CSC 480/580: Principles of Machine Learning

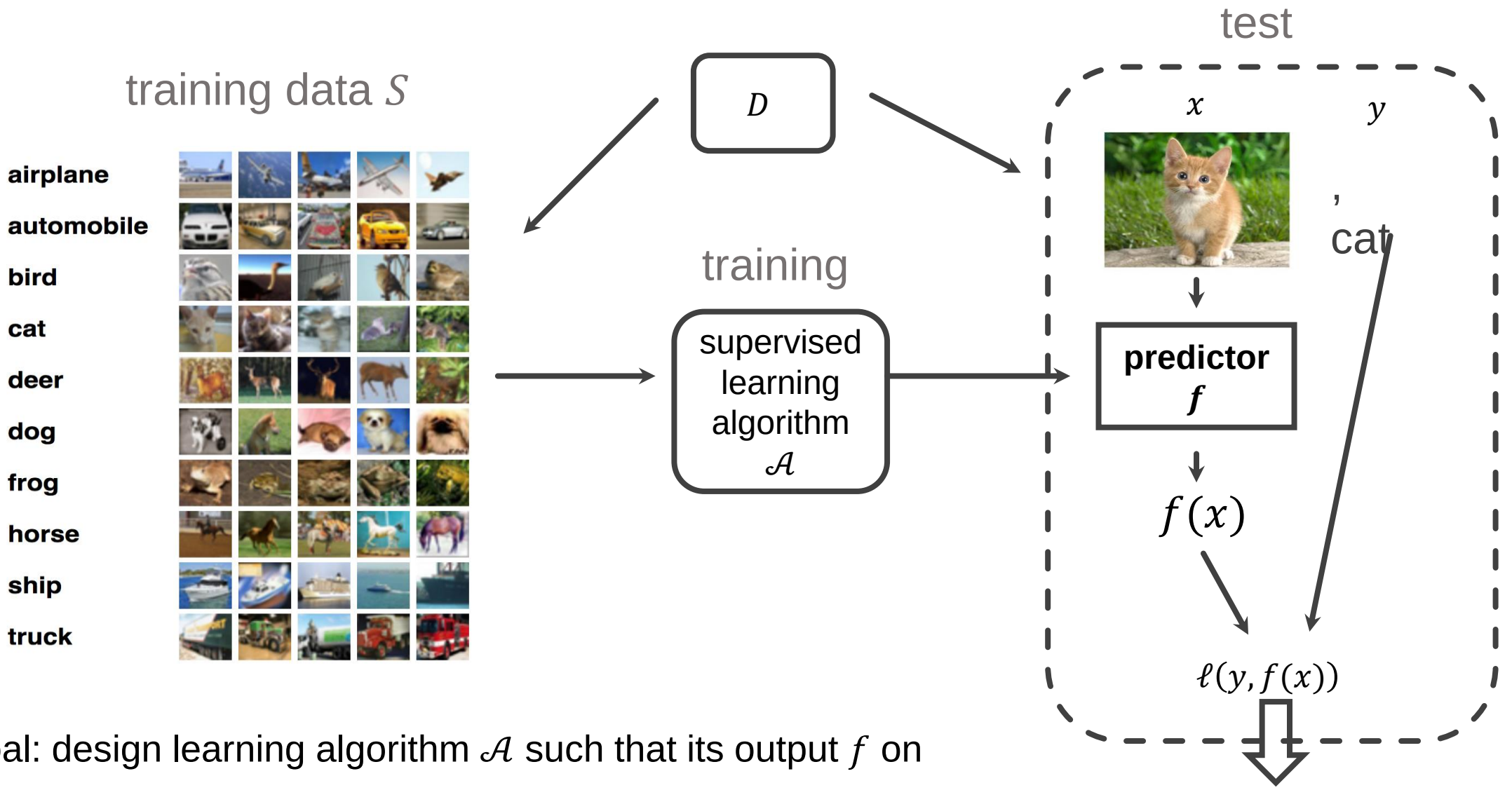
Midterm Review

Chicheng Zhang

Midterm

- Mar 12, in class (9:30-10:45am)
- About 6 questions
- You can bring a A4-size note
  - The process of preparing for the note is a good way to organize your knowledge

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

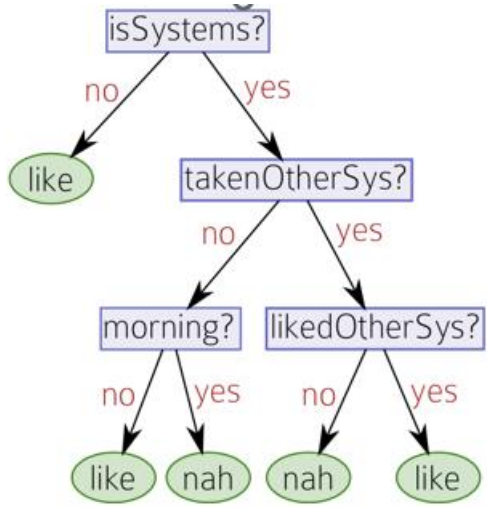
Generalization error: $L_D(f) = \mathrm{E}_{(x,y)\sim D}\, \ell(y, f(x))$

Decision trees

- Test: predict using a decision tree:

Algorithm 2 DecisionTreeTest(*tree*, *test point*)

```
1: if tree is of the form Leaf(guess) then
2:     return guess
3: else if tree is of the form Node(f, left, right) then
4:     if f = no in test point then
5:         return DecisionTreeTest(left, test point)
6:     else
7:         return DecisionTreeTest(right, test point)
8:     end if
9: end if
```



guess=prediction

left=no
right=yes

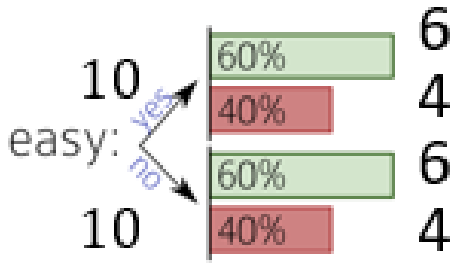
- Training: how to design a learning algorithm $\mathcal{A}$ that can build trees f from training data?

