Introduction to Machine Learning

Lecture 6

Instructor:
Dr. Tom Arodz

Probability rules recap

- **P(AB)=P(A)P(B) - Independent events:**

- P(snow,dog_barked) = P(snow)*P(dog_barked)
  - P(dog_barked)=0.01, P(snow)= 0.0526
    P(snow,dog_barked) = 0.00526

  P(snow | dog_barked) = P(snow)
  because P(A|B)=P(AB)/P(B)=P(A) since P(AB)=P(A)P(B)
  - P(dog_barked)=0.01, P(snow)= 0.0526
    P(snow,dog_barked) = 0.00526
  - Informal: *knowing if dog barked doesn't give us any additional info about snow*

- **P(i.i.d. dataset)=P(sample1)*P(sample2)*P(sample3)…**
  - Example: Gaussian-distributed dataset
  - P(x|mean) = exp($-(x-mean)^2$)
  - P(dataset| mean) = exp($-(x1-mean)^2$)
    *exp($-(x2-mean)^2$)
    *exp($-(x3-mean)^2$)

Probability rules recap

- **Conditional independence: P(AB|C)=P(A|C)P(B|C)**
  - 3 variables: car length (**A**), sportiness (**B**), type (**C**, car or van)
  - overall, length (**A**) and sportiness (**B**) are correlated
    - cars are shorter and more sporty than vans

  but:
  - within cars ("|C=car"), sportiness and length are independent
  - within vans ("|C=van"), sportiness and length are independent
  - P(sportiness,length) != P(sportiness)P(length)
    - Not independent
    - but
  - P(sportiness,length | vehicle_type)
    = P(sportiness|vehicle_type)*P(length|vehicle_type)
    - Conditionally independent

Probability rules recap

- **Conditional independence: P(AB|C)=P(A|C)P(B|C)**

- **Alternatively: P(A|BC)=P(A|C)**
  - Adding knowledge of B happening to knowledge of C happening doesn't change probability of A (similar to P(snow|bark)=P(snow), but "|C")
  - **If we assume conditional independence of A&B given C, we can eliminate B and simplify our equations**

- From definitions:

  P(AB|C) = P(ABC) / P(C)

  P(A|C) = P(AC) / P(C)

  P(B|C) = P(BC) / P(C)

- Together, blue expands using definitions as:
  - **P(ABC) / P(C) = [P(AC) / P(C)] * [P(BC) / P(C)]**
  - P(ABC) = [P(AC) * P(BC)] / P(C)
- Also from definition:
  - P(A|BC) = P(ABC) / P(BC)
- Substitute green gives us: P(A|BC) = {[P(AC) * P(BC)] / P(C)} / P(BC)
- Which is: **P(A|BC)** = P(AC) / P(C) = **P(A|C)**

Probability rules recap

- **Law of total probability**
  - **$P(A)=\Sigma_i P(A|\ B_i)\ P(B_i)$**
  - "B" may or may not be related to "A"
  - The equality is always true by rules of math, as long as we sum up over all options B_i for B in the summation
    - **P(snow) =**
      P(snow|dog_barked)P(dog_barked)
      +P(snow|dog_didn't_bark)P(dog_didn't_bark)
      =
      P(snow)P(dog_barked)
      +P(snow)(1-P(dog_barked))
      = P(snow)(P(dog_barked) + 1 - P(dog_barked))
      P(snow)*1

Probability rules recap

- **Simplifying the language:**
  likelihood L(A|B)* vs *conditional probability P(B|A)

  - **We define L(A|B) to be proportional to P(B|A)**
    - **It's not normalized to sum(A)=1!**

  - **Likelihood of subzero (given there's snow)** is proportional to **conditional probability of snow given subzero:**
    L(subzero|snow) = P(snow|subzero) = 0. 5

  - Instead of saying "**conditional probability of B given A**" we can simply say "**likelihood of A**" (for a given B, often with B implied from context and not mentioned)
    - "find A with highest **conditional probability of B given A**, for given B"
      can be shortened to:
      "find **A with highest likelihood**, for given B".

