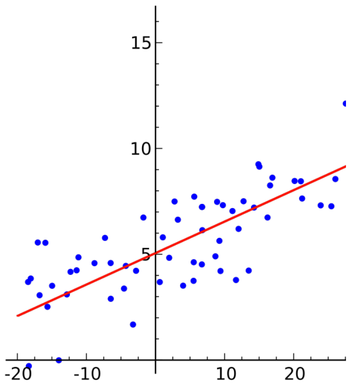
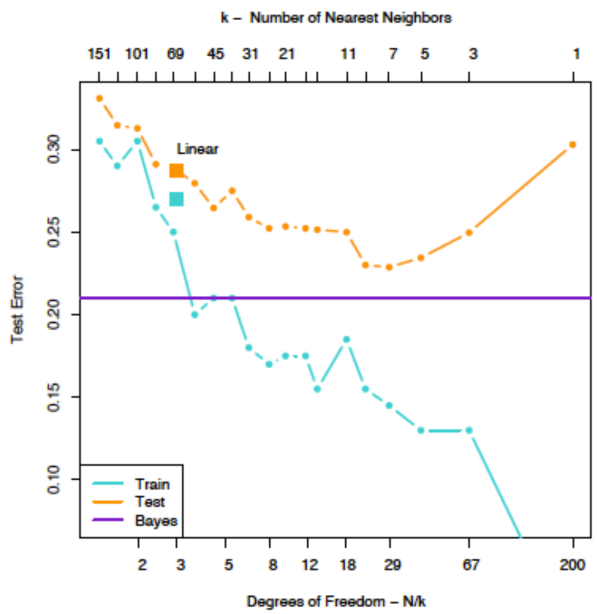
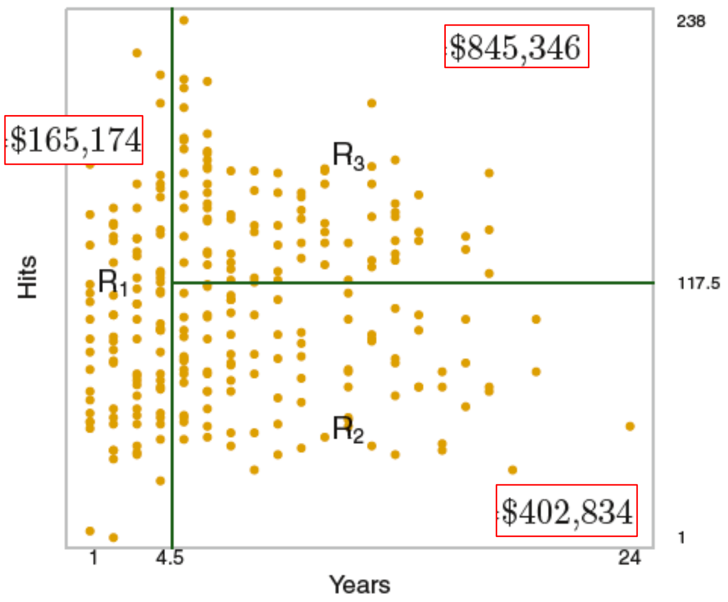
Tufts COMP 135: Introduction to Machine Learning
https://www.cs.tufts.edu/comp/135/2020f/

Regression Basics







Many slides attributable to:
Erik Sudderth (UCI)
Finale Doshi-Velez (Harvard)
James, Witten, Hastie, Tibshirani (ISL/ESL books)

Prof. Mike Hughes

Objectives for Today (day 02)

- Understand 3 steps of a regression task
  - Training
  - Prediction
  - Evaluation
    - Metrics: Mean Squared Error vs Mean Absolute Error