Algorithm 1 DecisionTreeTrain(*data*, *remaining features*)

```
guess ← most frequent answer in data                    // default answer for this data
if the labels in data are unambiguous then
    return Leaf(guess)                                  // base case: no need to split further
else if remaining features is empty then
    return Leaf(guess)                                  // base case: cannot split further
else                                                    // we need to query more features
    for all f ∈ remaining features do
        NO ← the subset of data on which f=no
        YES ← the subset of data on which f=yes
        score[f] ← # of majority vote answers in NO
                   + # of majority vote answers in YES
                                                        // the accuracy we would get if we only queried on f
    end for
    f ← the feature with maximal score(f)
    NO ← the subset of data on which f=no
    YES ← the subset of data on which f=yes
    left ← DecisionTreeTrain(NO, remaining features \ {f})
    right ← DecisionTreeTrain(YES, remaining features \ {f})
    return Node(f, left, right)
end if
```

answer=label

unambiguous=achieves 100% ac

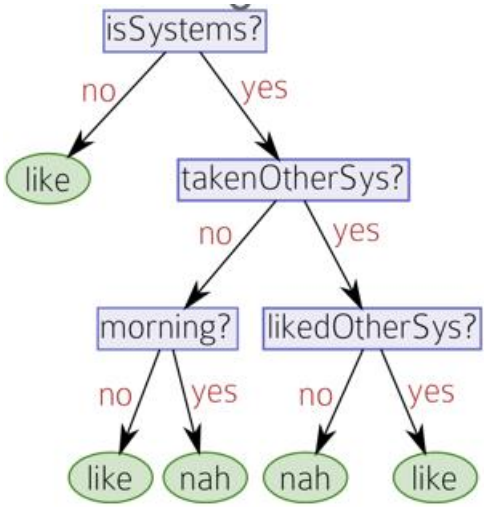


Score(f, S) = "informativeness of f (in predicting y) for dataset S"

Q: is this algorithm guaranteed to terminate?

Example questions

- Given a decision tree and a test example, find out the tree's prediction on the example



- Calculate the information score of a feature

$$\text{Score}(S, f) = u(S) - \left(p_L\, u(S_L) + p_R\, u(S_R)\right)$$

  - See. E.g. HW1, Problem 2

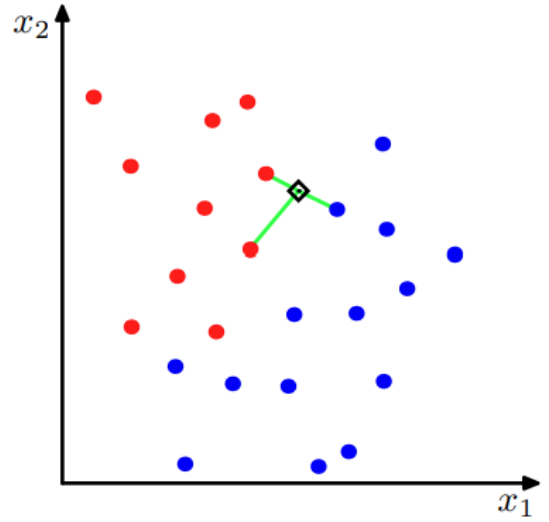
k-nearest neighbors (k-NN): main concept

