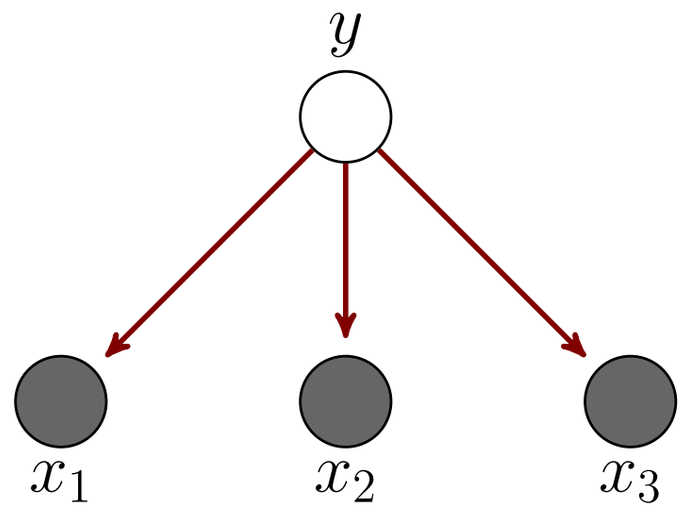
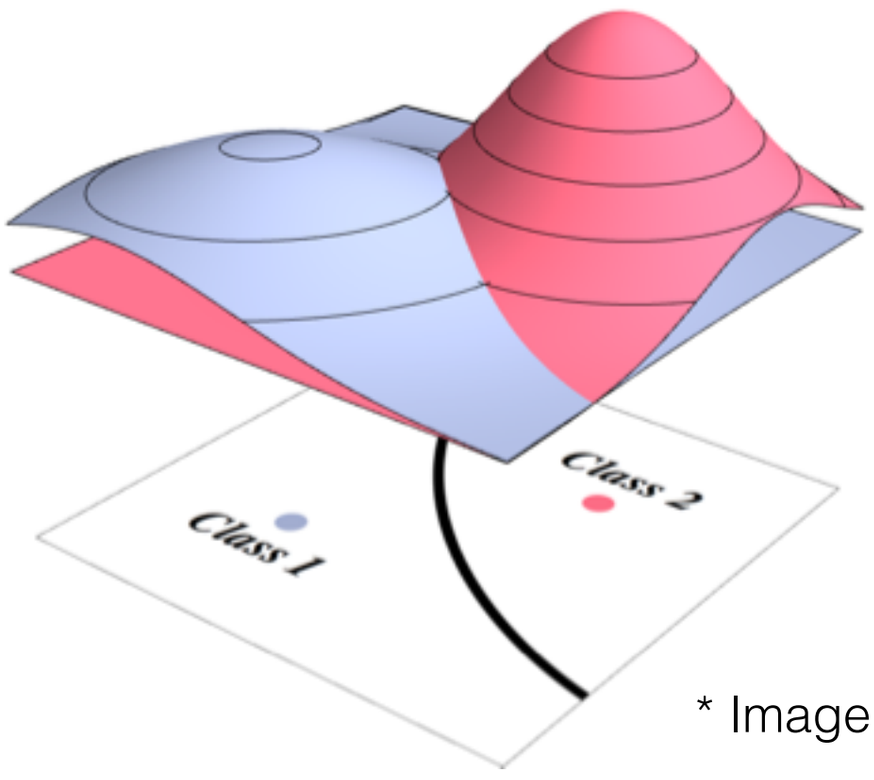


Generative models: an example with naive Bayes



* Image from Yaroslav Bulatov

Classification so far

- Have been learning p(y l x)
  - Either for binary classification, as a Bernoulli
  - Or for multi-class classification, as a Multinomial
- These are called **discriminative** classifiers
  - i.e., learning a function of x, to predict distribution over y or statistics about p(ylx) (e.g., f(x) = E[Y l x])
  - Do not learn the distribution over x itself
  - **Note**: the classifiers have been linear so far, but this is not a requirement for discriminative classifiers, f can be a nonlinear function of x

Discriminative versus generative

- Discriminative: learn p(y | x), as a function of x

- In generative learning,
  - learn p(x | y) p(y) (which gives the joint p(x, y) = p(xly) p(y))
  - compute p(x | y) p(y), which is proportional to p(y | x)

$$p(y|x) = \frac{p(x|y)p(y)}{p(x)}$$

- Question: how do we use p(xly) p(y) for prediction?

