DIGITAL TWIN AI and Machine Learning: Supervised Learning I

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

Dipartimento di Ingegneria dell'Informazione
Università degli Studi di Firenze

04 November 2020

Outline

Introduction

Overview

- In this lecture we will (finally) look at wide variety of different machine learning models.
- We start with supervised problems as they are:
  - The most common: classification and regression problems are everywhere.
  - The most well-studied: optimal decision has a history of at least 100 years, with useful theory going back to the 17th century.
- The easiest problems to confront: if you have labeled data, there is a vast number of reliable models you can easily apply.
- Note: the models I will talk about here are necessarily just a small sample of what is available.

Supervised Regression

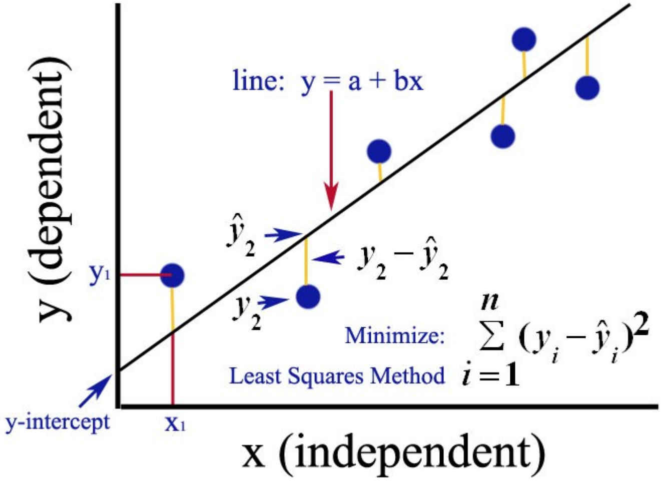
Ordinary Least Squares: The Big Math

- The Data: matched pairs $\mathbf{x}, y)$ of features $\mathbf{x}$ and targets y.
- The Model:

$$\begin{aligned} \hat{y} \equiv f(\mathbf{x}) &= \mathbf{w}^T \mathbf{x} + b \\ \mathcal{L}(D; \mathbf{w}, b) &= \sum_{(\mathrm{x}, y) \in D} ||y - f(\mathbf{x}, b)||_2 \end{aligned}$$

- The Parameters: Vector of weights $\mathbf{w}$ and scalar bias b.
- The Hyperparameters: None to speak of.
- Fitting: Efficient, exact, and closed-form solution using psuedo-inverse.

Ordinary Least Squares: The Big Picture

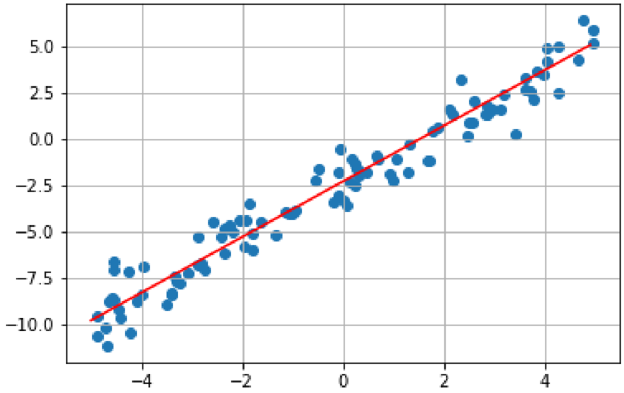


Ordinary Least Squares: Discussion

- Least Squares is simple and effective.
- Having one learned weight for each feature makes it robust to feature scaling.
- Assumes that there is a linear relation between features and target.
- Also assumes the features are independent and thus decorrelated.
- If there are approximate linear relationships between features, the matrix to be inverted can become singular.
- In sklearn: `sklearn.linear_model.LinearRegression`

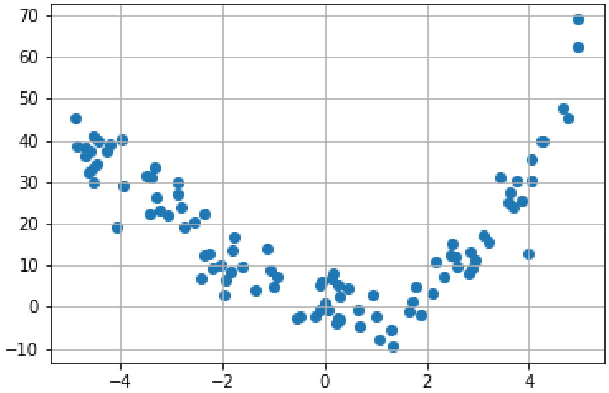
Explicit feature mapping

- We know what to do with this type of estimation problem:



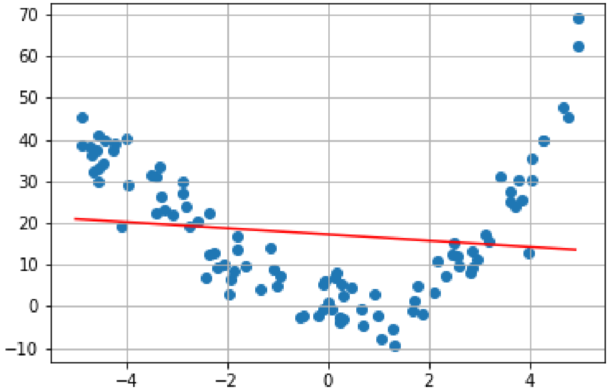
Explicit feature mapping

- What do we do if we have data like this:



Explicit feature mapping

- What do we do if we have data like this:



Explicit feature mapping: How it works

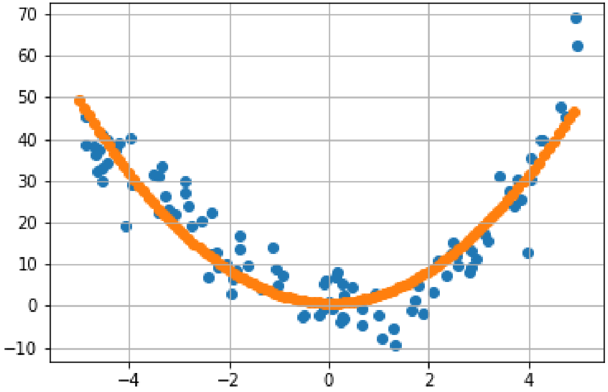
- The general idea is to map our original features into a higher dimensional space.
- For example, we can map linear features (one-dimensional) into a space with a polynomial basis:

$$\begin{aligned} e(x) &= [x\ x^2\ x^3\ \ldots\ x^k]^T \\ f(e(x)) &= \mathbf{w}^T e(x) + b \\ &= w_1 x + w_2 x^2 + \cdots + w_k x^k + b \end{aligned}$$

- So, by fitting a linear model in k features, we are really fitting a polynomial to the data.

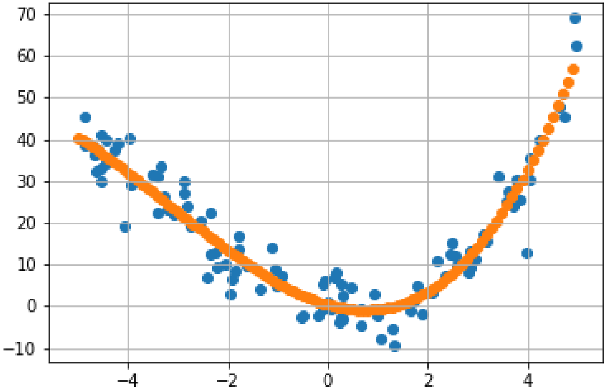
Explicit feature mapping

- Using a degree 2 polynomial embedding:



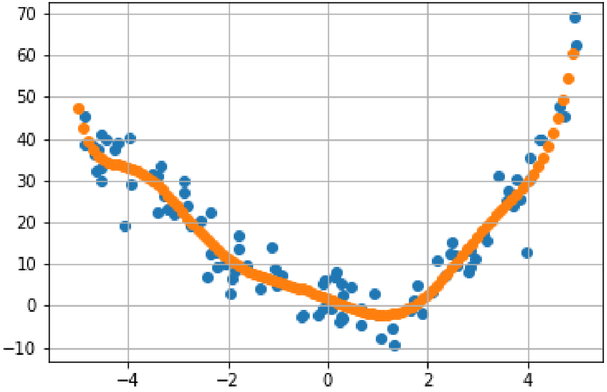
Explicit feature mapping

- Using a degree 3 polynomial embedding (which is correct):