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

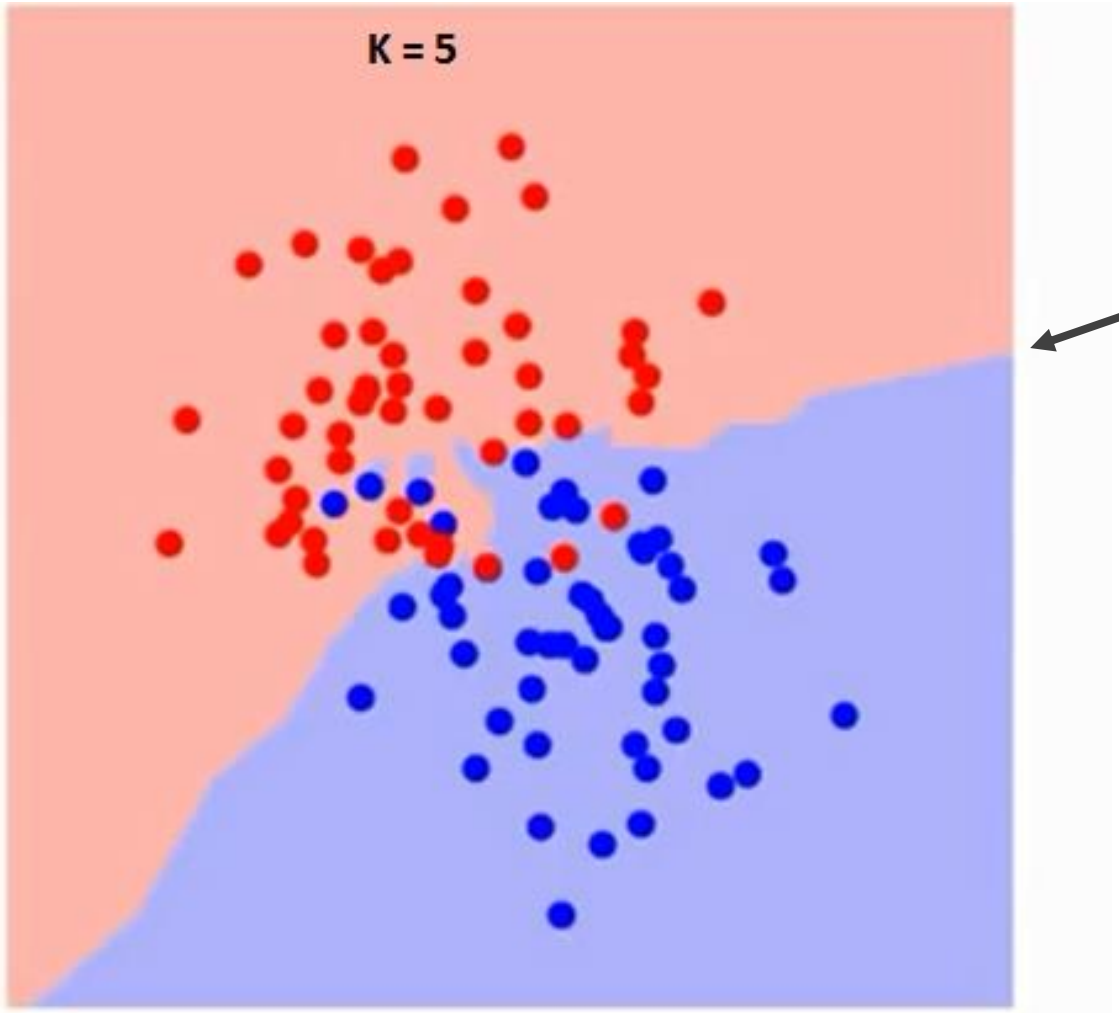
Inductive bias: given test example x, its label should resemble the labels of **nearby points**

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

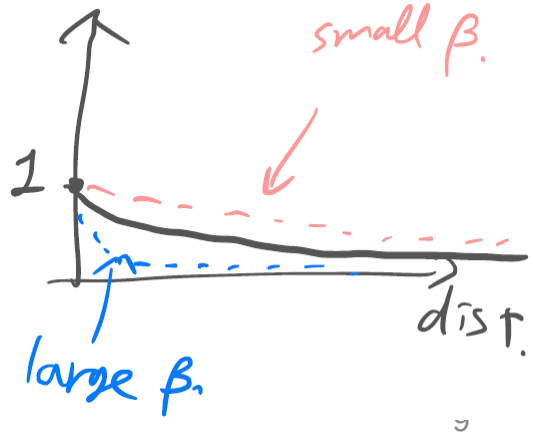


decision boundary

Variations

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

    where $w_i \propto \exp(-\beta \, d(x, x_i))$, or $\propto \frac{1}{1+d(x,x_i)^{\beta}}$
  - Quick question: is a larger β resembling a larger or smaller k?

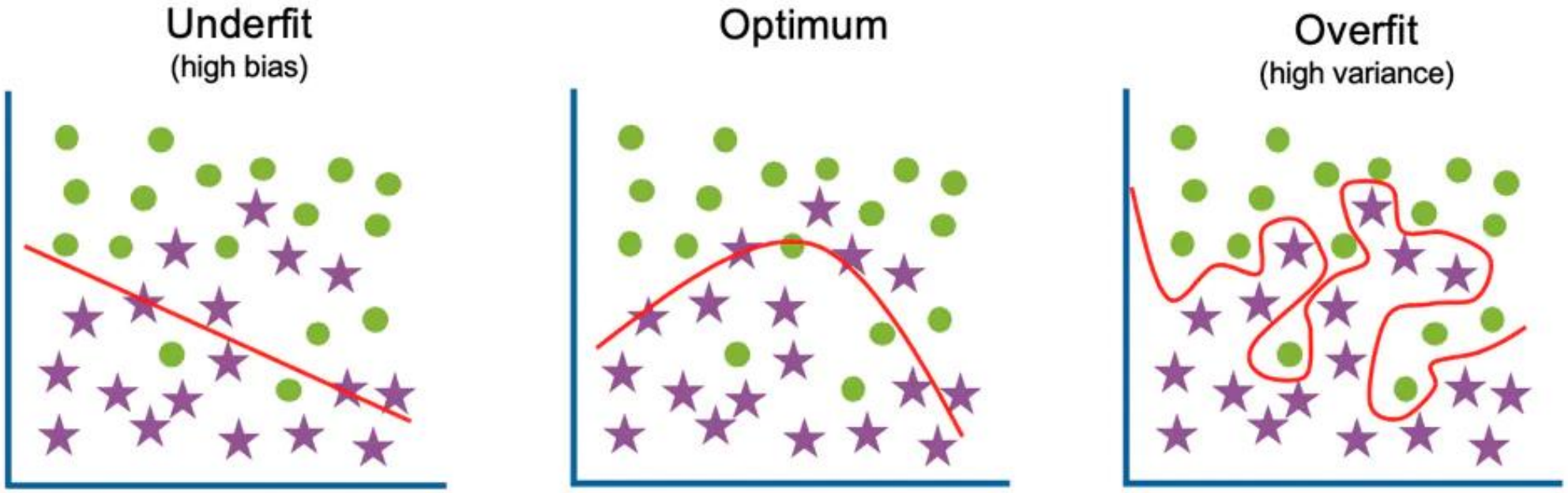


Example question

- Given a dataset (say, of size 5), compute its induced k-nearest neighbor prediction on a test example

- Given a training dataset, compute the training error of its induced 3-nearest neighbor classifier

Overfitting vs Underfitting



Source: ibm.com

Example question

- Give three scenarios of underfitting / overfitting when training a classifier
  - E.g. k-nearest neighbor, decision tree, logistic regression, training linear classifier using Perceptron

- How can we combat underfitting / overfitting?