- Question: how do we learn p(xly) and p(y)?

- Question: why might we want to use generative models?

How to use generative models

- Our decision rule for using these probabilities is the same as with logistic regression: pick class with the highest probability

- Assume you have learned p(x l y) and p(y) from a dataset

- Compute

$$
\begin{aligned}
f(\mathbf{x}) &= \arg\max_{y \in \mathcal{Y}} p(y|\mathbf{x}) \\
&= \arg\max_{y \in \mathcal{Y}} p(\mathbf{x}|y)p(y)/p(\mathbf{x}) \\
&= \arg\max_{y \in \mathcal{Y}} p(\mathbf{x}|y)p(y)
\end{aligned}
$$

How to learn generative models

- For discriminative, had to choose distribution over y given x
  - e.g. p(y | x) is Gaussian for continuous y
  - e.g. p(y | x) is Poisson for y in {1, 2, 3, …}
  - e.g. p(y | x) is Bernoulli for y in {0,1}
- Parameters to p(y | x) were w such that E[Y | x] = f(xw)
- Now we need to choose the distribution p(x | y) and p(y)
- How do we pick p(x | y) and p(y)? What are the parameters?

How do we pick distributions more generally?

- For p(x), picked Gaussian, Bernoulli, Poisson, Gamma
  - depending on the values x could take, or looking at a plot
- For conditional distributions, like p(y | x), we picked the distribution based on the properties of y
  - Once the given features x are fixed, we are just learning a distribution over y (where parameters to that distribution depend on x)
- Imagine y is a 1-d Gaussian and x is a 2-d real-valued vector
- What might you pick for p(x | y) and p(y)?

What if we have binary features?

- For binary features x in {0,1}, binary y in {0,1}, what is p(x l y)?

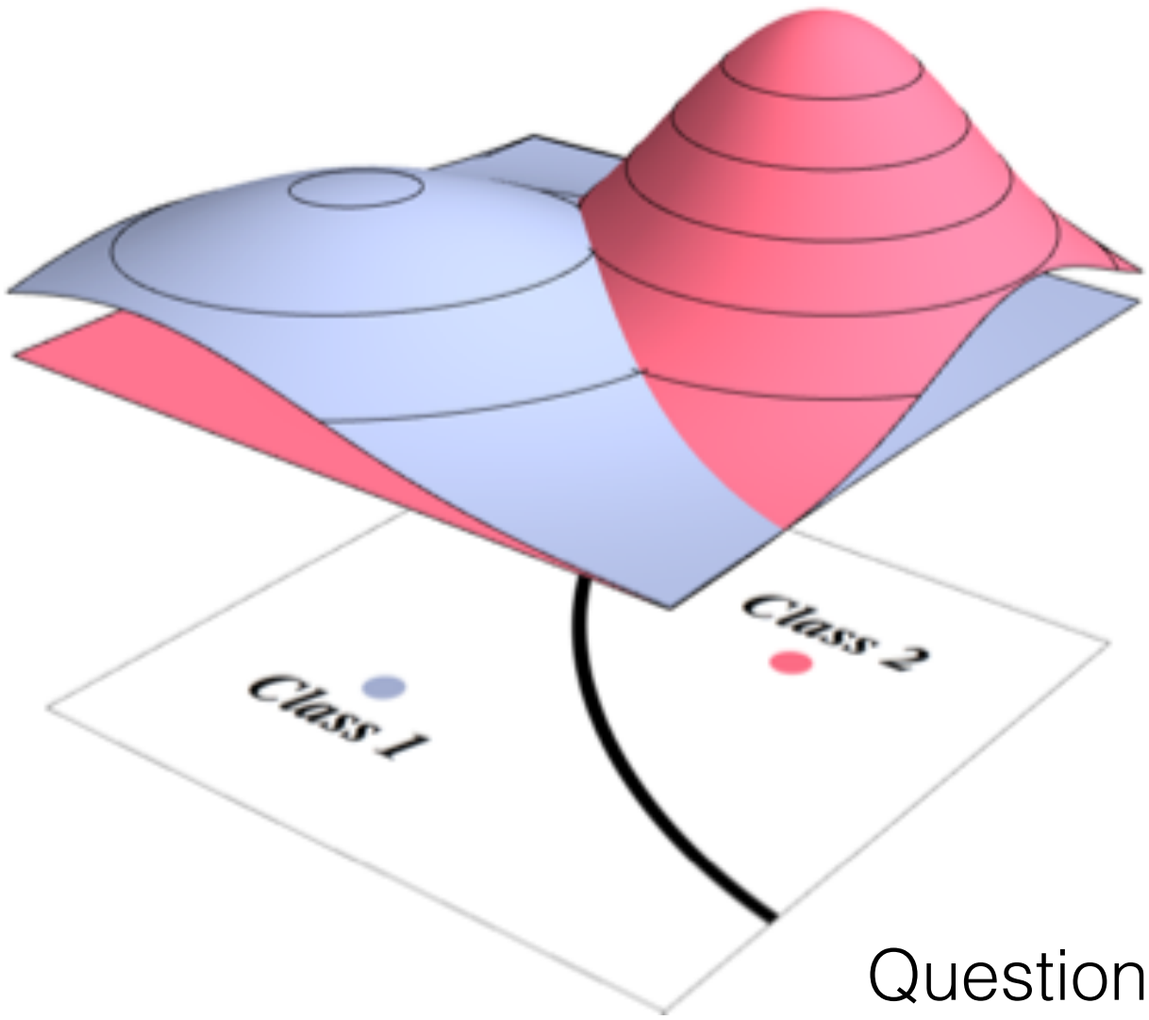
- How do we learn p(x l y)?

