# DIGITAL TWIN AI and Machine Learning: Unsupervised Learning

Prof. Andrew D. Bagdanov
andrew.bagdanov AT unifi.it



Dipartimento di Ingegneria dell'Informazione
Università degli Studi di Firenze

13 November 2020

# Outline

# Introduction

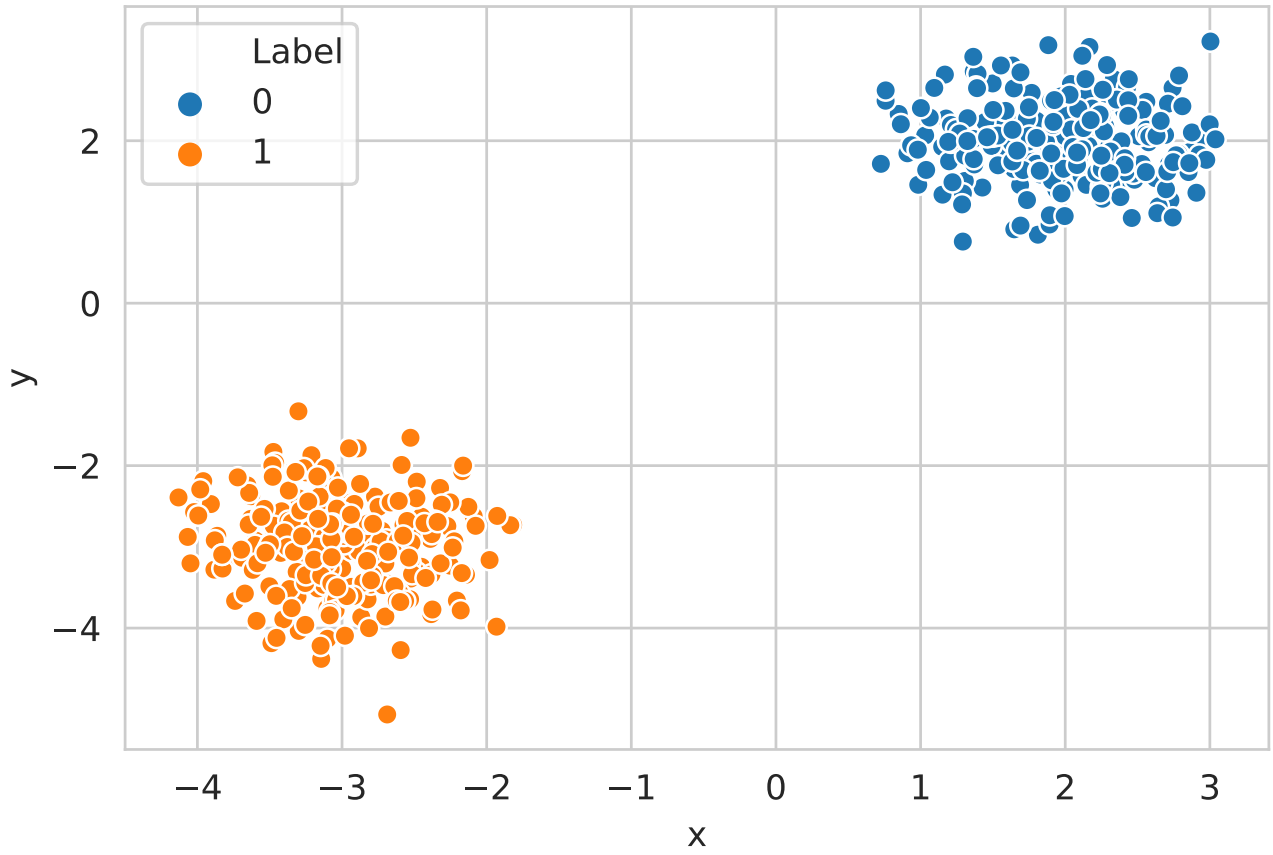
# Overview

- Today we will begin with a few leftover topics from supervised learning.
- We will see a few models that produce probabilistic estimates of class membership (as opposed to just raw scores).
- And we will look at some methods for evaluating classifiers.
- Then I will talk about three types of unsupervised learning approaches that are useful for data modeling and visualization.
- Finally, in the second part of today's lecture we will have a laboratory on supervised learning.

# Leftovers: Supervised Learning

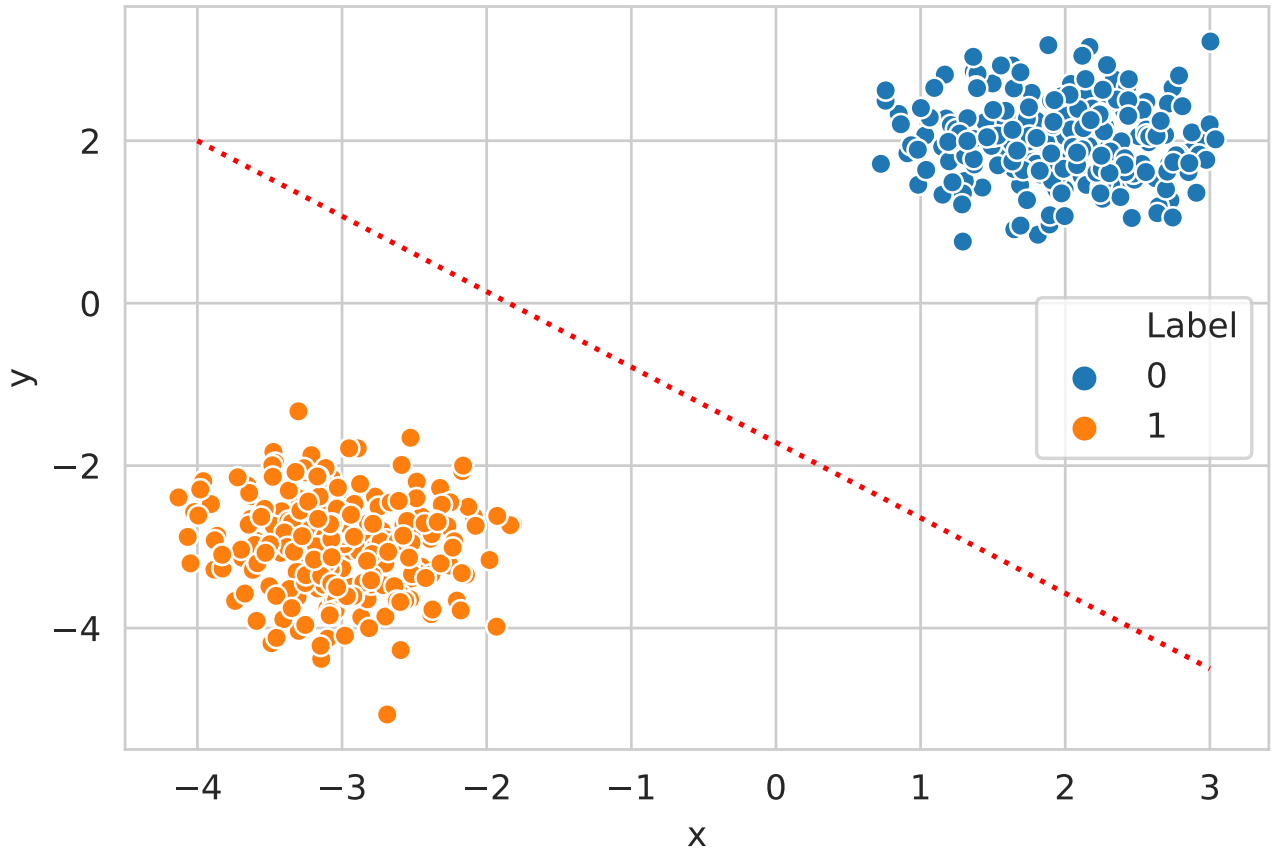
# A topological tangent

- Some of you may have noticed that I have been conveniently ignoring a few crucial issues.



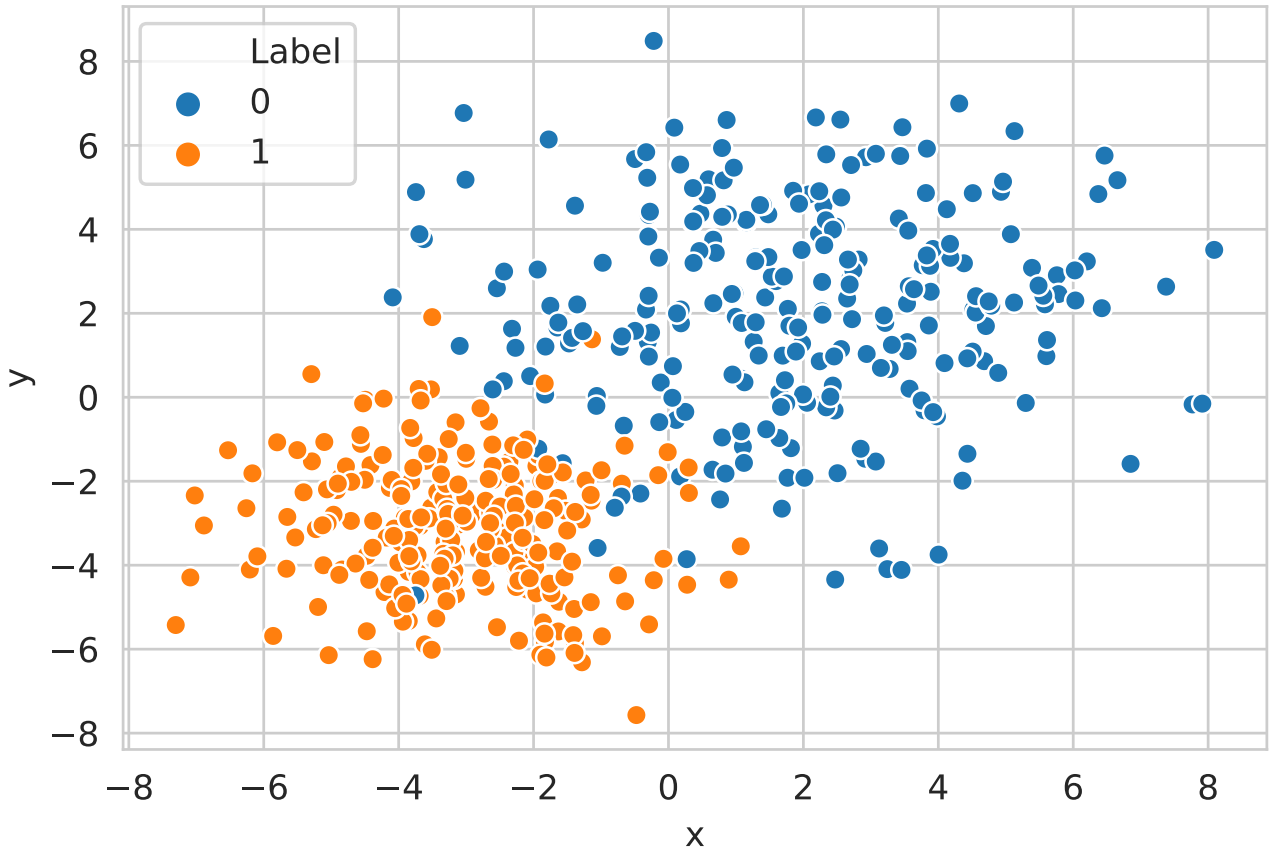
# A topological tangent (continued)

- Some of you may have noticed that I have been conveniently ignoring a crucial issue.



