



# Bayesian Classifiers and Kernel Density Estimation

PHYS 453

Dr Daugherty

# Neighbors???

Mystery for today: what is this thing? And what does it have to do with neighbors?

## `sklearn.neighbors.KernelDensity`

```
class sklearn.neighbors.KernelDensity(bandwidth=1.0, algorithm='auto', kernel='gaussian', metric='euclidean', atol=0, rtol=0,
breadth_first=True, leaf_size=40, metric_params=None)
```

[\[source\]](#)

Kernel Density Estimation.

Read more in the [User Guide](#).

### Parameters:

**bandwidth : float**

The bandwidth of the kernel.

**algorithm : str**

The tree algorithm to use. Valid options are ['kd\_tree'|'ball\_tree'|'auto']. Default is 'auto'.

**kernel : str**

The kernel to use. Valid kernels are ['gaussian'|'tophat'|'epanechnikov'|'exponential'|'linear'|'cosine'] Default is 'gaussian'.

**metric : str**

The distance metric to use. Note that not all metrics are valid with all algorithms. Refer to the documentation of [BallTree](#) and [KDTree](#) for a description of available algorithms. Note that the normalization of the density output is correct only for the Euclidean distance metric. Default is 'euclidean'.

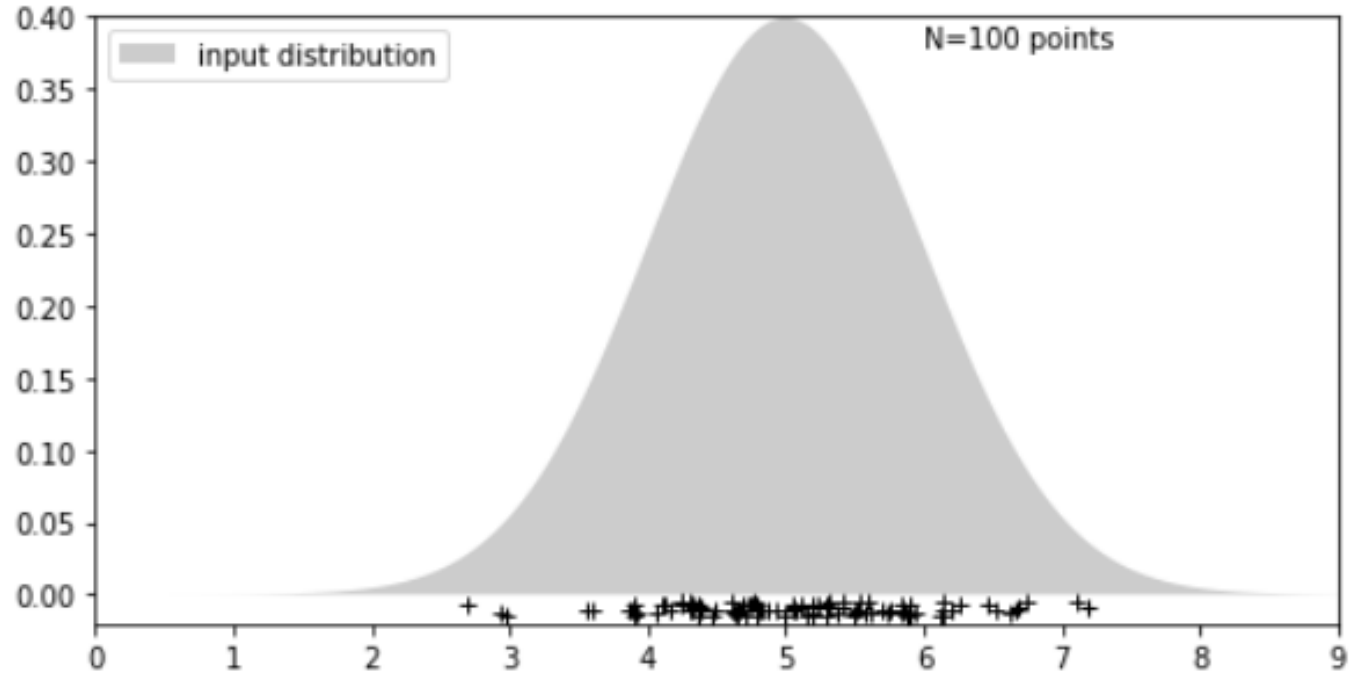
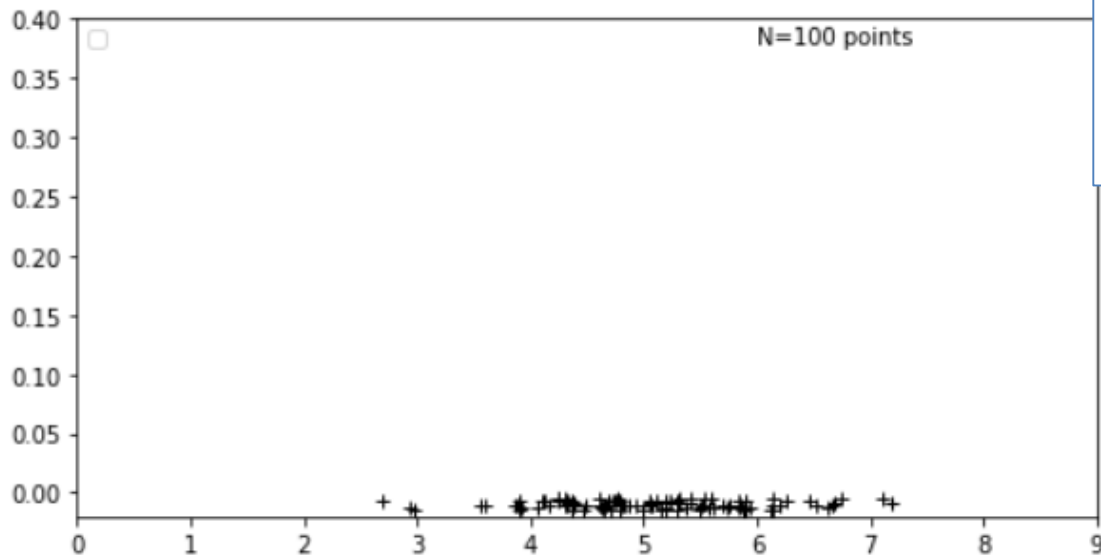
# Kernel Density Estimation

# The Problem

Given a list of training data points:

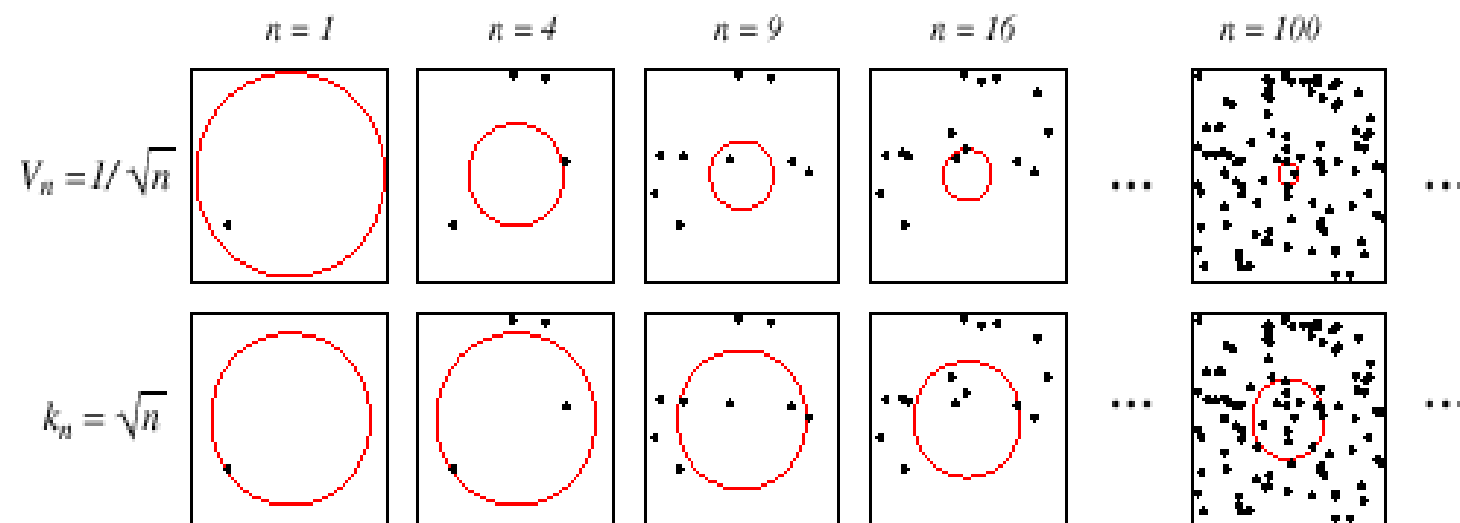
```
array([6.62434536, 4.38824359, 4.47182825, 3.92703138, 5.86540763,  
       2.6984613 , 6.74481176, 4.2387931 , 5.3190391 , 4.75062962,  
       6.46210794, 2.93985929, 4.6775828 , 4.61594565, 6.13376944,  
       3.90010873, 4.82757179, 4.12214158, 5.04221375, 5.58281521,  
       3.89938082, 6.14472371, 5.90159072, 5.50249434, 5.90085595,  
       4.31627214, 4.87710977, 4.06423057, 4.73211192, 5.53035547,  
       4.30833925, 4.60324647, 4.3128273 , 4.15479436, 4.32875387,  
       4.9873354 , 3.88268965, 5.2344157 , 6.65980218, 5.74204416,  
       4.80816445, 4.11237104, 4.25284171, 6.6924546 , 5.05080775,  
       4.36300435, 5.19091548, 7.10025514, 5.12015895, 5.61720311,  
       5.30017032, 4.64775015, 3.8574818 , 4.65065728, 4.79110577,  
       5.58662319, 5.83898341, 5.93110208, 5.28558733, 5.88514116,  
       4.24560206, 6.25286816, 5.51292982, 4.70190716, 5.48851815,  
       4.92442829, 6.13162939, 6.51981682, 7.18557541, 3.60350366,  
       3.55588619, 4.49553414, 5.16003707, 5.87616892, 5.31563495,  
       2.97779878, 4.69379599, 5.82797464, 5.23009474, 5.76201118,  
       4.77767186, 4.79924193, 5.18656139, 5.41005165, 5.19829972,  
       5.11900865, 4.32932771, 5.37756379, 5.12182127, 6.12948391])
```

Estimate a probability distribution function



## One approach:

To estimate the probability at a point, grow a circle until it contains  $k$  neighbors, then the probability is proportional to  $k / \text{“Volume”}$  of the region.