Bayes optimal classifier

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

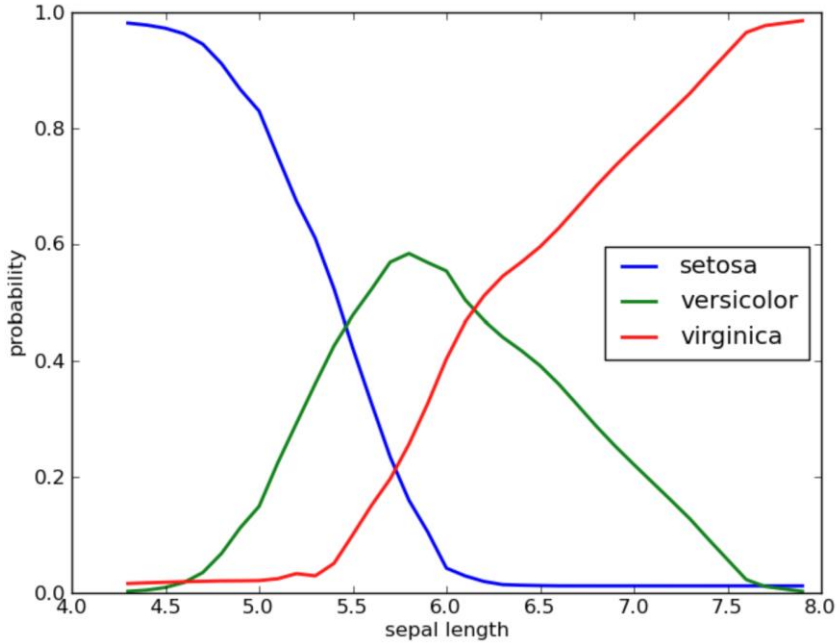
$$f_{BO}(x) = \arg\max_{y \in \mathcal{Y}} P_D(X = x, Y = y) = \arg\max_{y \in \mathcal{Y}} P_D(Y = y \,|X = x), \forall x \in \mathcal{X}$$

Example Iris dataset classification:

Iris Setosa

Iris Versicolor

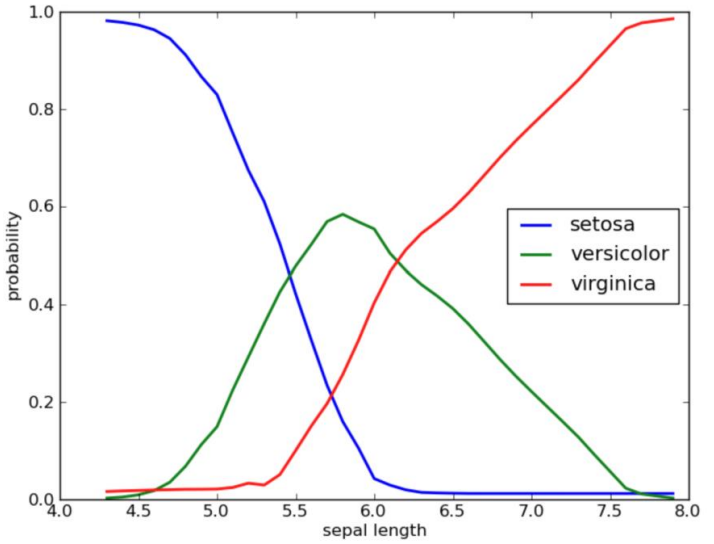
Iris Virginica



Bayes error rate: alternative form

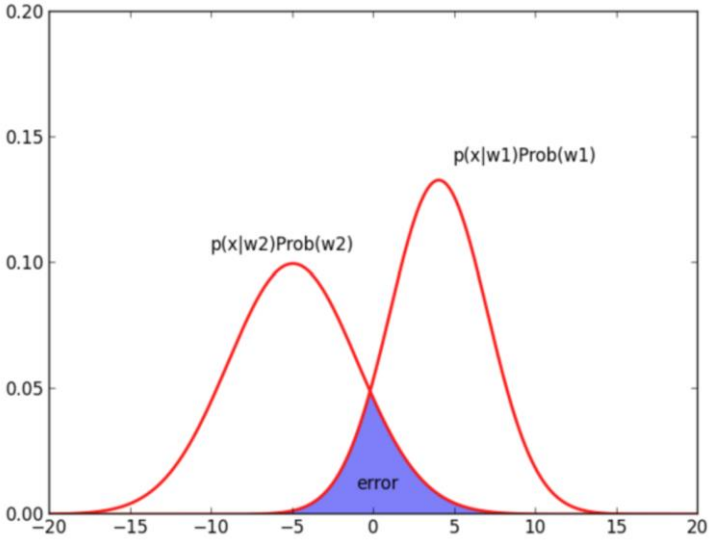
$$L_D(f_{BO}) = \sum_x \left(1 - \max_y P_D(Y = y \mid X = x)\right) P_D(X = x)$$

