Classification using probability models

A classification problem

You have a bottle of wine whose label is missing.

Which winery is it from, 1, 2, or 3?

Solve this problem using visual and chemical features of the wine.

The data set

Each data pt is the description of a bottle:

$x = (\_, \_, \_, \_, \ldots, \_) \in \mathbb{R}^{13}$

alcohol, malic acid, ash

Label y = 1 or 2 or 3

Three classes or labels

Training set: 130 pairs (x, y)

Use this training set to learn a classifier that predicts labels for new x.

Training set obtained from 130 bottles

- Winery 1: 43 bottles
- Winery 2: 51 bottles
- Winery 3: 36 bottles
- For each bottle, 13 features:
  'Alcohol', 'Malic acid', 'Ash', 'Alcalinity of ash', 'Magnesium',
  'Total phenols', 'Flavanoids', 'Nonflavanoid phenols', 'Proanthocyanins',
  'Color intensity', 'Hue', 'OD280/OD315 of diluted wines', 'Proline'

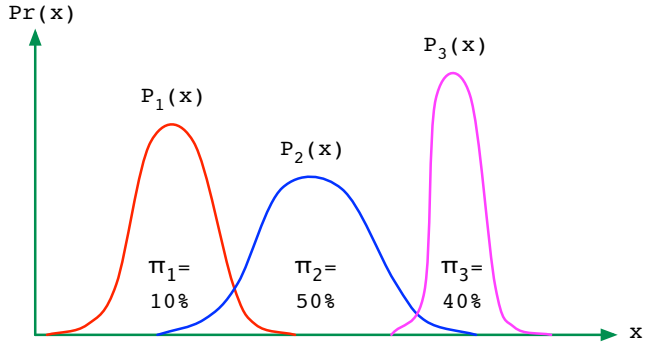
Also, a separate **test set** of 48 labeled points.

↑ we'll use this to evaluate our classifier

The generative approach to classification

There are many types of classifiers. We'll look at the generative approach.

Example:
Data space $\mathcal{X} = \mathbb{R}^d$ $d = 13$
Classes/labels $\mathcal{Y} = \{1, 2, 3\}$



The **learning process**:

- Use training data to fit a probability distribution to each class individually.
- For each class j, we have:
  - the probability of that class, $\pi_j = \Pr(y = j)$ — What fraction of the data (bottles) are label j? JUST A NUMBER.
  - the distribution of data in that class, $P_j(x)$ — What is the distribution of x-vectors in class j? AN ENTIRE DISTRIBUTION.

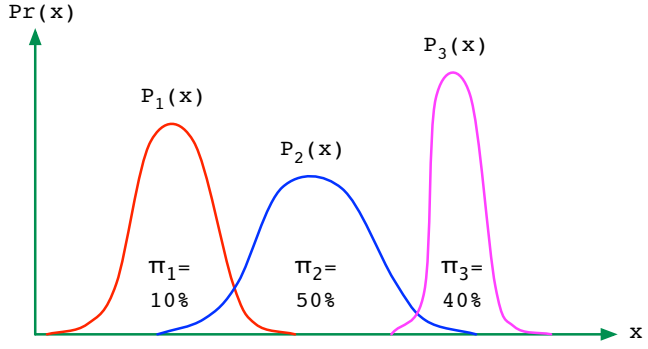
Overall **joint distribution**: $\Pr(x, y) = \Pr(y)\Pr(x|y) = \pi_y P_y(x)$.

Classifying a new point

Data space $\mathcal{X} = \mathbb{R}^d$
Classes/labels $\mathcal{Y} = \{1, 2, 3\}$

For each class j, we have

- $\pi_j = \Pr(y = j)$
- $P_j(x) = \Pr(x|y = j)$



Now we want to label a new point x. For any candidate label j,

Bayes' rule

$$\Pr(y = j|x) = \frac{\Pr(y = j)\Pr(x|y = j)}{\Pr(x)} = \frac{\pi_j P_j(x)}{\Pr(x)}$$

Optimal prediction: the class j with largest $\pi_j P_j(x)$. ← very simple rule

Fitting a generative model

Training set of 130 bottles:

- Winery 1: 43 bottles, winery 2: 51 bottles, winery 3: 36 bottles
- For each bottle, 13 features: 'Alcohol', 'Malic acid', 'Ash', 'Alcalinity of ash','Magnesium', 'Total phenols', 'Flavanoids', 'Nonflavanoid phenols', 'Proanthocyanins', 'Color intensity', 'Hue', 'OD280/OD315 of diluted wines', 'Proline'

Fitting a generative model

Proportion of each class: π_1, π_2, π_3

Training set of 130 bottles:

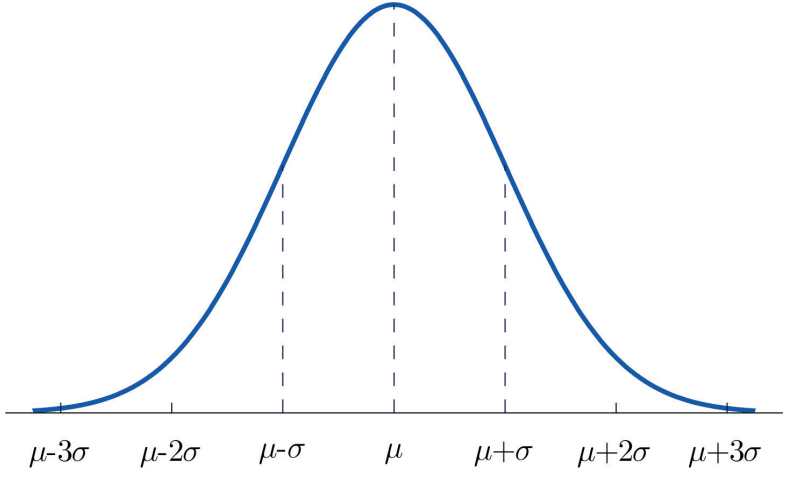
- Winery 1: 43 bottles, winery 2: 51 bottles, winery 3: 36 bottles
- For each bottle, 13 features: 'Alcohol', 'Malic acid', 'Ash', 'Alcalinity of ash','Magnesium', 'Total phenols', 'Flavanoids', 'Nonflavanoid phenols', 'Proanthocyanins', 'Color intensity', 'Hue', 'OD280/OD315 of diluted wines', 'Proline'

Class weights:
$\pi_1 = 43/130 = 0.33, \quad \pi_2 = 51/130 = 0.39, \quad \pi_3 = 36/130 = 0.28$

Need distributions P_1, P_2, P_3, one per class.
Base these on a single feature: 'Alcohol'.

Let's start with a simple model that uses only one of the 13 features.

