Introduction to Machine Learning
Summary

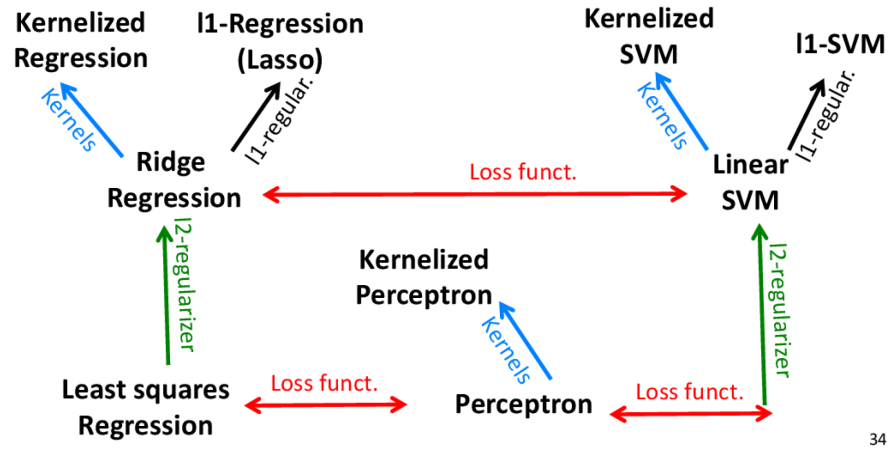
Philip Hartout

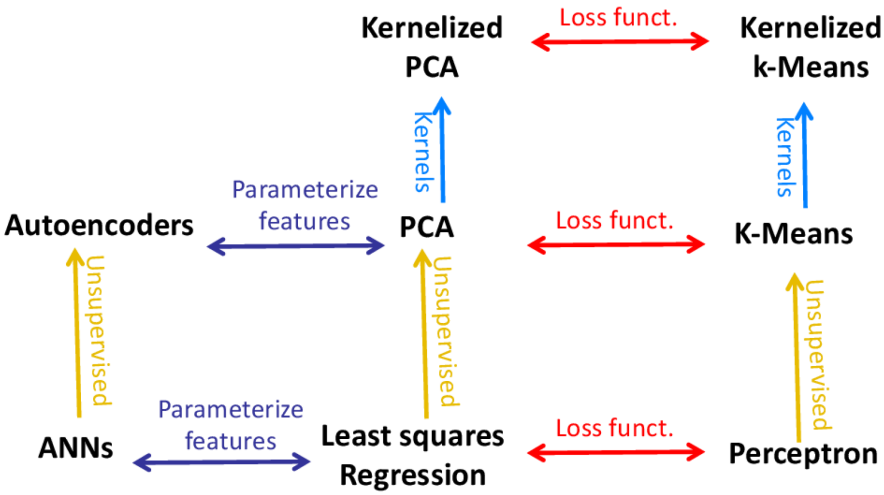
August 12, 2020

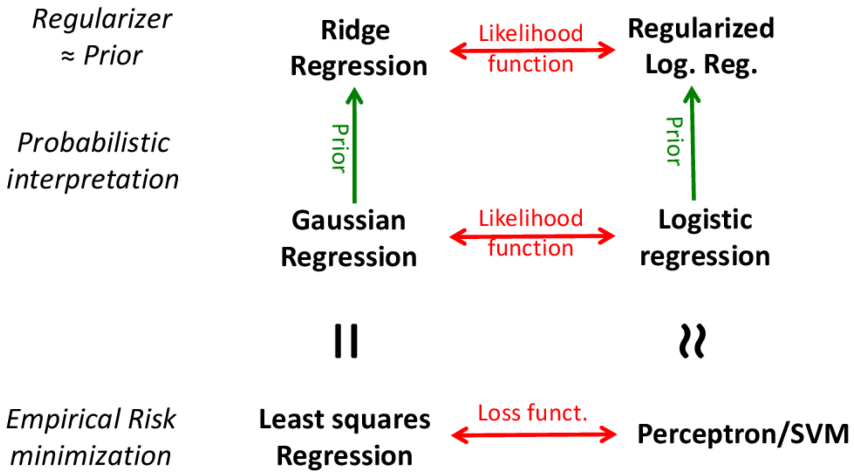
Contents

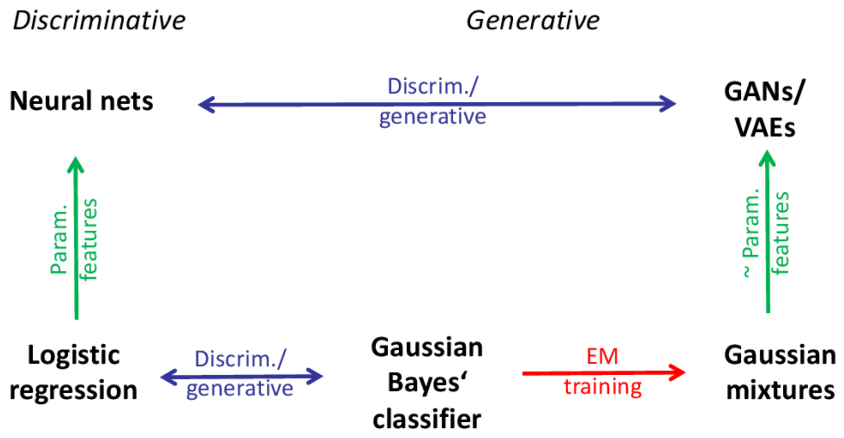
1 Overview

Here is an overview of the courses given in the diagrams below.









2 Linear Regression

Objective, approximate:

$$\begin{aligned} f(x) &= w_1 x_1 + \ldots + w_d x_d + w_0 \\ &= \sum_{i=1}^{d} w_i x_i + w_0 \\ &= \mathbf{w}^T \mathbf{x} + w_0 \end{aligned}$$

$\forall \mathbf{x}, \mathbf{w} \in \mathbb{R}^d$. This expression can be further compressed to the homogeneous representation where $\forall \tilde{\mathbf{x}}, \tilde{\mathbf{w}} \in \mathbb{R}^{d+1}$, i.e. $\tilde{x}_{d+1} = 1$. We have w.l.o.g.:

$$f(x) = \mathbf{w}^T \mathbf{x}$$

Quantify errors using residuals:

$$\begin{aligned} r_i &= y_i - f(x_i) \\ &= y_i - \mathbf{w}^T \mathbf{x_i} \end{aligned}$$

We can use squared residuals and sum over all residuals to get the cost:

$$\hat{R}(w) = \sum_{i=1}^{n} r_i^2 \tag{1}$$

$$= \sum_{i=1}^{n} (y_i - \mathbf{w}^T \mathbf{x_i})^2 \tag{2}$$

Optimization objective to find optimal weight vector $\mathbf{w}$ with least squares is the following:

$$\mathbf{w} = \arg\min_{\mathbf{w}} \sum_{i=1}^{n} \left(y_i - w^T x_i\right)^2$$

2.1 Closed form solution

This can be solved in closed form:

$$\hat{\mathbf{w}} = (\mathbf{X}^T \mathbf{X})^{-1} \mathbf{X}^T \mathbf{y}$$

where:

$$X = \begin{bmatrix} X_{1,1} & \ldots & X_{1,d} \\ \vdots & \ddots & \vdots \\ X_{n,1} & \ldots & X_{n,d} \end{bmatrix} \text{ and } y = \begin{bmatrix} y_1 \\ \vdots \\ y_n \end{bmatrix}$$

2.2 Optimization

2.2.1 Requirements

Requires a convex objective function.

Definition 2.1 (Convexity)**.** *A function is convex iff* $\forall \mathbf{x}, \mathbf{x}', \lambda \in [0,1]$ *it holds that* $f(\lambda x + (1-\lambda)x') \leq \lambda f(x) + (1-\lambda)f(x')$

Note that the least squares objective function defined in 1 is convex.

2.2.2 Gradient descent

We start with an arbitrary $w_0 \in \mathbb{R}^d$, then for $t = 0, 1, 2, \ldots$ we perform the following operation:

$$w_{t+1} = w_t - \eta_t \nabla \hat{R}(w_t)$$

where η_t is the learning rate.
Under mild assumptions, if the step size is sufficiently small, the gradient descent procedure converges to a stationary point, where the gradient is zero. For convex objectives, it therefore finds the optimal solution. In the case of the squared loss and a constant step size (e.g. 0.5), the algorithm converges at linear rate. If you look at the difference in empirical value at iteration t and compare that with the optimal value, then the gap is going to shrink at linear rate. If we look for a solution within a margin ϵ, it is found in $\mathcal{O}(\ln(\frac{1}{\epsilon}))$ iterations. The fact that the objective function congerges at linear rate can be formally described as follows:

$$\exists t_0 \forall t \geq t_0, \exists \alpha < 1 \text{ s.t. } (\hat{R}(w_{t+1}) - \hat{R}(\hat{w})) \leq \alpha(\hat{R}(w_t) - \hat{R}(\hat{w}))$$

where $\hat{w}$ is the optimal value for the hyperparameters.
For computing the gradient, we recall that:

$$\nabla \hat{R}(\hat{w}) = \begin{bmatrix} \frac{\partial}{\partial w_1}\hat{R}(w) & \ldots & \frac{\partial}{\partial w_d}\hat{R}(w) \end{bmatrix}$$

In one dimension, we have that:

$$\begin{aligned} \nabla \hat{R}(w) = \frac{d}{dw}\hat{R}(w) &= \frac{d}{dw}\sum_{i=1}^{n}(y_i - w \cdot x_i)^2 \\ &= \sum_{i=1}^{n}\frac{d}{dw}(y_i - w \cdot x_i)^2 \\ &= 2(y_i - w \cdot x_i) \cdot (-x_i) \\ &= \sum_{i=1}^{n} 2(y_i - w \cdot x_i) \cdot (-x_i) \\ &= -2\sum_{i=1}^{n} r_i x_i. \end{aligned}$$

In d-dimension, we have that:

$$\nabla \hat{R}(w) = -2\sum_{i=1}^{n} r_i x_i,$$

where $r_i \in \mathbb{R}$ and $x_i \in \mathbb{R}^d$

2.2.3 Adaptive step size for gradient descent

The step size can be updates adaptively, via either:

1. **Line search**:
   Suppose at iteration t, we have w_t, $g_t = \nabla \hat{R}(w_t)$. We then define:

$$y_t^* = \arg \min_{y \in [0,\infty)} \hat{R}(w_t) - \eta g_t$$

2. **Bold driver heuristic**:

   - If the function decreases, increase the step size.

$$\text{If } \hat{R}(w_{t+1}) < \hat{R}(w_t) : \eta_{t+1} \leftarrow \eta_t \cdot c_{acc}$$

     where $c_{acc} > 1$

   - If the function increases, decrease the step size.

$$\text{If } \hat{R}(w_{t+1}) > \hat{R}(w_t) : \eta_{t+1} \leftarrow \eta_t \cdot c_{dec}$$

     where $c_{dec} < 1$.

2.2.4 Tradeoff between gradient descent and closed form

Several reasons:

- Computational complexity:

$$\hat{w} = (X^T X)^{-1}(X^T y)$$

  $(X^T X)$ can be computed in $\mathcal{O}(nd^2)$, $(X^T X)^{-1}$ can be computed in $\mathcal{O}(d^3)$.
  By comparison, for gradient descent calculating $\nabla \hat{R}(w) = \sum_{i=1}^{n}(y_i - w^T x_i)x_i$ can be computed in $\mathcal{O}(nd)$, where $n = \ln(\frac{1}{\epsilon})$

- the problem may not require an optimal solution.
- many problems do not admit a closed form solution.

2.3 other loss functions

Least squares is part of a general case of the following general loss function, which is convex for $p \geq 1$.

$$l_p(r) = |r|^p \tag{3}$$

Least squares is where $p = 2$.

3 Probability (interlude)

3.1 Gaussians

The p.d.f. of a Gaussian distribution is given by:

$$\frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{x-\mu}{2\sigma^2}\right) \tag{4}$$

The p.d.f. of a multivariate Gaussian distribution is given by:

$$\frac{1}{2\pi\sqrt{|\sigma|}} \exp\left(-\frac{1}{2}(x-\mu)^T \sigma^{-1}(x-\mu)\right) \tag{5}$$

where:

$$\sigma = \begin{pmatrix}\sigma_1^2, \sigma_{12} \\ \sigma_{21}, \sigma_2^2\end{pmatrix} \text{ and } \begin{pmatrix}\mu_1 \\ \mu_2\end{pmatrix} \tag{6}$$

3.2 Expectations

Expected value of a random variable can be calculated as follows:

$$\mathbb{E} = \begin{cases} \sum_x xp(x) & \text{if } X \text{ is discrete} \\ \int xp(x)dx & \text{if } X \text{ is continuous} \end{cases}$$

Expectations respect linear properties, i.e. let X, Y be random variable and $a, b \in \mathbb{R}$, then we have $\mathbb{E}[aX + bY] = a\mathbb{E}[X] + b\mathbb{E}[Y]$.

4 Generalization and model validation

4.1 Fitting nonlinear functions via linear regression

Using nonlinear features of our data (basis functions), we can fit nonlinear functions via linear regression. Then, the model takes on the form:

$$f(\mathbf{x}) = \sum_{i=1}^{d} w_i \phi(\mathbf{x}) \tag{7}$$

where $\mathbf{x} \in \mathbb{R}^d$, $x \mapsto \tilde{x} = \phi(\mathbf{x}) \in \mathbb{R}^d$ and $w \in \mathbb{R}^d$.

- 1 dim.: $\phi(\mathbf{x}) = \left[1, x, x^2, \ldots, x^k\right]$
- 2 dim.: $\phi(\mathbf{x}) = \left[1, x_1, x_2, x_1^2, x_2^2, \ldots, x_1^k, x_2^k\right]$
- p dim.: $\phi(\mathbf{x})$ vector of all monomials in $x_1, \ldots, x_p$ of degree up to k.

4.2 Achieving generalization

4.2.1 Independence and identical distribution

A fundamental assumption needs to be met: the dataset is generated from an independently and identically distributed from some unknown distribution P, i.e:

$$(x_i, y_i) \sim P(\mathbf{X}, Y).$$

The i.i.d. assumption is invalid when:

- we deal with time series data
- spatially correlated data
- correlated noise

If violated, we can still use ML but the interpretation of the results needs to be carefully analyzed. The most important thing is to choose the train/test split to assess the desired generalization properties of the trained model.

4.3 Expected error and generalization error

Once the iid assumption is verified, our goal is then to minimize the expected error (true risk) under P, i.e.:

$$\begin{aligned} R(\mathbf{w}) &= \int P(\mathbf{x}, y)(y - \mathbf{w}^T\mathbf{x})^2 dxdy \\ &= \mathbb{E}[(y - \mathbf{w}^T\mathbf{x})^2] \end{aligned}$$

The true risk can be estimated by the empirical risk on a sample dataset D:

$$\hat{R}_D(\mathbf{w}) = \frac{1}{|D|} \sum_{\mathbf{x}, y \in D} (y - \mathbf{w}^T\mathbf{x})^2$$

The reason behind this approximation is because of the law of large numbers

Definition 4.1 (Law of large numbers)**.** $\hat{R}_D(\mathbf{w}) \to R_D(\mathbf{w})$ *for any fixed* $\mathbf{w}$ *as* $|D| \to \infty$.

$$\hat{\mathbf{w}}_D = \arg\min_{\mathbf{w}} \hat{R}_D(\mathbf{w}) \tag{8}$$

$$\mathbf{w}* = \arg\min_{\mathbf{w}} \hat{R}(\mathbf{w}) \tag{9}$$

We don't want to minimize the empirical risk given in equation 8 but the true risk given in equation 9, which are similar as the amount points in the dataset increases.

4.4 Uniform convergence

For learning via empirical risk minimization, uniform convergence is required, i.e.:

$$\sup_{\mathbf{w}} |R(\mathbf{w}) - \hat{R}_D(\mathbf{w})| \to 0 \text{ as } |D| \to \infty$$

Note that this is not implied by the law of large numbers alone, but depends on model class. It holds for instance for squared loss on data distributions with bouded support. Statistical learning theory is required to define these properties.

4.5 Evaluation of performance on training data

In general in holds that:

$$\mathbb{E}_D[\hat{R}_D(\hat{\mathbf{w}}_D)] \leq \mathbb{E}_D[R_D(\hat{\mathbf{w}}_D)]$$

Proof.

$$\begin{aligned}
\mathbb{E}[\hat{R}_D(\hat{\mathbf{w}}_D)] &= \mathbb{E}_D[\min_{\mathbf{w}} \hat{R}_D(\mathbf{w})] && \text{(ERM)} \\
&\leq \min_{\mathbf{w}} \mathbb{E}_D[\hat{R}_D(\mathbf{w})] && \text{(Jensen's inequality)} \\
&= \min_{\mathbf{w}} \mathbb{E}_D[\frac{1}{|D|} \sum_{i=1}^{|D|} (y_i - wx_i)^2] && \text{(Definition of } \hat{R}_D(.)) \\
&= \min_{\mathbf{w}} \mathbb{E}_D[\frac{1}{|D|} \sum_{i=1}^{|D|} (y_i - wx_i)^2] && \text{(linear expectations)} \\
&= \min_{\mathbf{w}} R(\mathbf{w}) \leq \mathbb{E}[R(\hat{w}_D)]
\end{aligned}$$

□

Thus, we obtain an overly optimistic estimate. A more realistic evaluation would be to use a separate test set from the same distribution P. Then:

- Optimize w on training set:
$$\hat{\mathbf{w}}_{D_{\text{train}}} = \arg\min_{\precapprox} \hat{R}_{\text{train}}(\precapprox)$$
- Evaluate on test set:
$$\hat{R}_{\text{test}}(\hat{\mathbf{w}}) = \frac{1}{|D_{\text{test}}|} \sum_{\mathbf{x},y \in D_{\text{test}}} (y - \hat{\mathbf{w}}^T \mathbf{x})^2$$
- Then:
$$\mathbb{E}_{D_{\text{train}}, D_{\text{test}}}[\hat{R}_{D_{\text{test}}}(\hat{\mathbf{w}}_{D_{\text{train}}})] = \mathbb{E}_{D_{\text{train}}}[R(\hat{\mathbf{w}}_{D_{\text{train}}})]$$

Proof. Let $D_train = D$, $D_test = V$ and $D, V \sim P$. Then:

$$\begin{aligned}
\mathbb{E}_{D,V}[\hat{R}_V(\hat{\mathbf{w}}_D)] &= \mathbb{E}_D[\mathbb{E}_V[\hat{R}_V(\hat{\mathbf{w}}_D)]] \qquad \text{independence of } D, V \\
&= \mathbb{E}_D[\mathbb{E}_V[\frac{1}{|V|}\sum_{i=1}^{|V|}(y_i - \hat{\mathbf{w}}_D^T x_i)^2]] \qquad \text{(Definition of } \hat{R}_D(.)) \\
&= \mathbb{E}_D[\frac{1}{|V|}\sum_{i=1}^{|V|}\mathbb{E}_{x_i,y_i}(y_i - \hat{\mathbf{w}}_D^T x_i)^2] \qquad \text{since } (x_i, y_i) \perp D \\
&= \mathbb{E}_D[R(\hat{\mathbf{w}}_D)]
\end{aligned}$$

□

4.6 Evaluation for model selection

For each candidate model m, we repeat the following procedure for $i = 1 : k$:

- We split the same dataset into training and validation sets:
$$D = D_{\text{train}}^{(i)} \biguplus D_{\text{val}}^{(i)}$$
- We train the model:
$$\hat{\mathbf{w}}_[i, m] = \arg\min_{\mathbf{w}} \hat{R}_{\text{train}}^{(i)}(\mathbf{w})$$
- Then we estimate the error:
$$\hat{R}_m^{(i)} = \hat{R}_{\text{val}}^{(i)}(\hat{\mathbf{w}}_i)$$

Finally, select the model:

$$\hat{m} = \arg\min_m \frac{1}{k}\sum_{i=1}^{k}\hat{R}_m^{(i)}$$

4.7 Splitting the data for model selection

This splitting can be done randomly through Monte Carlo cross-validation.

- Pick training set of given size uniformly at random
- Validate on remaining points
- Estimate prediction error by averaging the validation error over multiple random trials.

It can also be achieved through k-fold cross-validation, which is the default choice.

- Partition the data into k folds
- Train on $k-1$ folds, evaluating on remaining fold.
- Estimate prediction error by averaging the validation error obtained while varying the validation fold.

Note that the cross-validation error is almost unbiased for large enough k. The following should be considered to pick k:

- Too small:
  - Risk of overfitting on test set
  - Using too little data for training
  - Risk of underfitting to training set
- Too large:
  - In general, leads to better performance. $k = n$ is perfectly fine, specific instance called leave-one-out cross-validation
  - Higher computational complexity.

In practice, $k = 5$ or $k = 10$ is ofen used and works well.

4.8 Best practice for evaludating models in supervised learning

Follow the following steps:

- Split data set into training and test set
- Never look at test set when fitting the model. For example, use k-fold cross-validation on training set
- Report final accuracy on test set, but never optimize on it.

Note that this procedure only works if the data is i.i.d. I.e. one should be careful if there are temporal trends or other dependencies.

5 Regularization

We want to avoid having overly complex models when minimizing the loss function. This can be achieved through regularization, which encourages small weights via penalty functions, which are called regularizers.

5.1 Ridge regression

This is a regularized optimization problem:

$$\min_{\mathbf{w}} \frac{1}{n}\sum_{i=1}^{n}(y_i - \mathbf{w}^T x_i)^2 + \lambda \|\mathbf{w}\|_2^2 = \sum_{j=1}^{d} \mathbf{w}_j^2 \qquad \forall \lambda \geq 0$$

5.1.1 Closed form solution

This can be optimized using the closed form solution or gradient descent. The closed form for Ridge regression is:

$$\hat{\mathbf{w}} = (\mathbf{X}^T\mathbf{X} + \lambda \mathbf{R})^{-1}\mathbf{X}^T y$$

where $I \in \mathbb{R}^{d\times d}$ is the identity matrix.

5.1.2 Gradient descent

$$\nabla\left(\frac{1}{n}\sum_{i=1}^{n}(y_i - \mathbf{w}^T\mathbf{x}_i)^2 + \lambda\|\mathbf{w}\|_2^2\right) = \nabla_{\mathbf{w}}\hat{R}(\mathbf{w}) + \lambda\nabla_{\mathbf{w}}\|\mathbf{w}\|_2^2$$

One step of the gradient descent is therefore performed as follows:

$$\begin{aligned} w_{t+1} &\leftarrow w_t - \eta_t(\nabla_{\mathbf{w}}\hat{R}(\mathbf{w}_t) + 2\lambda\mathbf{w}_t) \\ &= (1 - 2\lambda\eta_t)\mathbf{w}_t - \eta_t\nabla_{\mathbf{w}}\hat{R}(\mathbf{w}_t) \end{aligned}$$

Choosing the regularization parameter is done through cross-validation. Typically, the choice is between values of λ values which are logarithmically spaced.

5.1.3 Generalization of a tradeoff in ML

A lot of supervised learning problems can be written in this way:

$$\min_{\mathbf{w}} \hat{R}(\mathbf{w}) + \lambda C(\mathbf{w}).$$

It's possible to control complexity by varying regulzation parameter λ.

5.2 Renormalizing data through standardization

This process ensures that each feature has zero mean and unit variance:

$$\tilde{x}_{i,j} = \frac{(x_{i,j}) - \hat{\mu}_j}{\hat{\sigma}_j}$$

where $x_{i,j}$ is the value of the j^{th} feature of the i^{th} data point:

$$\hat{\mu}_j = \frac{1}{n}\sum_{i=1}^{n} x_{i,j} \qquad \sigma_j^2 = \frac{1}{n}\sum_{i=1}^{n}(x_{i,j} - \hat{\mu}_j)^2$$

6 Classification

Definition 6.1 (Classification)**.** *Classification is an instance of supervised learning where Y is discrete (categorical). We wanto to assign data points X (documents, queries, images, user visits) a label Y (spam/not spam, topic such as sports, politics, entertainment, click/no-click etc).*

The input of the model is labeled data set with positive and negative examples, see Figure 1. The output is a decision rule, i.e. a hypothesis. Given a dataset $D = \{(x_1, y_1), \dots (x_n, y_n)\}$, we

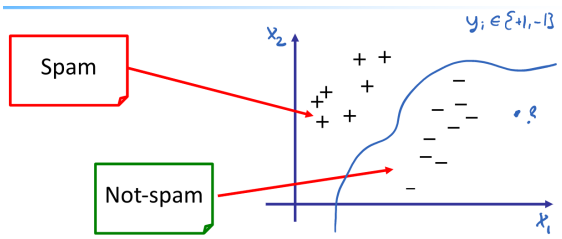


Figure 1: Illustration of binary classification

have:

$$y \approx h_{\mathbf{w}}(x) = \text{sign}(w^T x)$$

Linear classification works well in high-dimensional settings when using the right features; prediction is typically very efficient despite linear classification seeming very restrictive at first.

6.1 Finding linear separators

Writing the search for a classifier can be seen as an optimization problem: we seek the set of weights $\mathbf{w}$ that minimizes the number of mistakes, i.e.:

$$\hat{\mathbf{w}} = \arg\min_{\mathbf{w}\in\mathbb{R}} \sum_{i=1}^{n} [y_i \neq \text{sign}(w^T x_i)] = \begin{cases} 1 \text{ if } y_i \neq \text{sign}(w^T x_i) \\ 0 \text{ otherwise.} \end{cases}$$

The goal is then to optimize the following function:

$$\begin{aligned} \hat{\mathbf{w}} &= \arg\min_{\mathbf{w}\in\mathbb{R}} \frac{1}{n}\sum_{i=1}^{n} [y_i \neq \text{sign}(w^T x_i)] \\ &= \frac{1}{n}\sum_{i=1}^{n} \ell(w; x_i, y_i) \end{aligned}$$

Note that this poses as challenge as it is not convex or even differentiable. Therefore we need to replace this loss by a tractable loss function for the sake of optimization/model fitting. When evaluating a model, we then use the original cost/performance function. The function we can use to optimize in this case is the surrogate loss:

$$l_P(\mathbf{w}; y_i, x_i) = \max(0, -y_i \mathbf{w}^T x_i)$$

which is also referred to as the perceptron loss.

The gradient of the perceptron loss function can be computed as follows:

$$\begin{aligned} R(\hat{w}) &= \sum_{i=1}^{n} \max(0, -y_i w^T x_i) \\ \nabla R(\hat{w}) &= \sum_{i=1}^{n} \nabla_{\mathbf{w}} \max(0, -y_i w^T x_i) \\ &= \begin{cases} 0 \text{ if } y_i w^T x_i \geq 0 \text{ i.e. correctly classified} \\ -y_i x_i \text{ otherwise} \end{cases} \end{aligned}$$

So we have the following update rule:

$$\mathbf{w}_{t+1} \leftarrow \mathbf{w} w_t + \eta_t \sum_{i:(x_i,y_i)\text{incorrectly classified by } w}^{x_i y_i}$$

6.2 Stochastic gradient descent

Computing the gradient requires summing over all data, which is inefficient for large datasets. Additionally, our initial estimates are likely very wrong and we can get a good unbiased gradient estimate by evaluating the gradient on few points. In the worst case, we can evaluate only one randomly chosen point, which is a procedure called stochastic gradient descent. It consists of the following steps:

1. Start at an arbitrary $\mathbf{w}_0 \in \mathbb{R}^d$
2. For $t = 1, 2, \ldots$ do:
   - Pick data point $(\mathbf{x}', y') \in D$ from training set uniformly at random (with replacement), and set:
   $$\mathbf{w}_{t+1} = \mathbf{w}_t - \eta_t \nabla \ell(\mathbf{w}_t; \mathbf{x}', y')$$

Where η_t is called the learning rate. Guaranteed to converge under mild conditions, if:

$$\sum_t \eta_t = \infty \text{ and } \sum_t \eta_t^2 < \infty$$

for instance $\eta_t = \frac{1}{t}$ and $\eta_t = \min(c, \frac{c'}{t})$.
The perceptron algorithm is just stochastic gradient descent on the perceptron loss function ℓ_P with learning rate 1.

Theorem 1 (Perceptron algorithm)**.** *If the data is linearly separable, the perceptron will obtain a linear separator.*

The variance of the gradient estimate can be reduced by averaging over the gradients w.r.t. multiple randomly selected points, which are called minibatches. Adaptive learning rates can be additionally applie. There exist various approaches for adaptively tuning the learning rate. Often times, these even use a different learning rate per feature. Examples of adaptive learning rate algorithms include AdaGrad, RMSProp, Adam, ...

6.3 Hinge loss vs. perceptron loss

The Hinge loss encourages the margin of the classifier, and is defined as follows:

$$\ell_H(\mathbf{w}; \mathbf{x}, y) = \max\left\{0, 1 - y\mathbf{w}^T\mathbf{x}\right\}$$

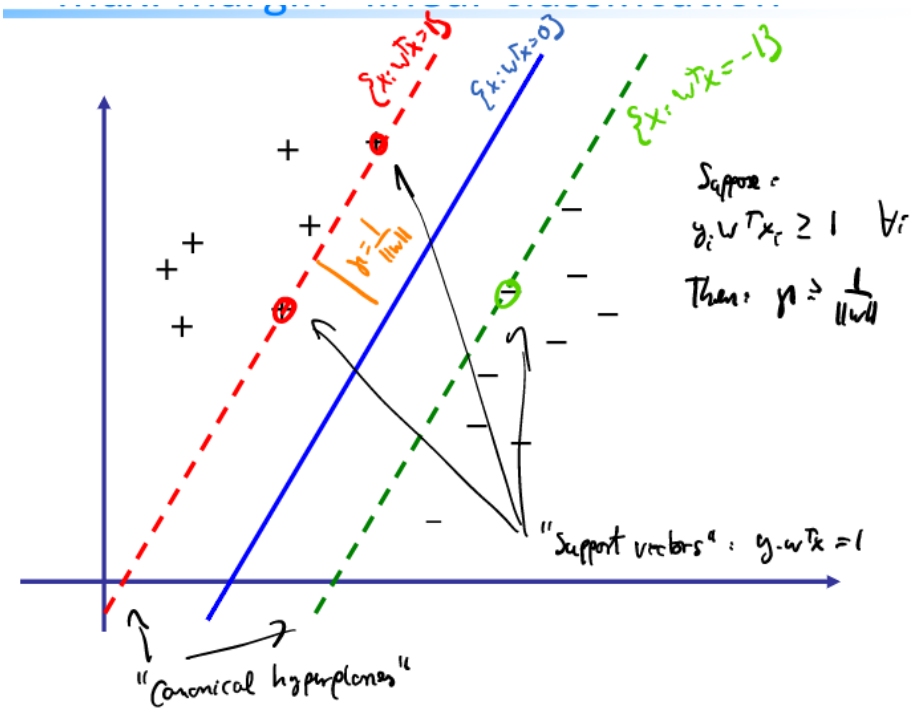


Figure 2: Important elements and their defining equations for support vector machines.