Recap: MLE

- Initial formula:

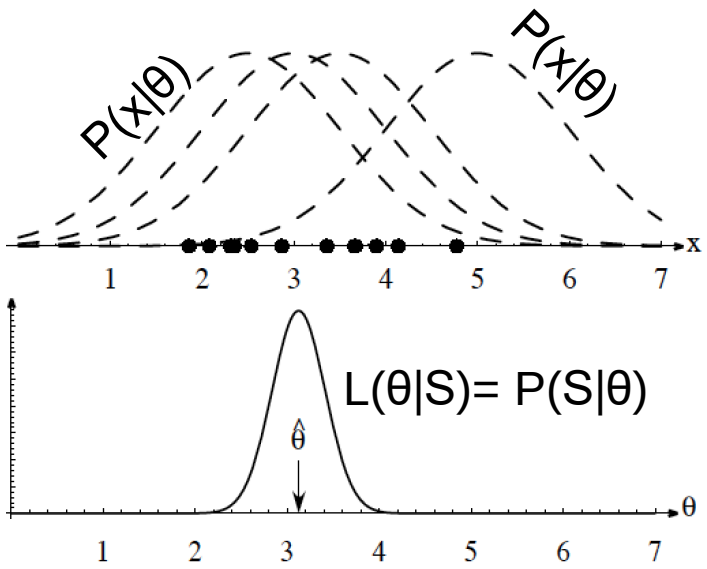
$$p(y_i \mid x,S) = p(x \mid y_i,S)\, p(y_i \mid S) \,/\, \Sigma_i\, p(x \mid y_i,S)\, P(y_i \mid S)$$

  - Maximum likelihood estimation (MLE) :
    - find single θ_i with highest $p(\theta_i \mid y_i, S)$,
      - $\theta_i = \arg\max_\theta P(S|\theta)$ or $\theta_i = \arg\max_\theta P(S|\theta)P(\theta)$
    - Use that parameter, it encapsulates all that we learned from the training set S:

$$p(x \mid y_i,S) = p(x \mid y_i,\theta_i)$$

- Final formula:

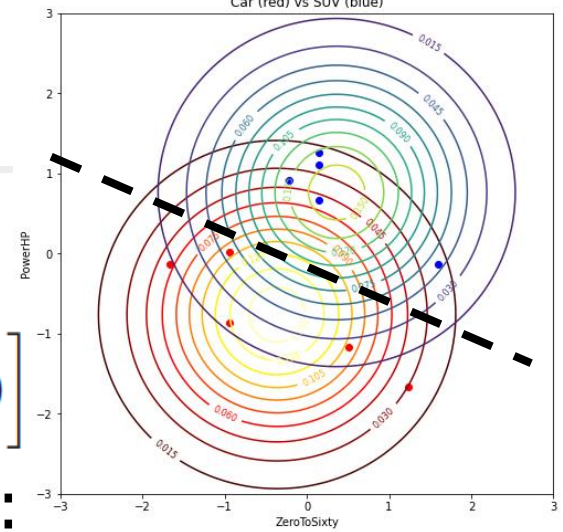
$$p(y_i \mid x,S) \sim p(x \mid y_i, \theta_i)\, p(y_i \mid S)$$



2D Gaussian – Car vs SUV

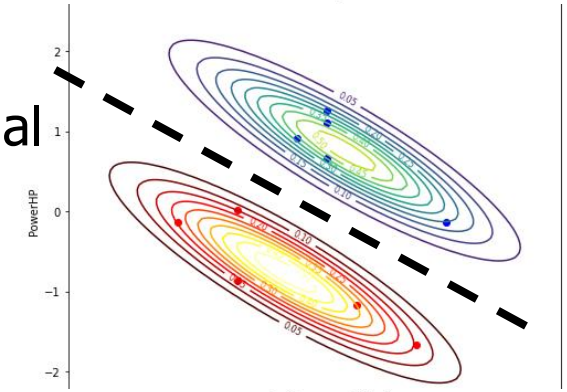
- Fit a gaussian (2 features => 2D Gaussian) to each class (Car, SUV)

$$p(\mathbf{x}) = \frac{1}{(2\pi)^{d/2}|\boldsymbol{\Sigma}|^{1/2}} \exp\left[-\frac{1}{2}(\mathbf{x}-\boldsymbol{\mu})^t \boldsymbol{\Sigma}^{-1}(\mathbf{x}-\boldsymbol{\mu})\right]$$



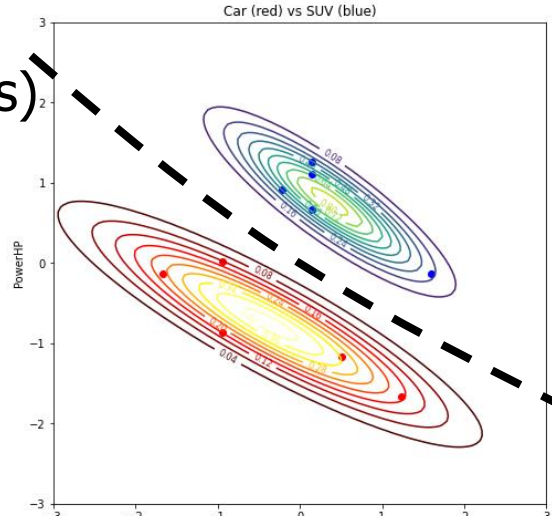
- We estimate mean and covariance from data:
  - Simplest option is to just estimate the mean for each class, assume covariance is fixed as a diagonal (identity) matrix

$$\overline{\mathbf{x}} = \frac{1}{n}\sum_{j=1}^{n} \mathbf{x}_j$$



  - More complex option is to estimate one covariance S for all data (average of covariances for each class)

$$S = \sum_{j=1}^{n} (\mathbf{x}_j - \overline{\mathbf{x}})(\mathbf{x}_j - \overline{\mathbf{x}})^T$$