- How do we learn p(y)?

Another setting

- Imagine x is a 2-d real-valued variable and y is a Bernoulli
- Now what might you pick for p(x | y) and p(y)?

$$p(\mathbf{x}|y=1) = \mathcal{N}(\boldsymbol{\mu}_1, \boldsymbol{\Sigma}_1)$$



$$p(\mathbf{x}|y=2) = \mathcal{N}(\boldsymbol{\mu}_2, \boldsymbol{\Sigma}_2)$$

Question: Is p(x) Gaussian?

What if there are lots of features?

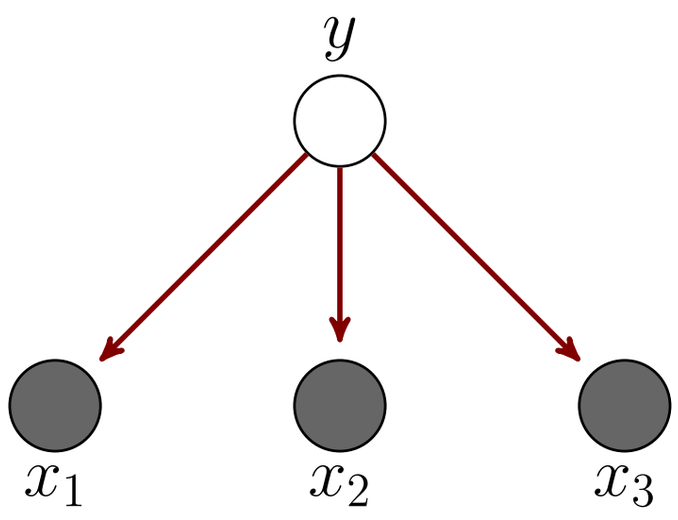
- Imagine x is a 1000-d random variable and y is a Bernoulli
- How can we determine what type of distribution to pick for x?
- What if we decide its a 1000-d multivariate Gaussian RV. Any issues with learning mean mu and covariance Sigma?

$$p(\mathbf{x}|y=1) = \mathcal{N}(\boldsymbol{\mu}_1, \boldsymbol{\Sigma}_1) \qquad p(\mathbf{x}|y=2) = \mathcal{N}(\boldsymbol{\mu}_2, \boldsymbol{\Sigma}_2)$$

Simplifying assumptions

- How do we realistically learn p(x l y)?
- One option is to make a (strong) conditional independence assumption: the features are independent given the label