Recall: Univariate Gaussian

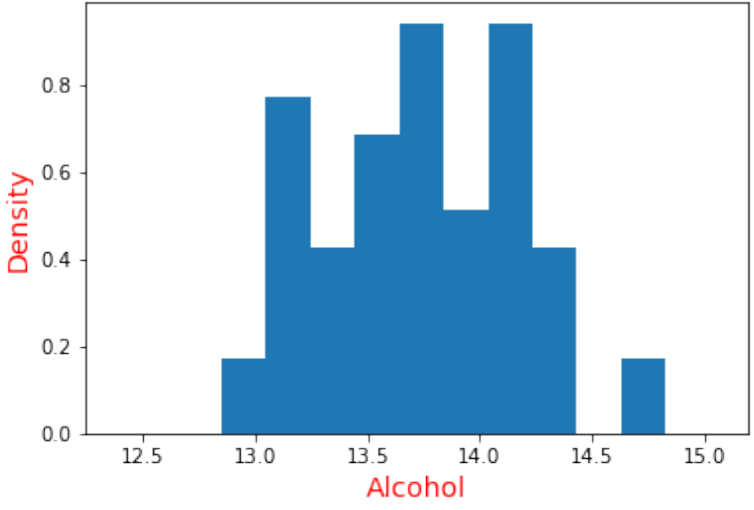


The Gaussian $N(\mu, \sigma^2)$ has mean μ, variance σ^2, and density function

$$p(x) = \frac{1}{(2\pi\sigma^2)^{1/2}} \exp\left(-\frac{(x-\mu)^2}{2\sigma^2}\right).$$

The distribution for winery 1

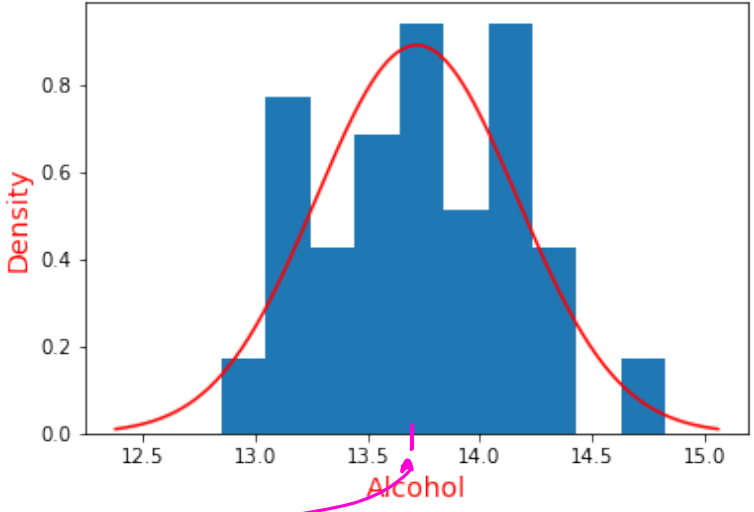
Single feature: 'Alcohol'



The distribution for winery 1

Single feature: 'Alcohol'

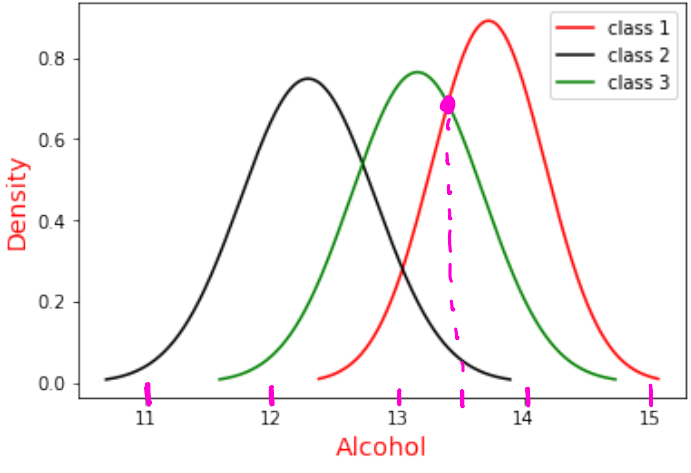
The Gaussian smooths out the histogram nicely.



Mean $\mu = 13.72$, Standard deviation $\sigma = 0.44$ (variance 0.20) ← two lines of Python

All three wineries

To classify x: compute
$\pi_1 P_1(x)$
$\pi_2 P_2(x)$
$\pi_3 P_3(x)$
} pick the largest

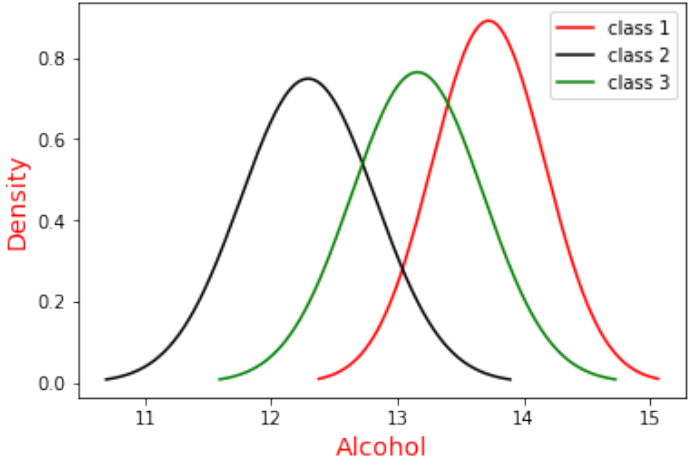


- $\pi_1 = 0.33,\ P_1 = N(13.7, 0.20)$
- $\pi_2 = 0.39,\ P_2 = N(12.3, 0.28)$
- $\pi_3 = 0.28,\ P_3 = N(13.2, 0.27)$

To classify x: Pick the j with highest $\pi_j P_j(x)$

All three wineries

Built model based on training set of 130 bottles.



- $\pi_1 = 0.33,\ P_1 = N(13.7, 0.20)$
- $\pi_2 = 0.39,\ P_2 = N(12.3, 0.28)$
- $\pi_3 = 0.28,\ P_3 = N(13.2, 0.27)$

To classify x: Pick the j with highest $\pi_j P_j(x)$

evaluate model using test set of 48 bottles.

