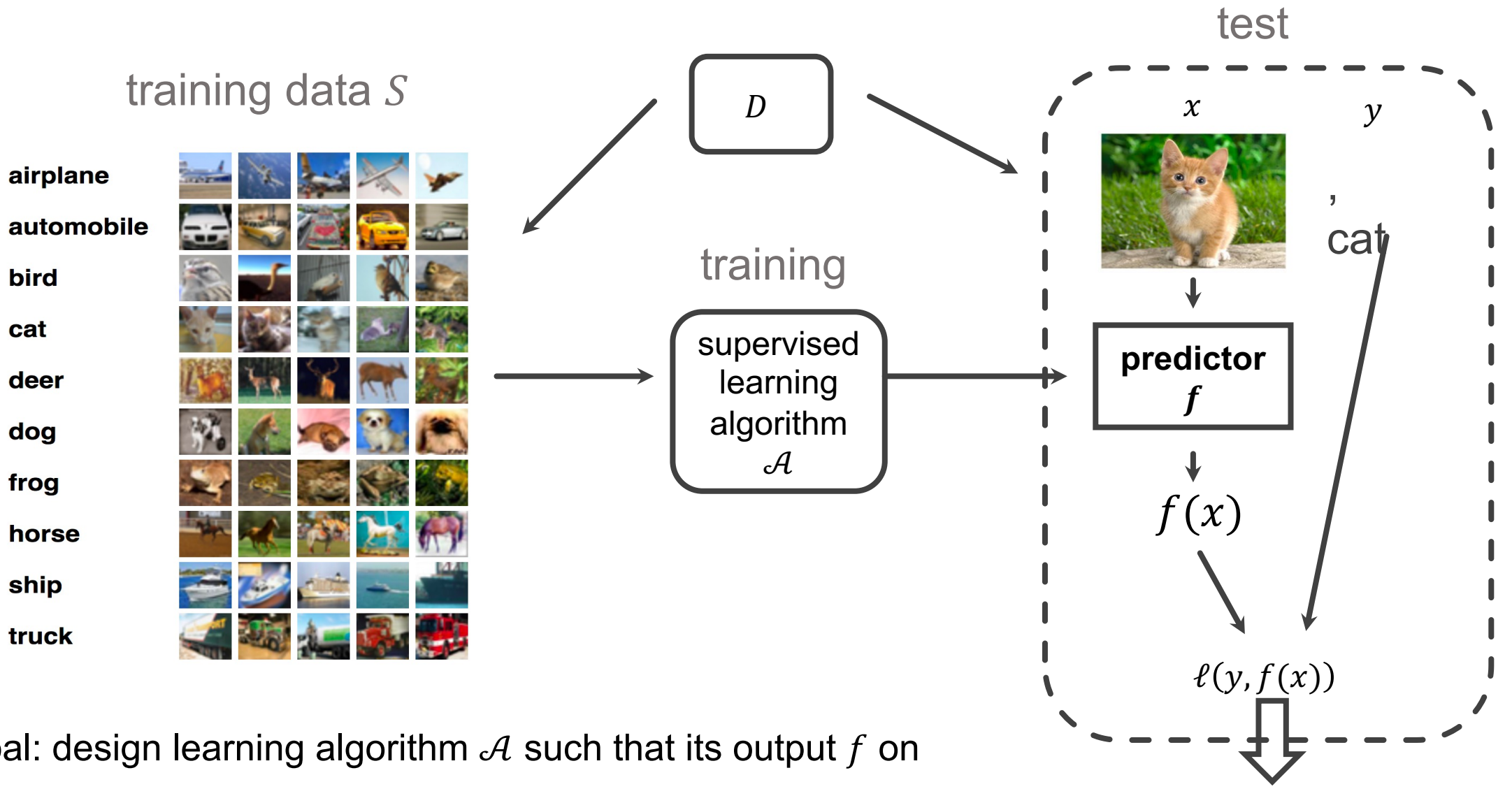
CSC580: Probabilistic Graphical Models

Midterm Review

Jason Pacheco

Supervised learning setup: putting it together



- Goal: design learning algorithm $\mathcal{A}$ such that its output f on iid training data S has low generalization error

Generalization error: $L_D(f) = \mathrm{E}_{(x,y)\sim D}\, \ell(y, f(x))$

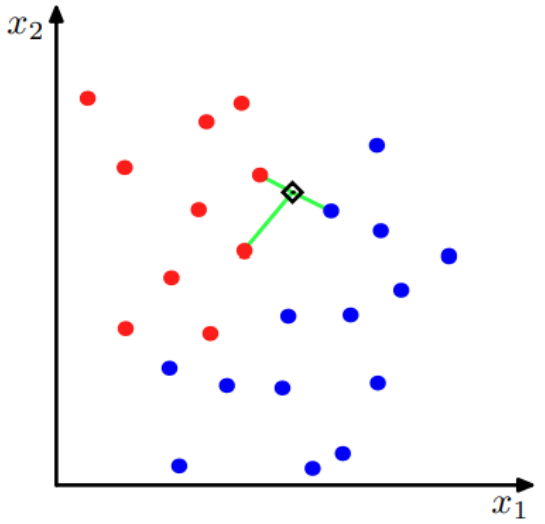
k-nearest neighbors (k-NN): main concept