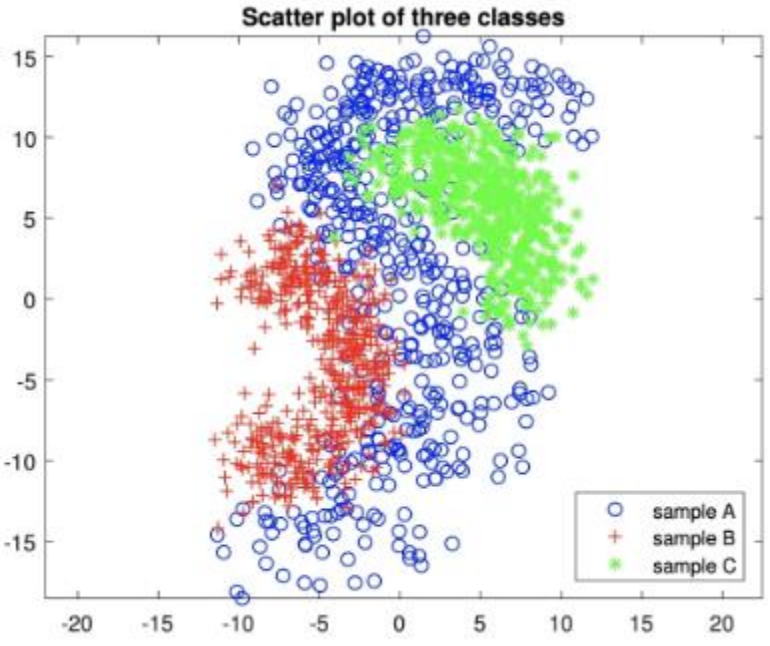
  - Even more complex: separate covariance for each class
    - Decision boundary no longer linear

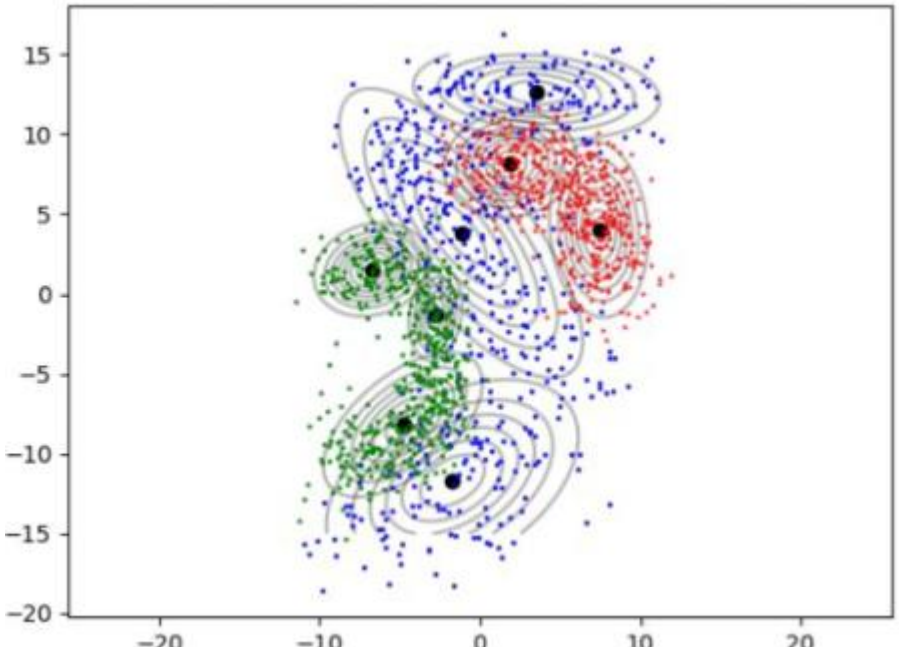
Is one Gaussian per class ideal?

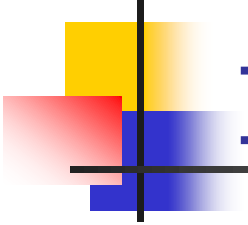
We estimated a single, best-fitting (according to maximum likelihood) Gaussian for each class

What if our data is not Gaussian?

