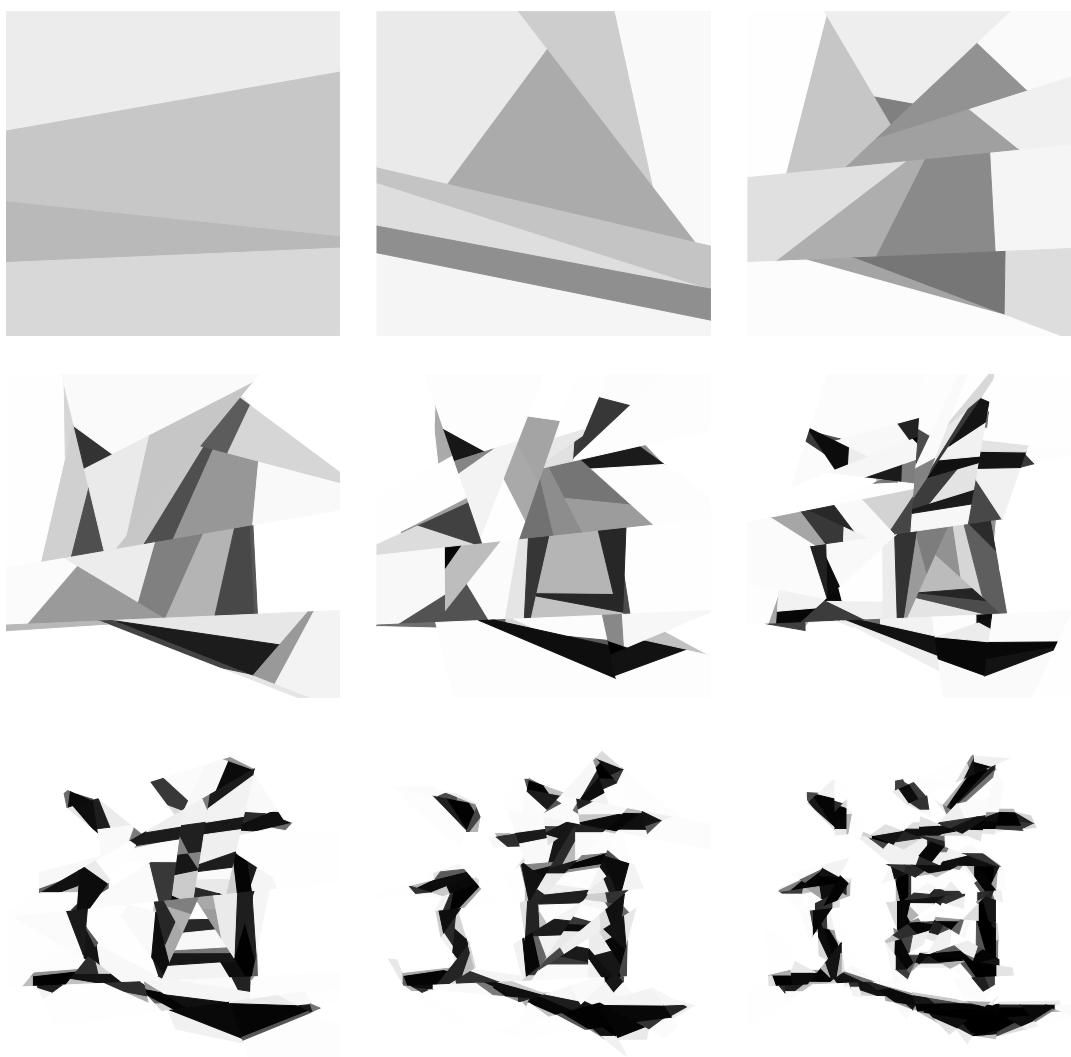


Lecture Notes on Machine Learning

Miguel Á. Carreira-Perpiñán
Dept. of CSE, University of California, Merced

December 7, 2023



These are notes for a one-semester undergraduate course on machine learning given by Prof. Miguel Á. Carreira-Perpiñán at the University of California, Merced.

These notes may be used for educational, non-commercial purposes.

©2015–2023 Miguel Á. Carreira-Perpiñán

1 Introduction

What is machine learning (ML)?

- Data is being produced and stored continuously (“big data”):
 - science: genomics, astronomy, materials science, particle accelerators...
 - sensor networks: weather measurements, traffic...
 - people: social networks, blogs, mobile phones, purchases, bank transactions...
 - etc.
- Data is not random; it contains structure that can be used to predict outcomes, or gain knowledge in some way.

Ex: patterns of Amazon purchases can be used to recommend items.
- It is more difficult to design algorithms for such tasks (compared to, say, sorting an array or calculating a payroll). Such algorithms need data.

Ex: construct a spam filter, using a collection of email messages labelled as spam/not spam.
- Explicit vs implicit programming:
 - Ex: write a program to sort an array of n numbers. A competent computer scientist can think hard and devise a specific algorithm (say, Quicksort), understand why the algorithm will work and program it in a few lines. This is *explicit programming*.
 - Ex: write a program to tell whether an image of $m \times n$ pixels contains a dog or not. Very hard to write this as an explicit program, and it would not achieve a high classification rate because of the complex variability of dog images. Instead: define an objective function (say, classification error) and a classifier with adjustable parameters (say, a neural network) and adjust the parameters (train or optimize the neural network) so the objective is as best as possible on a training set of labeled images. Properly done, this will achieve a much better classification on images beyond the training ones, although we may not understand how the neural net works internally. This is *implicit programming*, and it is what ML does.
- Data mining: the application of ML methods to large databases.
- Ex of ML applications: fraud detection, medical diagnosis, speech or face recognition...
- ML is programming computers using data (past experience) to optimize a performance criterion.
- ML relies on:
 - Statistics: making inferences from sample data.
 - Numerical algorithms (linear algebra, optimization): optimize criteria, manipulate models.
 - Computer sci.: data structures/programs/hardware that solve a ML problem efficiently.
- A model:
 - is a compressed version of a database;
 - extracts knowledge from it;
 - does not have perfect performance but is a useful approximation to the data.

Examples of ML problems

- *Supervised learning*: labels provided.

- *Classification* (pattern recognition):

- * Face recognition. Difficult because of the complex variability in the data: pose and illumination in a face image, occlusions, glasses/beard/make-up/etc.

Training examples:



Test images:

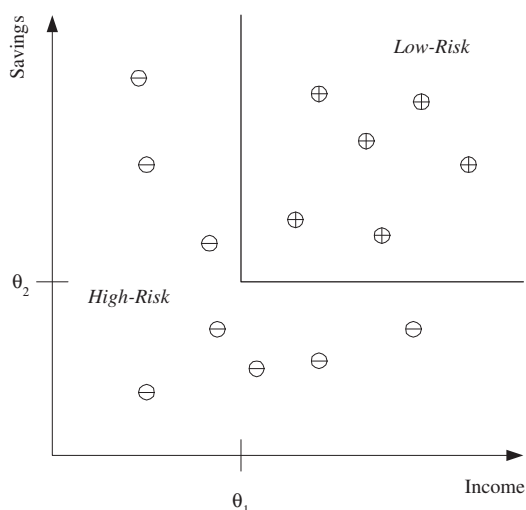


- * Optical character recognition: different styles, slant... 0 1 2 3 4 5 6 7 8 9
- * Medical diagnosis: often, variables are missing (tests are costly).
- * Speech recognition, machine translation, biometrics...
- * Credit scoring: classify customers into high- and low-risk, based on their income and savings, using data about past loans (whether they were paid or not).

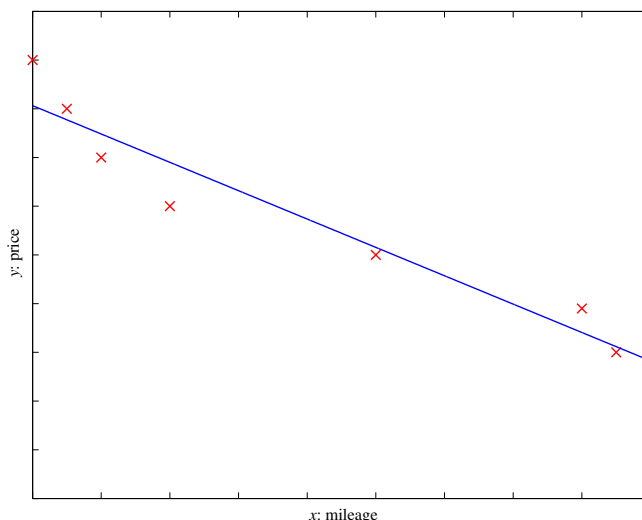
- *Regression*: the labels to be predicted are continuous:

- * Predict the price of a car from its mileage.
- * Navigating a car: angle of the steering.
- * Kinematics of a robot arm: predict workspace location from angles.

if income $> \theta_1$ and savings $> \theta_2$
then low-risk else high-risk



$$y = wx + w_0$$



- *Unsupervised learning*: no labels provided, only input data.

- *Learning associations*:

- * Basket analysis: let $p(Y|X)$ = “probability that a customer who buys product X also buys product Y ”, estimated from past purchases. If $p(Y|X)$ is large (say 0.7), associate “ $X \rightarrow Y$ ”. When someone buys X , recommend them Y .

- *Clustering*: group similar data points.

- *Density estimation*: where are data points likely to lie?

- *Dimensionality reduction*: data lies in a low-dimensional manifold.

- *Feature selection*: keep only useful features.
 - *Outlier/novelty detection*.
- *Semisupervised learning*: labels provided for some points only.
- *Reinforcement learning*: find a sequence of actions (policy) that reaches a goal. No supervised output but delayed reward.