Training set: $S = \{ (x_1, y_1), \ldots, (x_m, y_m)\}$

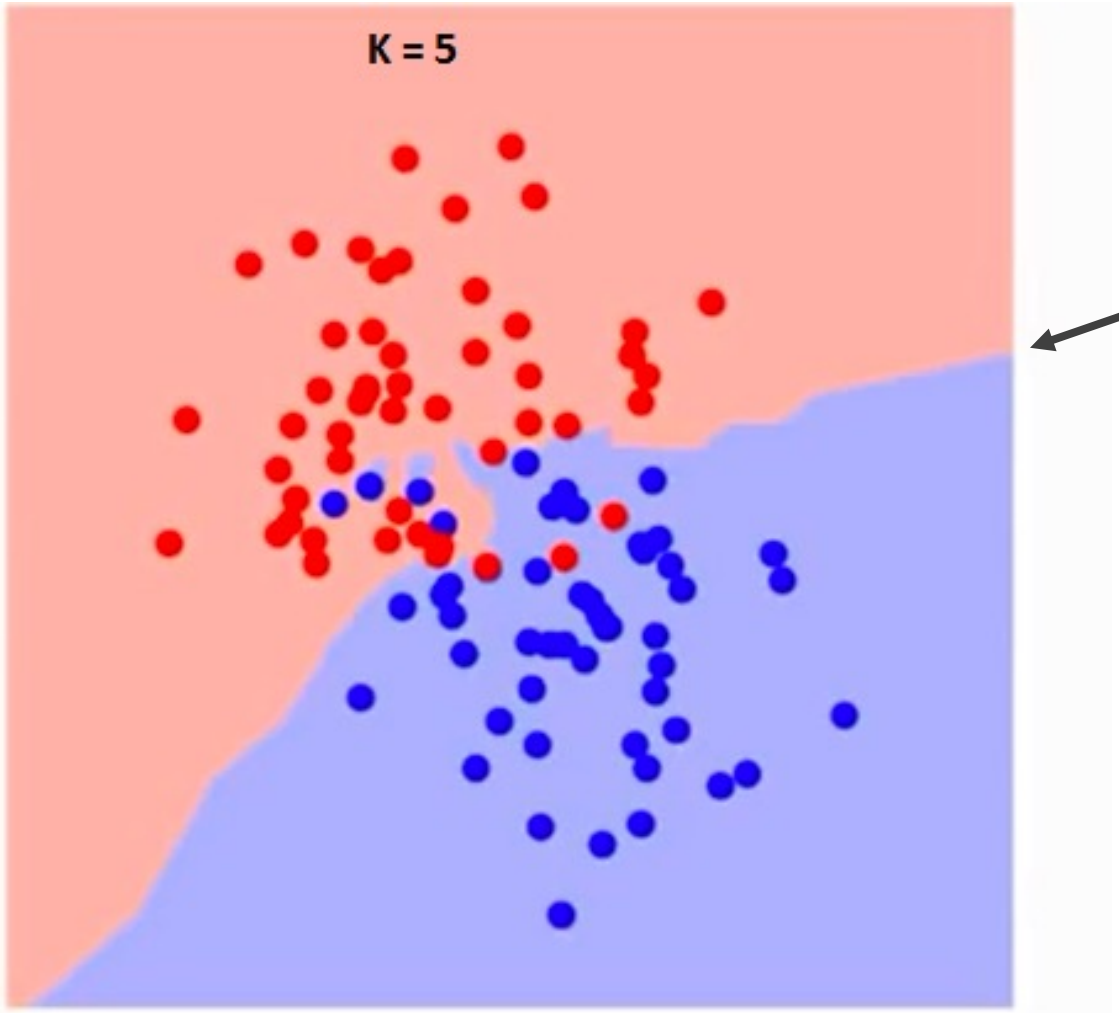
Inductive bias: given test example x, its label should resemble the labels of **nearby points**

Function

- input: x
- find the k nearest points to x from S; call their indices $N(x)$
- output: the majority vote of $\{y_i : i \in N(x)\}$
  - For regression, the average.



k-NN classification example



decision boundary

k-NN classification: pseudocode

- Training is trivial: store the training set
- Test:

Algorithm 3 KNN-PREDICT($\mathbf{D}$, K, $\hat{x}$)

```
S ← [ ]                                                  // list
for n = 1 to N do
    S ← S ⊕ ⟨d(x_n, x̂), n⟩                               // append to list: store distance to training example n
end for
S ← SORT(S)                                              // sort in first coordinate: put lowest-distance objects first
ŷ ← 0
for k = 1 to K do
    ⟨dist, n⟩ ← S_k                                      // n this is the kth closest data point
    ŷ ← ŷ + y_n                                          // vote according to the label for the nth training point
end for
return SIGN(ŷ)                                           // Majority vote of {y_i : i ∈ N(x)}; return +1 if ŷ > 0 and −1 if ŷ < 0
```

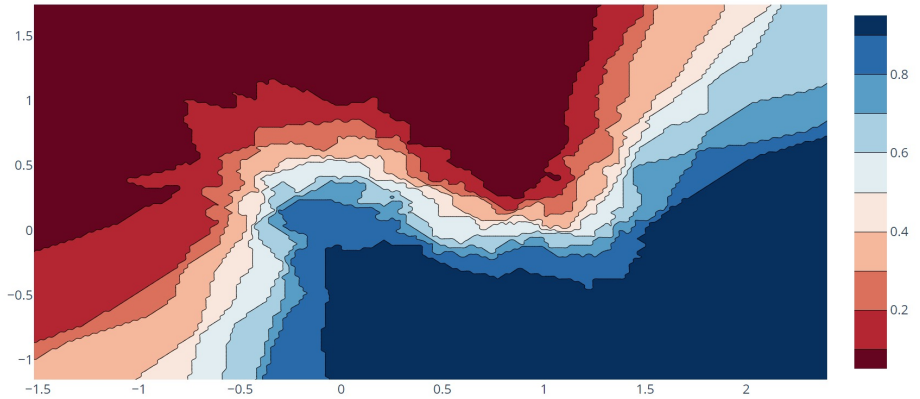
- Time complexity (assuming distance calculation takes $O(d)$ time)
  - $O(m\,d\ + m \log m\ + k\ ) = O\big(m(d\ + \log m)\big)$
- Faster nearest neighbor search: k-d trees, locality sensitive hashing

Variations

- Classification
  - Recall the majority vote rule: $\hat{y} = \arg\max_{y \in \{1,\ldots,C\}} \sum_{i \in N(x)} 1\{y_i = y\}$
  - Soft weighting nearest neighbors: $\hat{y} = \arg\max_{y \in \{1,\ldots,C\}} \sum_{i=1}^{m} w_i \, 1\{y_i = y\}$,

    where $w_i \propto \exp(-\beta \, d(x, x_i))$, or $\propto \frac{1}{1+d(x,x_i)^{\beta}}$

- Class probability estimates
  - $\hat{P}(Y = y \mid x) = \frac{1}{k} \sum_{i \in N(x)} 1\{y_i = y\}$
  - Useful for “classification with rejection”