6.4 Support vector machines

The optimization objective for the support vector machine is defined as minimizing Hinge loss while also adding another regularization term. There are several lines that need to be considered in the max. margin linear classification, which are summarised in figure 2

Support vector machinea are widely used, very effective linear classifiers. They behave almost like a perceptron. The only differences include:

- Optimize slightly different, shifted loss (hinge loss)
- They regularize the weights

It can be optimized using a stochastic gradient descent. A safe choice for the learning rate is:

$$\eta_t = \frac{1}{\lambda t}$$

6.4.1 Stochastic gradient descent for support vector machines

Let's recall the ojective function:

$$\hat{R}(\mathbf{w}) = \sum_{i=1}^{n} \ell_H(\mathbf{w}_i; x_i, y_i) + \lambda \|\mathbf{w}\|_2^2$$

This requires taking care of the regularizer, as follows:

$$\hat{R}(\mathbf{w}) = \sum_{i=1}^{n} \left(\ell_H(\mathbf{w}_i; x_i, y_i) + \frac{\lambda}{n} \|\mathbf{w}\|_2^2 \right) = f_i(\mathbf{w})$$

So, in order to estimate the gradient $\nabla_{\mathbf{w}} \hat{R}(\mathbf{w})$ which is equivalent to:

$$\nabla_{\mathbf{w}} \hat{R}(\mathbf{w}) = \sum_{i=1}^{n} \nabla_{\mathbf{w}} f_i(\mathbf{w})$$

where:

$$\nabla_{\mathbf{w}} f_i(\mathbf{w}) = \nabla \ell_H(\mathbf{w} + \frac{\lambda}{n} \nabla \|\mathbf{w}\|_2^2)$$

so the gradient of the regularization term is just $2\mathbf{w}$. For the Hinge loss:

$$\nabla \ell_H = \nabla \max(0, 1 - y_i \mathbf{w}^T x_i) = \begin{cases} 0 \text{ if } y_i w^T x_i \geq 1 \\ -y_i x_i \text{ otherwise.} \end{cases}$$

Therefore the entire update rule for stochastic gradient descent for SVM is:

$$\mathbf{w}_{t+1} \leftarrow w_t \left(1 - \eta_t \frac{2\lambda}{n}\right) + \left[y_i w^T x_i\right]$$

The regularization parameter can be picked via cross-validation just like in linear regression. Instead of using the Hinge loss for validation, the target performance metric needs to be used.

6.5 Key takeaways

The key takeaways are:

- The perceptron is an algorithm for linear classification
- It applies SGD on the perceptron loss
- Mini-batches exploit parallelism and reduce variance compared to a single sample
- The perceptron loss is a convex surrogate function for the 0-1 misclassification loss
- It is guaranteed to produce a feasible solution if the data is separable
- SGD is much more generally applicable
- SVMs are closely related to Perceptron, they use a hinge loss and regularization.

Summary so far:

- List of represnetations/features: linear hypotheses, nonlinear hypotheses with nonlinear feature transforms.
- Model/Objective: loss function (squared loss, 0/1 loss, perceptron loss, Hinge loss) + regularization (L^2 norm)
- Method: exact solution, gradient descent, mini-batch SGD, convex programming.
- Evaluation metric: MSE, accuracy
- Model selection: k-fold cross-validation, Monte Carlo CV.

7 Feature selection

Reasons why we don't want to work with all potentially available features:

- interpretability: understand which features are most important
- generalization: simpler models may generalize better
- storage computation and cost: if we select the most important features we don't need to store, sum and acquire data for unused features.

The naive way to select features is to try all subsets and pick the best features via cross-validation. Greedy feature selection, which is a general purpose approach, consists of greedily add or remove features to maximize cross-validated prediction accuracy and mutual information or other notions of informativeness not discussed here. It can be used for any method, not only linear regression or classifiers.

7.1 General greedy approach

Consider the set of features $V = \{1, \dots, d\}$. We then define the cost function for scoring subsets S of V. $\hat{L}(S)$ is the cross-validation error using features in S only. More precisely:

$$\mathbf{x}_i = [x_{i,1}, \dots, x_{i,d}] \to \mathbf{x}_{S,i} = [x_{i,j}, \dots, x_{i,j_k}]$$

where S is defined as:

$$S = \{j, \dots, j_k\}, k = |k|$$

We then train the model on $\{x_{j,1}, y_1, \dots, x_{j,n}, y_n\}$, in order to obtain an estimate based on the weights $\hat{\mathbf{w}}_S$ with associated loss $\hat{L}(S)$, which represents the cross-validated performance of the weight estimates $\hat{\mathbf{w}}_S$.

7.2 Greedy forward selection

Start with $S = \emptyset$ and $E_0 = \infty$, then for $i = 1 : d$, we find the best element to add, i.e.:

$$s_i = \arg \min_{j \in V \setminus S} \hat{L}(S \cup \{j\})$$

then we compute the error:

$$E_i = \hat{L}(S \cup \{s_i\})$$

if $E_i > E_{i-1}$ break, else set $S \leftarrow \cup \{s_i\}$

7.3 Greedy backward selection

Start with $S = V$ and $E_{d+1} = \infty$, then for $i = d : -1 : 1$, we find the best element to remove, i.e.:

$$s_i = \arg \min_{j \in S} \hat{L}(S \setminus \{j\})$$

then we compute the error:

$$E_i = \hat{L}(S \setminus \{s_i\})$$

if $E_i > E_{i-1}$ break, else set $S \leftarrow \setminus \{s_i\}$

7.4 Advantages and drawbacks of forward vs. backward feature selection

	Forward Feature Selection	Backward Feature Selection
Method Advantages	Faster (if few relevant features)	Can handle "dependent" features
Method drawback	Computational cost	Computational cost
	Suboptimal	Suboptimal

We want a method that simultaneously solves the learning and the feature selection problem via a single optimization step. So far we have only done optimization via sparsity: i.e. explicitely select a subset of features. This is equivalent to constraining $\mathbf{w}$ to be sparse, i.e. contain at most k non-zero entries. Alternatively, we can penalize the number of nonzero entries:

$$\hat{\mathbf{w}} = \arg \min_{\mathbf{w}} \sum_{i=1}^{n} (y_i - w^T x_i)^2 + \lambda \|w\|_0$$

However, this is a difficult combinatorial optimization problem. The key idea then is to replace $\|\mathbf{w}\|_0$ by a more tractable term.

The idea here is to use L_1 as a surrogate for L_0, where:

$$\|w\|_1 = \sum_{i=1}^{d} |w_i|$$

and we use $\|w\|_1$ instead of $\|w\|_0$.

7.5 Lasso regression

In ridge regression, we use $\|\mathbf{w}\|_2^2$ to control the weights. In Lasso, we replace $\|\mathbf{w}\|_2^2$ by $\|\mathbf{w}\|_1$, hence leading to the following L1-regularized regression:

$$\min_{\mathbf{w}} \lambda \|w\|_1 + \sum_{i=1}^{n} (y_i - \mathbf{w}^T x_i)^2$$

This alternative penalty encourages coefficients to be exactly 0, which entails an automatic feature selection.

The regularization parameter can be picked using cross-validation.

7.5.1 L1 regularization in SVM

The sparsity trick can be applied to SVMs as well:

$$\min_{\mathbf{w}} \|\mathbf{w}\|_1 + \sum_{i=1}^{n} \max(0, 1 - y_i \mathbf{w}^T \mathbf{x}_i)$$

This alternative penalty encourages coefficients to be exactly 0, which ignores thoses features as well, just like in Lasso regression.

7.6 Solving L1-regularized problems

The L1-norm is convex. Combined with convex losses, we can obtain convex optimization problems, which include Lasso and L1-SVM. Those problems can, in principle, be solved usng stochastic gradient descent. Convergence is, however, usually slow. and we rarely obtain exact 0 entries. Recent work in convex optimization deals with solving such problems very efficiently using proximal methods.

Method	Greedy	L1-regularization
Advantages	Applies to any prediction method	Faster (training and feature selection happen jointly)
Disadvantages	Slower (need to train many models	Only works for linear model

8 Non-linear prediction with kernels

8.1 Revisiting the perceptron/SVM

There is a fundamental observation to be made here where the optimal hyperplace lies in the span of the data.

$$\hat{\mathbf{w}} = \sum_{i=1}^{n} \alpha_i y_i \mathbf{x}_i$$

that means that the output of any of the models discussed so far can be rewritten as a linear combination of the feature inputs that we have seen so far. Proving this losely can be done by stating that SGD starts from 0 and constructs such a representation. A more abstract proof follows from the representer theorem.

8.2 Reformulating the perceptron

In order to make the objective function only depend on the inner product of pairs of data point and work implicitely in high-dimensional spaces as long as we can do inner products efficiently, we need to reformula the optimization problem in terms of α instead of $\mathbf{w}$. Therefore, we do the following:

$$\begin{aligned}
\hat{\mathbf{w}} \in \arg\min_{\mathbf{w}\in\mathbb{R}^d} \sum_{i=1}^{n} \max(0, -y_i \mathbf{w}^T x_i) \text{ Note: we make the following anzats: } \hat{\mathbf{w}} = \sum_{j=1}^{n} \alpha_j y_j x_j \\
= \sum_{i=1}^{n} \max(0, -y_i (\sum_{j=1}^{n} \alpha_j y_j x_j)^T x_i) \\
= \sum_{i=1}^{n} \max(0, -y_i \sum_{j=1}^{n} \alpha_j y_j (x_j^T x_i)) \\
\hat{\alpha} \in \arg\min_{\alpha\in\mathbb{R}^n} \sum_{i=1}^{n} \max(0, -c)
\end{aligned}$$

Often, computing $k(\mathbf{x}, \mathbf{x}')$ can be computed much more efficiently than $\phi(\mathbf{x})^T \phi(\mathbf{x}')$. For the polynomial kernel of degree 2, the computational complexity of computing the explicit feature map of the kernel function is in $\mathcal{O}(d^2)$ whereas it is in $\mathcal{O}(d)$ for the kernel computation.

The perceptron can then be reformulated as follows:

Algorithm 1: Kernelized perceptron

Result: Trained vector $\hat{\alpha}$ used for prediction

$\alpha_0 \leftarrow 0$;
for $t = 1, \ldots$ **do**
  Sample $(\mathbf{x}_i, y_i) \sim D$;
  if $y_i \sum_{j=1}^{n} \alpha_j y_j k(x_j^T x_i) > 0$ **then**
    $\alpha_{t+1} \leftarrow \alpha_t$;
  else
    $\alpha_{t+1} \leftarrow \alpha_t$;
    $\alpha_{t+1,i} \leftarrow \alpha_{t+1,i} + \eta_t$;
  end
end

For a new point, we can predict:

$$\hat{y} = \text{sign}(\sum_{j=1}^{n} \alpha_j y_j k(\mathbf{x}_j, \mathbf{x}))$$

8.3 The kernel trick

Non-linear decision boundaries can be found by using non-linear transformations of the feature vectors followed by linear classification. An important aspect to keep in mind when doing these feature transformation is the dimensionality of the data that is being used. We need, for instance, $\mathcal{O}(d^k)$ dimensions to represent multivariate polynomials of degree k on d features. The challenge then becomes to efficiently implicitely operate in such high-dimensional feature spaces withough ever explicitely computing the transformation.

Definition 8.1 (The kernel trick)**.** *The kernel trick consists in expressing a problem such that it only depends on inter products, which can then be replaced by kernels.*

An example of such a kernel can be applied to the perceptron loss:

$$\hat{\alpha} = \arg\min_{\alpha_{1:n}} \frac{1}{n} \sum_{i=1}^{n} \max \left\{ 0 - \sum_{j=1}^{n} \alpha_j y_i y_j x_i^T x_j \right\}$$

$$\Leftrightarrow \hat{\alpha} = \arg\min_{\alpha_{1:n}} \frac{1}{n} \sum_{i=1}^{n} \max \left\{ 0 - \sum_{j=1}^{n} \alpha_j y_i y_j k(\mathbf{x}_j, \mathbf{x}_i) \right\}$$

8.4 The kernelized perceptron

At training time, the problem can be solved as follows:

- Initialize $\alpha_1 = \ldots = \alpha_n = 0$
- For $t = 1, 2, \ldots$
  - Pick data point (x_i, y_i) uniformly at random
  - Predict:
  $$\hat{y} = \text{sign}(\sum_{j=1}^{n} \alpha_j y_j k(\mathbf{x}_j, \mathbf{x}_i))$$
  - If $\hat{y} \neq y_i$ set $\alpha \leftarrow \alpha_i + \eta_t$

At the time of prediction, for a new point x, we predict:

$$\hat{y} = \text{sign}(\sum_{j=1}^{n} \alpha_j y_j k(\mathbf{x}_j, \mathbf{x}_i))$$

8.5 Kernel functions

Definition 8.2 (Kernel functions)**.** *Given a data space X, a kernel is a function $k : X \times X \to \mathbb{R}$ satisfying the following properties:*

- ***Symmetry**: for any pair of vectors $\mathbf{x}, \mathbf{x}' \in X$ it must hold that:*
$$k(\mathbf{x}, \mathbf{x}') = k(\mathbf{x}', \mathbf{x})$$
- ***Positive semi-definiteness**: for any n, any set $S = \{\mathbf{x}_1, \ldots \mathbf{x}_n\} \subseteq X$, the kernel (Gram) matrix defined as:*
$$K = \begin{bmatrix} k(\mathbf{x}_1, \mathbf{x}_1) & \ldots & k(\mathbf{x}_1, \mathbf{x}_n) \\ \vdots & \ddots & \vdots \\ k(\mathbf{x}_n, \mathbf{x}_1) & \ldots & k(\mathbf{x}_n, \mathbf{x}_n) \end{bmatrix}$$
must be positive semi-definite.

A matrix is positive semidefinite iff:

i $\forall x \in \mathbb{R}^n : \mathbf{x}^T M \mathbf{x} \geq 0$

ii All eigenvalues of $M \geq 0$

Suppose the data space $X = \{1, \ldots, n\}$ is finite, and we are given a p.s.d. matrix $\mathbf{K} \in \mathbb{R}^{n \times n}$, then we can always construct a feature map:

$$\phi : Z \to \mathbb{R}^n$$

such that $\mathbf{K}_{i,j} = \phi(i)^T \phi(j)$.

Proof. $\mathbf{K}$ is p.s.d. $\Rightarrow \mathbf{K} = UDU^T$ where $D = \begin{bmatrix} \lambda_1 & \dots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \dots & \lambda_n \end{bmatrix}$ and $\lambda_i \geq 0 \forall i$ We then define a matrix $D = D^{1/2}TD^{1/2}$ where $D^{1/2} \begin{bmatrix} \sqrt{\lambda_1} & \dots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \dots & \sqrt{\lambda_n} \end{bmatrix}$ and $\lambda_i \geq 0 \forall i$. This gives us the following: $\mathbf{K} = UD^{1/2}D^{1/2}U^T = \phi^T\phi$ where $\phi = [\phi_1 | \dots | \phi_n]$. Now it holds that taking $k(i,j) = \mathbf{K}_{i,j} = \phi_i^T\phi_j$ which means that that $\phi : X \to \mathbb{R}^n$ and $\phi : i \to \phi_i$ which is a constructed valid feature map. It shows that for finite data spaces X, positive definiteness of the function is also a sufficient condition for it being a valid kernel. □

More generally:

Theorem 2 (Mercer's theorem)**.** *Let X be a compact subset of $\mathbb{R}^n$ and $k : X \times X \to \mathbb{R}^n$ a kernel function. Then one can expand k in a uniformly convergent series of bounded functions ϕ_i s.t.*

$$k(x, x') = \sum_{i=1}^{\infty} \lambda_i \phi_i(x) \phi_i(x').$$

8.6 Examples of kernels

- Linear kernel: $k(\mathbf{x}, \mathbf{x}') = \mathbf{x}^T\mathbf{x}'$
- Polynomial kernel: $k(\mathbf{x}, \mathbf{x}') = (\mathbf{x}^T\mathbf{x}' + 1)^d$
- Gaussian (RBF, squared exponential kernel): $k(\mathbf{x}, \mathbf{x}') = \exp(-\left\|\mathbf{x} - \mathbf{x}'\right\|_2^2)/h^2$ where h^2 is the bandwidth/length scale parameter.
- Laplacian kernel: $k(\mathbf{x}, \mathbf{x}') = \exp(-\left\|\mathbf{x} - \mathbf{x}'\right\|_1 / h)$

8.7 Examples of non-kernels

- $k(\mathbf{x}, \mathbf{x}') = \sin(\mathbf{x})cos(\mathbf{x}')$. It is not symmetric. Take for instance $x = 0$ and $x' = \frac{\pi}{2}$
- $k(\mathbf{x}, \mathbf{x}') = \mathbf{x}^T M \mathbf{x}' \forall \mathbf{x}, \mathbf{x}' \in \mathbb{R}^d, M \in \mathbb{R}^{d \times d}$

  Proof. If M is symmetric: $k(\mathbf{x}, \mathbf{x}') = \mathbf{x}^T M^T \mathbf{x}' = \mathbf{x}'^T M^T \mathbf{x} = k(\mathbf{x}', \mathbf{x})$. If M is not symmetric, k in general is not symmetric. If M is not positive semi-definite, then the kernel function is not definite, e.g. for the situation where $M = -1$ for the normal dot product. If M is positive semi-definite: $M = UD^{\frac{1}{2}}D^{\frac{1}{2}}TU^T = V^TV$ for $V = (UD^{\frac{1}{2}})^T$ then: $k(\mathbf{x}, \mathbf{x}') = \mathbf{x}^T M \mathbf{x}' = \mathbf{x}^T V^T V \mathbf{x} =)(V\mathbf{x})^T(V\mathbf{x}') = \phi(\mathbf{x})^T\phi(\mathbf{x}')$ for $\phi(\mathbf{x}) = V\mathbf{x}$ □

8.8 Effect of kernel on function class

Given a kernel k, predictors for kernelized classification have the form:

$$\hat{y} = \text{sign}(\sum_{j=1}^{n} \alpha_j y_j k(\mathbf{x}_j, \mathbf{x}))$$

Grafically, it looks like in figure 3.

8.9 Graphical representations of kernels

The graphical representation of a sample gaussian and exponential kernel can be found in figures

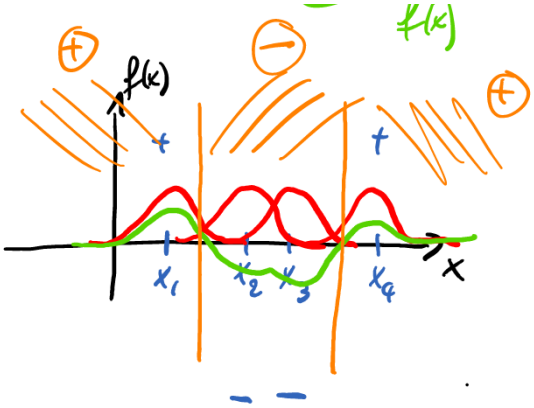


Figure 3: This figure shows that the green line is nothing but a scaled version of the indidiual Gaussian distributions and the associated decision function sign.

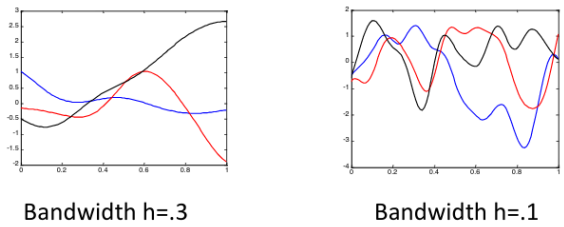


Figure 4: Graphical representation of the Gaussian kernel.

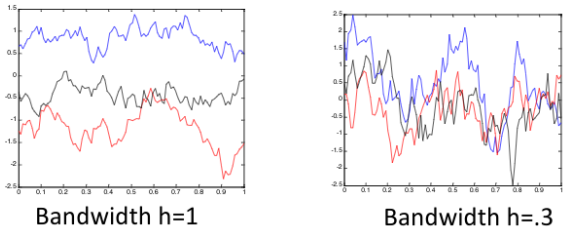


Figure 5: Graphical representation of the Laplacian kernel.

8.10 Objects where kernels can be used

The kernels can be defined on a variety of objects:

- Sequence kernels
- Graph kernels
- Diffusion kernels
- Kernels on probability distributions

Graph kernels can be used for measuring similarity between graphs by comparing random walks on both graphs. They can also be used to measure similarity among nodes in a graph via diffusion kernels not defined here.

8.11 Kernel engineering

Suppose we have two kernels:

$$k_1 : \mathcal{X} \times \mathcal{X} \to \mathbb{R} \qquad k_2 : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$$

Then the following functions are valid kernels:

- $k(\mathbf{x}, \mathbf{x}') = k_1(\mathbf{x}, \mathbf{x}') + k_2(\mathbf{x}, \mathbf{x}')$
- $k(\mathbf{x}, \mathbf{x}') = k_1(\mathbf{x}, \mathbf{x}')k_2(\mathbf{x}, \mathbf{x}')$
- $k(\mathbf{x}, \mathbf{x}') = ck_1(\mathbf{x}, \mathbf{x}')$ for $c > 0$
- $k(\mathbf{x}, \mathbf{x}') = f(k_1(\mathbf{x}, \mathbf{x}'))$ where f is a polynomial with positive coefficients or the exponential function.

8.12 The ANOVA kernel

$$k(\mathbf{x}, \mathbf{x}') = \sum_{j=1}^{d} k_j(x_j, x'_j)$$

where:

$$\mathbf{x}, \mathbf{x}' \in \mathbb{R}^d \qquad k : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R} \qquad k_j : \mathbb{R} \times \mathbb{R} \to \mathbb{R}$$

The functions modelled by this kernel can be shown by showing what forms $f(x)$ takes:

$$\begin{aligned} f(x) &= \sum_{i=1}^{n} \alpha_i y_i k(x^(i), \mathbf{x}) \\ &= \sum_{i=1}^{n} \alpha_i y_i \sum_{j=1}^{d} k_j(x^(i)_j, \mathbf{x}_j) \\ &= \sum_{j=1}^{d} \sum_{i=1}^{n} \alpha_i y_i k_j(x^(i)_j, \mathbf{x}_j) \\ &= \sum_{j=1}^{d} f_j(\mathbf{x}_j) \end{aligned}$$

which means that the function f decomposes into functions that depend on individual coordinates only, i.e. it is additive decomposition of the function. This property can be useful for high dimensional domains.

Method	k-NN	Kernelized perceptron
Advantages	No training required	Optimized weights can lead to improved performance, can capture global trends with suitable kernels
Disadvantages	Depends on all data	Depends on wrongly classified examples only Training requires optimisation

8.13 Modelling pairwise data

Suppose we have the following two kernels:

$$k((x,z),(x',z')) = k_x(x,x') \cdot k_z(z,z') \tag{10}$$
$$k((x,z),(x',z')) = k_x(x,x') + k_z(z,z') \tag{11}$$
$$\tag{12}$$

They can be used to represent the following data:

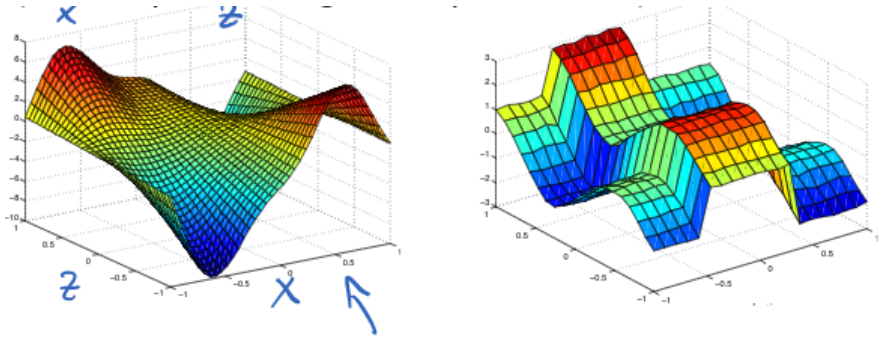


Figure 6: The left figure can be used to represent a situation where multiplying the kernel could be useful and the right figure can be used to represent a situation where adding the kernel could be useful.

8.14 Kernels as similarity functions

Kernels can be used as similarity measures. For instance, consider the Gaussian kernel $k(\mathbf{x},\mathbf{x}') = \exp(-\|\mathbf{x}-\mathbf{x}'\|^2/h^2)$. If a point $\mathbf{x}$ is close to $\mathbf{x}'$, then the value of $k(\mathbf{x},\mathbf{x}') \approx 1$, else it is closer to 0.

8.15 Comparing k-NN to kernel perceptron

The prediction for each point in k-NN is provided by:

$$y = \text{sign}(\sum_{i=1}^{n} y_i[\mathbf{x} \text{ among } k \text{ nearest neighbors of } \mathbf{x}])$$

As we can see, it compares to the loss of the perceptron:

$$y = \text{sign}(\sum_{i=1}^{n} y_i \alpha_i k(\mathbf{x}_i, \mathbf{x}))$$

Note: choose k in k-NN using cross-validation.

8.16 Deriving non-parametric models from parametric ones

Parametric models have a finite set of paramteters. Examples of such models include linear regression, linear perceptron, etc ... Nonparametric models grow in complexity with the size of the data. They have the potential to be much more expressive but also more computationally complex. Examples of such models include the kernelized perceptron, k-NN, etc... Kernels provide a principled way of deriving non-parametrics models from parametric ones.

8.17 Kernelized SVM

The SVM optimization step can be kernelized as follows:

$$\begin{aligned}
\hat{\mathbf{w}} = \arg\min_{\mathbf{w}} \frac{1}{n}\sum_{i=1}^{n} \max\left\{0, 1 - y_i \mathbf{w}^T \mathbf{x}_i\right\} + \lambda \left\|\mathbf{w}\right\|_2^2 \qquad \text{assuming } \mathbf{w} = \sum_{j=1}^{n} \alpha_j y_j x_j \\
&= \max\left\{0, 1 - y_i (\sum_{j=1}^{n} \alpha_j y_j x_j)^T x_i\right\} \\
&= \max\left\{0, 1 - y_i \sum_{j=1}^{n} \alpha_j y_j (x_j^T x_i)\right\} \\
&= \max\left\{0, 1 - y_i \sum_{j=1}^{n} \alpha_j y_j k(x_j, x_i)\right\} \\
&= \max\left\{0, 1 - y_i \alpha^T k_i\right\}
\end{aligned}$$

where $k_i = [y_1 k(\mathbf{x}_i, \mathbf{x}_1), \ldots, y_n k(\mathbf{x}_i, \mathbf{x}_n)]^T$ and $\alpha = [\alpha_1, \ldots, \alpha_n]^T$.

The regularizer can be kernelized as follows:

$$\begin{aligned}
&\lambda \left\|\mathbf{w}\right\|_2^2 \\
=&\lambda \mathbf{w}\mathbf{w}^T = \lambda (\sum_{i=1}^{n} \alpha_i y_i x_i)^T (\sum_{j=1}^{n} \alpha_j y_j x_j) = \qquad \lambda \sum_{i=1}^{n}\sum_{j=1}^{n} \alpha_i \alpha_j y_i y_j x_j^T x_i \\
=&\lambda \alpha^T D_y \mathbf{K} D_y \alpha
\end{aligned}$$

where $D_y = \begin{bmatrix} y_1 & \ldots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \ldots & y_n \end{bmatrix}$ and $\mathbf{K} = \begin{bmatrix} k(x_1, x_1) & \ldots & k(x_1, x_n) \\ \vdots & \ddots & \vdots \\ k(x_n, x_1) & \ldots & k(x_n, x_n) \end{bmatrix}$

8.18 Kernelizing linear regression

Original parametric linear regresion optimization problem is stated as follows:

$$\hat{\mathbf{w}} = \arg\min_{\mathbf{w}} \frac{1}{n}\sum_{i=1}^{n} (\mathbf{w}^T \mathbf{x}_i - y_i)^2 + \lambda \left\|\mathbf{w}\right\|_2^2$$

We kernelize linear regression in two parts:

$$
\begin{aligned}
&= \sum_{i=1}^{n} (\mathbf{w}^T \mathbf{x}_i - y_i)^2 \\
&= \sum_{i=1}^{n} ((\sum_{j=1}^{n} \alpha_j x_j x_i)^T \mathbf{x}_i - y_i)^2 \\
&= \sum_{i=1}^{n} (\sum_{j=1}^{n} \alpha_j (x_j^T x_i) \mathbf{x}_i - y_i)^2 \\
&= \sum_{i=1}^{n} (\sum_{j=1}^{n} \alpha_j (x_j^T x_i) \mathbf{x}_i - y_i)^2 \\
&= (\alpha^2 k_i - y_i)^2
\end{aligned}
$$

and

$$
\begin{aligned}
&= \sum_{i=1}^{n} \sum_{j=1}^{n} \alpha_i \alpha_j x_i^T x_j \\
&= \alpha^T \mathbf{K} \alpha
\end{aligned}
$$

hence we have:

$$\hat{\alpha} = \arg\min_{\alpha \in \mathbb{R}^n} \frac{1}{n} \sum_{i=1}^{n} (\alpha^T k_i - y_i)^2 + \lambda \alpha^T \mathbf{K} \alpha$$

$$\hat{\alpha} = \arg\min_{\alpha \in \mathbb{R}^n} \frac{1}{n} \left\| \alpha^T \mathbf{K} - \mathbf{y} \right\|_2^2 + \lambda \alpha^T \mathbf{K} \alpha$$

which has a closed-form solution:

$$\hat{\alpha} = (\mathbf{K} + n\lambda \mathbf{I})^{-1} \mathbf{y}$$

For prediction, given a data point $\mathbf{x}$, we predict the response y as:

$$\hat{y} = \sum_{i=1}^{n} \hat{\alpha}_i k(\mathbf{x_i}, \mathbf{x})$$

8.18.1 Application: semi-parametric regression

Often, parametric models are too rigid and non-parametric models fail to extrapolate. The solution to this problem is to use additive combination of linear and non-linear kernel function as shown below:

$$k(\mathbf{x}, \mathbf{x}') = c_1 \exp\left(\left\| \mathbf{x} - \mathbf{x}' \right\|_2^2 / h^2 \right) + c_2 \mathbf{x}^T \mathbf{x}'$$

The decision function is then defined as follows:

$$
\begin{aligned}
f(\mathbf{x}) = \sum_{i=1}^{n} \alpha_i k(\mathbf{x}_i, \mathbf{x}) &= \sum_{i=1}^{n} \left(\alpha_i (c_i \exp(- \left\| \mathbf{x}_i - \mathbf{x} \right\|_2^2 / h^2)) + c_2 \mathbf{x}_i \mathbf{x}' \right) \\
= c_1 \sum_{i=1}^{n} \alpha_i k(\mathbf{x}_i, \mathbf{x}) &= \sum_{i=1}^{n} \left(\alpha_i (c_1 \exp\left(\left\| \mathbf{x_i} - \mathbf{x}_2^2 / h^2 \right\| \right)) + c_2 \mathbf{x}_i^T \mathbf{x}' \right) \\
= c_1 \sum_{i=1}^{n} \alpha_i \exp\left(- \left\| \mathbf{x}_i - \mathbf{x} \right\|_2^2 / h^2 \right) + \left(c_2 \sum_{i=1}^{n} \alpha_i x_i \right)^T \mathbf{x} &= f_1(\mathbf{x}) + \mathbf{w}^T \mathbf{x}
\end{aligned}
$$

The efficiency of this methodology can be seen in the figure 7.