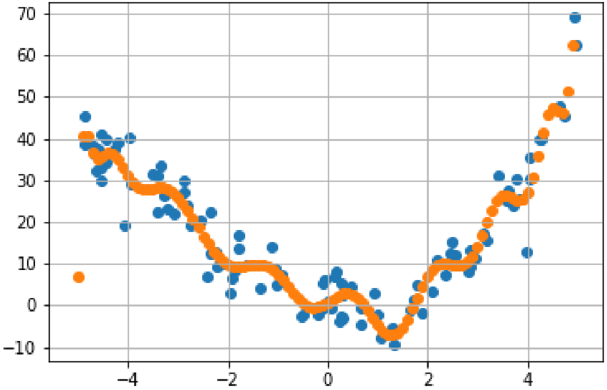
Explicit feature mapping

- Degree 10 – more complex isn't always better:



Explicit feature mapping

- Degree 20 – now we're just fitting noise:



Explicit feature mapping: Discussion

- Mapping features into a higher dimensional space is a powerful tool.
- It allows us to use simple tools (like good old linear regression) to fit non-linear functions to data.
- You must be careful, though, to not increase the power of the representation too much.
- At some point the model will begin capturing the noise and will never generalize.
- In sklearn: `sklearn.preprocessing.PolynomialFeatures`

Supervised Classification

Classification problems

- Classification problems are usually formulated in terms of a discriminant function and a decision rule.
- The discriminant function tells us something about the similarity of an input to all classes.
- The decision rule tells us how to act on this information.
- Training data: matched pairs $(\mathbf{x}, c_{\mathbf{x}})$ of features $\mathbf{x}$ and target labels $c_{\mathbf{x}}$.

KNN: Overview

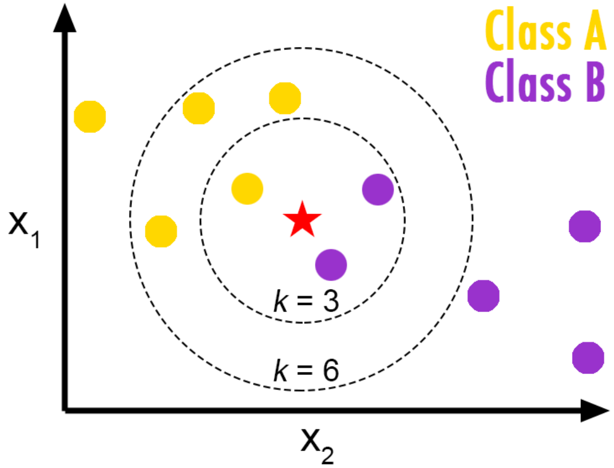
- We will first look at a non-parametric model for classification.
- Non-parametric models are, well, models without any parameters to be learned.
- KNN works by looking at the closest training samples in feature space.
- Its main advantage is that it is simple and intuitive.

KNN: The Algorithm

- The K-nearest Neighbors algorithm doesn't have a fancy mathematical model.
- Given a test sample $\mathbf{x}$ to classify:
  1. Load all training data into memory
  2. For each training example $\mathbf{x}'$ in the data:
     - Calculate the distance between $\mathbf{x}$ and $\mathbf{x}'$.
     - Add the distance and the index of $\mathbf{x}'$ to an ordered collection (ascending by distance).
  3. Pick the first K entries from the sorted collection.
  4. Get the labels of the selected K entries.
  5. Return the mode of the K selected labels (the most frequent label).

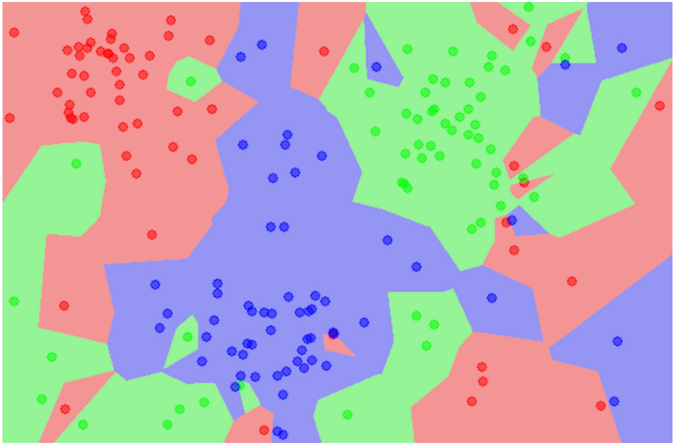
KNN: The Big Picture

- The KNN model is easily visualized:



KNN: A non-linear partitioning of feature

- KNN scales directly to multi-class problems with complex decision boundaries:



KNN: Discussion

- The KNN algorithm is extremely Simple to implement and works with arbitrary distance metrics.
- It naturally handles multi-class cases, and is optimal in practice with enough representative data.
- Its main disadvantage is that Computation cost is quite high because we need to compute the distance of each test sample to all training samples.
- Plus, we need to keep all training samples on hand, forever.
- In sklearn: sklearn.neighbors.NearestNeighbors

Linear SVMs: Overview

- The Support Vector Machine is one of the most tried and true models for classification.
- The basic theory goes back to the 1950s, but was solidified by Vapnik in the 1990s.
- It addresses many of the problems related to model complexity and overfitting.
- Nice feature: the theory developed by Vapnik lets us extend the SVM to non-linear versions using the kernel trick.

Linear SVMs: The Math

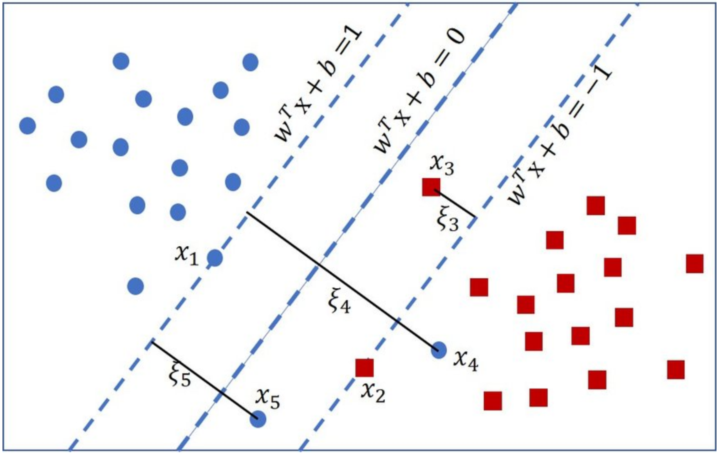
- The Model:

$$\begin{aligned} f(\mathbf{x}) &= \mathbf{w}^T\mathbf{x} + b \\ \mathcal{L}(D; \mathbf{w}, b) &= \min_{\mathrm{w}} ||\mathbf{w}||_2 + \sum_{(\mathrm{x},y)\in D} C \max(0, 1 - y f(\mathbf{x})) \\ \mathrm{class}(\mathbf{x}) &= \begin{cases} -1 & \text{if } f(\mathbf{x}) \leq 0 \\ +1 & \text{if } f(\mathbf{x}) > 0 \end{cases} \end{aligned}$$

- The Parameters: Vector of weights $\mathbf{w}$ and scalar bias b.
- The Hyperparameters: Trade-off parameter C (more if using more "fancy" formulations – should be cross-validated).
- Fitting: Efficient, exact, but iterative solution using convex optimization.

Linear SVMs: The Big Picture

- We are maximizing the margin between classes:



Linear SVMs: Discussion

- SVMs are reliable (convex guarantees global optimum) and robust (theory tells us SVM gives an "optimal" discriminant).
- Still effective in cases where number of dimensions is greater than the number of samples (C parameter).
- A disadvantage is that they do not provide probabilistic estimates of class membership.
- A linear SVM is almost always one of the first things you should try.
- In sklearn: `sklearn.svm.SVC`

Kernel Machines: The Math

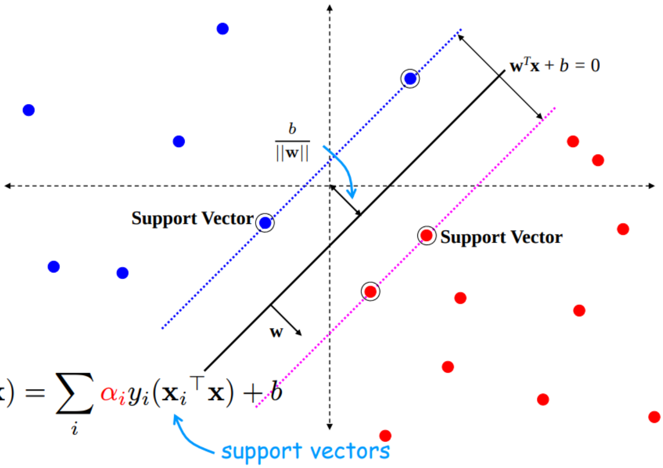
- SVMs can also be formulated in a different way (called the dual formulation:

$$f(\mathbf{x}) = \sum_{(\mathrm{x}_i, y_i) \in D} \alpha_i y_i (\mathbf{x}_i^T \mathbf{x}) + b$$