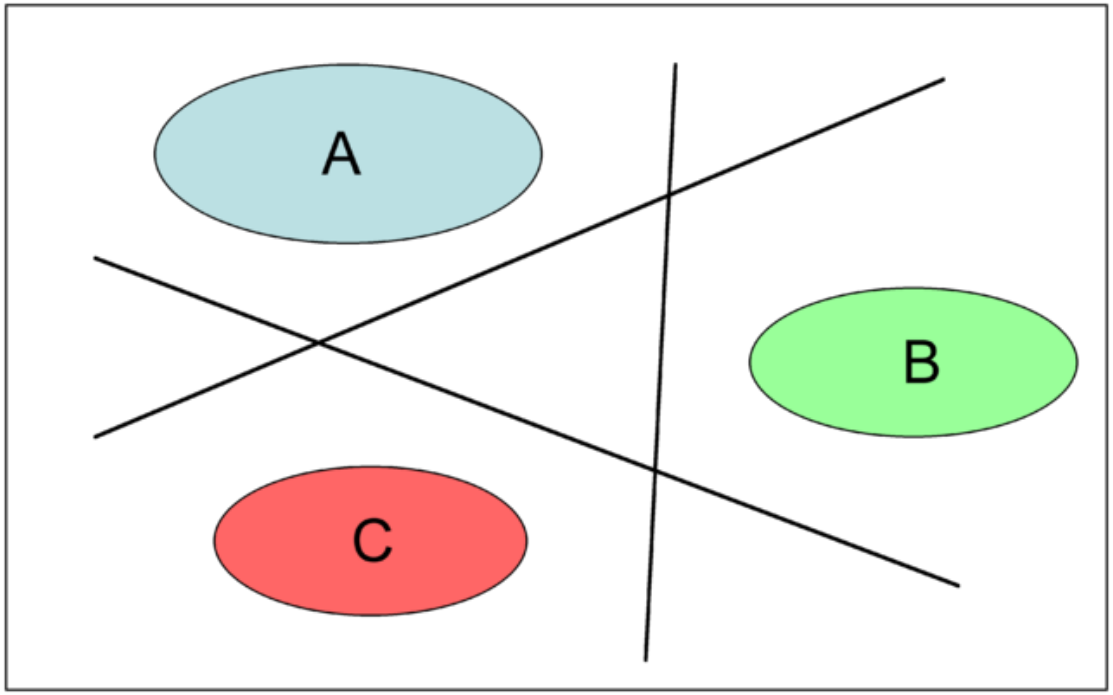
# A topological tangent (continued)

- Some of you may have noticed that I have been conveniently ignoring a crucial issue.



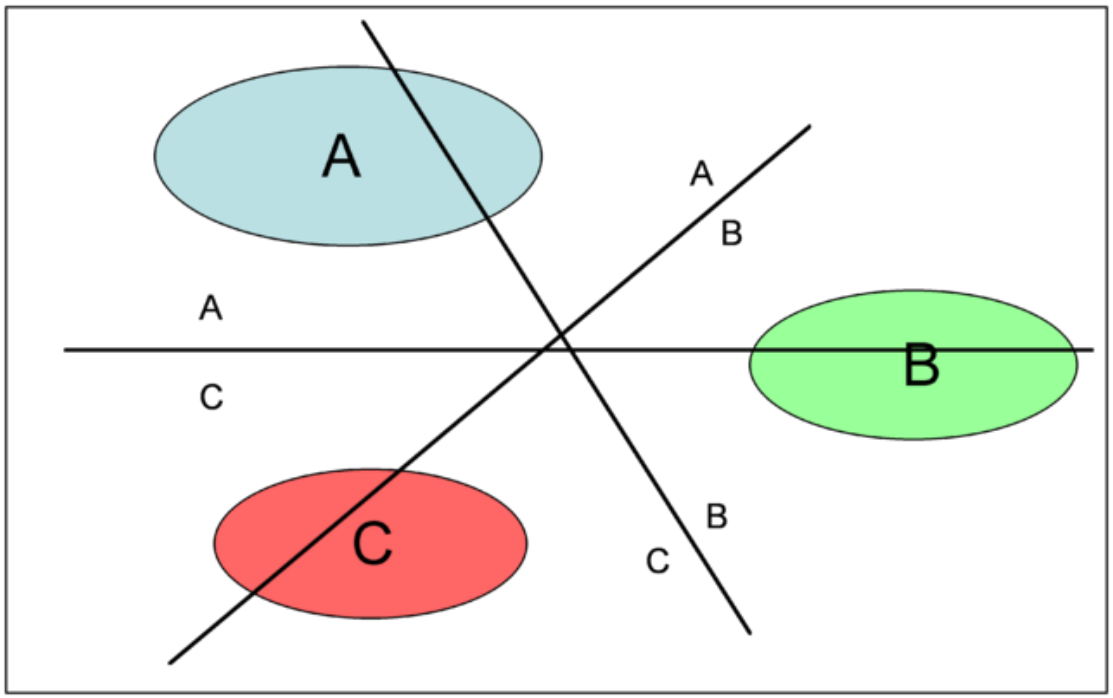
# Multiclass problems

- SVMs are a great for binary classification problems.
- What do we do, however, if we have three (or more) classes?
- This is a Multiclass SVM, and there are two main approaches.
- One Versus Rest (OVR):



# Multiclass problems (continued)

- OVR classifiers work well if classes are clustered together.
- But, they can leave portions of the space classified as multiple or no classes.
- The other strategy is One Versus One (OVR):



# Multiclass problems (continued)

- In sklearn: sklearn.svm.SVC
- `SVC` handles multiclass problems according to the `decision_function_shape` argument.
- It defaults to the One Versus Rest (OVR) since it is the easiest to interpret.
- Advice: Scikit-learn usually has reasonable default and I recommend starting from these and to override defaults only after you have a better understanding of the problem.

# Discrete probability distributions

To specify a discrete random variable, we need a sample space and a probability mass function:

- Sample space $\Omega$: Possible states $x$ of the random variable $X$ (outcomes of the experiment, output of the system, measurement).
- Discrete random variables have a finite number of states.
- Events: Possible combinations of states (subsets of $\Omega$)
- Probability mass function $P(X = x)$: A function which tells us how likely each possible outcome is:

$$P(X = x) = P_X(x) = P(x)$$

$$P(x) \geq 0 \text{ for each } x$$

$$\sum_{x \in \Omega} P(x) = 1$$

$$P(A) = P(x \in A) = \sum_{x \in A} P(X = x)$$

# Discrete probability distributions (continued)

- Conditional probability: Recalculated probability of event A after someone tells you that event B happened:

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

- Bayes Rule:

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

# Expectation and covariance of multivariate distributions

- Conditional distributions are just distributions which have a (conditional) mean or variance.
- Note: $E(X|Y) = f(Y)$ – *If I tell you what $Y$ is, what is the average value of $X$?*
- Covariance is the expected value of the product of fluctuations:

$$\begin{aligned} \mathrm{Cov}(X, Y) &= E\left((X - E(X))(Y - E(Y))\right) \\ &= E(XY) - E(X)E(Y) \\ \mathrm{Var}(X) &= \mathrm{Cov}(X, X) \end{aligned}$$

# Multivariate distributions: the same, but different

- Multivariate distributions are the same as bivariate distributions – just with more dimensions.
- $\mathbf{X}, \mathbf{x}$ are vector valued.
- Mean: $E(\mathbf{X}) = \sum_{\mathrm{x}} \mathbf{x} P(\mathbf{x})$
- Covariance matrix:

$$\mathrm{Cov}(X_i, X_j) = E(X_i X_j) - E(X_i)E(X_j)$$
$$\mathrm{Cov}(\mathbf{X}) = E(\mathbf{X}\mathbf{X}^\top) - E(\mathbf{X})E(\mathbf{X})^\top$$

- Conditional and marginal distributions: Can define and calculate any (multi or single-dimensional) marginals or conditional distributions we need: $P(X_1)$, $P(X_1, X_2)$, $P(X_1, X_2, X_3|X_4)$, etc..

# Mean, variance, and conditioning of continuous RVs

- Mostly the same as the discrete case, just with sums replaced by integrals.
- Mean: $E(X) = \int_x x p(x) dx$
- Variance: $\mathrm{Var}(X) = E(X^2) - E(X)^2$
- Conditioning: If $X$ has pdf $p(x)$, then $X|(X \in A)$ has pdf:

$$p_{X|A}(x) = \frac{p(x)}{P(A)} = \frac{p(x)}{\int_{x \in A} p(x) dx}$$

# The univariate Gaussian (normal) distribution

- The Univariate Gaussian:

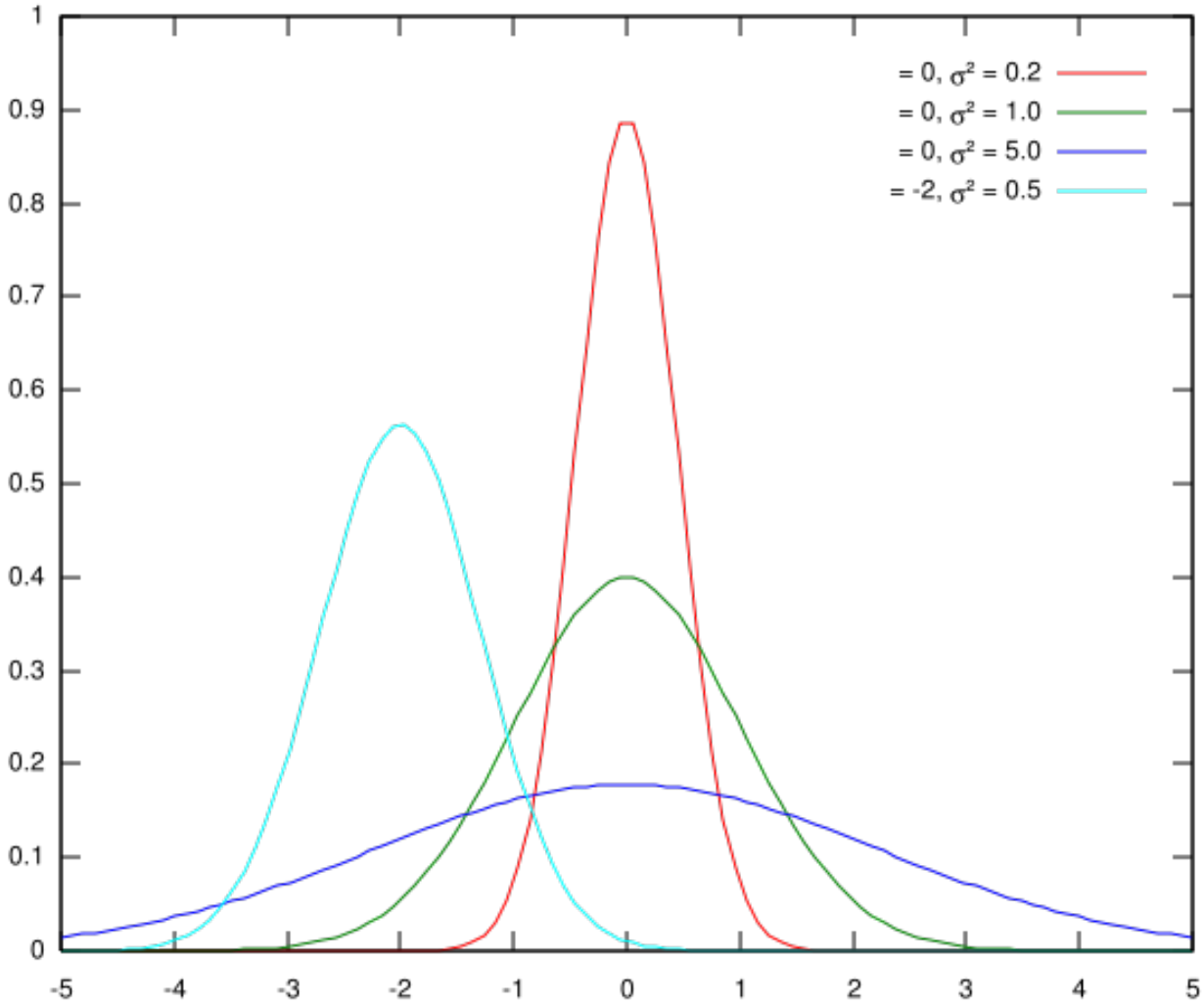
$$t \sim \mathcal{N}(\mu, \sigma^2)$$

$$p(t|\mu, \sigma^2) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{1}{2}\left(\frac{t-\mu}{\sigma}\right)^2\right)$$

- The Gaussian has mean $\mu$ and variance $\sigma^2$ and precision $\beta = 1/\sigma^2$
- What are the mode and the median of the Gaussian?