Test error: 14/48 = 29%

The data set, again

Training set obtained from 130 bottles

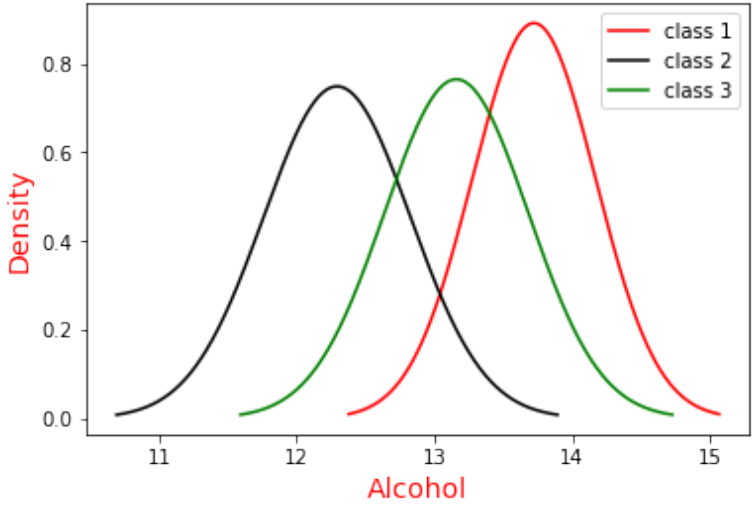
- Winery 1: 43 bottles
- Winery 2: 51 bottles
- Winery 3: 36 bottles
- For each bottle, 13 features:
  'Alcohol', 'Malic acid', 'Ash', 'Alcalinity of ash','Magnesium',
  'Total phenols', 'Flavanoids', 'Nonflavanoid phenols', 'Proanthocyanins',
  'Color intensity', 'Hue', 'OD280/OD315 of diluted wines', 'Proline'

Also, a separate test set of 48 labeled points.

This time: 'Alcohol' and 'Flavanoids'.

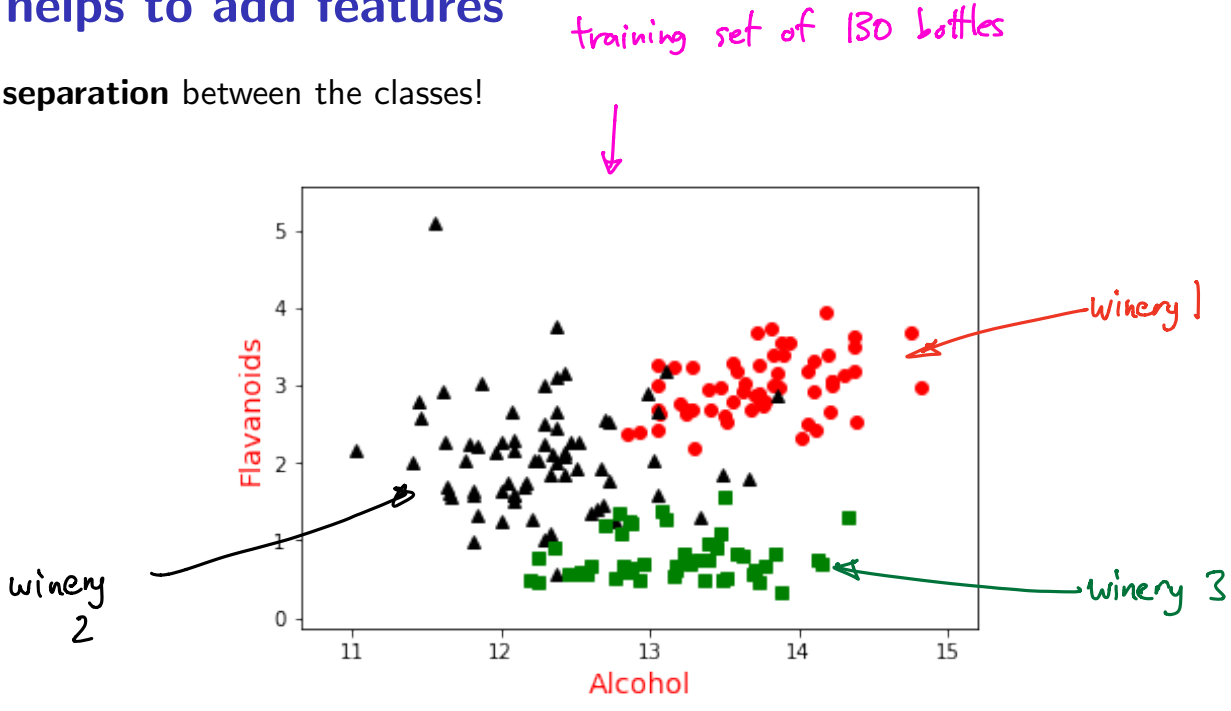
Why it helps to add features

Better **separation** between the classes!



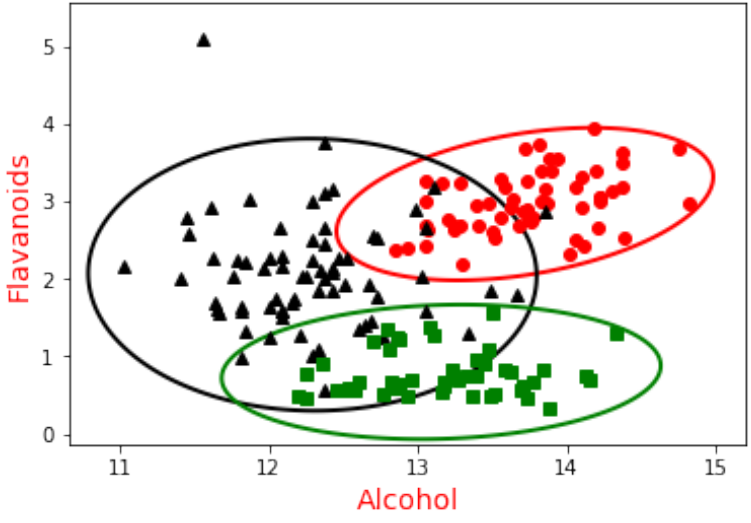
Why it helps to add features

Better **separation** between the classes!



Why it helps to add features

Better **separation** between the classes!