- So, we have as many parameters (α_i) as training samples now.
- At first, this seems absurd – like Nearest Neighbors, we have to keep all our training data around forever.
- In reality, usually only a small fraction of training samples have non-zero α – these are called support vectors.
- The real advantage: formulating the problem in a way that uses only inner products of vectors – this allows us to implicitly change the metric we are using to compare vectors.

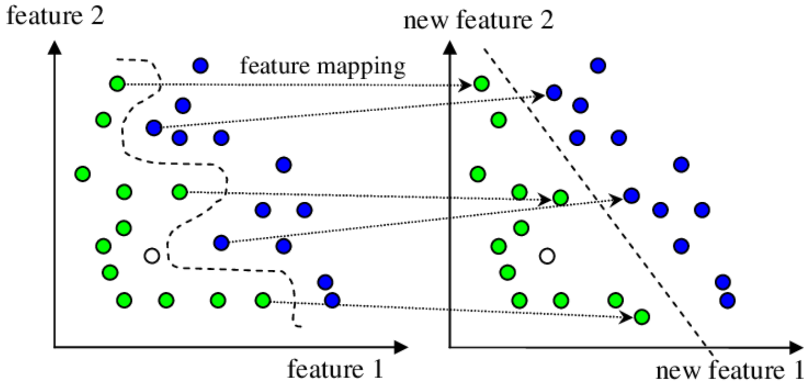
Kernel Machines: The Big Picture

- Margin maximization revisited:


$\mathbf{w}^T\mathbf{x} + b = 0$
$\frac{b}{||\mathbf{w}||}$
Support Vector
Support Vector
$\mathbf{w}$
$f(\mathbf{x}) = \sum_i \alpha_i y_i (\mathbf{x}_i{}^\top \mathbf{x}) + b$
support vectors

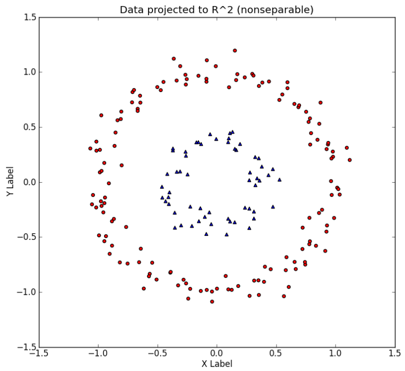
Kernel Machines: A Smaller Picture

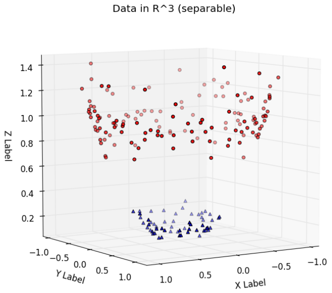
- Duality is key:



Kernel Machines: Explicit feature embeddings (revisited)

- Can think of it as an explicit feature embedding:





Kernel Machines: Discussion

- Using the dual formulation also reformulates the optimization problem in terms of a matrix of inner products between training samples.
- This is called the Kernel Matrix (or Gram matrix) – hence the name.
- The resulting classifiers are implicitly operating in another feature space – one that is of higher, or lower, or even with infinite dimensions.
- Disadvantages: the optimization problem is MUCH more computationally intensive, and introduces more hyperparameters (which depend on the kernel you use).
- In sklearn: `sklearn.svm.SVC`