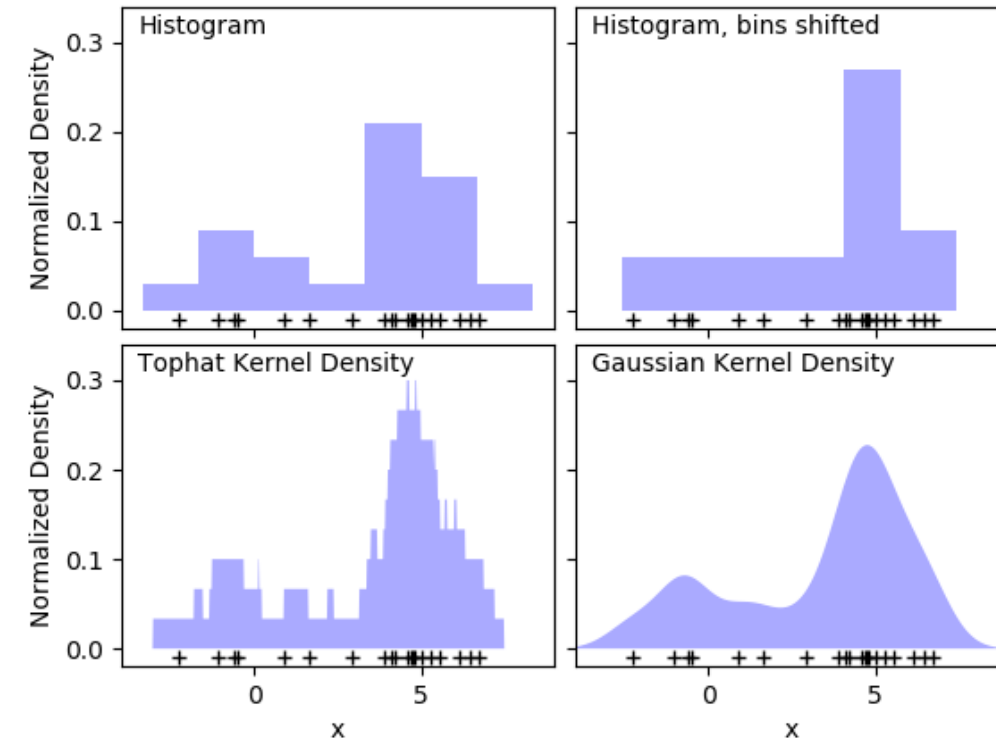
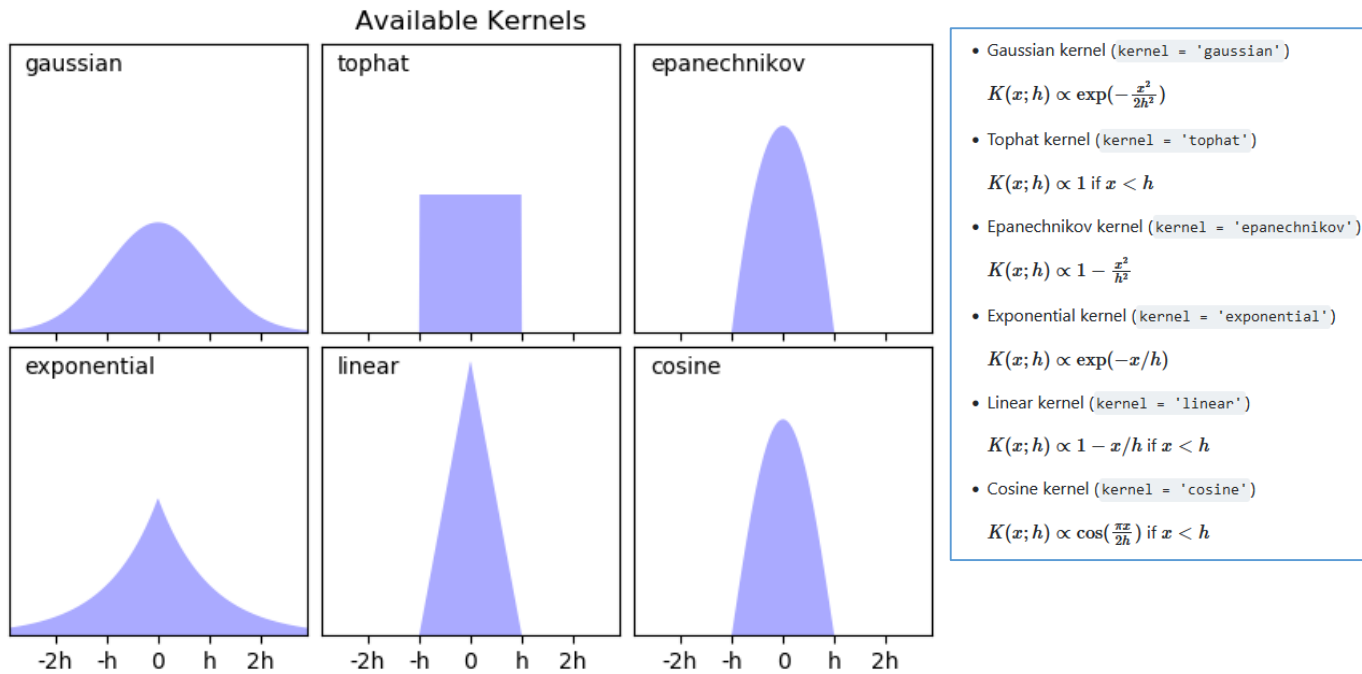


**FIGURE 4.2.** There are two leading methods for estimating the density at a point, here at the center of each square. The one shown in the top row is to start with a large volume centered on the test point and shrink it according to a function such as  $V_n = 1/\sqrt{n}$ . The other method, shown in the bottom row, is to decrease the volume in a data-dependent way, for instance letting the volume enclose some number  $k_n = \sqrt{n}$  of sample points. The sequences in both cases represent random variables that generally converge and allow the true density at the test point to be calculated. From: Richard O. Duda, Peter E. Hart, and David G. Stork, *Pattern Classification*. Copyright © 2001 by John Wiley & Sons, Inc.

# Reverse approach:

Draw a “kernel” (a shape with some set width) centered on every data point. The probability is proportional to the **sum** of all of these shapes.

[https://scikit-learn.org/stable/auto\\_examples/neighbors/plot\\_kde\\_1d.html](https://scikit-learn.org/stable/auto_examples/neighbors/plot_kde_1d.html)