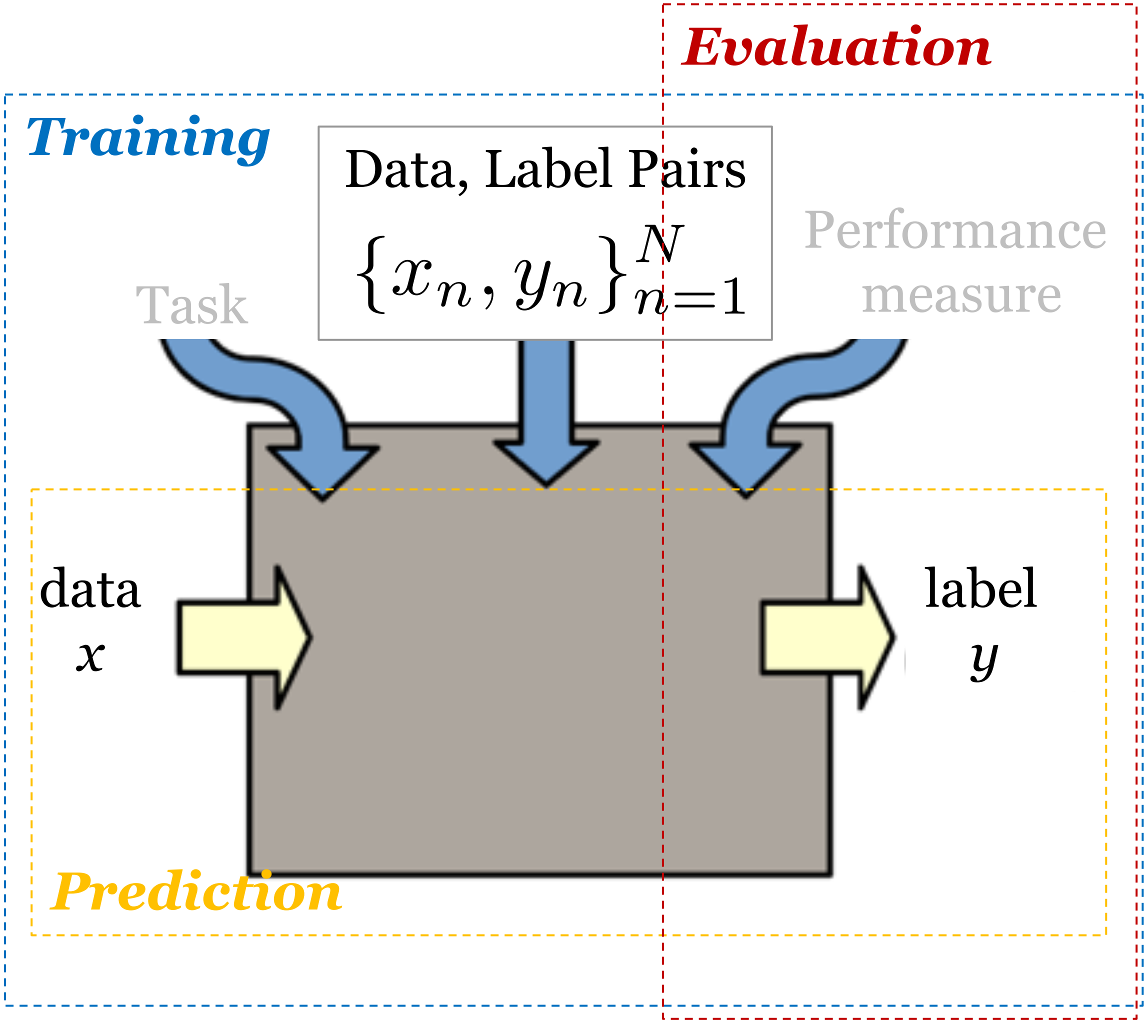
- Try two methods (focus: prediction and evaluation)
  - Linear Regression
  - K-Nearest Neighbors

What will we learn?