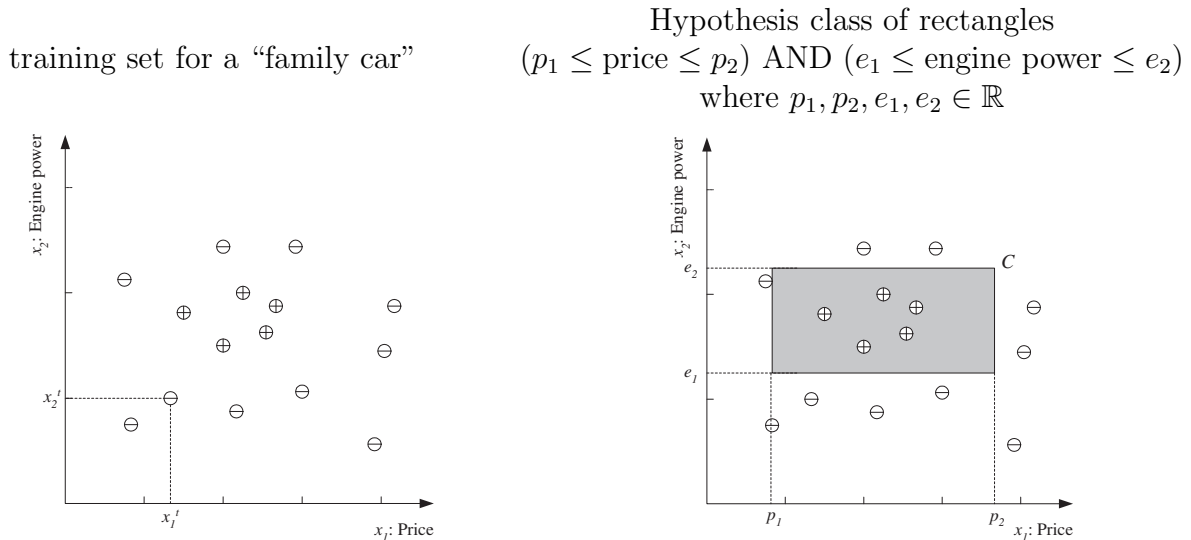
Ex: playing chess or a computer game, robot in a maze.

2 Supervised learning: classification & regression

Classification

Binary classification (two classes)

- We are given a training set of labeled examples (positive and negative) and want to learn a classifier that we can use to predict unseen examples, or to understand the data.
- *Input representation*: we need to decide what *attributes (features)* to use to describe the *input patterns (examples, instances, data points)*. This implies ignoring other attributes as irrelevant.



- *Training set*: $\mathcal{X} = \{(\mathbf{x}_n, y_n)\}_{n=1}^N$ where $\mathbf{x}_n \in \mathbb{R}^D$ is the n th input vector and $y_n \in \{0, 1\}$ its class label.
- *Classifier*: a function $h: \mathbb{R}^D \rightarrow \{0, 1\}$.
- *Hypothesis (model) class* \mathcal{H} : the set of classifier functions we will use. Ideally, the true class distribution can be represented by a function in \mathcal{H} (exactly, or with a small error).
- Having selected \mathcal{H} , learning the classifier reduces to finding an optimal $h \in \mathcal{H}$. We don't know the true class regions, but we can approximate them by the *empirical error*:

$$E(h; \mathcal{X}) = \sum_{n=1}^N I(h(\mathbf{x}_n) \neq y_n) = \text{number of misclassified instances}$$

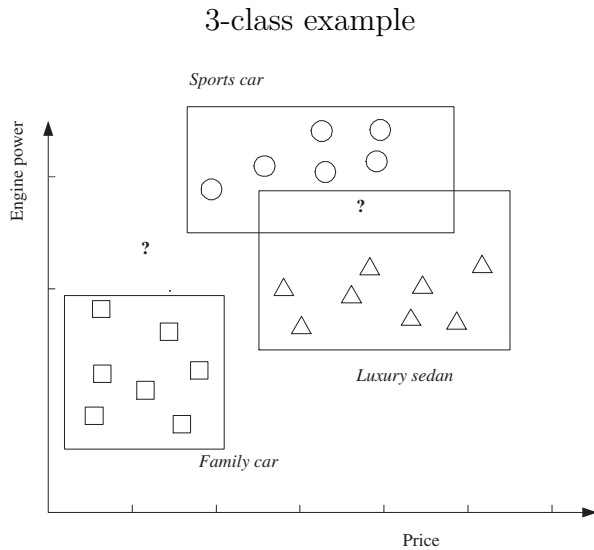
Multiclass classification (more than two classes)

- With K classes, we can code the label as an integer $y = k \in \{1, \dots, K\}$, or as a one-of- K (one-hot) binary vector $\mathbf{y} = (y_1, \dots, y_K)^T \in \{0, 1\}^K$ (containing a single 1 in position k).
- One approach for K -class classification: consider it as K two-class classification problems (“one-vs-all”), and minimize the total empirical error:

$$E(\{h_k\}_{k=1}^K; \mathcal{X}) = \sum_{n=1}^N \sum_{k=1}^K I(h_k(\mathbf{x}_n) \neq y_{nk})$$

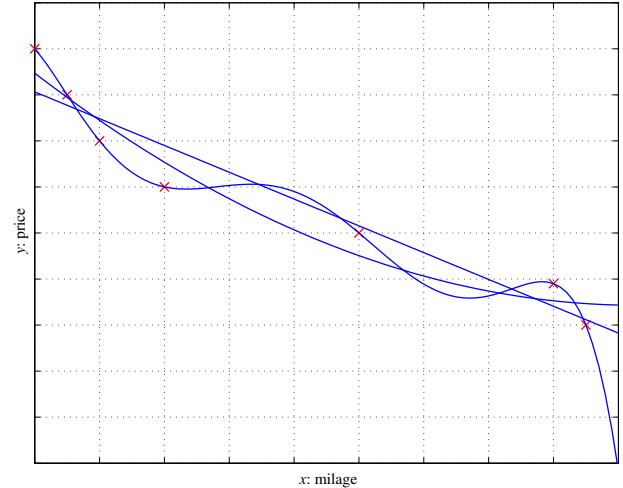
where \mathbf{y}_n is coded as one-of- K and h_k is the two-class classifier for problem k , i.e., $h_k(\mathbf{x}) \in \{0, 1\}$.

- Ideally, for a given pattern \mathbf{x} only one $h_k(\mathbf{x})$ is one. When no, or more than one, $h_k(\mathbf{x})$ is one then the classifier is in doubt and may reject the pattern.



regression: polynomials of order 1, 2, 6

(🔗 solve for order 1: optimal w_0, w_1)



Regression

- Training set $\mathcal{X} = \{(\mathbf{x}_n, y_n)\}_{n=1}^N$ where the label for a pattern $\mathbf{x}_n \in \mathbb{R}^D$ is a *real value* $y_n \in \mathbb{R}$.

In multivariate regression, $\mathbf{y}_n \in \mathbb{R}^d$ is a real vector.

- *Regressor*: a function $h: \mathbb{R}^D \rightarrow \mathbb{R}$. As before, we choose a hypothesis class \mathcal{H} .

Ex: \mathcal{H} = class of linear functions: $h(\mathbf{x}) = w_0 + w_1x_1 + \dots + w_Dx_D = \mathbf{w}^T\mathbf{x} + w_0$.

- Empirical error: $E(h; \mathcal{X}) = \frac{1}{N} \sum_{n=1}^N (y_n - h(\mathbf{x}_n))^2$ = sum of squared errors at each instance.

Other definitions of error possible, e.g. absolute error value instead of squared error (but harder to optimize).

(🔗 Find the optimal regression line for the case $D = 1$: $h(x) = w_1x + w_0$.)

- *Interpolation*: we learn a function h that passes through each training pair (\mathbf{x}_n, y_n) : $y_n = h(\mathbf{x}_n)$, $n = 1, \dots, N$. Not advisable if the data is noisy.

Ex: polynomial interpolation (requires a polynomial of degree $N - 1$ with N points in general position).

Noise

- *Noise* is any unwanted anomaly in the data. It can be due to:
 - Imprecision in recording the input attributes: \mathbf{x}_n .
 - Errors in labeling the input vectors: y_n .
 - Attributes not considered that affect the label (*hidden* or *latent* attributes, may be unobservable).
- Noise makes learning harder.
- Should we keep the hypothesis class simple rather than complex? A simpler class:

- Is easier to use and to train (fewer parameters, faster).
- Is easier to explain or interpret.
- Has less variance in the learned model than for a complex class (less affected by single instances), but also has higher bias.

Given comparable empirical error, a simple model will *generalize* better than a complex one. (*Occam's razor*: simpler explanations are more plausible; eliminate unnecessary complexity.)

In the regression example, a line with low enough error may be preferable to a parabola with a slightly lower error.

Outlier (novelty, anomaly) detection

- An *outlier* is an instance that is very different from the other instances in the sample. Reasons:
 - Abnormal behaviour. Fraud in credit card transactions, intruder in network traffic, etc.
 - Recording error. Faulty sensors, etc.
- Not usually cast as a two-class classification problem because there are typically few outliers and they don't fit a consistent pattern that can be easily learned.
- Instead, “one-class classification”: fit a density $p(\mathbf{x})$ to non-outliers, then consider \mathbf{x} as an outlier if $p(\mathbf{x}) < \theta$ for some threshold $\theta > 0$ (low-probability instance).
- We can also identify outliers as points that are far away from the training samples.

Estimation of missing values

- For any given example \mathbf{x} , the values of some features x_1, \dots, x_n may be missing.
Ex: survey nonresponse.
- Strategies to deal with missing values in the training set:
 - Discard examples having any missing values. Easy but throws away data.
 - Fill in the missing values by estimating them (*imputation*).
Ex: *mean imputation*: impute feature d as its average value over the examples where it is present. Independent of the present features in \mathbf{x}_n .
- Whether a feature is missing for \mathbf{x}_n may depend on the values of the other features at \mathbf{x}_n .
Ex: in a census survey, rich people may wish not to give their salary.

Model selection and generalization

- Machine learning problems (classification, regression and others) are typically *ill-posed*: the observed data is finite and does not uniquely determine the classification or regression function.
- In order to find a unique solution, and learn something useful, we must make assumptions (= *inductive bias* of the learning algorithm).
Ex: the use of a hypothesis class \mathcal{H} ; the use of the largest margin; the use of the least-squares objective function.
- We can always enlarge the class of functions that can be learned by using a larger hypothesis class \mathcal{H} (higher *capacity*, or *complexity* of \mathcal{H}).
Ex: a union of rectangles; a polynomial of order N .