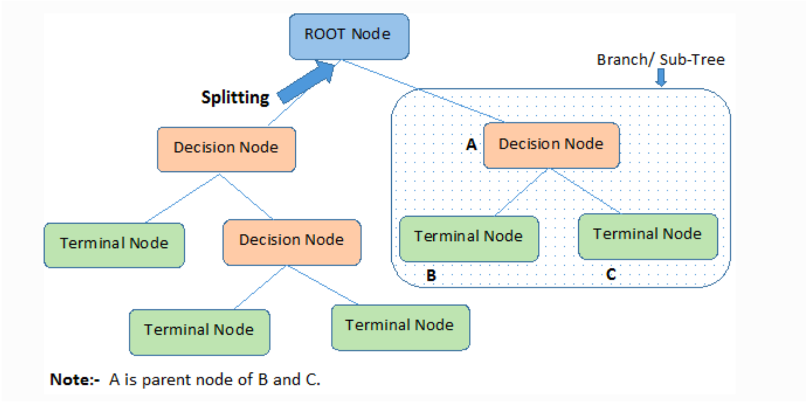
Decision Trees: Overview

- Decision trees are models that partition feature space – much like KNN.
- They are recursive models that learn how to best partition space using training data.
- They are also classical models that have been used for decades in statistics, medicine, biology – you name it.
- They are often preferred in practice because they mirror how humans think about decision making.

Decision Trees: The Algorithm (ID3)

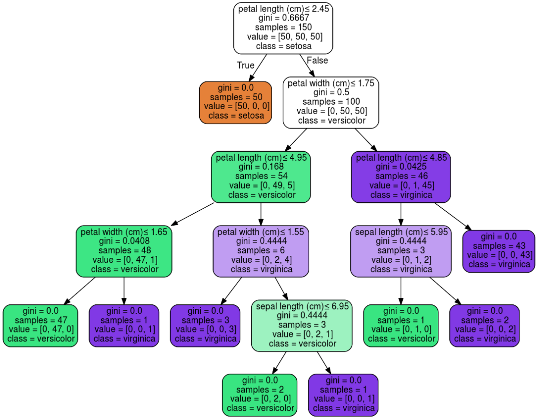
- The algorithm:
  1. Begins with the original set D as the root node.
  2. On each iteration of the algorithm, it iterates through the unused features of the set D and calculates the entropy H of this feature.
  3. It then selects the attribute which has the smallest entropy
  4. The set D is then split using the selected attribute to produce two subsets of the data.
  5. The algorithm then recursively splits each subset, considering only attributes never selected before.
  6. It terminates at nodes with pure (or nearly pure) subsets containing only one class.

Decision Trees: General Idea



Decision Trees: A Parametric View

- Decision trees can also use learned thresholds to split sets:



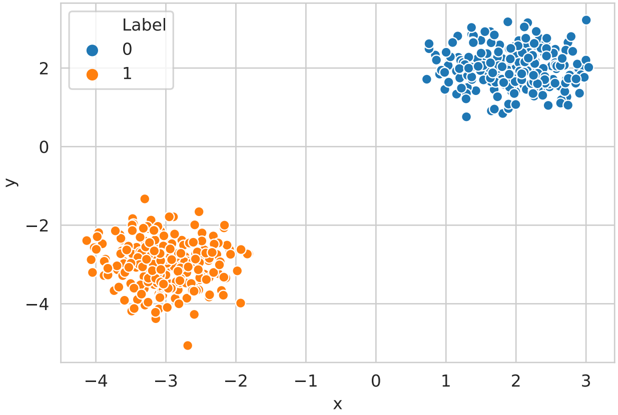
Decision Trees: Discussion

- Decision trees are nice and – most importantly – readily explainable models.
- There is a huge variety of algorithms: non-parametric, parametric, probabilistic, etc.
- They can also be used for regression.
- The main disadvantage is that they can very easily overfit the available training data (and this not generalize).
- In sklearn: `sklearn.tree.DecisionTreeClassifier`