This approach can be used to design P450 chimeras, predict protein fitness landscapes, and various other protein engineering applications.

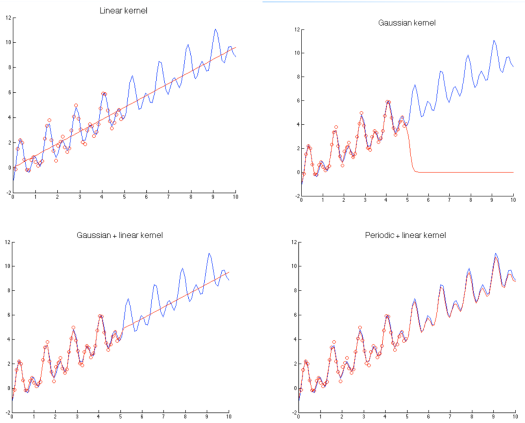


Figure 7: Various learned models overlayed with the function that we wish to learn based on the available data. Clearly, a linear kernel and a periodic parametric model seem to fit best the pattern observed.

8.19 Choosing kernels & risk of overfitting

Choosing correct kernels is a combination of domain knowledge, brute force or heuristic search using cross-validation. Since kernels map to very high dimensional spaces, it is difficult to see what we hope to be able to learn. Typically the number of parameters is drastically smaller than the number of dimensions. One way of tackling this problem is to set the number of parameters equal to that of the number of datapoints, which in this case would be called non-parametric learning. Another way of tackling this problem is to use regularization, which is built into kernelized linear regression and SVMs but not into the kernelized Perceptron. Recall the formulations of KLR and SVM:

$$\hat{\alpha} = \arg\min_{\alpha} \frac{1}{n} \left\| \alpha^T \mathbf{K} = \mathbf{y} \right\|_2^2 + \lambda \alpha^T \mathbf{K} \alpha$$

$$\hat{\alpha} = \arg\min_{\alpha} \frac{1}{n} \sum_{i=1}^{n} \max\left\{0, 1 - y_i \alpha^T \mathbf{k}_i\right\} + \lambda \alpha^T \mathbf{D_y K D_y} \alpha$$

9 Class imbalance

Often data looks very imbalanced, e.g. there are more false negatives than false positives, see figure 8. Sources of imbalanced data include fraud detection datasets, spam filtering, process monitoring, medical diagnosis, feedback in recommender systems. The main issues related with imbalanced data include *performance metrics*, where the performance assessment of the model is not a good metric. It may be good to prefer certain mistakes over others, i.e. we trade false positives for false negatives. The minority class instance contribute little to the empirical risk. It therefore may be ignored during optimization. There are two solutions to the problem of imbalanced datasets:

- **Subsampling**: this entails removing examples from the majority class (e.g. uniformly at random) such that the resulting dataset is balanced.
- **Upsampling**: this entails repeating data points from the minority class, possibly with small random perturbation to obtain a balanced data set.
- It's also a possibility to use cost-sensitive classification methods to deal with these issues.

Illustrations of these two approaches are shown in figure 9.

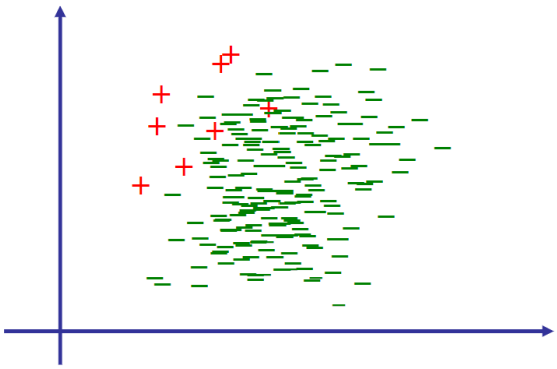


Figure 8: Example of a dataset with imbalanced data.

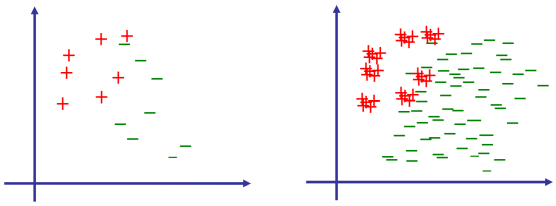


Figure 9: Illustration of subsampling (left) and upsampling (right).

9.1 Cost-sensitive classification

This entails modifying the perceptron/SVM to take class balance into account. The only different is in the cost function, where we add a class coefficient in front of the loss function:

$$\ell_{CS}(\mathbf{w}; \mathbf{x}, y) = c_y \ell(\mathbf{w}; \mathbf{x}, y)$$

For the perceptron, this becomes

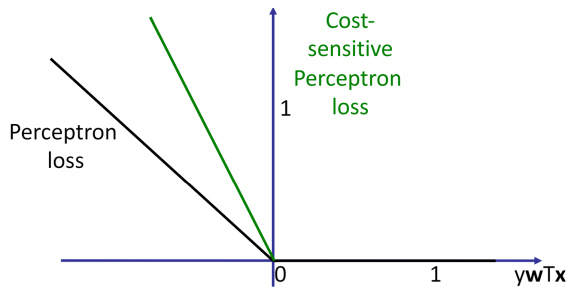
$$\ell_{CS}(\mathbf{w}; \mathbf{x}, y) = c_y \max(0, -y\mathbf{w}^T\mathbf{x})$$

For the SVM, this becomes

$$\ell_{CS}(\mathbf{w}; \mathbf{x}, y) = c_y \max(0, 1 - y\mathbf{w}^T\mathbf{x})$$

where, in all cases, $c_+, c_- > 0$ control the tradeoff to be made.

An illustration of the adapted cost function can be seen in figure 10.



$$\ell(\mathbf{w}; \mathbf{x}, y) = c_y \max(0, -y\mathbf{w}^T\mathbf{x})$$

Figure 10: Perceptron loss

9.2 Avoiding redundancy

Given that the empirical loss, given by:

$$\hat{R}(\mathbf{w}; c_+, c_-) = \frac{1}{n} \sum_{i_{y_i=1}} c_+ \ell(\mathbf{w}; x_i, y_i) + \frac{1}{n} \sum_{i_{y_i=1}} c_- \ell(\mathbf{w}; x_i, y_i)$$

$$\forall \alpha > 0 : \hat{R}(\mathbf{w}; \alpha c_+, \alpha c_-) = \alpha \hat{R}(\mathbf{w}; c_+, c_-)$$

$$\Rightarrow \arg\min_{\mathbf{w}} \hat{R}(\mathbf{w}; \alpha c_+, \alpha c_-) = \arg\min_{\mathbf{w}} \alpha \hat{R}(\mathbf{w}; c_+, c_-) = \arg\min_{\mathbf{w}} \alpha \hat{R}(\mathbf{w}; \frac{c_+}{c_-}, 1)$$

$$\Rightarrow \text{w.l.o.g.: } \alpha = \frac{1}{c_-}.$$

Which basically means that the tradeoff to make for imbalanced datasets can be determined by one parameter.

9.3 Metrics for imbalanced datasets

A generally inappropriate metric for imbalanced datasets is accuracy, defined as

$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN} = \frac{TP + TN}{N}$$

Other, more suitable metrics include:

$$\text{Precision} = \frac{TP}{TP + FP} = \frac{TP}{P+}$$

$$\text{Recall} = \text{True positive rate (TPR)} = \frac{TP}{TP + FN} = \frac{TP}{N+}$$

$$\text{F1 score} = \frac{2TP}{2TP + FN + FP} = \frac{2}{\frac{1}{\text{Prec}} + \frac{1}{\text{Rec}}}$$

$$\text{False positive rate (FPR)} = \frac{FP}{TN + FP}$$

An interesting observation to make is that given the probability p of observing a positive sample, we have:

$$\mathbb{E}[\text{TPR}] = \frac{\mathbb{E}[\text{TP}]}{n_+} = \frac{p \cdot n_+}{n_+} = p$$

$$\mathbb{E}[\text{TPR}] = \frac{\mathbb{E}[\text{TP}]}{n_-} = \frac{p \cdot n_-}{n_-} = p$$

The receiver operator characteristic (ROC) curve (see figure 11) can also be used to make an assessment of a classifier.

Note that a performance measure associated to the precision recall curve is the area under the curve. A random classifier has an AUC of 0.5 and an ideal classifier has an AUC of 1.s

It is also important that the confusion matrix itself is also often used to evaluate the performance of a binary or even multi-class classifier.

9.4 Obtaining the optimal tradeoff

Two ways of obtaining a good tradeoff is to either use a cost sensitive classifier and vary the tradeoff parameter. The second option is to find a single classifier and vary the classification threshold τ in the following formula:

$$y = \text{sign}(\mathbf{w}^T \mathbf{x} - \tau)$$

The optimal value for the threshold can be determined using the precision-recall curve 12.

An interesting note to make in terms of the relationship between the ROC curve and the precision recall curve is captured in the following theorem:

Theorem 3. *Algorithm 1 dominates Algorithm 2 in terms of ROC curve ⇔ Algorithm 1 dominates Algorithm 2 in terms of precision recall curve.*

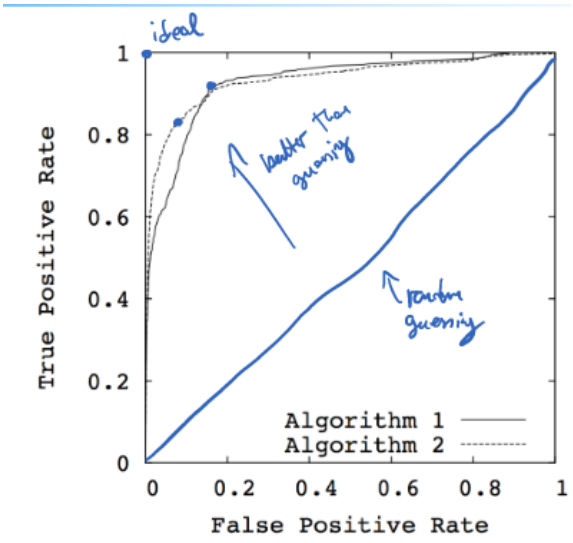


Figure 11

Prediction outcome

actual value	**p**	**n**	**total**
p′	True Positive	False Negative	P′
n′	False Positive	True Negative	N′
total	P	N	

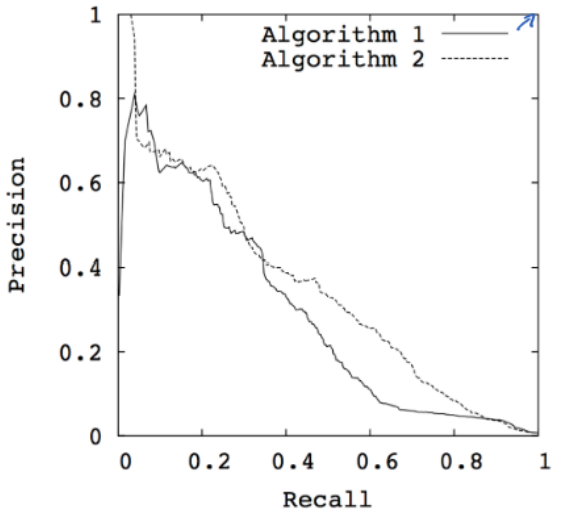


Figure 12: An example of a precision-recall curve.

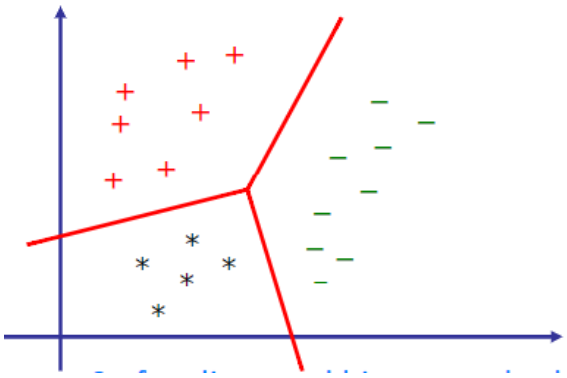


Figure 13: Illustration of a multiclass problem.

10 Multiclass problems

A multiclass problem is defined as follows. Given a dataset $\mathcal{D} = (\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_n, y_n)$ for $y_i \in 1, \ldots, c$, we want a function $f : \mathcal{X} \to \mathcal{Y}$ where $y_i \in \mathcal{Y} = 1, \ldots, c$ and $\mathbf{x}_i \in \mathcal{X} \subseteq \mathbb{R}^d$, as illustrated in figure 13.

10.1 One-vs-all

One approach to solve the problem is to solve c binary classifiers, i.e. one for each class, where all positive samples are from class i and all negative samples represent all other points. We then classify using the classifier with the largest confidence. In other words, we fit $f^{(i)} : \mathcal{X} \to \mathbb{R}$ and $f^{(i)}(x) = \mathbf{w}^{(i)T}\mathbf{x}$ and we predict $\hat{y} = \arg\max_i f^{(i)}(\mathbf{x}) = \arg\max_i \mathbf{w}^{(i)T}\mathbf{x}$.

10.1.1 Confidence in a classification

The confidence in a classification decision can be visualized as in figure 14. For $\alpha > 0$, we have

$$\text{sign}((\alpha\mathbf{w})^T\mathbf{x}) = \text{sign}(\mathbf{w}^T\mathbf{x})$$

Thus although $\alpha\mathbf{w}$ and $\mathbf{w}$ implement the same decision boundary, they have a different confidence level. There are two solutions to this problem:

1. $\mathbf{w} \leftarrow \frac{\mathbf{w}}{\|\mathbf{w}\|_2}$, i.e. this is normalized to unit length.
2. In practice, when using regularization, the magnitude of $\|\mathbf{w}\|_2$ is kept under control.

The decision boundary of the one-vs-all decision can be represented as in figure 15.

10.1.2 Challenges

There are several challenges associated with the one-vs-all technique:

- It only works if classifiers produce confidence scores on the same scale.
- Individual binary classifier see imbalanced data, even if the whole data set is balanced.
- One class might not be linearly separable from all other classes.

10.2 One-vs-one

The idea is to train $c(c-1)/2$ binary classifiers, one for each pair of classes (i, j). Positive examples contain all points from class i and negative examples contain all points from class j. Then, we apply a voting scheme, i.e. the class with the highest number of positive prediction wins. A graphical representation of the one-vs-one classification is shown in figure 16.

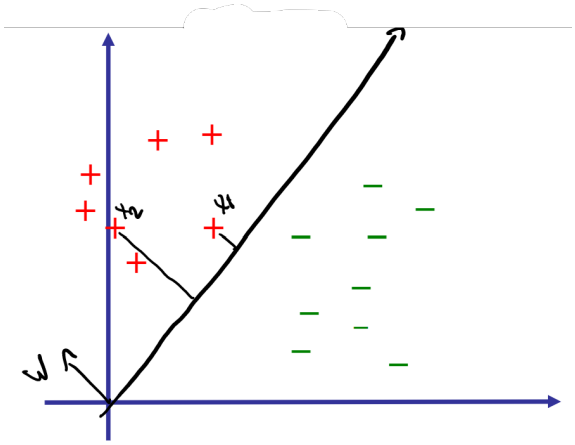


Figure 14: Illustration of the confidence in classification. We see that $\mathbf{w}^T\mathbf{x}_2 > \mathbf{w}^T\mathbf{x}_1$ so that means that the label $+$ is given with more confidence to $\mathbf{x}_2$ than for $\mathbf{x}_1$

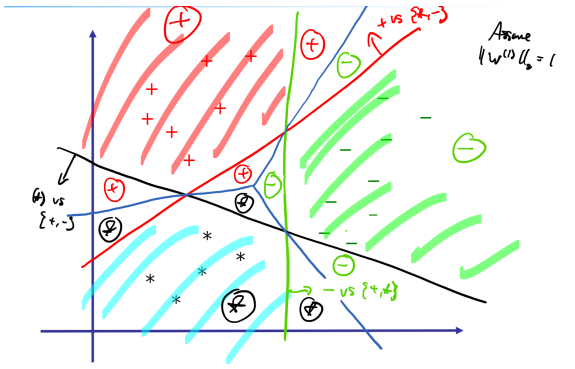


Figure 15: One-vs-all decision boundary.

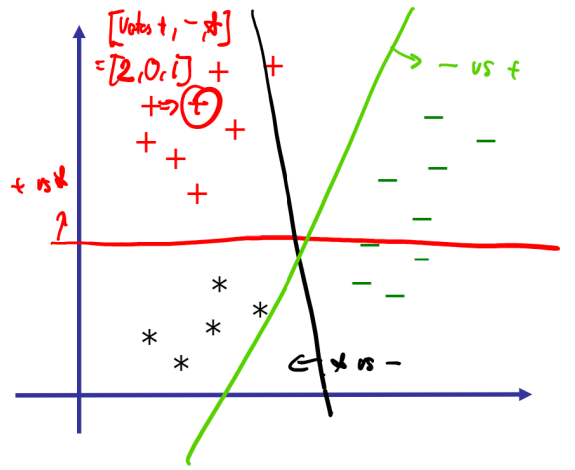


Figure 16: Illustration of the one-vs-one classification technique. Here, we see that each decision boundary is fit for each pair of classes (i, j).

Method	*One-vs-all*	*One-vs-one*
Advantages	Only c classifiers needed (faster!)	No confidence needed
Disadvantages	Requires confidence + leads to class imbalance	Need to train $c(c-1)/2$)

Class	*Encoding*
0	[0,0,0,0]
1	[0,0,0,1]
2	[0,0,1,0]
⋮	⋮
$c-1$	[1,1,1,1]

Table 1: Table containing the class name and corresponding encoding

10.3 Encodings

There are other methods to tackle multiclass problems, e.g. using other encodings (error correcting output codes). There are also other explicit multiclass models, such as the multi-class perceptron/svm etc. Some models are naturally multi-class, such as nearest neighbors, generative probabilistic models, etc. However, one should keep in mind that one-vs-all/one-vs-one usually works very well.

A way to reduce the number of classes we need to work with can be reduce by using an encoding. The length of the encoding will then scale with the $log_2 c$, see table 1. Then in principle the multiclassfication problem is phrased as a decoding task of the class label, where each classifier predicts one bit, which in this case we would get away with $\mathcal{O}(\log c)$ classifiers. In this case, ideas from coding theory to do multi-class classification can be used to do multi-class classification.

10.4 Multi-class SVMs

The key idea of multi-class SVMs is to maintain c weight vectors $\mathbf{w}^1, \dots, \mathbf{w}^c$, one for each class. Then we predict using:

$$\hat{y} \leftarrow \arg \max_{i \in 1,\dots,c} \mathbf{w}^{(i)T}\mathbf{x}$$

Given each data point $(\mathbf{x}, y)$, we want to achieve that:

$$\mathbf{w}^{(y)}\mathbf{x} \geq \mathbf{w}^{(i)}\mathbf{x} + 1 \forall i \in \{1, \dots, c\} \setminus y \tag{13}$$

$$\equiv \mathbf{w}^{(y)}\mathbf{x} \geq \max_{i \in \{1,\dots,c\} \setminus \{y\}} \mathbf{w}^{(i)}\mathbf{x} + 1 \tag{14}$$

10.5 Multi-class Hinge loss

$$\ell_{MC-H}(\mathbf{w}^1, \dots, \mathbf{w}^c; \mathbf{x}, y) = \max(0, 1 + \max_{j \in \{1,\dots,y-1,y+1,\dots,c\}} \mathbf{w}^{(j)T}\mathbf{x} - \mathbf{w}^{(y)T}\mathbf{x})$$

which is equal to 0 when equation 13 is satisfied.

The gradient of the Hinge loss is then defined as:

$$\nabla_{w^{(i)}} \ell_{MC-H}(\mathbf{w}^{(1:c)}; \mathbf{x}, y) = \begin{cases} 0 \text{ if equation 13 is satisfied or } i \neq y \wedge i \neq \arg\max_j \mathbf{w}^{(j)T}\mathbf{x} \\ -\mathbf{x} \text{ if 13 is not satisfied and } i = y \\ +\mathbf{x} \text{ otherwise.} \end{cases}$$

11 Neural networks

11.1 Importance and characteristics of features

Succes in learning crucially depends on the quality of features. Hand-designing features however requires domain knowledge. However, kernels could be used to engineer features as they provide a rich set of feature maps, they can fit almost any function with infinite data. However, choosing the right kernel can be challenging and the computational complexity grows exponentially with

the size of the data. The question then becomes whether or not we can learn good features from the data directly. The overall objective function for neural networks to optimize is the following:

$$\mathbf{w}* = \arg\min_{\mathbf{w}} \sum_{i=1}^{n} \ell(y_i; \sum_{j=1}^{m} w_j \phi_j(\mathbf{x}_j)) = \arg\min_{\mathbf{w}} \sum_{i=1}^{n} \ell(y_i; f_i)$$

where $\ell(y_i, f_i)$ is usually defined as $\ell(y_i, f_i) = (y_i - f_i)^2$ (squared loss). A key idea then is to parametrize the feature maps, and optimize over the parameters, i.e. our optimization objective is to tune both the parameters and the weights to minimize the loss:

$$\mathbf{w}* = \arg\min_{\mathbf{w},\theta} \sum_{i=1}^{n} \ell(y_i; \sum_{j=1}^{m} w_j \phi(\mathbf{x}_i, \theta))$$

One possibility of achieving this optimization is to define the following feature map:

$$\phi(\mathbf{x}, \theta) = \varphi(\theta^T \mathbf{x})$$

where $\theta \in \mathbb{R}^d$ and $\phi : \mathbb{R} \to \mathbb{R}$ is a nonlinear function, called an *activation function*. An example of such an activation function is the sigmoid and tanh activation functions. The signmoid function is given by:

$$\varphi(z) = \frac{1}{1 + \exp(-z)}$$

The tanh function is given by:

$$\varphi(z) = \tanh(z) = \frac{exp(z) - \exp(-z)}{exp(z) + \exp(-z)}$$

The rectified linear unit is also often used, which is defined as:

$$\phi(z) = \max(z, 0)$$

The decision function of an artificial neural network is of the form:

$$f(\mathbf{x}_i; \mathbf{w}, \theta) = \sum_{j=1}^{m} w_j \varphi(\theta_j^T \mathbf{x}) = \sum_{j=1}^{m} w_j v_j$$

More generally, the term artificial neural network refers to nonlinear functions which are nested compositions of variable linear functions composed with fixed nonlinearities.

11.2 Graphical representation

A neural network can be represented as shown in figure 17 Note that neural networks can have multiple outputs for multi-class prediction (one output per class or multi-output regression). They can also have more than one hidden layer – networks with large amounts of neural networks with several hidden layers are called deep neural networks. Indexing in neural networks is done as shown in figure 18.

11.3 Making predictions using ANNs

Suppose we have learned all paramters $w_{i,j}$. Given an input, we now make predictions using forward propagation. The forward propagation procedure is described as follows:

1. For each unit j on the input layer, we set its value to $v_j = x_j$.
2. For each layer $\ell = 1 : L - 1$
   (a) For each unit j on layer ℓ we set its value:

$$v_j = \varphi\left(\sum_{i \in \text{Layer}_{\ell-1}} w_{j,i} v_i\right)$$

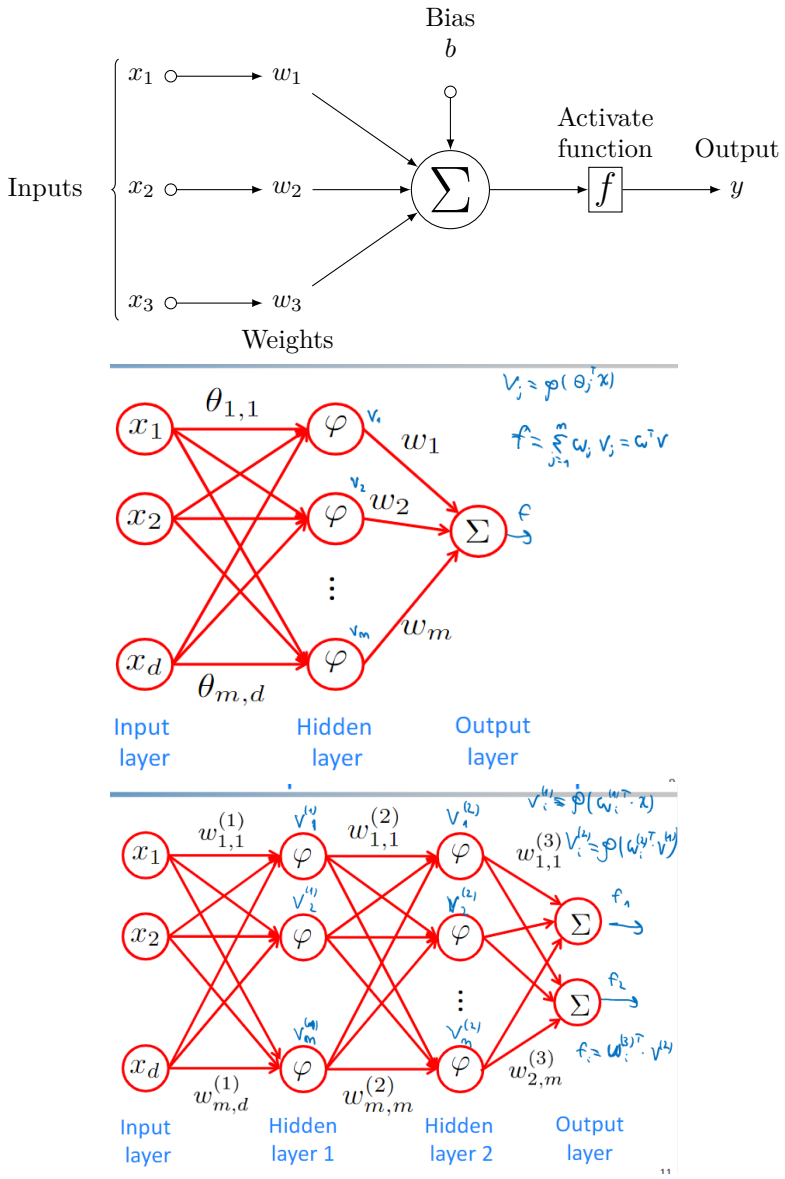


Figure 17: Graphical representations of an artificial neural network.

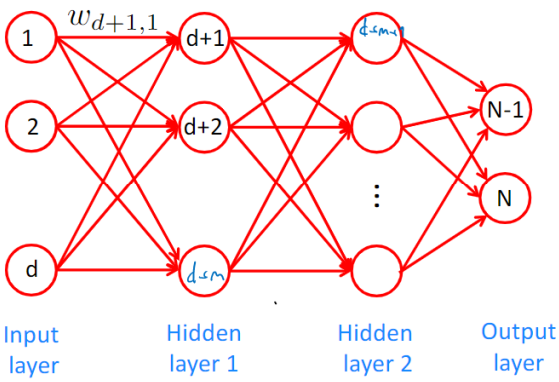


Figure 18: Indexing convention of ANNs

3. For each unit j on output layer, set its value:

$$f_j = \sum_{i \in \text{Layer}_{L-1}}^{w_{j,i} v_i}$$

4. Predict $y_j = f_j$ for regression, and $y_j = \text{sign}(f_j)$ for regression. For multiclass problems, the prediction is made as $\hat{y} = \arg\max_j f_j$.

Another way of writing the forward propagation algorithm is as follows:

1. For input layer: $\mathbf{v}^{(0)} = \mathbf{x}$
2. For each layer $\ell = 1 : L - 1$

$$\mathbf{z}^{(\ell)} = \mathbf{W}^{(\ell)} \mathbf{x}^{(\ell-1)}, \mathbf{z}^{(\ell)} \in \mathbb{R}^{m^{(d)}}$$
$$\mathbf{v}^{(\ell)} = \varphi(\mathbf{z}^{(\ell)})$$

   where $m^{(d)}$ is the number of units in the ℓ^{th} layer. and $\varphi(\mathbf{z}^{(\ell)}) = [\varphi(\mathbf{z}_1^{(\ell)}), \varphi(\mathbf{z}_2^{(\ell)}), \ldots, \varphi(\mathbf{z}_m^{(\ell)})]$.

3. For the output layer: $f = \mathbf{W}^L \mathbf{v}^{L-1}$
4. Predict $y = f$ for regression, and $y = \text{sign}(f)$ for regression. For multiclass problems, the prediction is made as $\hat{y} = \arg\max_i f_i$.

11.4 Universal approximation theorem

Neural networks are powerful modelling tools, because any decision function can be modelled by a sufficiently complex artificial networ'k

Theorem 4 (Universal approximation theorem)**.** *Let σ be any continuous sigmoidal function. Then finite sums of the form:*

$$G(x) = \sum_{j=1}^{N} \alpha_j \sigma(y_j^T x + \sigma_j)$$

are dense in $C(I_n)$. In other words, given any $f \in C(I_n)$ and $\epsilon > 0$, there is a sum, $G(x)m$ of the above form, for which:

$$|G(x) - f(x)| < \epsilon \text{ for all } x \in I_n$$

11.5 Training the weights of a neural network

Given a dataset $\mathcal{D} = (\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_n, y_n)$, we optimize the weights $\mathbf{W} = (\mathbf{W}^{(1)}, \ldots, \mathbf{W}^{(L)})$ y applying a loss function (e.g. the perceptron loss, the multi-class hinge loss, square loss, etc.) to output:

$$\ell(\mathbf{W}; \mathbf{y}, \mathbf{x}) = \ell(\mathbf{y}; f(\mathbf{x}, \mathbf{x})).$$

Then, we optimize the weights to minimise the loss over $\mathcal{D}$:

$$\mathbf{W}* = \arg\min_{\mathbf{W}} \sum_{i=1}^{n} \ell(\mathbf{W}; \mathbf{y}_i, \mathbf{x}_i)$$

When predicting multiple outputs at the same time, we usually define the loss as a sum of per-output losses as follows:

$$\ell(\mathbf{W}; \mathbf{y}, \mathbf{x}) = \sum_{i=1}^{p} \ell_i(\mathbf{W}; y_i, \mathbf{x}).$$

For regression tasks we usually use the squared loss and for classification we use the perceptron or the hinge loss.

Jointy optimizng over all weights for all layers as described in 11.5 is, in general, a non-convex optimization problem. We can nevertheless try to find a local optimum.