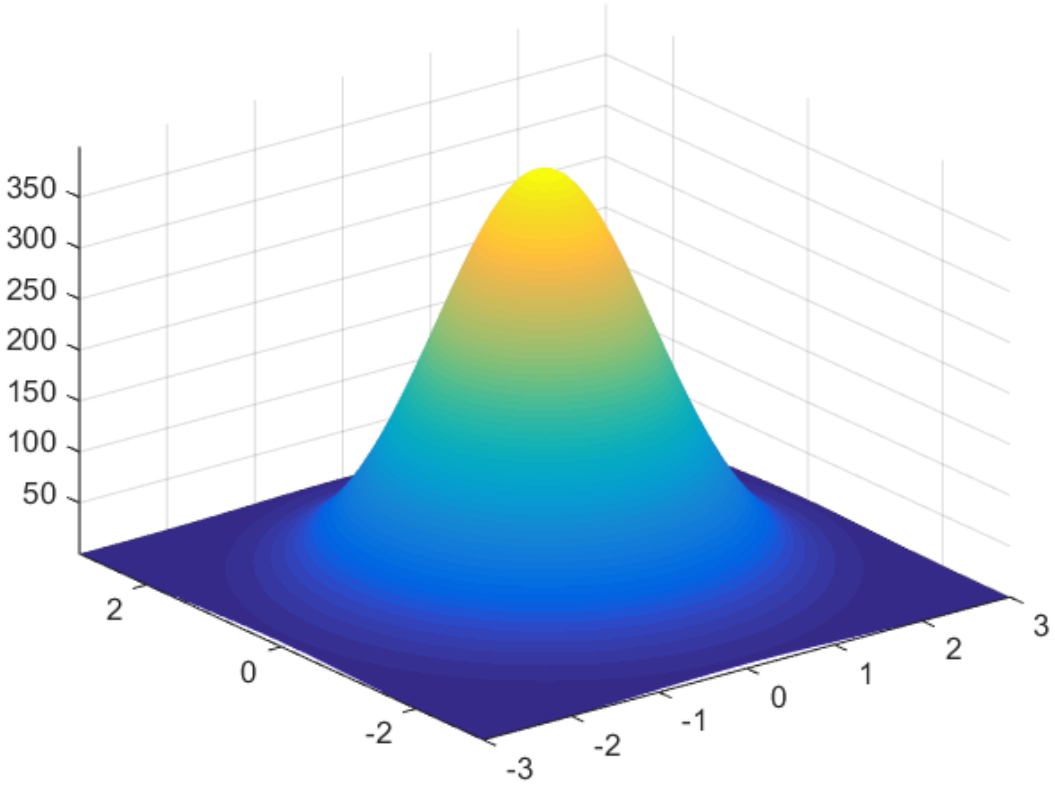
# Gaussian distributions

- As they say, a picture is worth a thousand words:



# The multivariate Gaussian

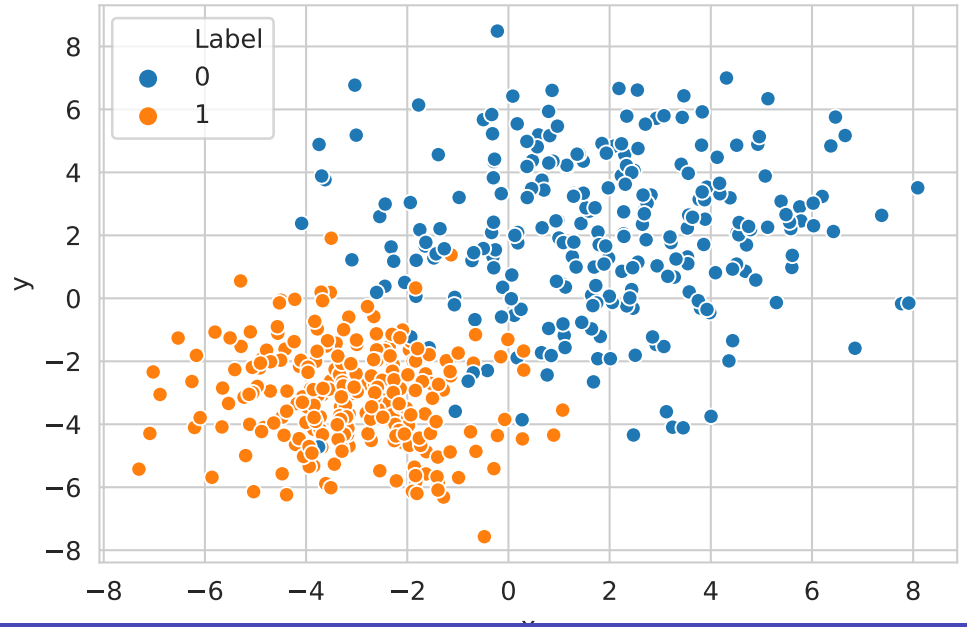
$$f(\mathbf{x}; \boldsymbol{\mu}, \Sigma) = \frac{1}{\sqrt{(2\pi)^k |\Sigma|}} \exp\left(-\frac{1}{2}(\mathbf{x} - \boldsymbol{\mu})^\top \Sigma^{-1} (\mathbf{x} - \boldsymbol{\mu})\right)$$



# Why is ANY of this important?

- Well, given data the only thing we can do is estimate $P(\mathbf{x}|C = c)$ and $P(\mathbf{x})$ – the empirical data distribution.
- Plus $P(C = c)$ – the class priors.
- But then Bayes Rule tells us the posterior:

$$P(C = c|\mathbf{x}) = \frac{P(\mathbf{x}|C = c)P(C = c)}{P(\mathbf{x})}$$



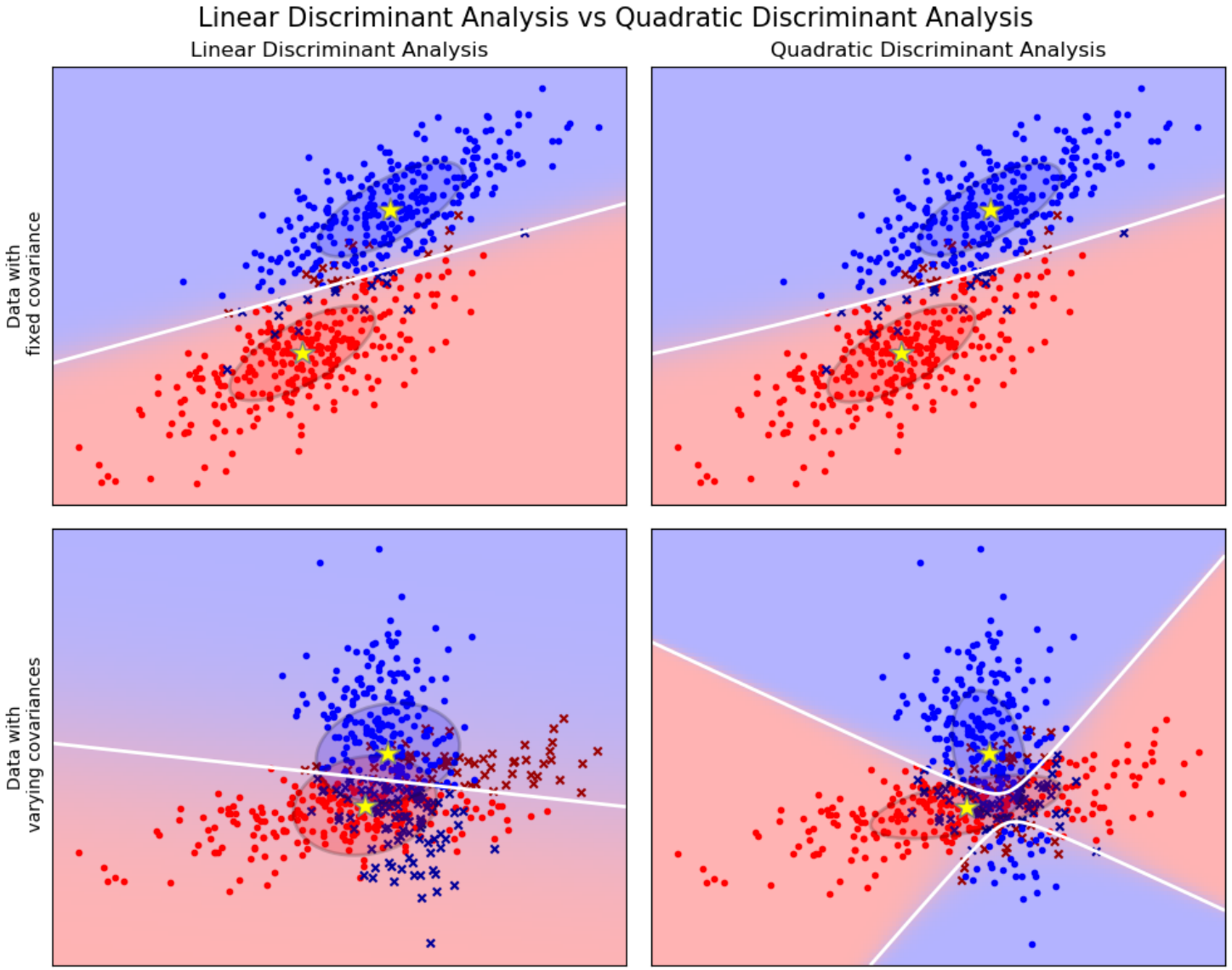
# The Math

- Both the Linear Discriminant Classifier (LDC) and Quadratic Discriminant Classifier (QDA) can be derived from class-conditional distribution of the data for each class.
- More specifically, for Bayesian discriminant analysis the class-conditional data distributions are modeled as multivariate Gaussian distributions:

$$f(\mathbf{x}|C = c; \boldsymbol{\mu}_c, \Sigma_c) = \frac{1}{\sqrt{(2\pi)^k |\Sigma_c|}} \exp(-\frac{1}{2}(\mathbf{x} - \boldsymbol{\mu}_c)^\top \Sigma_c^{-1}(\mathbf{x} - \boldsymbol{\mu}_c))$$

- What's the difference between quadratic and linear?
  - Linear: assume all $\Sigma_c$ are equal.
  - Quadratic: estimate $\Sigma_c$ independently.