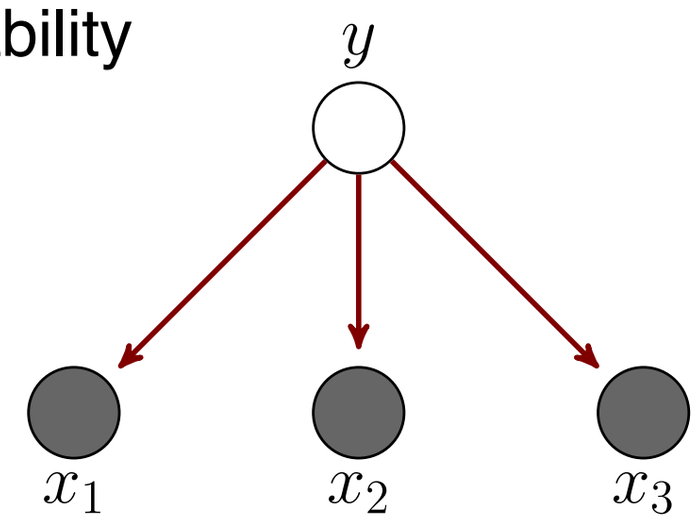
$$p(\mathbf{x}|y) = \prod_{i=1}^{d} p(x_i|y).$$



Conditional independence assumption

$$p(\mathbf{x}|y) = \prod_{j=1}^{d} p(x_j|y)$$

- Example: given a patient has the disease (y=1), attributes about patient are uncorrelated (e.g., age & smokes)
  - likely wrong, as age and smokes could be correlated even within class
- Surprisingly, despite the fact that this seems unrealistic, in practice this can work okay
  - one hypothesis is we are running these algorithms on “easy” data
  - another is that dependencies skew the distribution equally across all classes, so no one class gets an increased probability



Naive Bayes

- For naive Bayes, we learn p(x | y) and p(y) exactly given this conditional independence assumption
  - For the classification setting

$$p(\mathbf{x}|y) = \prod_{j=1}^{d} p(x_j|y)$$

- Then we still need to choose p(x_j | y) and p(y)
- What is p(y)? Table of values

$$p(y = 1), \ldots, p(y = k - 1), \text{ with } p(y = k) = 1 - \sum_{i=1}^{k-1} p(y = i)$$

	probability
1	p1
2	p2
.	.
.	.
.	.
k	pk

What if we have continuous features?

- For continuous features x, binary y in {0,1}, what could we choose for p(x l y)?
  - e.g., 5 features x = [x_1, …, x_5]

- Need to identify p(x_j l y) — what could this be?

- What are the parameters and how do we learn them?

- Can choose p(x_j l y) to be Gaussian, and learn mean and a scalar variance (since x_j is a scalar)

- Can actually examine p(x_j l y) for each class, and make different distributional assumptions

Reminders: Oct. 29, 2019

- Additional Information about Midterm posted
  - Covers Chapters 1-6
  - Midterm will be in room CCIS 1 440
  - I will give some insight into how I mark on Nov 5, when we do the midterm review
- Thought Questions due this Thursday

Thought Question

- “My question relates to the discussion of the bias variance trade-off in Chapter 5. I wonder if the bias variance trade-off really applies to all classes of machine learning, or if it is more algorithm dependent. For example, it is clear that for linear regression the bias variance trade-off regime holds true, as when we increase the complexity of the model, through polynomial regression for instance, we will start to overfit the data, i.e. begin to trade-off bias for variance. It is not so clear that this trade-off is as true for techniques such as boosting, or deep neural networks. We often see examples of deep neural networks that appear to be massively overparameterized, but instead of overfitting the data, they actually generalize better than many simpler models. Due to this, I wonder if it is perhaps misleading to think of the bias variance trade-off applying equally to all classes of algorithms in machine learning.”

What does it mean to increase the representational capacity?