Gaussian Mixture Models: fit multiple Gaussians







Is one Gaussian per class ideal?

We estimated a single, best-fitting (according to maximum likelihood (MLE)) Gaussian for each class

If our data is (approximately) Gaussian, is one gaussian ideal?

Let's assume yes, but:

Did we pick the right one? We only had limited training data….

MLE for hidden dice

- Example ("*hidden dice*"):
  - we have positive integer samples
  - from a uniform distribution with an unknown maximum M
    - We only know M<=10
  - we observe set S of four values: 2, 4, 7, 8
  - Our goal: estimate p(x|S)

    - Most interestingly,
      what is the probability of seeing
      9 and 10
      based on the above information?
      - i.e., p(x=9|S)=? and p(x=10|S)=?

MLE for hidden dice

- Example ("*hidden dice*"):
  - we have positive integer samples
  - from a uniform distribution with an unknown maximum M
    - We only know M<=10
  - we observe set S of four values: 2, 4, 7, 8
  - Our goal: estimate p(x|S)
- Maximum likelihood (MLE) approach:
  - We choose M for which P(S|M) is highest
    P(S|M)=P({2,4,7,8}|M) = P(2|M)*P(4|M)*P(7|M)*P(8|M)
  - Based on that M, we assume p(x|S) = p(x|M)
  - What's the M?

MLE for hidden dice

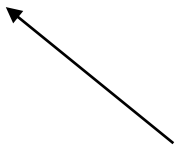
- Example:
  - we have positive integer samples
  - from a uniform distribution with an unknown maximum M
    - We only know M<=10
    - P(x|M)=1/M
  - we observe set S of four values: 2, 4, 7, 8
  - what is p(x|S)?

- Maximum likelihood (MLE) approach:
  - We choose M that leads to highest value of:
    $$P(S|M)=P(2|M)*P(4|M)*P(7|M)*P(8|M)$$
  - If M<8 P(S|M)=0
  - If M=8 $P(S|M)=(1/8)^4$
  - If M=9 $P(S|M)=(1/9)^4$
  - If M=10 $P(S|M)=(1/10)^4$

MLE choice:
highest prob. of training set S among all possible values of M

MLE for hidden dice

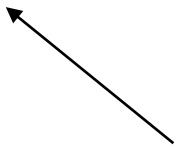
- Example:
  - we have positive integer samples
  - from a uniform distribution with an unknown maximum M
    - We only know M<=10
  - we observe set S of four values: 2, 4, 7, 8
  - what is p(x|S)?
- Maximum likelihood (MLE) approach:
  - We choose M=8, and have: p(x|S)=p(x|M=8)
  - P(x|S) = 1/8= 0.125 for any x<=8
    = 0 for x=9, x=10



Would you give seeing 9 when we saw 2,4,7,8 a 0% probability?

MLE for hidden dice

- Example:
  - we have positive integer samples
  - from a uniform distribution with an unknown maximum M
    - We only know M<=10
  - we observe set S of four values: 2, 4, 7, 8
  - what is p(x|S)?
- Maximum likelihood (MLE) approach:
  - We choose M=8, and have: p(x|S)=p(x|M=8)
  - P(x|S) = 1/8= 0.125 for any x<=8
    = 0 for x=9, x=10



Would you give seeing 9 when we saw 2,4,7,8 a 0% probability?

MLE provides a *point estimate*

- Maximum likelihood (MLE) :
  - find single θ_i with highest $p(\theta_i \mid y_i, S)$,
    - $\theta_i = \arg\max_\theta P(S|\theta)$ or $\theta_i = \arg\max_\theta P(S|\theta)P(\theta)$
  - Use that parameter, it encapsulates all that we learned from the training set S: $p(x \mid y_i, S) = p(x \mid y_i, \theta_i)$
- For the simple hidden dice problem, MLE resulted in conclusion that the hidden dice has M=8 faces
  - **That single number (M=8) is all we get**
- It is a serious limitation of the MLE approach:
  - We get **one**, most likely, value(s) for the parameter(s)
  - It's called a *point estimate*
- Most "modern" machine learning methods used in practice are similar
  - E.g. training results in a single neural network (a single set of weights)
  - An alternative, more complete approach (Bayesian learning) is often too computationally complex

MLE vs Bayesian estimation

- Known distribution shape, unknown parameters of the distribution (M here), need to be estimated from data
- Maximum likelihood (MLE) approach:
  - We try to estimate single, most likely values of parameters from the training set
  - We use that single estimated value in all reasoning
    - $p(x \mid S) = p(x \mid M)$