# The bit picture



# Bayesian Discriminant Analysis: Analysis

- Bayesian discriminants have a number of advantages:
  - The naturally handle multiclass classification problems – just estimate multiple class-dependent data distributions.
  - Parameter estimation is relatively easy: just a per-class empirical mean and a single (or $n$) covariance matrices.
  - They are intuitive since they directly output a probability distribution over classes – something SVMs and KNN classifiers do not easily do.
- They do have a number of disadvantages:
  - Not all data is Gaussian, and this assumption can lead to poor performance.
  - Can be sensitive to unbalanced problems (due to poor estimates of $\Sigma_c$ and $\boldsymbol{\mu}_c$).
- In sklearn:
  - `sklearn.discriminant_analysis.LinearDiscriminantAnalysis`
  - `sklearn.discriminant_analysis.QuadraticDiscriminantAnalysis`

# Evaluating Classifiers: Balanced Problems

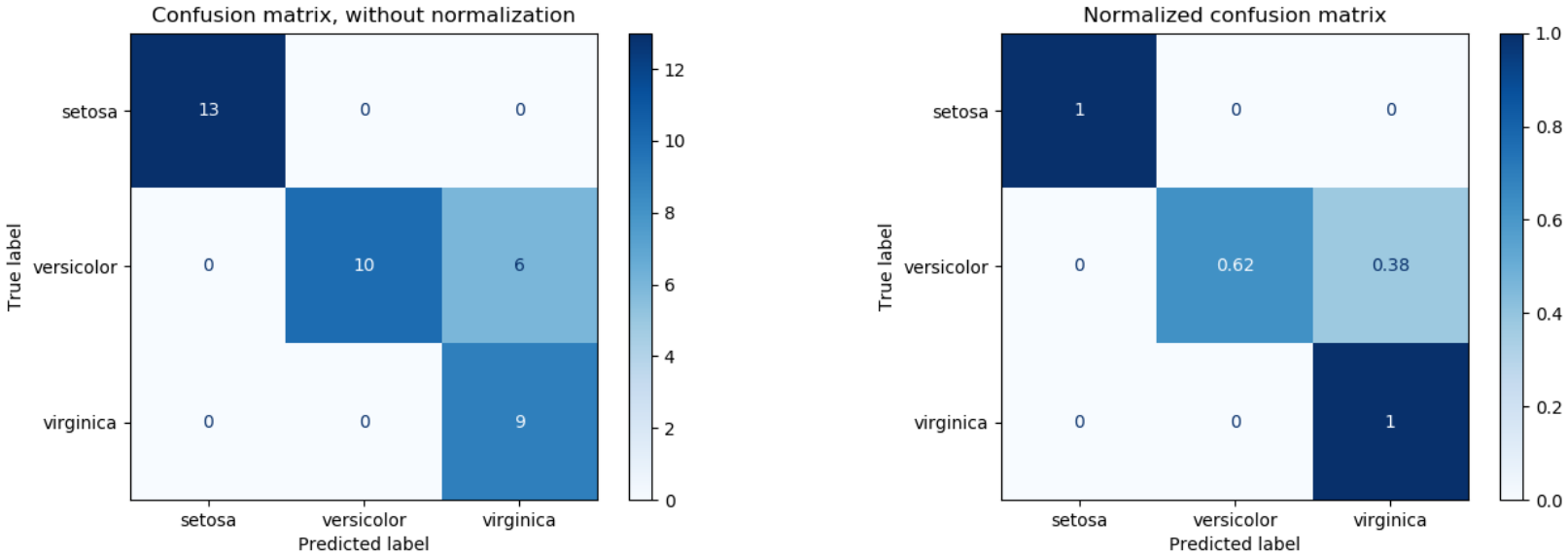
- How to we measure the performance of trained classifiers?
- If you have a balanced classification problem (like most we will see in the labs), use the classifier accuracy:

$$\text{accuracy} = \frac{\#\text{ correctly classified test samples}}{\#\text{ test samples}}$$

- Why might this be a bad metric if the problem is unbalanced?

# Evaluating Classifiers: Confusion Matrices

- A good tool to get an overview of the types of errors a classifier is making is the confusion matrix:



# Evaluating Classifiers: Unbalanced Problems

- We begin by dissecting the classifications:
  - True positives (TP): we predicted yes, and sample does belong to class.
  - True negatives (TN): we predicted no, and the sample does not belong to class.
  - False positives (FP): we predicted yes, but the sample does not belong to class – also known as a Type I error.
  - False negatives (FN): we predicted no, but sample does belong to the class – also known as a Type II error.
- We can then define some useful metrics for unbalanced problems:

$$\begin{aligned} \text{Precision(c)} &= \frac{\text{TP}}{\text{TP} + \text{FP}} \\ \text{Recall(c)} &= \frac{\text{TP}}{\text{TP} + \text{FN}} \\ \text{F1(c)} &= 2\frac{\text{Precision(c)} * \text{Recall(c)}}{\text{Precision(c)} + \text{Recall(c)}} \end{aligned}$$

# Evaluating Classifiers: Analysis

- The evaluation metric to use is usually clear given the type of problem (classification, regression, retrieval) and the distribution of training data (balanced, unbalanced).
- Getting a reliable estimate of a classifier can be tricky as it clearly depends on your train/test split.
- We will return to this tomorrow when we look at the model selection problem.
- In sklearn: `sklearn.metrics`

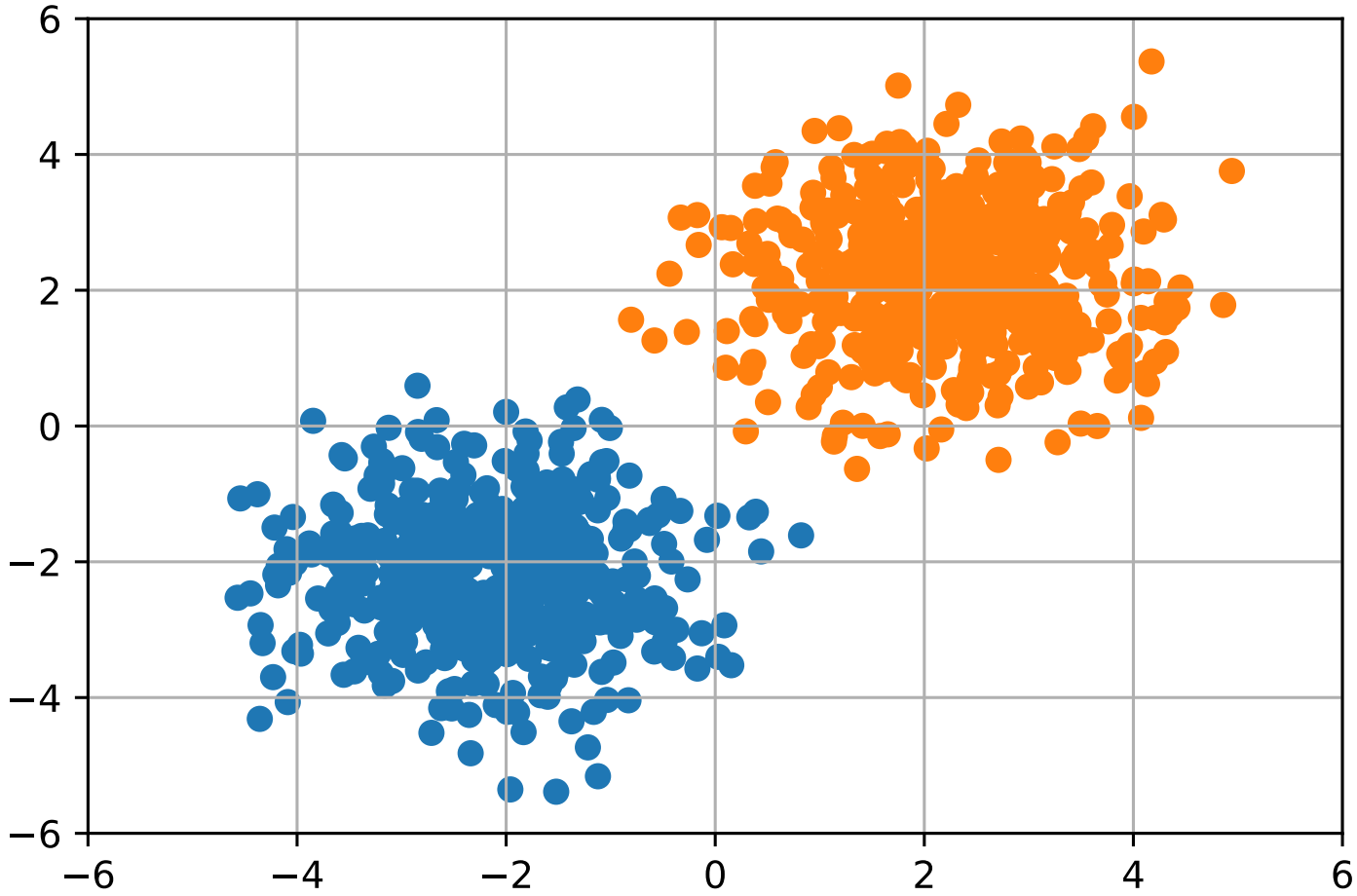
# Unsupervised Learning

# Why unsupervised learning?

- Unsupervised learning is learning something from data in the absence of labels or target values.
- It is an active area of current research.
- Note that it is often used even in the context of supervised learning problems:
  - For dimensionality reduction: often not all of the input features are needed (or even desirable), and unsupervised techniques can be to reduce the input dimensionality of a problem.
  - For visualization: to get a sense of the data, we may want to visualize it in some meaningful way – this often uses dimensionality reduction to reduce high dimensional features to just two or three.
- In this part of the lecture we will see three useful unsupervised techniques: Principal Component Analysis (PCA), Clustering, and t-Distributed Stochastic Neighbor Embedding (t-SNE).