To estimate the probability at a point, find all “close” neighbors and add their kernels.

```
mean = 5
stdev = 1
X_train = np.random.normal(loc=mean, scale=stdev, size=(50,1))

X_plot = np.linspace(0, 10, num=1000).reshape(-1,1) # make a 2D array
y_true = norm(mean, stdev).pdf(X_plot)

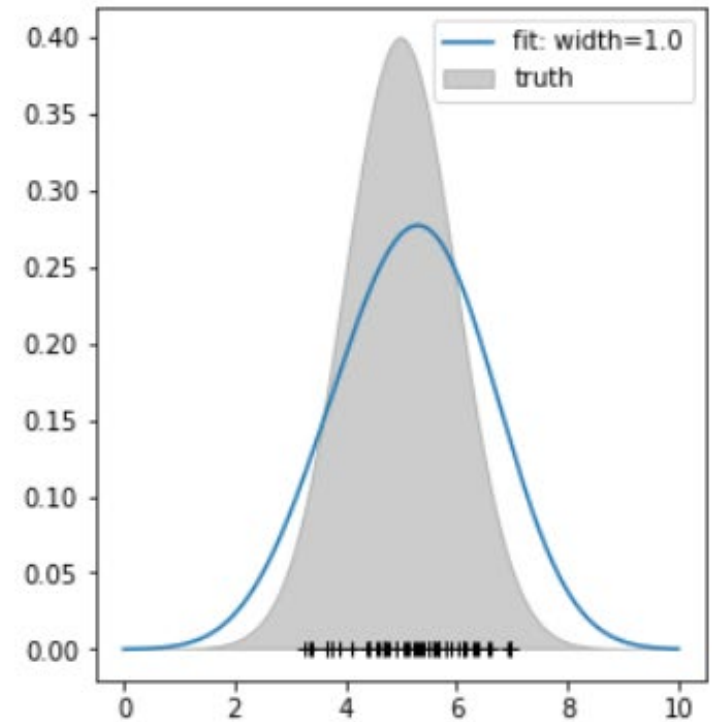
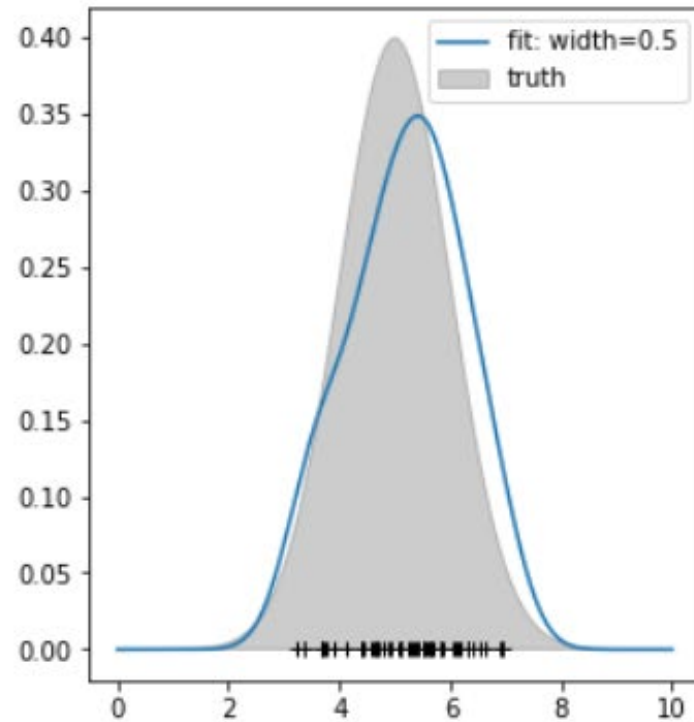
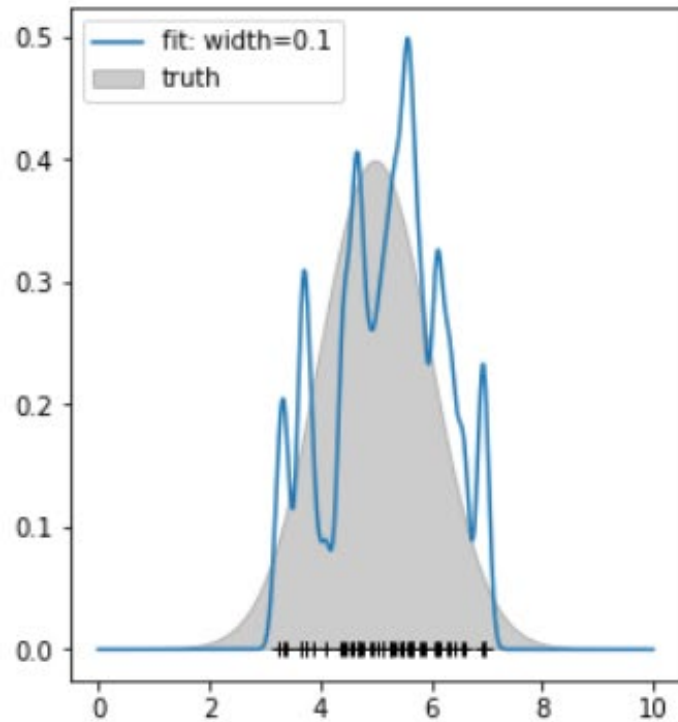
plt.figure(figsize=(15,5))
widths = [0.1, 0.5, 1.0]
for i,w in enumerate(widths):
    plt.subplot(1,3,i+1)

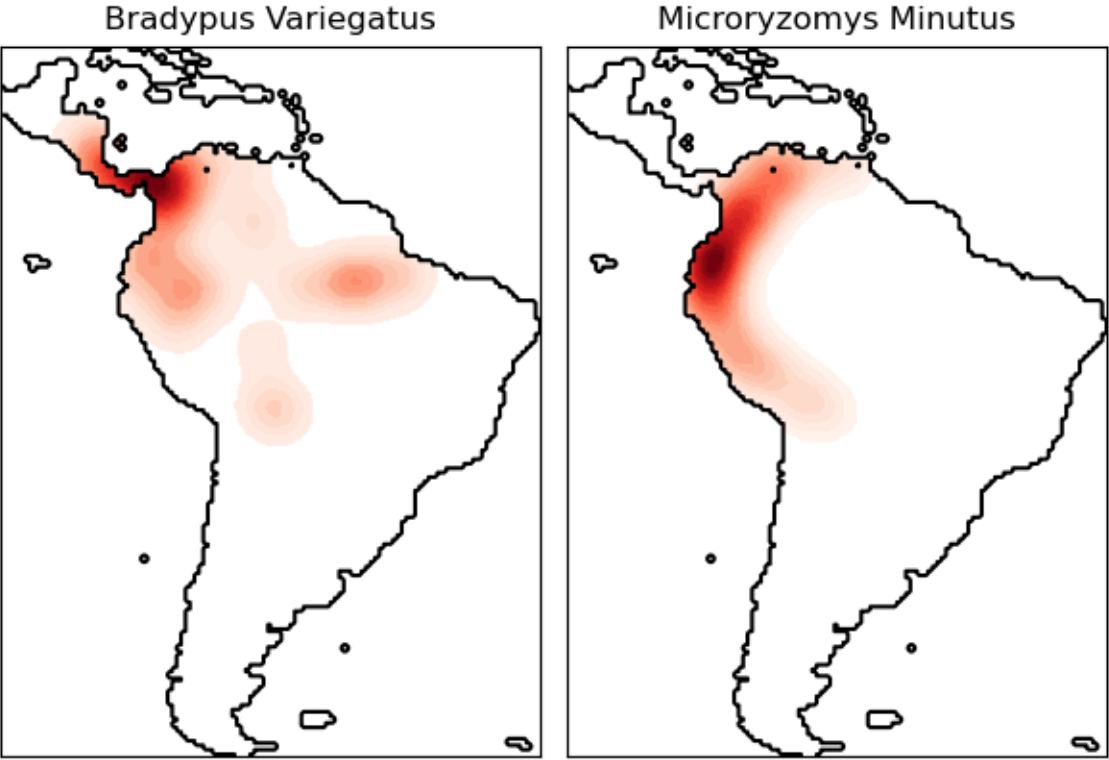
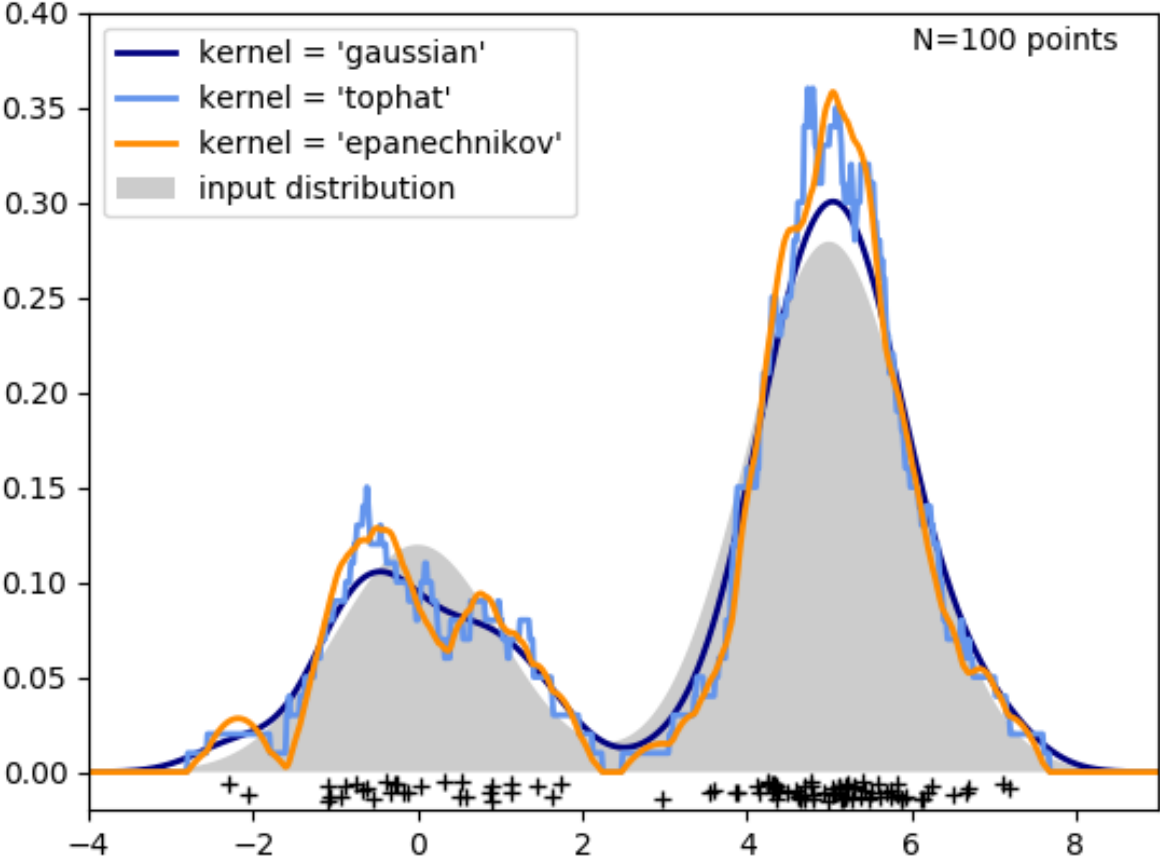
    plt.plot(X_train, np.zeros_like(X_train), 'k+')

    plt.fill(X_plot, y_true, c='black', alpha=0.2, label='truth')

    kde = KernelDensity(bandwidth=w)
    kde.fit(X_train)
    log_dens = kde.score_samples(X_plot)
    plt.plot(X_plot, np.exp(log_dens), label=f'fit: width={w}')

plt.legend()
plt.show()
```