- How to choose the right inductive bias, in particular the right hypothesis class \mathcal{H} ? This is the *model selection problem*.
- The goal of ML is not to replicate the training data, but *to predict unseen data well*, i.e., to *generalize well*.
- For best generalization, we should match the complexity of the hypothesis class \mathcal{H} with the complexity of the function underlying the data:
 - If \mathcal{H} is less complex: *underfitting*. Ex: fitting a line to data generated from a cubic polynomial.
 - If \mathcal{H} is more complex: *overfitting*. Ex: fitting a cubic polynomial to data generated from a line.
- In summary, in ML algorithms there is a tradeoff between 3 factors:
 - the complexity $c(\mathcal{H})$ of the hypothesis class
 - the amount of training data N
 - the generalization error E

so that

- as $N \uparrow$, $E \downarrow$
- as $c(\mathcal{H}) \uparrow$, first $E \downarrow$ and then $E \uparrow$

Cross-validation Often used in practice to select among several hypothesis classes $\mathcal{H}_1, \mathcal{H}_2, \dots$. Divide the available dataset into three *disjoint* parts (say, $\frac{1}{3}$ each):

- *Training set*:
 - Used to train, i.e., to fit a hypothesis $h \in \mathcal{H}_i$.
 - *Optimize parameters of h given the model structure and hyperparameters.*
 - Usually done with an optimization algorithm (the *learning algorithm*).

Ex: learn the weights of a neural net (with backpropagation); construct a k -nearest-neighbor classifier (by storing the training set).

- *Validation set*:
 - Used to minimize the generalization error.
 - *Optimize hyperparameters or model structure.*
 - Usually done with a “grid search”. Ex: try all values of $H \in \{10, 50, 100\}$ and $\lambda \in \{10^{-5}, 10^{-3}, 10^{-1}\}$.

Ex: select the number of hidden units H , the architecture, the regularization parameter λ , or how long to train for a neural net; select k for a k -nearest-neighbor classifier.

- *Test set*:
 - Used to report the generalization error.
 - We optimize nothing on it, we just evaluate the final model on it.

Then:

1. For each class \mathcal{H}_i , fit its optimal hypothesis h_i using the training set.
2. Of all the optimal hypotheses, pick the one that is most accurate in the validation set.
We can then train the selected one on the training and validation set together (useful if we have little data altogether).
3. Report its error in the test set.

Analogy: learning a subject. Training set: problems solved in class, validation set: exam problems, test set: professional-life problems.

Dimensions of a supervised ML algorithm

- We have a sample $\mathcal{X} = \{(\mathbf{x}_n, y_n)\}_{n=1}^N$ (usually independent and identically distributed, “iid”) drawn from an unknown distribution.
- We want to learn a useful approximation to the underlying function that generated the data.
- We must choose:
 1. A model $h(\mathbf{x}; \Theta)$ (hypothesis class) with parameters Θ . A particular value of Θ determines a particular hypothesis in the class.
Ex: for linear models, Θ = slope w_1 and intercept w_0 .
 2. A *loss function* $L(\cdot, \cdot)$ to compute the difference between the desired output (label) y_n and our prediction to it $h(\mathbf{x}_n; \Theta)$. *Approximation error (loss)*:

$$E(\Theta; \mathcal{X}) = \sum_{n=1}^N L(y_n, h(\mathbf{x}_n; \Theta)) = \text{sum of errors over instances}$$

Ex: 0/1 loss for classification, squared error for regression.

3. An optimization procedure (learning algorithm) to find parameters Θ^* that minimize the error:

$$\Theta^* = \arg \min_{\Theta} E(\Theta; \mathcal{X})$$

Different ML algorithms differ in any of these choices.

- The model, loss and learning algorithm are chosen by the ML system designer so that:
 - The model class is large enough to contain a good approximation to the underlying function that generated the data in \mathcal{X} in a noisy form.
 - The learning algorithm is efficient and accurate.
 - We must have sufficient training data to pinpoint the right model.

3 Bayesian decision theory

Probability review: [appendix A](#).

Joint distribution: $p(X = x, Y = y)$.

Conditioning (product rule): $p(Y = y | X = x) = \frac{p(X = x, Y = y)}{p(X = x)}$.

Marginalizing (sum rule): $p(X = x) = \sum_y p(X = x, Y = y)$.

Bayes' theorem:
(inverse probability) $p(X = x | Y = y) = \frac{p(Y = y | X = x) p(X = x)}{p(Y = y)}$.

Probability theory (and Bayes' rule) is the framework for making decisions under uncertainty. It can also be used to make rational decisions among multiple actions to minimize expected risk.

Classification

- Binary classification with random variables $\mathbf{x} \in \mathbb{R}^D$ (example) and $C \in \{0, 1\}$ (label).
 - *Joint probability* $p(\mathbf{x}, C)$: how likely it is to observe an example \mathbf{x} and a class label C .
 - *Prior probability* $p(C)$: how likely it is to observe a class label C , regardless of \mathbf{x} .
 $p(C) \geq 0$ and $p(C = 1) + p(C = 0) = 1$.
 - *Class likelihood* $p(\mathbf{x}|C)$: how likely it is that, having observed an example with class label C , the example is at \mathbf{x} .
This represents how the examples are distributed for each class. We need a model of \mathbf{x} for each class.
 - *Posterior probability* $p(C|\mathbf{x})$: how likely it is that, having observed an example \mathbf{x} , its class label is C .
 $p(C = 1|\mathbf{x}) + p(C = 0|\mathbf{x}) = 1$.
This is what we need to classify \mathbf{x} . We infer it from $p(C)$ and $p(\mathbf{x}|C)$ using Bayes' rule.
 - *Evidence* $p(\mathbf{x})$: probability of observing an example \mathbf{x} at all (regardless of its class).
Marginal distribution of \mathbf{x} : $p(\mathbf{x}) = p(\mathbf{x}|C = 0)p(C = 0) + p(\mathbf{x}|C = 1)p(C = 1)$.
 - *Bayes' rule*: posterior = $\frac{\text{prior} \times \text{likelihood}}{\text{evidence}}$

$$p(C|\mathbf{x}) = \frac{p(\mathbf{x}|C)p(C)}{p(\mathbf{x})} = \frac{p(\mathbf{x}|C)p(C)}{p(\mathbf{x}|C = 0)p(C = 0) + p(\mathbf{x}|C = 1)p(C = 1)}$$

- Making a decision on a new example: given \mathbf{x} , classify it as class 1 iff $p(C = 1|\mathbf{x}) > p(C = 0|\mathbf{x})$.

- Examples:

- Gaussian classes in 1D: $p(\mathbf{x}|C_k) = \frac{1}{\sqrt{2\pi}\sigma_k} e^{-\frac{1}{2}\left(\frac{x-\mu_k}{\sigma_k}\right)^2}$.

- **Exe 3.1**: disease diagnosis based on a test.

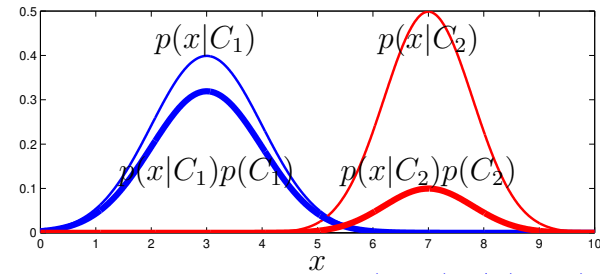
Approx. as $p(d = 1|t = 1) \approx p(d = 1) \frac{p(t=1|d=1)}{p(t=1|d=0)}$ in terms of likelihood ratio. How can we increase $p(d = 1|t = 1)$?

- K -class case: $C \in \{1, \dots, K\}$.

- Prior probability: $p(C_k) \geq 0$, $k = 1, \dots, K$, and $\sum_{k=1}^K p(C_k) = 1$.
- Class likelihood: $p(\mathbf{x}|C_k)$.
- Bayes' rule: $p(C_k|\mathbf{x}) = \frac{p(\mathbf{x}|C_k)p(C_k)}{p(\mathbf{x})} = \frac{p(\mathbf{x}|C_k)p(C_k)}{\sum_{i=1}^K p(\mathbf{x}|C_i)p(C_i)}$.
- Choose as class $\arg \max_{k=1, \dots, K} p(C_k|\mathbf{x})$.

- We learn the distributions $p(C)$ and $p(\mathbf{x}|C)$ for each class from data using an algorithm.

- *Naive Bayes classifier*: a very simple classifier were we assume $p(\mathbf{x}|C_k) = p(x_1|C_k) \cdots p(x_D|C_k)$ for each class $k = 1, \dots, K$, i.e., within each class the features are independent from each other.



Losses and risks

- Sometimes the decision of what class to choose for \mathbf{x} based on $p(C_k|\mathbf{x})$ has a cost that depends on the class. In that case, *we should choose the class with lowest risk*.

Ex: medical diagnosis, earthquake prediction.

- Define:

– λ_{ik} : *loss (cost)* incurred for choosing class i when the input actually belongs to class k .

– *Expected risk* for choosing class i : $R_i(\mathbf{x}) = \sum_{k=1}^K \lambda_{ik} p(C_k|\mathbf{x})$.

- Decision: we choose the class with minimum risk: $\arg \min_{k=1,\dots,K} R_k(\mathbf{x})$.

Ex: C_1 = no cancer, C_2 = cancer; $(\lambda_{ik}) = \begin{pmatrix} 0 & 1000 \\ 1 & 0 \end{pmatrix}$; $p(C_1|\mathbf{x}) = 0.8$. $R_1 = ?$, $R_2 = ?$

- Particular case: *0/1 loss*.

– Correct decisions have no loss and all incorrect decisions are equally costly:

$\lambda_{ik} = 0$ if $i = k$, 1 if $i \neq k$.

– Then $R_i(\mathbf{x}) = 1 - p(C_i|\mathbf{x})$, hence to minimize the risk we pick the most probable class.

- Up to now we have considered K possible decisions given an input \mathbf{x} (each of the K classes). When incorrect decisions (misclassifications) are costly, it may be useful to define an additional decision of *reject* or *doubt* (and then defer the decision to a human). For the 0/1 loss:

– Define a cost $\lambda_{\text{reject},k} = \lambda \in (0, 1)$ for rejecting an input that actually belongs to class k .

– Risk of choosing class i : $R_i(\mathbf{x}) = \sum_{k=1}^K \lambda_{ik} p(C_k|\mathbf{x}) = 1 - p(C_i|\mathbf{x})$.

Risk of rejecting: $R_{\text{reject}}(\mathbf{x}) = \sum_{k=1}^K \lambda_{\text{reject},k} p(C_k|\mathbf{x}) = \lambda$.