$$= \mathrm{E}\left[1 - \max_y P_D(Y = y \mid X)\right]$$



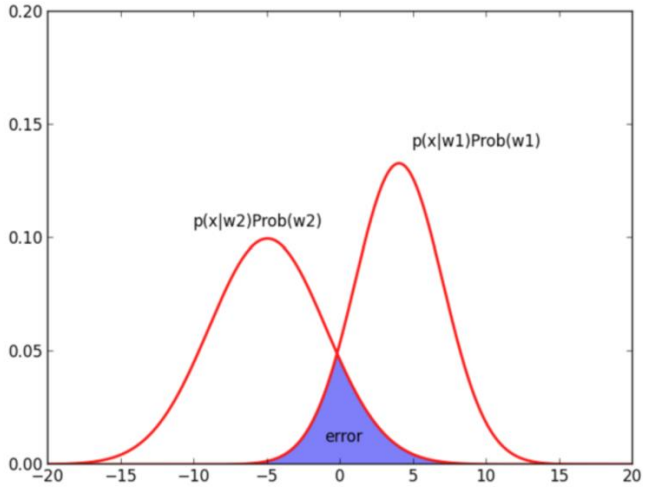
- Special case: binary classification

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

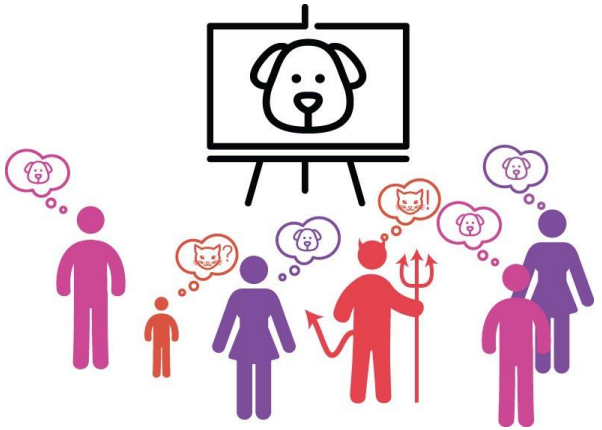


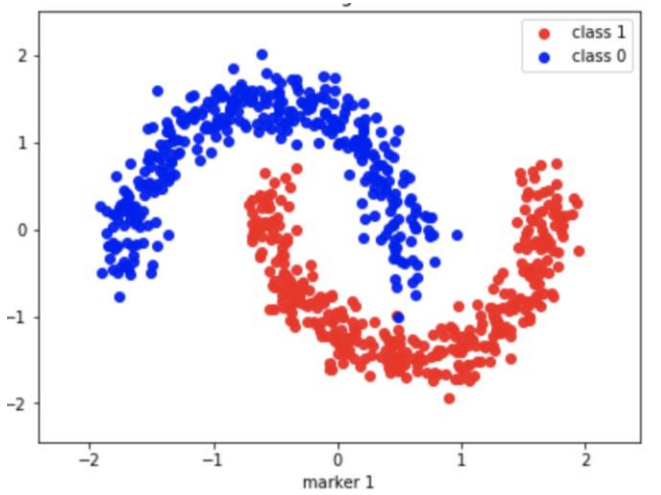
Example question

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



- Give two real-world example data distributions whose Bayes errors are nonzero
  - Limited feature representation
  - Noise in the training data
  - …
  - (see slides)