Supervised Learning

Unsupervised Learning

Reinforcement Learning



Task: Regression

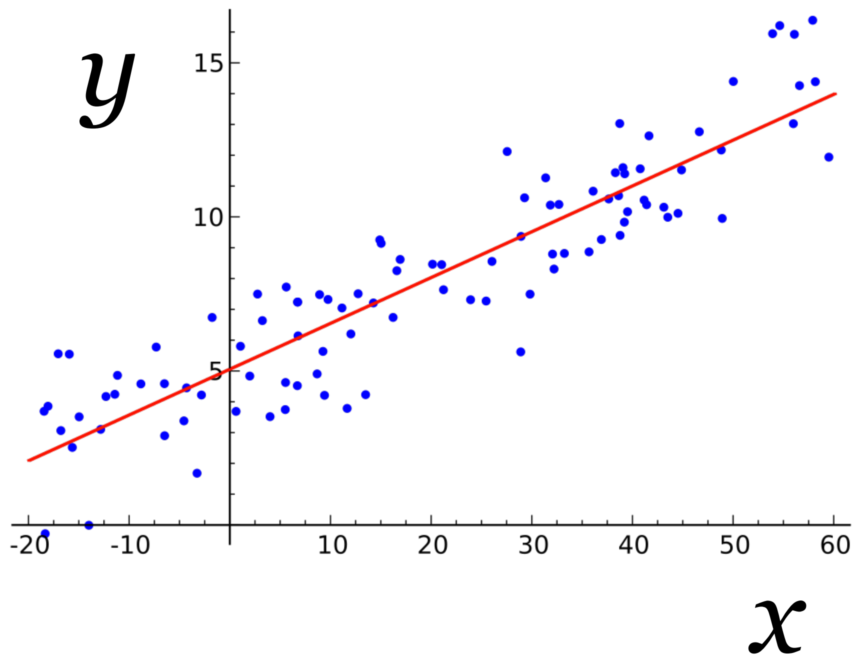
Supervised Learning

regression

Unsupervised Learning

Reinforcement Learning

y is a numeric variable
e.g. sales in $$



Regression Example: RideShares

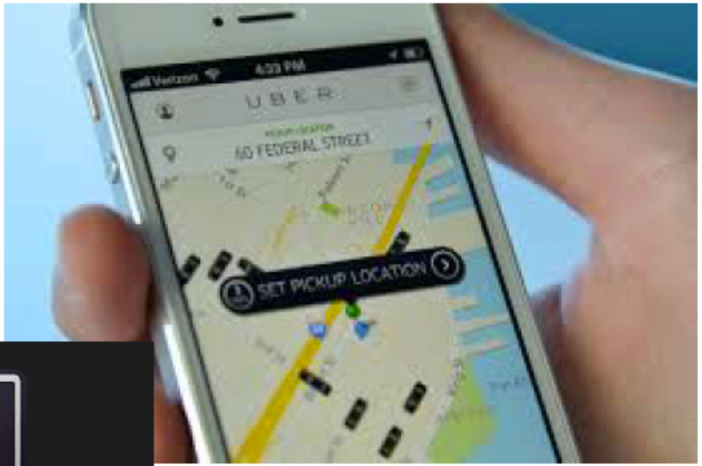
Supervised Learning

regression

Unsupervised Learning

Reinforcement Learning

Predictions of travel time, price, supply, demand

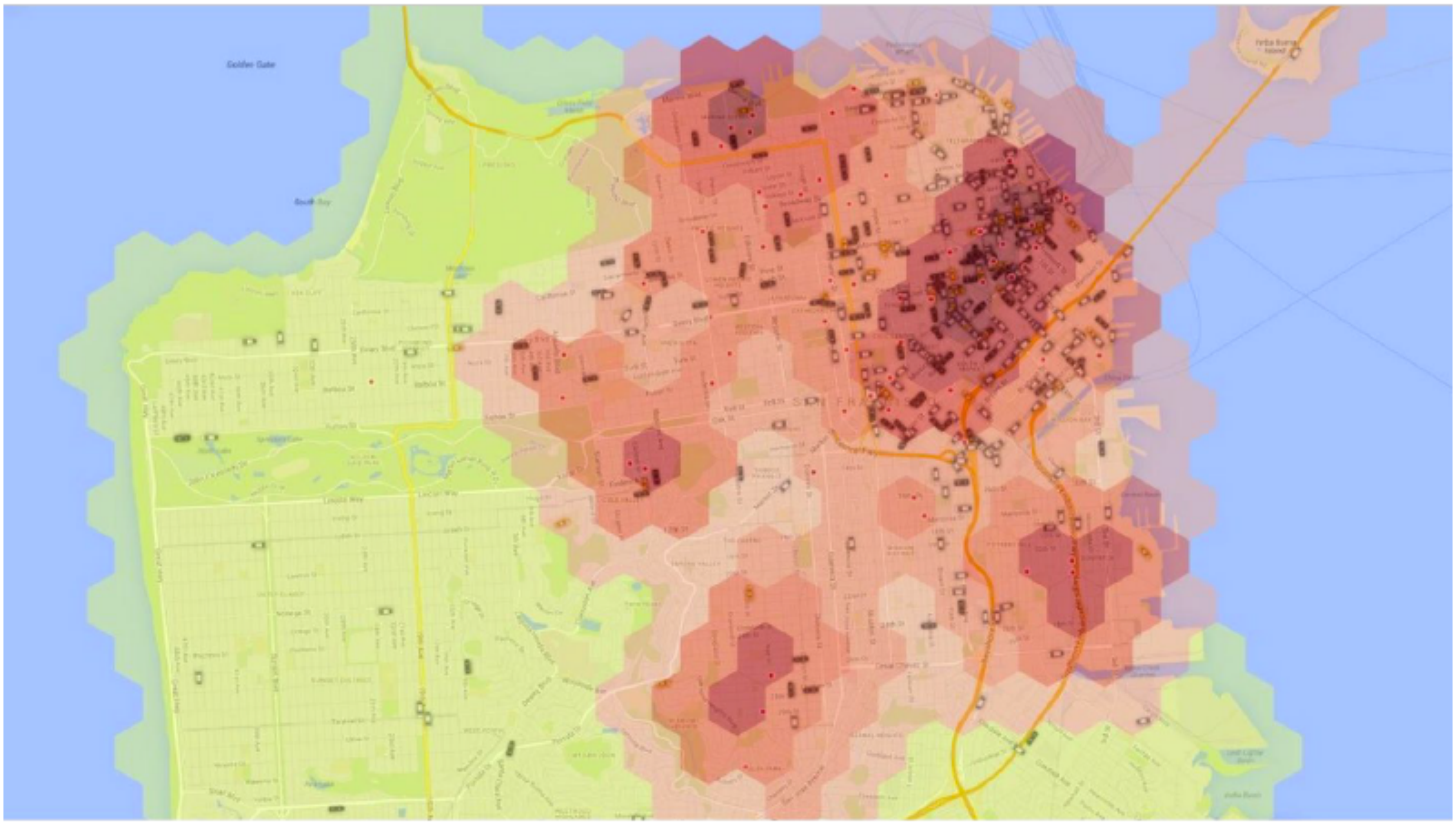


UBER

Regression Example: RideShare

(Keith Chen)

Regression Example: RideShare



(Keith Chen)

Regression: **Prediction Step**

Goal: Predict response *y* well given features x

- Input: $$x_i \triangleq [x_{i1}, x_{i2}, \ldots x_{if} \ldots x_{iF}]$$

  "features"
  "covariates"
  "predictors"
  "attributes"

  Entries can be real-valued, or other numeric types (e.g. integer, binary)

- Output: $$\hat{y}(x_i) \in \mathbb{R}$$

  "responses"
  "labels"

  Scalar value like 3.1 or -133.7

Regression: **Prediction Step**

```
>>> # Given: pretrained regression object model
>>> # Given: 2D array of features x_NF

>>> x_NF.shape
(N, F)

>>> yhat_N1 = model.predict(x_NF)


>>> yhat_N1.shape
(N,1)
```

Regression: **Training Step**

Goal: Given a labeled dataset, learn a **function** that can perform prediction well

- Input: Pairs of features and labels/responses

$$\{x_n, y_n\}_{n=1}^{N}$$

- Output: $\hat{y}(\cdot) : \mathbb{R}^F \rightarrow \mathbb{R}$

Regression: **Training Step**

```
>>> # Given: 2D array of features x_NF
>>> # Given: 1D array of responses/labels y_N1
```

```
>>> y_N1.shape
(N, 1)
>>> x_NF.shape
(N, F)
```

```
>>> model = RegressionModel()
>>> model.fit(x_NF, y_N1)
```