- Have a set of functions that we can learn (choose from)

- E.g., in linear regression, hypothesis space (set of possible functions we could find) is from

$$\mathcal{F}_1 = \{f : \mathbb{R}^d \to \mathbb{R} \mid f(\mathbf{x}) = \mathbf{x}^\top \mathbf{w} \text{ for some } \mathbf{w} \in \mathbb{R}^d\}$$

- Using linear regression directly on inputs means we cannot represent all functions from a space like

$$\mathcal{F}_2 = \{f : \mathbb{R}^d \to \mathbb{R} \mid f(\mathbf{x}) = \mathbf{x}^\top \mathbf{w} + x_1^2 \beta \text{ for some } \mathbf{w} \in \mathbb{R}^d, \beta \in \mathbb{R}\}$$

- Transforming x to phi(x) expands the set of functions we can represent, and so hopefully learn

$$\text{Question: Is } \mathcal{F}_1 \subset \mathcal{F}_2$$

More nuanced view (see https://arxiv.org/abs/1812.11118)

Hypothesis class $\mathcal{H}$ corresponds to Function class $\mathcal{F}$

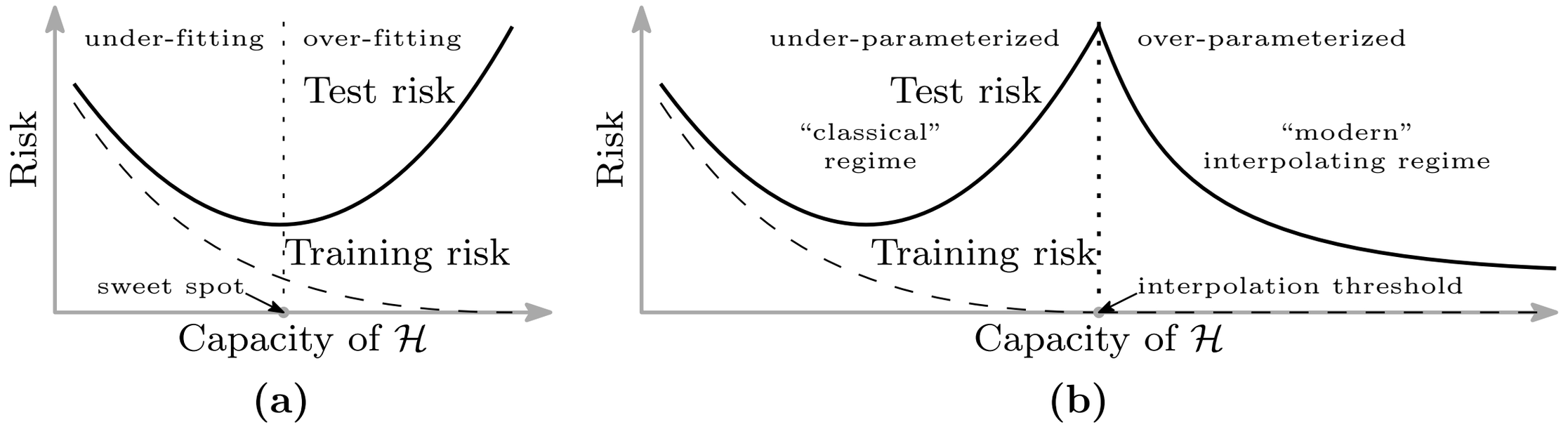
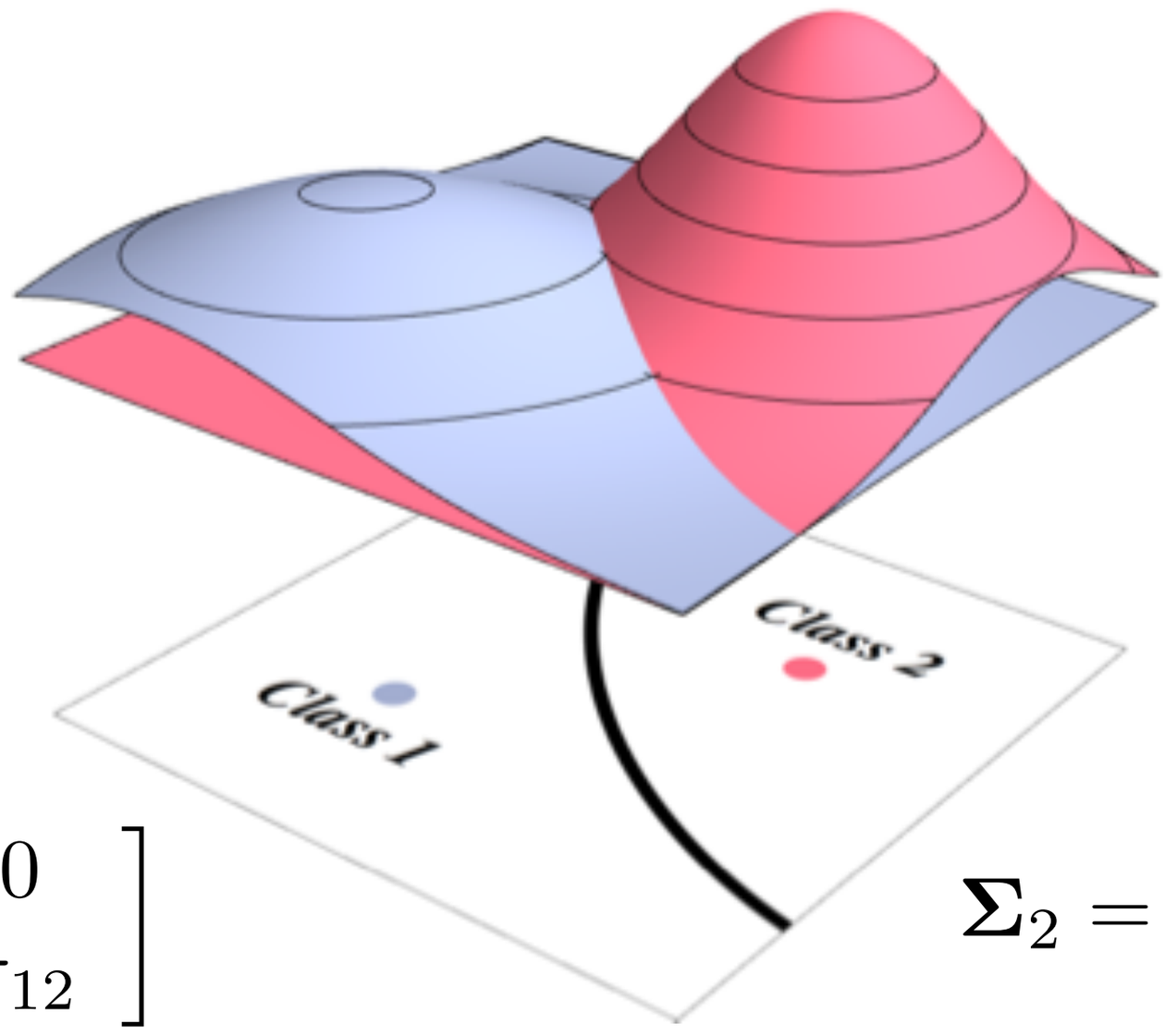