Why it helps to add features

Random guessing 67%
One feature 29%
Two features 8%

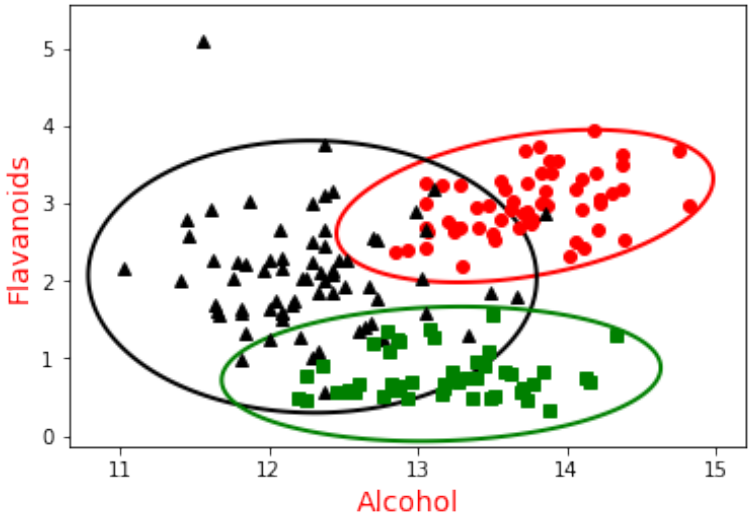
Better **separation** between the classes!

same as before

π_1 $P_1(x)$
π_2 $P_2(x)$
π_3 $P_3(x)$

pick largest

Now, these are bivariate Gaussians

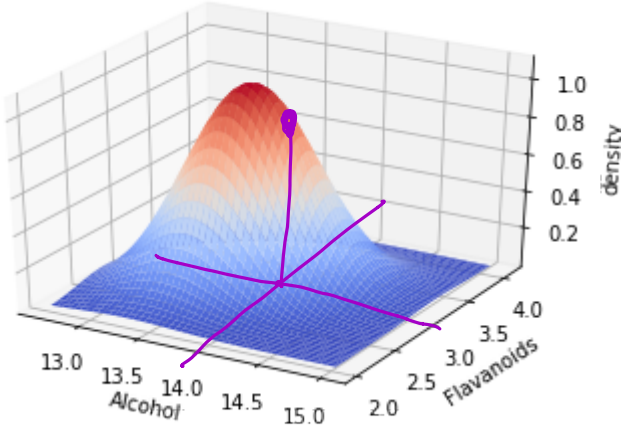


$\frac{8}{100} \times 48$

Error rate drops from 29% to 8%.

Test set has 48 points.
How many did we get wrong? 4.

Bivariate Gaussian



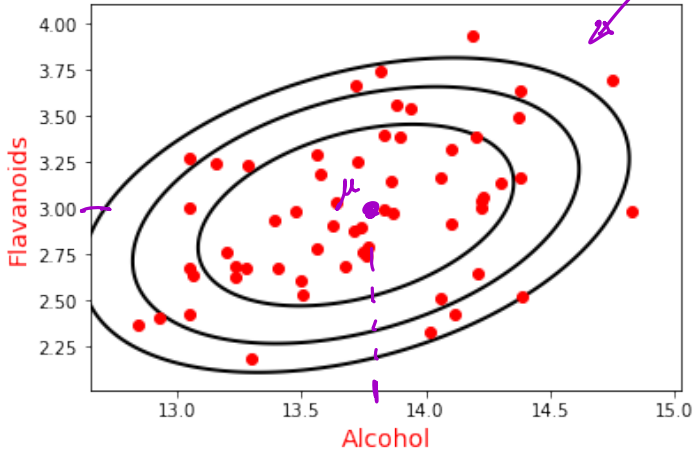
Bivariate Gaussian

Density for winery #1

$P_1(x)$ density for winery 1

P_1 : $N(\mu_1, \Sigma_1)$

$N(\mu^{(1)}, \Sigma^{(1)})$



What is the correlation btwn these two features?

$$\frac{0.06}{\sqrt{0.2 \times 0.12}} = 0.39$$

weak correlation

Model class 1 by a bivariate Gaussian, parametrized by:

mean $\mu = \begin{pmatrix} 13.7 \\ 3.0 \end{pmatrix}$ and covariance matrix $\Sigma = \begin{pmatrix} 0.20 & 0.06 \\ 0.06 & 0.12 \end{pmatrix}$

mean alcohol

mean flavanoid

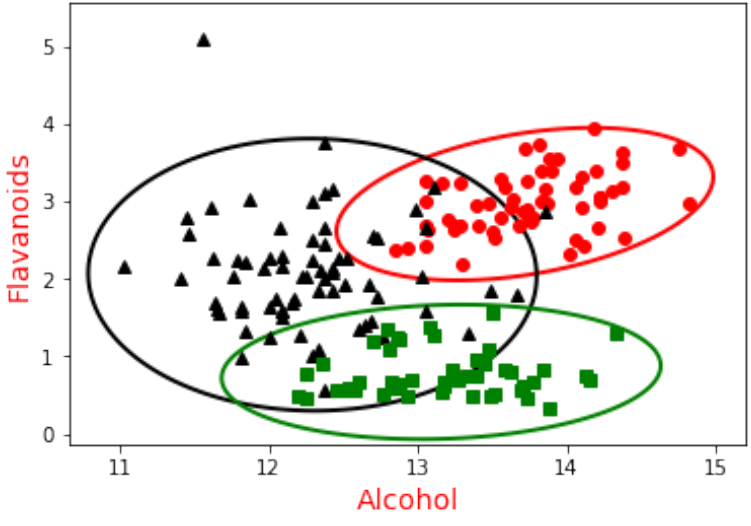
alcohol variance

covariance btwn alcohol, flavanoid

flavanoid variance

The decision boundary

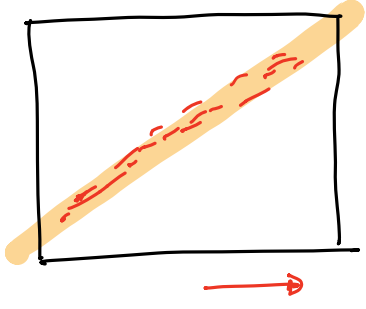
Go from 1 to 2 features: error rate goes from 29% to 8%.