Regression: **Evaluation Step**

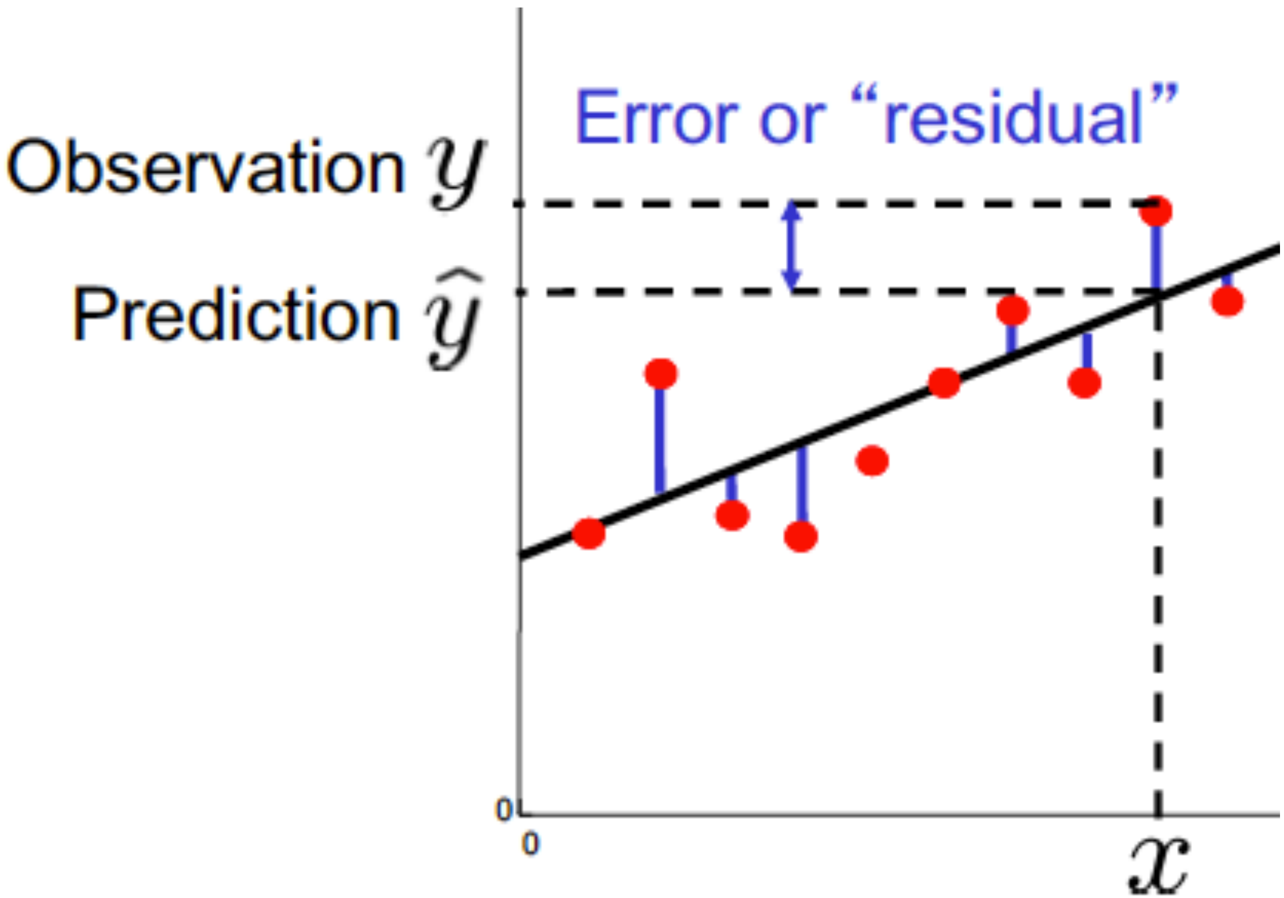
Goal: Assess quality of predictions

- Input: Pairs of predicted and "true" responses

$$\{\hat{y}(x_n), y_n\}_{n=1}^{N}$$

- Output: Scalar measure of error/quality
  - Measuring Error: **lower** is better
  - Measuring Quality: **higher** is better

Visualizing errors



Regression: Evaluation Metrics

- mean squared error $$\frac{1}{N}\sum_{n=1}^{N}(y_n - \hat{y}_n)^2$$

- mean absolute error $$\frac{1}{N}\sum_{n=1}^{N}|y_n - \hat{y}_n|$$

Regression: Evaluation Metrics

https://scikit-learn.org/stable/modules/model_evaluation.html

Regression	
'explained_variance'	`metrics.explained_variance_score`
'neg_mean_absolute_error'	`metrics.mean_absolute_error`
'neg_mean_squared_error'	`metrics.mean_squared_error`
'neg_mean_squared_log_error'	`metrics.mean_squared_log_error`
'neg_median_absolute_error'	`metrics.median_absolute_error`
'r2'	`metrics.r2_score`