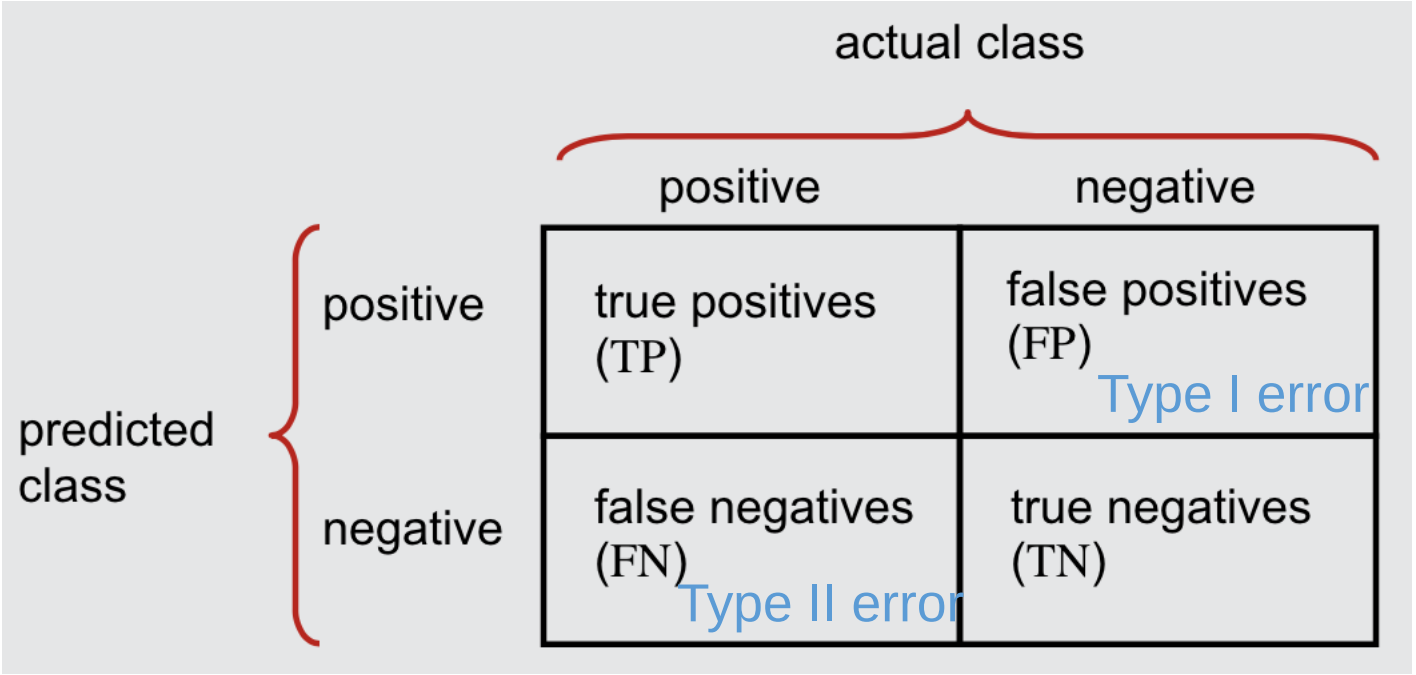


Example question

- Given a data distribution D over (x, y), compute the error rate of a classifier & $D's$ Bayes error rate
  - See HW1, problem 1

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

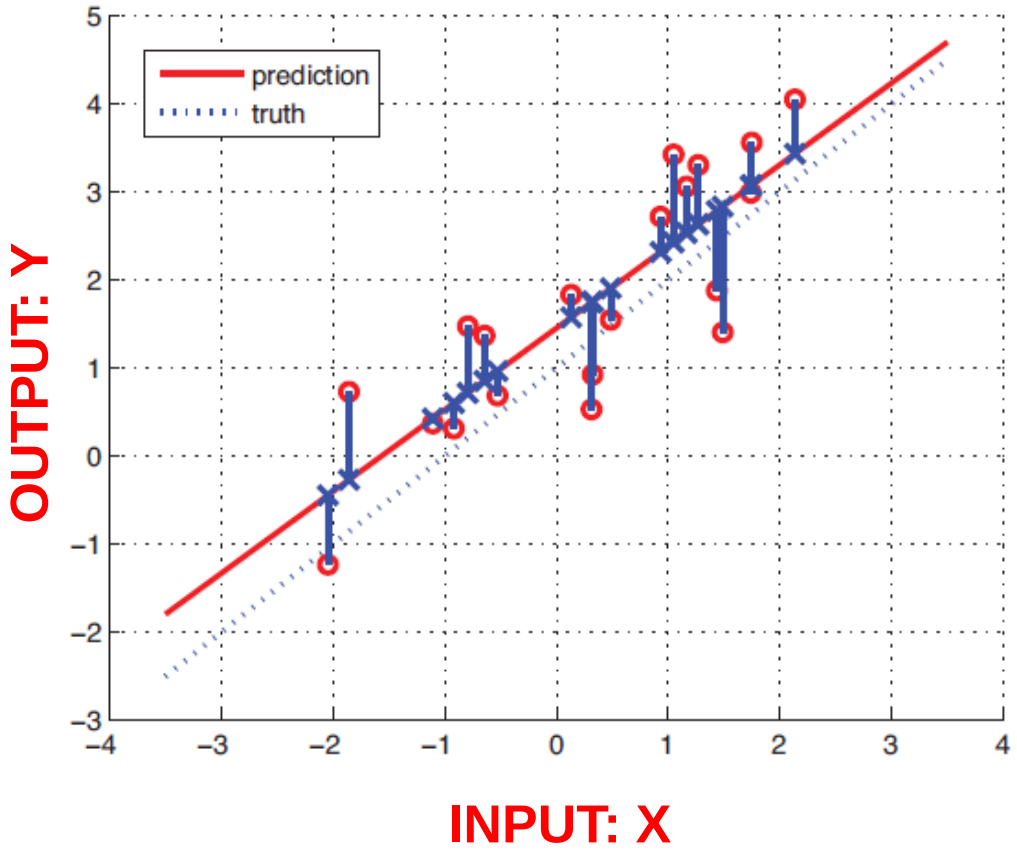
- Precision = $\frac{\mathrm{TP}}{\mathrm{P}-called} = \frac{P(\hat{y}=+1, y=+1)}{P(\hat{y}=+1)}$, $\mathrm{P} - called = \mathrm{TP} + \mathrm{FP}$

Example question

- Given a small binary classification dataset (say of size 5) and a predictor's prediction score on it; compute the ROC curve
- What are the x and y values in the ROC curve?
- X: FPR
- Y: TPR

$c(x_i)$	y_i
.99	+
.98	+
.72	-
.51	-
.24	+

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*:

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

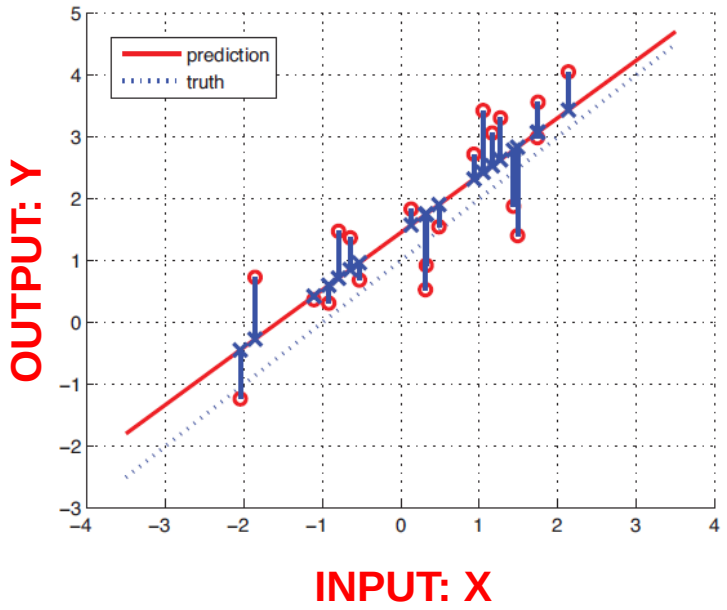
We will add noise later…

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 $c \to w^T x$

Learning linear regression models

We need to learn the model from data
by learning the regression weights

Data – We have this

$$y = w^T x + \epsilon$$

Random; Can't do anything about it

Don't know these; need to learn them

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

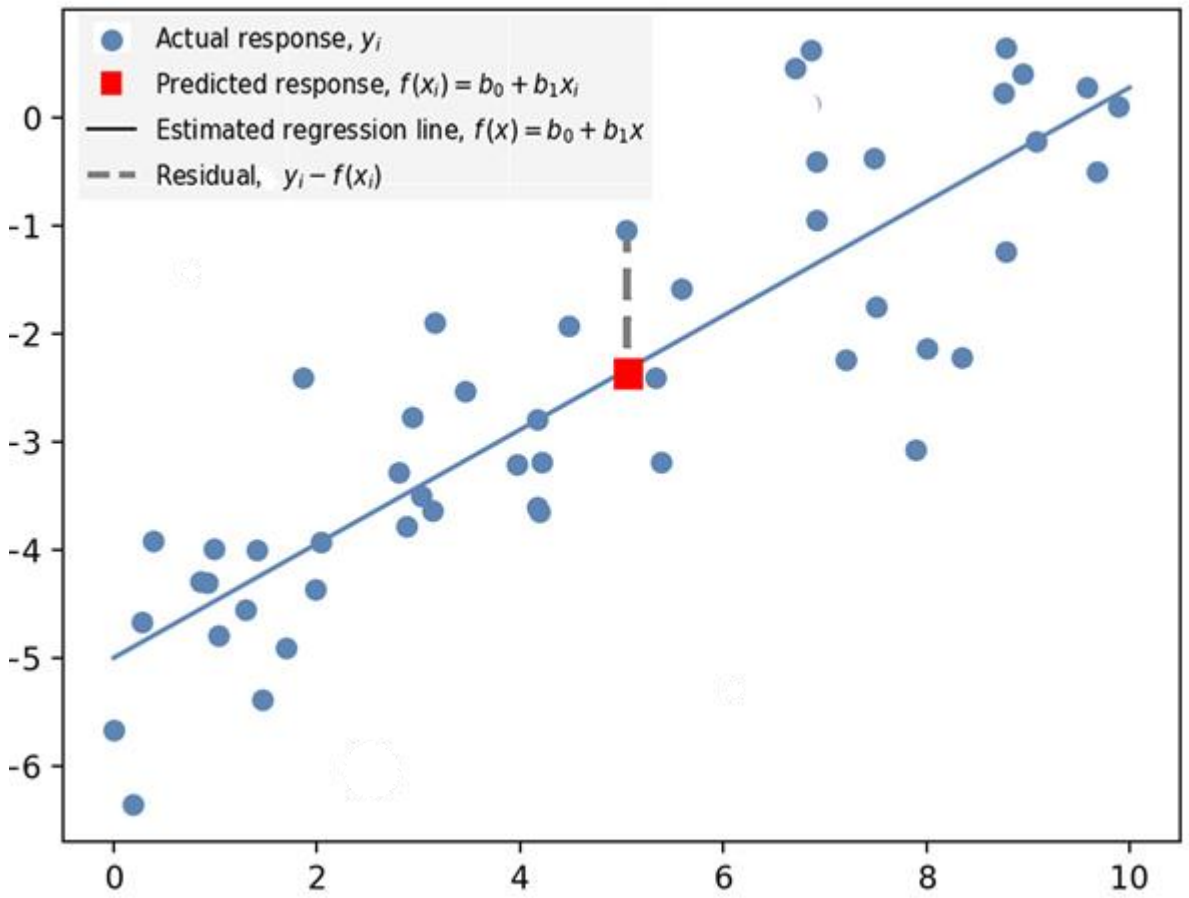
Learning Linear Regression Models

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…

MLE for 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,

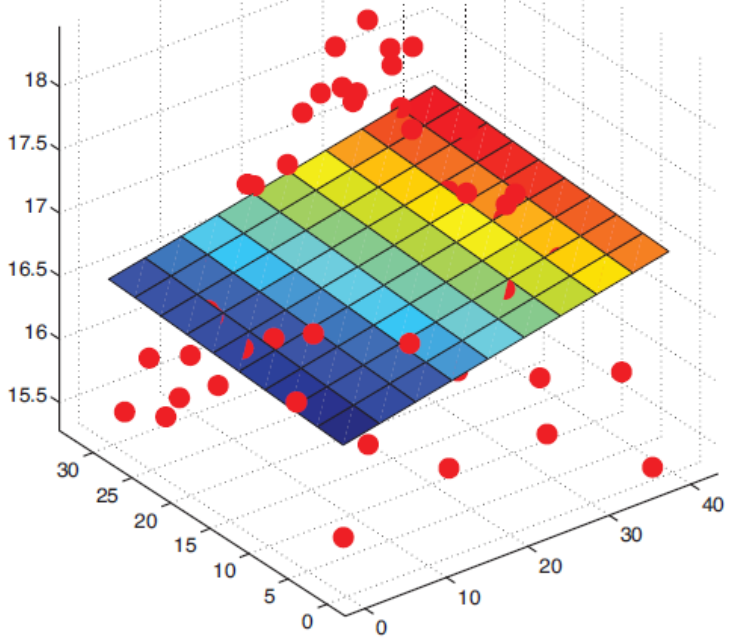
$$\min_w \sum_{i=1}^{N} (y_i - w^T x_i)^2 = \|\mathbf{y} - w^T \mathbf{X}\|^2$$

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

Example question

- Given a small dataset (say, size 5):
  - Write down the optimization problem least square regression solves
  - Be able to write down the derivative of the objective wrt w
  - Compute the least square linear regression coefficient

- Given a small dataset, write down the optimization problem logistic regression solves

$$w^{\mathrm{MLE}} = \arg\max_{w} \sum_{i} \left\{ y_i w^T x_i - \log\left(1 + e^{w^T x_i}\right) \right\}$$

  - How about SVM?

HW1: additional remarks

- P3

(b) Implement the decision tree in Python as described in the book (handles only the binary features) but use the entropy instead of the classification error. Implement an option of `max_depth` so the trained tree will have depth at most `max_depth` (in our case, it corresponds to only considering at most `max_depth` features). Make sure to email your code to `csc580homeworks@gmail.com` so that I can run it.

Use the data in the book (Table 1) while taking the rating 2/1/0 as positive and -1/-2 as negative. Train your decision tree with your code with `max_depth` = 2. Report your tree along with the following information (in whatever form a person can reasonably comprehend)

- What are the branching questions at each node?
- What are the uncertainty scores at each node?
- Show the predicted label for each leaf node.

- Sanity checks:
  - Ensure that max_depth=2 feature is implemented
  - Does your root node's feature have a high information score?

HW1: additional remarks

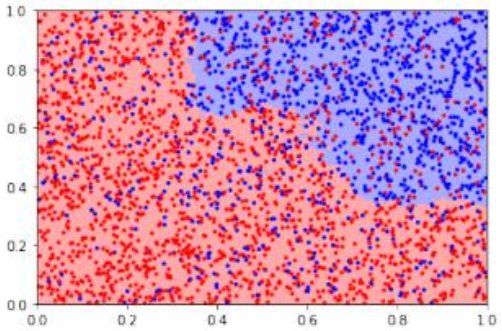
- P4