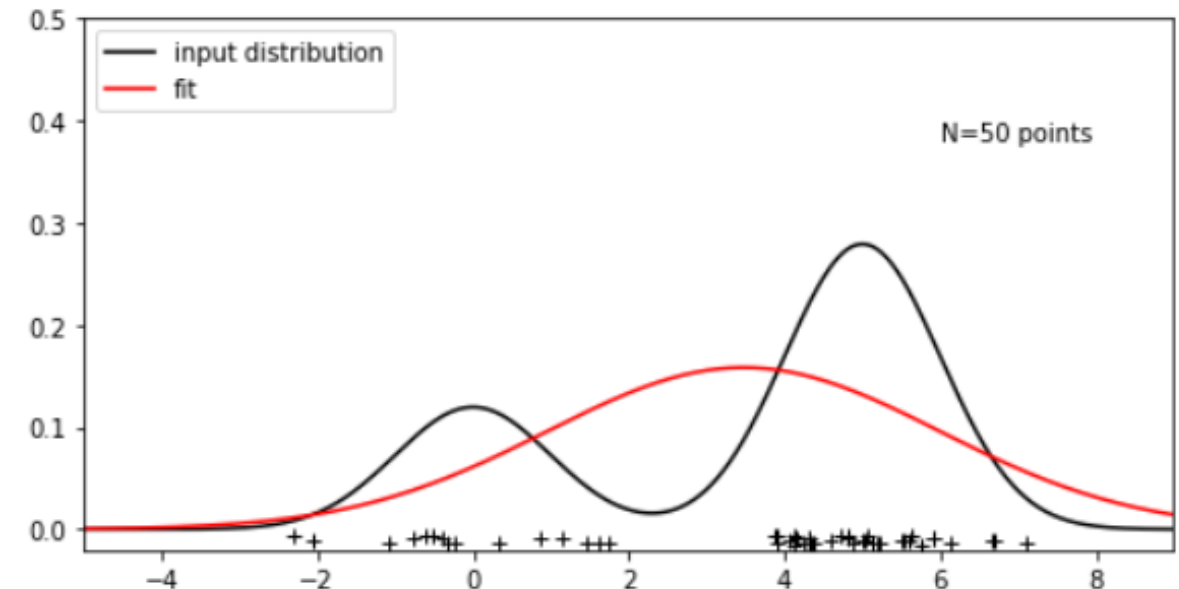
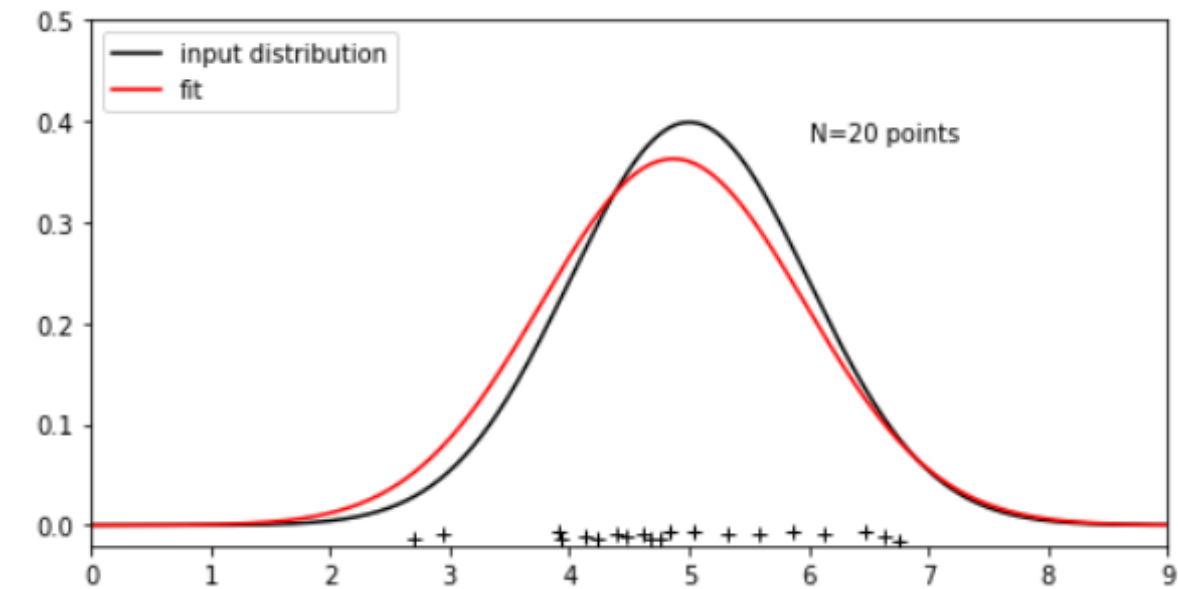






### Third approach:

Fit the entire distribution to some function and hope for the best.



# Bayesian Statistics

# Bayesian Statistics

So what can I do if I have an equation for the probability distributions?

A great option is to apply Bayes' Theorem to get the probabilities of each class

<https://vimeo.com/71446628>



## Randy Harris - Week 6

More from ACU Core

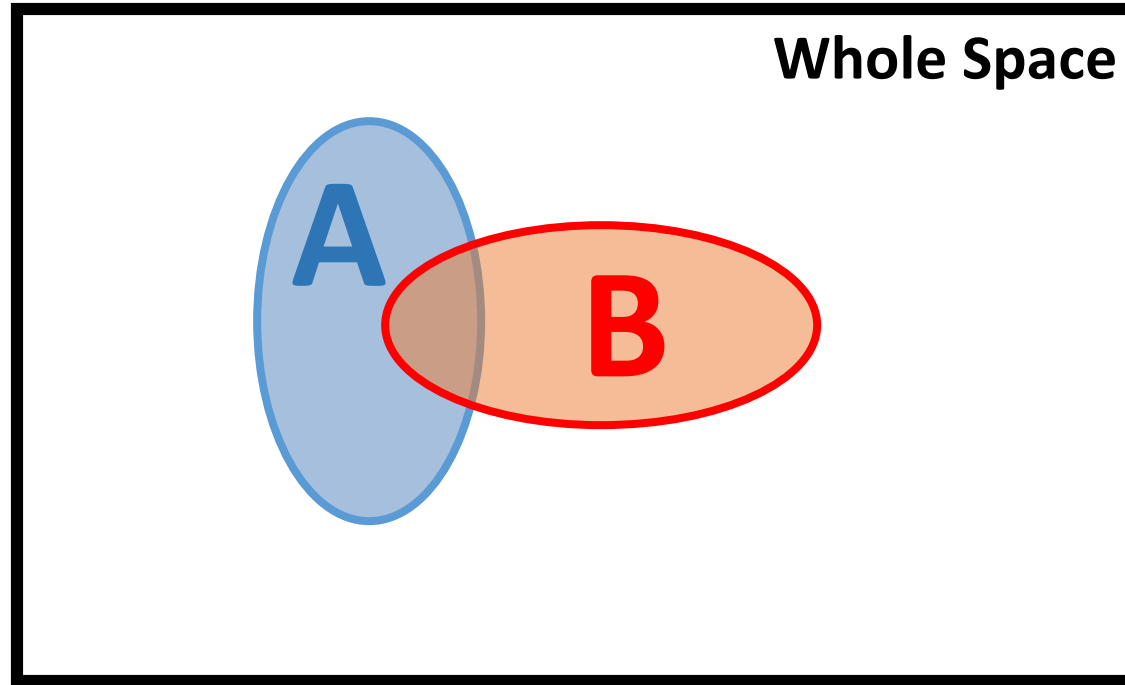
☒ Autoplay next video

Randy Harris  
September 30, 2013  
(Start at 4:00 mark )

# Graphical Derivation of Bayes' Theorem

Source: [http://www.science20.com/quantum\\_diaries\\_survivor/you\\_bayesian](http://www.science20.com/quantum_diaries_survivor/you_bayesian)

Define subsets A and B



Basic Probabilities:

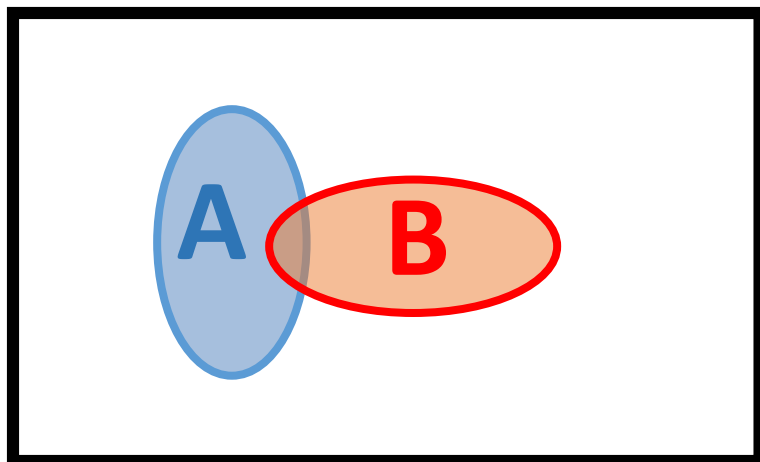
$$P(A) = \frac{\text{Area of A}}{\text{Area of Whole Space}}$$

A diagram illustrating the probability P(A). It consists of a horizontal line. Above the line is a small light blue oval, representing subset A. Below the line is a gray rectangle, representing the whole space. The fraction P(A) = is placed to the left of the line.

$$P(B) = \frac{\text{Area of B}}{\text{Area of Whole Space}}$$

A diagram illustrating the probability P(B). It consists of a horizontal line. Above the line is a small light orange oval, representing subset B. Below the line is a gray rectangle, representing the whole space. The fraction P(B) = is placed to the left of the line.

# Combined Probabilities



Intersection:

$$P(A \cap B) = \frac{\text{Intersection}}{\text{Universal Set}}$$

**A and B**

Conditional Probabilities:

$$P(A | B) = \frac{\text{Intersection}}{\text{Set B}}$$

**A given B**

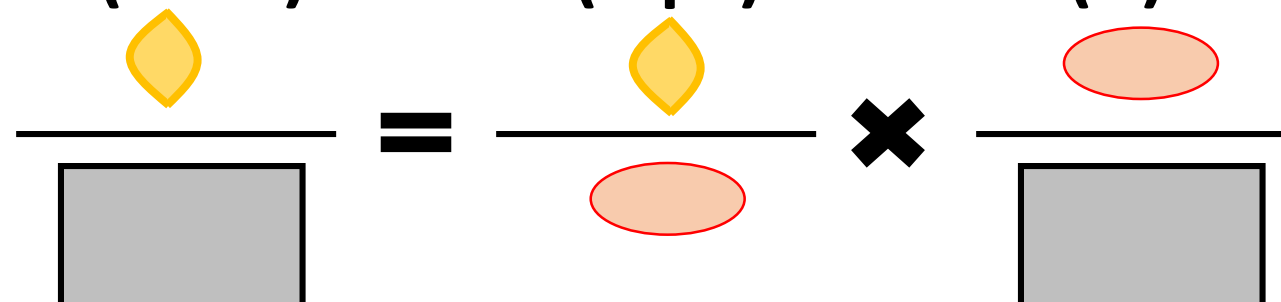
$$P(B | A) = \frac{\text{Intersection}}{\text{Set A}}$$