\Rightarrow optimal decision rule (given by the minimal risk)[?]: $\arg \max\{p(C_1|\mathbf{x}), \dots, p(C_K|\mathbf{x}), 1 - \lambda\}$
(so reject if $\max\{p(C_1|\mathbf{x}), \dots, p(C_K|\mathbf{x})\} < 1 - \lambda$).

– Extreme cases of λ :

* $\lambda = 0$: always reject (rejecting is less costly than a correct classification).

* $\lambda = 1$: never reject (rejecting is costlier than any misclassification).

Discriminant functions

- Classification rule: choose $\arg \max_{k=1,\dots,K} g_k(\mathbf{x})$
where we have a set of K *discriminant functions* g_1, \dots, g_K (not necessarily probability-based).

- Examples:

– Risk based on Bayes' classifier: $g_k = -R_k$.

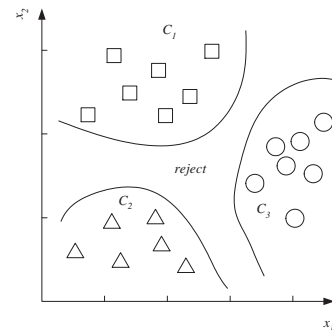
– With the 0/1 loss:

$g_k(\mathbf{x}) = p(C_k|\mathbf{x})$, or[?] $g_k(\mathbf{x}) = p(\mathbf{x}|C_k)p(C_k)$, or[?] $g_k(\mathbf{x}) = \log p(\mathbf{x}|C_k) + \log p(C_k)$.

– Many others: SVMs, neural nets, etc.

- They divide the feature space into K decision regions $\mathcal{R}_k = \{\mathbf{x}: k = \arg \max_i g_i(\mathbf{x})\}$, $k = 1, \dots, K$. The regions are separated by decision boundaries, where ties occur.

- With two classes we can define a single discriminant[?] $g(\mathbf{x}) = g_1(\mathbf{x}) - g_2(\mathbf{x})$ and choose class 1 iff $g(\mathbf{x}) > 0$.



Frequently Bought Together



Price for all three: **\$174.78**

Add all three to Cart

Add all three to Wish List

Some of these items ship sooner than the others. [Show details](#)

- ☑ **This item:** Introduction to Machine Learning (Adaptive Computation and Machine Learning series) by Ethem Alpaydin Hardcover **\$49.99**
- ☑ [An Introduction to Statistical Learning: with Applications in R \(Springer Texts in Statistics\)](#) by Gareth James Hardcover **\$61.97**
- ☑ [The Elements of Statistical Learning: Data Mining, Inference, and Prediction, Second Edition \(Springer ...\)](#) by Trevor Hastie Hardcover **\$62.82**

Association rules

- *Association rule*: implication of the form $X \rightarrow Y$ (X : antecedent, Y : consequent).
- Application: basket analysis (recommend product Y when a customer buys product X).
- Define the following measures of the association rule $X \rightarrow Y$:

- $Support = p(X, Y) = \frac{\# \text{ purchases of } X \text{ and } Y}{\# \text{ purchases}}$.

For the rule to be significant, the support should be large. High support means items X and Y are frequently bought together.

- $Confidence = p(Y|X) = \frac{p(X, Y)}{p(X)} = \frac{\# \text{ purchases of } X \text{ and } Y}{\# \text{ purchases of } X}$.

For the rule to hold with enough confidence, should be $\gg p(Y)$ and close to 1.

- Generalization to more than 2 items: $X, Z \rightarrow Y$ has confidence $p(Y|X, Z)$, etc.

- Example: given this database of purchases... consider the following rules:

purchase	items in basket	association rule	support	confidence
1	milk, bananas, chocolate	milk \rightarrow bananas	2/6	2/4
2	milk, chocolate	bananas \rightarrow milk	2/6	2/2
3	milk, bananas	milk \rightarrow chocolate	?	?
4	chocolate	chocolate \rightarrow milk	?	?
5	chocolate	etc.		
6	milk, chocolate			

- Given a database of purchases, we want to find all possible rules having enough support and confidence.

- Algorithm *Apriori*:

1. Find itemsets with enough support (by finding frequent itemsets).
We don't need to enumerate all possible subsets of items: for (X, Y, Z) to have enough support, all its subsets (X, Y) , (Y, Z) , (X, Z) must have enough support[?]. Hence: start by finding frequent one-item sets (by doing a pass over the database). Then, inductively, from frequent k -item sets generate $k + 1$ -item sets and check whether they have enough support (by doing a pass over the database).
2. Convert the found itemsets into rules with enough confidence (by splitting the itemset into antecedent and consequent).
Again, we don't need to enumerate all possible subsets: for $X \rightarrow Y, Z$ to have enough confidence, $X, Y \rightarrow Z$ must have enough confidence[?]. Hence: start by considering a single consequent and test the confidence for all possible single consequents (adding as a valid rule if it has enough confidence). Then, inductively, consider two consequents for the added rules; etc.

- A more general way to establish rules (which may include hidden variables) are graphical models.

Measuring classifier performance

Binary classification problems ($K = 2$ classes)

- The most basic measure is the classification error (or accuracy) in %: $\frac{\# \text{ misclassified patterns}}{\# \text{ patterns}}$.

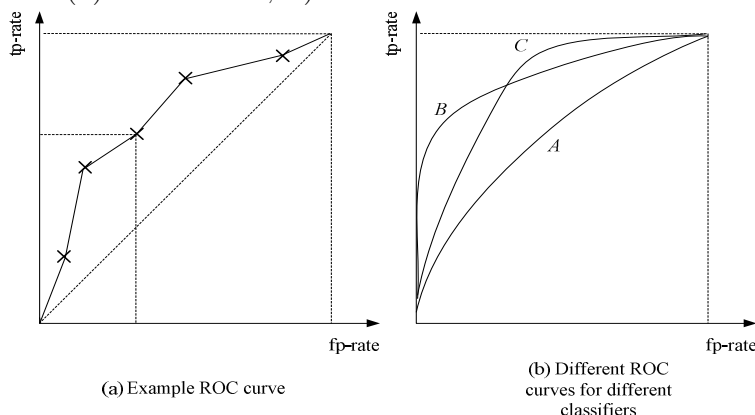
- It is often of interest to distinguish the different types of errors: confusing the positive class with the negative one, or vice versa.

Ex: voice authentication to log into a user account.
A false positive (allowing an impostor) is much worse than a false negative (refusing a valid user).

True class	Predicted class		
	Positive	Negative	Total
Positive	tp: true positive	fn: false negative	p
Negative	fp: false positive	tn: true negative	n
Total	p'	n'	N

- Having trained a classifier, we can control fn vs fp through a threshold $\theta \in [0, 1]$. Let C_1 be the positive class and C_2 the negative class. If the classifier returns $p(C_1|\mathbf{x})$, let us choose the positive class if $p(C_1|\mathbf{x}) > \theta$ (so $\theta = \frac{1}{2}$ gives the usual classifier):
 - $\theta \approx 1$: almost always choose C_2 , nearly no false positives but nearly no true positives.
 - Decreasing θ increases the number of true positives but risks introducing false positives.
- ROC curve** (“receiver operating curve”): the pair values (fp-rate, tp-rate) as a function of $\theta \in [0, 1]$. Properties:
 - It is always increasing.
 - An ideal classifier is at (0, 1) (top left corner).
 - Diagonal (fp-rate = tp-rate): random classifier, which outputs $p(C_1|\mathbf{x}) = u \sim U(0, 1)$?. This is the worst we can do.
 - Any classifier that is below the diagonal can be improved by flipping its decision?.
 - For a dataset with N points, the ROC curve is really a set of N points (fp-rate, tp-rate)?. To construct it we don't need to know the classifier, just its outputs $p(C_1|\mathbf{x}_n) \in [0, 1]$ on the points $\{(\mathbf{x}_n, y_n)\}_{n=1}^N$ of the dataset.
- Area under the curve (AUC)**: reduces the ROC curve to a number. Ideal classifier: $AUC = 1$.
- ROC and AUC allow us to compare classifiers over different loss conditions, and choose a value of θ accordingly. Often, there is not a dominant classifier.

(a) ROC curve; b) ROC curves for 3 classifiers



Measure $\in [0, 1]$	Formula
error	$(fp + fn)/N$
accuracy = 1 - error	$(tp + tn)/N$
tp-rate (hit rate)	tp/p
fp-rate (false alarm rate)	fp/n
precision	tp/p'
recall = tp-rate	tp/p
F-score	$\frac{\text{precision} \times \text{recall}}{(\text{precision} + \text{recall})/2}$
sensitivity = tp-rate	tp/p
specificity = 1 - fp-rate	tn/n

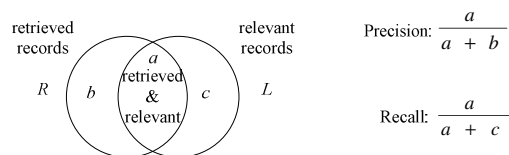
Information retrieval

We have a database of records (documents, images...) and a query, for which some of the records are relevant (“positive”) and the rest are not (“negative”). A retrieval system returns K records for the query. Its performance is measured using a *precision/recall curve*:

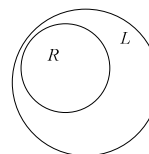
- *Precision*: $\frac{\# \text{ relevant retrieved records}}{\# \text{ retrieved records}} \in [0, 1]$.
- *Recall*: $\frac{\# \text{ relevant retrieved records}}{\# \text{ relevant records}} \in [0, 1]$.

We can always achieve perfect recall by returning the entire database (but it will contain many irrelevant records).

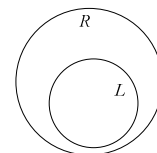
Precision & recall using Venn diagrams



(a) Precision and recall



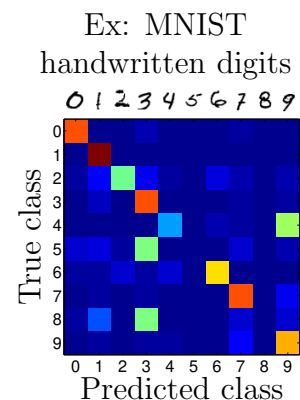
(b) Precision = 1



(c) Recall = 1

$K > 2$ classes

- Again, the most basic measure is the classification error.
- *Confusion matrix*: $K \times K$ matrix where entry (i, j) contains the number of instances of class C_i that are classified as C_j .
- It allows us to identify which types of misclassification errors tend to occur, e.g. if there are two classes that are frequently confused. Ideal classifier: the confusion matrix is diagonal.



