Ordlista:

Statistically significant: "In statistical hypothesis testing,[1][2] a result has statistical significance when it is very unlikely to have occurred given the null hypothesis.[3] More precisely, the significance level defined for a study, α, is the probability of the study rejecting the null hypothesis, given that it were true;[4] and the p-value of a result, p, is the probability of obtaining a result at least as extreme, given that the null hypothesis were true. The result is statistically significant, by the standards of the study, when $p < \alpha$."

• Semi-supervised: targets are known only for some observations.
• Active learning. Strategies for deciding which observations to label
• Reinforcement learning. Find suitable actions to maximize the reward. True targets are discovered by trial and error.

- **In *k-NN classification***, the output is a class membership. An object is classified by a majority vote of its neighbors, with the object being assigned to the class most common among its *k* nearest neighbors (*k* is a positive integer, typically small). If *k* = 1, then the object is simply assigned to the class of that single nearest neighbor.
- **In *k-NN regression*,** the output is the property value for the object. This value is the average of the values of its *k* nearest neighbors.

Discriminative models, (Logistic regression, linear regression, SVM, neural networks) also called conditional models, are a class of models used in machine learning for modeling the dependence of unobserved (target) variables y on observed variables x. Within a probabilistic framework, this is done by modeling the conditional probability distribution P(y|x), which can be used for predicting y from x. Discriminative models, as opposed to generative models, do not allow one to generate samples from the joint distribution of observed and target variables. However, for tasks such as classificationand regression that do not require the joint distribution, discriminative models can yield superior performance (in part because they have fewer variables to compute).On the other hand, generative models are typically more flexible than discriminative models in expressing dependencies in complex learning tasks. In addition, most discriminative models are inherently supervisedand cannot easily support unsupervised learning. Application-specific details ultimately dictate the suitability of selecting a discriminative versus generative model.

ROC Curves - ROC=Receiver operating characteristics

• Use various thresholds, measure TPR and FPR

• Same FPR, higher TPR better classifier

• Best classifier = greatest Area Under Curve (AUC)

Degrees of Freedom: in R: df = trace(S) (see logistic regression code) for linear regression df = # of features in model

Ridge Regression: Ridge regression is particularly useful if the explanatory variables are strongly correlated to each other: Correlated variables often correspond large w -> shrunk

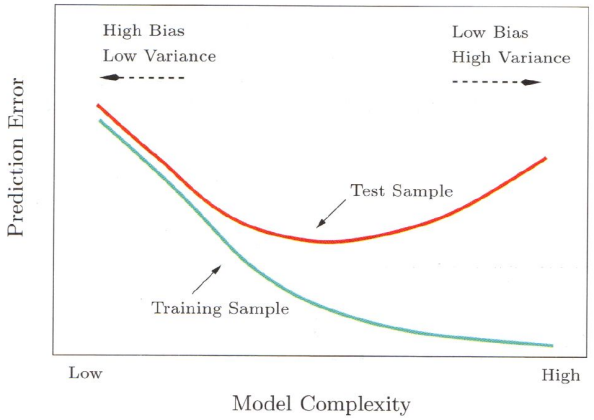
LASSO: Why Lasso leads to sparse solutions? Feasible area for Ridge is a circle (2D) Feasible area for LASSO is a polygon (2D)
Lasso is widely used when $\boldsymbol{p} \gg \boldsymbol{n}$ (more features than observations)

– Linear regression breaks down when $p > n$

– Application: DNA sequence analysis, Text Prediction

Bias: If Bias(y^x0) = 0, the estimator is unbiased

ML estimators are asymptotically unbiased if the model is enough complex



Cross-validation:

K-fold cross-validation (rough scheme, show picture):

1. Permute the observations randomly

2. Divide data-set in K roughly equally-sized subsets

3. Remove subset #i and fit the model using remaining data.

4. Predict the function values for subset #i using the fitted model.

5. Repeat steps 3-4 for different i

6. CV = squared difference between observed values and predicted values **(another function is possible)**

Note: if K=N then method is leave-one-out crossvalidation.

fit1=lm(V9~V3+V4+V5+V6+V7, data=data)

fit2=lm(V9~V3+V4+V5+V6, data=data)

f1=cvFit(fit1, y=data$V9, data=data,K=10,foldType="consecutive")

f2=cvFit(fit2, y=data$V9, data=data,K=10,foldType="consecutive")

res=cvSelect(f1,f2)

plot(res)

Holdout vs Cross:

• Holdout is easy to do (a few model fits to each data)

• Cross validation is computationally demanding (many

model fits)

• Holdout is applicable for large data

– Otherwise, model selection performs poorly

• Cross validation is more suitable for smaller data

AIC:

The Akaike information criterion (AIC) is an estimator of the relative quality of statistical models for a given set of data. Given a collection of models for the data, AIC estimates the quality of each model, relative to each of the other models. Thus, AIC provides a means for model selection.

AIC is founded on information theory: it offers an estimate of the relative information lost when a given model is used to represent the process that generated the data. (In doing so, it deals with the trade-off between the goodness of fit of the model and the simplicity of the model.)

AIC does not provide a test of a model in the sense of testing a null hypothesis. It tells nothing about the absolute quality of a model, only the quality relative to other models. Thus, if all the candidate models fit poorly, AIC will not give any warning of that.