This local optimum can be found, for instance, using stochastic gradient descent, described as follows:

1. Initialize the weights $\mathbf{W}$.
2. For $t = 1, 2, \ldots$
   (a) Pick data point $(\mathbf{x}, \mathbf{y}) \in \mathcal{D}$ uniformly at random.
   (b) Take a step in negative gradient direction:

$$\mathbf{W} \leftarrow \mathbf{W} - \eta_t \nabla_{\mathbf{W}} \ell(\mathbf{W}, \mathbf{y}, \mathbf{x})$$

11.6 Deep learning

Generally, deep learning refers to models with nested, layered non-linearities. Common examples of deep neural networks include:

1. Classical ANN with multiple layers
2. Trained via SGD and variants
3. Some new algorithmic insights (e.g. dropout regularization) and exntensions (Convnets, resnets, RNNs, LSTMs, GRUs,...)

Some deep learning success stories include the fact that deep neural networks achieve state of the art performance on some difficult classification tasks such as speech recognition, image recognision, natural language processing and speech translation. A lot of recent work on sequential models such as RNNs, LSTMs, GRUs,... have also proven to model data effectively.

Some crucial questions to be answered when training a deep neural network include the following:

- How can we compute the gradients?
- How should we initialize the weights?
- When should we terminate?
- How do we choose parameters (number of units/layers/activation functions/learning rate/...)?
- What about overfitting?

11.7 Computing the gradient

In order to apply SGD, we need to compute:

$$\nabla_{\mathbf{W}} \ell(\mathbf{W}; \mathbf{y}, \mathbf{x})$$

i.e. for each weight between any two connected units i and j, we need to compute:

$$\frac{\partial}{\partial w_{i,j}} \ell(\mathbf{W}; \mathbf{y}, \mathbf{x})$$

In a simple example, we have an ANN with 1 output, 1 hidden and 1 input unit: $\mathbf{W} = [\mathbf{w}, \mathbf{w}']$ as shown in figure 19. In this example we have that:

$$f(x, \mathbf{W} = w\varphi(w'x)$$

and a dataset containing 1 datapoint containing $\mathcal{D} = (x, y)$. Then the loss is defined as:

$$L(\mathbf{w}', \mathbf{w}) : L = \ell(f, y) = \ell_y(f)$$

Note that $\ell_y(f)$ is computed using forward propagation. Differentiating the loss function w.r.t. the weights gives:

$$\frac{\partial L}{\partial \mathbf{w}} = \frac{\partial L}{\partial f} \frac{\partial f}{\partial \mathbf{w}} = \ell_y'(f) \cdot v$$
$$\frac{\partial L'}{\partial \mathbf{w}} = \frac{\partial L}{\partial f} \frac{\partial f}{\partial \mathbf{v}} \frac{\partial \mathbf{v}}{\partial \mathbf{w}'} = \delta \cdot \mathbf{w} \cdot \phi'(z) \cdot \mathbf{x}.$$

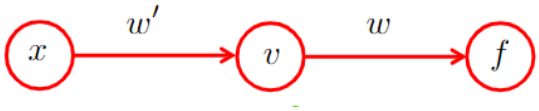


Figure 19: The computation graph of a simple ANN

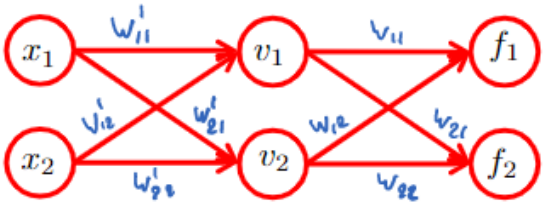


Figure 20: A more complex neural network.

for instance in the case of the square loss:

$$\ell_y(f) = (f-y)^2$$
$$\Rightarrow l'_y(f) = 2(f-y)$$

For a slightly more complex example, as depicted in figure 20, the loss is defined as follows:

$$L = \sum_{i=1}^{2} \ell_i(\sum_j w_{i,j}\varphi(\sum_k w'_{j,k}x_k)).$$

The weight vector $\mathbf{W}$ is defined as $\mathbf{W} = [(w_{i,j})_{i,j}, (w'_{i,j})_{i,j}]$. The loss is then defined as:

$$\frac{\partial L}{\partial w_{i,j}} = \frac{\partial L}{\partial f_i}\frac{\partial f_i}{\partial w_{i,j}} = \ell'_i(f_i)\cdot v_j$$

Note that $\ell'_i(f_i)$ and $\cdot v_j$ are computed using forward propagation.

$$\frac{\partial L}{\partial w'_{j,k}} = \sum_{i=1}^{2}\frac{\partial L}{\partial f_i}\frac{\partial f_i}{\partial v_j}\frac{\partial v_j}{\partial w'_{j,k}} = \sum_{i=1}^{2}\delta_i w_{i,j}\cdot\phi'(z_j)\cdot x_k$$

where $\delta_i = \frac{\partial L}{\partial f_i}$ and $\delta'_j = \delta_i w_{i,j}\cdot\phi'(z_j)\cdot x_k$

11.7.1 Derivatives of activation functions

Sigmoid:

$$\varphi(z) = \frac{1}{1+\exp(-z)}$$

$$\phi'(z) = \frac{-1}{(1+\exp(-z))^2}\cdot e^{-z}\cdot(-1) = \frac{e^{-z}}{1+e^{-z}}\cdot\frac{1}{1+e^{-z}} = \phi(z)(1-\phi(z))$$

The advantage of this function is that it is differentiable everywhere, however $\phi(z) \approx 0$ unless $z = \approx 0$. This leads to a problem that is called a vanishing gradient for deep models.

Rectified linear unit (ReLU):

$$\varphi(z) = \max(z, 0)$$

$$\varphi'(z) = \begin{cases} 1 \text{ if } z > 0 \\ 0 \text{ otherwise} \end{cases}$$

The disadvantage of this function is that it is not differentiable at 0, but in practice this is not a problem. A huge advantage is $\varphi'(z) = 1$ for all $z > 0$ which helps with avoiding vanishing gradients.

11.8 Backpropagation

The unvectorized version of the backpropagation algorithm is as follows:

1. For each unit j on the output layer
   - Compute the error signal $\delta_j = \ell'_j(f_j)$
   - For each unit i on layer L, compute $\frac{\partial}{\partial w_{i,j}} = \delta_j v_i$
2. For each unit j on hidden layer $\ell = L - 1 : -1 : 1$
   - Compute the error signal: $\delta_j = \phi'(z_j) \sum_{i \in \text{Layer}_{\ell+1}} w_{i,j}\delta_i$
   - For each unit i on layer ℓ, compute $\frac{\partial}{\partial w_{i,k}} = \delta_j v_i$

The vectorized implementation of the backpropagation algorithm can be phrased as follows:

1. For the output layer
   - Compute the error $\delta^L = \ell'(\mathbf{f}) = [\ell'(f_!), \dots, \ell'(f_p)]$
   - Gradient: $\nabla_{\mathbf{W}^{(L)}} \ell(\mathbf{W}; \mathbf{y}, \mathbf{x}) = \delta^{(L)} \mathbf{v}^{(L-1)T}$
2. For each hidden layer $\ell = L - 1 : -1 : 1$
   - Compute the error $\delta^{(\ell)} = \varphi'(\mathbf{z}^{(\ell)}) \odot (\mathbf{W}^{(\ell+1)T} \delta^{(\ell+1)})$:
   - Compute the gradient: $\nabla_{\mathbf{W}^{(\ell)}} \ell(\mathbf{W}; \mathbf{y}, \mathbf{x}) = \delta^{(\ell)} \mathbf{v}^{(\ell-1)T}$

11.9 Initializing weights

Because the optimization problem is non-convex, the initialization of the weights matters, i.e. if we pick innapropriate weights we are more likely to converge towards a suboptimal point in the optimization landscape. Random initialization is often preferred. The rationale of this comes from the propagation of variance. In order to understand why this is the case, assume a node $v_i^{(\ell)}$ with inputs $v_1^{(\ell-1)}, \dots, v_{n_{in}}^{(\ell-1)}$. We know that we carry out the computation $z_i^{(\ell)} = \sum_{j=1}^{n_{in}} w_{ij} v_j^{(\ell-1)} \Rightarrow v_i^{(\ell)} = \phi(z_i^{(\ell)})$ where $\phi(z) = \max(0, z)$.

11.9.1 Variance propagation

Now, we assume $\mathbb{E}[x_i] = \mathbb{E}[v_i^{(0)}] = 0$ where $v_i^{(0)}$ is the input layer and we also assume that $\text{Var}[x_i] = \text{Var}[v_i^{(0)}] = 1$. Let us further assume that the $X_1, \dots, X_d$ are independent. Additionally, we assume that the weights are draw from a normal distribution, i.e. $w_{ij} \sim \mathcal{N}(0, 1)$. Finally, we know that, supposing X, Y are independent with $\mathbb{E}[X] = 0$:

a if $\mathbb{E}[Y] = 0$, then $\text{Var}(X, Y) = \text{Var}(X)\text{Var}(Y)$

b if $\mathbb{E}[Y] \neq 0$, then $\text{Var}(X, Y) = \text{Var}(X)\mathbb{E}[Y^2]$

c For ReLU and symmetric Y: $\mathbb{E}[\phi(Y)^2] = \frac{1}{2}\text{Var}(Y)$

We then compute:

$$\mathbb{E}[z_i^{(1)}] = \mathbb{E}\left[\sum_{j=1}^{n_{in}} w_{i,j}x_j\right] = \sum_j \mathbb{E}[w_{ij}]\mathbb{E}[x_j] = 0$$
$$\mathrm{Var}[z_i^{(1)}] = \sum_j \mathrm{Var}[w_{ij}x_j] \overset{a}{=} \sum_j \mathrm{Var}(w_{ij}) \sum_j \mathrm{Var}(x_j) = n \cdot \sigma^2$$
$$\mathrm{Var}[z_i^{(1)}] = \sum_j \mathrm{Var}\left[w_{ij}v_j^{(\ell-1)}\right] = \sum_j \mathrm{Var}(w_{ij})\mathbb{E}\left[v_j^{(\ell-1)2}\right] = \frac{1}{2}n_{\text{in}}\sigma^2 \coloneqq 1 \Rightarrow \sigma^2 = \frac{2}{n_{\text{in}}}$$

This means that in this setting (of course heavily idealized), that the random variables will have the same scale with the same variance across each layer.

11.9.2 Weight initialization strategies

The goal of any weight initialization procedure is to keep the variance of weights approximately constant across layers to avoid vanishing and exploding gradients. Usually, random initialization works well, for instance:

- Glorot initialization using tanh:

$$w_{i,j} \sim \mathcal{N}(0, 1/(n_{in}))$$
$$w_{i,j} \sim \mathcal{N}(0, 1/(n_{in} + n_{out}))$$

- He (ReLU):

$$w_{i,j} \sim \mathcal{N}(0, 2/n_{in})$$

The effect of improper initialization strategies is that the weights do not converge over time, as shown in figure 21.

11.10 Learning rate

To implement the SGD rule, a learning rate needs to be chosen. Usually, its a good idea to start with a fixed small learing rate, and decrease slowly after some iteration, e.g.:

$$\eta_t = \min(0.1, 100/t)$$

Often, a piecewise constant learning rate schedules the drop in learning rate, e.g. drop after some number of epochs.

Learning with momentum is a common extension to training with SGD as it can help to escale local minima. The idea behind this approach is to not only move into the direction of the gradient, but also in direction of the last weight update. In this case the weight update looks like this:

$$a \leftarrow m \cdot a + \eta_t \nabla \ell(\mathbf{W}; \mathbf{y}, \mathbf{x})\mathbf{W} \qquad \leftarrow \mathbf{W} - a$$

This allows getting over flatter surfaces of the optimization function quicker while also avoiding oscillations towards the end of the optimization process as we near the optimal point, see figure 22.

11.11 Weight-space symmetries

Multiple distinct weights compute the same prediction. I.e. given a network of the topology provided in figure 23, and we define the following variables:

$$v_i = \phi(\mathbf{w}_i, x)$$
$$f = w_1' v_1 + w_2' v_2$$
$$\mathbf{w} = [w_i, w_i']_i, \bar{\mathbf{w}} = [\bar{w_i}, \bar{w_i'}]_i$$

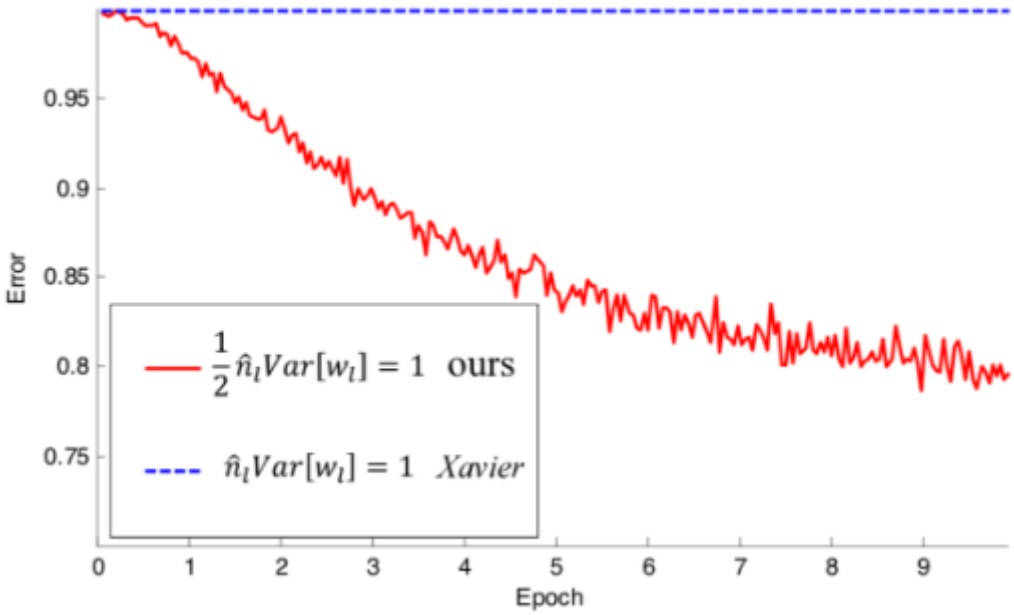


Figure 21: Weight initialization procedure and effect on training

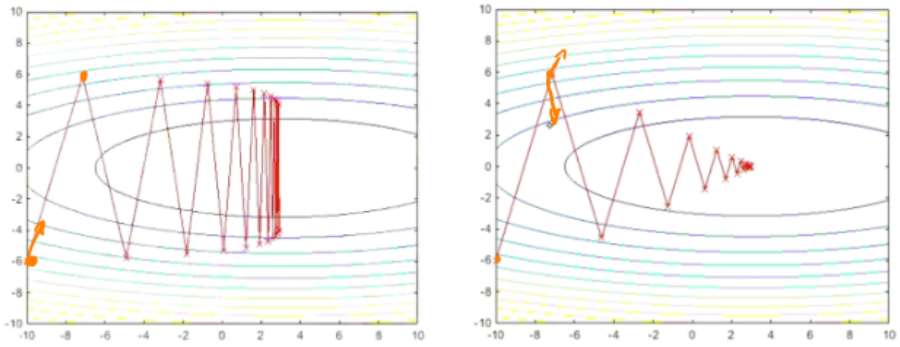


Figure 22: Oscillations of the optimization function.

Figure 23: Network topology used to illustrate weight-space symmetries shown in section 11.11.

Suppose $\bar{\mathbf{w}}_2 = \mathbf{w}_1$, $\bar{\mathbf{w}}_1 = \mathbf{w}_2$, $\bar{\mathbf{w}}'_1 = \mathbf{w}'_2$, $\bar{\mathbf{w}}'_2 = \mathbf{w}'_1$, we then have:

$$f(\mathbf{x}; \bar{\mathbf{w}}) = f(\mathbf{x}; \mathbf{w})$$

which means that multiple local minima can be equivalent in terms of input-output mapping.

For instance, this is valid for the tanh activation function, where $\varphi(z) = -\varphi(-z)$

11.12 Avoiding overfitting

Due to the fact that neural networks have multiple parameters, they have the potential of overfitting the data. There are several countermeasures that can be adopted to mitigate this phenomenon:

- Early stopping: it consists in stopping SGD until it converges. This is done by monitoring the prediction performance on a validation set and stop trainign once the validation error starts to increase.
- Regularization: add penalty term to keep the weights small.

$$\hat{\mathbf{W}} = \arg\min_{\mathbf{W}} \sum_{i=1}^{n} \ell(\mathbf{W}; \mathbf{x}_i, y_i) + \lambda \|W\|_F^2$$

- Dropout: the key idea here is to randomly ignore hidden units during each iteration of SGD with probability p. At test time, we then multiply the weights by the probability p.
- Batch normalization: inputs are shifted and scaled through each layer. Batch normalization is a widely used technique that normalizes inputs to each layer according to mini-batch statistics.
  - It reduces internal covariate shift
  - Enables larger learning rates
  - It helps with regularization.

The algorithm for batch normalization goes as follows:

1. **Input**: values of x over a mini-batch: $\mathcal{B} = \{x_1, \dots, m\}$; Parameters to be learned: γ, β
2. **Output**: $\{y_i = \mathrm{BN}_{\gamma,\beta}(x_i)\}$

$$\mu_{\mathcal{B}} \leftarrow \frac{1}{n} \sum_{i=1}^{m} x_i \qquad \text{mini-batch mean}$$

$$\sigma^2_{\mathcal{B}} \leftarrow \frac{1}{n} \sum_{i=1}^{m} (x_i - \mu_{\mathcal{B}})^2 \qquad \text{mini-batch variance}$$

$$\hat{x_i} \leftarrow \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma^2_{\mathcal{B}} + \epsilon}} \qquad \text{normalize}$$

$$y_i \leftarrow \gamma \hat{x}_i + \beta \equiv \mathrm{BN}_{\gamma,\beta}(x_i) \qquad \text{scale and shift}$$

After this procedure, $\phi(\mathbf{wx}) \rightarrow \phi(\mathbf{w}\mathrm{B}_{\gamma,\beta}(\mathbf{x}))$

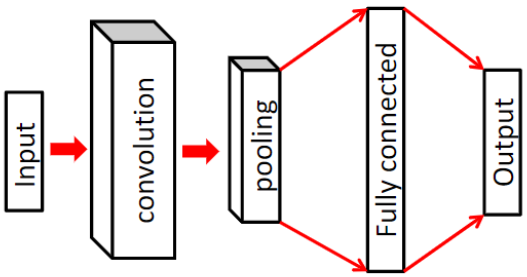


Figure 24: Typical architecture of a CNN

11.13 Invariances

Predictions should be unchanged under some transformations of the data, e.g.

- Classification of handwritten digits: the digit can be anywhere on the picture
- Speech recognition: the pitch should not matter for classification the speed of the speech should also not matter.

A model can be encouraged to learn specific invariances via the following methods:

- Augmentation of the training set
- Special regularization terms
- Invariance built into pre-processing
- Implement invariance into structure of ANN, e.g. using convolutional neural networks.

11.14 Convolutional neural networks

Convolutional neural networks are ANNs for specialized applications, e.g. for image recognition. The hidden layers closest to the input layer shares parameters: each hidden unit only depends on all closeby inputs, e.g. pixels and weights contrained to be identical across all units on the layer. This reduces the number of parameters, and encourages robustness against small amounts of translation. The weights can still be optimized using backpropagation.

The typical CNN architecture that we see is that similar to the one shown in figure 24.

Important aspects of CNN have specific terminologies:

- Filters/kernels: the size of the filter that passes over an image
- Stride: the size of the step taken by the filter at each iteration
- Padding: the distance between the border of the image and the first pixel to be processed.

Computing the output dimensions of a CNN can be done as follows. Applying m different $f \times f$ filtes to an $n \times n$ image with padding p and stride s, we get an output measurement of $\ell = \frac{n+2p-f}{s} + 1$. In some applications (e.g. image classification), it can make sense to aggregate several units to decrease the width of the network (and hence the number of parameters).

Pooling layers (equivalent to subsampling) can be added. In some applications, e.g. in image classification, it can make sense to aggregat several units to decrease the width of the network and hence the number of parameters. Usually, one considers either the average or the maximum value of the resulting output of the filter, see figure 25.

An example of CNN architecture can be found in figure 24.

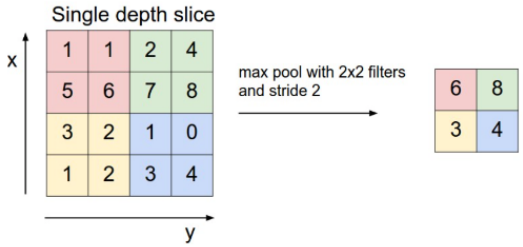


Figure 25: Max pooling layers example.

11.15 Computational efficiency

A key operation for backpropagation training is the dense matrix vector multiplications. These are very suitable on general purpose GPUs and much work on special purpose hardware, for instance TPUs. They have been reponsible for large improvement in computations in recent years.

11.16 Choice of parameters and architectures

Several possible activation functions are available. Popular in the past were sigmoid and tanh activations, because they were differentiable. Recently, however, rectified linear units (ReLUs) have been used. Despite being not differentiable, they are very fast to compute and gradients do not vanish, which is important for deep neural networks.

A number of parameters have to be chosen:

- The number and width of hidden layers
- Use convolution/pooling layers and how many
- Skip connections
- Type of activation functions
- Weight initialization
- Learning rate schedule
- Regularization method

Frameworks for autodifferentiation, e.g. theano, torch and tensorflow. They can use cross-validation to compare models, but training is usually very expensive. They often use single validation (=development) set. Much work is also being done on ML-based experimentaion, i.e. AutoML.

11.17 ANNs vs. kernels

When using the tanh activation function, an ANN with a single hidden layer learns functions of the form:

$$f(\mathbf{x}) = \sum_i w_i \tanh \theta_i^T \mathbf{x}$$

This is exactly the same type of functions learned with kernels, when using the tanh kernel.

$$f(\mathbf{x}) = \sum_i \alpha_i \tanh \mathbf{x}_i^T \mathbf{x}$$

The difference is that kernels optimize α's only. It is therefore convex. ANNs, however, optimize both $\mathbf{w}'s$ and θ_i, which is a non-convex problem. A more comprehensive comparison of ANNs and kernels is summerized in table **??**.

Method	*Method*	*ANNs*
Advantages	Convex optimization, no local minima, robust against noise, models grow with the size of the data	Flexible nonlinear models, with fixed parametrization. Multiple layers discover representations at multiple levels of abstraction
Disadvantages	Models grow with the size of data, don't allow multiple layers.	Many free parameters/architectural choices that need to be tuned. Often suffer from very noisy data.

12 Clustering

Definition 12.1 (Unsupervised learning)**.** *Unsupervised learning consists of learning a representation of the data without labels, which is typically useful for data analysis, finding patterns and visualization. Most common methods to perform unsupervised learning are clustering and dimensionality reduction.*

Definition 12.2 (Clustering)**.** *Given a set of datapoints, the goal is to group them into clusters such that similar points are in the same cluster and that dissimilar points are in different clusters. Points are typically represented either in (high-dimensional) Euclidean space or with distances specified by a metric or kernel. A related field of clustering is anomaly/outlier detection – identification of points that don't fit well in any of the clusters.*

Examples of clusterings tasks include:

- Documents based on the words they contain
- Images based on image features
- DNA sequences based on mutation distance
- Products based on which customers bought them
- Customers based on their purchase history
- Web surfers based on their queries/sites they visit.

There are several standard approaches to clustering:

- **Hierarchical clustering**: it consists of building a tree (bottom-up or top-down) representing distances among data points. For instance, this includes single-, average-linkage clustering.
- **Partitional approaches**: it consists in defining and optimizing a notion of cost defined over partitions: for instance, this includes spectral clustering, graph-cut based approaches.
- **Model-based approaches**: these methods consist in maintaining cluster models and infer cluster membership, e.g. assign each point to the closest cluster. Examples of model-based approaches include k-means clustering, Gaussian mixture models, etc...

12.1 k-means clustering

12.1.1 Intuition

The idea behind k-means clustering is that every cluster is represented by a single point (named $\mu_j \in \mathbb{R}$) and each point is assigned to the closest center, for instance for cluster 1:

$$\left\{ \mathbf{x} : 1 \arg\min_j \|\mathbf{x} - \mu_j\| \right\}$$

This induces a Voronoi partition. See also figure 26 for an illustration.

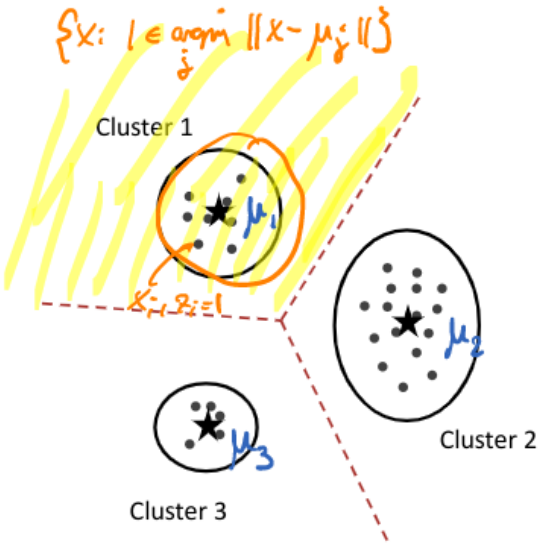


Figure 26: k-means clustering illustration

12.1.2 The problem

For k-means clustering, we assume that the points are in Euclidean space, i.e. $\mathbf{x} \in \mathbb{R}^d$. We represent each cluster as a center $\mu \in \mathbb{R}^d$ and each point is assigned to the closest center. The overall goal is to pick centers to minimize the average squared distance:

$$\hat{R}(\mu) = \hat{R}(\mu_1, \ldots, \mu_k) = \sum_{i=1}^{n} \min_{j \in \{1,\ldots,k\}} \|\mathbf{x_i} - \mu_j\|_2^2$$

and:

$$\hat{\mu} = \arg\min_{\mu} \hat{R}(\mu)$$

This is a non-convex optimization problem and is NP-hard in nature, which means that we can't solve this problem optimally in general.

12.1.3 Lloyd's heuristic

Lloyd's algorithm consists in:

- Initialize the cluster centers $\mu_0 = [\mu_1^{(0)}, \ldots, \mu_k^{(0)}]$
- While the function does not converge:
  - Assign each point $\mathbf{x}_i$ to closest center, i.e.:

$$z_i^t \leftarrow \arg\min_{j \in \{1,\ldots,k\}} 4 \left\|\mathbf{x}_i - \mu_j^{t-1}\right\|.$$

  - We then update the center as being the mean of all the points assigned to that center, i.e.:

$$\mu_j^{(t)} \leftarrow \frac{1}{n_j} \sum_{i: z_i^{(t)} = j} \mathbf{x}_i$$

    where:

$$n_j = \left|\left\{i : z_i^{(t)=j}\right\}\right|.$$

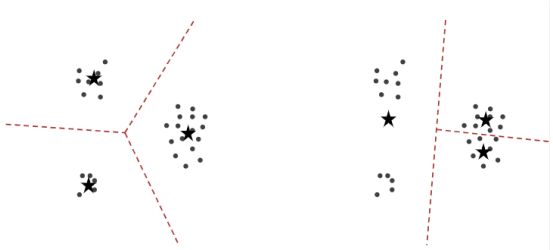


Figure 27: Local optima of a k-means algorithmiteration

12.1.4 Properties of k-means

k-means is guaranteed to monotonically decrease the average squared distance in each iteration, i.e. $\hat{R}(\mu^{(t)}) \geq \hat{R}(\mu^{(t+1)})$.

Proof that k-means monotonically decreases. We know that $\hat{R}(\mu^{(t)})$ is defined by $\hat{R}(\mu^{(t)}) = \sum_{i=1}^{n} \min_{j \in \{1,\dots,k\}} \left\| \mathbf{x}_i - \mu_j^{(t)} \right\|_2^2$ at a given timestep t. Our objective is to find μ_{z_i} s.t. $\hat{R}(\mu) = \min_z \hat{R}(\mu, z)$ (i.e. the set of cluster center assignments such that the distance is minimized).

We know that $\hat{R}(\mu, z) = \sum_{i=1}^{n} \|\mathbf{x}_i - \mu_{z_i}\|_2^2$. Then, it can be shown that:

$$\hat{R}(\mu^{(t)}, z^{(t)}) \geq \hat{R}(\mu^{(t)}, z^{(t+1)}) \geq \hat{R}(\mu^{(t+1)}, z^{(t+1)})$$

Since $z^{(t+1)} = \arg\min_{\mathbf{z}} \hat{R}(\mu^{(t)}, \mathbf{z})$ we can infer that $\hat{R}(\mu^{(t)}, z^{(t)}) \geq \hat{R}(\mu^{(t)}, z^{(t+1)})$ and since $\mu^{(t)} = \arg\min_{\mu} = \hat{R})(\mu, z^{(t+1)})$, which means it converges to a local optimum. □

The complexity per iteration of the algorithm is in $\mathcal{O}(n \cdot k \cdot d)$.

As noted in the proof, the algorithm converges towards a local optimum. A graphical representation of such an event can be seen in figure **??**.

12.1.5 Challenges with k-means

There are multiple challenges when performing k-means. Generally, the algorithm only converges to a local optimum, and the performance of the methodology is strongly dependent on the initialization. The number of iterations required can be exponential, even in the place. In practice however the algorithm converges very quickly. Determining the number of clusters k is difficult, and cannot model clusters of arbitrary shape, although kernel Kernel-k-means can be used to fix this.

12.1.6 Initializing k-means

Lloyd's heuristic does not generally converge to the optimal solution. The performance heavily depends on the initialization. Several approaches towards solving the initialization problem are listed below:

- Multiple random restarts
- Farthest points heuristic, which often works well but is prone to outliers.
- Seeding with k-means ++.
- Other methods exist as well.

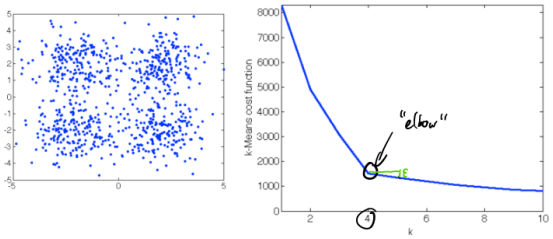


Figure 28: An instance of a choice of k, here 4.

12.2 k-means ++

The idea behind k-means ++ is to increase the sampling probability of sampling initial points that are likely far away from the center of the cluster.

The algorithm consists of the following steps:

1. Start with a random data point as a center, where $i \sim \text{Uniform}(\{1, \dots, n\})$ and $\mu_0 = x_i$.
2. Add centers 2 to k randomly, proportionally to the squared distance to the closest selected center, i.e. for $j = 2 : k$, pick ij with probability $\frac{1}{z} \cdot d(x_{ij}; \mu^{(0)}_{1:j-1})$ where $d(x_{ij}; \mu^{(0)}_{1:j-1}) = \min_{\ell \in \{1,\dots,j-1\}} \left\| \mathbf{x}_i j - \mu^{(0)}_\ell \right\|_2^2$ and assign: $\mu^{(0)}_j = x_{ij}$

We can show that the expected cost of this procedure is in $\mathcal{O}(\log k)$ times that of optimal k-means solutions: $\mathbb{E}[\hat{R}(\mu^{(0)})] \leq \mathcal{O}(\log k) \min_\mu \hat{R}(\mu)$

12.3 Model selection in clustering

In general, model selection is very difficult in clustering. There are, however, several approaches that can be adopted. For instance:

- heuristic quality measures
- regularization (i.e. favor simple models with few parameters by penalizing complex models)
- Information theoretic basis (tradeoff between robustness (stability) and informativeness), which can be derived from statistical learning theory.