4 Univariate parametric methods

- How to learn probability distributions from data (in order to use them to make decisions).
- We assume such distributions follow a particular parametric form (e.g. Gaussian), so we need to estimate its parameters (μ, σ) .
- Several ways to learn them:
 - by optimizing an objective function (e.g. maximum likelihood)
 - by Bayesian estimation.
- This chapter: univariate case; next chapter: multivariate case.

Joint distribution: $p(X = x, Y = y)$.

Conditioning (product rule): $p(Y = y | X = x) = \frac{p(X = x, Y = y)}{p(X = x)}$.

Marginalizing (sum rule): $p(X = x) = \sum_y p(X = x, Y = y)$.

Bayes' theorem: (inverse probability) $p(X = x | Y = y) = \frac{p(Y = y | X = x) p(X = x)}{p(Y = y)}$.

X and Y are independent $\Leftrightarrow p(X = x, Y = y) = p(X = x) p(Y = y)$.

Maximum likelihood estimation: parametric density estimation

- Problem: estimating a density $p(x)$. Assume an iid sample $\mathcal{X} = \{x_n\}_{n=1}^N$ drawn from a known probability density family $p(x; \Theta)$ with parameters Θ . We want to estimate Θ from \mathcal{X} .
- *Log-likelihood* of Θ given \mathcal{X} : $\mathcal{L}(\Theta; \mathcal{X}) = \log p(\mathcal{X}; \Theta) \stackrel{?}{=} \log \prod_{n=1}^N p(x_n; \Theta) = \sum_{n=1}^N \log p(x_n; \Theta)$.
- *Maximum likelihood estimate (MLE)*: $\Theta_{\text{MLE}} = \arg \max_{\Theta} \mathcal{L}(\Theta; \mathcal{X})$.
- Examples:

- **Bernoulli**: $\Theta = \{\theta\}$, $p(x; \theta) = \theta^x (1 - \theta)^{1-x} = \begin{cases} \theta, & \text{if } x = 1 \\ 1 - \theta, & \text{if } x = 0 \end{cases}$, $x \in \{0, 1\}$, $\theta \in [0, 1]$.
MLE[?]: $\hat{\theta} = \frac{\# \text{ ones}}{\# \text{ tosses}} = \frac{1}{N} \sum_{n=1}^N x_n$ (sample average).
- **Gaussian**: $\Theta = \{\mu, \sigma^2\}$, $p(x; \mu, \sigma^2) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{1}{2}(\frac{x-\mu}{\sigma})^2}$, $x \in \mathbb{R}$, $\mu \in \mathbb{R}$, $\sigma \in \mathbb{R}^+$.
MLE[?]: $\hat{\mu} = \frac{1}{N} \sum_{n=1}^N x_n$ (sample average), $\hat{\sigma}^2 = \frac{1}{N} \sum_{n=1}^N (x_n - \hat{\mu})^2$ (sample variance).

For more complicated distributions, we usually need an algorithm to find the MLE.

The Bayes' estimator: parametric density estimation

- Consider the parameters Θ as random variables themselves (not as unknown numbers), and assume a *prior distribution* $p(\Theta)$ over them (based on domain information): how likely it is for the parameters to take a value *before* having observed any data.
- *Posterior distribution* $p(\Theta | \mathcal{X}) = \frac{p(\mathcal{X} | \Theta) p(\Theta)}{p(\mathcal{X})} = \frac{p(\mathcal{X} | \Theta) p(\Theta)}{\int p(\mathcal{X} | \Theta') p(\Theta') d\Theta'}$: how likely it is for the parameters to take a value *after* having observed a sample \mathcal{X} .
- Resulting estimate for the probability at a new point x [?]: $p(x | \mathcal{X}) = \int p(x | \Theta) p(\Theta | \mathcal{X}) d\Theta$. Hence, rather than using the prediction of a single Θ value ("frequentist statistics"), *we average the prediction of every parameter value Θ using its posterior distribution* ("Bayesian statistics").
- Approximations: reduce $p(\Theta | \mathcal{X})$ to a single point Θ .
 - *Maximum a posteriori (MAP) estimate*: $\Theta_{\text{MAP}} = \arg \max_{\Theta} p(\Theta | \mathcal{X})$.
Particular case: if $p(\Theta) = \text{constant}$, then $p(\Theta | \mathcal{X}) \propto p(\mathcal{X} | \Theta)$ and MAP estimate = MLE.
 - *Bayes' estimator*: $\Theta_{\text{Bayes}} = \mathbb{E} \{\Theta | \mathcal{X}\} = \int \Theta p(\Theta | \mathcal{X}) d\Theta$.

Works well if $p(\Theta | \mathcal{X})$ is peaked around a single value.

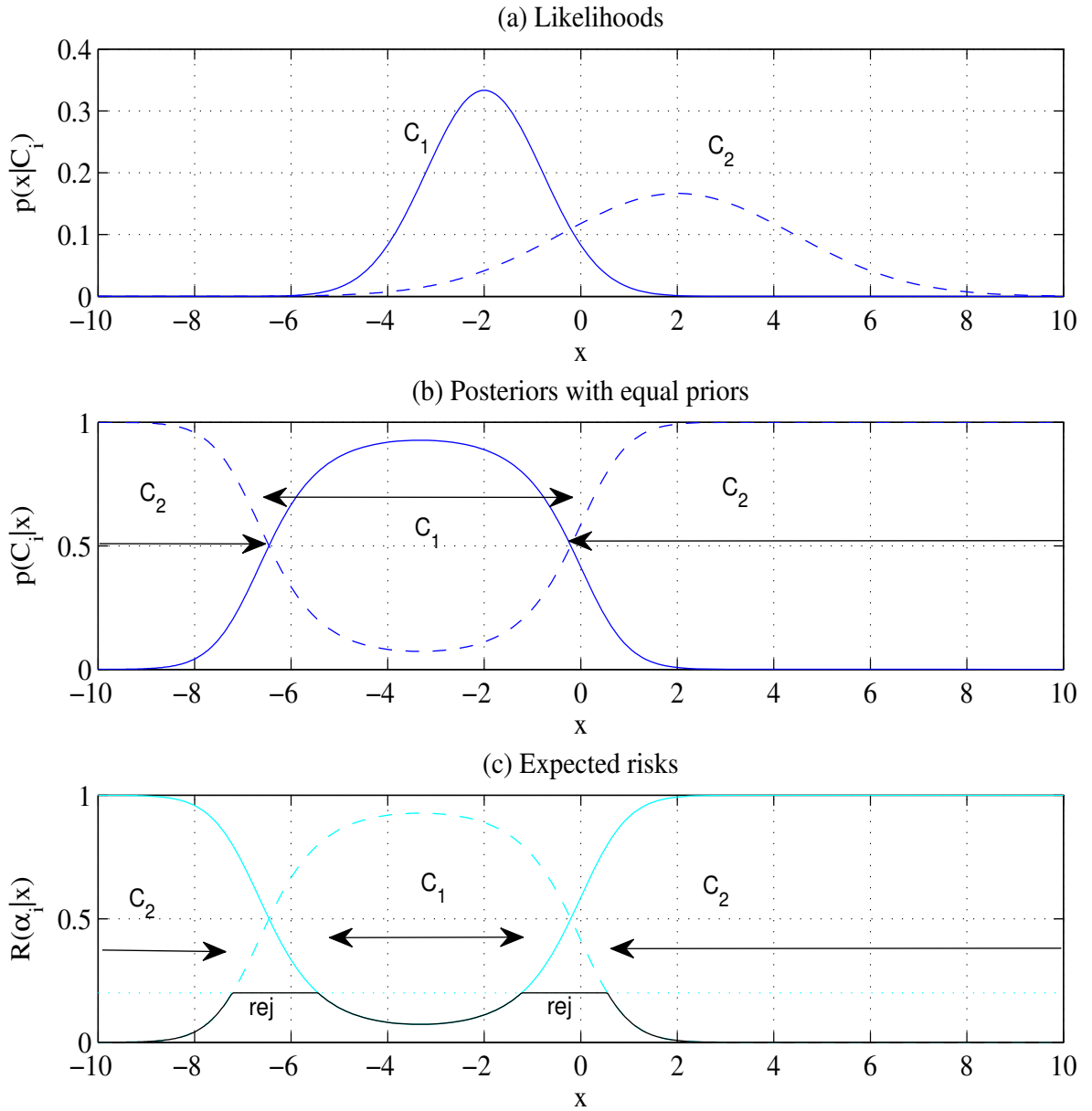
- Example: suppose $x_n \sim \mathcal{N}(\theta, \sigma^2)$ and $\theta \sim \mathcal{N}(\mu_0, \sigma_0^2)$ where $\mu_0, \sigma^2, \sigma_0^2$ are known. Then[?]: $\mathbb{E} \{\theta | \mathcal{X}\} = \frac{N/\sigma^2}{N/\sigma^2 + 1/\sigma_0^2} \hat{\mu} + \frac{1/\sigma_0^2}{N/\sigma^2 + 1/\sigma_0^2} \mu_0$.
- Advantages and disadvantages of Bayesian learning:
 - ✓ works well when the sample size N is small (if the prior is helpful).
 - ✗ computationally harder (needs to compute, usually approximately, integrals or summations); needs to define a prior.

Maximum likelihood estimation: parametric classification

- From ch. 3, for classes $k = 1, \dots, K$, we use:

$$p(C_k|x) = \frac{p(x|C_k)p(C_k)}{p(x)} = \frac{p(x|C_k)p(C_k)}{\sum_{i=1}^K p(x|C_i)p(C_i)} \quad \text{discriminant function } g_k(x) = p(x|C_k)p(C_k).$$

- Ex: assume $x|C_k \sim \mathcal{N}(x; \mu_k, \sigma_k^2)$. Estimate the parameters from a data sample $\{(x_n, y_n)\}_{n=1}^N$:
 - $\hat{p}(C_k)$ = proportion of y_n that are class k .
 - $\hat{\mu}_k, \hat{\sigma}_k^2$ = as the MLE above, separately for each class k .