Decision trees:

Decision trees: comments

- Similar algorithms work for regression trees – replace $N \cdot Q(R)$ by $SSE(R)$
- Easy to interpret
- Easy to handle all types of predictors in one model
- **Automatic variable selection**
- Relatively robust to outliers
- Handle large datasets

- Trees have high variance: a small change in response→ totally different tree
- Greedy algorithms → fit may be not so good
- Lack of smoothness

Optimal trees

- **Postpruning**

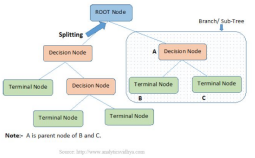
Weakest link pruning:

1. Merge two leaves that have smallest *N(parent)*Q(parent)-N(leave1)Q(leave1)-N(leave2)Q(leave2)*
2. For the current tree T, compute

$$I(T) = \sum_{R_i \in leaves} N(R_i)Q(R_i) + \alpha|T|$$

$|T|$ =#leaves

3. Repeat 1-2 until the tree with one leave is obtained
4. Select the tree with smallest I(T)



How to find the optimal α? Cross validation!

Decision trees are also not sensitive to outliers since the splitting happens based on proportion of samples within the split ranges and not on absolute values.

Logistic Regression:

Logistic regression

Fitting logistic regression

- In binary case,

$$\log P(D|w) = \sum_{i=1}^{N} y_i \log(\, sigm(w^T x_i)) \; + (1 - y_i) \log(1 - sigm(w^T x_i))$$

  - Can not be maximized analytically, but unique maximizer exists
- To maximize loglikelihood, optimization used
  - Newton's method traditionally used (Iterative Reweighted Least Squares)
  - Steepest descent,Quasi-newton methods…

Estimation:

For new x, estimate $p(y) = [p_1, \dots p_C]$ and classify as $\arg \max_i p_i$

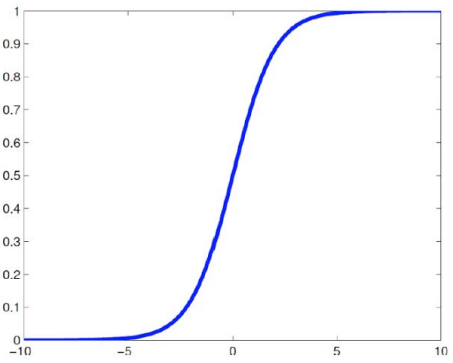
Decision boundaries of logistic regression are linear

Logistic regression

- Discriminative model
- Model for binary output
  - $C = \{C_1 = 1, C_2 = 0\}$
    $p(Y = C_1|X) = sigm(\boldsymbol{w}^T \boldsymbol{x})$

$$sigm(a) = \frac{1}{1 + e^{-a}}$$

What is $P(Y = C_2|X)$?



- Alternatively

$$Y \sim Bernoulli(sigm(a)), a = \boldsymbol{w}^T \boldsymbol{x}$$

$$sigm(a) = \frac{1}{1 + e^{-a}}$$

- In R, use glm() with family="binomial"
  - Predicted probabilities: predict(fit,newdata, type="response")

We know how to model

– Normally distributed targets -> linear regression

– Bernoulli and Multinomial targets -> logistic regression

```
> set.seed(666)
> x1 = rnorm(1000)           # some continuous variables
> x2 = rnorm(1000)
> z = 1 + 2*x1 + 3*x2        # linear combination with a bias
> pr = 1/(1+exp(-z))         # pass through an inv-logit function
> y = rbinom(1000,1,pr)      # bernoulli response variable
>
> #now feed it to glm:
> df = data.frame(y=y,x1=x1,x2=x2)
> glm( y~x1+x2,data=df,family="binomial")
```

Generalized Linear Model:

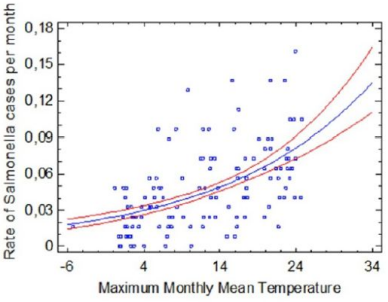
Try to fit usual linear regression, study histogram of residuals

Generalized linear models

- Canonical links are normally used
  - MLE computations simplify
  - MLE $\widehat{w} = F(X^T Y)$ → computations do not depend on all data but rather a summary (sufficient statistics)→ computations speed up

Example: Poisson regression

$$f^{-1}(\mu) = e^{\mu}, Y \sim Poisson\left(e^{\boldsymbol{w}^T \boldsymbol{x}}\right)$$



Common distributions with typical uses and canonical link functions

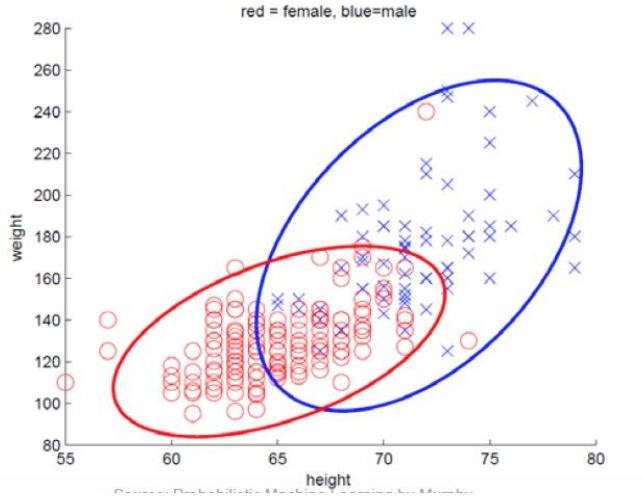
Distribution	Support of distribution	Typical uses	Link name	Link function, $\mathbf{X}\boldsymbol{\beta} = g(\mu)$	Mean function
Normal	real: $(-\infty, +\infty)$	Linear-response data	Identity	$\mathbf{X}\boldsymbol{\beta} = \mu$	$\mu = \mathbf{X}\boldsymbol{\beta}$
Exponential	real: $(0, +\infty)$	Exponential-response data, scale parameters	Inverse	$\mathbf{X}\boldsymbol{\beta} = \mu^{-1}$	$\mu = (\mathbf{X}\boldsymbol{\beta})^{-1}$
Gamma					
Inverse Gaussian	real: $(0, +\infty)$		Inverse squared	$\mathbf{X}\boldsymbol{\beta} = \mu^{-2}$	$\mu = (\mathbf{X}\boldsymbol{\beta})^{-1/2}$
Poisson	integer: $0, 1, 2, \ldots$	count of occurrences in fixed amount of time/space	Log	$\mathbf{X}\boldsymbol{\beta} = \ln(\mu)$	$\mu = \exp(\mathbf{X}\boldsymbol{\beta})$
Bernoulli	integer: $\{0, 1\}$	outcome of single yes/no occurrence	Logit	$\mathbf{X}\boldsymbol{\beta} = \ln\left(\frac{\mu}{1-\mu}\right)$	$\mu = \frac{\exp(\mathbf{X}\boldsymbol{\beta})}{1+\exp(\mathbf{X}\boldsymbol{\beta})} = \frac{1}{1+\exp(-\mathbf{X}\boldsymbol{\beta})}$
Binomial	integer: $0, 1, \ldots, N$	count of # of "yes" occurrences out of N yes/no occurrences			
Categorical	integer: $[0, K)$	outcome of single K-way occurrence			
	K-vector of integer: $[0, 1]$, where exactly one element in the vector has the value 1				
Multinomial	*K*-vector of integer: $[0, N]$	count of occurrences of different types (1 .. *K*) out of *N* total *K*-way occurrences			