Let us define the data distribution $\mathcal{D}$ consisting of two-dimensional features $X \in \mathbb{R}^2$. Each dimension of $X = (X_1, X_2)^T$ is uniformly distributed on the unit interval: $X_1 \sim \text{Uniform}[0,1]$ and $X_2 \sim \text{Uniform}[0,1]$. Let $i(X)$ be 1 if $X_1 < 1/3$, 2 if $X_1 \in [1/3, 2/3)$, and 3 if $X_1 \geq 2/3$. Define $j(X)$ similarly for the second dimension X_2 (i.e., replace X_1 above by X_2). Furthermore, the labels are binary with $\mathbb{P}(Y = 1 \mid X = x) = A_{i(x),j(x)}$ and,

$$A = \begin{pmatrix} .1 & .2 & .2 \\ .2 & .4 & .8 \\ .2 & .8 & .9 \end{pmatrix}$$

($A_{i,j} \in \mathbb{R}$ is the entry of matrix A at i-th row, j-th column) Throughout, we abuse notation and use $\mathbb{P}$ for both probability of events and the density function for continuous random variables.

- (a) Bayes optimal classifier
  - Should be an indicator function of a subset of $[0,1]^2$
  - See solution guide



HW1: additional remarks

(b) Plot the *learning curve* for nearest-neighbor classification. Let $k = 4$. Define $\mathcal{M} = \{10, 30, 100, 300, 1000, 3000\}$. Call the following one 'trial':

- Draw 3000 fresh data points from $\mathcal{D}$ and call it S.
- Then, for each $m \in \mathcal{M}$, choose the first m data points from S, train a k-NN classifier with them, and then evaluate its test set error.

Perform 5 trials, compute the average test set error, and report the plot of 'test error rate' vs m. In the same plot, plot a horizontal line that shows the Bayes error rate so we know how close we get to the Bayes error rate.

- Sanity check:
  - The test error of your k-NN classifiers should never be lower than Bayes error rate (why?)

Backup

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
$O(M^3)$

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
$O(N^3)$

Number of training data N greater than basis functions M

k-NN classification: pseudocode

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

list →
append to list →
sort in first coordinate →
Majority vote of $\{y_i : i \in N(x)\}$ →

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

```
S ← [ ]
for n = 1 to N do
    S ← S ⊕ ⟨d(x_n, x̂), n⟩        // store distance to training example n
end for
S ← SORT(S)                        // 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)                     // return +1 if ŷ > 0 and −1 if ŷ < 0
```

- Time complexity (assuming distance calculation takes $O(d)$ time)
  - $O(m\,d\ + m\log m\ + k\ ) = O\big(m(d\ + \log m)\big)$

Inductive Bias

Training

Test

How would you label the test examples?