A heuristic for determing k in k means is to use a loss function and plot this value against the number of clusters chosen. Then the optimal choice is made when there is a k such that increasing k leads to neglibible decreases in loss, as see in figure 28

12.4 Regularization

A regularization term can be added to the optimization objective such that more complex models are penalized. For k-means, this can be done as follows:

$$\min_{k, \mu_{i:k}} \hat{R}(\mu_{i:k}) + \lambda \cdot k.$$

Graphically, this is introduces an elbow in the plot of the cost of the loss functions vs. the number of clusters, see figure 29.

Note that the test loss cannot be used to determine the number of clusters, as this loss typically also keeps decreasing.

Challenges with k-means which are not a big issue in practice include:

- Only a local optimum can be reached in general.
- The number of iterations required can be exponential.

Challenges with k-means which can be fixed via kernel k-means, etc.

- It is not possible to model clusters of arbitrary shape.

A major unsolved practical problem is the choice of number of clusters k, which is difficult.

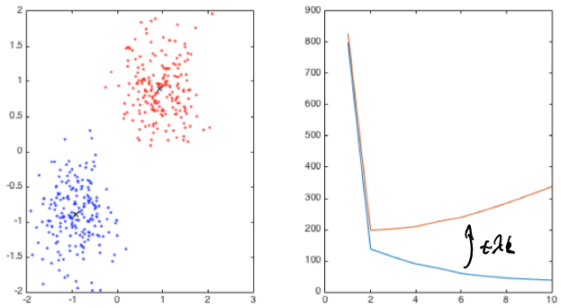


Figure 29: Illustration of the effect of regularization on the cost function.

13 Dimensionality reduction

The basic challenge is that, given a dataset $\mathcal{D} = \{\mathbf{x}_1, \dots, \mathbf{x}_n\} \forall \mathbf{x}_i \in \mathbb{R}^d$, we want to obtain an "embedding", i.e. a low dimensional representation $\mathbf{z}_1, \dots, \mathbf{z}_n \in \mathbb{R}^k$ where $k < d$. This serves multiple purpose, including:

- visualization for $k = 1, 2, 3$
- Regularization (model selection)
- Unsupervised feature discovery (i.e. determine features from data)

Typical approaches to achieve this problem achieve the following:

1. Assume $\mathcal{D} = \{\mathbf{x}_1, \dots, \mathbf{x}_n\} \forall \mathbf{x}_i \subseteq \mathbb{R}^d$
2. Obtain a mapping $\mathbf{f} : \mathbb{R}^d \to \mathbb{R}^k$ where $k \ll d$.

It is possible to distinguish between the following:

- Linear dimension reduction, i.e. $\mathbf{f}(\mathbf{x}) = \mathbf{A}\mathbf{x} \forall A \in \mathbb{R}^{k \times d}$
- Nonlinear dimension reduction:
  - Parametric
  - Non-parametric

A key question in the field then arises: which mappings should we prefer?

13.1 Linear dimension reduction as compression

The motivation behind linear dimension reduction is that a low-dimensional representation should allow to compress the original data (accurate reconstruction). For instance, if we set $k = 1$, and given a dataset $\mathcal{D} = \{\mathbf{x}_1, \dots, \mathbf{x}_n\} \forall \mathbf{x}_i \subseteq \mathbb{R}^d$, we want to represent data as points on a line $\mathbf{w} \in \mathbb{R}^d$ with coefficiens $z_1, \dots, z_n$, i.e. we want $z_i \mathbf{w} \approx \mathbf{x}_i$, assuming $\mu = \frac{1}{n} \sum_i \mathbf{x}_i = 0$ as shown in figure 30.

13.2 Linear dimension reduction for reconstruction

Given a dataset $\mathcal{D} = \{\mathbf{x}_1, \dots, \mathbf{x}_n\} \forall \mathbf{x}_i \subseteq \mathbb{R}^d$, we want $z_i \mathbf{w} \approx \mathbf{x}_i$, e.g. minimizing $\|z_i \mathbf{w} = \mathbf{x}_i\|_2^2$. To ensure uniqueness, we can normalize $\|\mathbf{w}\|_2 = 1$, and then we optimize over $\mathbf{w}, z_1, \dots, z_n$ jointly:

$$(\mathbf{w}*, z_{1:n}*) = \arg \min_{\|\mathbf{w}\|_2 = 1, \mathbf{z}} \sum_{i=1}^{n} \|\mathbf{x}_i - z_i \cdot \mathbf{w}\|_2^2 = \hat{R}(\mathbf{w}, z_{1:n})$$

Solving this equation for $\mathbf{z}$ given a vector $\mathbf{w}$ and we get:

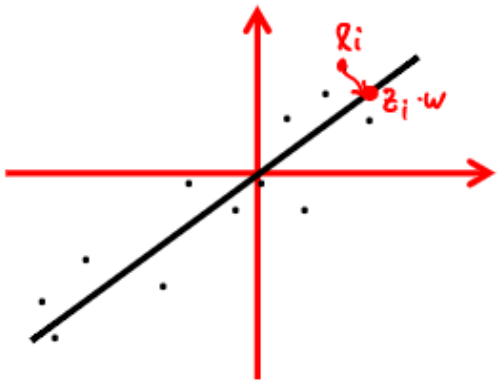


Figure 30: Graphical representation of linear dimension reduction with $k = 1$.

$$z_i* = \mathbf{w}^T \mathbf{x}_i$$

Thus we effectively solve a regression problem, interpreting $\mathbf{x}$ as features and z as labels. So, we obtain:

$$\begin{aligned} \mathbf{w}* &= \arg \min_{\|w\|_2=1} \sum_{i=1}^{n} \left\| \mathbf{w}\mathbf{w}^T \mathbf{x}_i - \mathbf{x}_i \right\|_2^2 \\ &= \sum_{i=1}^{n} \|x_i\|_2^2 - \sum_{i=1}^{n} (\mathbf{w}^T x_i)^2 \end{aligned}$$

Since $\sum_{i=1}^{n} \|x_i\|_2^2$ is constant w.r.t. $\mathbf{w}$, we have that:

$$\arg\min_{\mathbf{w}} \hat{R}(\mathbf{w}) = \arg\max_{w} \sum_{i=1}^{n} \left(\mathbf{w}^T x_i\right)^2$$

Solving the optimization objective for $\mathbf{w}$ yields:

$$\mathbf{w}* = \arg \max_{\|w\|_2=1} \sum_{i=1}^{n} (\mathbf{w}^T \mathbf{x}_i)^2$$

where:

$$\sum_{i=1}^{n} (\mathbf{w}^T \mathbf{x}_i)^2 = \sum_{i=1}^{n} \mathbf{w}^T \mathbf{x}_i \mathbf{w} \mathbf{x}_i^T = \mathbf{w}^T \sum_{i=1}^{n} (\mathbf{x}_i \mathbf{x}_i^T) \mathbf{w} = n \cdot \Sigma$$

where Σ is the empirical variance of the data assuming mean 0. So, we have that:

$$\mathbf{w}* = \arg \max_{\|\mathbf{w}\|_2=1} \mathbf{w}^T \Sigma \mathbf{w} \tag{15}$$

where $\Sigma = \frac{1}{n} \sum_{i=1}^{n} \mathbf{x}_i \mathbf{x}_i^T$ is the empirical covariance assuming the data is centered as stated above. The optimal solution for equation 15 is given by the principal eigenvector of Σ, i.e. $\mathbf{w}* = \mathbf{v}_1$ where

$$\Sigma = \sum_{i=1}^{d} \lambda_i \mathbf{v}_i \mathbf{v}_1^T \qquad \lambda_1 \geq \ldots \geq \lambda_d \geq 0 \tag{16}$$

To justify this, we have:

$$\mathbf{w} = \sum_{i=1}^{d} \alpha_i v_i$$

thus, our objective, which we want to minimize in equation 15 becomes:

$$\begin{aligned}\mathbf{w}^T \Sigma \mathbf{w} &= (\sum_i \alpha_i v_i)^T (\sum_{j=1}^{d} \lambda_j v_j v_j^T)(\sum_k \alpha_k v_k) \\ &= \sum_{ijk} \alpha_i \alpha_k \lambda_j (v_i^T v_j)(v_j^T v_k) = \sum_{i=1}^{d} \alpha_i^2 \lambda_i\end{aligned}$$

Since this problem is constrained ($\|\mathbf{w}\|_2 = 1$), we have that $\sum_{i=1}^{d} \alpha_i^2 = 1$, thus, in addition to the consition $\lambda_1 \geq \ldots \geq \lambda_d \geq 0$ given in equation 16, we have $\alpha_1 = 1$, $\alpha_i = 0 \forall i > 1$.

13.2.1 Extrapolating for $k > 1$

Suppose we wish to project to more than one dimension. Thus, we want:

$$(\mathbf{W}, \mathbf{z}_1, \ldots, \mathbf{z}_n) = \arg\min \sum_{i=1}^{n} \|\mathbf{W}\mathbf{z}_i - \mathbf{x}_i\|_2^2$$

where $\mathbf{W} \in \mathbb{R}^{d \times k}$ is orthogonal, $\mathbf{z}_1, \ldots, \mathbf{z}_n \in \mathbb{R}^k$.

This is called the principal component analysis problem. Its solution can be obtained in closed form even for $k > 1$.

13.3 Principal Component Analysis

Given a dataset $\mathcal{D} = \{\mathbf{x}_1, \ldots, \mathbf{x}_n\} \forall \mathbf{x}_i \subseteq \mathbb{R}^d, 1 \leq k \leq d$. We further assume that the data is centered:

$$\Sigma = \frac{1}{n} \sum_{i=1}^{n} \mathbf{x}_i \mathbf{x}_i^T \qquad \mu = \frac{1}{n} \sum_i \mathbf{x}_i = 0$$

The solution to the PCA problem is given below:

$$(\mathbf{W}, \mathbf{z}_1, \ldots, \mathbf{z}_n) = \arg\min \sum_{i=1}^{n} \|\mathbf{W}\mathbf{z}_i - \mathbf{x}_i\|_2^2$$

where $\mathbf{W} \in \mathbb{R}^{d \times k}$ is orthogonal, $\mathbf{z}_1, \ldots, \mathbf{z}_n \in \mathbb{R}^k$ is given by $\mathbf{W} = (\mathbf{v}_1 | \ldots | \mathbf{v}_k)$ and $\mathbf{z}_i = \mathbf{W}^T \mathbf{x}_i$ where:

$$\Sigma = \sum_{i=1}^{d} \lambda_i \mathbf{v}_i \mathbf{v}_i^T \qquad \lambda_1 \geq \ldots \geq \lambda_d \geq 0$$

Essentially, PCA is a projection. The linear mapping $\mathbf{f}(\mathbf{x}) = \mathbf{W}^T \mathbf{x}$ obtained from PCA projects vectors $\mathbf{x} \in \mathbb{R}^d$ into a k-dimensional subspace. [1]. This projection is chosen to minimize the reconstruction error, measured in Euclidean norm.

An illustration of PCA can be seen in figure 31.

The PCA can be obtained through singular value decomposition. Recall that it is possible to represent any $\mathbf{X} \in \mathbb{R}^{n \times d}$ as $\mathbf{X} = \mathbf{U}\mathbf{S}\mathbf{V}^T$ where $\mathbf{U} \in \mathbb{R}^{n \times n}$ and $\mathbf{V} \in \mathbb{R}^{d \times d}$ are orthogonal, and $\mathbf{S} \in \mathbb{R}^{n \times d}$ is diagonal (w.l.o.g. in decreasing order). Its entries are called singular values.

The top k principal components are exactly the first k columns of $\mathbf{V}$. Note that:

$$n\Sigma = \mathbf{X}^T \mathbf{X} = \mathbf{U}\mathbf{S}^t \mathbf{U}^T \mathbf{U}\mathbf{S}\mathbf{V}^T = \mathbf{V}\mathbf{S}^T \mathbf{S}\mathbf{V}^T \tag{17}$$

Choosing k for PCA can be done depending on the application domain:

[1] If $\mathbf{f}(\mathbf{f}(\mathbf{x})) = \mathbf{f}(\mathbf{x})$, then it holds that $\mathbf{f}$ is a projection

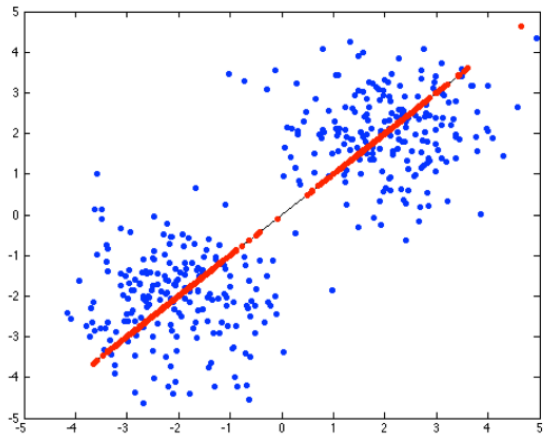


Figure 31: Illustration of PCA.

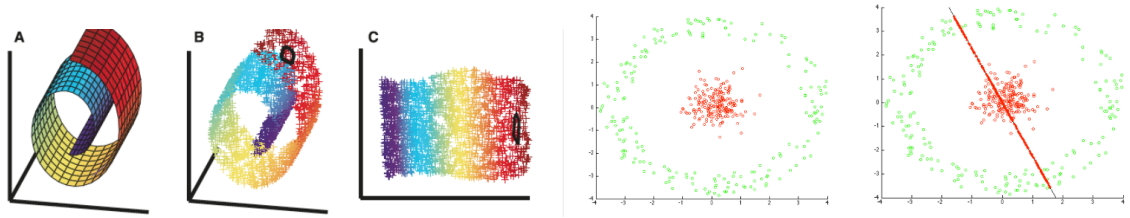


Figure 32: Linear feature reduction cannot reduce the dimensionality of this data very well.

- For visualization: by inspection
- For feature induction: by cross-validation
- Otherwise: pick k so that most of the variance is explained.

Note that the PCA and the k-means problems are not very dissimilar. They both solve the following problem:

$$(\mathbf{W}, \mathbf{z}_1, \ldots, \mathbf{z}_n) = \arg\min \sum_{i=1}^{n} \|\mathbf{W}\mathbf{z}_i - \mathbf{x}_i\|_2^2$$

Except that $\mathbf{W}$ in PCA is orthogonal, and arbitrary in k-means. Additionally, $\mathbf{z}_1, \ldots, \mathbf{z}_n) \in \mathbb{R}^d$ in PCA and $\mathbf{z}_1, \ldots, \mathbf{z}_n) \in E_k$ where $E_k = \{[1, 0, \ldots, 0], \ldots, [0, \ldots, 0, 1]\}$ is the set of unit vectors in $\mathbf{R}^d$

One can think of PCA and k-mans to solve a similar unsupervised learning problem with different constraints. Both aim to compress the data with maximum fidelity under constraints on model complexity, This insight gives rise to a much broader class of techniques, such as matrix factorization.

13.4 Kernel PCA

In order to reduce the dimensionality of data in higher dimensions in nonlinear forms, which can lead to significant loss of information as seen in figure 32.

Therefore we can use kernels. This way we can solve non-linear problems by reducing them to linear ones in high-dimensional, implicitely represented spaces. A similar approach for supervised learning can be taken.

The Ansatz we make is the following: $\mathbf{w} = \sum_{i=1}^{n} \alpha_i \mathbf{x}_i$. For PCA, $w*$ is leading eigenvector of $\mathbf{X}^T\mathbf{X}$. We then have that $\mathbf{X}^T\mathbf{X}\mathbf{w} = \lambda\mathbf{w}$, therefore $\mathbf{w} = \frac{1}{\lambda}\sum_i$. If we define $\beta = \mathbf{X}\mathbf{w}$, we then have that $\mathbf{w} = \frac{1}{\lambda}\sum_i \beta_i x_i$ and $\alpha_i = \frac{\beta_i}{\lambda}$.

The objective we want to minimize is the following:

$$\begin{aligned}
\mathbf{w}^T\mathbf{X}^T\mathbf{X}\mathbf{w} &= \sum_{i=1}^{n}(\mathbb{W}^T x_i)^2 \\
&= \sum_{i=1}^{n}((\sum_{j=1}^{n}\alpha_j x_j)^T x_i)^2 \\
&= \sum_{i=1}^{n}(\sum_{j}\alpha_j(\mathbf{x}_j^T\mathbf{x}_i))^2 \\
&= \sum_{i=1}^{n}(\sum_{j}\alpha_j k(\mathbf{x}_j,\mathbf{x_i}))^2 \\
&= \sum_{i=1}^{n}(\alpha^T k_i)^2 \\
&= \alpha^T K^T K\alpha
\end{aligned}$$

The constraints we want to put in place are the following:

$$\begin{aligned}
\|w\|_2 &= \mathbf{w}^T\mathbf{w} \\
&= (\sum_{i=1}^{n}\alpha_i\mathbf{x}_i)^T(\sum_{j=1}^{n}\alpha_j\mathbf{x}_j) \\
&= \sum_{ij}\alpha_i\alpha_j\mathbf{x}_i^T\mathbf{x}_j \\
&= \sum_{ij}\alpha_i\alpha_j k(\mathbf{x}_i,\mathbf{x}_j) \\
&= \alpha^T\mathbf{K}\alpha \overset{!}{=} 1
\end{aligned}$$

Thus, kernel PCA requires solving $\max_\alpha \alpha^T\mathbf{K}^T\mathbf{K}\alpha$ such that $\alpha^T K\alpha = 1$.
Applying feature maps using:

$$\mathbf{w} = \sum_{j=1}^{n}\alpha_j\phi(\mathbf{x}_j)$$

and observing $\|\mathbf{w}\|_2^2 = \alpha^T\mathbf{K}\alpha$, we have that:

$$\begin{aligned}
\arg\max_{\|\mathbf{w}\|_2^2=1}\sum_{i=1}^{n}(\mathbf{w}^T\phi(\mathbf{x}_i))^2 &= \arg\max_{\alpha^T\mathbf{K}\alpha=1}\sum_{i=1}^{n}(\sum_{j=1}^{n}\alpha_j\phi(\mathbf{x}_j)^T\phi(\mathbf{x_i}))^2 \\
&= \arg\max_{\alpha^T\mathbf{K}\alpha=1}\sum_{i=1}^{n}(\sum_{j=1}^{n}\alpha_j k(\mathbf{x}_j,\mathbf{x}_i))^2 \\
&= \arg\max_{\alpha^T\mathbf{K}\alpha=1}\sum_{i=1}^{n}(\alpha^T\mathbf{K}_i)^2 \\
&= \arg\max_{\alpha^T\mathbf{K}\alpha=1}\sum_{i=1}^{n}(\alpha^T\mathbf{K}_i)^2 \\
&= \arg\max_{\alpha^T\mathbf{K}\alpha=1}\alpha^T\mathbf{K}^T\mathbf{K}\alpha.
\end{aligned}$$

13.4.1 Kernel PCA for $k = 1$

The Kernel-PCA problem ($k = 1$) requires solving:

$$\alpha* = \arg\max_{\alpha^T\mathbf{K}\alpha=1}\alpha^T\mathbf{K}^T\mathbf{K}\alpha.$$

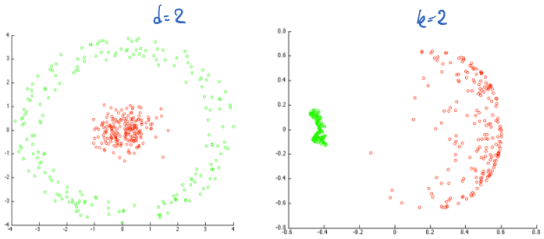


Figure 33: Kernelized PCA illustration

The optimal solution is obtained in closed form from the eigendecomposition of $\mathbf{K}$:

$$\alpha* = \frac{1}{\sqrt{\lambda_1}}\mathbf{v}_1$$

This scaling is useful because:

$$\begin{aligned}\alpha^T\mathbf{K}\alpha &= \frac{1}{\sqrt{\lambda_1}}v_1^T\mathbf{K}\frac{1}{\sqrt{\lambda_1}}v_1 \\ &= \frac{1}{\lambda_1}\cdot v_1^T\mathbf{K}v_1 \\ &= \frac{1}{\lambda_1}v_1^T(\lambda_1 v_1) \\ &= v_1^T v_1 = 1\end{aligned}$$

$$\mathbf{K} = \sum_{i=1}^{n}\lambda_i\mathbf{v}_i\mathbf{v}_i^T \qquad \lambda_1 \geq \ldots \geq \lambda_d \geq 0$$

Given this, a new projected point $\mathbf{x}$ is projected as $\mathbf{z} \in \mathbb{R}^k$ such that:

$$\mathbf{z}_i = \mathbf{w}^{(i)T}\mathbf{x} = (\sum_{j=1}^{n}\alpha_j^{(i)}\mathbf{x}_j)^T\mathbf{x} = \sum_{j=1}^{n}\alpha_j^{(i)}\mathbf{x}_j^T\mathbf{x} = \sum_{j=1}^{n}\alpha_j^{(i)}k(\mathbf{x}_j, x)$$

13.4.2 Kernel PCA for any $0 < k < d$

For any $1 \leq k < d$, the kernel principal components are given by $\alpha^{(1)}, \ldots, \alpha^{(k)} \in \mathbb{R}^n$ where:

$$\alpha^{(i)} = \frac{1}{\sqrt{\lambda_i}}\mathbf{v}_i$$

is obtained from:

$$\mathbf{K} = \sum_{i=1}^{n}\lambda_i\mathbf{v}_i\mathbf{v}_i^T \qquad \lambda_1 \geq \ldots \geq \lambda_d \geq 0. \tag{18}$$

Given this, a new point $\mathbf{x}$ is projected as $\mathbf{z} \in \mathbb{R}^k$:

$$z_i = \sum_{j=1}^{n}\alpha_j^{(i)}k(\mathbf{x}, \mathbf{x}_j)$$

Applying k-means on kernel principal components is sometimes called kernel-k-means or spectral clustering. The illustration of the advantage of kernelizing PCA is shown in figure 33 and 34.

Kernel-PCA corresponds to applying PCA in the feature space induced by the kernel k. It can be used to discover non-linear feature maps in closed form. This can be used as inputs, e.g. to SVMs given multi layer SVMs. The kernel could also be centered, i.e.:

$$\mathbf{K}' = \mathbf{K} - \mathbf{KE} - \mathbf{EK} + \mathbf{EKE}$$

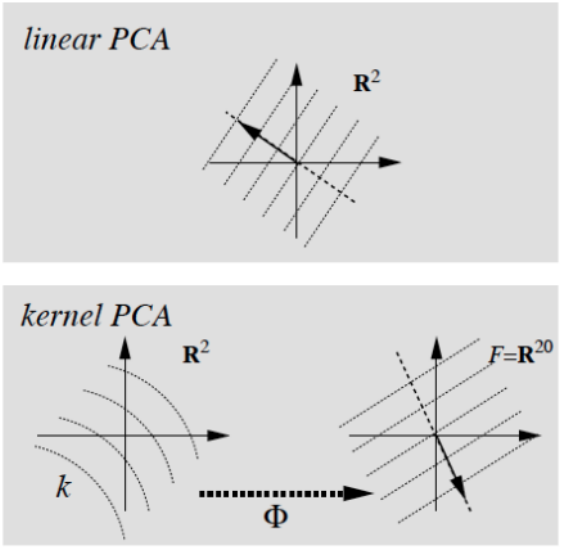


Figure 34: Another kernelized PCA illustration

and $\mathbf{E} = \frac{1}{n}[1, \ldots, 1][1, \ldots, 1]^T$

It should be noted that kernelized PCA requires data specified as kernel. The complexity grows with the number of datapoints and cannot be easily explicitely embed high-dimensional data unless we have an appropriate kernel, which may not always be the case. Alternatives include:

- Autoencoders
- Locally linear embedding
- Multi-dimensional scaling
- t-SNE

13.5 Neural network autoencoders

The key idea behind autoencoders is to try to learn the identity function:

$$\mathbf{x} \approx f(\mathbf{x}; \theta)$$

The function f can be decomposed by an encoder and a decoder, f_1 and f_2 respectively:

$$f(\mathbf{x}; \theta) = f_2(f_1(\mathbf{x}; \theta_1); \theta_2)$$

Neural networks can be used to learn the parameters *theta*, as shown in figure 35.

Neural network autoencoders are ANNs where there is one output unit for each of the d input units. The number k of hidden units is uually smaller than the number of inputs. The goal is to optimize the weights such that the output agrees with the input.

13.5.1 Training autoencoders

The goal is to optimize the weights such that the output agrees with the input. This entails, for instance, minimizing the squared loss:

$$\min_{\mathbf{W}} \sum_{i=1}^{n} \|\mathbf{x_i} - \mathbf{f}(\mathbf{x}_i; \mathbf{W})\|_2^2 = \min_{\mathbf{W}} \sum_{i=1}^{n} (\sum_{j=1}^{d} (\mathbf{x}_{i,j} - \mathbf{f}(\mathbf{x}_i; \mathbf{W}))^2)$$

The aim is to find a local minimum via SGD (backpropagation). The initialization of such networks matters and is challenging

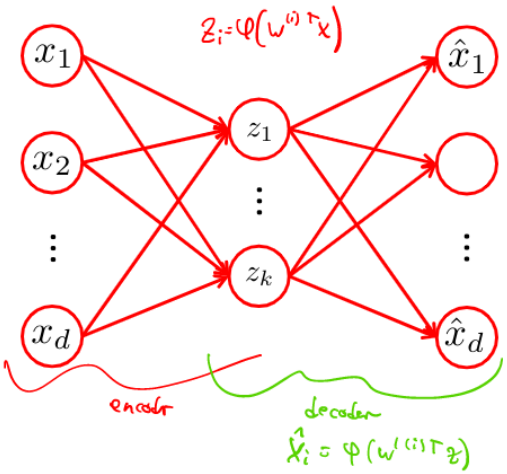


Figure 35: An autoencoder network

14 Probabilistic modelling

The reason why we want to have a statistical perspective on supervised learning is that we want to quantify uncertainty, and express prior knowledge and assumptions about the data.