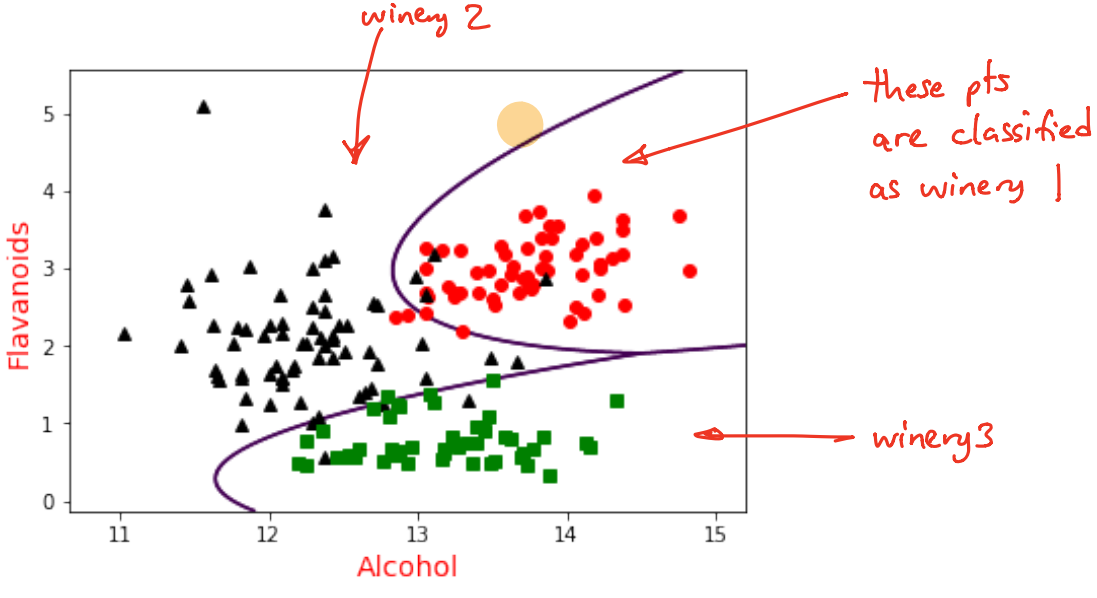


The decision boundary

$$Pr(y=j|x) = \frac{\pi_j P_j(x)}{\pi_1 P_1(x) + \pi_2 P_2(x) + \pi_3 P_3(x)}$$

Go from 1 to 2 features: error rate goes from 29% to 8%.

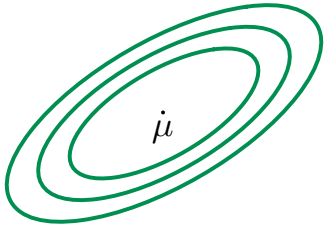




Quadratic

What kind of function is this? And, can we use more features?

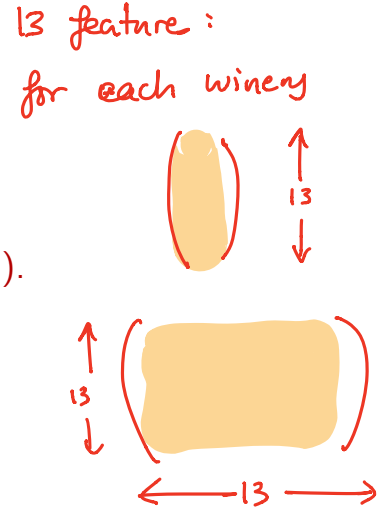
Recall: Multivariate Gaussian



$N(\mu, \Sigma)$: Gaussian in $\mathbb{R}^d$

- mean: $\mu \in \mathbb{R}^d$
- covariance: $d \times d$ matrix Σ

Generates points $X = (X_1, X_2, \ldots, X_d)$.



- μ is the vector of coordinate-wise means:

$$\mu_1 = \mathbb{E}X_1, \ \mu_2 = \mathbb{E}X_2, \ldots, \ \mu_d = \mathbb{E}X_d.$$

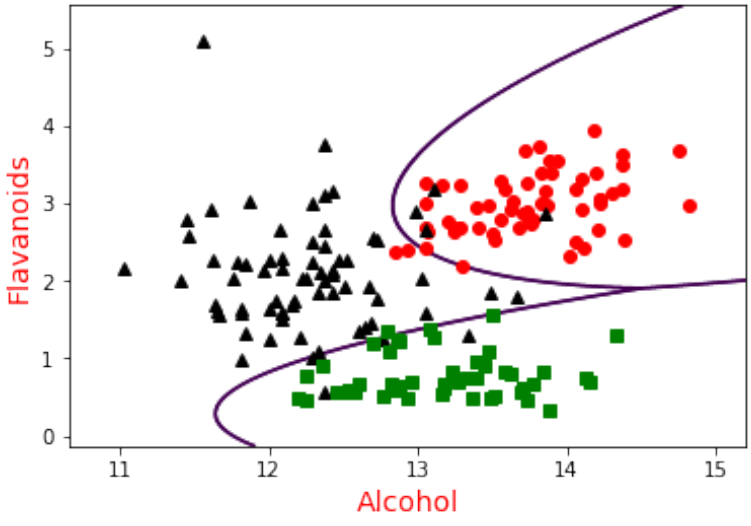
- Σ is a matrix containing all pairwise covariances:

$$\Sigma_{ij} = \Sigma_{ji} = \text{cov}(X_i, X_j) \quad \text{if } i \neq j$$
$$\Sigma_{ii} = \text{var}(X_i)$$

Density $$p(x) = \frac{1}{(2\pi)^{d/2}|\Sigma|^{1/2}} \exp\left(-\frac{1}{2}(x-\mu)^T\Sigma^{-1}(x-\mu)\right)$$

Back to the winery data

Go from 1 to 2 features: test error goes from 29% to 8%.



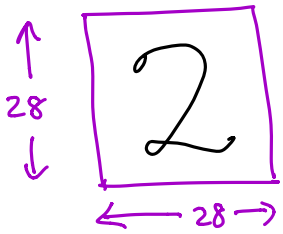
With all 13 features: test error rate goes to zero.

Is this a perfect classifier?

Classification of handwritten digits

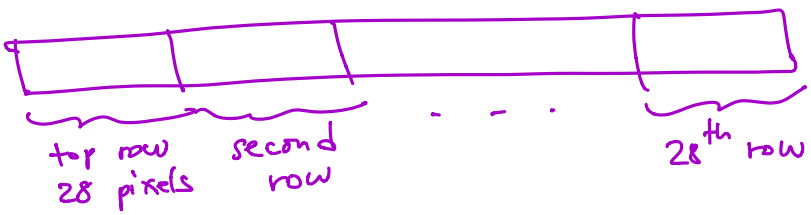
MNIST → 60,000 training images of digits 0-9
MNIST → 10,000 test images

Each image



greyscale
each pixel is 0-255
(0 = white, 255 = black)

Vector of length 784:



Data pts: $x \in \mathbb{R}^{784}$
$y \in \{0, 1, \ldots, 9\}$

---

10 digits

Training set size 60,000

Roughly 6,000 of each

- For each class:
  Covariance matrix 784 × 784 ← lots of parameters
  Need to smooth these matrices.
  Say Σ_j = empirical covariance for class j
  (numpy.cov)
  Instead, use $\Sigma_j + cI$ to "smooth" it.
  ↑ how to choose this?
  should only use the training set

Instead of
$\underset{j}{\text{argmax}}\ \pi_j P_j(x)$
Can use
$\underset{j}{\text{argmax}}\ \log \pi_j + \log P_j(x)$

- In high dimensional spaces, probabilities are tiny.
  To avoid problems, often useful to work with log-probabilities.