**B given A**

Derivation:

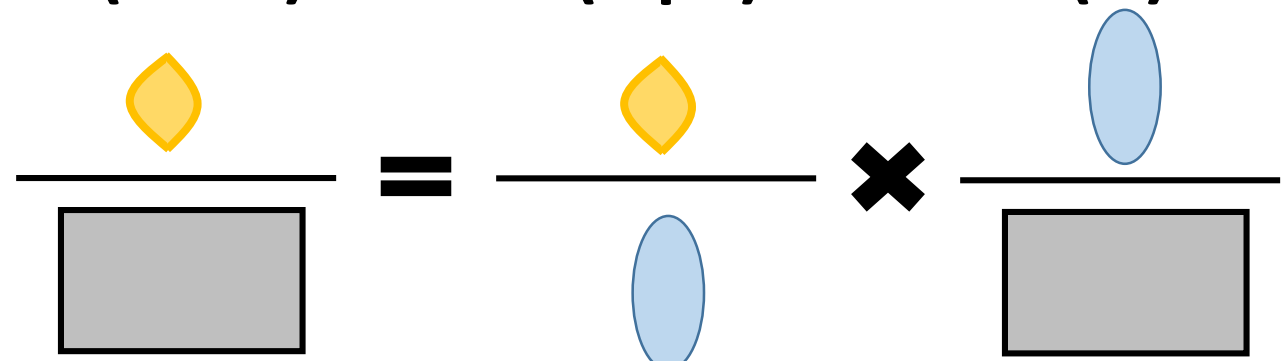
$$P(A \cap B) = P(A|B) * P(B) = P(B|A) * P(A)$$

Check 1:  $P(A \cap B) = P(A|B) * P(B)$



The diagram illustrates the relationship between the probability of the intersection of two events, the conditional probability of A given B, and the probability of B. It consists of three parts: 1. A Venn diagram with two overlapping circles, A and B. The intersection of A and B is shaded gray. 2. An equals sign. 3. A Venn diagram with two overlapping circles, A and B. The intersection of A and B is shaded gray. 4. A multiplication sign. 5. A Venn diagram with two overlapping circles, A and B. The intersection of A and B is shaded gray. 6. A yellow smiley face.

Check 2:  $P(A \cap B) = P(B|A) * P(A)$

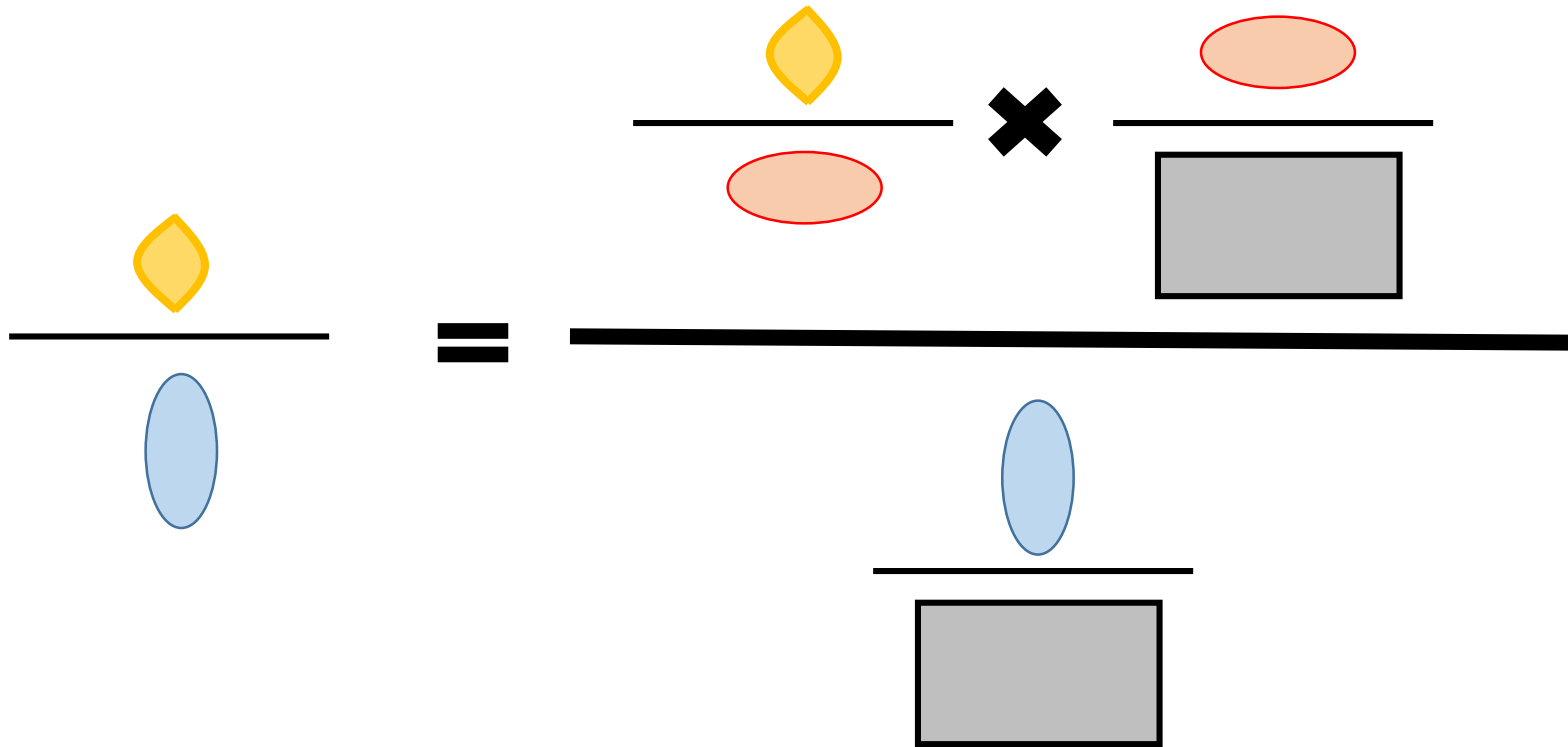


The diagram illustrates the relationship between the probability of the intersection of two events, the conditional probability of B given A, and the probability of A. It consists of three parts: 1. A Venn diagram with two overlapping circles, A and B. The intersection of A and B is shaded gray. 2. An equals sign. 3. A Venn diagram with two overlapping circles, A and B. The intersection of A and B is shaded gray. 4. A multiplication sign. 5. A Venn diagram with two overlapping circles, A and B. The intersection of A and B is shaded gray. 6. A yellow smiley face.

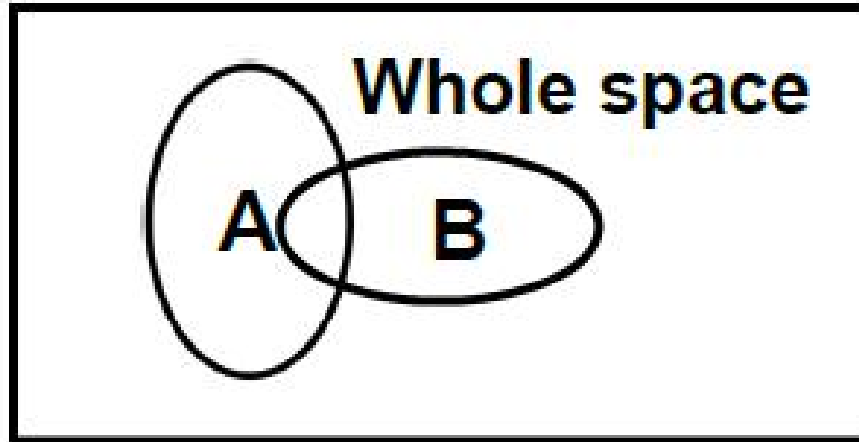


Final “Proof”:

$$P(B|A) = \frac{P(A \cap B)}{P(A)} = \frac{P(A|B) P(B)}{P(A)}$$



# P, Conditional P, and Derivation of Bayes' Theorem in Pictures



$$P(A) = \frac{\text{Area of A}}{\text{Area of Whole space}}$$

$$P(B) = \frac{\text{Area of B}}{\text{Area of Whole space}}$$

$$P(A|B) = \frac{\text{Area of A} \cap B}{\text{Area of B}}$$

$$P(B|A) = \frac{\text{Area of A} \cap B}{\text{Area of A}}$$

$$P(A \cap B) = \frac{\text{Area of A} \cap B}{\text{Area of Whole space}}$$

$$P(A) \times P(B|A) = \frac{\text{Area of A}}{\text{Area of Whole space}} \times \frac{\text{Area of A} \cap B}{\text{Area of A}} = \frac{\text{Area of A} \cap B}{\text{Area of Whole space}} = P(A \cap B)$$

$$P(B) \times P(A|B) = \frac{\text{Area of B}}{\text{Area of Whole space}} \times \frac{\text{Area of A} \cap B}{\text{Area of B}} = \frac{\text{Area of A} \cap B}{\text{Area of Whole space}} = P(A \cap B)$$