14.1 Probabilistic interpretation of LSR

We recall given $(\mathbf{x}_i, y_i) \sim P(\mathbf{X}, Y)$, we wish to identify a hypothesis $h : \mathcal{X} \to \mathcal{Y}$ that minimizes the prediction error:

$$R(h) = \int P(\mathbf{x}, y)\ell(y; h(\mathbf{x}))\mathrm{d}\mathbf{x}\mathrm{d}y = \mathbb{E}_{\mathbf{x},y}[\ell(y; h(\mathbf{x}))]$$

In least-squares regression, the risk is defined as:

$$R(h) = \mathbb{E}_{\mathbf{X},Y}[(Y - h(\mathbf{X}))^2]$$

Suppose we know $P(\mathbf{X}, Y)$ (which is unrealistic), then h is minimized by:

$$\begin{aligned}\min_{h:\mathbb{R}^d\mathbb{R}} R(h) &= \min_h \mathbb{E}_{(\mathbf{x},y)\sim P}[(y - h(\mathbf{x}))^2] \\ &= \min_h \mathbb{E}_{\mathbf{x}}[\mathbb{E}_y[(Y - h(\mathbf{x}))^2|\mathbf{X} = \mathbf{x}]] \\ &= \mathbb{E}_{\mathbf{x}}[\min_{h(\mathbf{x})} \mathbb{E}_y[(Y - h(\mathbf{x}))^2|\mathbf{X} = \mathbf{x}]]\end{aligned}$$

Since we consider an arbitrary h, we choose $h(\mathbf{x})$ and $h(\mathbf{x}')$ independently for $\mathbf{x} \neq \mathbf{x}'$

In the context of least squares regression, the optimal prediction is given by:

$$y*(\mathbf{x}) \in \arg\min_{\hat{y}} \mathbb{E}_Y[(\hat{y} - Y)^2|\mathbf{X} = \mathbf{x}] = \arg\min_{\hat{y}} \ell(\hat{y})$$

where $\ell(\hat{y})$ can be rewritten as:

$$\frac{d}{d\hat{y}}\ell(\hat{y}) = \int (\hat{y} - y)^2 p(y|\mathbf{x})dy$$

In order to find the minimum of that function we find its derivative and set it to 0:

$$\begin{aligned}\frac{d}{d\hat{y}}\ell(\hat{y}) &= \int \frac{d}{d\hat{y}}(\hat{y}-y)^2 p(y|\mathbf{x})dy \\ &= \int 2(\hat{y}-y)p(y|\mathbf{x})dy \stackrel{!}{=} 0 \\ &\Rightarrow \int \hat{y}p(y|\mathbf{x})dy = \int yp(y|\mathbf{x})dy \\ &\Leftrightarrow \hat{y} = \mathbb{E}[Y|\mathbf{X}=\mathbf{x}]\end{aligned}$$

So, assuming the data is iid according to $(\mathbf{x}_i y_i) \sim P(\mathbf{X}, Y)$, the hypothesis $h*$ minimizing $R(h) = \mathbb{E}_{\mathbf{x},y}[(y - h(\mathbf{x}))^2]$ is given by the conditional mean:

$$h*(\mathbf{x}) = \mathbb{E}[Y|\mathbf{X}=\mathbf{x}]$$

This (in practice unattainable) hypothesis is called the Bayes optimal predictor for the squared loss. However in practice we have finite data. Thus, one strategy for estimating a predictor from training data is to estimate the conditional distribution:

$$\hat{P}(Y|\mathbf{X})$$

and then, for test point $\mathbf{x}$, we predict the label:

$$\hat{y} = \hat{\mathbb{E}}[Y|\mathbf{X}=\mathbf{x}] = \int \hat{P}(y|\mathbf{X}=\mathbf{x})ydy$$

14.2 Estimating conditional distribution

From the derivation above, we hence need to estimate $\hat{P}(Y|\mathbf{X}, \theta)$ and then we optimize the parameters using parametric estimation, specifically maximum likelihood estimation.

$$\begin{aligned}\theta* &= \arg\max_\theta \hat{P}(y_1, \ldots, y_n|\mathbf{x}_1, \ldots, \mathbf{x}_n, \theta) \\ &\stackrel{iid}{=} \arg\max_\theta \prod_{i=1}^n \hat{P}(y_i|\mathbf{x}_i, \theta) = \arg\max_\theta \log \prod_{i=1}^n \hat{P}(y_i|\mathbf{x}_i, \theta) \\ &= \arg\max_\theta \sum_{i=1}^n \log \hat{P}(y_i|\mathbf{x}_i, \theta) = \arg\min_\theta -\sum_{i=1}^n \log \hat{P}(y_i|\mathbf{x}_i, \theta)\end{aligned}$$

14.2.1 Conditional linear Gaussian

Suppose we assume $Y = h(X)+\epsilon$, with $\epsilon \sim \mathcal{N}(0, \sigma^2)$. Additionally, we also assume $h(\mathbf{x}) = \mathbf{w}^T\mathbf{x}$. Then, we have:

$$\hat{P}(Y = y|\mathbf{X}, \mathbf{w}, \sigma^2) = \mathcal{N}(y; \mathbf{w}^T\mathbf{x}, \sigma^2)$$

and

$$\begin{aligned}\hat{\mathbf{w}} &= \arg\max_{\mathbf{w}} \hat{P}(y_{1:n}|X_{1:n}, \mathbf{w}, \sigma^2) \\ &= \arg\min_{\mathbf{w}} -\sum_{i=1}^n \log \hat{P}(y_i|x_i, \mathbf{w}, \sigma^2)\end{aligned}$$

14.3 Probabilistic model for regression

Consider linear regression. Let's make the statistical assumption that the noise is Gaussian, i.e. that $y_i \sim \mathcal{N}(\mathbf{w}^T\mathbf{x}_i, \sigma^2)$. Then, we can compute the conditional likelihood of the data given

any candidate model $\mathbf{w}$ as:

$$\begin{aligned}
-\log \hat{p}(y|X,\mathbf{w}) &= -\log \mathcal{N}(y|\mathbf{w}^T x, \sigma^2) = -\log(\frac{1}{\sqrt{2\pi\sigma^2}})\exp(-\frac{(y-\mathbf{w}^T\mathbf{x})^2}{2\sigma^2}) \\
&= \frac{1}{2}\log 2\pi\sigma^2 + \frac{1}{2\sigma^2}(y-\mathbf{w}^T\mathbf{x})^2 \\
\Rightarrow \hat{\mathbf{w}} &= \arg\max_{\mathbf{w}} P(y_{1:n}|\mathbf{X}_{1:n},\mathbf{w},\sigma^2) = \arg\min_{\mathbf{w}} \sum_{i=1}^{n}(\frac{1}{2}\log 2\pi\sigma^2 + \frac{1}{2\sigma^2}(y_i-\mathbf{w}^T\mathbf{x}_i)^2) \\
&= \arg\max_{\mathbf{w}} \frac{n}{2}\log 2\pi\sigma^2 + \frac{1}{2\sigma^2}\sum_{i=1}^{n}(y_i-\mathbf{w}^T\mathbf{x}_i)^2 = \arg\min_{\mathbf{w}} \sum_{i=1}^{n}(y_i-\mathbf{w}^T\mathbf{x}_i)^2
\end{aligned}$$

The negative log likelihood for the conditional linear Gaussian is given by:

$$L(\mathbf{w}) = -\log P(y_1,\ldots,y_n|\mathbf{x}_1,\ldots,\mathbf{x}_n,\mathbf{w}) = \frac{n}{2}\log(2\pi\sigma^2) + \sum_{i=1}^{n}\frac{(y_i-\mathbf{w}^T\mathbf{x}_i)^2}{2\sigma^2}$$

Thus, under the conditional linear Gaussian assumption, maximizing the likelihood is equivalent to least squares estimation, i.e.:

$$\arg\max_{\mathbf{w}} P(y_1,\ldots,y_n|\mathbf{x}_1,\ldots,\mathbf{x}_n,\mathbf{w}) = \arg\min_{\mathbf{w}} \sum_{i=1}^{n}(y_i-\mathbf{w}^T\mathbf{x}_i)^2$$

More generally, the MLE for data with Gaussian iid noise can be derived as follows. Suppose $\mathcal{H} = \{h : \mathcal{X} \to \mathbb{R}\}$ is a class of functions. Assuming that $P(Y = y|\mathbf{X} = \mathbf{x}) = \mathcal{N}(y|h*(\mathbf{x}),\sigma^2)$ for some function $h* : \mathcal{X} \to \mathbb{R}$ and some $\sigma^2 > 0$ the MLE for data $D = \{(\mathbf{x}_1,y_1),\ldots,(\mathbf{x}_n,y_n)\}$ in $\mathcal{H}$ is given by:

$$\hat{h} = \arg\min_{h\in\mathcal{H}} \sum_{i=1}^{n}(y_i - h(\mathbf{x}_i))^2$$

The fact that the MLE is given by the least squares solution, assuming that the noise is idd Gaussian with constant variance is useful since it satisfies several nice statistical properties:

Consistency Parameter estimate converges to true parameters in probability

Asymptotic efficiency the smallest variance among all well-behaved estimators for large n.

Asymptotic normality

However, all these properties are asymptotic. For finite n, we still need to be careful about overfitting.

14.4 Bias variance tradeoff

The bias variance tradeoff is defined as:

$$\text{Prediction error} = \text{Bias}^2 + \text{Variance} + \text{Noise}$$

where:

Bias : Excess risk of best model considered compared to minimal achievable risk knowing $P(\mathbf{X},Y)$, i.e. given infinite data.

Variance : Risk incurred due to estimating model from limited data.

Noise : Risk incurred by optimal model (i.e. irreducible error).

To investigate where bias comes from, the MLE solution depends on training data D:

$$\hat{h} = \hat{h}_D = \arg\min_{h \in \mathcal{H}} \sum_{(\mathbf{x},y)\in D} (y - h(\mathbf{x}))^2$$

But training data D is itself random (drawn iid fro P). We might want to choose H to have small bias (i.e., have small squared error on average, see red part of the equation below).

$$\mathbb{E}_{\mathbf{X}}[\mathbb{E}_D \hat{h}_D(\mathbf{X}) - h*(\mathbf{X})]^2$$

To see where variance comes from, the MLE solution depends on training data D:

$$\hat{h} = \hat{h}_D = \arg\min_{h \in \mathcal{H}} \sum_{(\mathbf{x},y)\in D} (y - h(\mathbf{x}))^2$$

This estimator is itself random, and has some variance:

$$\mathbb{E}_{\mathbb{X}}\text{Var}_D[\hat{h}_D(\mathbf{X})]^2 = \mathbb{E}_{\mathbb{X}}\mathbb{E}_D[\hat{h}_D(\mathbf{X}) - \mathbb{E}_D, \hat{h}_D, (\mathbf{X})]^2$$

Finally, even if we know the Bayes' optimal hypothesis h*, we'd still incur some error due to noise:

$$\mathbb{E}_{\mathbf{X},Y}[(Y - h*(\mathbf{X}))^2]$$

This error is irreducible, i.e., independent of the choice of the hypothesis class.

In summary:

$$\begin{aligned}\mathbb{E}_D\mathbb{E}_{\mathbf{X},Y}[(Y - \hat{h}_D(\mathbf{X}))^2] = &\mathbb{E}_{\mathbf{X}}[\mathbb{E}_D \hat{h}_D(\mathbf{X}) - h*(\mathbf{X})]^2 \\ &+ \mathbb{E}_{\mathbf{X}}\mathbb{E}_D[\hat{h}_D(\mathbf{X}) - \mathbb{E}_D, \hat{h}_D, (\mathbf{X})]^2 \\ &+ \mathbb{E}_{\mathbf{X},Y}[Y - h*(\mathbf{X})]^2\end{aligned}$$

where the terms are expected risk, bias, variance and noise and ideally we wish to find an estimator that simultaneously minimizes bias and variance.

14.5 Bias and variance in regression

The MLE for linear regresison is unbiased (if $h*$ is in class $\mathcal{H}$). Furhtermore, it is the minimum variance estimator among all unbiased estimators according to the Gauss-Markov theorem. Hoever, we have already seen that the least-squares solution can overfit. Thus, it is sometimes important to trade some bias for a reduction of variance through regularization, like Ridge and Lasso.

14.6 Bias in Bayesian modelling

It is possible to introduce bias by expressing assumptions on parameters through a Bayesian prior. For example, let's assume $\mathbf{w} \sim \mathcal{N}(0, \beta^2\mathbf{I})$ which is equivalent to $w_i \sim \mathcal{N}(0, \beta^2) \forall i \in d$. Then, the posterior distribution of $\mathbf{w}$ is given using Bayes' rule by:

$$\begin{aligned}P(\mathbf{w}|\mathbf{X}_{1:n}y_{1:n}) &= \frac{P(\mathbf{w}|X_{1:n})P(y_{1:n}|w_1, \mathbf{X}_{1:n})}{P(y_{1:n}|\mathbf{X}_{1:n})} \\ &= \frac{P(\mathbf{w})P(y_{1:n}|w_1, \mathbf{X}_{1:n})}{P(y_{1:n}|\mathbf{X}_{1:n})}\end{aligned}$$

The second line was obtained assuming the independence of $\mathbf{w}$ of $\mathbf{x}$. With this we can determine which parameters $\mathbf{w}$ are most likely a posteriori, which is equivalent to the maximum a posteriori estimate. I.e., given

$$P(\mathbf{w}|\mathbf{x}_1, \ldots, \mathbf{x}_n, y_1, \ldots, y_n) = \frac{P(\mathbf{w})P(y_1, \ldots, y_n|\mathbf{x}_1, \ldots, \mathbf{x}_n, \mathbf{w})}{P(y_1, \ldots, y_n|\mathbf{x}_1, \ldots, \mathbf{x}_n)}$$

we want:

$$\arg\max_{\mathbf{w}} P(\mathbf{w}|\mathbf{X}_{1:n}, y_{1:n}) = \arg\min_{\mathbf{w}} -\log P(\mathbf{w}) - \log P(\mathbf{Y}_{1:n}|\mathbf{X}_{1:n}, \mathbf{w}) + \log P(Y_{i:n}|X_{1:n}) \quad (19)$$

Note: we can neglect $\log P(Y_{i:n}|X_{1:n})$ because it is independent of $\mathbf{w}$.

$$\begin{aligned} -\log P(\mathbf{w}) &= -\log \prod_{i=1}^{d} P(w_i) = -\sum_{i=1}^{d} \log \mathcal{N}(w_i; 0, \beta^2) \\ &= -\sum_{i=1}^{d} \log \frac{1}{\sqrt{2\pi\beta^2}} \exp(\frac{-w_i^2}{2\beta^2}) = \frac{d}{2}\log 2\pi\beta^2 + \frac{1}{2\beta^2}\sum_{i=1}^{d} w_i^2 \\ &= c + \frac{1}{2\beta^2}\left\|w\right\|_2^2 \end{aligned}$$

Coming back to equation to equation 19, we now can derive:

$$\begin{aligned} (*) &= \arg\min_{\mathbf{w}} \frac{1}{2\beta^2}\left\|w\right\|_2^2 + \frac{1}{2\beta^2}\sum_{i=1}^{n}(y_i - \mathbf{w}^T x_i)^2 \\ &= \arg\min_{\mathbf{w}} \frac{\sigma^2}{\beta^2}\left\|\mathbf{w}\right\|_2^2 + \sum_{i=1}^{n}(y_i - \mathbf{w}x_i)^2 \end{aligned}$$

which is equivalent to Ridge regression for $\lambda = \frac{\sigma^2}{\beta^2}$.

14.7 Ridge regression is equivalent to MAP estimation

Ridge regression can be understood as finding the MAP parameter estimate for a linear regression problem, assuming that the noise is iid Gaussian and the prior on the model parameters is Gaussian, i.e.:

$$\arg\min_{\mathbf{w}} \sum_{i=1}^{n}(y_i - \mathbf{w}^T\mathbf{x}_i)^2 + \lambda\left\|\mathbf{w}\right\|_2^2 \equiv \arg\max_{\mathbf{w}} P(\mathbf{w}) \prod_i P(y_i|\mathbf{x}_i, \mathbf{w})$$

More generally, regularized estimation can often be understood as MAP inference:

$$\begin{aligned} \arg\min_{\mathbf{w}} \sum_{i=1}^{n} \ell(\mathbf{w}\mathbf{x}_i; \mathbf{x}_i, y_i) + C(\mathbf{w}) &= \arg\max_{\mathbf{w}} \prod_i P(y_i|\mathbf{x}_i, \mathbf{w})P(\mathbf{w}) \\ &= \arg\max_{\mathbf{w}} P(\mathbf{w}|D) \end{aligned}$$

where $C(\mathbf{w}) = -\log P(\mathbf{w})$ and $\ell(\mathbf{w}^T\mathbf{x}_i; \mathbf{x}_i, y_i) = -\log P(y_i|\mathbf{x}_i, \mathbf{w})$. This perspective allows changing priors ($\approx$ regularizers) and likelihoods ($\approx$ loss functions).

14.8 L1-regularization

The prior that corresponds to l1-regularization is the Laplace prior. Its shape can is depicted in figure 36 and is given in equation 20.

$$p(x; \mu, b) = \frac{1}{2b}\exp(-\frac{|x - \mu|}{b}) \quad (20)$$

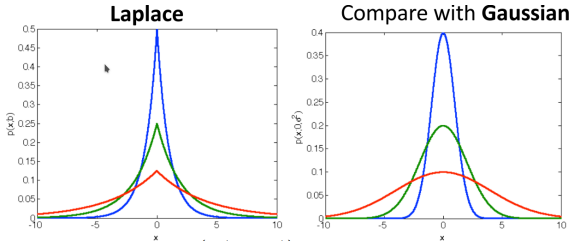


Figure 36: Laplace prior compared to the Gaussian prior.

14.9 Student-t likelihood

It is possible to introduce robustness by changing the likelihood (=loss) function, for instance using the non-standardized Student's-t likelihood given by:

$$P(y|\mathbf{x}, \mathbf{w}, \nu, \sigma^2) = \frac{\Gamma(\frac{\nu+1}{2})}{\sqrt{\pi\nu\sigma^2}\Gamma(\frac{\nu}{2})}(1 + \frac{(y - \mathbf{w}^T\mathbf{x})^2}{\nu\sigma^2})^{-\frac{\nu+1}{2}}$$

which yields heavier tails than normal distributions.

14.10 Bayes' optimal predictor

Assuming the data is generated iid, i.e. $(x_i, xy_i) \sim P(X, Y)$, the hypothesis h^* $R(h) = \mathbb{E}[Y|\mathbf{X} = \mathbf{x}]$ is given by the conditional mean $h^*(x) = \mathbb{E}[Y|\mathbf{X} = \mathbf{x}]$ which is called the Bayes' optimal predictor for the squared loss. In this setting, one strategy for estimating a predictor from training data is to estimate the conditional distribution $\hat{P}(X|\mathbf{X})$ and then, for test point $\mathbf{x}$, the predicted label:

$$\hat{y} = \hat{\mathbb{E}}[Y|\mathbf{X} = \mathbf{x}] = \int \hat{P}(y|\mathbf{X} = \mathbf{x})ydy$$

Estimating the conditional distribution can be done using a common approach, which consists in (i) choose a particular parametric form from $\hat{P}(Y|\mathbf{X}, \theta)$ and then optimize the parameters. This is done through maximum conditional likelihood estimation:

$$\theta^* = \arg\max_\theta \hat{P}(y_1, \ldots, y_n|\curvearrowleft_1, \ldots, \mathbf{x}_n, \theta)$$

In this context, Ridge regression can be understood as finding the MAP parameter estimate for a linear regression problem, assuming that:

- The noise $P(y|\mathbf{x}, \mathbf{w})$ is iid Gaussian and
- The prior $P(\mathbf{w})$ on the model parameters $\mathbf{w}$ is Gaussian.

Then, we have the equivalence:

$$\arg\min_{\mathbf{w}} \sum_{i=1}^{n}(y_i - \mathbf{w}^T\mathbf{x}_i)^2 + \lambda \|\mathbf{w}\|_2^2 \equiv \arg\max_{\mathbf{w}} P(\mathbf{w}) \prod_i P(y_i|\mathbf{x}_i, \mathbf{w})$$

More generally, regularized estimation can often be understood as MAP inference:

$$\begin{aligned}\arg\min_{\mathbf{w}} \sum_{i=1}^{n} \ell(\mathbf{w}^T\mathbf{x}_i; \mathbf{x}_i, y_i) + C(\mathbf{w}) &= \arg\max_{\mathbf{w}} \prod_i P(y_i|\mathbf{x}_i, \mathbf{w})P(\mathbf{w}) \\ &= \arg\max_{\mathbf{w}} P(\mathbf{w}|D)\end{aligned}$$

where $C(\mathbf{w}) = -\log P(\mathbf{w})$ and $\ell(\mathbf{w}^T\mathbf{x}_i; \mathbf{x}_i, y_i) = -\log P(y_i|\mathbf{x}_i, \mathbf{w})$. This allows changing priors, which are equivalent to regularizers and likelihoods, which are equivalent to loss functions.

14.11 Probablistic modelling (risk) in classification

In classification, risk is defined as:

$$R(h) = \mathbb{E}_{\mathbf{X},Y}[[Y \neq h(\mathbf{X})]]$$

which is equal to 1 if $y \neq h(\mathbf{X})$ and 0 otherwise.
Now, suppose we know $P(\mathbf{X}, Y)$, which h minimizes the risk then?

$$h^*(\mathbf{x}) = \arg\min_{\hat{y}} \underbrace{\mathbb{E}[[Y \neq \hat{y}|\mathbf{X} = \mathbf{x}]]}_{\ell(\hat{y})}$$

Expanding $\ell(\hat{y})$ yields:

$$\ell(\hat{y}) = \sum_{y=1}^{C} P(Y = y|\mathbf{X} = \mathbf{x})[y \neq \hat{y}] = \sum_{y_i, y \neq \hat{y}} P(Y = y|\mathbf{X} = \mathbf{x}) = 1 - P(Y = \hat{y}|\mathbf{X} = \mathbf{x})$$

So $h^*(\mathbf{x}) = \arg\max_{\hat{y}} P(Y = \hat{y}|\mathbf{X} = \mathbf{x})$.

In the context of logistic regression, we want to use a generalized linear model for the class probability. We know that $Y \in \{+1, -1\}$. The classification of a positive example is given by:

$$P(Y = 1|\mathbf{x} = \sigma(\mathbf{w}^T\mathbf{x}))$$

where:

$$\sigma(z) = \frac{1}{1 + \exp(-z)}$$

Note that this is equivalent to saying that $Y|\mathbf{X} = \mathbf{x} \sim \text{Bernoulli}(\sigma(\mathbf{w}^T\mathbf{x}))$ The derivation of $P(Y = -1|\mathbf{X})$ is given below:

$$\begin{aligned} P(Y = -1|\mathbf{x}) &= 1 - P(Y = 1|\mathbf{x}) \\ &= 1 - \frac{1}{1 + \exp(-\mathbf{w}^T\mathbf{x})} \\ &= \frac{\exp(-\mathbf{w}^T\mathbf{x})}{1 + \exp(-\mathbf{w}^T\mathbf{x})} \\ &= \frac{1}{1 + \exp(\mathbf{w}^T\mathbf{x})} \end{aligned}$$

So, we get the general classification for $P(Y = y|\mathbf{x}) = \frac{1}{1+\exp(-y \cdot \mathbf{w}^T\mathbf{x})}$.

Logistic regression (a classification method) replaces the assumption of Gaussian noise (squared loss) by i.i.d. Bernoulli noise, and yields the following classification equation:

$$P(y|\mathbf{x}, \mathbf{w}) = \text{Bernoulli}(y; \sigma(\mathbf{w}^T\mathbf{x}))$$

vs.

$$P(y|\mathbf{x}, \mathbf{w}) = \mathcal{N}(y; \mathbf{w}^T\mathbf{x}, \sigma^2)$$

for least squares regression. Similar to least square, the parameters $\mathbf{w}$ can be estimated using MLE or MAP.

The MLE for logistic regression is given by the following:

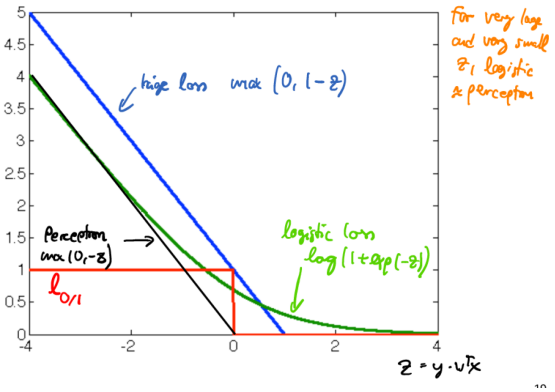


Figure 37: Comparing logistic loss to other loss functions.

$$\hat{w} \in \arg\max_{\mathbf{w}} P(D|\mathbf{w})^{\text{iid}} = \arg\max_{\mathbf{w}} \prod_{i=1}^{n} P(y_i|\mathbf{x}_i, \mathbf{w}) \tag{21}$$

$$= \arg\min_{\mathbf{w}} -\sum_{i=1}^{n} \log P(y_i|x_i, \mathbf{w}) \tag{22}$$

$$= \arg\min_{\mathbf{w}} \underbrace{\sum_{i-1}^{n} \underbrace{\log(1 + \exp(-y_i \mathbf{w}^T \mathbf{x}_i))}_{\ell_{\text{logistic}}(\mathbf{w}; \mathbf{x}_i, y_i)}}_{\hat{R}(\mathbf{w})} \tag{23}$$

Where the loss function we're trying to optimize is given by the following:

$$-\log P(y_i|\mathbf{x}_i, \mathbf{w}_i) = -\log \frac{1}{1 + \exp(-y\mathbf{w}^T \mathbf{x}_i)} = \log(1 + \exp(-y\mathbf{w}^T \mathbf{x}_i))$$

Plugging this result back into equation 21 yields:

$$P(y_i|\mathbf{x}_i, \mathbf{w}) = \frac{P(y_i, \mathbf{x}_i|\mathbf{w})}{P(\mathbf{x}_i, \mathbf{y})} \underbrace{= C_i P(y_i, x_i|\mathbf{w})}_{\text{if } P(\mathbf{x}_i|\mathbf{w}) = P(\mathbf{x}_i) \to \mathbf{x} \text{ independent of } \mathbf{w}}$$

So, the negative log likelihood, which is equivalent to our objective function, is given by:

$$\hat{R}(\mathbf{w}) = \sum_{i=1}^{n} \log(1 + \exp(-y_i \mathbf{w}^T \mathbf{x}_i))$$

which is a convex function, which means that we can use convex optimization techniques like SGD.

Comparing the logistic loss to other loss functions already reviewed yields Figure 37.

The loss for a data point $(\mathbf{x}, y)$ is defined by:

$$\ell(\mathbf{w}) = \log(1 + \exp(-y\mathbf{w}^T \mathbf{x}))$$

and its gradient is defined by:

$$\begin{aligned}
\nabla_{\mathbf{w}} \ell(\mathbf{w}) &= \frac{1}{1 + \exp(-y\mathbf{w}^T \mathbf{x})} \cdot \exp(-y\mathbf{w}^T \mathbf{x}) \cdot (-y \cdot \mathbf{x}) \\
&= \frac{\exp(-y\mathbf{w}^T \mathbf{x})}{1 + \exp(-y\mathbf{w}^T \mathbf{x})} (-y \cdot \mathbf{x}) \\
&= \underbrace{\frac{1}{1 + \exp(y\mathbf{w}^T \mathbf{x})}}_{P(Y \neq y|\mathbf{x})} (-y \cdot \mathbf{x})
\end{aligned}$$

Once the gradient is defined, we can use SGD for logistic regression. The steps are:

1. Initialize $\mathbf{w}$
2. For $t = 1, 2, \ldots$:
   (a) Pick data point $(\mathbf{x}, y)$ uniformly at random from data D.
   (b) We compute the probability of misclassification with the current model, equivalent to:

$$\hat{P}(Y = -y|\mathbf{w}, \mathbf{x}) = \frac{1}{1 + \exp(y\mathbf{w}^T\mathbf{x})}$$

   (c) Take a gradient step:

$$\mathbf{w} \leftarrow \mathbf{w} + \eta_t y\mathbf{x}\hat{P}(Y = -y|\mathbf{w}, \mathbf{x})$$

To control model complexity, regularization is desired. Thus, instead of solving the MLE, which corresponds to:

$$\min_{\mathbf{w}} \sum_{i=1}^{n} (\log(1 + \exp(-y_i\mathbf{w}^T\mathbf{x}_i)))$$

we solve the regularized problem, which corresponds to either the Gaussian prior (L2):

$$\min_{\mathbf{w}} \sum_{i=1}^{n} \log(1 + \exp(-y_i\mathbf{w}^T\mathbf{x}_i)) + \lambda \|\mathbf{w}\|_2^2$$

or Laplacian prior (L1):

$$\min_{\mathbf{w}} \sum_{i=1}^{n} \log(1 + \exp(-y_i\mathbf{w}^T\mathbf{x}_i)) + \lambda \|\mathbf{w}\|_1$$

For L2-regularized SGD we add the following:

1. Initialize $\mathbf{w}$
2. For $t = 1, 2, \ldots$:
   (a) Pick data point $(\mathbf{x}, y)$ uniformly at random from data D.
   (b) We compute the probability of misclassification with the current model, equivalent to:

$$\hat{P}(Y = -y|\mathbf{w}, \mathbf{x}) = \frac{1}{1 + \exp(y\mathbf{w}^T\mathbf{x})}$$

   (c) Take a gradient step:

$$\mathbf{w} \leftarrow \mathbf{w}(1 - 2\lambda\eta_t) + \eta_t y\mathbf{x}\hat{P}(Y = -y|\mathbf{w}, \mathbf{x})$$

In summary regularized logistic regression has two steps:

- Learning
  - Find optimal weights by minimizing logistic loss + regularizer, equivalent to:

$$\hat{\mathbf{w}} = \arg\min_{\mathbf{w}} \sum_{i=1}^{n} \log(1 + \exp(-y_i\mathbf{w}^T\mathbf{x}_i)) + \lambda \|\mathbf{w}\|_2^2 = \arg\max_{\mathbf{w}} P(\mathbf{w}|\mathbf{x}_1, \ldots, \mathbf{x}_n, y_1, \ldots, y_n)$$

- Classification
  - Use a conditional distribution:

$$\hat{P}(y|\mathbf{x}, \hat{\mathbf{w}}) = \frac{1}{1 + \exp(-y\hat{\mathbf{w}}^T\mathbf{x})}$$

  - Then, it's possible to predict the more likely class label:

$$\hat{y} = \sigma(\hat{\mathbf{w}}^T\mathbf{x})$$

Let us derive that last result:

$$\begin{aligned}\arg\max_{\hat{y}} P(\hat{y}|\mathbf{x}, \hat{\mathbf{w}}) &= \arg\max_{\hat{y}} \frac{1}{1+\exp(-\hat{y}\hat{\mathbf{w}}^T)} \\ &= \arg\min_{\hat{y}} \exp(-\hat{y}\hat{\mathbf{w}}^T\mathbf{x}) \\ &= \arg\min_{\hat{y}} -\hat{y}\hat{\mathbf{w}}^T\mathbf{x} \\ &= \arg\min_{\hat{y}\in\{+1,-1\}} -\hat{y}\hat{\mathbf{w}}^T\mathbf{x} = \sigma(\mathbf{w}^T\mathbf{x})\end{aligned}$$

Additional desirable features of logistic regression:

- Can kernelize
- Can apply logistic loss function to neural networks in order to have them output probabilities
- Natural multi-class variants

For the kernelized logistic regression, we follow these steps:

- Learning:
  - Find optimal weights by minimizing logistic loss + regularizer

$$\hat{\alpha} = \arg\min_{\alpha} \sum_{i=1}^{n} \log(1+\exp(-y_i\alpha^T\mathbb{K}_i)) + \lambda\alpha^T\mathbf{K}\alpha$$

    where $\mathbf{K} = (\mathbf{K}_1|\ldots|\mathbf{K}_n)$

- Classification:
  - Use conditional distribution

$$\hat{P}(y|\mathbf{x}, \hat{\mathbf{w}}) = \frac{1}{1+\exp(-y\sum_{j=1}^{n}\alpha_j k(\mathbf{x}_j, \mathbf{x}))}$$

  - Predict the most likely label

We can further extend multi-class logistic regression to a multi-class setting by maintaining one weight vector per class and model. We then have:

$$P(Y=i|\mathbf{x}, \mathbf{w}_1, \ldots, \mathbf{w}_c) = \underbrace{\frac{\exp(\mathbf{w}_i^T\mathbf{x})}{\sum_{j=1}^{c}\exp(\mathbf{w}_j^T\mathbf{x})}}_{\hat{p}_i}$$

By default these weight vectors are not unique, and it is possible to force uniqueness by setting $w_c = 0$, which recovers logistic regression as a special case.

The corresponding function (cross-entropy loss) is equivalent to:

$$\ell(y; \mathbf{x}, \mathbf{w}_1, \ldots, \mathbf{w}_c) = -\log P(Y=y|\mathbf{x}, \mathbf{w}_1, \ldots, \mathbf{w}_c)$$

The multiclass classification illustration is shown in figure

Side note: training a neural network with multiple classes entails having multiple outputs and then compute the cross-entropy loss as well:

$$\ell(Y=i; f_1, \ldots, f_c) = -\log\frac{\exp(f_i)}{\sum_{j=1}^{c}\exp(f_i)}$$

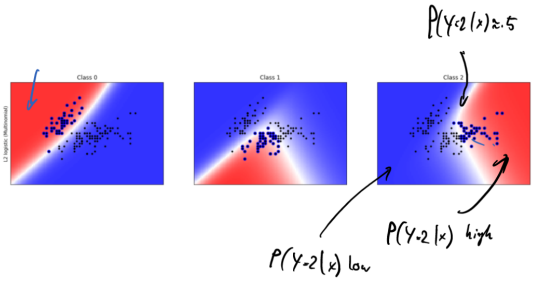


Figure 38: Multiclass classification with a probabilistic interpretation.

Method	SVM/Perceptron	Logistic regression
Advantages	Sometimes higher classification accuracy; Sparse sol's	Can obtain probabilities
Disadvantages	Can't easily get class probabilities	Dense solutions

Table 2: (Dis)advantages in SVM vs. logistic regression

14.12 Bayesian Learning Theory

In the context of Bayesian decision theory, we consider the following. Given:

- A conditional distribution over labels $P(y|\mathbf{x})$ where $y \in \mathcal{Y}$ and $\mathcal{Y}$ can either be $+1, -1$, $1, \ldots, c$, $\mathbb{R}$
- A set of actions $\mathcal{A}$ where $\mathcal{A} \neq \mathcal{Y}$
- A cost function $C : \mathcal{Y} \times \mathcal{A} \to \mathbb{R}$

Bayesian decision theory recommends to pick the action that minimizes the expected cost:

$$a^* = \arg\min_{a \in \mathcal{A}} \mathbb{E}_y[C(y, a)|\mathbf{x}]$$

If we had access to the true distribution $P(y|\mathbf{x})$ this implements the Bayesian optimal decision. In practice, however, we can only estimate the distribution using e.g. logistic regression.