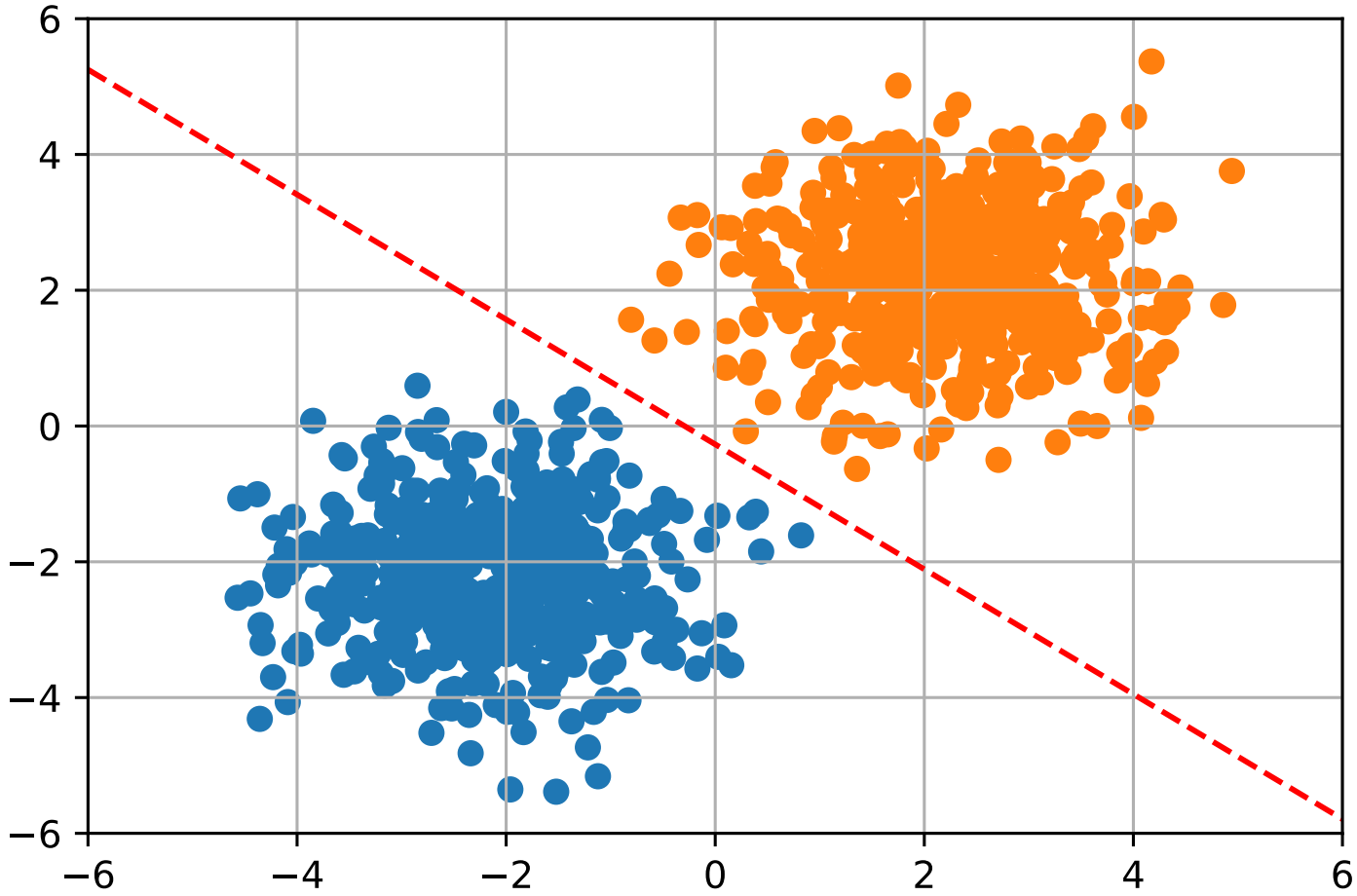
# PCA: Motivation

- Say we have a classification problem with data distributed like this:



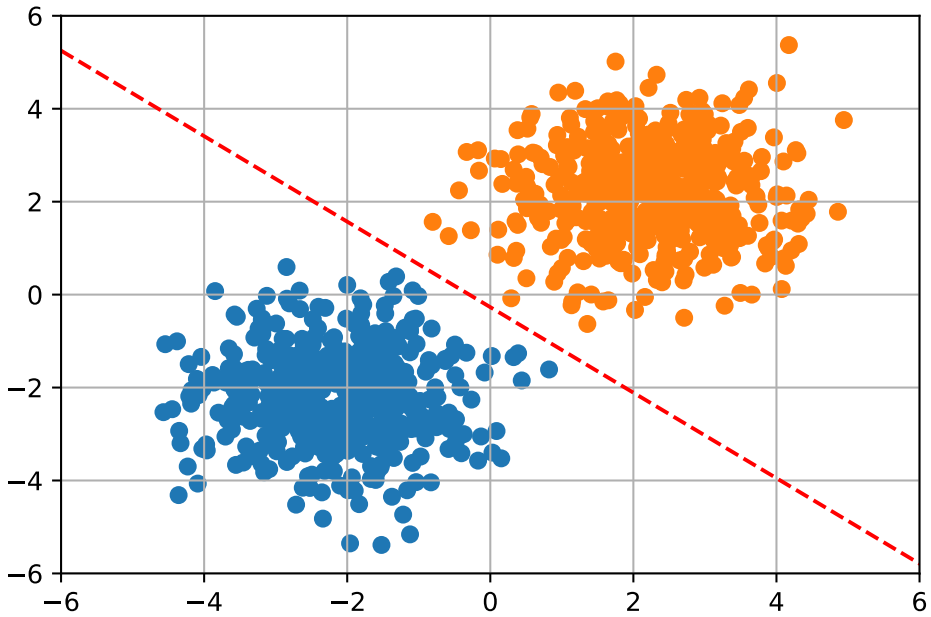
# PCA: Motivation (continued)

- We know how to train a classifier for linearly separable classes:



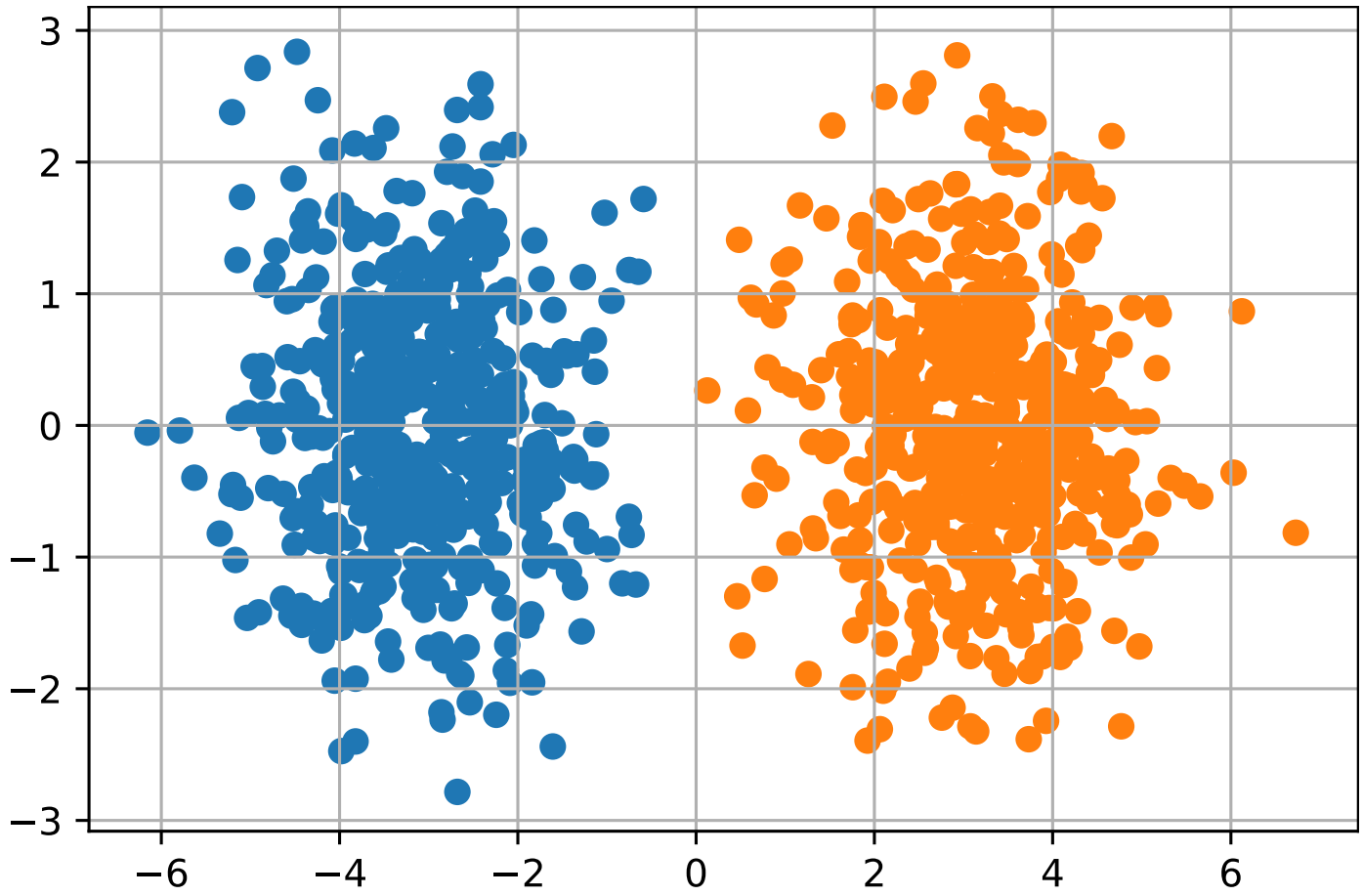
# PCA: Motivation (continued)

- But, let's take a closer look at this situation.
- In the original feature space we need both features to define the discriminant dividing the two classes.
- But, maybe if we could somehow transform this space so that the features are decorrelated...



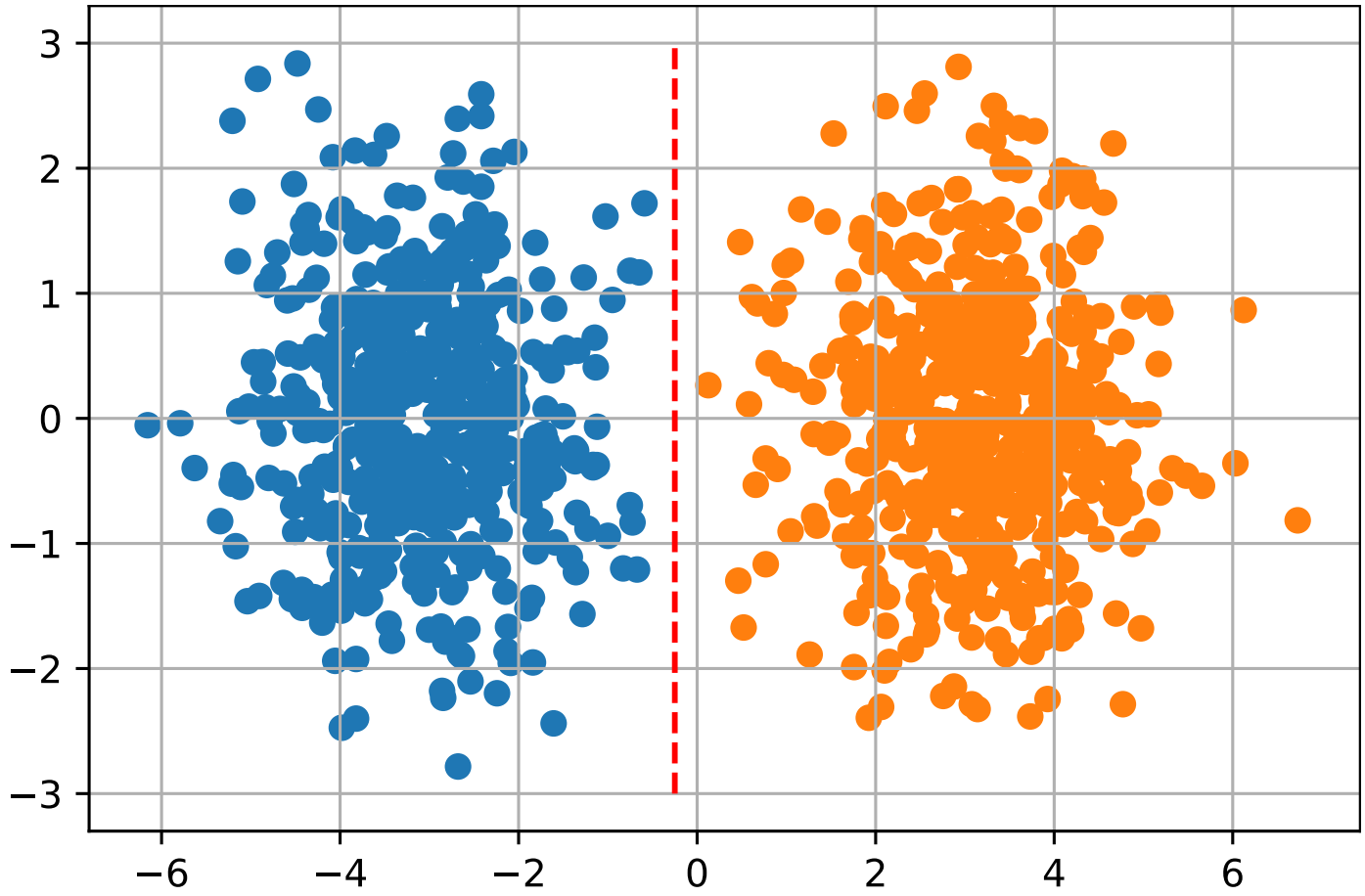
# PCA: Motivation (continued)

- What if we rotate the feature space so that the principal data directions are aligned with the axes?