Quadratic discriminant analysis:

qda(formula, data, ..., subset, na.action)

Generative classifier

• Main assumptions: x is now random as well as y



SVM:

C: The cost parameter penalizes large residuals. So a larger cost will result in a more flexible model with fewer misclassifications. In effect the cost parameter allows you to adjust the bias/variance trade-off. The greater the cost parameter, the more variance in the model and the less bias.

The greater the cost, the fewer misclassifications are allowed.

Note how this is the opposite of regularization which penalizes large coefficients, resulting in higher bias and lower variance. Here we penalize the residuals resulting in higher variance and lower bias

Neural Networks:

A neural network can be thought of as a series of logistic regressions stacked on top of each other. This means we could say that a logistic regression is a neural-network (with sigmoid activations) with no hidden-layer.

How many iterations are needed to converge?

- This depends on how strong the learning rate we are applying. High learning rate means faster learning, but with higher chance of instability.
- It depends as well on the meta-parameters of the network (how many layers, how complex the non-linear functions are). The more it has variables the more it takes time to converge, but the higher precision it can reach.
- It depends on the optimisation method use, some weight updates rule are proven to be faster than others for instance AdaDelta.
- It depends on the random initialisation of the network. Maybe with some luck you will initialise the network with W=1.99 and you are only one step away from the optimal solution.
- It depends on the quality of the training set. If the input and output has no correlation between each other, the neural network will not do magic and can't learn a random correlation.

Linear Discriminant Analysis:

Linear discriminant analysis (LDA) is a generalization of Fisher's linear discriminant, a method used in statistics, pattern recognition and machine learning to find a linear combination of features that characterizes or separates two or more classes of objects or events. The resulting combination may be used as a linear classifier, or, more commonly, for dimensionality reduction before later classification.

library MASS

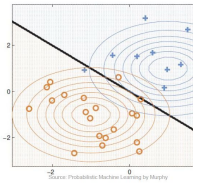
lda(formula, data, ..., subset, na.action)

• Prior – class probabilities

• Subset – indices, if training data should be used

Linear discriminant analysis (LDA)

- Assumtion $\Sigma_i = \Sigma, i = 1, \dots K$
- Then $p(y = c_i|x) = softmax(w_i^T x + w_{0i})$ →exactly the same form as the logistic regression
  - $w_{0i} = -\frac{1}{2}\boldsymbol{\mu}_i^T \Sigma^{-1} \boldsymbol{\mu}_i + \log \pi_i$
  - $w_i = \boldsymbol{\Sigma}^{-1} \boldsymbol{\mu}_i$



- Decision boundaries are linear
  - **Discriminant function**:

$$\delta_k(x) = x^T \Sigma^{-1} \mu_k - \frac{1}{2} \mu_k^T \Sigma^{-1} \mu_k + \log \pi_k$$

Linear discriminant analysis (LDA)

- Difference LDA vs logistic regression??
  - Coefficients will be estimated differently! (models are different)
- How to estimate coefficients
  - find MLE.

$$\hat{\mu}_c = \frac{1}{N_c} \sum_{i:y_i=c} \mathbf{x}_i, \quad \hat{\Sigma}_c = \frac{1}{N_c} \sum_{i:y_i=c} (\mathbf{x}_i - \hat{\mu}_c)(\mathbf{x}_i - \hat{\mu}_c)^T$$

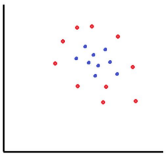
$$\hat{\Sigma} = \frac{1}{N} \sum_{c=1}^{k} N_c \hat{\Sigma}_c$$

  - Sample mean and sample covariance are MLE!
  - If class priors are parameters (**proportional priors**),

$$\hat{\pi}_c = \frac{N_c}{N}$$

LDA versus Logistic regression

- Generative classifiers are easier to fit, discriminative involve numeric optimization
- LDA and Logistic have same model form but are fit differently
- LDA has stronger assumptions than Logistic, some other generative classifiers lead also to logistic expression
- New class in the data?
  - Logistic: fit model again
  - LDA: estimate new parameters from the new data
- Logistic and LDA: complex data fits badly unless interactions are included



- LDA (and other generative classifiers) handle missing data easier
- Standardization and generated inputs:
  - Not a problem for Logistic
  - May affect the performance of the LDA in a complex way
- Outliers affect Σ → LDA is not robust to gross outliers
- LDA is often a good classification method even if the assumption of normality and common covariance matrix are not satisfied.