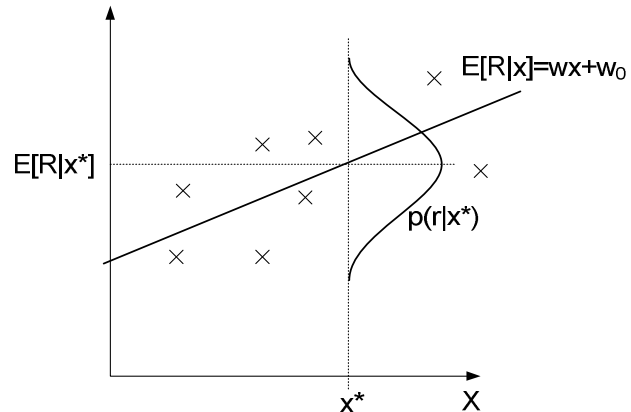
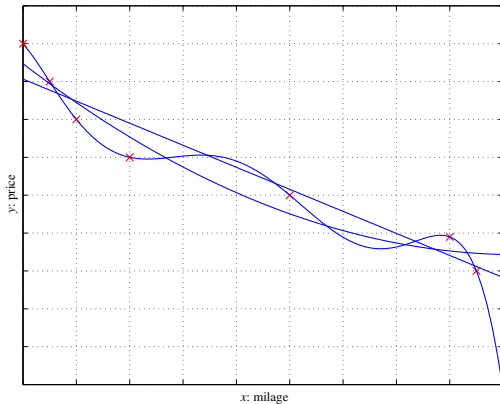
Maximum likelihood estimation: parametric regression

- Assume there exists an unknown function f that maps inputs x to outputs $y = f(x)$, but that what we observe as output is a noisy version $y = f(x) + \epsilon$, where ϵ is a random error. We want to estimate f by a parametric function $h(x; \Theta)$. In ch. 2 we saw the least-squares error was a good loss function to use for that purpose. We will now show that maximum likelihood estimation under Gaussian noise is equivalent to that.
- Log-likelihood of Θ given a sample $\{(x_n, y_n)\}_{n=1}^N$ drawn iid from $p(x, y)$:
 $\mathcal{L}(\Theta; \mathcal{X}) = \log \prod_{n=1}^N p(x_n, y_n) \stackrel{?}{=} \sum_{n=1}^N \log p(y_n | x_n; \Theta) + \text{constant}.$
- Assume an error $\epsilon \sim \mathcal{N}(0, \sigma^2)$, so $p(y|x) \sim \mathcal{N}(h(x; \Theta), \sigma^2)$. Then maximizing the log-likelihood is equivalent[?] to minimizing $E(\Theta; \mathcal{X}) = \sum_{n=1}^N (y_n - h(x_n; \Theta))^2$, i.e., the least-squares error.
- Examples:
 - Linear regression: $h(x; w_0, w_1) = w_1 x + w_0$. LSQ estimate[?]:

$$\mathbf{w} = \mathbf{A}^{-1} \mathbf{y}, \quad \mathbf{A} = \begin{pmatrix} 1 & \frac{1}{N} \sum_{n=1}^N x_n \\ \frac{1}{N} \sum_{n=1}^N x_n & \frac{1}{N} \sum_{n=1}^N x_n^2 \end{pmatrix}, \quad \mathbf{w} = \begin{pmatrix} w_0 \\ w_1 \end{pmatrix}, \quad \mathbf{y} = \begin{pmatrix} \frac{1}{N} \sum_{n=1}^N y_n \\ \frac{1}{N} \sum_{n=1}^N y_n x_n \end{pmatrix}.$$

- Polynomial regression: $h(x; w_0, \dots, w_k) = w_k x^k + \dots + w_1 x + w_0$. The model is still linear on the parameters. LSQ estimate also of the form $\mathbf{w} = \mathbf{A}^{-1} \mathbf{y}$.

p. 79



5 Multivariate parametric methods

- Sample $\mathcal{X} = \{\mathbf{x}_n\}_{n=1}^N$ where each example \mathbf{x}_n contains D features (continuous or discrete).
- We often write the sample (or dataset) as a matrix $\mathbf{X} = (\mathbf{x}_1, \dots, \mathbf{x}_N)$ (examples = columns).
Sometimes as its transpose (examples = rows).

Review of important moments

For a sample:

- *Mean vector*: $\boldsymbol{\mu} = \frac{1}{N} \sum_{n=1}^N \mathbf{x}_n$.
- *Covariance matrix*: $\boldsymbol{\Sigma} = \frac{1}{N} \sum_{n=1}^N (\mathbf{x}_n - \boldsymbol{\mu})(\mathbf{x}_n - \boldsymbol{\mu})^T \stackrel{?}{=} \frac{1}{N} \sum_{n=1}^N \mathbf{x}_n \mathbf{x}_n^T - \boldsymbol{\mu} \boldsymbol{\mu}^T$.
Symmetric of $D \times D$ with elements σ_{ij} , positive definite, diagonal = variances along each variable ($\sigma_{11} = \sigma_1^2, \dots, \sigma_{DD} = \sigma_D^2$).
- *Correlation matrix*: $\text{corr}(X_i, X_j) = \frac{\sigma_{ij}}{\sigma_i \sigma_j} \in [-1, 1]$.
If $\text{corr}(X_i, X_j) = 0$ (equivalently $\sigma_{ij} = 0$) then X_i and X_j are uncorrelated (but not necessarily independent).
If $\text{corr}(X_i, X_j) = \pm 1$ then X_i and X_j are linearly related.

For a continuous distribution of the r.v. \mathbf{x} (for a discrete distribution, replace $\int \rightarrow \sum$):

- *Mean vector (expectation)*: $\mathbb{E}\{\mathbf{x}\} = \int \mathbf{x} p(\mathbf{x}) d\mathbf{x}$.
- *Covariance matrix*: $\text{cov}\{\mathbf{x}\} = \mathbb{E}\{(\mathbf{x} - \mathbb{E}\{\mathbf{x}\})(\mathbf{x} - \mathbb{E}\{\mathbf{x}\})^T\} \stackrel{?}{=} \mathbb{E}\{\mathbf{x} \mathbf{x}^T\} - \mathbb{E}\{\mathbf{x}\} \mathbb{E}\{\mathbf{x}\}^T$.

In 1D: $x \in \mathbb{R}$, mean $\mu \in \mathbb{R}$, variance $\Sigma = \sigma^2 > 0$

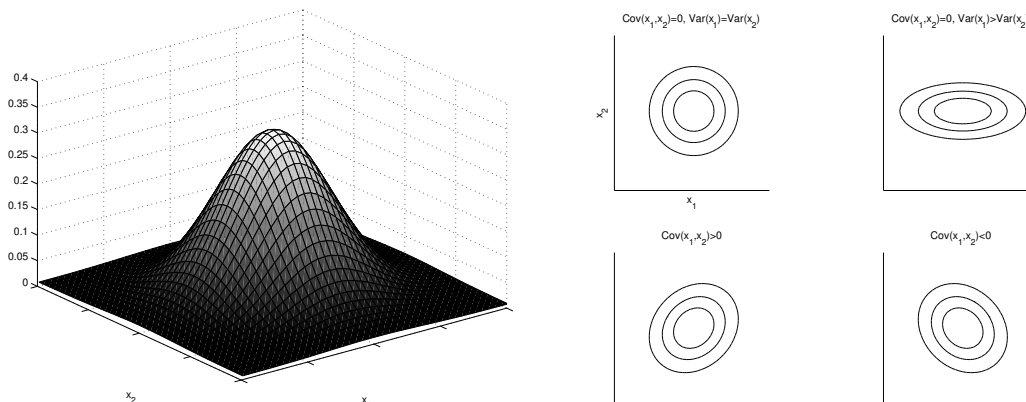
$$\sigma^2 = \frac{1}{N} \sum_{n=1}^N (x_n - \mu)^2 \stackrel{?}{=} \frac{1}{N} \sum_{n=1}^N x_n^2 - \mu^2$$

$$p(x) = \frac{1}{\sqrt{2\pi\sigma}} e^{-\frac{1}{2}\left(\frac{x-\mu}{\sigma}\right)^2}$$

Review of the multivariate normal (Gaussian) distribution

- Density at $\mathbf{x} \in \mathbb{R}^D$: $p(\mathbf{x}) = |2\pi\boldsymbol{\Sigma}|^{-1/2} e^{-\frac{1}{2}(\mathbf{x}-\boldsymbol{\mu})^T \boldsymbol{\Sigma}^{-1}(\mathbf{x}-\boldsymbol{\mu})}$. $\mathbb{E}\{\mathbf{x}\} = \boldsymbol{\mu}$ and $\text{cov}\{\mathbf{x}\} = \boldsymbol{\Sigma}$.
- *Mahalanobis distance*: $d_{\boldsymbol{\Sigma}}(\mathbf{x}, \mathbf{x}') = (\mathbf{x} - \mathbf{x}')^T \boldsymbol{\Sigma}^{-1}(\mathbf{x} - \mathbf{x}')$. $\boldsymbol{\Sigma} = \mathbf{I} \Rightarrow d_{\boldsymbol{\Sigma}}(\mathbf{x}, \mathbf{x}') = \|\mathbf{x} - \mathbf{x}'\|^2$ (Euclidean distance).
- Properties:
 - If $\boldsymbol{\Sigma}$ is diagonal \Rightarrow the D features are uncorrelated and independent: $p(\mathbf{x}) = p(x_1) \cdots p(x_D)$.
 - If \mathbf{x} is Gaussian \Rightarrow each marginal $p(x_i)$ and conditional distribution $p(x_i|x_j)$ is also Gaussian.
 - If $\mathbf{x} \sim \mathcal{N}_D(\boldsymbol{\mu}, \boldsymbol{\Sigma})$, $\mathbf{W}_{D \times K} \Rightarrow \mathbf{W}^T \mathbf{x} \sim \mathcal{N}_K(\mathbf{W}^T \boldsymbol{\mu}, \mathbf{W}^T \boldsymbol{\Sigma} \mathbf{W})$. A linear projection of a Gaussian is Gaussian.
- Widely used in practice because:
 - Its special mathematical properties simplify the calculations.
 - Many natural phenomena are approximately Gaussian.
 - Models well blobby distributions centered around a prototype vector $\boldsymbol{\mu}$ with a noise characterized by $\boldsymbol{\Sigma}$.

It does make strong assumptions: symmetry, unimodality, non-heavy tails.