$$\Rightarrow P(B|A) = P(A|B) \times P(B) / P(A)$$

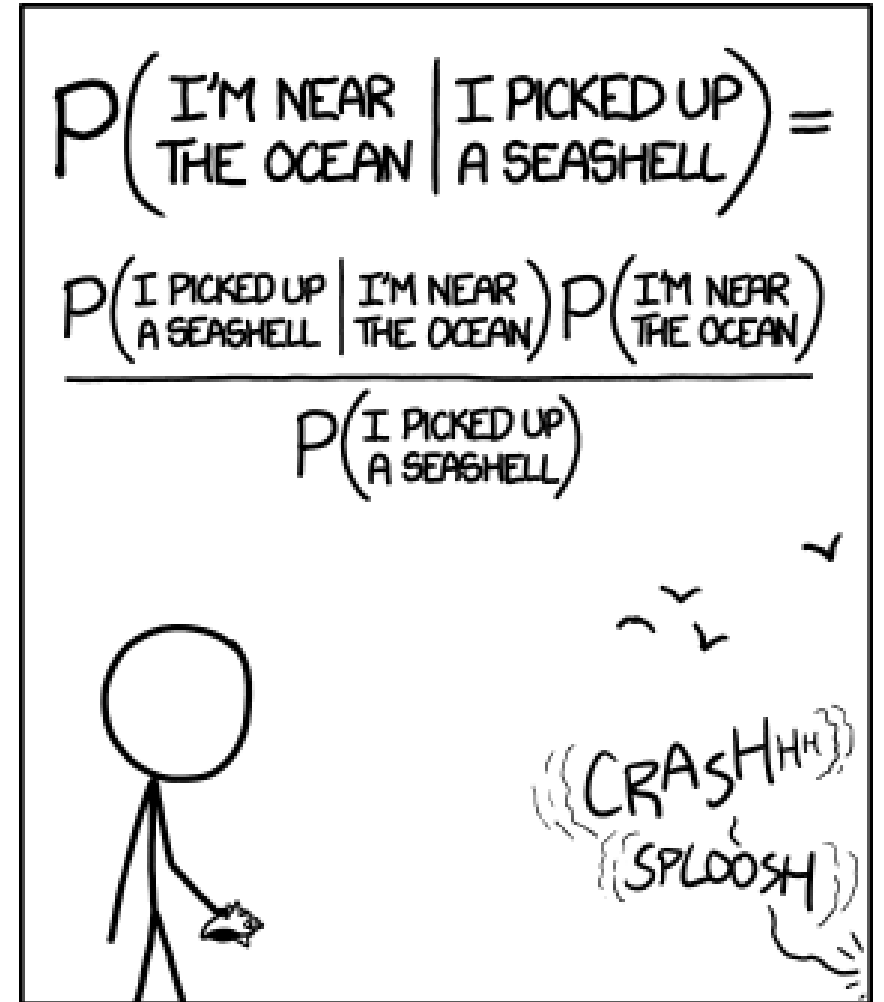
# Is it Christmas?

NO\*

\*99.73% ACCURATE

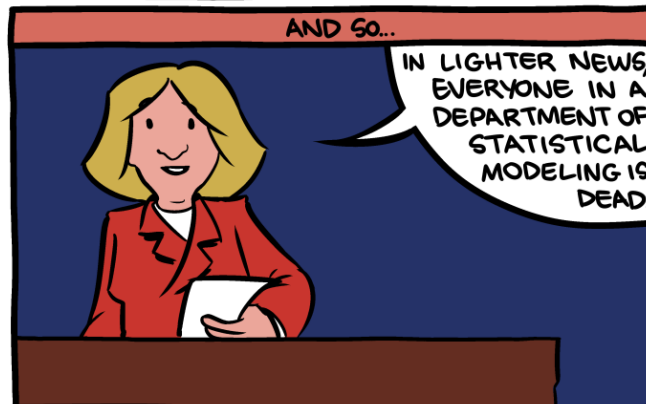
XKCD.COM PRESENTS A NEW "IS IT CHRISTMAS"  
SERVICE TO COMPETE WITH ISITCHRISTMAS.COM

<https://xkcd.com/2236/>



STATISTICALLY SPEAKING, IF YOU PICK UP A  
SEASHELL AND DON'T HOLD IT TO YOUR EAR,  
YOU CAN PROBABLY HEAR THE OCEAN.

<https://xkcd.com/1236/>



# Bayesian Statistics for Classifiers

# Bayes Theorem in Classification

**Prior**

Prob of target  $y$

**Likelihood**

Prob of features  $X$  given target  $y$ ,  
measured in training data

$$P(y \mid x_1, \dots, x_n) = \frac{P(y)P(x_1, \dots, x_n \mid y)}{P(x_1, \dots, x_n)}$$

**Posterior**

Prob of target  $y$   
given features  $X$

**Evidence**

Prob of features  $X$ , only purpose  
is normalization so we can just  
ignore this!

$$P(y \mid x_1, \dots, x_n) = \frac{P(y)P(x_1, \dots, x_n \mid y)}{P(x_1, \dots, x_n)}$$

Each feature is fit separately (as a 1D fit), so total prob of getting this sample is the product of all of these 1D gaussians.

$$P(y \mid x_1, \dots, x_n) = \frac{P(y) \prod_{i=1}^n P(x_i \mid y)}{P(x_1, \dots, x_n)}$$

### 1.9.1. Gaussian Naive Bayes

`GaussianNB` implements the Gaussian Naive Bayes algorithm for classification. The likelihood of the features is assumed to be Gaussian:

$$P(x_i \mid y) = \frac{1}{\sqrt{2\pi\sigma_y^2}} \exp\left(-\frac{(x_i - \mu_y)^2}{2\sigma_y^2}\right)$$

The parameters  $\sigma_y$  and  $\mu_y$  are estimated using maximum likelihood.

Since the denominator  $P(x_1, \dots, x_n)$  is a constant, we ignore it and just choose the largest numerator

$$\hat{y} = \arg \max_y P(y) \prod_{i=1}^n P(x_i \mid y),$$

## sklearn.naive\_bayes.GaussianNB

```
class sklearn.naive_bayes.GaussianNB(*, priors=None, var_smoothing=1e-09)
```

[\[source\]](#)

Gaussian Naive Bayes (GaussianNB).

Can perform online updates to model parameters via [partial\\_fit](#). For details on algorithm used to update feature means and variance online, see Stanford CS tech report STAN-CS-79-773 by Chan, Golub, and LeVeque:

<http://i.stanford.edu/pub/cstr/reports/cs/tr/79/773/CS-TR-79-773.pdf>

Read more in the [User Guide](#).

### Parameters:

**priors** : *array-like of shape (n\_classes,)*

Prior probabilities of the classes. If specified the priors are not adjusted according to the data.

**var\_smoothing** : *float, default=1e-9*

Portion of the largest variance of all features that is added to variances for calculation stability.

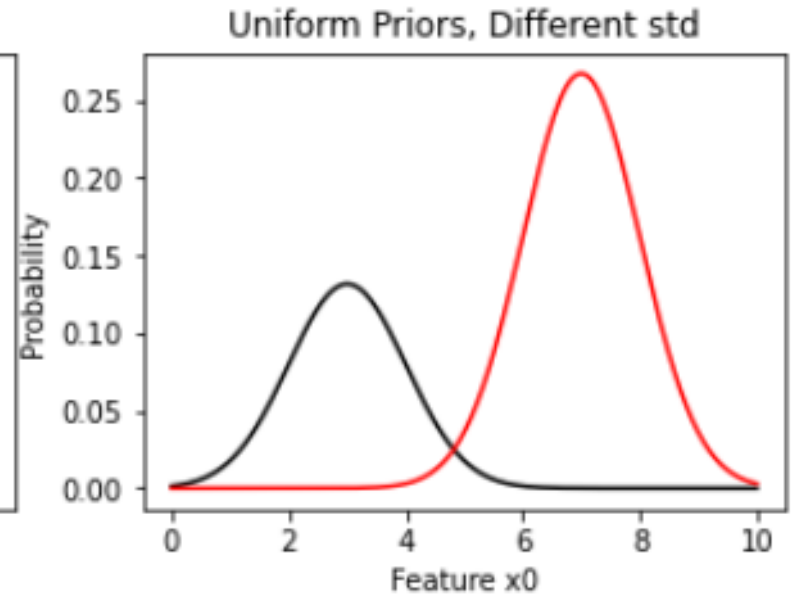
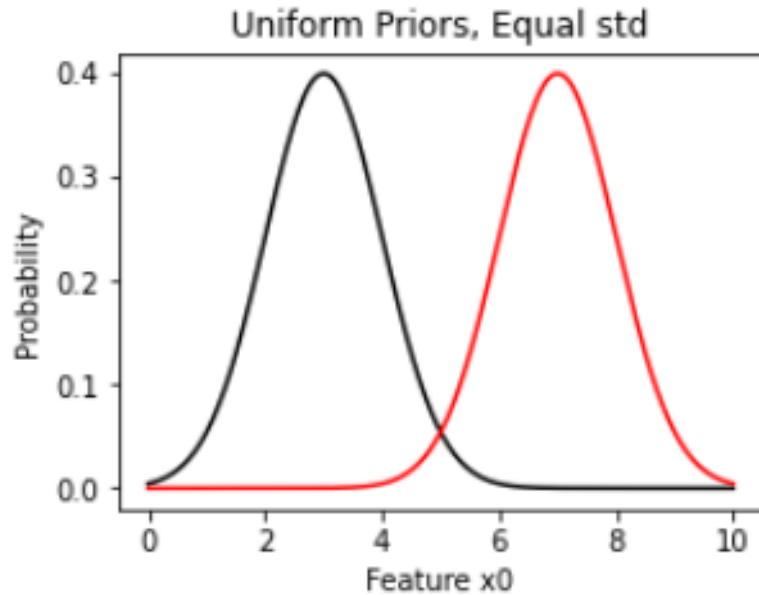
*New in version 0.20.*



# 1-D Examples

$$\hat{y} = \arg \max_y P(y) \prod_{i=1}^n P(x_i | y),$$

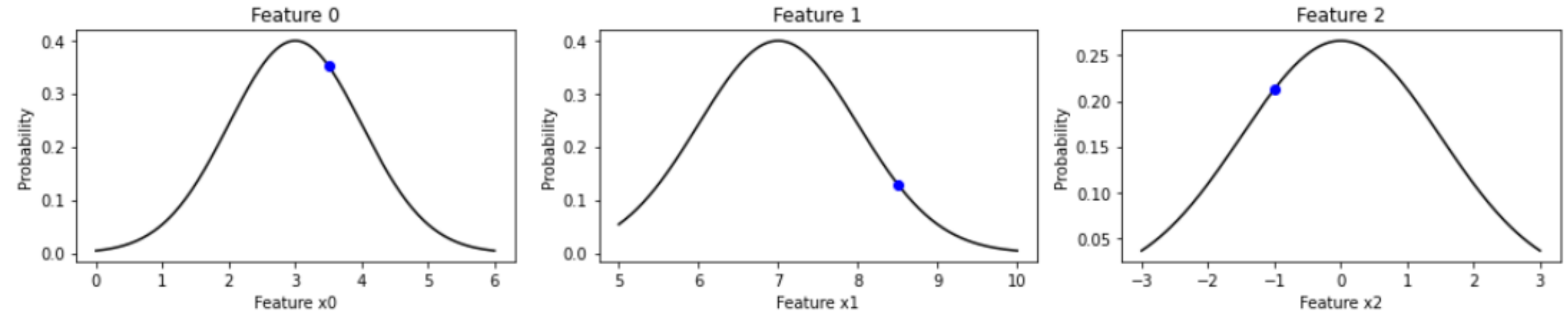
- 1) Use the  $\mu$  and  $\sigma$  for each feature to construct a Gaussian
- 2) Draw the Gaussian times the prior  $P(y)P(x_i|y)$
- 3) To classify, just pick which probability is biggest!