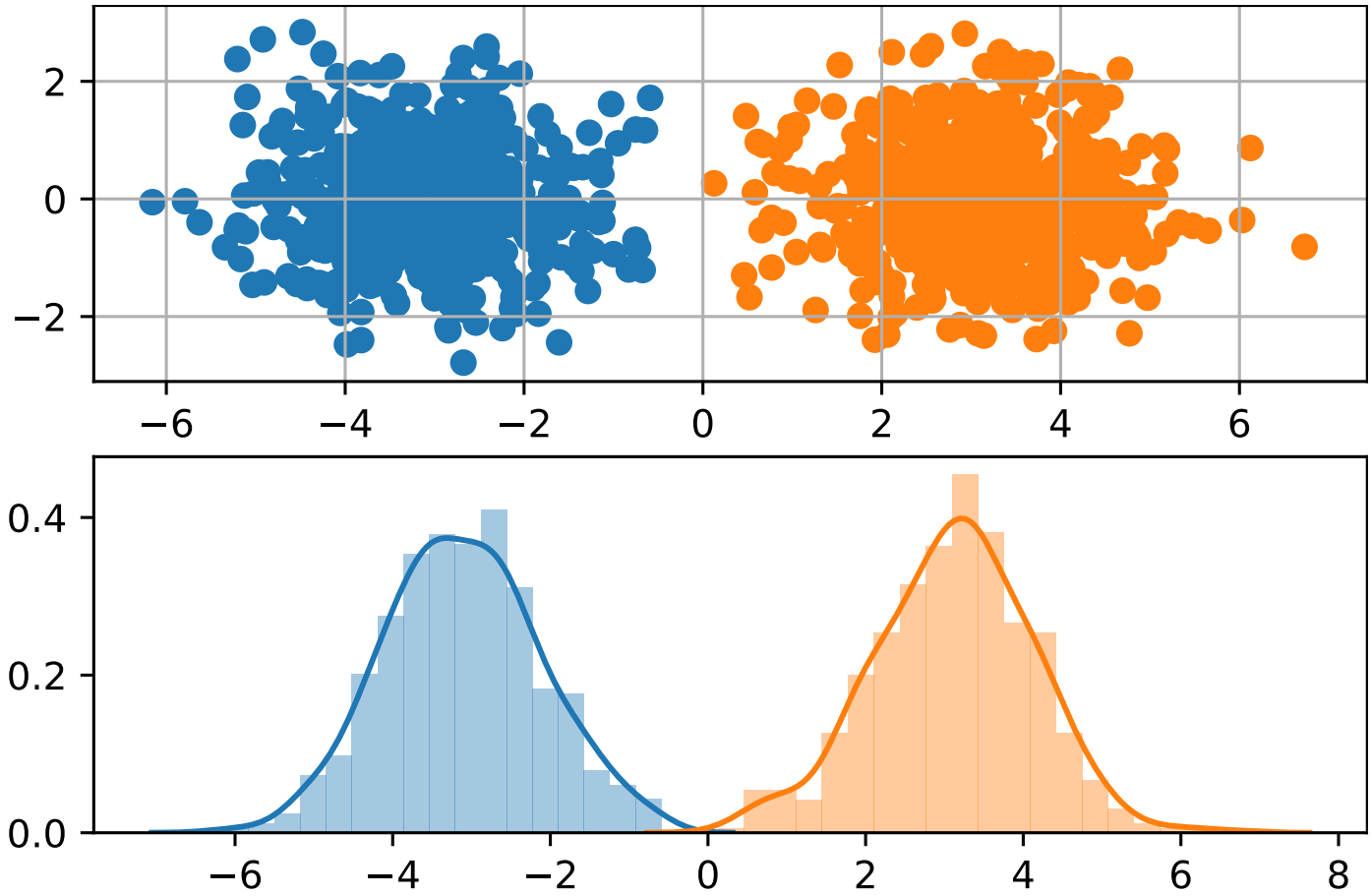
# PCA: Motivation (continued)

- Well now we have an easier problem to solve, since the discriminant function needs only one feature:



# PCA: Motivation (continued)

- We have turned a two-dimensional problem into a one-dimensional proble.



# PCA: The Math

- How do we find these principal directions of the data in feature space?
- How do we even define what a principal direction is?
- Well, one natural way to define it is iteratively:
  1. The first principal component is the *direction in space along which projections have the largest variance*.
  2. The second principal component is the *direction which maximizes variance among all directions orthogonal to the first*.
  3. etc.
     - This defines up to $D$ principal components, where $D$ is the original feature dimensionality.
     - If we project onto all principal components we are rotating the original features so that in the new axes the features are decorrelated.

# PCA: The Math (continued)

- If we project onto the first $d < D$ principal components, we are performing dimensionality reduction.
- How do we do any of this? We use the eigenvectors of the data covariance matrix $\mathbf{\Sigma}$.
- Recall the multivariate Gaussian:

$$f(\mathbf{x}; \boldsymbol{\mu}, \Sigma) = \frac{1}{\sqrt{(2\pi)^k |\Sigma|}} \exp(-\frac{1}{2}(\mathbf{x} - \boldsymbol{\mu})^\top \Sigma^{-1} (\mathbf{x} - \boldsymbol{\mu}))$$

- $\mathbf{\Sigma}$ this defines the hyperellipsoidal shape of the Gaussian.
- If $\Sigma$ is diagonal, the features are decorrelated, so if we diagonalize it we are decorrelating the features.
- We do this by finding the eigenvectors of (an estimate of) $\Sigma$ – the data covariance matrix.

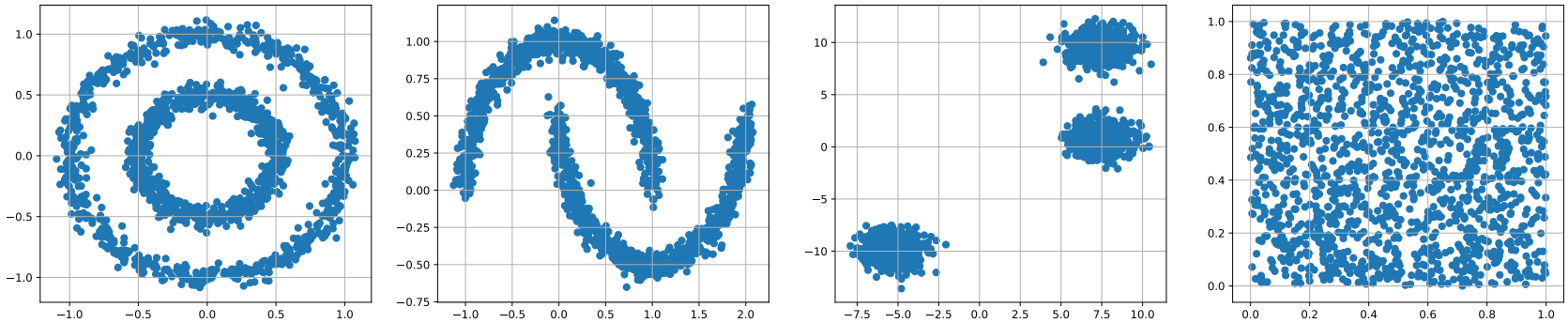
# PCA: Analysis

- Principal Component Analysis is used for many purposes:
  - Dimensionality reduction: in high-dimensional features spaces with limited data, PCA can help reduce the problem to a simpler one in fewer, decorrelated dimensions.
  - Feature decorrelation: some models assume feature decorrelation (or are simpler to solve with decorrelated features).
  - Visualization: looking at data projected down to the first two principal dimensions can tell you a lot about the problem.
- In sklearn: sklearn.decomposition.PCA

```
from sklearn.decomposition import PCA
model = PCA()
model.fit(xs)
new_xs = model.transform(xs)
```

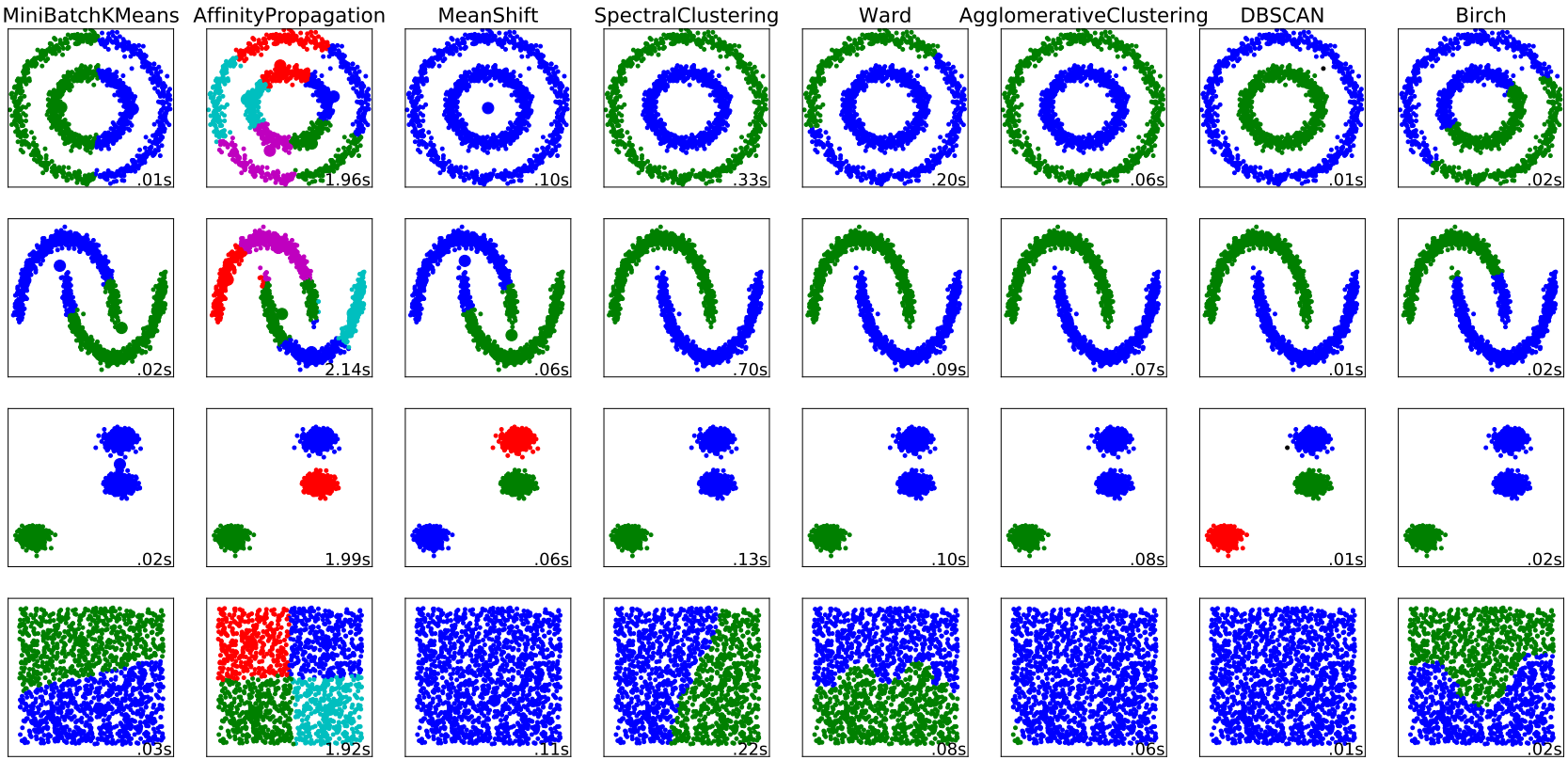
# Clustering: Motivation (continued)

- What if you have data with structure, but you don't know what this structure is or have any a priori model for it?