Naive Bayes:

Naive Bayes classifiers - discrete inputs

- Given $D = \{(X_{m1}, \dots X_{mp}, Y_m), \ m = 1, \dots n\}$
- Assume $X_i \in \{x_1, \dots x_J\}, i = 1, \dots p, Y \in \{y_1 \dots y_K\}$
- Denote $\theta_{ijk} = p(X_i = x_j | Y = y_k)$
  - How many parameters? $(J-1)Kp$
- Denote $\pi_k = p(Y = y_k)$

- Maximum likelihood: assume θ_{ijk} and π_k are constants
  - $\hat{\theta}_{ijk} = \frac{\#\{X_i = x_j \,\&\, Y = y_k\}}{\#\{Y = y_k\}}$
  - $\hat{\pi}_k = \frac{\#\{Y = y_k\}}{n}$
  - Classification using 0-1 loss: $\hat{Y} = \arg\max_y p(Y = y|X)$

predict(newx

Lab 1

- Göra om från xlsx till csv:
  - Öppna xlsx
  - Spara som -> .csv
  - Spara
  - “Använd formatet Text CSV”
  - OK
  - data = read.csv(“file.csv”, dec=’,’)
- n = dim(data)[1]
- $\sqrt{\sum_i X_i^2} = as.matrix(sqrt(rowSums(X\hat{}2)),\ nrow = n,\ ncol = 1)$
- $$c(X, Y) = \frac{X^T Y}{\sqrt{\sum_i X_i^2}\sqrt{\sum_i Y_i^2}}$$
- Implement from scratch the K-nearest neighbors method which is based on distance function $d(X, Y)$ (**use only basic R functions).** Your code should be presented as a function *knearest*(*data, K, newdata*) that uses *data* as training data and then returns the predicted class probabilities for *newdata* by using K-nearest neighbor approach.
  - **Note**: R implementations can be very slow if inner loops are used. In order to efficiently compute $d(X, Y)$ for several observations X and Y represented by rows of matrices $\boldsymbol{X}$ and $\boldsymbol{Y}$, you may do the following:
    i. Compute $\widehat{\boldsymbol{X}}$ by dividing each row X_i of matrix $\boldsymbol{X}$ by $\sqrt{\sum_j X_{ij}^2}$ (use function rowSums())
    ii. Compute $\widehat{\boldsymbol{Y}}$ by dividing each row Y_i of matrix $\boldsymbol{Y}$ by $\sqrt{\sum_j Y_{ij}^2}$ (use function rowSums())
    iii. Compute matrix $C = \|c(X_i, Y_j)\|$ as $\widehat{\boldsymbol{X}}\widehat{\boldsymbol{Y}}^{\boldsymbol{T}}$
    iv. Compute distance matrix $D = 1 - C$
- **TIPS:** Om du är osäker på vilket som är vilken i confusion matrix så kör sum(test[,49]) eller sum(pred) och kolla vilken rad/kolumn som stämmer in med summan.
  table(y[,49], yfit, dnn=c("TRUE", "PRED"))
-
-

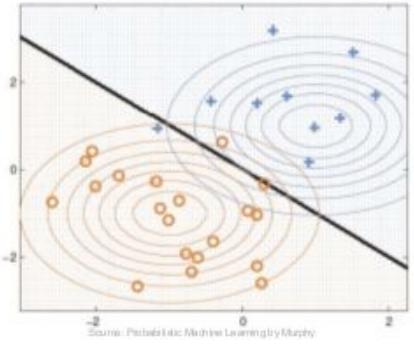
Lab 2

To extract the boundary we set the function for male and female equal and solve the system.

$$\beta_{0m} + \beta_{1m} RW + \beta_{2m} CL = \beta_{0f} + \beta_{1f} RW + \beta_{2f} CL$$

$$RW = \frac{\beta_{0f} + \beta_{2f} CL - (\beta_{0m} + \beta_{2m} CL)}{\beta_{1m} - \beta_{1f}} = \frac{-9.87 - 1.95 CL}{-5.68} = 1.73 + 0.34 CL$$

- Assumtion $\Sigma_i = \Sigma, i = 1, \dots K$
- Then $p(y = c_i|x) = softmax(w_i^T x + w_{0i})$ →exactly the same form as the logistic regression
  - $w_{0i} = -\frac{1}{2}\boldsymbol{\mu}_i^T \Sigma^{-1} \boldsymbol{\mu}_i + \log \pi_i$
  - $w_i = \boldsymbol{\Sigma}^{-1} \boldsymbol{\mu}_i$



- Decision boundaries are linear
  - **Discriminant function**:

$$\delta_k(x) = x^T \Sigma^{-1} \mu_k - \frac{1}{2} \mu_k^T \Sigma^{-1} \mu_k + \log \pi_k$$

Linear discriminant analysis (LDA)

- Difference LDA vs logistic regression??
  - Coefficients will be estimated differently! (models are different)
- How to estimate coefficients
  - find MLE.