- Bayesian learning approach:
  - We treat parameters as a random variable
  - We treat the training set as evidence that allows us to assign probabilities to different values of parameters
    - P(M=8), P(M=9), P(M=10)
  - We use all possible parameter values M, but each value carries different weight, based on its probability P(M|S)
    - $p(x \mid S) = \Sigma_M\, p(x \mid M)\, P(M \mid S)$

$P(A)=\Sigma_i P(A|B_i)P(B_i)$
$P(A|C)=\Sigma_i P(A|B_i C)P(B_i|C)$

Bayesian estimation

- $p(x \mid S) = \Sigma_i\, p(x \mid \theta_i)\, P(\theta_i \mid S)$

  $P(\theta_i \mid S) = P(S|\theta_i)P(\theta_i) \,/\, \Sigma_i\, P(S|\theta_i)P(\theta_i)$

  - Let's assume $P(M=1)=P(M=2)=\ldots=P(M=10)=0.1$
- $P(S|\theta_i) = \Pi_k\, P(x_k|\theta_i)$
  - $P(x|S) = p(x|M=8)\, P(M=8|S) + p(x|M=9)\, P(M=9|S) + p(x|M=10)\, P(M=10|S)$
    - $P(M=8|S) = P(S|M=8)P(M=8) \,/\, \ldots = (1/8)^4 \,/\, [\,(1/8)^4+(1/9)^4+(1/10)^4\,] = 0.4916$
    - $P(M=9|S) = P(S|M=9)P(M=9) \,/\, \ldots = (1/9)^4 \,/\, [\,(1/8)^4+(1/9)^4+(1/10)^4\,] = 0.3069$
    - $P(M=10|S) = P(S|M=10)P(M=10) \,/\, \ldots = (1/10)^4 \,/\, [\,(1/8)^4+(1/9)^4+(1/10)^4\,] = 0.2014$
- **$P(x|S) = 0.49\, p(x|M=8) + 0.31\, p(x|M=9) + 0.20\, p(x|M=10)$**
  - $P(x=9|S) = 0.49*0 + 0.31*1/9 + 0.20*1/10 = 0.054$ (not 0.0 as in MLE)
  - $P(x=8|S) = 0.49*1/8 + 0.31*1/9 + 0.20*1/10 = 0.116$ (not 0.125)

MLE vs Bayesian estimation

- Example:
  - we have positive integer samples
  - from a uniform distribution with an unknown maximum M
    - We only know M<=10
    - P(x|M)=1/M
  - we observe set S of four values: 2, 4, 7, 8
  - what is p(x|S)?
- Let's get more data:
  - We observed 4 more points (now we have 8 samples)
  - Still, we haven't see 9 or 10,
  - the highest number we've seen is still 8
- How would that affect MLE estimate of P(x|S)?
- How would that affect Bayesian estimate of P(x|S)?

MLE vs Bayesian estimation

- Maximum likelihood (ML) approach:
  - We choose M that leads to highest value of:
    $P(S|M)=P(2|M)*P(4|M)*P(7|M)*P(8|M)$
    If $M<8$ $P(S|M)=0$
  - If $M=8$ $P(S|M)=(1/8)^8$
  - If $M=9$ $P(S|M)=(1/9)^8$
  - If $M=10$ $P(S|M)=(1/10)^8$

  - Still the same estimate:

  - $P(x|S) = 1/8 = 0.125$ for any $x<=8$
    $= 0$ for $x=9$, $x=10$

Bayesian estimation

- $p(x \mid S)=\Sigma_i\, p(x \mid \theta_i)\, P(\theta_i \mid S)$

  $P(\theta_i \mid S) = P(S|\theta_i)P(\theta_i) / \Sigma_i P(S|\theta_i)P(\theta_i)$