Inductive Bias

Training

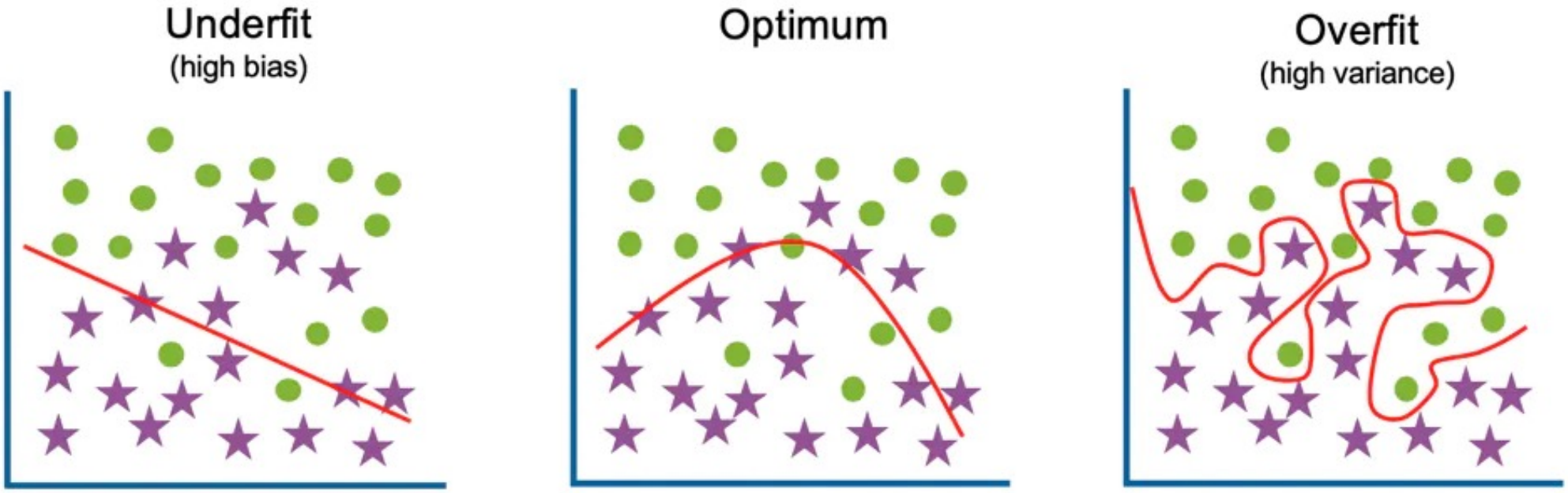
class A

class B

Test

How would you label the test examples?

Overfitting vs Underfitting



Source: ibm.com

Bayes optimal classifier

Theorem f_{BO} achieves the smallest 0-1 error among all classifiers.

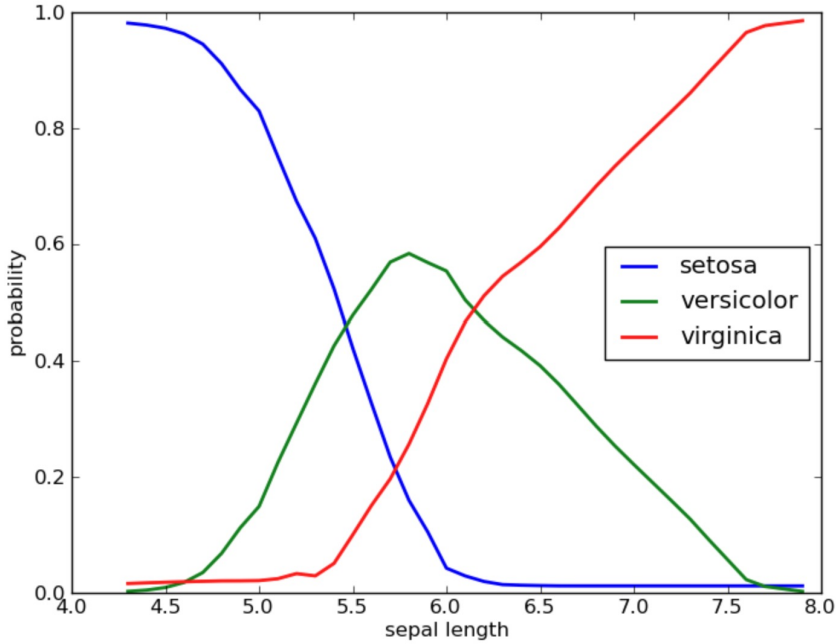
$$f_{BO}(x) = \arg\max_{y \in \mathcal{Y}} P_D(X = x, Y = y) = \arg\max_{y \in \mathcal{Y}} P_D(Y = y \,|X = x), \forall x \in \mathcal{X}$$

Example Iris dataset classification:

Iris Setosa

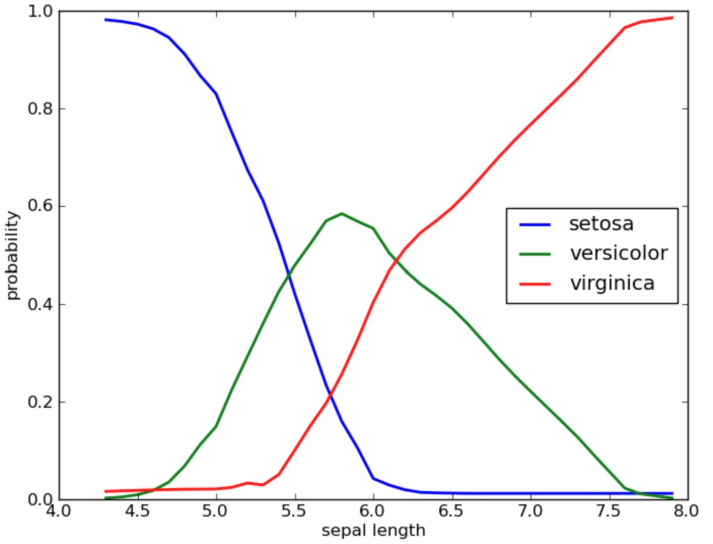
Iris Versicolor

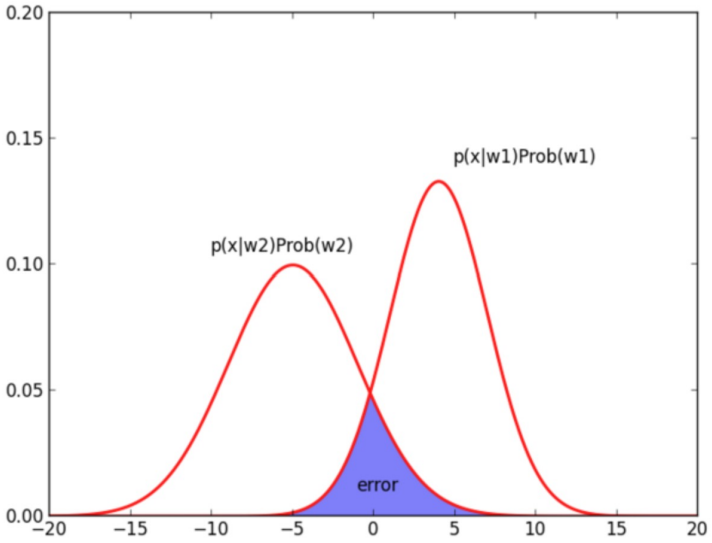
Iris Virginica



Bayes error rate: alternative form

$$
\begin{aligned}
L_D(f_{BO}) &= P_D\big(Y \neq f_{BO}(X)\big) \\
&= \textstyle\sum_x P_D(Y \neq f_{BO}(x) \mid X = x)\, P_D(X = x) \\
&= \textstyle\sum_x (1 - P_D(Y = f_{BO}(x) \mid X = x))\, P_D(X = x) \\
&= \textstyle\sum_x \left(1 - \max_y P_D(Y = y \mid X = x)\right) P_D(X = x) \\
&= \mathrm{E}\left[1 - \max_y P_D(Y = y \mid X)\right]
\end{aligned}
$$

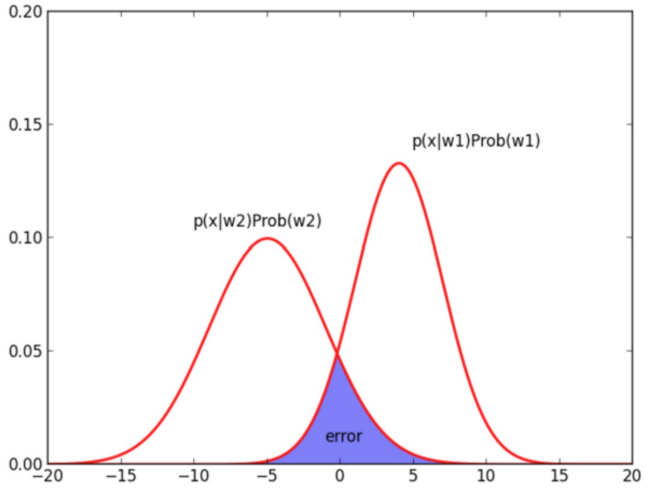




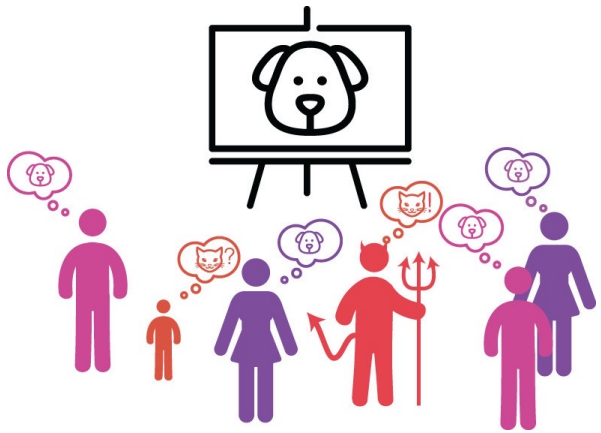
- Special case: binary classification
  - $L_D(f_{BO}) = \sum_x P_D(Y \neq f_{BO}(x), X = x)$
    $= \sum_x \min(\, P_D(Y = +1, X = x)\,, P_D(Y = -1, X = x))$

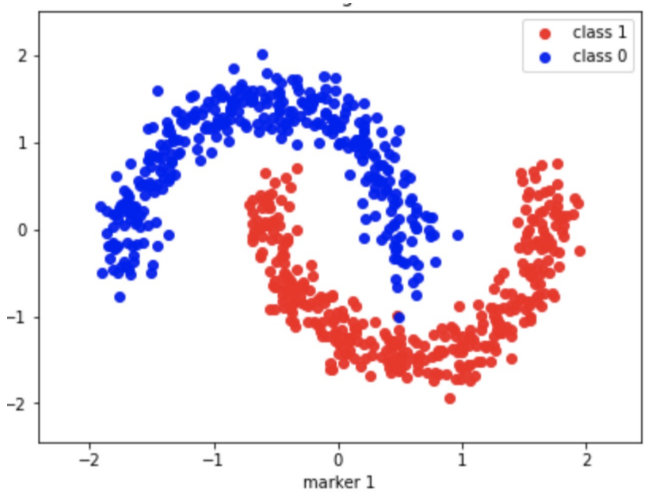
When is the Bayes error rate nonzero?