$$\hat{\boldsymbol{\mu}}_c = \frac{1}{N_c} \sum_{i:y_i=c} \mathbf{x}_i, \quad \hat{\boldsymbol{\Sigma}}_c = \frac{1}{N_c} \sum_{i:y_i=c} (\mathbf{x}_i - \hat{\boldsymbol{\mu}}_c)(\mathbf{x}_i - \hat{\boldsymbol{\mu}}_c)^T$$

$$\hat{\Sigma} = \frac{1}{N} \sum_{c=1}^{k} N_c \hat{\Sigma}_c$$

  - Sample mean and sample covariance are MLE!
  - If class priors are parameters (**proportional priors**),

$$\hat{\pi}_c = \frac{N_c}{N}$$

Naive Bayes in R

- naiveBayes in package **e1071**

Example: Satisfaction of householders with their present housing circumstances

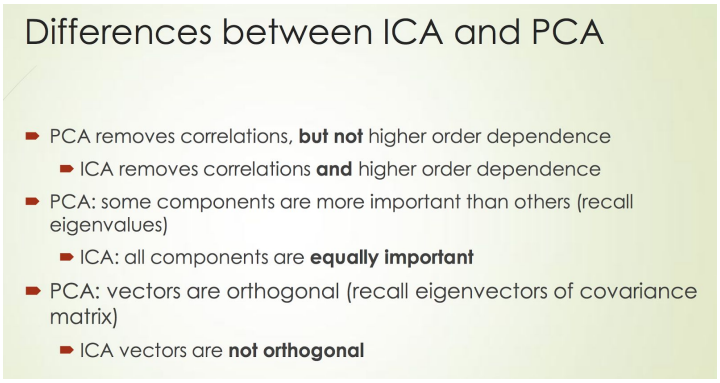
```
library(MASS)
library(e1071)
n=dim(housing)[1]
ind=rep(1:n, housing[,5])
housing1=housing[ind,-5]

fit=naiveBayes(Sat~., data=housing1)
fit

Yfit=predict(fit, newdata=housing1)
table(Yfit,housing1$Sat)
```

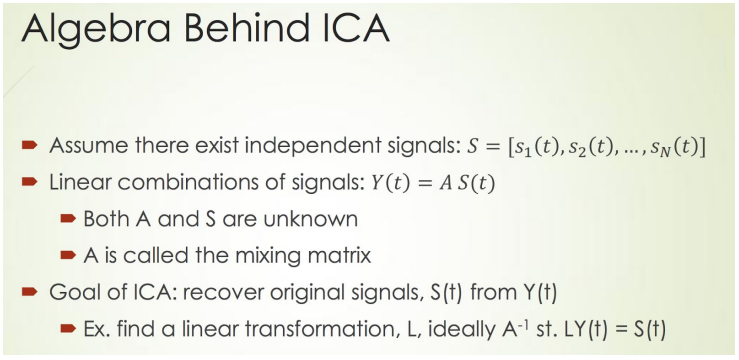
```
> table(Yfit,housing1$Sat)

Yfit      Low Medium High
  Low     294    162  144
  Medium   20     23   20
  High    253    261  504
```

PCA:

When to use PCA:

- Latent features driving the patterns in data
- Dimensionality reduction
    - Visualize high-dimensional data (PC1 and PC2 only, identify outliers)
    - Reduce noise (first few PCs count for the actual variation, the rest is noise)
    - Make other algorithms (regression, classification) work better because there are fewer features





- More settings to specify→ more flexible
- Can be faster when M<<p
- Missing values can be handled
- M can be derived if a Bayesian version is used
- Probabilistic PCA can be applied to classification problems directly
- Probabilistic PCA can generate new data

PCR:

Principle component regression

Step 1: Compute principal components $v_1 \ldots v_M$

Step 2: Compute derived data as $z_i = Xv_i$

Step 3: Compute linear regression using data set Z with columns $z_1 .. z_M$, Y

Package: pls

pcr(formula, ncomp, data, scale = FALSE, validation = c("none", "CV", "LOO"...),
pcr.fit=pcr(f, data=train, validation="CV"), summary(pcr.fit), validationplot(pcr.fit, val.type="MSEP")
pcr.fit1=pcr(f, 3,data=train, validation="none")
scores(pcr.fit1) # Scores matrix (new coordinates)

PLS:

Partial least squares regression (PLS)

Step 1: Standardize features to mean zero and variance one

Step 2: Compute the first derived feature by setting

$$\mathbf{z}_1 = \sum_{j=1}^{p} \varphi_{1j} \mathbf{x}_j$$

where the φ_{1j} is projection of **Y** on $\mathbf{x}_j$

Step 3: Orthogonalize $\mathbf{x}_1 \ldots \mathbf{x}_m$ with respect to z_1

Step 4: repeat from step 2 and find $\mathbf{z}_2 .. \mathbf{z}_M$

Step 5: Compute regression of **Y** on $\mathbf{z}_1 .. \mathbf{z}_M$

in R (package: pls)

- plsr

Bootstrap:

Nonparametric bootstrap can be applied to any deterministic estimator, distribution-free
Parametric bootstrap is more precise if the distribution form is correct
Parametric bootstrap works even for small samples
in R:
boot.ci() – 1 parameter
envelope() – many parameters
• Compute value **mle** that estimates model parameters from the data
• Write function **ran.gen** that depends on data and mle and which generates new data
• Write function **statistic** that depend on data which will be generated by ran.gen and should return the estimator

- Confidence interval for Y|X= interval for mean $EY|X$
- Prediction interval for Y|X= interval for $Y|X$

$$Y \sim Distribution(x, w)$$

Prediction band for parametric bootstrap

1. Run parametric bootstrap and get $D_1, \dots D_B$
2. Fit the model to the data and get $\widehat{w}(D_1), \dots \widehat{w}(D_B)$
3. For each X, generate from $Distribution\big(X, \widehat{w}(D_1)\big), \dots$ $Distribution\big(X, \widehat{w}(D_B)\big)$ and apply percentile method
4. Connect the intervals→get the band

Kernel Classification

- The moving window rule gives equal weight to all the points in the ball, which may be counterintuitive. Then,

$$y_k(\boldsymbol{x}) = \begin{cases} 0 & \text{if } \sum_n \mathbf{1}_{\{t_n=1\}} k\left(\frac{\boldsymbol{x}-\boldsymbol{x}_n}{h}\right) \leq \sum_n \mathbf{1}_{\{t_n=0\}} k\left(\frac{\boldsymbol{x}-\boldsymbol{x}_n}{h}\right) \\ 1 & \text{otherwise} \end{cases}$$

where $k : \mathbb{R}^D \to \mathbb{R}$ is a kernel function, which is usually non-negative and monotone decreasing along rays starting from the origin. The parameter h is called smoothing factor or width.