Making the optimal decision for logistic regression is achieved as follows:

- We have an estimated conditional distribution:

$$\hat{P}(y|\mathbf{x}) = \text{Bernoulli}(y; \sigma(\hat{\mathbf{w}}^T\mathbf{x}))$$

- An action set:

$$\mathcal{A} = +1, -1$$

- A cost function:

$$C(y, a) = \underbrace{[y \neq a]}_{\begin{cases} 1 \text{ if } y \neq 0 \\ 0 \text{ otherwise} \end{cases}}$$

- Then the action that minimizes the expected cost

$$a^* = \arg\min_{a \in \mathcal{A}} \mathbb{E}_y[C(y, a)|\mathbf{x}] = \sum_y P(y|\mathbf{x})[y \neq a] = \frac{1}{a + \exp(a \cdot \mathbf{w}^T\mathbf{x})}$$

is the most likely class:

$$a^* = \arg\max_y \hat{P}(y|\mathbf{x}) = \text{sign}(\mathbf{w}^T\mathbf{x})$$

Proof.

$$\begin{aligned} a^* &= \arg\min_a \frac{1}{1+\exp(a\mathbf{w}^T\mathbf{x})} \\ &= \arg\max_a 1+\exp(a\mathbf{w}^T\mathbf{x}) \\ &= \arg\max_{a\in\{-1,+1\}} 1+\exp(a\mathbf{w}^T\mathbf{x}) \\ &= \arg\max_y \hat{P}(y|\mathbf{x}) = \text{sign}(\mathbf{w}^T\mathbf{x}) \end{aligned}$$

□

For asymmetric costs, we amend the procedure as follows:

1. Estimated conditional distribution

$$\hat{P}(y|\mathbf{x}) = \text{Bernoulli}(y; \sigma(\hat{\mathbf{w}}^T\mathbf{x}))$$

2. An action set:

$$\mathcal{A} = +1, -1$$

3. A cost function:

$$C(y,a) = \begin{cases} c_{FP} & \text{if } y=-1 \text{ and } a=+1 \\ c_{FN} & \text{if } y=+1 \text{ and } a=-1 \\ 0 & \text{otherwise} \end{cases}$$

4. Then the action that minimizes the expected cost is:

$$\begin{aligned} c_+ = \mathbb{E}_y[C(y,+1)|\mathbf{x}] &= P(y=-1|\mathbf{x})\cdot c_{FP} + \underbrace{P(y=+1|\mathbf{x})}_{p}\cdot 0 \\ &= (1-p)\cdot c_{FP} \\ c_- = \mathbb{E}_y[C(y,-1)|\mathbf{x}] &= p\cdot C_{FN} + (1-p)\cdot 0 = p\cdot C_{FN} \end{aligned}$$

In this context, if we predict $+1$, we can derive the following:

$$\begin{aligned} \text{predict } +1 &\Leftrightarrow c_+ < c_- \\ &\Leftrightarrow (1-p)c_{FP} < pc_{FN} \\ &\Leftrightarrow p > \frac{C_{FP}}{C_{FP}+C_{FN}} \end{aligned}$$

Asymmetric costs are shown in Figure 39

It's also possible to made a "doubtful" logistic regression by making the following adjustments to the steps described above:

- We have an estimated conditional distribution:

$$\hat{P}(y|\mathbf{x}) = \text{Bernoulli}(y; \sigma(\hat{\mathbf{w}}^T\mathbf{x}))$$

- An action set:

$$\mathcal{A} = +1, -1, D$$

- A cost function:

$$C(y,a) = \begin{cases} [y \neq a] & \text{if } a \in \{+1,-1\} \\ c & \text{if } a = D \end{cases}$$

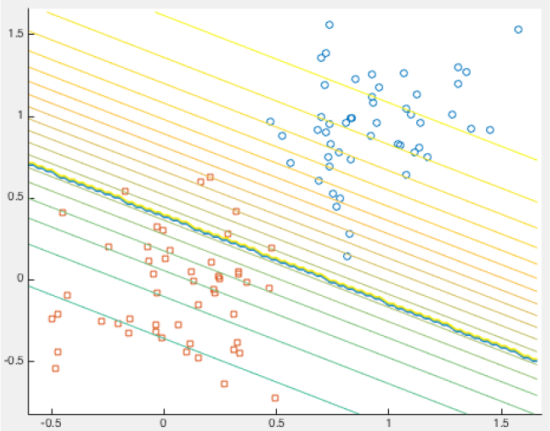


Figure 39: Asymetric cost illustration

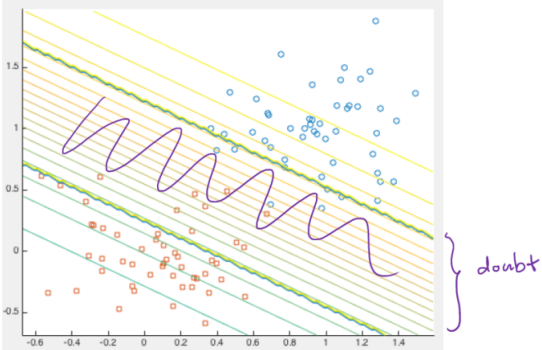


Figure 40: Area of doubt in doubtful logistic regression/

- Then the action that minimizes the expected cost

$$a^* = \arg\min_{a \in \mathcal{A}} \mathbb{E}_y[C(y, a)|\mathbf{x}]$$

  is given by:

$$a^* = \begin{cases} y & \text{if } \hat{P}(y|\mathbf{x}) \geq 1 - c \\ D & \text{otherwise} \end{cases}$$

This corresponds to picking a given class if confidence enough.

For least squares, the optimal decision is given by:

- Estimated conditional distribution

$$\hat{P}(y|\mathbf{x}) = \mathcal{N}(y; \hat{\mathbf{w}}^T\mathbf{x}, \sigma^2)$$

- Action set is $\mathcal{A} = \mathbb{R}$
- Cost function: $C(y, a) = (y - a)^2$
- Then the action that minimizes the expected cost is:

$$a^* = \arg\min_{a \in \mathcal{A}} \mathbb{E}_y[C(y, a)|\mathbf{x}]$$

  is the conditional mean:

$$\begin{aligned} a^* = \mathbb{E}_y[y|\mathbf{x}] &= \int \hat{P}(y|\mathbf{x})dy \\ &= \hat{\mathbf{w}}^T\mathbf{x} \end{aligned}$$

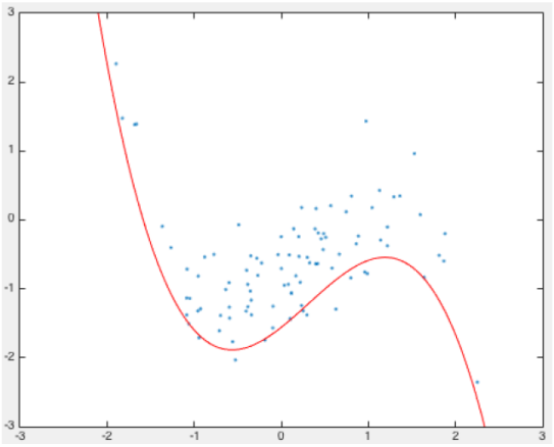


Figure 41: Asymmetric costs for regression

And the asymmetric cost for regression is given by:

- Estimated conditional distribution

$$\hat{P}(y|\mathbf{x}) = \mathcal{N}(y; \hat{\mathbf{w}}^T\mathbf{x}, \sigma^2)$$

- Action set is $\mathcal{A} = \mathbb{R}$
- Cost function:

$$C(y,a) = c_1 \underbrace{\max(y-a,0)}_{underestimation} + c_@ \underbrace{\max(a-y,0)}_{overestimation}$$

$$= (y-a)^2$$

- Then the action that minimizes the expected cost is:

$$a^* = \arg\min_{a\in\mathcal{A}} \mathbb{E}_y[C(y,a)|\mathbf{x}]$$

is:

$$a^* = \hat{\mathbf{w}}^T\mathbf{x} + \sigma \cdot \phi^{-1}(\frac{c_1}{c_1+c_2})$$

where ϕ^{-1} is the inverse Gaussian CDF.

Note: asking an expert for labels is expensive, so we want to minimize the number of labels. A strategy to pick the examples we are most uncertain about is called uncertainty sampling.

- Given $D = (\mathbf{x}_i, \mathbf{y}_i)_{i=1}^n$ you estimate $\hat{P}(y|\mathbf{x})$
- For every $t = 1, 2, 3, \ldots$:

$$\hat{P}(y_i|\mathbf{x}_i)$$

given data D.

- we pick the sample we are most uncertain about:

$$i_t \in \arg\min_i |0.5 - \hat{P}(Y_i|\mathbf{x}_i)|$$

- We query the label y_{it}, and set $D \leftarrow D \cup \{(\mathbf{x}_{i_t}, y_{i_t})\}$

An example of obtaining the uncertainty score u_j in logistic regression is:

$$u_j = -|\mathbf{w}^T \mathbf{x}_j|$$

Note that active learning and uncertainty sampling violate the i.i.d. assumption. It's also possible to get stuck with bad models. More advanced selection criteria are available.

Bayesian decision theory provides a principled way to derive decision rules from conditional distributions.

It can accomodate more complex settings like doubt, asymmetric losses, and active learning.

In summary:

1. Start with statistical assumptions on data. These data points are modeled iid but can be relaxed.
2. Choose likelihood function, e.g. Gaussian, student-t, logistic, exponential
3. We choose a prior, e.g. Gaussian, Laplace, exponential
4. Optimize for MAP parameters
5. Choose hyperparameters (variance, etc) through cross-validation.
6. Make predictions via BDT.

15 Discriminative vs. Generative modelling

15.1 Motivating example

So far, we have considered only methods that estimate conditional distributions:

$$P(y|\mathbf{x})$$

Examples: linear regression, logistic regression, etc...

Such models do not attempt to model $P(\mathbf{x})$. Thus, they will not be able to detect outliers, i.e. unusual points for which $P(\mathbf{x})$ is very small.

The crucial difference between discriminative and generative modelling is the following. In discriminative models, we estimate:

$$P(y|\mathbf{x})$$

and in generative models, we seek to estimate the joint distribution

$$P(y, \mathbf{x})$$

It is possible to derive the conditional distribution from the joint distribution, but not vice versa.

$$(y|\mathbf{x}) \rightarrow P(y|\mathbf{x}) = \frac{P(\mathbf{x}, y)}{P(\mathbf{x})}$$

where:

$$P(\mathbf{x}) = \sum_{y'} P(\mathbf{x}, y')$$

The typical approach for generative modelling is to follow these steps:

1. Estimate priors on labels $P(y)$[2]
2. Estimate the conditional distribution $P(\mathbf{x}|y)$ for each class y.
3. Obtain predictive distribution using Bayes' rule:

$$P(y, \mathbf{x}) = \underbrace{\frac{1}{Z}}_{P(\mathbf{x})} \underbrace{P(y)P(\mathbf{x}|y)}_{P(y,\mathbf{x})}$$

Generative modeling attempts to infer the process according to which examples are generated ($P(\mathbf{x}, y)$). First, we generate class label $P(y)$. Then, we generate features given class $P(\mathbf{x}|y)$. An example of generative modelling is shown in figure 42.

[2]Mind the chain rule in probabilities: $P(\mathbf{x}, y) = P(\mathbf{x}|y)P(y) = P(y|\mathbf{x})P(\mathbf{x})$

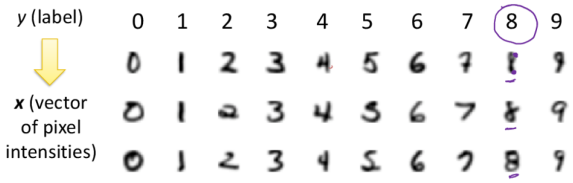


Figure 42: Example of a generative model setting, producing handwritten digits.

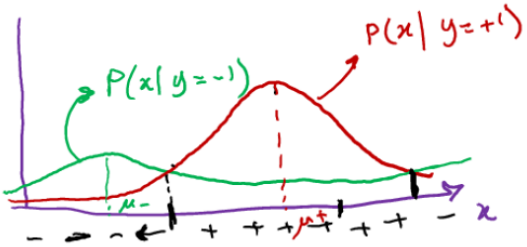


Figure 43: Decision boundaries of a 1D scenario of binary NB classifier

Gaussian naive Bayes classifiers:

Learning Given data $\mathcal{D} = \{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_n, y_n)\}$ we compute:

- MLE for class prior: $\hat{P}(Y = y) = \hat{p}_y = \frac{\text{Count}(Y=y)}{n}$
- MLE for feature distribution:

$$\hat{P}(x_i|y) = \mathcal{N}(x_i; \mu_{y,i}\sigma^2_{y,i})$$

$$\hat{\mu}_{y,i} = \frac{1}{\text{Count}(Y=y)} \sum_{j:y_j=y} \underbrace{x_{j,i}}_{\text{the value of feature } i \text{ for instance } j(x_j, y_j)}$$

$$\hat{\sigma}^2_{y,i} = \frac{1}{\text{Count}(Y=y)} \sum_{j:y_j=y} (x_{j,i} - \hat{\mu}_{y,i})^2$$

- Prediction given a new point $\mathbf{x}$:

$$y = \arg\max_{y'} \hat{P}(y'|\mathbf{x}) = \arg\max \hat{P}(y') \prod_{i=1}^{d} \hat{P}(x_i|y')$$

An example of decision boundaries under the following conditions:

$$\begin{cases} d = 1 \quad 1 \text{ feature } x \\ \mathcal{Y} = \{-1, +1\} \\ P(Y = +1) = P(Y = -1) = 0.5 \\ \mu_+ > \mu_-, \sigma^2_+ < \sigma^2_- \end{cases}$$

is shown in figure 43.

15.2 Generative modelling with Bayes

- Model class label as generated from categorical variable:

$$P(Y = y) = p_y \quad y \in \mathcal{Y} = \{1, \ldots, c\}$$

- Model features as conditionally independent given Y.

$$P(X_1, \dots, X_d|Y) = \prod_{i=1}^{d} P(X_i|Y)$$
$$P(X_1 = \mathbf{x}_1, \dots, X_d = \mathbf{x}_d|Y = y) = \prod_{i=1}^{d} P(X_i = \mathbf{x}_i|Y = y_i)$$

  Which is equivalent to saying that, given a class label, each feature is generated independently of the other features.

  The feature distributions still need to be specified ($P(X_i|Y)$).

The model features can also be generated from conditionally independent Gaussians:

$$P(x_i|y) = \mathcal{N}(x_i|\mu_{y,i}, \sigma^2_{y,i})$$

So the features depend on class y and feature $i \in \{1, \dots, d\}$

To estimate these parameters, we use MLE for $P(y)$. We assume that $\mathcal{Y} = \{-1, +1\}$. We want $P(Y = +1) = p$ and $P(Y = -1) = 1 - p$, and we have $\mathcal{D} = \{(\mathbf{x}_1, y_1), \dots, (\mathbf{x}_n, y_n)\}$.

We can use $\mathcal{D}$ to estimate p via MLE:

$$\max_{p'} P(\mathcal{D}|p') = \prod_{i=1}^{n} p'^{[y_i=1]}(1-p)^{[y_i=0]}$$
$$= p'^{n_+}(1-p')^{n_-} \quad \text{where} \begin{cases} n_+ & \text{number of positive cases in } \mathcal{D} \\ n_- & \text{number of negative cases in } \mathcal{D} \end{cases} = \max_{p'} \ell(p')$$

where

$$\ell(p') = n_+ \log p' + n_- \log(1-p')$$

and

$$\frac{\partial \ell}{\partial p} = \frac{n_+}{p'} + \frac{n_-}{1-p'} = 0 \Leftrightarrow p' = \frac{n_+}{n_+ + n_-}$$

Now, when devising decision rules for binary classification, we want to know:

$$y = \arg\max_{\hat{y'}} P(y'|\mathbf{x})$$

For binary tasks ($c = 2$), this is equivalent to:

$$y = \text{sign}(\log \frac{P(Y = 1|\mathbf{x})}{P(Y = -1|\mathbf{x})})$$

The function $f(\mathbf{x}) = \log \frac{P(Y=1|\mathbf{x})}{P(Y=-1|\mathbf{x})}$ is called the discriminant function.

In case of GNB and $c = 2$ with constant variance, we have:

$$f(\mathbf{x}) = \mathbf{w}^T\mathbf{x} + w_0$$

where:

$$w_0 = log\frac{\hat{p}_+}{1-\hat{p}_+} + \sum_{i=1}^{d} \frac{\hat{\mu}^2_{-,i} - \hat{\mu}^2_{+,i}}{2\hat{\sigma}^2_i}$$
$$w_i = \frac{\hat{\mu}_{+,i} - \hat{\mu}_{-,i}}{\hat{\sigma}^2_{-,i}}$$

There is a difference between the discriminant function vs. the class probability. Let us define

$$
\begin{aligned}
P(Y=+1|\mathbf{x}) &\coloneqq p(x) \\
&\Rightarrow f(\mathbf{x}) = \log \frac{P(\mathbf{x})}{1-p(\mathbf{x})} \\
&\Rightarrow exp(f(\mathbf{x})) = \frac{p(\mathbf{x})}{1-p(\mathbf{x})} \\
&\Rightarrow p(\mathbf{x}) = \frac{\exp(f(\mathbf{x}))}{1+\exp(f\mathbf{x})} = \frac{1}{1+\exp(-f(\mathbf{x}))} = \sigma(f(\mathbf{x}))
\end{aligned}
$$

The same function holds for GNB and logistic regression provided assumptions are not violated; GMG and Log Reg will yield the same prediction.

A significant issue with Naive Bayes model is the conditional independence assumption, which means that features are generated independently given a class label. If there is conditional correlation between class labels, then this assumption is violated. Due to this phenomenon, predictions can become overconfident. This might be fine if we care about the most likely class only, but not if we want to use probabilities for making decision, e.g. with asymmetric losses, etc.

A more general solution is the Gaussian Bayes classifier. Model class label are generated from a categorical variable. Model features are generated by a multivariate Gaussian:

$$P(\mathbf{x}|y) = \mathcal{N}(\mathbf{x}; \mu_y, \sum_y)$$

In GNB we assumed the covariance matrix $\sum_y$ to equate

$$\sum_y = \begin{pmatrix} \sigma_{y,1}^2 & \dots & 0 \\ \vdots & \ddots & \dots \\ 0 & \dots & \sigma_{y,d}^2 \end{pmatrix}.$$

The MLE for the Gaussian Bayes Classifier is, for the class label distribution:

$$\hat{p}_y = \frac{\text{Count}(Y=y)}{n}$$

and for the features:

$$
\begin{aligned}
\hat{P}(\mathbf{x}|y) &= \mathcal{N}(\mathbf{x}; \hat{\mu}_y, \hat{\Sigma}_y) \\
\hat{\mu}_y &= \frac{1}{\text{Count}(Y=y)} \sum_{i:y_i=y} \mathbf{x}_i \\
\hat{\Sigma}_y &= \frac{1}{\text{Count}(Y=y)} \sum_{i:y_i=y} (\mathbf{x}_i - \hat{\mu}_y)(\mathbf{x}_i - \hat{\mu}_y)^T
\end{aligned}
$$

The discriminant function for Gaussian Bayesian classifiers is given by:

$$f(\mathbf{x}) = \log(\frac{p}{1-p}) + \frac{1}{2}\left[\log\frac{|\hat{\Sigma}_-|}{|\hat{\Sigma}_+|} + ((\mathbf{x}-\hat{\mu}_-)^T\Sigma_-^{-1}(\mathbf{x}-\hat{\mu}_-)) - ((\mathbf{x}-\hat{\mu}_+)^T\Sigma_+^{-1}(\mathbf{x}-\hat{\mu}_+))\right] \quad (24)$$

15.3 Fisher's linear discriminant analysis

Suppose we fix $p = 0.5$. Further, we assume that the covariance matrices are equal. Then equation 24 simplifies to:

$$f(\mathbf{x}) = \mathbf{x}^T\hat{\Sigma}^{-1}(\hat{\mu}_+ - \hat{\mu}_-) + \frac{1}{2}(\hat{\mu}_-^T\Sigma^{-1}\hat{\mu}_- - \hat{\mu}_=^T\Sigma^{-1}\hat{\mu}_+)$$

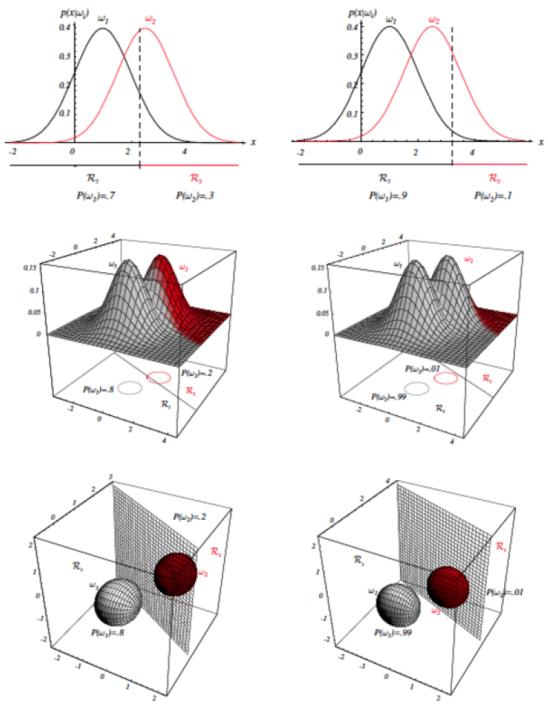


Figure 44: GNB vs NB

Under these assumptions, we predict

$$y = \text{sign}(f(\mathbf{x})) = \text{sign}(\mathbf{w}^T\mathbf{x} + w_0)$$

where

$$\mathbf{w} = \hat{\Sigma}^{-1} * (\hat{\mu}_+ - \hat{\mu}_-)$$
$$w_0 = \frac{1}{2}(\hat{\mu}_-^T\Sigma^{-1}\hat{\mu}_- - \hat{\mu}_+^T\Sigma^{-1}\hat{\mu}_+)$$

This linear classifier is called the linear discriminant analysis. Provided assumptions are met, LDA will make the same predictions as logistic regression.

An illustration of GNB vs. NB is given in Figure 44.

Note on LDA vs. PCA. LDA can be viewed as a projection to a 1-dim subspace that maximizes the ratio of between-class and within class variances. In contrast, PCA with $k = 1$ maximizes the variance of the resulting 1-dim projection. An illustration is shown in Figure 45.

15.4 Quadratic Discriminant analysis

In the general case:

$$f(\mathbf{x}) = \log\frac{p}{1-p} + \frac{1}{2}\left[\log\frac{|\hat{\Sigma}_-|}{\hat{\Sigma}_+}\right] + ((\mathbf{x} - \hat{\mu}_-)^T\Sigma_-^{-1}(\mathbf{x} - \hat{\mu}_-)) - ((\mathbf{x} - \hat{\mu}_+)^T\Sigma_+^{-1}(\mathbf{x} - \hat{\mu}_+))$$

We predict $y = \text{sign}f(\mathbf{x})$. This is called quadratic discriminant analysis. This covers a lot more cases, as seen in figure 46. An Euler Diagram showing GBC, LDA, GNB and LR is shown in Figure 47.

Comparing LDA with regression:

- Fisher's LDA:

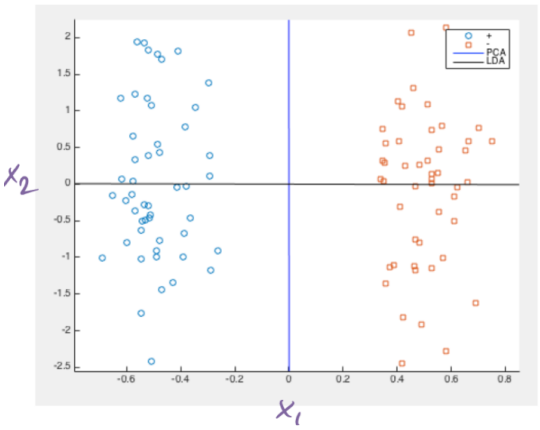


Figure 45: LDA vs. PCA.

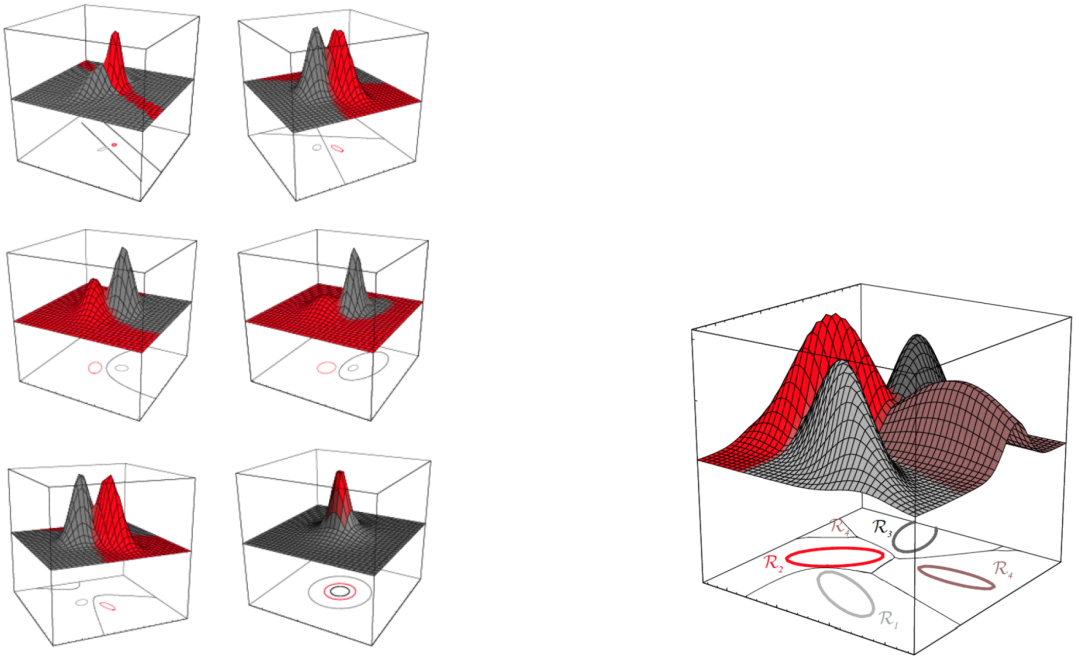


Figure 46: Quadratic discriminant analysis illustration

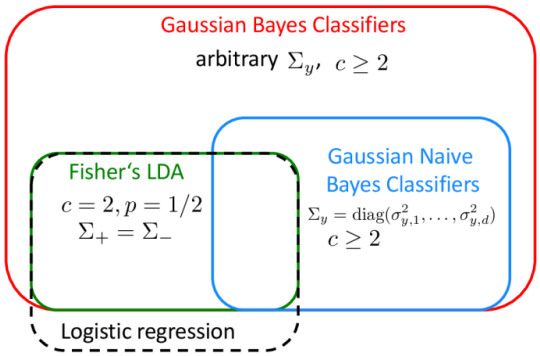


Figure 47: Big picture of GB classifiers

  - Generative model, i.e., models $P(\mathbf{X}, Y)$
  - Can be used to detect outliers: $P(\mathbf{X}) < t$
  - Assumes normality of $\mathbf{X}$.
  - Not very robust against violation of this assumption

- Logistic regression
  - Discriminative model, i.e. models $P(\mathbf{X}|Y)$
  - Cannot detect outliers
  - Makes no assumptions on $\mathbf{X}$
  - More robust

Comparing GNB vs. more general GBCs:

- GNB models:
  - Conditional independence assumption may lead to overconfidence
  - Predictions might still be useful
  - Number of parameters = $\mathcal{O}(cd)$
  - Complexity (memory + inference) linear in d.
- General Gaussian Bayes models
  - Captures correlations among features
  - avoids overconfidence
  - number of Parameters = $\mathcal{O}(cd^2)$
  - Complexity quadratic in d.

15.5 Working with categorical features

If we have discrete features (gender, nationality, ...) it might not make sense to model X_i as Gaussian. Generative models allow to easily swap different distributions, e.g. model $P(X_i|Y)$ as:

- Bernoulli
- Categorical
- Multinomial
- ...

For instance: take a categorical naive Bayes classifier. Model class label as generated from categorical variable, i.e. $P(Y = y) - p_y$ where $y \in \mathcal{Y} = \{1, \dots, c\}$. Model features are generated by conditionally independent categorical r.v.

$$P(\mathbf{X}_i = \mathbf{x}|Y = y) = \theta^{(i)}_{\mathbf{x}|y}$$
$$\forall i, \mathbf{x}, y : \theta^{(i)}_{\mathbf{x}|y} \geq 0$$
$$\forall i, y : \sum_{i=1}^{c} \theta^{(i)}_{\mathbf{x}|y} = 1$$

Given dataset $\mathcal{D} = \{(\mathbf{x}_1, y_1), \dots (\mathbf{x}_n, y_n)\}$. We want and MLE for the class label distribution $\hat{P}(Y = y) = \hat{p}_y$ given by

$$\hat{p}_y = \frac{\text{Count}(Y = y)}{n}$$

and the MLE for the distribution of feature i: $\hat{P}(X_i = c|y) = \theta^{(i)}_{c|y}$ is given by:

$$\theta^{(i)}_{c|y} = \frac{\text{Count}(X_i = c, Y = y)}{\text{Count}(Y = y)} \tag{25}$$

The learning is done as follows:

- MLE for class prior:
$$\hat{P}(Y = y) = \hat{p}_y = \frac{\text{Count}(Y = y)}{n}$$
- MLE for distribution of feature i is given by equation 25.
- The prediction given a new point $\mathbf{x}$ is given by:

$$\begin{aligned} y = \arg\max_{y'} \hat{P}(y'|\mathbf{x}) &= \arg\max_{y'} \hat{P}(y') \prod_{i=1}^{d} \overbrace{\hat{P}(x_i|y')}^{\text{Categorical}} \\ &= \arg\max_{y'} \log \hat{p}_y + \sum_{i=1}^{d} \log \theta^{(i)}_{x_i|y'} \end{aligned}$$

15.6 Beyond categorical Naive Bayes

We could in principle lift the NB assumption by modelling a joint conditional distribution of the features: $P(\mathbf{X}_1, \ldots, \mathbf{X}_d|Y)$. The issue here is that the probability of each assignment needs to be specified. It therefore requires exponentially many parameters in d, which is computational intractable and overfits. A good remedy is to use graphical models and Bayesian networks.

The Naive Bayes classifier does not require each feature to follow the same type of conditional distribution. For example, this model features that are both Gaussians while other features are categorical. The training (MLE) and prediction remain the same. Given:

$$\begin{aligned} X_{1:10} &: P(X_i|Y) = \text{Categorical}(X_i|Y, \theta) \\ X_{1:10} &: P(X_i|Y) = \mathcal{N}(X_i|\mu_{i,y}, \sigma^2_{i,y}) \end{aligned}$$

We then have:

$$P(X_{1:20}|y) = \prod_{i=1}^{10} \text{Categorical}(X_i|Y, \theta) \prod_{i=11}^{20} \mathcal{N}(X_i|\mu_{i,y}, \sigma^2_{i,y})$$

MLE is generally prone to overfitting. This is avoidable by restricting the model class (e.g. assumptions on covariance structure in GNB) which aim to reduce parameters. Using priors can also help.