Discriminant Analysis

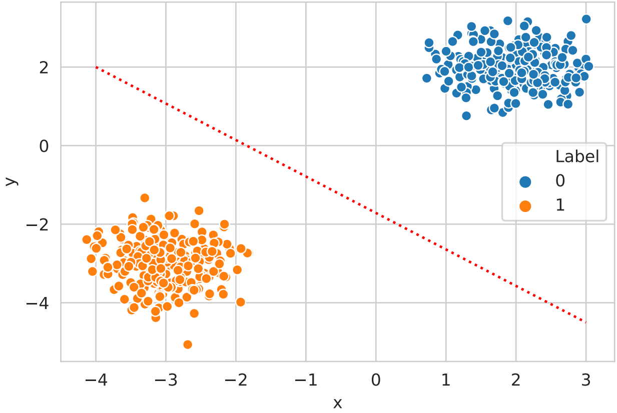
A topological tangent

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



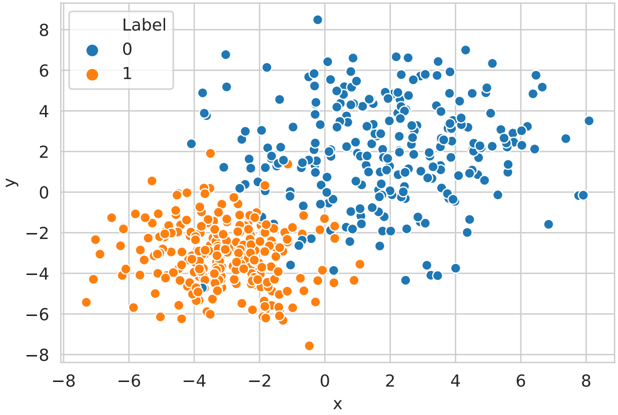
A topological tangent (continued)

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



A topological tangent (continued)

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



Discrete probability distributions

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

- Sample space Ω: 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 Ω)
- 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:

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

- 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: $E(X) = \int_x xp(x)dx$
- 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 μ and variance σ^2 and precision $\beta = 1/\sigma^2$
- What are the mode and the median of the Gaussian?

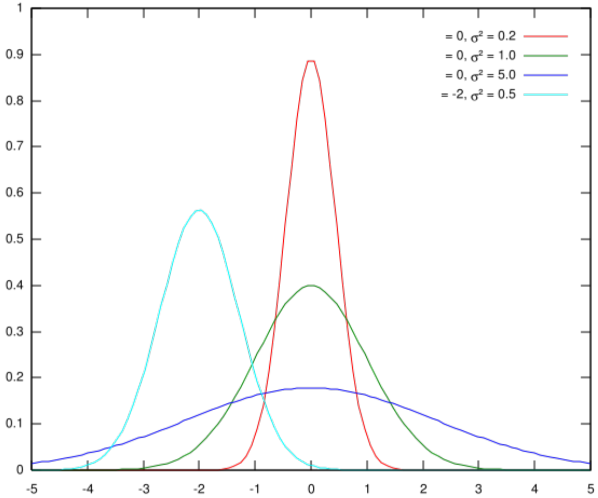
Products of Gaussians

- An aside: products of Gaussian pdfs are (unnormalized) Gaussians pdfs.
- Suppose $p_1(x) = \mathcal{N}(x, \mu_1, 1/\beta_1)$ and $p_2(x) = \mathcal{N}(x, \mu_2, 1/\beta_2)$, then:

$$\begin{aligned} p_1(x)p_2(x) &\propto \mathcal{N}(x, \mu, 1/\beta) \\ \beta &= \beta_1 + \beta_2 \\ \mu &= \frac{1}{\beta}(\beta_1\mu_1 + \beta_2\mu_2) \end{aligned}$$

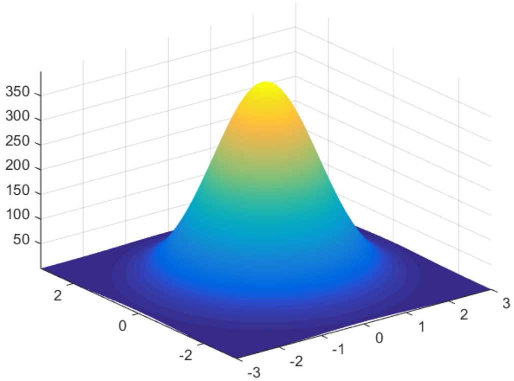
Gaussian distributions

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



The multivariate Gaussian

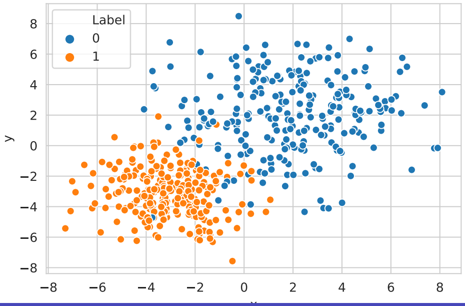
$$f(\mathbf{x}; \boldsymbol{\mu}, \Sigma) = \frac{1}{\sqrt{(2\pi)^k |\Sigma|}} \exp(-\frac{1}{2}(\mathbf{x} - \boldsymbol{\mu})^T \Sigma^{-1} (\mathbf{x} - \boldsymbol{\mu}))$$



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:

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



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.
- Next week we will start talking about the bias/variance tradeoff and the problem of overfitting.