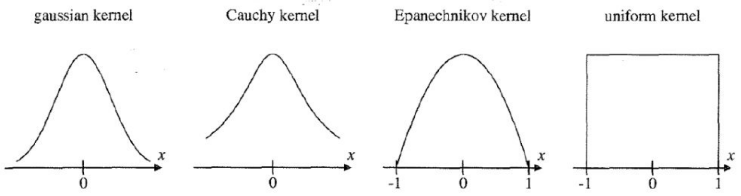


FIGURE 10.3. *Various kernels on $\mathcal{R}$.*

- Gaussian kernel: $k(u) = exp(-||u||^2)$ where $||\cdot||$ is the Euclidean norm.
- Cauchy kernel: $k(u) = 1/(1 + ||u||^{D+1})$
- Epanechnikov kernel: $k(u) = (1 - ||u||^2)\mathbf{1}_{\{||u||\leq 1\}}$
- Moving window kernel: $k(u) = \mathbf{1}_{\{u \in S(0,1)\}}$

Histogram, Moving Window, and Kernel Regression

- Consider regressing an unidimensional continuous random variable on a D-dimensional continuous random variable.
- The best regression function under the squared error loss function is $y^*(\boldsymbol{x}) = \mathbb{E}_Y[y|\boldsymbol{x}]$.
- Since $\boldsymbol{x}$ may not appear in the finite training set $\{(\boldsymbol{x}_n, t_n)\}$ available, then we average over the points in $C(\boldsymbol{x}, h)$ or $S(\boldsymbol{x}, h)$, or kernel-weighted average over all the points.
- In other words,

$$y_C(\boldsymbol{x}) = \frac{\sum_{\boldsymbol{x}_n \in C(\boldsymbol{x},h)} t_n}{|\{\boldsymbol{x}_n \in C(\boldsymbol{x}, h)\}|}$$

or

$$y_S(\boldsymbol{x}) = \frac{\sum_{\boldsymbol{x}_n \in S(\boldsymbol{x},h)} t_n}{|\{\boldsymbol{x}_n \in S(\boldsymbol{x}, h)\}|}$$

or

$$y_k(\boldsymbol{x}) = \frac{\sum_n k\left(\frac{\boldsymbol{x}-\boldsymbol{x}_n}{h}\right) t_n}{\sum_n k\left(\frac{\boldsymbol{x}-\boldsymbol{x}_n}{h}\right)}$$

Histogram, Moving Window, and Kernel Density Estimation

- Consider density estimation for a D-dimensional continuous random variable.
- Let $R \subseteq \mathbb{R}^D$ and $\boldsymbol{x} \in R$. Then,

$$P = \int_R p(\boldsymbol{x}) d\boldsymbol{x} \simeq p(\boldsymbol{x}) Volume(R)$$

and the number of the N training points $\{\boldsymbol{x}_n\}$ that fall inside R is

$$|\{\boldsymbol{x}_n \in R\}| \simeq P\, N$$

and thus

$$p(\boldsymbol{x}) \simeq \frac{|\{\boldsymbol{x}_n \in R\}|}{N\, Volume(R)}$$

- Then,

$$p_C(\boldsymbol{x}) = \frac{|\{\boldsymbol{x}_n \in C(\boldsymbol{x}, h)\}|}{N\, Volume(C(\boldsymbol{x}, h))}$$

or

$$p_S(\boldsymbol{x}) = \frac{|\{\boldsymbol{x}_n \in S(\boldsymbol{x}, h)\}|}{N\, Volume(S(\boldsymbol{x}, h))}$$

or

$$p_k(\boldsymbol{x}) = \frac{1}{N} \sum_n k\Big(\frac{\boldsymbol{x} - \boldsymbol{x}_n}{h}\Big)$$

assuming that $k(u) \geq 0$ for all u and $\int k(u) du = 1$.

Histogram, Moving Window, and Kernel Density Estimation

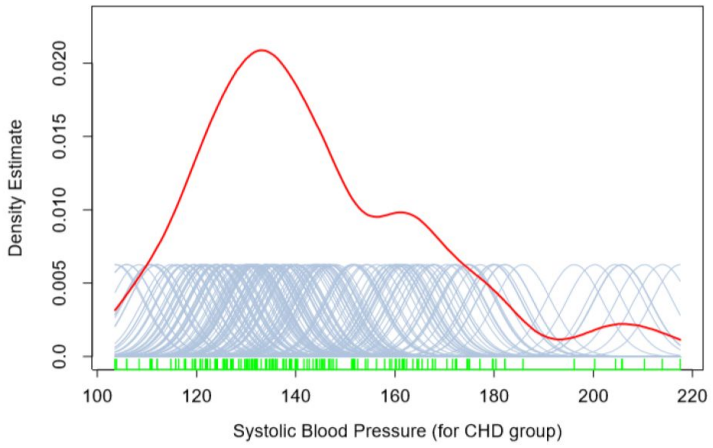
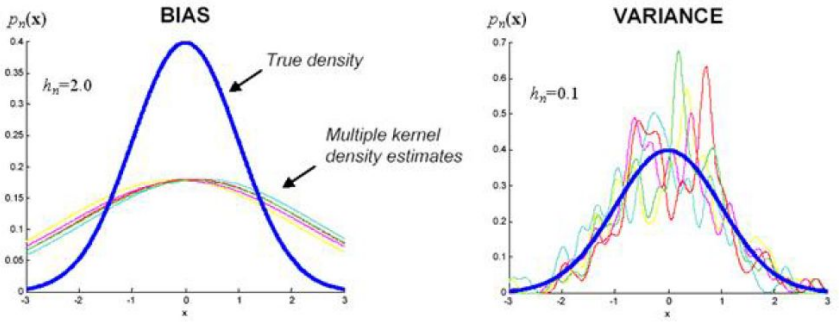