Binary classification with Gaussian generative model

- Estimate class probabilities π_1, π_2
- Fit a Gaussian to each class: $P_1 = N(\mu_1, \Sigma_1),\ P_2 = N(\mu_2, \Sigma_2)$

Given a new point x, predict class 1 if

$$\pi_1 P_1(x) > \pi_2 P_2(x)$$

Binary classification with Gaussian generative model

- Estimate class probabilities π_1, π_2
- Fit a Gaussian to each class: $P_1 = N(\mu_1, \Sigma_1),\ P_2 = N(\mu_2, \Sigma_2)$

Given a new point x, predict class 1 if

$$\pi_1 P_1(x) > \pi_2 P_2(x)$$

Binary classification with Gaussian generative model

$$\pi_1 \frac{1}{(2\pi)^{d/2}|\Sigma_1|^{1/2}} \exp\left(-\frac{1}{2}(x-\mu_1)^T \Sigma_1^{-1}(x-\mu_1)\right) > \pi_2 \frac{1}{(2\pi)^{d/2}|\Sigma_2|^{1/2}} \exp(\cdots)$$

- Estimate class probabilities π_1, π_2
- Fit a Gaussian to each class: $P_1 = N(\mu_1, \Sigma_1),\ P_2 = N(\mu_2, \Sigma_2)$

Given a new point x, predict class 1 if

$$\pi_1 P_1(x) > \pi_2 P_2(x) \Leftrightarrow \underbrace{x^T M x}_{\text{quadratic term}} + \underbrace{2w^T x}_{\text{linear term}} \geq \theta,$$

where:

$$M = \frac{1}{2}(\Sigma_2^{-1} - \Sigma_1^{-1})$$
$$w = \Sigma_1^{-1}\mu_1 - \Sigma_2^{-1}\mu_2$$

Q: When is $M = 0$?

A: When $\Sigma_1 = \Sigma_2$.

and θ is a threshold depending on the various parameters.

Linear or **quadratic** decision boundary.

Common covariance: $\Sigma_1 = \Sigma_2 = \Sigma$

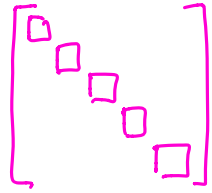
Linear decision boundary: choose class 1 if

$$x \cdot \underbrace{\Sigma^{-1}(\mu_1 - \mu_2)}_{w} \geq \theta.$$

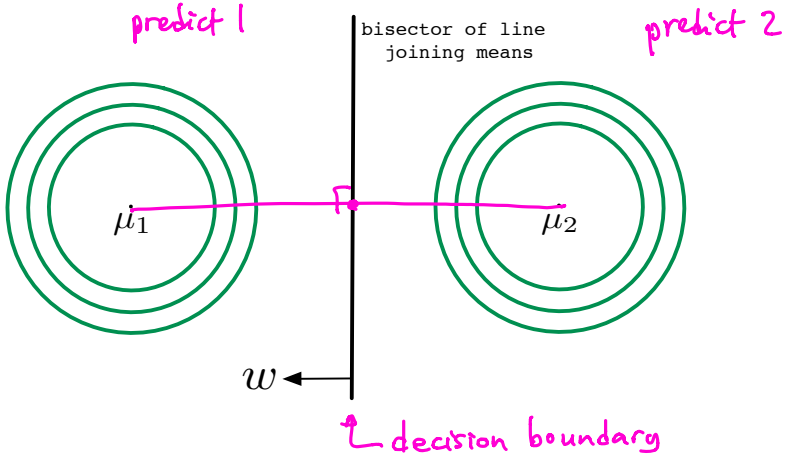
Common covariance: $\Sigma_1 = \Sigma_2 = \Sigma$

Linear decision boundary: choose class 1 if

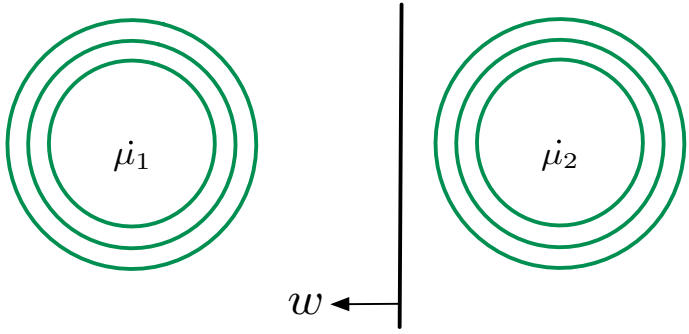
$$x \cdot \underbrace{\Sigma^{-1}(\mu_1 - \mu_2)}_{w} \geq \theta.$$



Example 1: Spherical Gaussians with $\Sigma = I_d$ and $\pi_1 = \pi_2 = 1/2$.

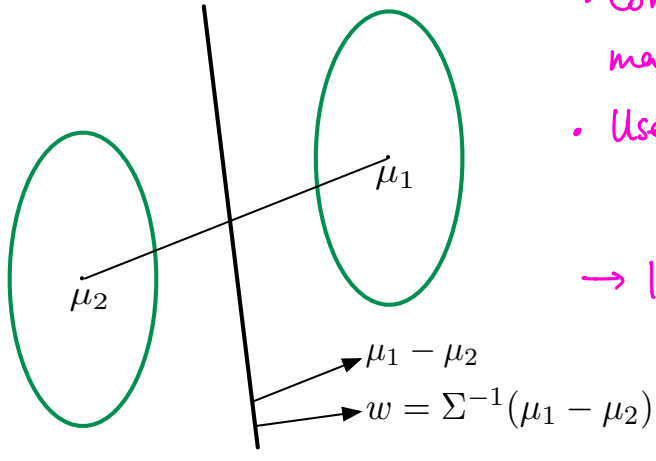


Example 2: Again spherical, but now $\pi_1 > \pi_2$.



Example 3: Non-spherical.

$\Sigma_1 = \Sigma_2$



Common case.