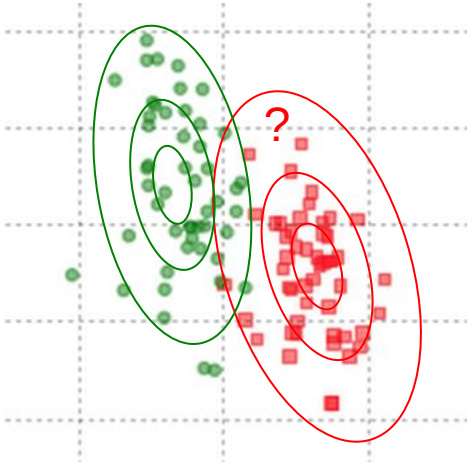
  - Let's assume P(M=1)=P(M=2)=…=P(M=10)=0.1
- $P(S|\theta_i) = \Pi_k P(x_k|\theta_i)$
  - P(x|S)= p(x|M=8) P(M=8|S) + p(x|M=9) P(M=9|S) + p(x|M=10) P(M=10|S)
    - P(M=8|S) = P(S|M=8)P(M=8) / …
      = $(1/8)^8$ / [$(1/8)^8$+$(1/9)^8$+ $(1/10)^8$] =0.6420
    - P(M=9|S) = P(S|M=9)P(M=9) / …
      = $(1/9)^8$ / [$(1/8)^8$+$(1/9)^8$+ $(1/10)^8$] =0.2502
    - P(M=10|S) = P(S|M=10)P(M=10) / …
      = $(1/10)^8$ / [$(1/8)^8$+$(1/9)^8$+ $(1/10)^8$] =0.1077
  - P(x|S)= 0.64 p(x|M=8) + 0.25 p(x|M=9) + 0.11 p(x|M=10)
  - P(x=9|S)= 0 + 0.25/9 + 0.11/10 = **0.038** (not 0 as in ML)
    - was **0.054** with 4 instead of 8 samples
  - P(x=8|S)= 0.65/8 + 0.25/9 + 0.11/10 = **0.118** (not 0.125)
    - was **0.116**

Bayesian learning for classification

- For predictions, we use $p(y_i|x,S)$, we can get it from:
  - $p(x \mid y_i,S)$
  - If you have seen m_r red examples in the training set, what's the probability of seeing "red ?", i.e., what's $p(x \mid red,S)$
    - Easier if we fit a distribution to the points $p(x \mid red,S) = \mathcal{N}(x \mid \theta_{best\_for\_red})$

$P(A)=\Sigma_i P(A|B_i)P(B_i)$
$P(A|C)=\Sigma_i P(A|B_iC)P(B_i|C)$

  - But why pick just one "best" Gaussian?
  - **Bayesian learning:** average over all possible Gaussians, weighted by how good a fit to the data they are



  - $p(x \mid y_i,S) = \Sigma_{\theta i}\, \mathcal{N}(x \mid y_i, \theta_i, S)\, p(\theta_i \mid y_i, S)$
    $= \Sigma_{\theta i}\, \mathcal{N}(x \mid y_i,\theta_i)\, p(\theta_i \mid y_i, S)$

    x and S are conditionally independent given θ_i, i.e., if we know θ_i, knowing also S doesn't change our knowledge of x

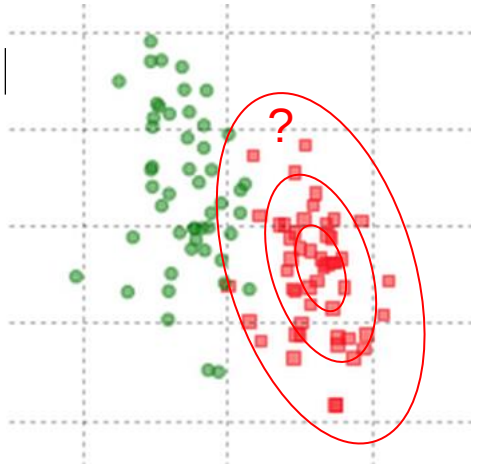
Bayesian learning for classification

- $p(y_i \mid x,S) = p(x \mid y_i,S)\, p(y_i \mid S) \,/\, \Sigma_i\, p(x \mid y_i,S)$
  - $p(x \mid y_i,S) = \Sigma_{\theta i}\, p(x \mid y_i,\theta_i)\, p(\theta_i \mid y_i, S)$

    or in fact $= \int p(x \mid y_i,\theta_i)\, p(\theta_i \mid y_i, S)\, d\theta_i$

    because θ_i is not discrete

    it's a vector of real numbers

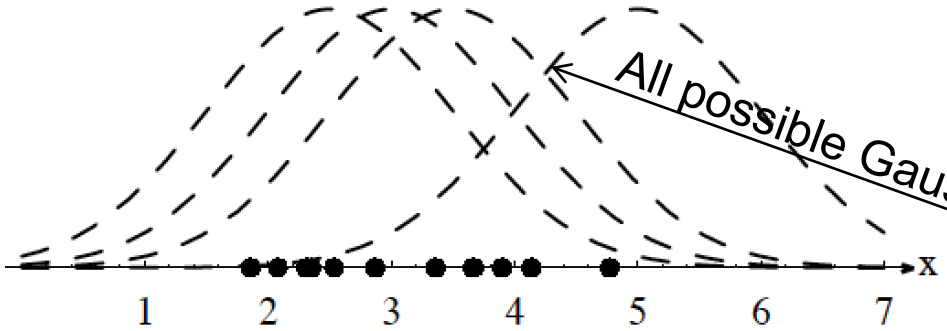


- What do the terms on the right mean?
  - $p(x \mid y_i,\theta_i)$ means: the distribution for class y_i is parameterized by a vector θ_i

    i.e., probability of x depends on the value of θ_i
    - We assume we know the distribution shape $p(x \mid \theta_i)$
    - E.g. normal distribution: θ_i is info about mean and covariance
    - If we know what θ_i is, we can easily calculate probability of x
  - $p(\theta_i \mid y_i, S)$ means: the value of parameter θ_i depends on the samples from class i in the training set S
    - E.g. some Gaussians (some means) are a "better fit" to the data in the training set than other