FIGURE 6.13. *A kernel density estimate for systolic blood pressure (for the CHD group). The density estimate at each point is the average contribution from each of the kernels at that point. We have scaled the kernels down by a factor of 10 to make the graph readable.*

- From kernel density estimation to kernel classification:
  1. Estimate $p(\boldsymbol{x}|y = 0)$ and $p(\boldsymbol{x}|y = 1)$ using the methods just seen.
  2. Estimate $p(y)$ as class proportions.
  3. Compute $p(y|\boldsymbol{x}) \propto p(\boldsymbol{x}|y)p(y)$ by Bayes theorem.

Kernel Selection

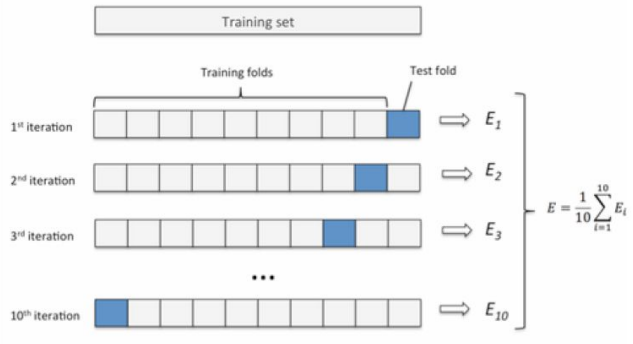
- How to choose the right kernel and width ? E.g., by cross-validation.
- What does "right" mean ? E.g., minimize loss function.
- Note that the width of the kernel corresponds to a bias-variance trade-off.



- Small width implies considering few points. So, the variance will be large (similar to the variance of a single point). The bias will be small since the points considered are close to $\boldsymbol{x}$.
- Large width implies considering many points. So, the variance will be small and the bias will be large.

Kernel Selection

- Recall the following from previous lectures.
- Cross-validation is a technique to estimate the prediction error of a model.

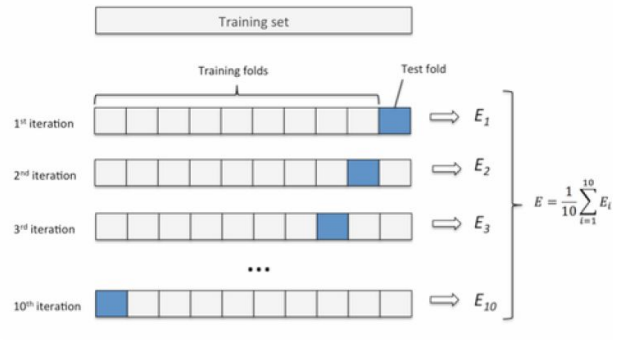


- If the training set contains N points, note that cross-validation estimates the prediction error when the model is trained on $N - N/K$ points.
- Note that the model returned is trained on N points. So, cross-validation overestimates the prediction error of the model returned.
- This seems to suggest that a large K should be preferred. However, this typically implies a large variance of the error estimate, since there are only N/K test points.
- Typically, $K = 5, 10$ works well.

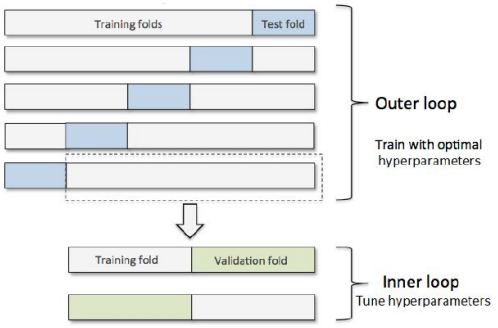
Kernel Selection

- Model: For example, ridge regression with a given value for the penalty factor λ. Only the parameters (weights) need to be determined (closed-form solution).
- Model selection: For example, determine the value for the penalty factor λ. Another example, determine the kernel and width for kernel classification, regression or density estimation. In either case, we do not have a continuous criterion to optimize. Solution: **Nested** cross-validation.

Cross-validation for estimating model prediction error



Nested cross-validation for estimating model **selection** prediction error



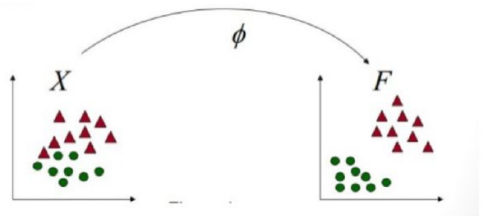
- Error overestimation may not be a concern for model selection. So, $K = 2$ may suffice in the inner loop.
- Which is the fitted model returned by nested cross-validation ?

Kernel Trick

- The kernel function $k\left(\frac{x-x'}{h}\right)$ is invariant to translations, and it can be generalized as $k(\boldsymbol{x}, \boldsymbol{x}')$. For instance,
  - Polynomial kernel: $k(\boldsymbol{x}, \boldsymbol{x}') = (\boldsymbol{x}^T \boldsymbol{x}' + c)^M$
  - Gaussian kernel: $k(\boldsymbol{x}, \boldsymbol{x}') = exp(-\|\boldsymbol{x} - \boldsymbol{x}'\|^2 / 2\sigma^2)$
- If the matrix

$$\begin{pmatrix} k(\boldsymbol{x}_1, \boldsymbol{x}_1) & \dots & k(\boldsymbol{x}_1, \boldsymbol{x}_N) \\ \vdots & \dots & \vdots \\ k(\boldsymbol{x}_N, \boldsymbol{x}_1) & \dots & k(\boldsymbol{x}_N, \boldsymbol{x}_N) \end{pmatrix}$$

is symmetric and positive semi-definite for all choices of $\{\boldsymbol{x}_n\}$, then $k(\boldsymbol{x}, \boldsymbol{x}') = \phi(\boldsymbol{x})^T \phi(\boldsymbol{x}')$ where $\phi(\cdot)$ is a mapping from the input space to the feature space.