As prior for our class probabilities in the case that $c = 2$, we have $P(Y = 1) = \theta$. The MLE is $\hat{\theta} = \frac{\text{Count}(Y=1)}{n}$. In the case that $n = 1$, we end up with $\hat{\theta} = \frac{1}{1} = 1$. We therefore may want to put a prior distribution $P(\theta)$ and compute the posterior distribution using the available data.

$$P(\theta|y_1, \ldots, y_n) = \frac{1}{z} P(\theta) \cdot P(y_{1:n}|\theta)$$

where $x = \int P(\theta)p(y_{1:n}|\theta)d\theta$.

A natural choice for providing a prior of a r.v. is the beta distribution given by:

$$\text{B}(\theta; \alpha_+, \alpha_-) = \frac{1}{\text{B}(\alpha_+, \alpha_-)} \theta^{\alpha_+ - 1}(1 - \theta)^{\alpha_- - 1}$$

Shapes of the p.d.f. of this distribution is shown in Figure 48

Assume observation data D with n_+ times $Y = +1$ and n_- times $Y = -1$, then we have:

$$p(\theta|D) = \frac{1}{z} p(\theta)p(D|\theta) = \frac{1}{z'} \theta^{\alpha_+ - 1}(1 - \theta)^{\alpha_- - 1}\theta^{n_+}(1 - \theta)^{n_-}$$

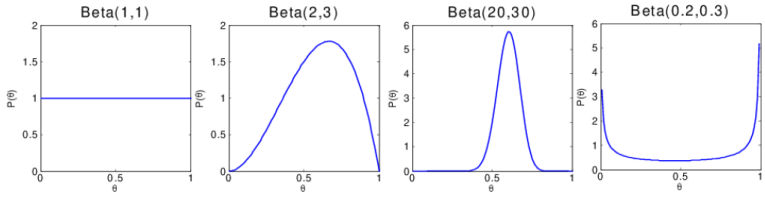


Figure 48: Beta distribution under different values of α_+ and α_-

Prior/posterior	**Likelihood function**
Beta	Bernouilli/Binomial
Dirichtlet	Categorial/Multinomial
Gaussian (fixed variance)	Gaussian
Gaussian-inverse Wishart	Gaussian
Gaussian process	Gaussian

Table 3: Conjugate prior and likelihood function

15.7 Conjugate prior and distributions

A pair of prior distributions and likelihood functions is called conjugate if the posterior distribution remains in the same family as the prior. For instance, if we take a prior $\mathrm{B}(\theta; \alpha_+, \alpha_-)$ and oberve n_+ positives and n_- negatives, the posterior is given by $\mathrm{B}(\theta; \alpha_+ + x_+, \alpha_- + n_-)$. Here α_+ and α_- act as pseudo-counts. The MAP estimate is given by:

$$\hat{\theta} = \arg\max_{\theta} p(\theta|y_1, \ldots, y_n; \alpha_+, \alpha_-) = \frac{\alpha_+ + n_+ - 1}{\alpha_+ + n_+ + \alpha_- + n_- - 2}$$

It is possible to use conjugates as regularizers and add no substantial computational cost. The hyperparameters of these conjugate priors can be chosen using cross validation.

15.8 Generative vs. Discriminative models

- Discriminative models:
  - Model $P(y|\mathbf{x})$. Does not attempt to model $P(\mathbf{x})$
  - Cannot detect outliers (property of $P(\mathbf{x})$
  - Are typically more robust, since accurately modelling $\mathbf{x}$ may be difficult.
- Generative models
  - Model joint distribution $P(\mathbf{x}, y)$, which is more ambitious
  - Can be more powerful if model assumptions are met (can detect outliers)
  - Are typically less robust against outliers.

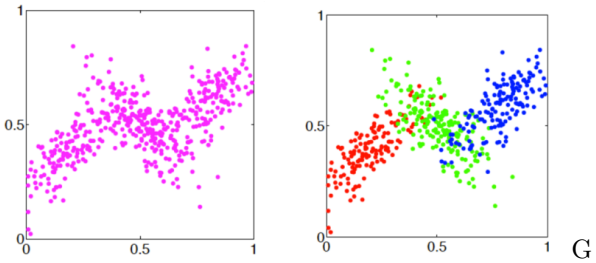
15.9 Dealing with missing data

Missing data can either contain missing labels or missing features. An example of trying to infer class labels from feature distributions is shown in Figure 49.

Figure 49 is provided by a convex combination of Gaussian distributions:

$$P(\mathbf{x}|\theta) = P(\mathbf{x}|\mu, \Sigma, \mathbf{w}) = \sum_{i=1}^{c} w_i \mathcal{N}(\mathbf{x}\mu_i, \Sigma_i)$$

where $w_i \geq 0$ and $\sum_i w_i = 1$.



G

Figure 49: Gaussian Mixture allows to recover class labels from feature distribution.

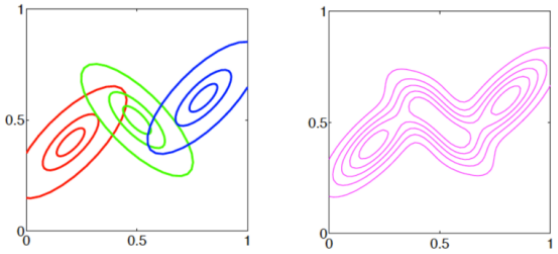


Figure 50: Gaussian mixuture, contour and scatterplot

Fitting a mixture model means finding:

$$(\mu^*, \Sigma^*, w^*) = \arg\min - \sum_i \log \sum_{j=1}^{k} w_j \mathcal{N}(\mathbf{x}_i | \mu_j \Sigma_j)$$

The MLE for Gaussian mixture is found by minimizing:

$$L(w_{1:k}, \mu_{1:k}, \Sigma_{1:k}) = -\sum_{i=1}^{n} \log \sum_{j=1}^{k} w_j \mathcal{N}(\mathbf{x}_i | \mu_j, \Sigma_j)$$

which is non-convex objective.
It is still possible to apply stochastic gradient descent. However, several challenges remain:

- Covariance matrices must remain symmetric positive definite
- These constraints might be difficult to maintain.

Both GBC and GBB work with the joint distribution. The difference between GBCs in GMMs is that GMMs, the variable z is unobserved. Therefore, fitting a GMM is equivalent to training a GBC without labels. This is equivalent to clustering, because we want to model a latent variable. If we could get the labels, we could compute the MLE in closed form.
Enter: the Hard-EM algorithm. It consists of the following steps:

- Initialize the parameters $\theta^{(0)}$
- For $t = 1, 2, \ldots$:
  - **E-step**: predict the most likely class for each data point:

$$\begin{aligned} z_i^{(t)} &= \arg\max_z P(z | \mathbf{x}_i, \theta^{(t-1)}) \\ &= \arg\max_z \underbrace{P(z|\theta^{t-1})}_{w_z^{(t-1)}} \underbrace{P(\mathbf{x}_i | z, \theta^{(t-1)})}_{\mathcal{N}(\mathbf{x}_i | \mu_z^{(t-1)}, \Sigma_z^{t-1})} \end{aligned}$$

  - The result is complete data.

– **M-step**: compute the MLE as for the GBC, using:

$$\theta^{(t)} = \arg\max_{\theta} P(D^{(t)}|\theta)$$

The problems with the EM algorithm is that points assigned a fixed label, even though the model is uncertain. Intuitively, this tries to extract too much information from a single point. In practice, this may work poorly if clusters are overlapping.

Ideally we want to compute the posterior probabilities of belonging to a cluster. Suppose we're given a model $P(z|\theta)$, $P(\mathbf{x}|z,\theta)$. Then for each data point, we compute a posterior distribution over cluster membership. This means inferring distributions over latent hidden variables z as follows:

$$\gamma_j * (\mathbf{x}) = P(Z = j|\mathbf{x}, \Sigma, \mu, \mathbf{w}) = \frac{w_j P(\mathbf{x}|\Sigma_j, \mu_j}{\sum_\ell w_{ell} P(\mathbf{x}|\Sigma_{ell}, \mu_\ell))}$$

We can look at this optimality condition as a MLE problem.

Theorem 5. *At MLE*

$$(\mu^*, \Sigma^*, w^*) = \arg\min - \sum_i \log \sum_{j=1}^{k} w_j \mathcal{N}(\mathbf{x}_i|\mu_j, \Sigma_j)$$

the following must hold:

$$\mu_j^* = \frac{\sum_{i=1^n} \gamma_j(\mathbf{x}_i)\mathbf{x}_i}{\sum_{i=1^n} \gamma_j(\mathbf{x}_i)}$$
$$\Sigma_j^* = \frac{\sum_{i=1}^n \gamma_j(\mathbf{x})(\mathbf{x}_i - \mu_j^*)(\mathbf{x}_i - \mu_j^*)^T}{\sum_{i=1}^n \gamma_j(\mathbf{x}_i)}$$
$$w_j^* = \frac{1}{n}\sum_{i=1}^{n} \gamma_j(\mathbf{x}_i)$$

This allows us to define the soft EM algorithm.

- While not converged
  - **E-step**: calculate cluster membership weights (aka. expected sufficient statistic or responsibilities) for each point. We calculate $\gamma_j^{(t)}(\mathbf{x}_i)$ for each i and j givenestimates of μ^{t-1},Σ^{t-1},$\mathbf{w}^{(t-1)}$ from the previous iteration.
  - **M-step**: fit cluster to weighted data points which corresponds to the maximum likelihood solution:

$$w_j^{(t)} \leftarrow \frac{1}{n}\sum_{i=1}^{n} \gamma_j^{(t)}(\mathbf{x}_i)$$
$$\mu_j^{(t)} \leftarrow \frac{\sum_{i=1}^n \gamma_j^{(t)}(\mathbf{x}_i)\mathbf{x}_i}{\sum_{i=1}^n \gamma_j^{(t)}(\mathbf{x}_i)}$$
$$\Sigma_j^{(t)} \leftarrow \frac{\sum_{i=1}^n \gamma_j^{(t)}(\mathbf{x}_i)(\mathbf{x}_i - \mu_j^{(t)})(\mathbf{x}_i - \mu_j^{(t)})^T}{\sum_{i=1}^n \gamma_j^{(t)}(\mathbf{x}_i)}$$

An illustration of the EM algorithm in action is shown in figure 51.
Note: we can also have constrained GMMs by tying together parameters in MLE:

- Spherical: $\Sigma_j = \sigma_j^2 \cdot I_j$
- Diagonal: $\Sigma_j = \begin{pmatrix} \sigma_{j,1}^2 & \dots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \dots & \sigma_{j,1}^2 \end{pmatrix}$ (a.k.a. GNB).

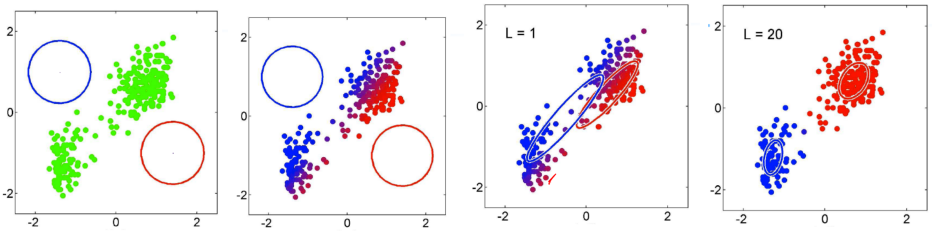


Figure 51: EM algorithm illustration

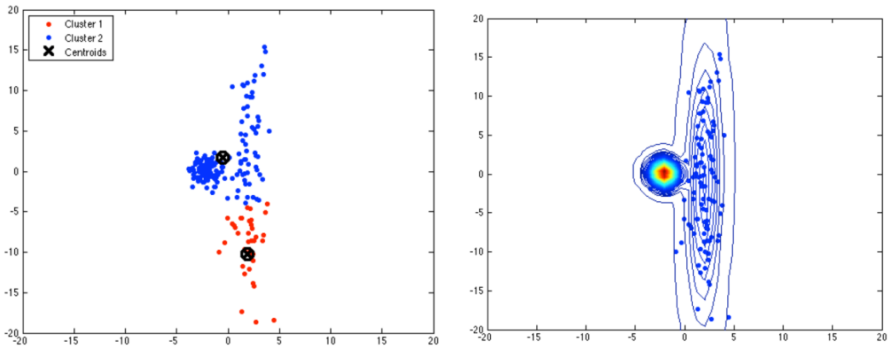


Figure 52: k-means vs. GMM

- Tied $\Sigma_1 = \ldots = \Sigma_c$
- Full

The EM algorithm estimating using soft assignments to clusters is called soft EM. The intuitive first attempt we discussed first uses hard assignments, and is called hard EM. In general, soft EM will typically result in higher likelihood values. Reason it can deal better with overlapping cluster. The term EM alone typically refers to soft EM.

Theorem 6. *Hard EM with uniform weights and spherical covariances is equivalent to k-means.*

For proof, see slide 32 of lecture 23.
Initialization of EM:

- For weights, we can typically use a uniform distribution
- For means, we randomly initialize them and k-means++.
- For variances, we initialize them as spherical, e.g. according to empirical variance in the data.

Selecting k: in GMMs, we face a similar challenge of selecting the number of clusters. We can generally use the same techniques. However, in contrast to k-means, for GMMs typically cross-validation works fairly well. It aims to maximize log-likelihood on the validation set. If the data is truly generated from a GMM, using cross-validation can be used to select k. See Figure 53.

15.10 Degeneracy of GMMs

Suppose we are given a single data point. What is the optimal log-likelihood that can be achieved? Well, because

$$-\log P(x|\mu,\sigma) = \frac{1}{2}\log(2\pi\sigma^2) + \frac{1}{2\sigma^2}(x-\mu)^2$$

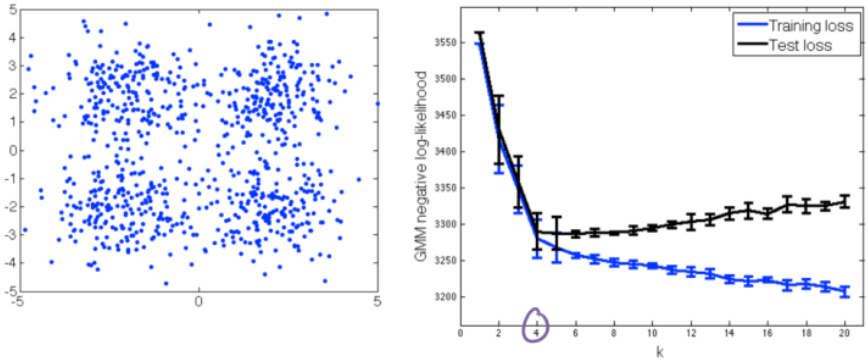


Figure 53: Selecting k in GMM.

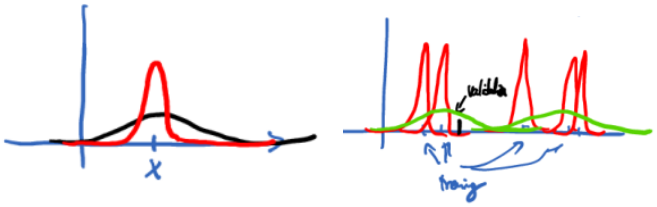


Figure 54: Degeneracy of GMM. Red, what happens without regularization, green is what we want: wider variance for better generalization.

Small variances will be encouraged and the model won't generalize at all. See Figure 54. The loss essentially converges to 0 as $\mu = x, \sigma \to 0$. Thus, an optimal GMM chosses $k = n$, and puts one Gaussian around each data point with variance tending to 0. This is overfitting.

Overfitting can be avoided by adding a small term to the diagonal of the MLE as follows:

$$\Sigma_j \leftarrow \frac{\sum_{i=1}^n \gamma_j^{(t)}(\mathbf{x})(\mathbf{x} - \mu_i^{(j)})(\mathbf{x} - \mu_i^{(j)})^T}{\sum_{i=1}^n \gamma_j^{(t)}(\mathbf{x})} + \mu^2 \mathbf{I}$$

This can be motivated from a Bayesian standpoint: this is equivalent to placing a conjugate Wishart prior on the covariance matrix, and computing the MAP instead of the MLE to regularize. μ can be chosen by cross-validation.

15.11 Gaussian Mixtures Bayes Classifiers

GMMs are useful because:

- Can encode information about shape of clusters.
- Can be part of more complex statistical models.
- Probabilistic models can output the likelihood $P(\mathbf{x})$ of a point $\mathbf{x}$, which is useful for anomaly or outlier detection, and therefore can be naturally used for semi-supervised learning.

A Gaussian Mixture Bayes classifier is defined as follows:

- Given $\mathcal{D} = (\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_n, y_n)$, where the label $y \in \{1, \ldots, m\}$, we estimate the class prior $P(y)$ and estimate the conditional distribution for each class:

$$P(\mathbf{x}|y) = \sum_j w_j^{(y)} \mathcal{N}(\mathbf{x}; \mu_j^{(y)}, \Sigma_j^{(y)})$$

  as a Gaussian mixture model.

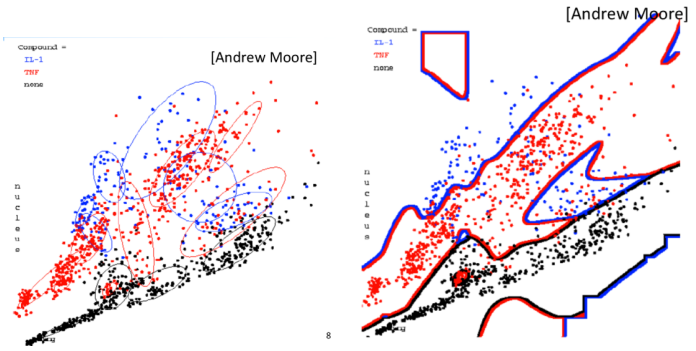


Figure 55: Clustering for non linear features vs. Gaussian mixture Bayes classifier.

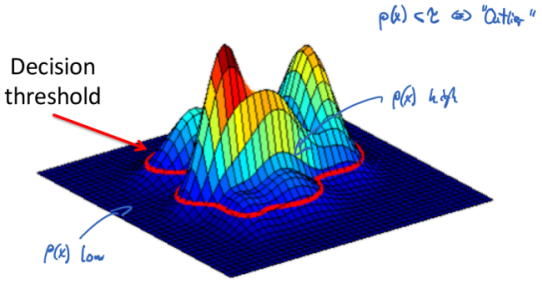


Figure 56: GMM for anomaly detection.

- To use this model for classification, we perform the following:

$$P(y|\mathbf{x}) = \frac{1}{Z}p(y)\sum_{j=1}^{k_y} w_j^{(y)}\mathcal{N}(\mathbf{x};\mu_j^{(y)},\Sigma_j^{(y)})$$

GMMs can also be used for density estimation. We may be interested in fitting a Gaussian mixture model not for clustering but for density estimation. This combines the advantage of accurate predictions and robustness from discriminative models with the ability to detect outliers.

We can perform anomaly detection with mixture models by comparing the estimated density of the a data point against a threshold, see Figure 56.

Picking the threshold is tricky if we have no examples. If we do have some examples, varying the threshold trades false-positives and false-negatives. One can use PR-curves and ROC curves as evaluation criterion; e.g. maximize the F1 score. the threshold can also be optimized via cross-validation.

GMMs allow one to combine unlabeled and labeled data in a semi supervised learning context. In SSL, for instances $\gamma_j(\mathbf{x}_i) = [j = y_i]$

The EM algorithm for semi-supervised learning with GMM goes as follows:

While not converged:

- **E-step**: for unlabeled points:

$$\gamma_j^{(t)}(\mathbf{x}_i) = P(Z = j|\mathbf{x}_i, \mu^{(t-1)}, \Sigma^{(t-1)}, \mathbf{w}^{(t-1)})$$

  and for labeled points

$$\gamma_j^{(t)}(\mathbf{x}_i) = [j = y_i]$$

- **M-step**: Fit clusters to weighted data points, which corresponds to the closed form of the maximum likelihood solution described above, i.e. maximize

$$\theta^{(t)} = \arg\max_{\theta} Q(\theta;\theta^{(t-1)})$$

15.12 Theory behind the EM algorithm

We can show that the EM algorithm is equivalent to the following procedure:

- **E-step**: calculate the expected complete data log likelihood, which is equivalent to a function of θ:

$$\begin{aligned} Q(\theta; \theta^{(t-1)}) &= \mathrm{E}_{z_{1:n}}[\log P(\mathbf{x}_{1:n}, \mathbf{z}_{1:n}|\theta)|\mathbf{x}_{1:n}, \theta^{(t-1)}] \\ &= \sum_{z_{1:n}} P(z_{1:n}|\mathbf{x}_{1:n}, \theta^{(t-1)}) \underbrace{\log P(\mathbf{x}_{1:n}, z_{1:n}|\theta^{(t-1)})}_{\text{Complete data log-likelihood}} \end{aligned}$$

- **M-step**: maximize $\theta^{(t)} = \arg\max_\theta Q(\theta; \theta^{(t-1)})$

Note that the EM objective function can be simplified to:

$$Q(\theta; \theta^{(t-1)}) = \sum_{i=1}^{n} \sum_{j=1}^{k} \underbrace{P(z_i = j|\mathbf{x}_i, \theta^{(t-1)})}_{\gamma_j(\mathbf{x}_i))} \log \underbrace{P(\mathbf{x}_i, z_i = j|\theta)}_{w_j \mathcal{N}(\mathbf{x}_i, \mu_j, \Sigma_j)}$$

Thus, the E-step is equivalent to computing $\gamma_z(\mathbf{x}_i)$, which are called expected sufficient statistics. See slide 25 of lecture 24 for the complete derivation.

In the M-step, we need to compute:

$$\theta^{(t)} = \arg\max_\theta Q(\theta; \theta^{(t-1)})$$

which is very similar to the MLE of the GBC:

$$\theta^* = \arg\max_\theta \sum_{i-1}^{n} \log P(\mathbf{x}, z_i|\theta)$$

Thus, each iteration of the M-step is equivalent to training a GBC with weighted data and has a closed form solution.

Theorem 7. *The EM algorithm monotonically increases the likelihood, i.e. the following holds:*

$$\log P(\mathbf{x}_{i:n}|\theta^{(t)}) \geq \log P(\mathbf{x}_{i:n}|\theta^{(t-1)})$$

For GMMs, EM is guaranteed to converge to a local maximum. The quality of the solution highly depends on the initialization, as in k-means. A common strategy is to rerun the algorithm multiple times and use the solution with the largest likelihood. The proof of Theorem 7 is shown in slides 29-31 of slide deck 24 and slides 1-9 of slide deck 25.

The EM algorithm is much more widely applicable. It can be used whenever the E and M steps are tractable, that means we must just be able to compute and maximize the complete data likelihood. Furthermore, this algorithm can be used, for instance, for some missing features, and likelihoods beyond Gaussian (e.g. categorical).

16 Generative Modelling with Neural Networks

So far, we have considered very simple probabilistic models (hand selected priors, and likelihood functions). These fail to capture complex, high-dimensional data types like images and audio.

16.1 Implicit generative models

Given a sample of unlabeled points $\{\mathbf{x}_1, \ldots, \mathbf{x}_n\}$, the goal is to learn a model $\mathbf{X} = G(\mathbf{Z}; \mathbf{w})$ where $\mathbf{Z}$ is a simple distribution (e.g. low dimensional Gaussian). and G some flexible nonlinear function approximated by a neural net. A schemaic mapping of this function is shown in Figure 57:

The key challenge here is to compute the likelihood of the data. Thus, we need an alternative/surrogate objective functions for training. Variants exist, such as VAE and GANs.

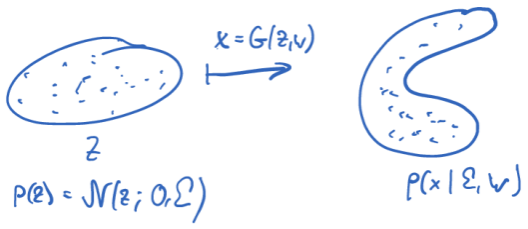


Figure 57: Implicit generative models

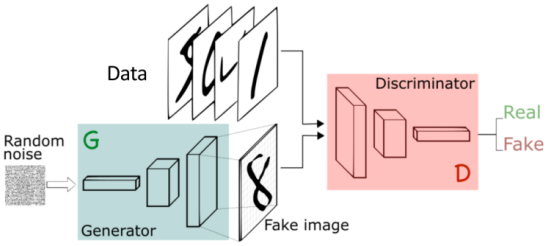


Figure 58: GAN setup

16.2 Generative Adversarial Networks

By Goodfellow et al, the goal is to optimize parameters $\mathbf{w}$ to make samples from model hard to distinguish from a data sample. Therefore, one can use a discriminative model to train a generative model. The generator tries to produce realistic examples which the discriminator tries to detect fake examples.

Formally:

$$D : \mathrm{R}^d \to [0,1] \text{ wants } : D(\mathbf{x}) = \begin{cases} \approx 1 \text{ if } x\text{is real} \\ \approx 0 \text{ if } x\text{is fake} \end{cases} \quad G : \mathrm{R}^m \to \mathrm{R}^d \text{ wants } : D(G(\mathbf{z})) \approx 1 \text{ for samples} z$$

The overall training objective is:

$$\min_G \max_D \underbrace{\mathrm{E}_{\mathbf{X}\sim\mathcal{D}} \log D(\mathbf{x}) + \mathrm{E}_{\mathbf{X}\sim\text{Noise}} \log(1 - D(\mathbf{x}))}_{M(G,D)}$$

Training a GAN requires finding a saddle point rather than a local minimimum, due to:

$$\min_{\mathbf{w}_G} \max_{\mathbf{w}_D} M(\mathbf{w}_G, \mathbf{w}_D)$$

as illustrated in Figure 59. Another illustration is shown in Figure to show how the training process approximates the distribution of the data.

GANs have a convergence guarantee. If G and D have enough capacity, then the data generating distribution is indeed the saddle point of:

$$\min_{\mathbf{w}_G} \max_{\mathbf{w}_D} M(\mathbf{w}_G, \mathbf{w}_D)$$

Key idea: the best possible discriminator $\ell^* = \max_{\mathbf{w_D}} M(\mathbf{w}_G, \mathbf{w}_D)$ is up to constants the Jensen-Shannon divergence $JS(p_{\text{Data}}||q_{\text{Generator}})$. Where:

$$JS(p||q) = \frac{1}{2}\mathrm{KL}(p||\frac{p+q}{2}) + \frac{1}{2}\mathrm{KL}(q||\frac{p+q}{2}); JS(p||q) = 0 \Leftrightarrow p = q.$$

In practice, we train a GAN on a finite sample. While we run into the danger of running into the memorization trap, diminishing the power of the discriminator can help alleviate that.

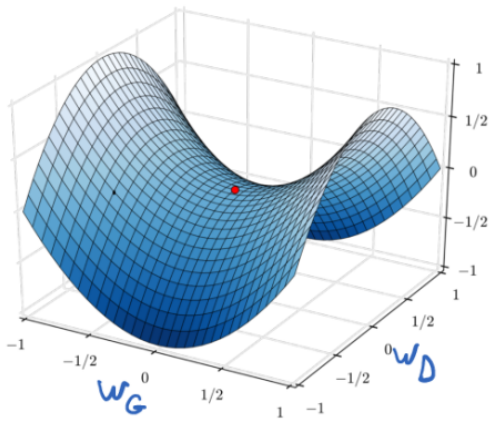


Figure 59: Gan optimization landscape and objective (red dot).

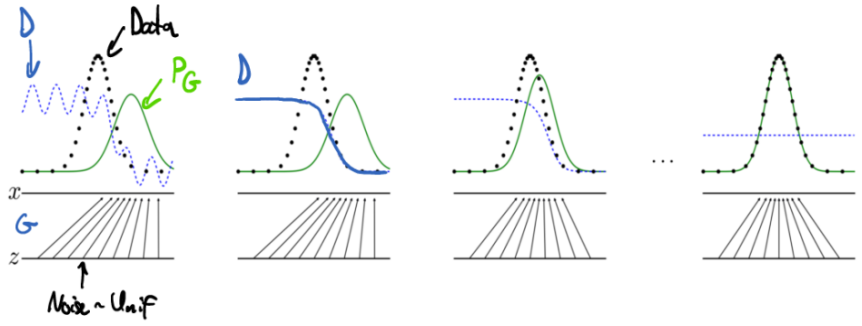


Figure 60: GAN training process, the number of iterations increases from left to right

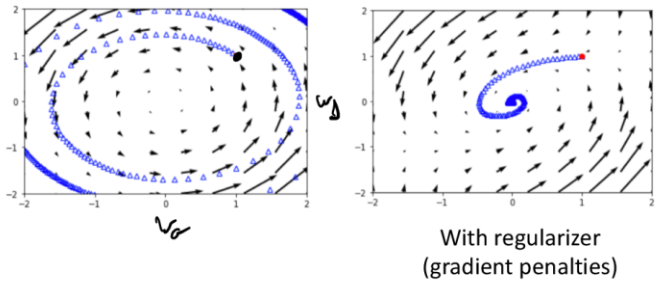


Figure 61: Oscillations: a challenge when training GANs.

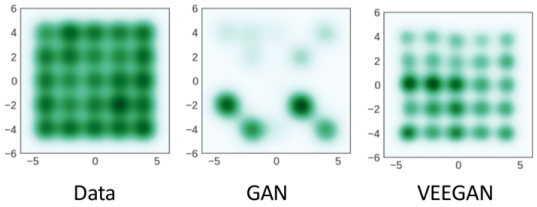


Figure 62: VEEGAN as a solution to mode collapse.

Training approaches for GAN is to simultaneously apply stochastic gradient descent to the empirical GAN objective.

$$\mathbf{w}_G^{(t+1)} = \mathbf{w}_G^{(t)} - \eta \nabla_{\mathbf{w}_G} M(\mathbf{w}_G, \mathbf{w}_D^{(t)})$$
$$\mathbf{w}_G^{(t+1)} = \mathbf{w}_G^{(t)} + \eta \nabla_{\mathbf{w}_D} M(\mathbf{w}_G^{(t)}, \mathbf{w}_D)$$

Gradients are approximated by samples of data points.

Note that when training GANs, there is a risk of oscillations and divergence. See Figure **??**.

Another issue that arises is mode collapse, when GANs only model parts of the mode of a distribution. One way to solve this is e.g. in VEEGAN, see Figure 62.

When evaluating GANs, it is not possible to compute the likelihood on a holdout set. Generally, it's a difficult problem with no well-accepted and domain independent solution. There are various heuristics though such as the inception score & FID for images. This remains an open area of research.