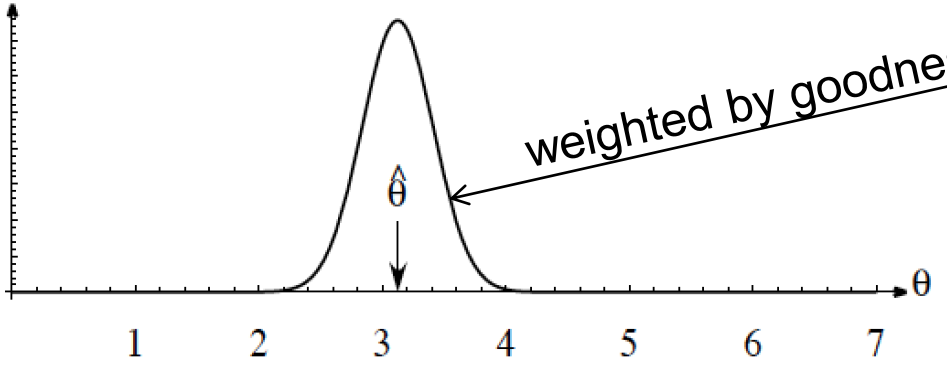
1D Gaussian example

- Take into account all possible Gaussians, weighted by their "goodness of fit"



$p(x \mid y_i, S)$

$= \Sigma_{\theta i} \mathcal{N}(x \mid y_i, \theta_i)\, p(\theta_i \mid y_i, S)$



MLE vs Bayesian learning

- $p(y_i \mid x,S) = p(x \mid y_i,S)\, p(y_i \mid S) / \Sigma_i\, p(x \mid y_i,S)\, P(y_i \mid S)$

  $\mathbf{p(x \mid y_i,S) = \Sigma_{\theta i}\, p(x \mid \theta_i, y_i, S)\, p(\theta_i \mid y_i, S)}$

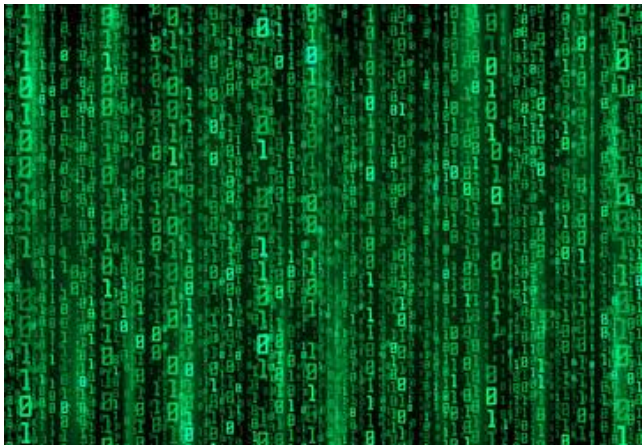
$P(A)=\Sigma_i P(A|B_i)P(B_i)$
$P(A|C)=\Sigma_i P(A|B_iC)P(B_i|C)$

- Two options:
  - Bayesian estimation – use full formula:
    - use formula $p(x \mid y_i,S) = \Sigma_{\theta i}\, p(x \mid y_i,\theta_i)\, p(\theta_i \mid y_i, S)$
    - No approximations, we're using all possible values of θ_i weighing them by their probability given training set S
      - often computationally v. intensive!
  - Maximum likelihood (MLE) :
    - find single θ_i with highest probability of the training set, $p(S \mid \theta_i)$,
      - $\theta_i = \arg\max_\theta P(S|\theta)$
    - Make an approximation:
      - since all other θ_i had smaller $p(S \mid \theta_i)$, approximate them by 0
    - We end up with an approximation, but a much simpler formula: $p(x \mid y_i,S) = p(x \mid y_i,\theta_i)$