Linear Regression

Parameters:

weight vector $w = [w_1, w_2, \dots w_f \dots w_F]$

bias scalar b

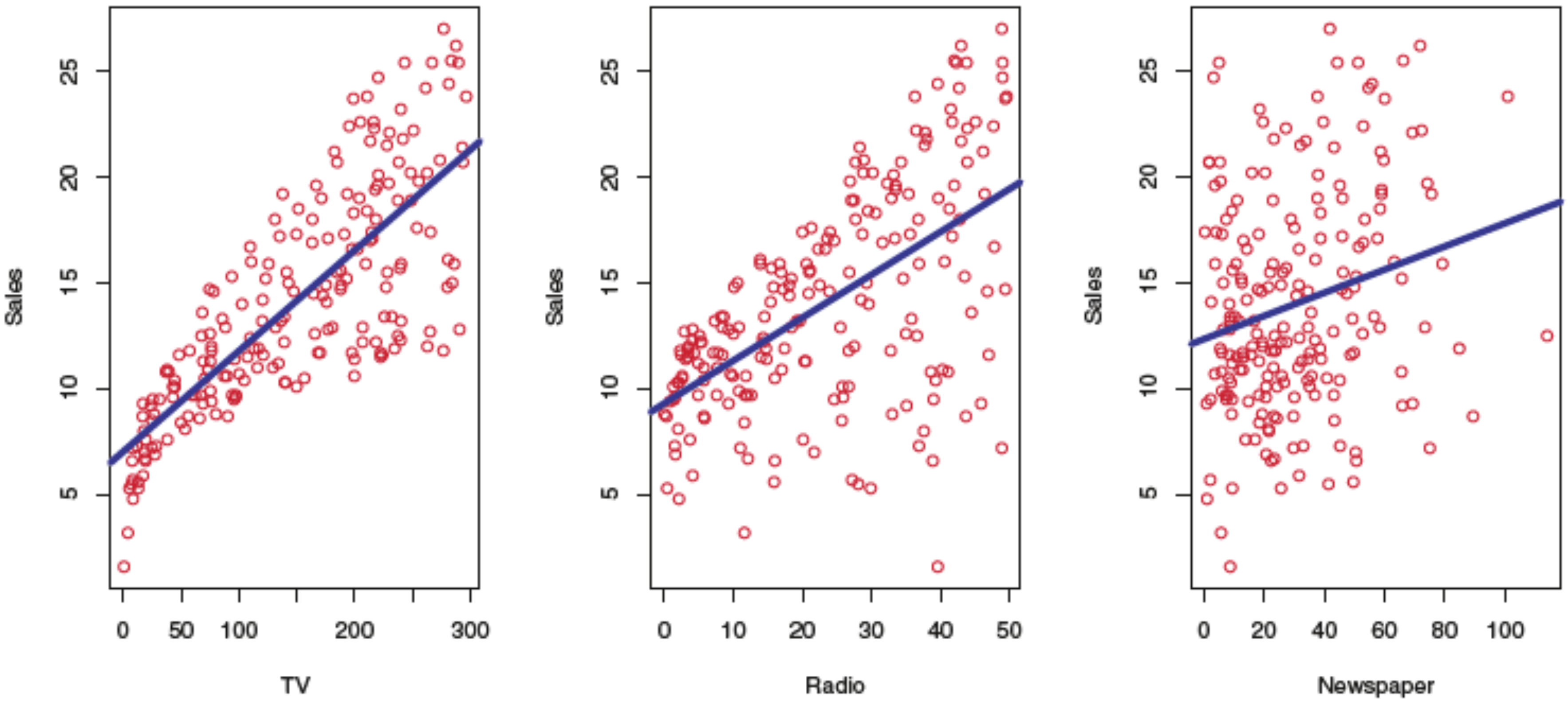
Prediction:

$$\hat{y}(x_i) \triangleq \sum_{f=1}^{F} w_f x_{if} + b$$

Training:

find weights and bias that minimize error

Linear Regression predictions as a function of one feature are **linear**



Linear Regression: Training

Goal:

Want to find the weight coefficients w and intercept/bias b that minimizing the mean squared error on the N training examples

Optimization problem: “Least Squares”

$$\min_{w,b} \sum_{n=1}^{N} \Big(y_n - \hat{y}(x_n, w, b) \Big)^2$$

Linear Regression: Training

Optimization problem: “Least Squares”

$$\min_{w,b} \sum_{n=1}^{N} \Big(y_n - \hat{y}(x_n, w, b) \Big)^2$$

An exact solution for optimal values of w, b exists!