$$L_D(f_{BO}) = \sum_x \min(\, P_D(Y = +1, X = x)\,, P_D(Y = -1, X = x))$$



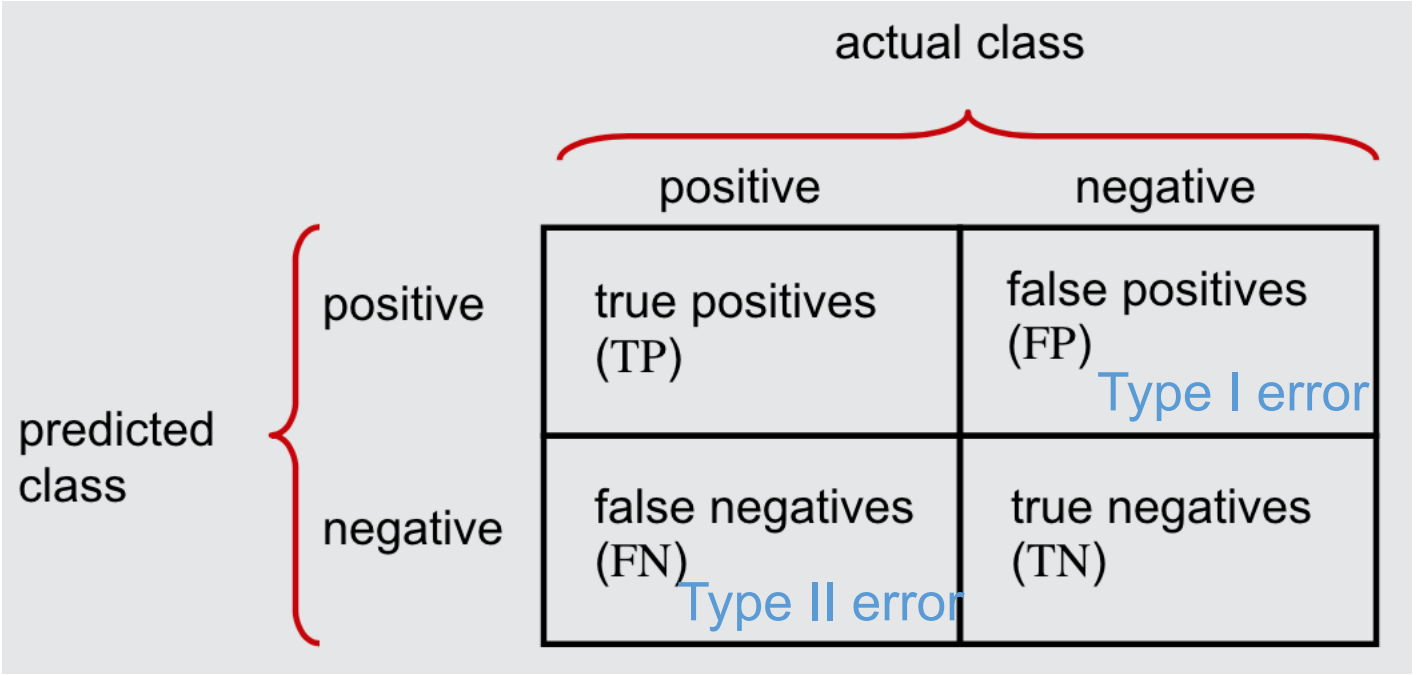
- Limited feature representation
- Noise in the training data
  - Feature noise
  - Label noise
  - Sensor failure
  - Typo in reviews for sentiment classification
- May not be a single “correct” answer
- Inductive bias of the model / learning algorithm





New measures of classification performance

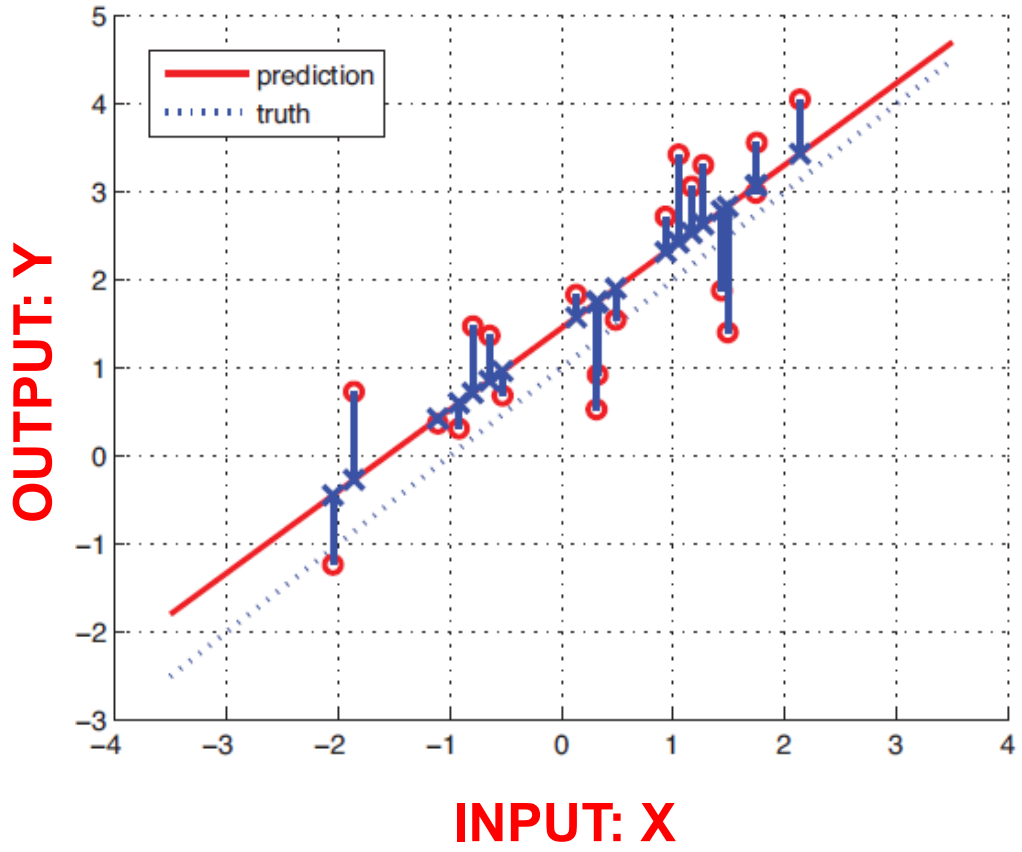
- True positive rate (TPR)
  $= \frac{\mathrm{TP}}{\mathrm{P}} = \frac{P(\hat{y}=+1, y=+1)}{P(y=+1)}$
  (aka recall, sensitivity)
- True negative rate (TNR) = $\frac{\mathrm{TN}}{\mathrm{N}}$
  (specificity)
- False positive rate (FPR) = $\frac{\mathrm{FP}}{\mathrm{N}}$
- False negative rate (FNR) = $\frac{\mathrm{FN}}{\mathrm{P}}$



P = TP + FN N = FP + TN

- Precision = $\frac{\mathrm{TP}}{\mathrm{P}-called} = \frac{P(\hat{y}=+1, y=+1)}{P(\hat{y}=+1)}$, $\mathrm{P} - called = \mathrm{TP} + \mathrm{FP}$