ML Approach #2 - problems

- E.g. Maximum likelihood (MLE) with Gaussians:
  - Training:
    - for each class, find single θ_{class} with highest $p(S_{class} \mid \theta_{class})$
    - θ_{class}=arg $\max_{\theta}$ `mult_normal(θ).pdf(S_class)`
  - Predicting: use those θ_{class} to make probabilistic predictions:
    - p(class|x,S)=`mult_normal(θ_class).pdf(x)*P[class]`
- Problems:
  - Why Gaussian? Why not other distribution?
  - Any distribution on input features limits us to working directly in input features space
    - Any common "textbook" distribution will be a rather simple formula that uses raw input features
    - Prevents us from doing more flexible, complex reasoning

"*true hackers browse the web in binary*" ;)



Towards ML Approach #3

- Probability distribution governing the problem is not (*fully*) known
  - We can only learn about the problem from the training set

- Machine Learning Approach #2 (dist. *not **fully** known*)
  - E.g. we know (assume) it's e.g. a Gaussian, we just don't know the mean and/or covariance matrix
  - Typically, the distribution shape/formula is specified:
    - in the original features space
    - separate for each class (i.e., p(x|y=class))
  - Typically, we use per-class Parametric Models (e.g. MLE or Bayesian learning):
    - Training: we estimate parameters of known distribution shape from training data
    - Inference: we use the distributions with those parameters to make predictions

Towards ML Approach #3

- **Machine Learning Approach #3**
  - We typically work directly with p(y|x), instead of per-class p(x|y=a class)
  - Shape of p(y|x) is often not as obvious as p(x|y)
    - It involves all classes together
    - while p(x|y) is class-specific, and often can be assumed/derived based on domain knowledge about generative model (how samples/features arise) for the class
  - We don't **directly** assume anything about the distribution shape
  - Instead, we model the distribution through some function class `some_f` that has trainable parameters θ
    - a specific vector of values θ_{opt} of these parameters gives us a specific function to make predictions: $f(x)=some\_f(\theta_{opt})(x)$
  - We still work in the original feature space
    - But, sometimes we may reinterpret models as providing new, better feature space

Towards ML Approach #3

- E.g. Maximum likelihood (MLE) with a distribution:
  - Training:
    - for each class, find single θ_{class} with highest $p(S_{class} \mid \theta_{class})$
    - θ_{class}=arg max$_\theta$ `distribution(θ).pdf(S_class)`
  - Predicting: use those θ_{class} to make probabilistic predictions:
    - p(class|x,S)=`distribution(θ_class).pdf(x)*P[class]`
- "Modern" machine learning (plug in f instead of distr.):
  - Predicting: use trained parameters θ_{opt}
    - p(class|x,S)=`some_f(θ_opt)(x).to_class_probs()`
  - Training:
    - Search for single value of parameters θ that minimizes:

θ_{opt} = arg max$_\theta$ `error_metric(S, some_f(θ)(x in S).to_class_probs())`

Towards ML Approach #3

- "Modern" machine learning:

```
minimize_θ: error_metric(S,
      some_f(θ)(x in S).to_class_probs())
```

  - Filling the gaps:
    - `some_f(θ)(x)` or `some_f(θ;x)`
      - Some function that takes two groups of inputs
        - Raw input features x
        - Trainable parameters `θ`
          - the parameters help us pick a specific function from a family of functions
    - `to_class_probs()`
      - `some_f(θ;x)` may not return probabilities, how to convert to: ≥0, sum=1
    - `error_metric`
      - Classification error? Do we need something more complex?
    - `minimize_θ`
      - how? For Gaussians in MLE, we had analytical solution, that's unlikely for `some_f`

The story so far…

- Ideal way of making predictions:
  - Perfect feature
  - Accurate, efficient simulation
  - Fully-known probability distribution p(y|x) from which data comes from
    - or p(x|y), we can use Bayes theorem to get p(y|x) from it
- Traditional machine learning:
  - heuristics / ad-hoc approaches (e.g. perceptron in HW1)
  - maximum likelihood (or Bayesian learning)
    - pick a distribution shape for p(x|y), based on domain knowledge
    - fit best distribution p(x|y) to each class in training data
    - get p(y|x) from these distributions using Bayes theorem
- "Modern" machine learning:
  - produce numbers that we can treat as p(y|x) from some function `f(x)` – not any specific distribution
  - Tweak the parameters θ of function `f(x;θ)` to make predictions better

Scheduling…

- HW2 is in Canvas
  - Due Tuesday, 10/1, 5pm
  - Gradescope autograder will become available by Wednesday

- On Wednesday, 9/18, we will have a class focused heavily on practical aspects / coding
  - I will upload a jupyter notbook to Canvas in advance (by tomorrow end of day)
  - Ideally, have a machine+setup ready to execute it during class
    - Either on your machine, or in the cloud e.g. on Google Colab (this requires you to set up an account in advance)
  - The class will be fully online, no in-person part