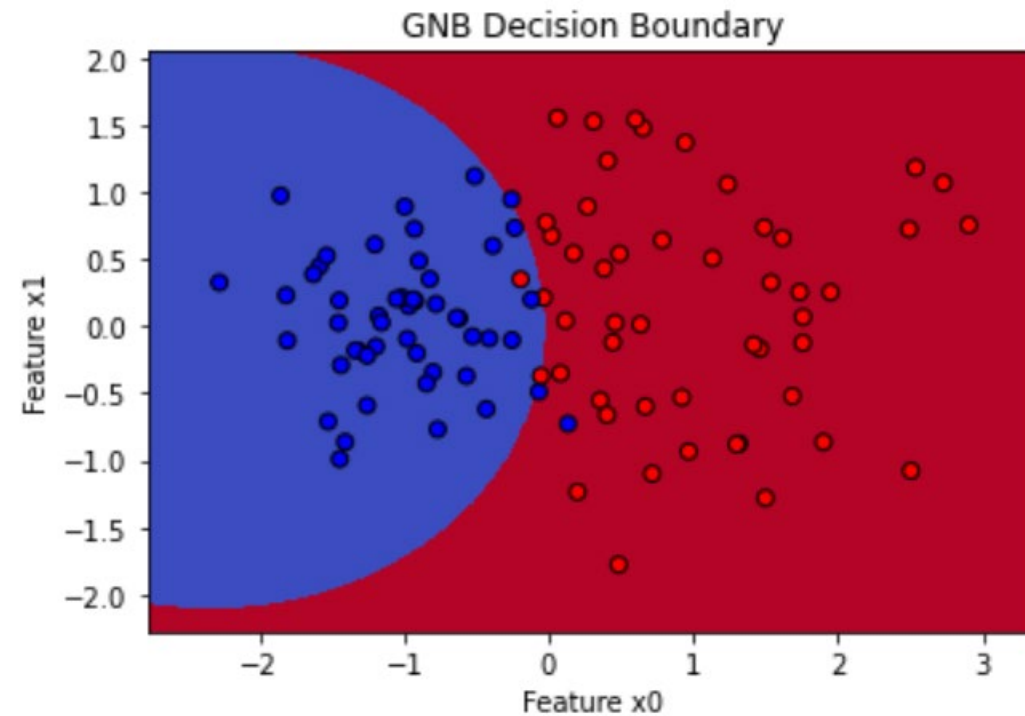
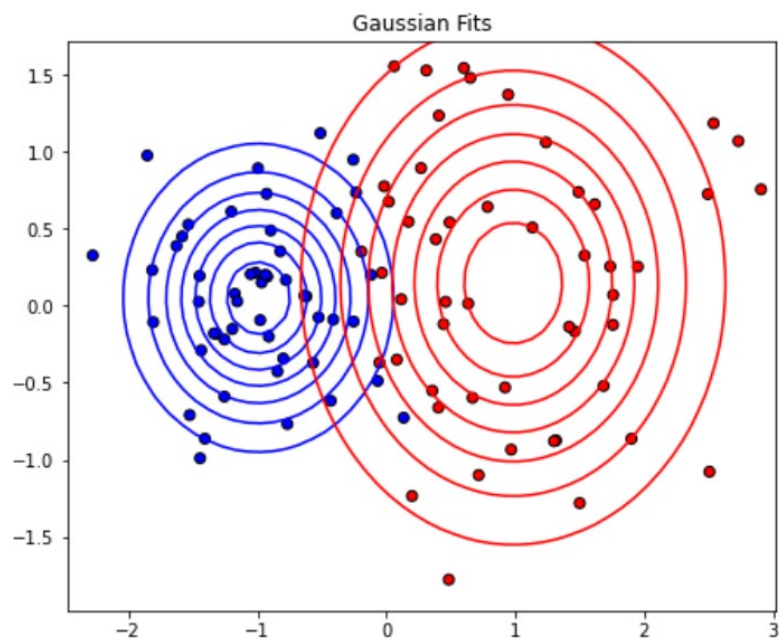
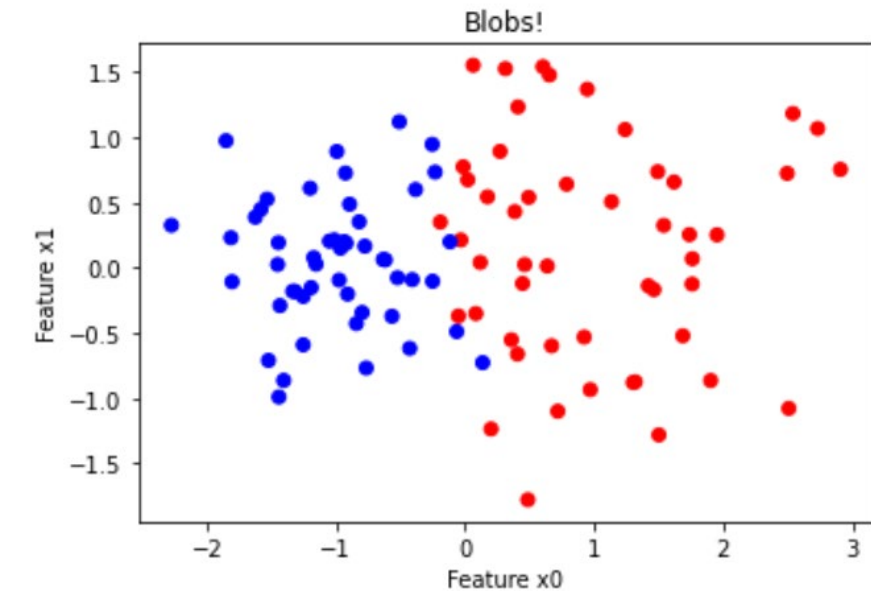


# Higher D

In finding the probability of a sample belonging to a certain class, first look at the Gaussian for each feature which is most likely to be near the mean. We expect that the further it gets away from the mean, the less likely it is to belong to this target.

For multiple features, since we **assume** the features are independent we just multiply the probabilities for each feature together!