Linear Regression



Regression Learn a function that predicts outputs from inputs,

$$y = f(x)$$

Outputs y are real-valued

Linear Regression As the name suggests, uses a *linear function*:

$$y = w^T x + b$$

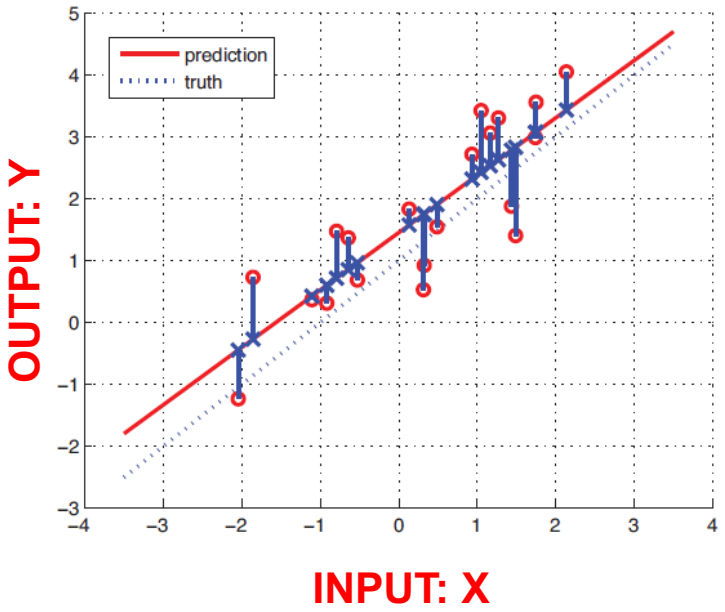
We will add noise later…

Linear Regression

Input-output mapping is not exact, so we will add zero-mean Gaussian noise,

Multivariate Normal (uncorrelated)

$$y = w^T x + \epsilon \quad \text{where} \quad \epsilon \sim \mathcal{N}(0, \sigma^2)$$



This is equivalent to the likelihood function,

$$p(y \mid w, x) = \mathcal{N}(y \mid w^T x, \sigma^2)$$

Because Adding a constant to a Normal RV is still a Normal RV,

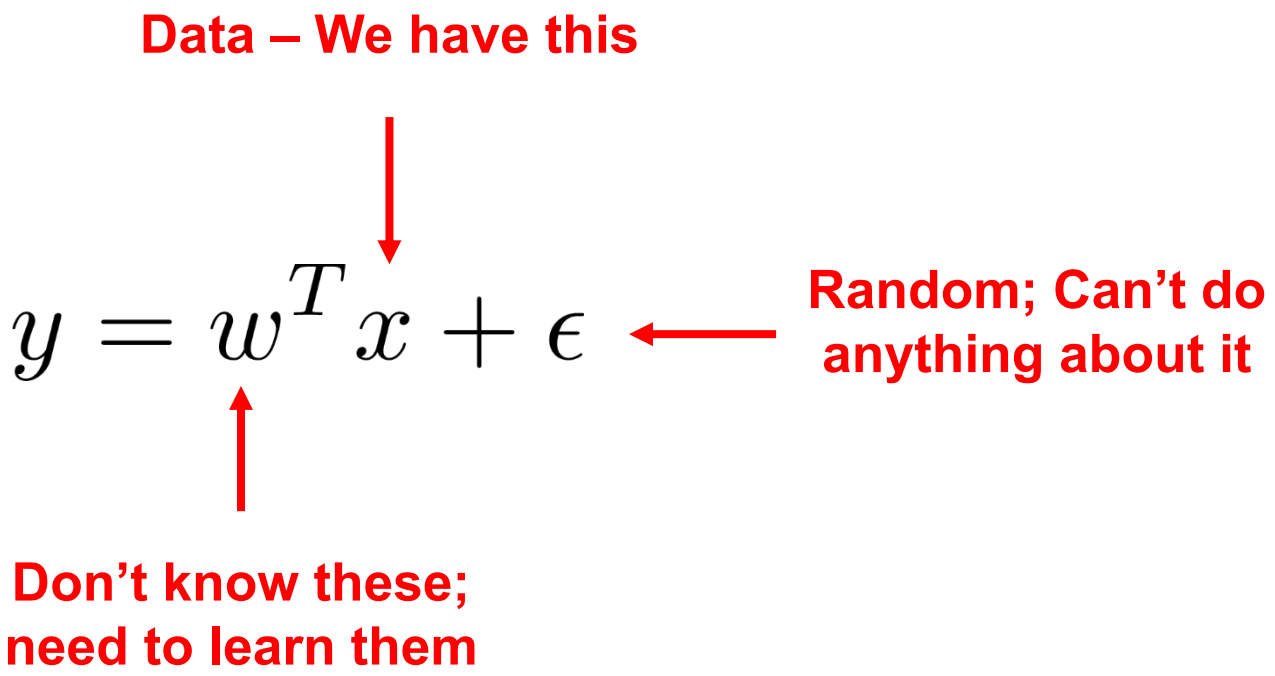
$$z \sim \mathcal{N}(m, P) \qquad z + c \sim \mathcal{N}(m + c, P)$$

In the case of linear regression $z \to \epsilon$ and $c \to w^T x$

Great, we're done right?

We need to fit it to data by learning the regression weights

How to do this? What makes *good* weights?