Multivariate classification with Gaussian classes

Assume the class-conditional densities are Gaussian: $\mathbf{x}|C_k \sim \mathcal{N}(\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)$. The maths carry over from the 1D case in a straightforward way:

p. 100

- MLE[?]: class proportions for $p(C_k)$; sample mean and sample covariance for $\boldsymbol{\mu}_k$ and $\boldsymbol{\Sigma}_k$, resp.
- Discriminant function $g_k(\mathbf{x}) = \log p(\mathbf{x}|C_k) + \log p(C_k) \stackrel{?}{=}$ quadratic form on \mathbf{x} .
- Number of parameters per class $k = 1, \dots, K$: $p(C_k)$: 1; $\boldsymbol{\mu}_k$: D ; $\boldsymbol{\Sigma}_k$: $\frac{D(D+1)}{2}$ [?].

A lot of parameters for the covariances! Estimating them may need a large dataset.

Special cases, having fewer parameters:

- Equal (“shared”) covariances: $\boldsymbol{\Sigma}_k = \boldsymbol{\Sigma} \forall k$. Total $\frac{D(D+1)}{2}$ parameters (instead of $K\frac{D(D+1)}{2}$).
 - \Rightarrow MLE for $\boldsymbol{\Sigma} = \sum_{k=1}^K p(C_k) \boldsymbol{\Sigma}_k$, where $\boldsymbol{\Sigma}_k$ = covariance matrix of class k .
 - \Rightarrow the discriminant $g_k(\mathbf{x})$ becomes linear on \mathbf{x} .
 - If, in addition, equal priors: $p(C_k) = \frac{1}{K} \forall k$: **Mahalanobis distance classifier**
 \Rightarrow assign \mathbf{x} to the closest centroid $\boldsymbol{\mu}_k$ in Mahalanobis distance $(\mathbf{x} - \boldsymbol{\mu}_k)^T \boldsymbol{\Sigma}^{-1} (\mathbf{x} - \boldsymbol{\mu}_k)$.

p. 103

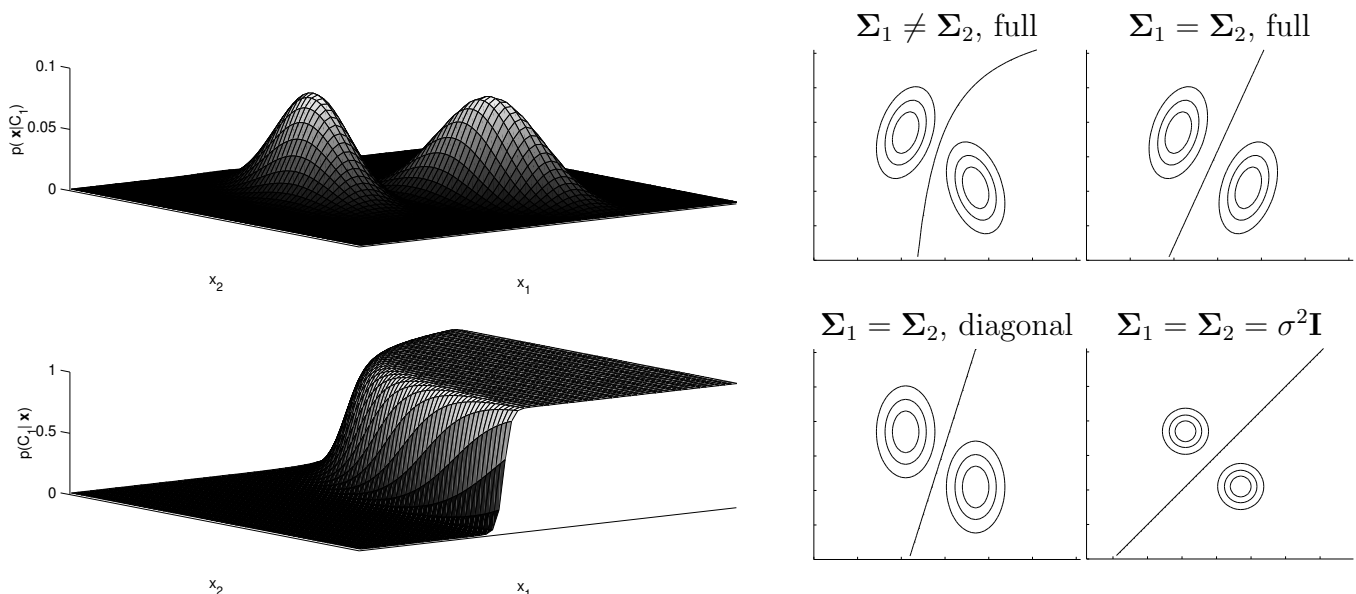
- Equal, isotropic covariances: $\boldsymbol{\Sigma}_k = \sigma^2 \mathbf{I} \forall k$. Total 1 parameter for all K covariances.
 - \Rightarrow MLE for $\sigma^2 = \frac{1}{D} \sum_{d=1}^D \sum_{k=1}^K p(C_k) \sigma_{kd}^2$, where $\sigma_{kd}^2 = d$ th diagonal element of $\boldsymbol{\Sigma}_k$.
 - If, in addition, equal priors: **Euclidean distance classifier** (“template matching”)
 - $\Rightarrow g_k(\mathbf{x}) = \|\mathbf{x} - \boldsymbol{\mu}_k\|^2 \Leftrightarrow g_k(\mathbf{x}) = \boldsymbol{\mu}_k^T \mathbf{x} - \frac{1}{2} \|\boldsymbol{\mu}_k\|^2$.
 - If $\|\boldsymbol{\mu}_k\|^2 = 1 \Rightarrow g_k(\mathbf{x}) = \boldsymbol{\mu}_k^T \mathbf{x}$: **dot product classifier**.

p. 105

- Diagonal covariances (not shared): $\boldsymbol{\Sigma}_k = \text{diag}(\sigma_{k1}^2, \dots, \sigma_{kD}^2)$. Total KD parameters.

This assumes the features are independent within each class, and gives a naive Bayes classifier.

- \Rightarrow MLE for $\sigma_{kd}^2 = d$ th diagonal element of $\boldsymbol{\Sigma}_k$ = variance of class k for feature d .



Tuning complexity

- Again a bias-variance dilemma:
 - Two few parameters (e.g. $\Sigma_1 = \Sigma_2 = \sigma^2 \mathbf{I}$): high bias.
 - Too many parameters (e.g. $\Sigma_1 \neq \Sigma_2$, full): high variance.
- We can use cross-validation, regularization, Bayesian priors on the covariances, etc.

Discrete features with Bernoulli distributions

- Assume that, in each class k , (1) the features are independent and (2) each feature $d = 1, \dots, D$ is Bernoulli with parameter θ_{kd} :

$$p(\mathbf{x}|C_k) \stackrel{(1)}{=} \prod_{d=1}^D p(x_d|C_k) \stackrel{(2)}{=} \prod_{d=1}^D \theta_{kd}^{x_d} (1 - \theta_{kd})^{1-x_d} \quad \mathbf{x} \in \{0, 1\}^D.$$

- This is a naive Bayes classifier. Its discriminant $g_k(\mathbf{x})$ is linear[?] (but \mathbf{x} is binary). p. 108
- MLE: class proportions for $p(C_k)$; sample mean for θ_{kd} .
- Works well with *document categorization* (e.g. classifying news reports into politics, sports and fashion) and *spam filtering* (classifying email messages into spam or non-spam). We typically represent a document as a *bag of words* vector \mathbf{x} : given a predetermined dictionary of D words, \mathbf{x} is a binary vector of dimension D where $x_d = 1$ iff word d is in the document.

Multivariate regression

- Linear regression where $\mathbf{x} \in \mathbb{R}^D$: $y = f(\mathbf{x}) + \epsilon$ with $f(\mathbf{x}) = w_D x_D + \dots + w_1 x_1 + w_0 = \mathbf{w}^T \mathbf{x}$ (where we define $x_0 = 1$).

- The maths carry over from the 1D case in a straightforward way: p. 110
 - Least-squares error (or equivalently maximum likelihood where ϵ is Gaussian with zero mean and constant variance): $E(\mathbf{w}) = \sum_{n=1}^N (y_n - \mathbf{w}^T \mathbf{x}_n)^2$.
 - The error E is quadratic on \mathbf{w} . Equating the derivatives of E wrt \mathbf{w} (the gradient) to $\mathbf{0}$ we obtain the *normal equations* (a linear system for \mathbf{w}):

$$\frac{\partial E}{\partial \mathbf{w}} = -2 \sum_{n=1}^N (y_n - \mathbf{w}^T \mathbf{x}_n) \mathbf{x}_n = \mathbf{0} \Rightarrow \left(\sum_{n=1}^N \mathbf{x}_n \mathbf{x}_n^T \right) \mathbf{w} = \sum_{n=1}^N y_n \mathbf{x}_n \Rightarrow (\mathbf{X}\mathbf{X}^T) \mathbf{w} = \mathbf{X}\mathbf{y}$$

where $\mathbf{X}_{D \times N} = (\mathbf{x}_1, \dots, \mathbf{x}_N)$, $\mathbf{w}_{D \times 1} = (w_0, \dots, w_D)^T$, $\mathbf{y}_{N \times 1} = (y_1, \dots, y_N)^T$.

- Inspecting the resulting values of the model parameters can give insights into the data:
 - the sign of w_d tells us whether x_d has a positive or negative effect on y ;
 - the magnitude of w_d tells us how influential x_d is (if x_1, \dots, x_D all have the same range).
- With multiple outputs $\mathbf{y} = (y_1, \dots, y_{D'})^T$, the linear regression $\mathbf{y} = \mathbf{f}(\mathbf{x}) + \boldsymbol{\epsilon}$ is equivalently defined as D' independent single-output regressions.

$E(\mathbf{W}) = \sum_{n=1}^N \|\mathbf{y}_n - \mathbf{W}\mathbf{x}_n\|^2$. Taking $\frac{\partial E}{\partial \mathbf{W}} = \mathbf{0}$ gives $(\mathbf{X}\mathbf{X}^T)\mathbf{W} = \mathbf{X}\mathbf{Y}^T$ with $\mathbf{Y}_{D' \times N} = (\mathbf{y}_1, \dots, \mathbf{y}_N)$ and $\mathbf{W}_{D' \times D}$.