Figure 1: **Curves for training risk (dashed line) and test risk (solid line).** (**a**) The classical *U-shaped risk curve* arising from the bias-variance trade-off. (**b**) The *double descent risk curve*, which incorporates the U-shaped risk curve (i.e., the "classical" regime) together with the observed behavior from using high capacity function classes (i.e., the "modern" interpolating regime), separated by the interpolation threshold. The predictors to the right of the interpolation threshold have zero training risk.

Reasoning: x-axis should instead be the ability to find a smooth (simple) function

Continuous naive Bayes

$\boldsymbol{\mu}_1, \boldsymbol{\Sigma}_1$



$\boldsymbol{\mu}_2, \boldsymbol{\Sigma}_2$

$$\boldsymbol{\Sigma}_1 = \begin{bmatrix} \sigma_{11} & 0 \\ 0 & \sigma_{12} \end{bmatrix}$$

$$\boldsymbol{\Sigma}_2 = \begin{bmatrix} \sigma_{21} & 0 \\ 0 & \sigma_{22} \end{bmatrix}$$

* Image from Yaroslav Bulatov

Exercise: Prediction with Naive Bayes Model

- Imagine someone gives you p(x l y) and p(y)
- They give you a test instance, with features x
  - e.g., x is an image, of pixel values (values between 0 to 255)
- Your goal is to predict the output y
  - e.g., if the image contains a cat or not
- How do you use p(x l y) and p(y) to make a prediction?