Learning Linear Regression Models

There are several ways to think about fitting regression:

- **Intuitive** Find a plane/line that is close to data
- **Functional** Find a line that minimizes the *least squares* loss
- **Estimation** Find maximum likelihood estimate of parameters

They are all the same thing…

Learning Linear Regression Models

There are several ways to think about fitting regression:

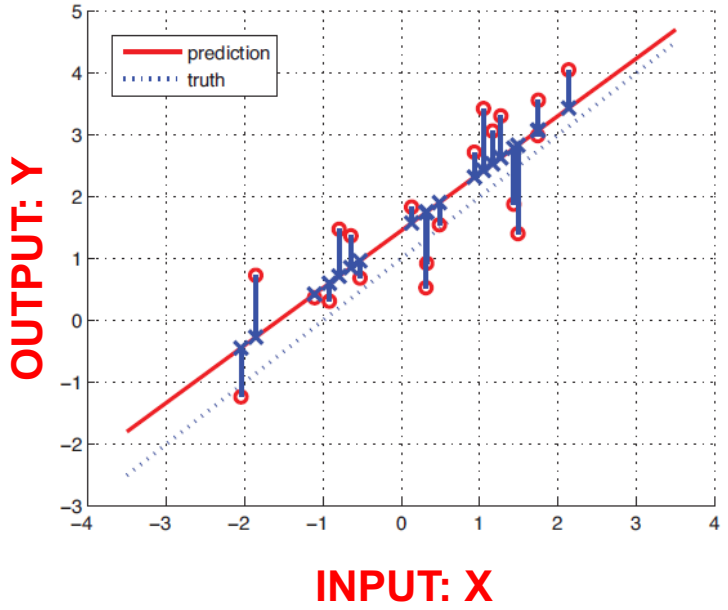
- **Intuitive** Find a plane/line that is close to data
- **Functional** Find a line that minimizes the *least squares* loss
- **Estimation** Find maximum likelihood estimate of parameters

They are all the same thing…

MLE for Linear Regression

Given training data $\{(x_i, y_i)\}_{i=1}^N$ likelihood function is given by,

$$\log \prod_{i=1}^{N} p(y_i \mid x_i, w) = \sum_{i=1}^{N} \log p(y_i \mid x_i, w)$$



Recall that the likelihood is Gaussian:

$$p(y \mid w, x) = \mathcal{N}(y \mid w^T x, \sigma^2)$$

So MLE maximizes the log-likelihood over the whole data as,

$$w^{\mathrm{MLE}} = \arg\max_w \sum_{i=1}^{N} \log \mathcal{N}(y_i \mid w^T x_i, \sigma^2)$$

MLE of Gaussian Mean

Assume data are i.i.d. univariate Gaussian,

$$p(\mathcal{Y} \mid \mu) = \prod_{i=1}^{N} \mathcal{N}(y_i \mid \mu, \sigma^2)$$

Variance is known

Log-likelihood function:

$$\mathcal{L}(\mu) = \sum_{i=1}^{N} \log \left(\frac{1}{\sqrt{2\pi\sigma^2}} \exp \left(-\frac{1}{2}(y_i - \mu)^2 \sigma^{-2} \right) \right)$$

Constant doesn't depend on mean

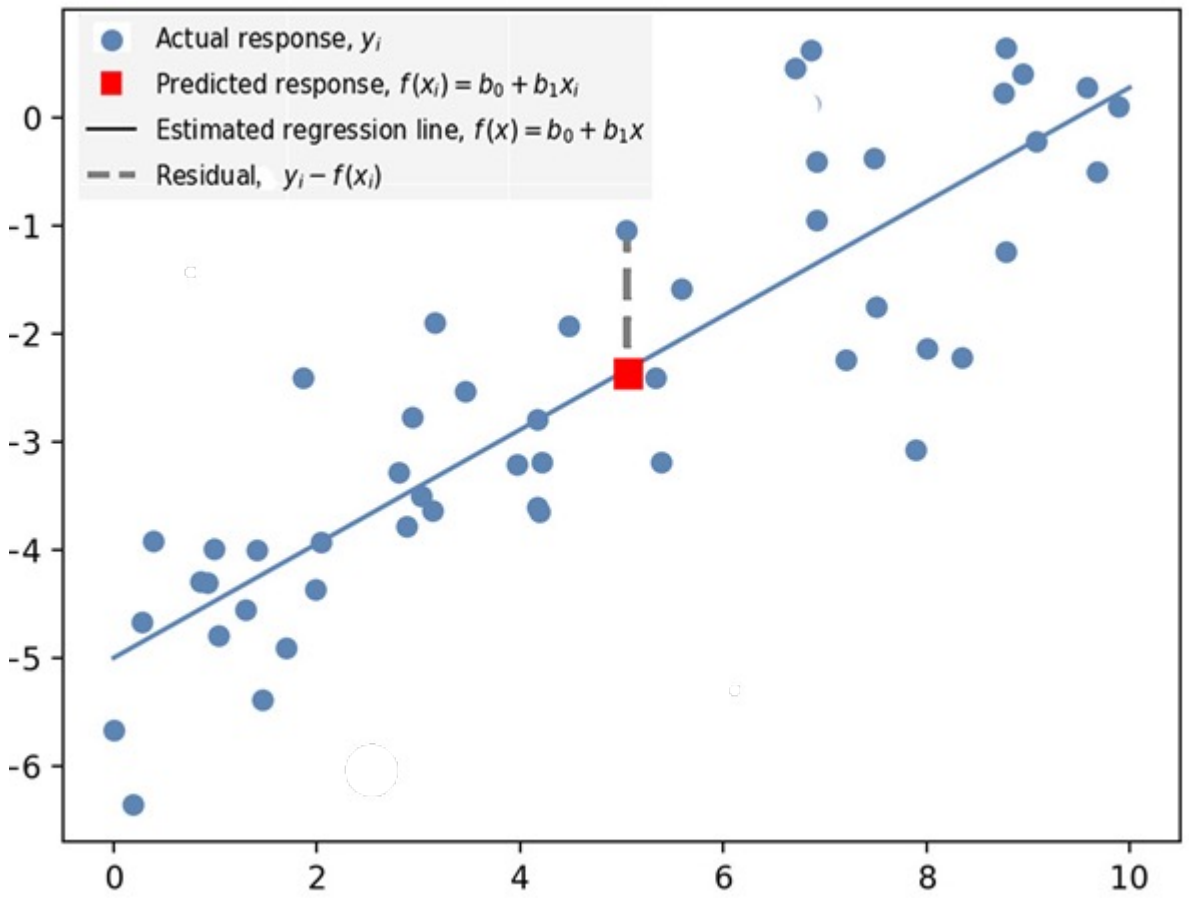
$$= \text{const.} - \frac{1}{2} \sum_{i=1}^{N} \left((y_i - \mu)^2 \sigma^{-2} \right)$$

MLE doesn't change when we:
1) Drop constant terms (in μ)
2) Minimize negative log-likelihood