- We can fit a nonlinear function $h(\mathbf{x})$ similarly to a linear regression by defining additional, nonlinear features such as $x_2 = x^2$, $x_3 = e^x$, etc. (just as happens with polynomial regression). Then, using a linear model in the augmented space $\mathbf{x} = (x, x^2, e^x)^T$ will correspond to a nonlinear model in the original space x . Radial basis function networks, kernel SVMs...

6 Bias, variance and model selection

Evaluating an estimator: bias and variance

- Example: assume a Bernoulli distribution $p(x; \theta)$ with true $\mathcal{X}_1 = 1, 1, 0, 0, 0, 0, 0, 1, 1 \rightarrow \hat{\theta} = 0.4$ parameter $\theta = 0.3$. We want to estimate θ from a sample $\mathcal{X}_2 = 0, 1, 0, 0, 1, 0, 0, 0, 0 \rightarrow \hat{\theta} = 0.2$ $\{x_1, \dots, x_N\}$ of N iid tosses using $\hat{\theta} = \frac{1}{N} \sum_{n=1}^N x_n$ as estimator. ...
But, the estimated $\hat{\theta}$ itself varies depending on the sample!
Indeed, $\hat{\theta}$ is a r.v. with a certain average and variance (over all possible samples \mathcal{X}). We'd like its average to be close to θ and its variance to be small.

Another ex: repeated measurements of your weight $x \in \mathbb{R}$ in a balance, assuming $x \sim \mathcal{N}(\mu, \sigma^2)$ and an estimator $\hat{\mu} = \frac{1}{N} \sum_{n=1}^N x_n$.

- *Statistic* $\phi(\mathcal{X})$: any value that is calculated from a sample \mathcal{X} (e.g. average, maximum...). It is a r.v. with an expectation (over samples) $E_{\mathcal{X}} \{\phi(\mathcal{X})\}$ and a variance $E_{\mathcal{X}} \{(\phi(\mathcal{X}) - E_{\mathcal{X}} \{\phi(\mathcal{X})\})^2\}$.

Ex.: if the sample \mathcal{X} has N points $\mathbf{x}_1, \dots, \mathbf{x}_N$:

$$E_{\mathcal{X}} \{\phi(\mathcal{X})\} = \int \phi(\mathbf{x}_1, \dots, \mathbf{x}_N) p(\mathbf{x}_1, \dots, \mathbf{x}_N) d\mathbf{x}_1 \dots d\mathbf{x}_N \stackrel{\text{iid}}{=} \int \phi(\mathbf{x}_1, \dots, \mathbf{x}_N) p(\mathbf{x}_1) \dots p(\mathbf{x}_N) d\mathbf{x}_1 \dots d\mathbf{x}_N.$$

- $\mathcal{X} = (x_1, \dots, x_N)$ iid sample from $p(x; \theta)$. Let $\phi(\mathcal{X})$ be an *estimator* for θ . How good is it?

mean square error of the estimator ϕ : $\text{error}(\phi, \theta) = E_{\mathcal{X}} \{(\phi(\mathcal{X}) - \theta)^2\}$.

- *Bias* of the estimator: $b_{\theta}(\phi) = E_{\mathcal{X}} \{\phi(\mathcal{X})\} - \theta$. How much the expected value of the estimator over samples differs from the true parameter value.

If $b_{\theta}(\phi) = 0$ for all θ values: *unbiased estimator*.

Ex: the sample average $\bar{x} = \frac{1}{N} \sum_{n=1}^N x_n$ is an unbiased estimator of the true mean μ ?, regardless of the distribution $p(x)$. The sample variance $\frac{1}{N} \sum_{n=1}^N (x_n - \bar{x})^2$ is a *biased* estimator of the true variance?

- *Variance* of the estimator: $\text{var} \{\phi\} = E_{\mathcal{X}} \{(\phi(\mathcal{X}) - E_{\mathcal{X}} \{\phi(\mathcal{X})\})^2\}$. How much the estimator varies around its expected value from one sample to another.

If $\text{var} \{\phi\} \rightarrow 0$ as $N \rightarrow \infty$: *consistent estimator*.


Ex: the sample average is a consistent estimator of the true mean?

- *Bias-variance decomposition*:

$$\text{error}(\phi, \theta) = E_{\mathcal{X}} \{(\phi - \theta)^2\} \stackrel{?}{=} \underbrace{E_{\mathcal{X}} \{(\phi - E_{\mathcal{X}} \{\phi\})^2\}}_{\text{variance}} + \underbrace{(E_{\mathcal{X}} \{\phi\} - \theta)^2}_{\text{bias}^2} = \text{var} \{\phi\} + b_{\theta}^2(\phi).$$

We want estimators that have both low bias and low variance; this is difficult.

- Ex: assume a Gaussian distribution $p(x; \mu, \sigma^2)$. We want to estimate μ from a sample $\{x_1, \dots, x_N\}$ of N iid tosses using each of the following estimators: $\hat{\mu} = \frac{1}{N} \sum_{n=1}^N x_n$, $\hat{\mu} = 7$, $\hat{\mu} = x_1$, $\hat{\mu} = x_1 x_2$.

 What is their bias, variance and error?

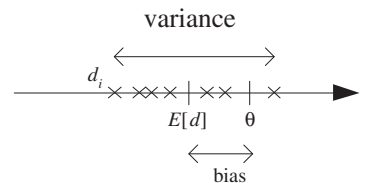
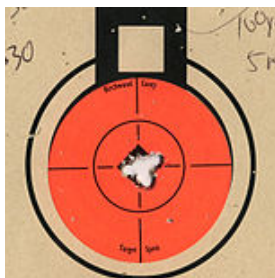


Illustration: estimate the bullseye location given the location of N shots at it.



bias↓, var↓



bias↓, var↑



bias↑, var↓

all over the place
outside the picture

bias↑, var↑

Tuning model complexity: bias-variance dilemma

The ideal regression function: the conditional mean $E\{y|x\}$

- Consider the regression setting with a true, unknown regression function f and additive noise ϵ : $y = f(x) + \epsilon$. Given a sample $\mathcal{X} = \{(x_n, y_n)\}_{n=1}^N$ drawn iid from $p(x, y)$, we construct a regression estimate $h(x)$. Then, for any h :

$$\text{Expected square error at } x: E\{(y - h(x))^2|x\} \stackrel{?}{=} \underbrace{E\{(y - E\{y|x\})^2|x\}}_{\text{noise}} + \underbrace{(E\{y|x\} - h(x))^2}_{\text{squared error wrt } E\{y|x\}}.$$

The expectations are over the joint density $p(x, y)$ for fixed x , or, equivalently[?], over $p(y|x)$.

- *Noise term*: variance of y given x . Equal to the variance of the noise ϵ added, independent of h or \mathcal{X} . Can never be removed no matter which estimator we use.
- *Squared error term*: how much the estimate $h(x)$ deviates (at each x) from the regression function $E\{y|x\}$. Depends on h and \mathcal{X} . It is zero if $h(x) = E\{y|x\}$ for each x .

The ideal[?], optimal regression function is $h(x) = f(x) = E\{y|x\}$ (*conditional mean* of $p(y|x)$).

The bias-variance decomposition with the estimator of a curve f

- Consider $h(x)$ as an estimate (for each x) of the true $f(x)$. Bias-variance decomposition at x :

$$E_{\mathcal{X}}\{(E\{y|x\} - h(x))^2|x\} = \underbrace{(E\{y|x\} - E_{\mathcal{X}}\{h(x)\})^2}_{\text{bias}^2} + \underbrace{E_{\mathcal{X}}\{(h(x) - E_{\mathcal{X}}\{h(x)\})^2\}}_{\text{variance}}.$$

The expectation $E_{\mathcal{X}}\{\cdot\}$ is over samples \mathcal{X} of size N drawn from $p(x, y)$. The other expectations are over $p(y|x)$.

- *Bias*: how much $h(x)$ is wrong on average over different samples.
- *Variance*: how much $h(x)$ fluctuates around its expected value as the sample varies.

We want both to be small. Ex: let h_m be the estimator using a sample \mathcal{X}_m :

- $h_m(x) = 2 \forall x$ (constant fit): zero var, high bias (unless $f(x) \approx 2 \forall x$), high total error.
 - $h_m(x) = \frac{1}{N} \sum_{n=1}^N y_n^{(m)}$ (average of sample \mathcal{X}_m): higher var, lower bias, lower total error.
 - $h_m(x)$ = polynomial of degree k : if $k \uparrow$ then bias \downarrow , var \uparrow .
- Low bias requires sufficiently flexible models, but flexible models have high variance. As we increase complexity, bias decreases (better fit to data) and variance increases (fit varies more with data). The optimal model has the best trade-off between bias and variance.
 - *Underfitting*: model class doesn't contain the solution (it is not flexible enough) \Rightarrow bias.
 - *Overfitting*: model class too general and also learns the noise \Rightarrow variance. Also, the variance due to the sample decreases as the sample size increases.

Model selection procedures

- Methods to find the model complexity that is best suited to the data.
- **Cross-validation**: the method most used in practice. We cannot calculate the bias and variance (since we don't know the true function f), but we can estimate the total error.
 - Given a dataset \mathcal{X} , divide it into 3 disjoint parts (by sampling at random without replacement from \mathcal{X}) as **training**, **validation** and **test** sets: \mathcal{T} , \mathcal{V} and \mathcal{T}' .
 - Train candidate models of different complexities on \mathcal{T} . Ex: polynomials of degree $0, 1, \dots, K$.
 - Pick the trained model that gives lowest error on \mathcal{V} . This is the final model.
 - Estimate the generalization error of the final model by its error on \mathcal{T}' .

If \mathcal{X} is small in sample size, use *K-fold cross-validation*, to make better use of the available data.

This works because, as the model complexity increases:

- the training error keeps decreasing;
 - the validation error first decreases then increases (or stays about constant).
- **Regularization**: instead of minimizing just the error on the data, minimize error + **penalty**:

$$\min_h \sum_{n=1}^N (y_n - h(x_n))^2 + \lambda C(h)$$

where $C(h) \geq 0$ measures the model complexity and $\lambda \geq 0$. Ex:

- Model selection criteria (AIC, BIC...): $C(h) \propto$ number of parameters in h (model size). λ is set to a certain constant depending on the criterion. We try multiple model sizes.
- Smoothness penalty: e.g. $C(h) = \sum_{i=0}^k w_i^2$ for polynomials of degree k , $h(x) = \sum_{i=0}^k w_i x^i$. λ is set by cross-validation. We try a single, relatively large model size.