Formula with many features (F >= **1**):

$$\tilde{X} = \begin{bmatrix} x_{11} & \dots & x_{1F} & 1 \\ x_{21} & \dots & x_{2F} & 1 \\ & & \dots & \\ x_{N1} & \dots & x_{NF} & 1 \end{bmatrix}$$

$$[w_1 \dots w_F \;\; b]^T = (\tilde{X}^T \tilde{X})^{-1} \tilde{X}^T y$$

We will cover and derive this in next class

Nearest Neighbor Regression

Parameters:

none

Prediction:

- find "nearest" training vector to given input x
- predict y value of this neighbor

Training:

none needed (use training data as lookup table)

K nearest neighbor regression

Parameters:

K : number of neighbors

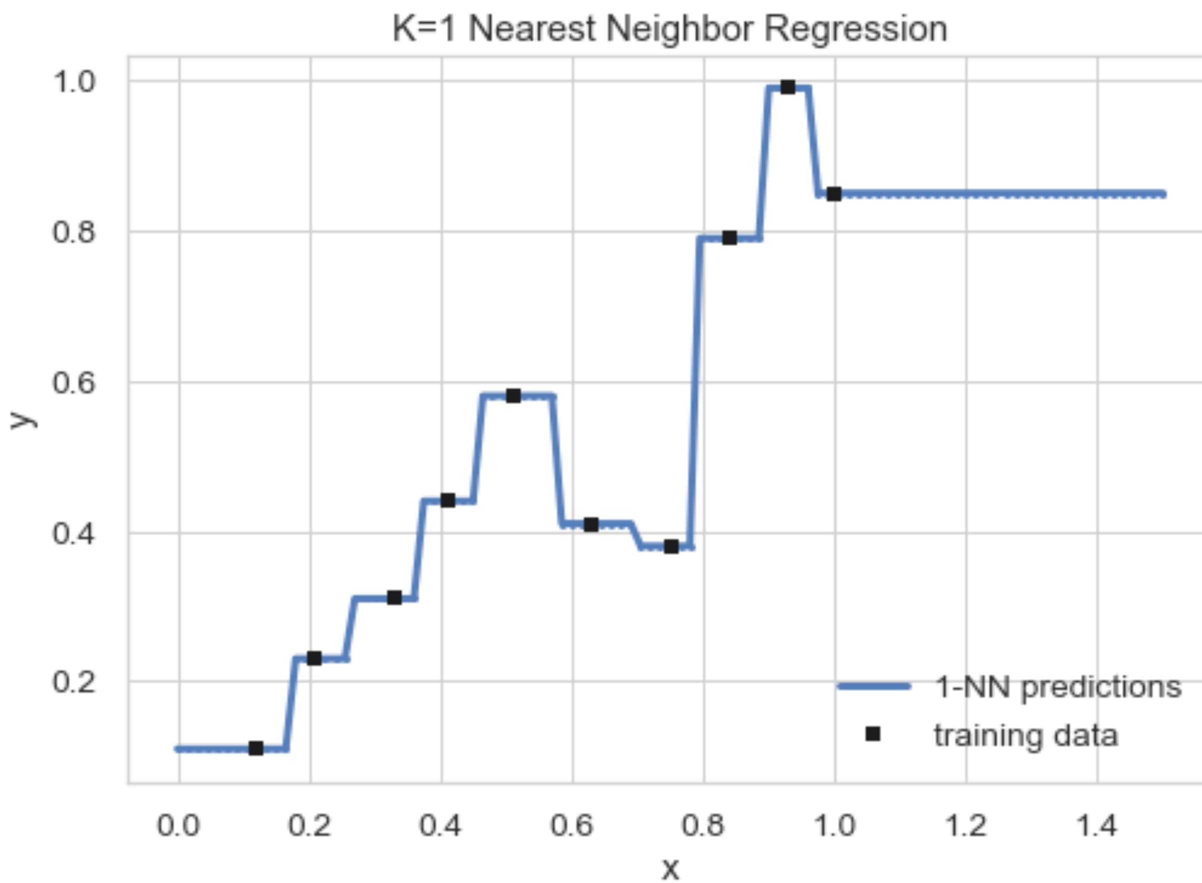
Prediction:

- find K "nearest" training vectors to input *x*
- predict **average** *y* of this neighborhood

Training:

none needed (use training data as lookup table)

Nearest Neighbor predictions as a function of one feature are **piecewise constant**