MLE estimate is *least squares estimator*:

$$\mu^{\text{MLE}} = -\frac{1}{2\sigma^2} \arg\max_{\mu} \sum_{i=1}^{N} (y_i - \mu)^2 = \arg\min_{\mu} \sum_{i=1}^{N} (y_i - \mu)^2$$

MLE of Linear Regression



Substitute linear regression prediction into MLE solution and we have,

$$\min_w \sum_{i=1}^{N} (y_i - wx_i)^2$$

So for Linear Regression, MLE = Least Squares Estimation

https://www.activestate.com/resources/quick-reads/how-to-run-linear-regressions-in-python-scikit-learn/

MLE of Linear Regression

Using previous results, MLE is equivalent to minimizing squared residuals,

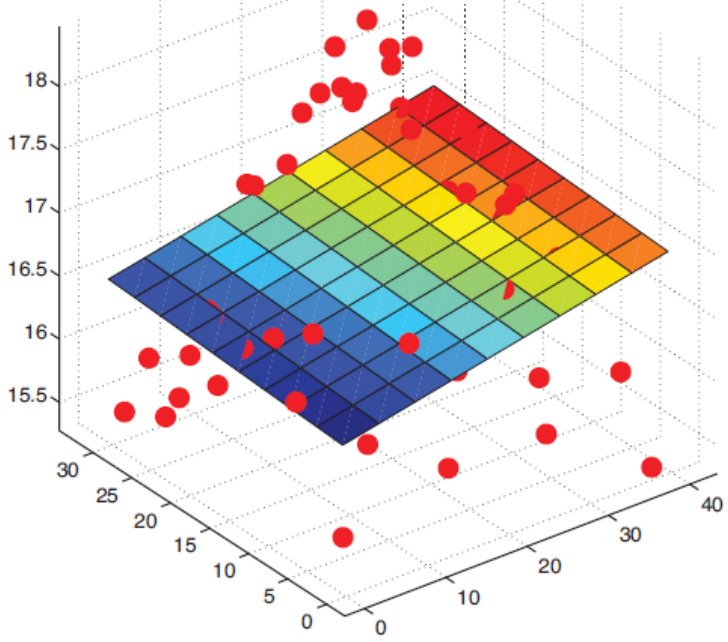
$$\min_w \sum_{i=1}^{N} (y_i - w^T x_i)^2 = \|\mathbf{y} - w^T \mathbf{X}\|^2$$

Some slightly more advanced linear algebra gives us a solution,

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

Ordinary Least Squares (OLS) solution

[Image: Murphy, K. (2012)]



Derivation a bit involved for lecture but…

- We know it has a closed-form and why
- We can evaluate it
- Generally know where it comes from

Basis Functions

- A **basis function** can be any function of the input features X
- Define a set of m basis functions $\phi_1(x), \ldots, \phi_m(x)$
- Fit a linear regression model in terms of basis functions,

$$y = \sum_{i=1}^{m} w_i \phi_i(x) = w^T \phi(x)$$

- Regression model is *linear in the basis transformations*
- Model is *nonlinear in the data X*

Kernel Functions

*A **kernel function** is an inner-product of some basis function computed on two inputs*

$$k(x, x') = \boldsymbol{\phi}(x)^{\mathrm{T}} \boldsymbol{\phi}(x') = \sum_{i=1}^{M} \phi_i(x)\phi_i(x')$$

A consequence is that kernel functions are non-negative real-valued functions over a pair of inputs,

$$\kappa(x, x') \in \mathbb{R} \qquad \kappa(x, x') \geq 0$$

Kernel functions can be interpreted as a measure of distance between two inputs

Kernel Functions

Example The *linear basis* $\phi(x) = x$ produces the kernel,

$$\kappa(x, x') = \phi(x)^T \phi(x') = x^T x'$$

It is often easier to directly specify the kernel rather than the basis function…

Example Gaussian kernel models similarity according to an unnormalized Gaussian distribution,

$$\kappa(x, x') = \exp\left(-\frac{1}{2\sigma^2}(x - x')^2\right)$$

Note Despite the name, this is **not** a Gaussian probability density.

Also called a *radial basis function* (RBF)

Kernel Functions

Given *any* set of data $\{x_i\}_{i=1}^n$ a necessary and sufficient condition of a valid kernel function is that the nxn **gram matrix**,

$$\mathbf{K} = \begin{pmatrix} \kappa(x_1, x_1) & \kappa(x_1, x_2) & \dots & \kappa(x_1, x_n) \\ \kappa(x_2, x_1) & \kappa(x_2, x_2) & \dots & \kappa(x_2, x_n) \\ \vdots & \vdots & \vdots & \vdots \\ \kappa(x_n, x_1) & \kappa(x_n, x_2) & \dots & \kappa(x_n, x_n) \end{pmatrix}$$

Is a *symmetric positive semidefinite matrix.*

Kernel Ridge Regression

Kernel representation requires inversion of NxN matrix

Primal

$$\mathbf{\Phi} = \begin{pmatrix} 1 & \phi_1(x_1) & \dots & \phi_M(x_1) \\ 1 & \phi_1(x_2) & \dots & \phi_M(x_2) \\ \vdots & \vdots & \vdots & \vdots \\ 1 & \phi_1(x_N) & \dots & \phi_M(x_N) \end{pmatrix}$$

$$w = (\mathbf{\Phi}^T\mathbf{\Phi} + \lambda I)^{-1}\mathbf{\Phi}^T\mathbf{y}$$

MxM Matrix Inversion
O(M³)

Dual

$$\mathbf{K} = \begin{pmatrix} \kappa(x_1, x_1) & \kappa(x_1, x_2) & \dots & \kappa(x_1, x_n) \\ \kappa(x_2, x_1) & \kappa(x_2, x_2) & \dots & \kappa(x_2, x_n) \\ \vdots & \vdots & \vdots & \vdots \\ \kappa(x_n, x_1) & \kappa(x_n, x_2) & \dots & \kappa(x_n, x_n) \end{pmatrix}$$

$$y(x) = \mathbf{k}(\mathbf{x})^T(\mathbf{K} + \lambda I)^{-1}\mathbf{y}$$

NxN Matrix Inversion
O(N³)

Number of training data N greater than basis functions M