# Clustering: Motivation (continued)

- Clustering refers to techniques that learn groups of related data points in feature space:



# Clustering: The k-Means Algorithm

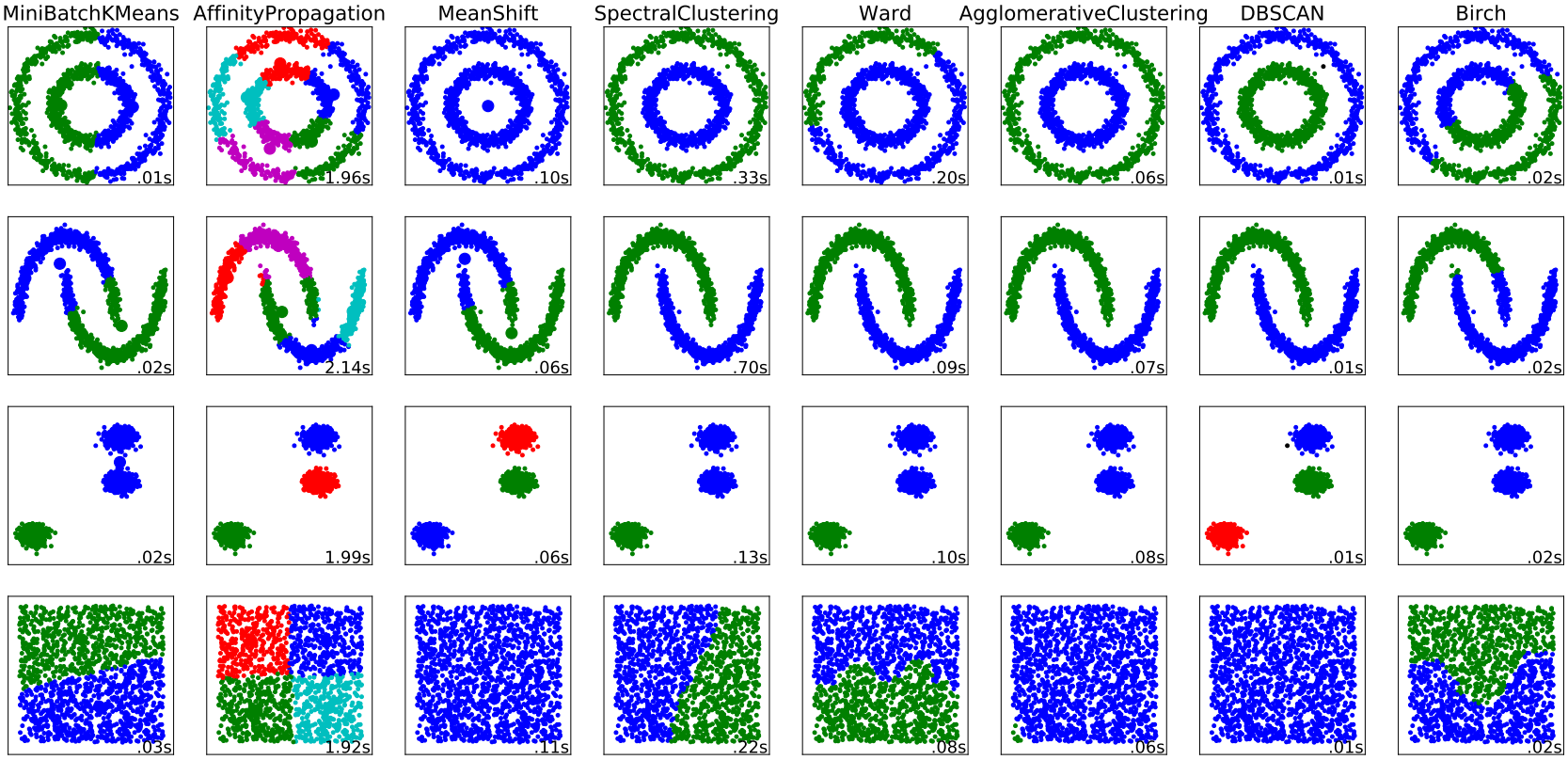
- k-Means is a very simple algorithm that associates each data point with one of $k$ means or centroids in the data space:
- Given an initial set of $k$ means $\mathbf{m}_1^1, \ldots, \mathbf{m}_k^1$, the alternates between two steps:
  1. Assignment: $S_i^t = \{ \mathbf{x}_p \mid ||\mathbf{x}_p - \mathbf{m}_i^t|| \leq ||\mathbf{x}_p - \mathbf{m}_j^t|| \forall j \}$.
  2. Update: $\mathbf{m}_i^{t+1} = \frac{1}{|S_i^t|} \sum_{\mathrm{x}_i \in S_i^t} \mathbf{x}_j$.
- The algorithm converges when the cluster assignments no longer change.
- Note: this algorithm is not guaranteed to find the global optimum clusters.

# Clustering: Analysis

- Clustering is a robust technique (in that it never fails).
- Its usefulness depends on the data distribution and which type of clustering you perform.
- In sklearn:
  - `cluster.AffinityPropagation`
  - `cluster.AgglomerativeClustering`
  - `cluster.Birch`
  - `cluster.DBSCAN`
  - `cluster.KMeans`
  - `cluster.MeanShift`
  - `cluster.SpectralClustering`
- See `sklearn.cluster` for more details.

# Clustering: Motivation (continued)

- It's useful to revisit this plot:



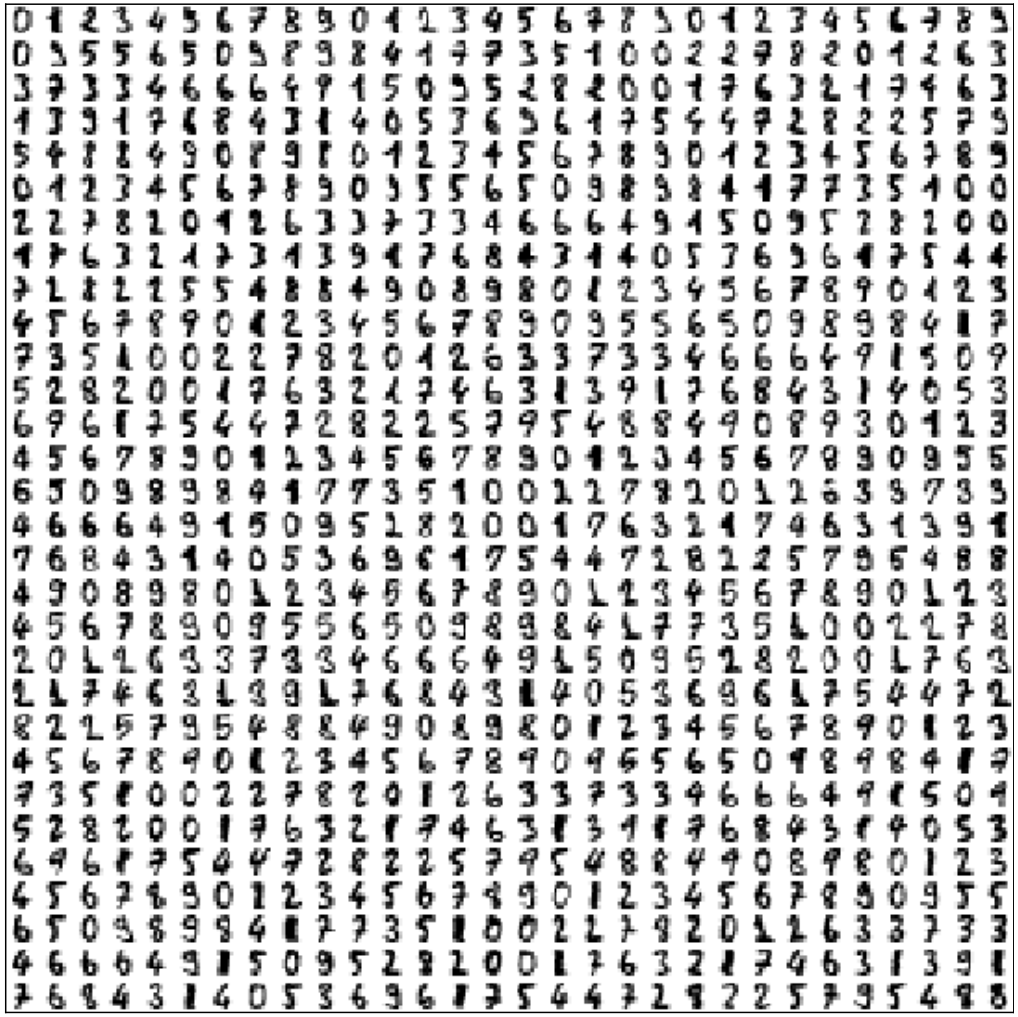
# t-SNE: Motivation

- Sometimes the structure of data in high dimensional data is obscured by its very high-dimensional nature.
- Questions like this are related to data design: you often need to discover if the features you have collected are at all related to the questions you want to ask about it.
- In this last section we will look at a very powerful algorithm for uncovering latent structure in high-dimensional data.
- This approach is called the t-Distributed Stochastic Neighbor Embedding (t-SNE) algorith,
- It works by reducing dimensionality (usually to just two dimensions) in a way that preserves the distances between samples.

# t-SNE: Motivation (continued)

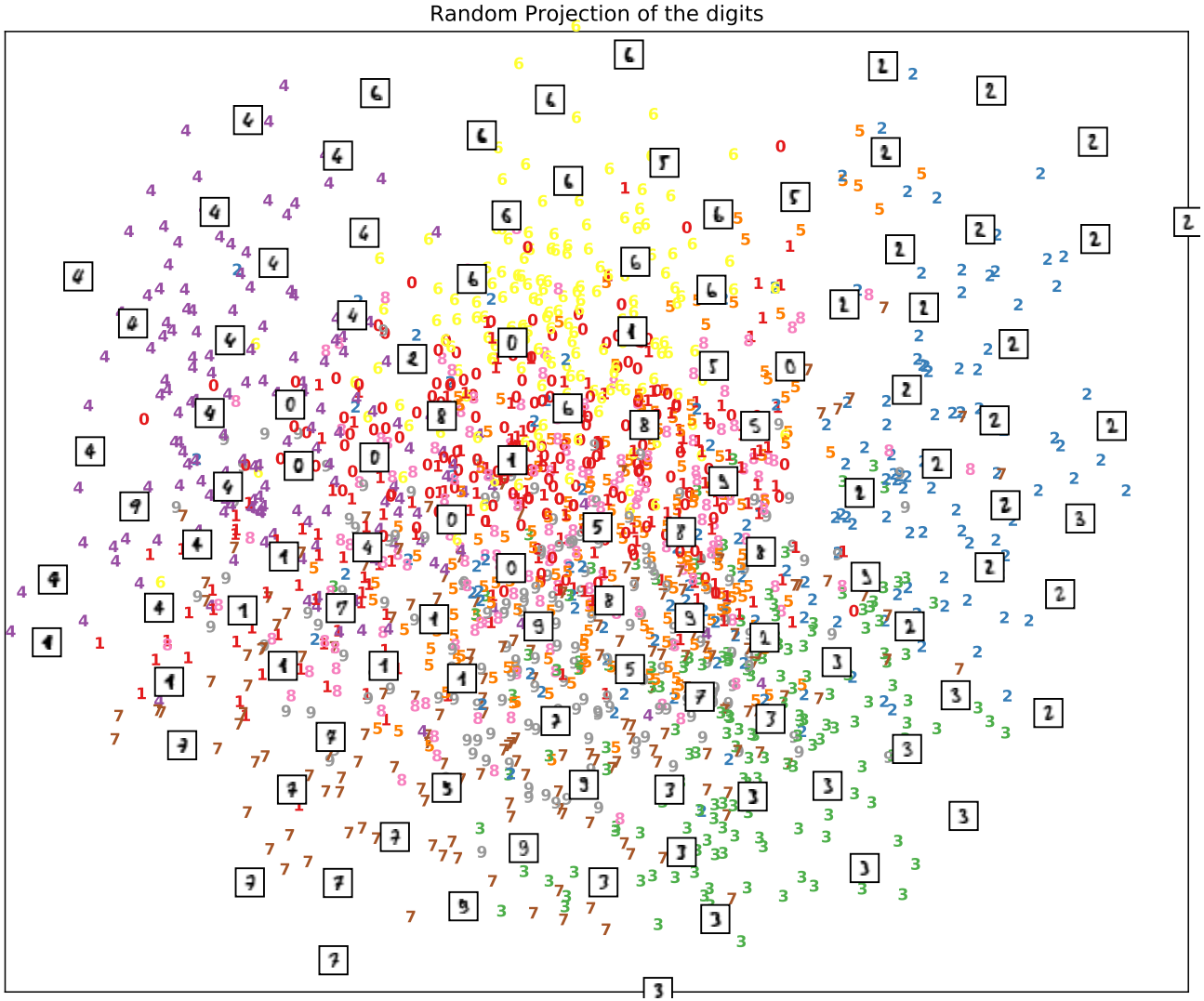
- Even in just 64 dimensions data can be complex:

A selection from the 64-dimensional digits dataset



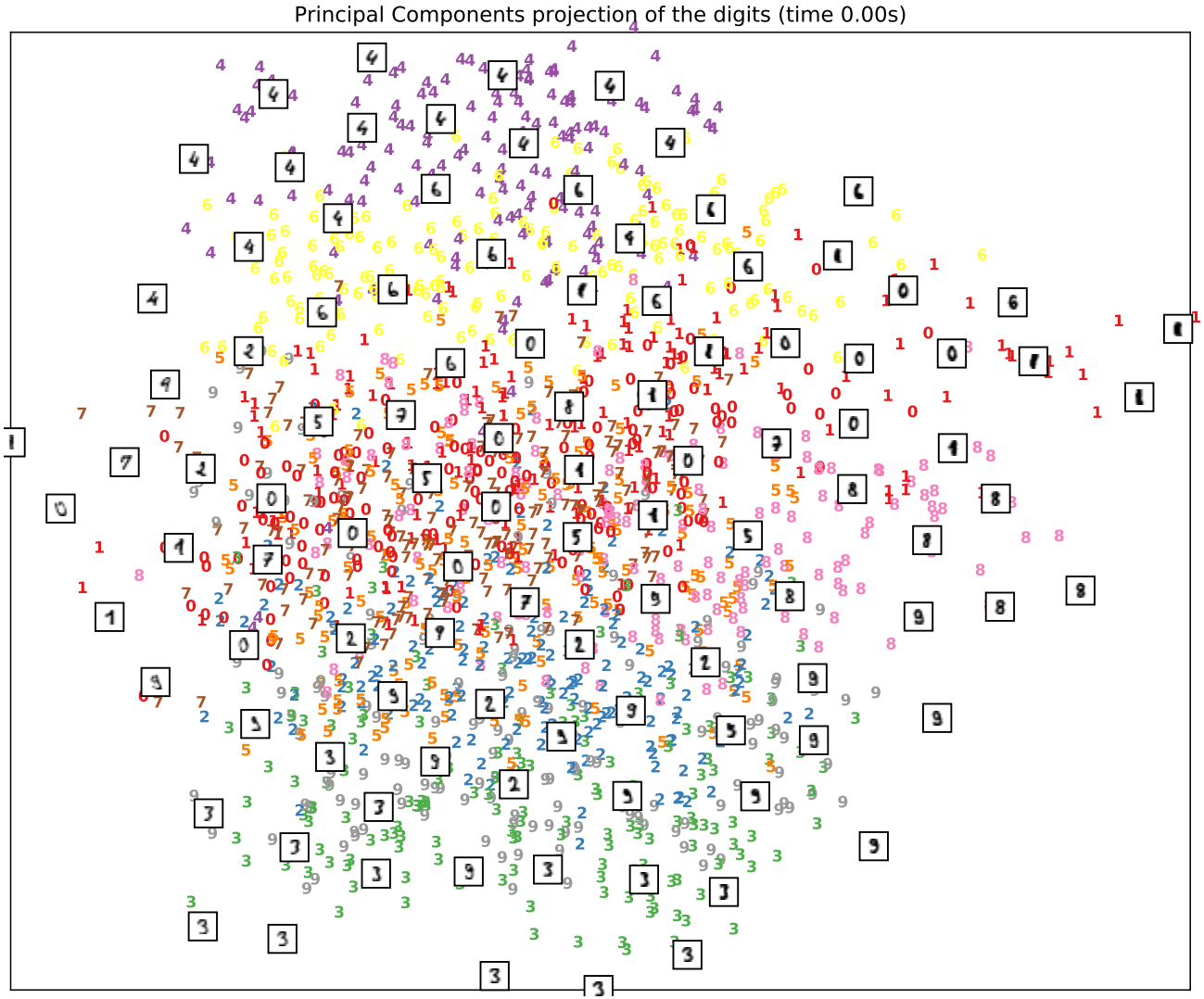
# t-SNE: Motivation (continued)

- We can try random projection to two dimensions:



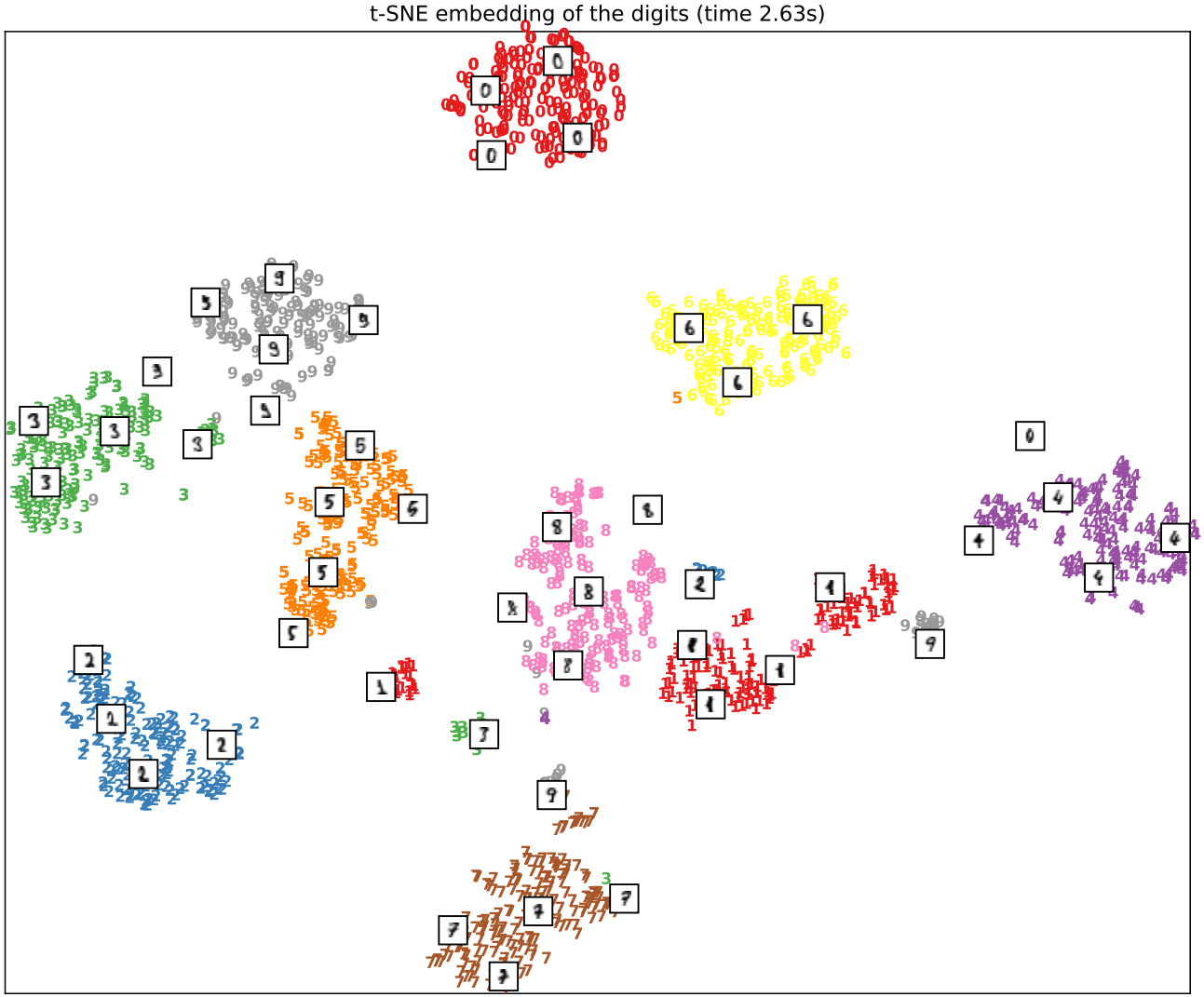
# t-SNE: Motivation (continued)

- Or projection onto first two principal components:



# t-SNE: Motivation (continued)

- What we want is something like:



# t-SNE: The Math

- t-distributed Stochastic Neighbor Embedding (t-SNE) is a algorithm for visualization.
- It is a nonlinear dimensionality reduction for embedding high-dimensional data in a low-dimensional space of two or three dimensions.
- It models each high-dimensional point by a low-dimensional point so that that similar objects map to nearby points and dissimilar objects are to distant points with high probability.
- The short version: points that are close in the high-dimensional space are close in the mapped space.

# T-SNE: Analysis

- t-SNE is a very powerful tool for visualization and dimensionality reduction.
- It can give you visual feedback that indicates if, on average, the high-dimensional features represent your problem well (or not).
- Note that t-SNE visualization can sometimes be misleading in that the associations it learns are not always present in the original space.
- In sklearn: `sklearn.manifold.TSNE`

# Reflections

# Supervised learning

- Given data, the world is full of supervised learning problems.
- There are robust and mature techniques for addressing many of these.
- Here we have only seem a few examples of the variety of models that exist.
- Those we have seen are some of the simplest, and therefore I strongly urge you to try them before looking to more complex models.
- Even these simple models, explicitly or implicitly (via kernel machine) embedded in a high-dimensional space can also scale in complexity.

# Unsupervised learning

- Of course, there is far more unlabeled data than labeled data in the world.
- Unsupervised learning techniques can help us visualize and understand data distributions in feature space.
- Clustering is useful to uncover latent structure that is often hidden in high-dimensional data.
- Principal Component Analysis (PCA) allows us to reduce dimensionality in ways that preserve variance in the high-dimensional feature space.
- Tomorrow we will start talking about the bias/variance tradeoff and the problem of overfitting.
- These unsupervised tools are often helpful for understanding when models are overfitting and often help avoid overfitting outright.

# Supervised Learning Lab

- The laboratory notebook for today:

`http://bit.ly/DTwin-ML4`