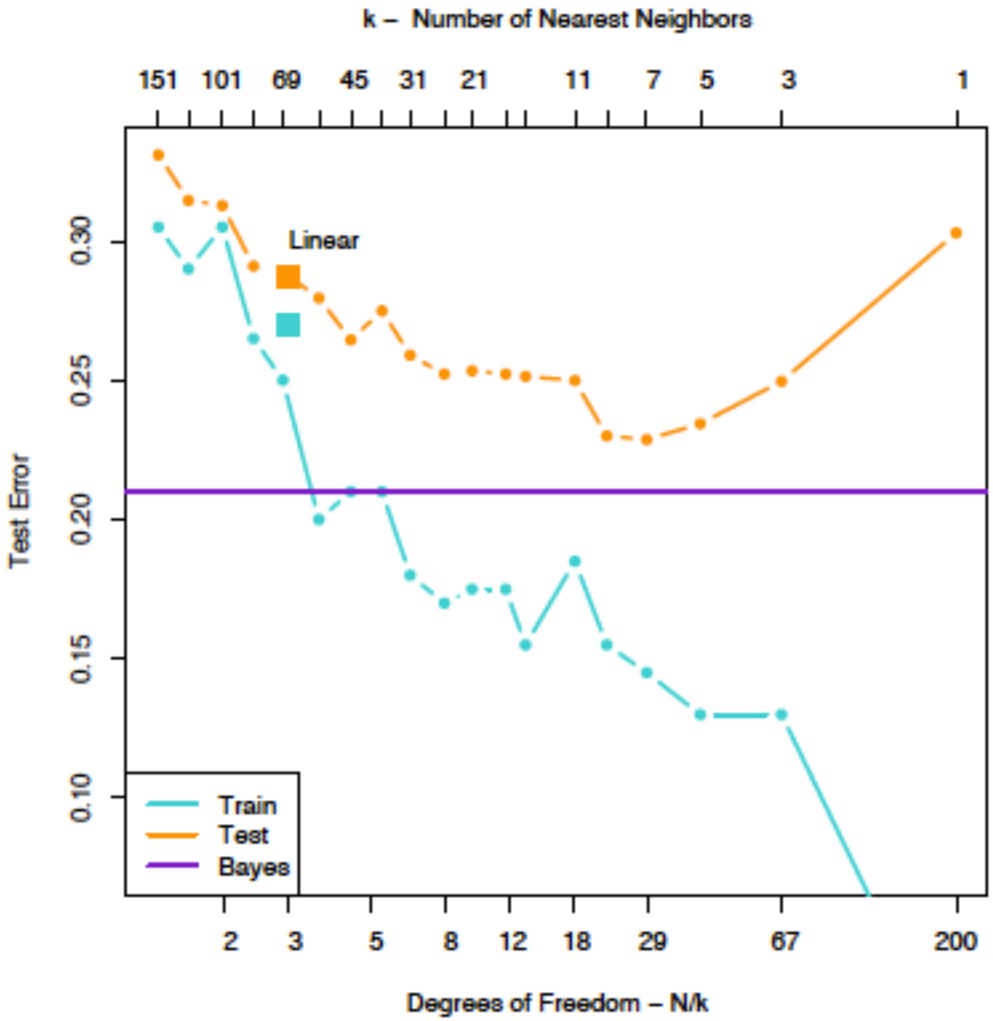
Distance metrics

- Euclidean $$\mathrm{dist}(x, x') = \sqrt{\sum_{f=1}^{F} (x_f - x'_f)^2}$$

- Manhattan $$\mathrm{dist}(x, x') = \sum_{f=1}^{F} |x_f - x'_f|$$

- Many others are possible

Error vs Model Complexity



Credit:
Fig 2.4
ESL textbook

Summary of Methods

	Function class flexibility	**Knobs to tune**	**How to interpret?**
Linear Regression	Linear	*Penalize weights* (*more next week*)	Inspect weights
K Nearest Neighbors Regression	Piecewise constant	Number of Neighbors Distance metric	Inspect neighbors

Objectives for Today (day 02)

- Understand 3 steps of a regression task
  - Training
  - Prediction
  - Evaluation
    - Mean Squared Error
    - Mean Absolute Error

- Chosen performance metric should be integrated at training
- Mean squared error is “easy”, but not always the right thing to do

- Try two methods (focus: prediction and evaluation)
  - Linear Regression
  - K-Nearest Neighbors

Breakout!

Lab for day02