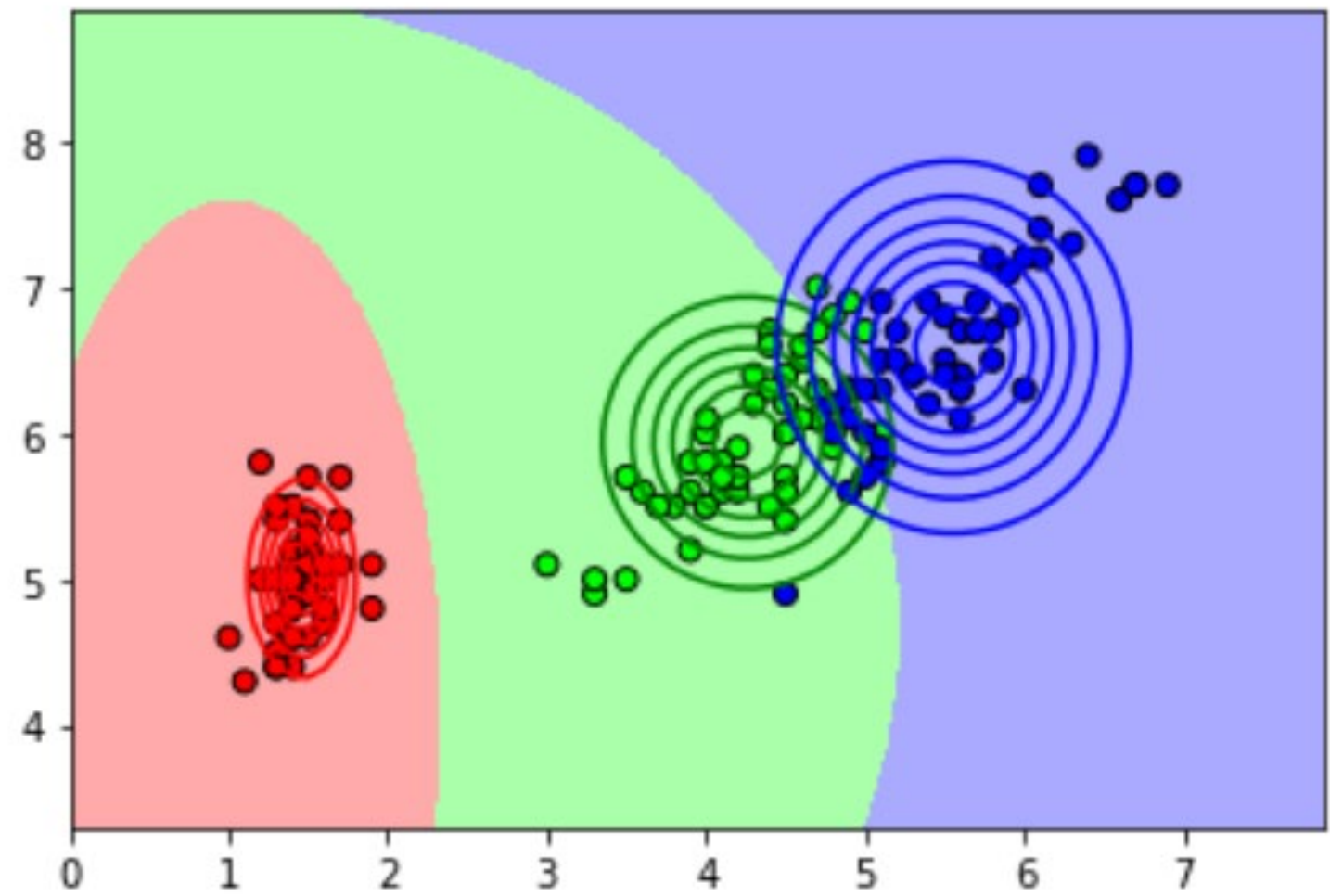


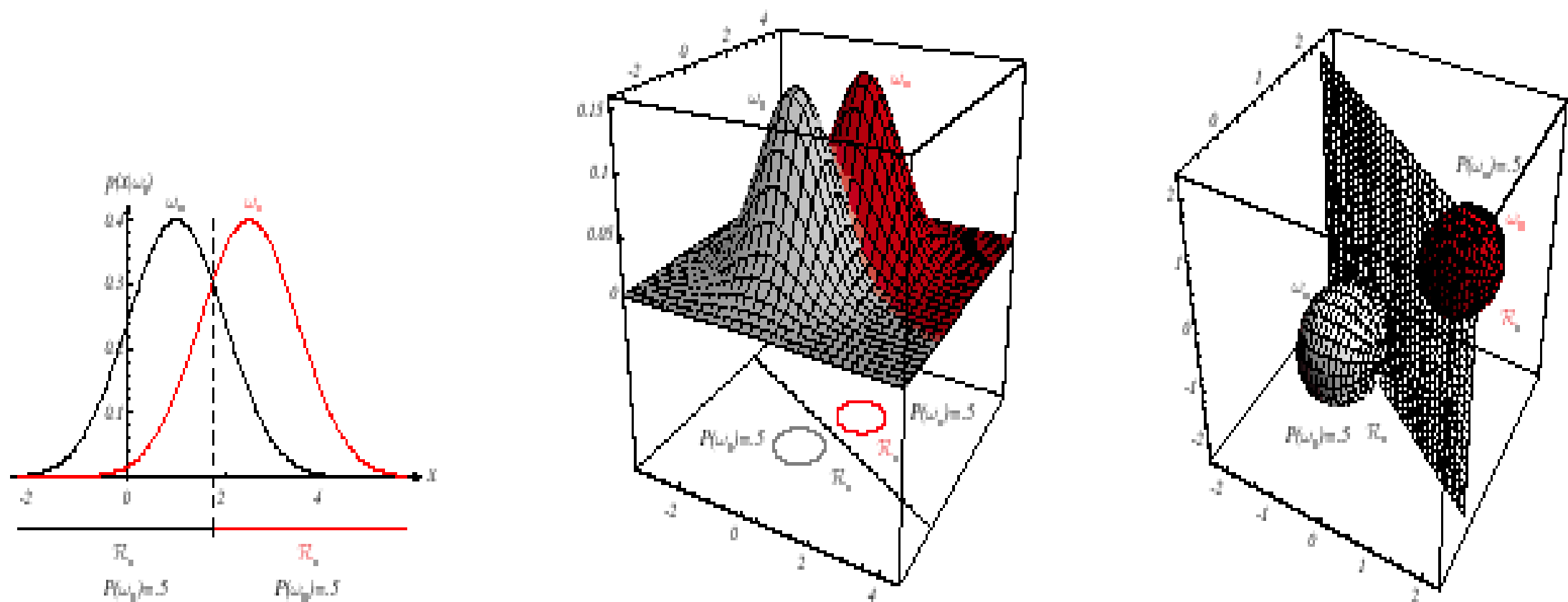


Classifier just picks which Gaussian is tallest at any given point!

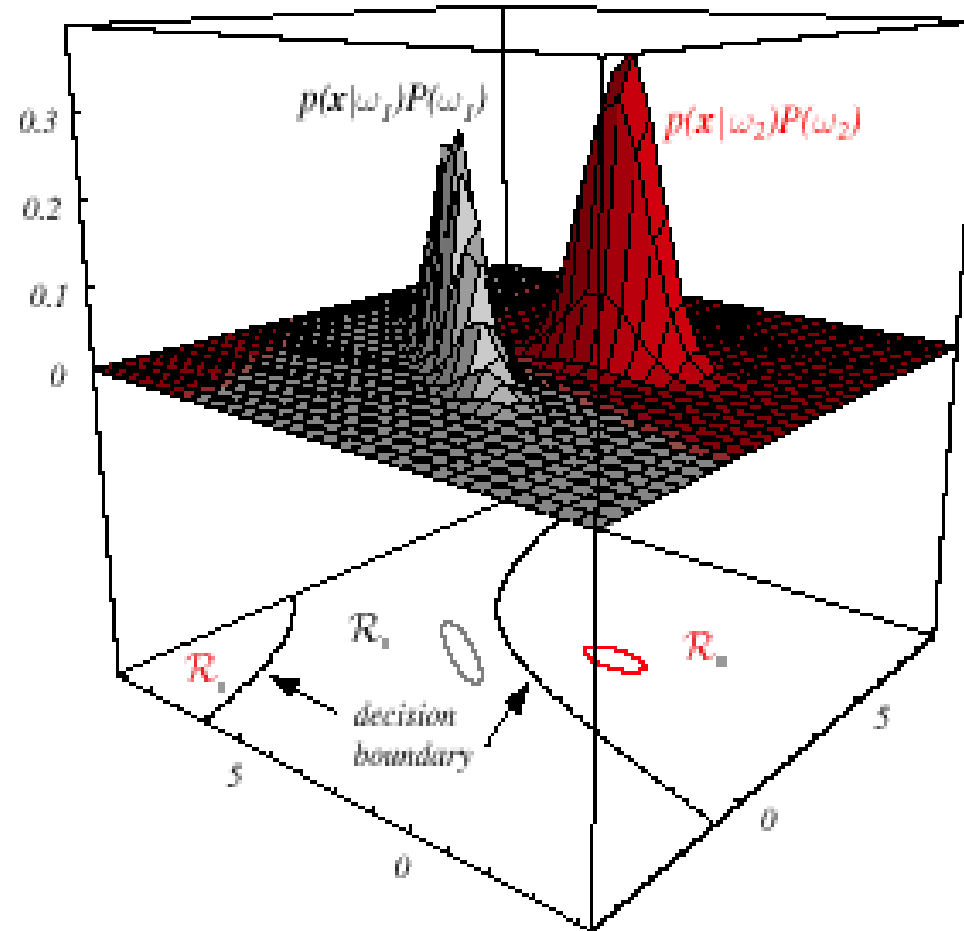
- Equivalent choosing smallest number of  $\sigma$ 's away from mean
- Priors weight the relative heights for each target

# Two feature Iris data





**FIGURE 2.10.** If the covariance matrices for two distributions are equal and proportional to the identity matrix, then the distributions are spherical in  $d$  dimensions, and the boundary is a generalized hyperplane of  $d - 1$  dimensions, perpendicular to the line separating the means. In these one-, two-, and three-dimensional examples, we indicate  $p(\mathbf{x}|\omega_i)$  and the boundaries for the case  $P(\omega_1) = P(\omega_2)$ . In the three-dimensional case, the grid plane separates  $\mathcal{R}_1$  from  $\mathcal{R}_2$ . From: Richard O. Duda, Peter E. Hart, and David G. Stork, *Pattern Classification*. Copyright © 2001 by John Wiley & Sons, Inc.



**FIGURE 2.6.** In this two-dimensional two-category classifier, the probability densities are Gaussian, the decision boundary consists of two hyperbolas, and thus the decision region  $\mathcal{R}_2$  is not simply connected. The ellipses mark where the density is  $1/e$  times that at the peak of the distribution. From: Richard O. Duda, Peter E. Hart, and David G. Stork, *Pattern Classification*. Copyright © 2001 by John Wiley & Sons, Inc.

# Summary

## Pros:

- Bayesian Classifiers actually use real statistics!
- Few parameters
- Fast and memory efficient
- Gaussian works surprisingly well on many problems

## Cons:

- Some problems just aren't Gaussian

# Classifier Comparison

Name	sklearn	Algorithm	Params	Training	Pros	Cons	Notes
Nearest Neighbors (kNN)	<a href="#">neighbors.KNeighborsClassifier</a>	Finds “closest” point in training data	n_neighbors=5 voting weights = (uniform / distance)	Easy! Just copies training data	<ul style="list-style-type: none"><li>• Simple to use</li><li>• Understandable</li></ul>	<ul style="list-style-type: none"><li>• SLOW for big datasets</li></ul>	<ul style="list-style-type: none"><li>• Always scores 100% on training when k=1</li><li>• Increasing k averages over outliers</li></ul>
Decision Tree	<a href="#">tree.DecisionTreeClassifier</a>	20 questions, learns rules ( yes/no questions on one feature at a time) that minimizes impurity	max_depth = None	Easy! Default max_depth=None will overfit	<ul style="list-style-type: none"><li>• Understandable</li><li>• Super fast classification</li></ul>	<ul style="list-style-type: none"><li>• Overfits by default</li><li>• “Stair-step” decision boundaries</li></ul>	<ul style="list-style-type: none"><li>• Limit max_depth!</li></ul>
Gaussian Naïve Bayes	<a href="#">naive_bayes.GaussianNB</a>	Fits 1D gauss to each feature	None (but see note about priors)	Easy! Just finds mean and var of each feature	<ul style="list-style-type: none"><li>• Simple to use</li><li>• Understandable</li><li>• Great if your data is gaussian</li></ul>	<ul style="list-style-type: none"><li>• Terrible if your problem isn’t gaussian</li></ul>	<ul style="list-style-type: none"><li>• Priors learned from y by default, can specify other values</li></ul>