Whiteboard

- Naive Bayes derivation
- Exercise showing how to use naive Bayes
- Exercise dealing with missing values

Exercise: Bias-variance for naive Bayes model

- Do you think naive Bayes has high bias and/or high variance?
  - is naive Bayes unbiased?
  - Recall: bias refers to error to the true function, in expectation
    - is f(y) = p(x l y) an unbiased estimate of f^*(y)?
- How do we compute bias and variance?
- naive Bayes can perform better in the small sample setting
  - see “On discriminative vs. generative classifiers: A comparison of logistic regression and naive Bayes”, Ng and Jordan, 2002

Pros and Cons

- Discriminative models: learn p(y l x)
  - focus learning on the value we care about: p(y l x)
  - can be much simpler, particularly if features x complex: p(x l y) can be difficult to model without strong assumptions
- Generative models: learn p(xly) and p(y)
  - can be easier for expert to encode prior beliefs
  - can sample from the generative model, obtain explanations
  - can be more amenable to missing values

Example: trees types

- Imagine classifying trees into coniferous or deciduous
  - y in {C, D}, i.e., {0,1}
  - x is a vectorized image (e.g., 64 x 64 image of pixel values becomes a 64*64*3 approx= 13k, where *3 is from 3 RGB values
  - or x is a vector of features extracted from the images, e.g., height, location, leave size, etc.
- Expert info: can encode different information about possible ranges for each feature, given its Conif. or Decid.,
  - e.g. p(leaf size | Y = C) is likely much different than p(leaf size | Y = D)
- Can specify such info for a given class, by using p(x | y), BUT p(y | x) does not allow this information to be encoded

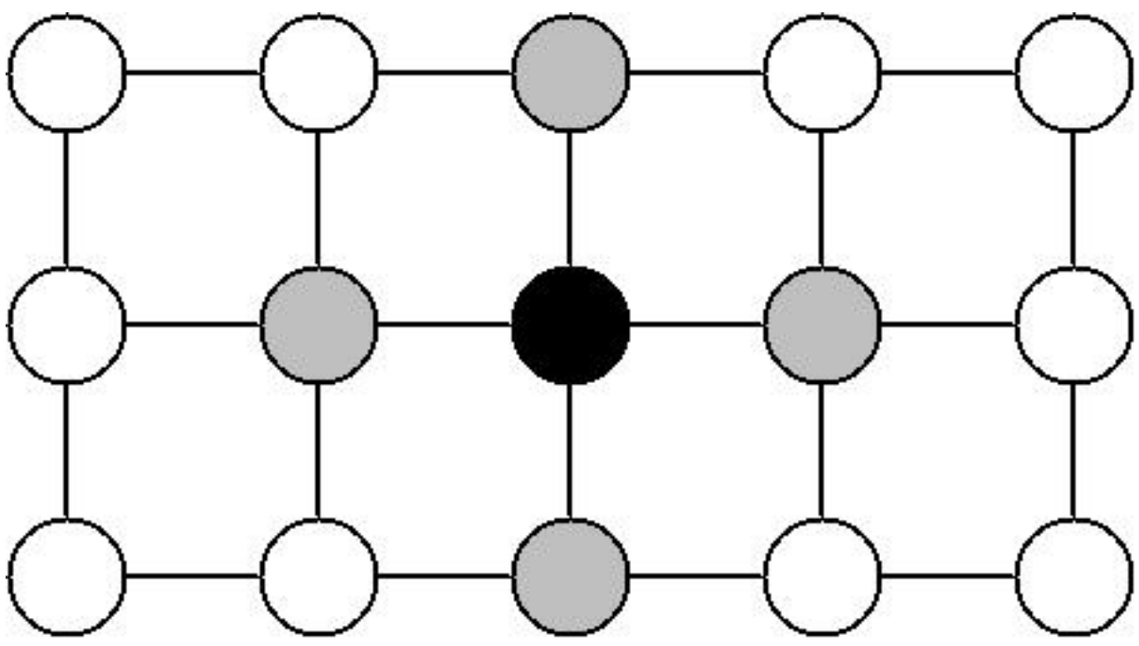
Sampling from generative model

- How sample (x,y) from generative model p(x,y) = p(x | y) p(y)
  - Sample y from p(y)
  - Then sample x from p(x | y)