- Compute empirical covariance matrices $\hat{\Sigma}_1$ and $\hat{\Sigma}_2$
- Use a common covariance

$$\Sigma_1 = \Sigma_2 = \frac{1}{2}\left(\hat{\Sigma}_1 + \hat{\Sigma}_2\right)$$

→ linear decision boundary

Classification rule: $w \cdot x \geq \theta$

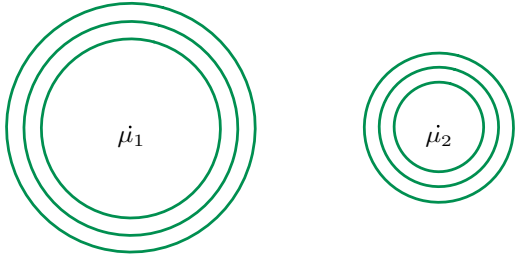
- Choose w as above
- Common practice: fit θ to minimize training or validation error

Different covariances: $\Sigma_1 \neq \Sigma_2$

Quadratic boundary: choose class 1 if $x^T Mx + 2w^T x \geq \theta$, where:

$$M = \frac{1}{2}(\Sigma_2^{-1} - \Sigma_1^{-1})$$
$$w = \Sigma_1^{-1}\mu_1 - \Sigma_2^{-1}\mu_2$$

Example 1: $\Sigma_1 = \sigma_1^2 I_d$ and $\Sigma_2 = \sigma_2^2 I_d$ with $\sigma_1 > \sigma_2$

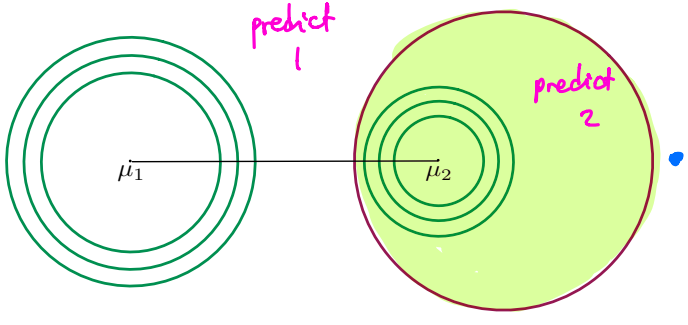


Different covariances: $\Sigma_1 \neq \Sigma_2$

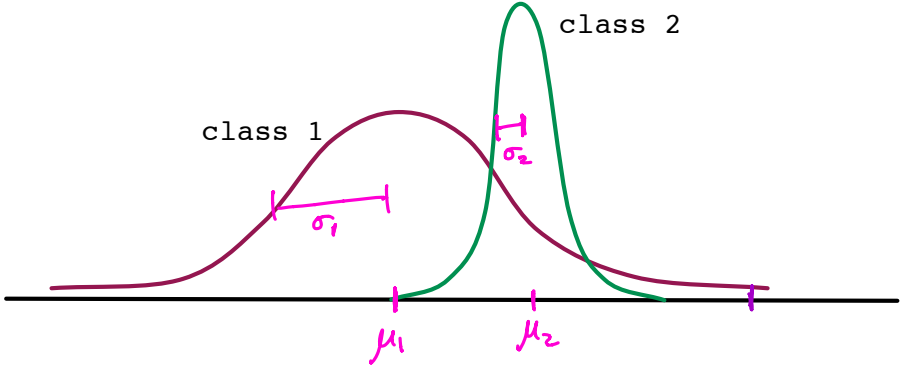
Quadratic boundary: choose class 1 if $x^T M x + 2w^T x \geq \theta$, where:

$$M = \frac{1}{2}(\Sigma_2^{-1} - \Sigma_1^{-1})$$
$$w = \Sigma_1^{-1}\mu_1 - \Sigma_2^{-1}\mu_2$$

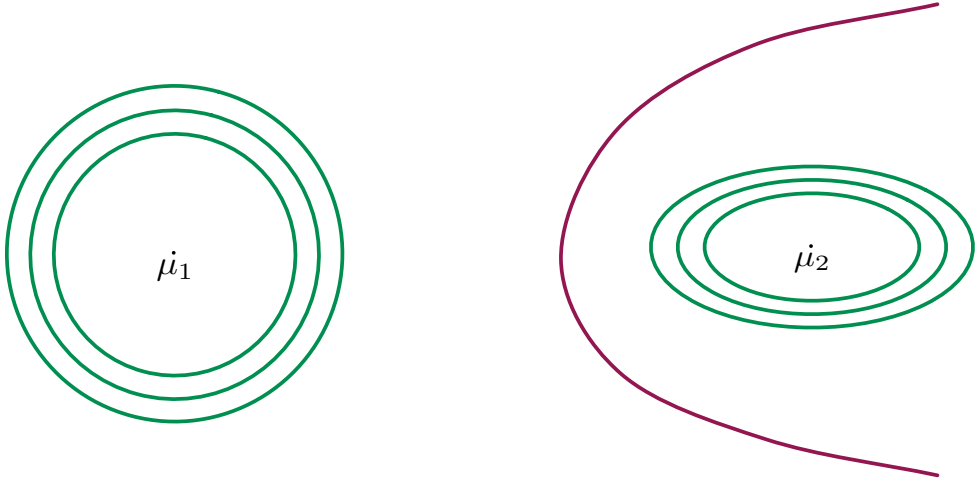
Example 1: $\Sigma_1 = \sigma_1^2 I_d$ and $\Sigma_2 = \sigma_2^2 I_d$ with $\sigma_1 > \sigma_2$



Example 2: Same thing in 1-d. $\mathcal{X} = \mathbb{R}$.



Example 3: A parabolic boundary.



Multiclass discriminant analysis

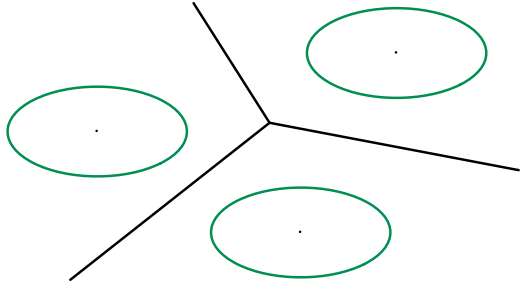
k classes: weights π_j, class-conditional densities $P_j = N(\mu_j, \Sigma_j)$.

Each class has an associated **quadratic** function

$$f_j(x) = \log(\pi_j P_j(x))$$

To classify point x, pick $\arg\max_j f_j(x)$.

If $\Sigma_1 = \cdots = \Sigma_k$, the boundaries are **linear**.