- Why would you do this? Let's use the trees example again
  - Could sample trees from p(x | y) to see what your model produces
  - Could answer additional questions about the features, such as average leaf diameter in population
  - Could depict average tree, within a type or across types
  - Question: how would you do it within a type (deciduous or coniferous)?
  - Question: how would you do it across types?
  - More realistic example for explanations: obtain profile of typical person with heart disease

Other generative models

- Often used generative models for images (Markov random field)
  - e.g., can model the spatial structure in the distribution p(x l y)
  - We know a lot about image structure, can take advantage of that expert knowledge in choice of p(x l y)



Generative models becoming more popular again

- Using generative models to actually generate data, rather than necessarily for classification or regression
- Some work also to using generative models with neural networks, in a semi-supervised setting
  - e.g., see “Semi-supervised Learning with Deep Generative Models”, Kingma et al., 2014

Thought question

- Is it possible to provide a prediction and an estimate of how confident we are in that prediction?
  - “Let's say the data you are modeling is inherently high-variance and it may not be appropriate to give exact estimates for new data. Is it possible to generate a distribution range, i.e., the outcome will likely fall within some N(mu, sigma^2)”
- For p(y | x) Gaussian, we could try to estimate mean mu = <x,w> and estimate variance sigma^2
  - when predict E[Y | x], have a sense of how much y can vary
- But what if our estimates are wrong? To be confident in predictions, want estimate of confidence in parameters

Practical considerations: incorporating new data

- Imagine you have a discriminative model (say weights w) and now want to incorporate new data

- How can you do this with regression approaches?

- Option 1: add data to your batch and recompute entire solution

- Option 2: start from previous w (warm start), add data to your batch and do a few gradient descent steps

- Option 3: use stochastic gradient descent update and step in the direction of the gradient for those samples
  - for a constant stream of data, might only ever use one sample and then throw it away

How about naive Bayes?

- How can we update means and covariances in naive Bayes model?

- Learned mu_{j,c} and sigma_{j,c} for each feature j, each class c

Updating naive Bayes model

- Keep a running average of mu_{j,c} and sigma_{j,c}, with number of samples n_{j,c} for feature j class c

- For a new sample (x, y), can update parameters mu_{j,y} and sigma_{j,y} using:

$$\mu_{j,y} = \mu_{j,y} + (x_j - \mu_{j,y})/n_{j,y}$$
$$\sigma^2_{j,y} = \sigma^2_{j,y} + ((x_j - \mu_{j,y})^2 - \sigma^2_{j,y})/n_{j,y}$$

- There are other more numerically stable running average and running variance algorithms (e.g., Welford)

Exercise: Non-stationarity

- A theme in thought questions: “What if the distribution changes over time?”

- Say you have trained your model so far, and now are getting new data that is from a slightly different distribution (drift)
  - Data could be coming from a physical system that wears out, such as a robot vacuum collecting data where its wheels wear-out
  - Data could reflect continually (but slowly) changes preferences in human population

- How can we handle this?

Exercise: Non-stationarity

- A theme in thought questions: “What if the distribution changes over time?”
- How can we handle this?
  - Don’t want to throw away previous solution, since drift is slow so old solution is still (mostly) reflective
  - We can update parameters with this new data, and hope that works
  - How can we give more weight to new data? Say for naive Bayes

Exponential average

$$\begin{aligned}
\mu_t &= \alpha x_t + (1-\alpha)\mu_{t-1} \\
&= \alpha x_t + (1-\alpha)(\alpha x_{t-1} + (1-\alpha)\mu_{t-2}) \\
&= \alpha x_t + \alpha(1-\alpha)x_{t-1} + (1-\alpha)^2(\alpha x_{t-2} + (1-\alpha)\mu_{t-3}) \\
&= \alpha \sum_{i=0}^{\infty}(1-\alpha)^i x_{t-i}
\end{aligned}$$

where $0 < \alpha < 1$

$$\sum_{i=0}^{\infty}(1-\alpha)^i = \frac{1}{\alpha}$$