To predict at x, return

$\operatorname{argmax}_j \pi_j P_j(x)$

$\equiv \operatorname{argmax}_j \log(\pi_j P_j(x))$

$f_j(x)$

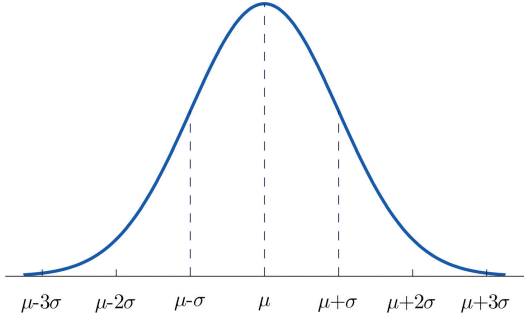
quadratic function

Beyond Gaussians

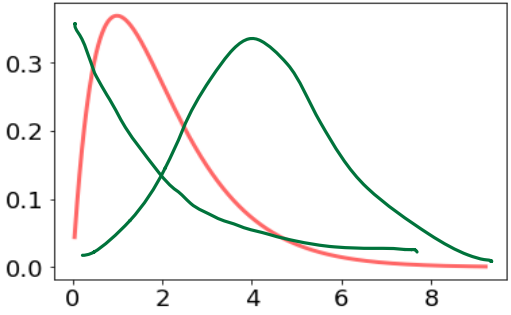
The generative methodology:

- Fit a **distribution** to each class separately
- Use Bayes' rule to classify new data

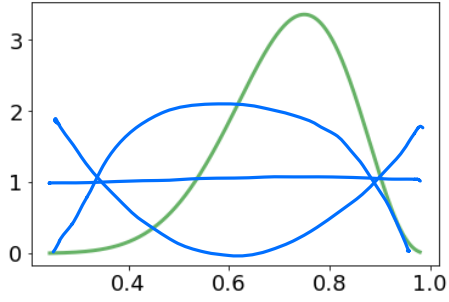
What distribution to use? Are Gaussians enough?



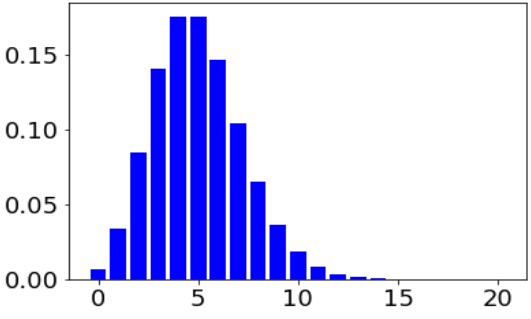
Exponential families of distributions



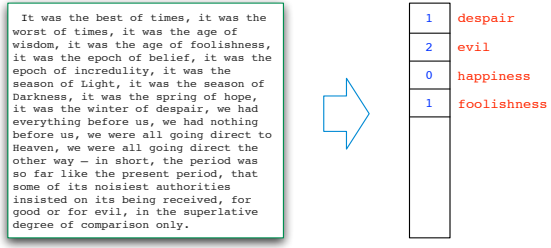
GAMMA



BETA



POISSON



CATEGORICAL

Multivariate distributions

We've described a variety of distributions for **one-dimensional** data.
What about higher dimensions?

1. **Naive Bayes**: Treat coordinates as independent.
   For $x = (x_1, \ldots, x_d)$, fit separate models $\Pr_i$ to each x_i, and assume

   $$\Pr(x_1, \ldots, x_d) = \Pr_1(x_1)\Pr_2(x_2)\cdots\Pr_d(x_d).$$

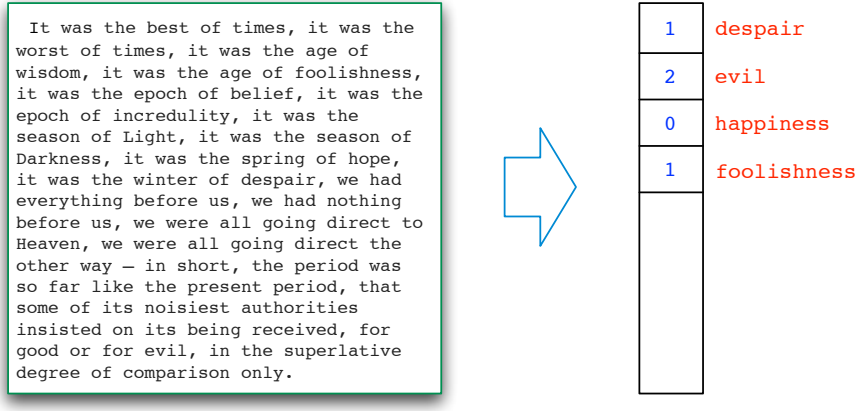
   This assumption is typically inaccurate.

2. **Multivariate Gaussian**.
   Model correlations between features: we've seen this in detail.

3. **Graphical models**.
   Arbitrary dependencies between coordinates.

Handling text data

Bag-of-words: vectorial representation of text documents.



- Fix V = some vocabulary.
- Treat each document as a vector of length $|V|$:

$$x = (x_1, x_2, \ldots, x_{|V|}),$$

where $x_i = \#$ of times the ith word appears in the document.

A standard distribution over such document-vectors x: the **multinomial**.

Multinomial naive Bayes

Multinomial distribution over a vocabulary V:

$$p = (p_1, \ldots, p_{|V|}), \quad \text{such that} \quad p_i \geq 0 \text{ and } \sum_i p_i = 1$$

Document $x = (x_1, \ldots, x_{|V|})$ has probability $\propto p_1^{x_1} p_2^{x_2} \cdots p_{|V|}^{x_{|V|}}$.

For naive Bayes: one multinomial distribution per class.

- Class probabilities $\pi_1, \ldots, \pi_k$
- Multinomials $p^{(1)} = (p_{11}, \ldots, p_{1|V|}), \ldots, p^{(k)} = (p_{k1}, \ldots, p_{k|V|})$

Classify document x as

$$\arg\max_j \quad \pi_j \prod_{i=1}^{|V|} p_{ji}^{x_i}.$$

(As always, take log to avoid underflow: linear classifier.)

Improving performance of multinomial naive Bayes

A variety of heuristics that are standard in text retrieval, such as:

1. Compensating for burstiness.
   Problem: Once a word has appeared in a document, it has a much higher chance of appearing again.

Improving performance of multinomial naive Bayes

A variety of heuristics that are standard in text retrieval, such as:

1. Compensating for burstiness.
   Problem: Once a word has appeared in a document, it has a much higher chance of appearing again.

   Solution: Instead of the number of occurrences f of a word, use $\log(1 + f)$.

Improving performance of multinomial naive Bayes

A variety of heuristics that are standard in text retrieval, such as:

1. Compensating for burstiness.
   Problem: Once a word has appeared in a document, it has a much higher chance of appearing again.

   Solution: Instead of the number of occurrences f of a word, use $\log(1 + f)$.

2. Downweighting common words.
   Problem: Common words can have a unduly large influence on classification.

Improving performance of multinomial naive Bayes

A variety of heuristics that are standard in text retrieval, such as:

1. Compensating for burstiness.
   Problem: Once a word has appeared in a document, it has a much higher chance of appearing again.

   Solution: Instead of the number of occurrences f of a word, use $\log(1 + f)$.

2. Downweighting common words.
   Problem: Common words can have a unduly large influence on classification.

   Solution: Weight each word w by **inverse document frequency**:

$$\log \frac{\#\text{ docs}}{\#(\text{docs containing } w)}$$