- The feature space may be non-linear and even infinite dimensional. For instance,

$$\phi(\boldsymbol{x}) = (x_1^2, x_2^2, \sqrt{2}x_1x_2, \sqrt{2c}x_1, \sqrt{2c}x_2, c)$$

for the polynomial kernel with $M = D = 2$.

Kernel Trick

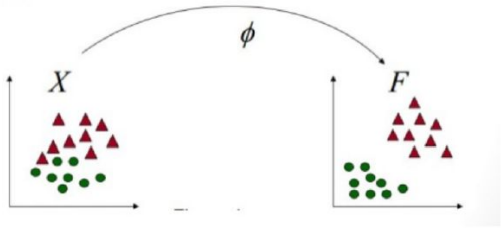
- Consider again moving window classification, regression, and density estimation.
- Note that $\boldsymbol{x}_n \in S(\boldsymbol{x}, h)$ if and only if $\|\boldsymbol{x} - \boldsymbol{x}_n\| \leq h$.
- Note that

$$\|\boldsymbol{x} - \boldsymbol{x}_n\| = (\boldsymbol{x} - \boldsymbol{x}_n)^T(\boldsymbol{x} - \boldsymbol{x}_n) = \boldsymbol{x}^T\boldsymbol{x} + \boldsymbol{x}_n^T\boldsymbol{x}_n - 2\boldsymbol{x}^T\boldsymbol{x}_n$$

- Then,

$$\begin{aligned} \|\phi(\boldsymbol{x}) - \phi(\boldsymbol{x}_n)\| &= \phi(\boldsymbol{x}^T)\phi(\boldsymbol{x}) + \phi(\boldsymbol{x}_n^T)\phi(\boldsymbol{x}_n) - 2\phi(\boldsymbol{x}^T)\phi(\boldsymbol{x}_n) \\ &= k(\boldsymbol{x}, \boldsymbol{x}) + k(\boldsymbol{x}_n, \boldsymbol{x}_n) - 2k(\boldsymbol{x}, \boldsymbol{x}_n) \end{aligned}$$

- So, the distance is now computed in a (hopefully) more convenient space.



- Note that we do not need to compute $\phi(\boldsymbol{x})$ and $\phi(\boldsymbol{x}_n)$.

Kernel Trick

- Two alternatives for building $k(\boldsymbol{x}, \boldsymbol{x}')$:
  - Choose a convenient $\phi(\boldsymbol{x})$ and let $k(\boldsymbol{x}, \boldsymbol{x}') = \phi(\boldsymbol{x})^T\phi(\boldsymbol{x}')$.
  - Build it from existing kernel functions as follows.

Techniques for Constructing New Kernels.

Given valid kernels $k_1(\mathbf{x}, \mathbf{x}')$ and $k_2(\mathbf{x}, \mathbf{x}')$, the following new kernels will also be valid:

$$\begin{aligned} k(\mathbf{x}, \mathbf{x}') &= ck_1(\mathbf{x}, \mathbf{x}') && (6.13) \\ k(\mathbf{x}, \mathbf{x}') &= f(\mathbf{x})k_1(\mathbf{x}, \mathbf{x}')f(\mathbf{x}') && (6.14) \\ k(\mathbf{x}, \mathbf{x}') &= q\left(k_1(\mathbf{x}, \mathbf{x}')\right) && (6.15) \\ k(\mathbf{x}, \mathbf{x}') &= \exp\left(k_1(\mathbf{x}, \mathbf{x}')\right) && (6.16) \\ k(\mathbf{x}, \mathbf{x}') &= k_1(\mathbf{x}, \mathbf{x}') + k_2(\mathbf{x}, \mathbf{x}') && (6.17) \\ k(\mathbf{x}, \mathbf{x}') &= k_1(\mathbf{x}, \mathbf{x}')k_2(\mathbf{x}, \mathbf{x}') && (6.18) \\ k(\mathbf{x}, \mathbf{x}') &= k_3\left(\phi(\mathbf{x}), \phi(\mathbf{x}')\right) && (6.19) \\ k(\mathbf{x}, \mathbf{x}') &= \mathbf{x}^{\mathrm{T}}\mathbf{A}\mathbf{x}' && (6.20) \\ k(\mathbf{x}, \mathbf{x}') &= k_a(\mathbf{x}_a, \mathbf{x}_a') + k_b(\mathbf{x}_b, \mathbf{x}_b') && (6.21) \\ k(\mathbf{x}, \mathbf{x}') &= k_a(\mathbf{x}_a, \mathbf{x}_a')k_b(\mathbf{x}_b, \mathbf{x}_b') && (6.22) \end{aligned}$$

where $c > 0$ is a constant, $f(\cdot)$ is any function, $q(\cdot)$ is a polynomial with nonnegative coefficients, $\phi(\mathbf{x})$ is a function from $\mathbf{x}$ to $\mathbb{R}^M$, $k_3(\cdot, \cdot)$ is a valid kernel in $\mathbb{R}^M$, $\mathbf{A}$ is a symmetric positive semidefinite matrix, $\mathbf{x}_a$ and $\mathbf{x}_b$ are variables (not necessarily disjoint) with $\mathbf{x} = (\mathbf{x}_a, \mathbf{x}_b)$, and k_a and k_b are valid kernel functions over their respective spaces.

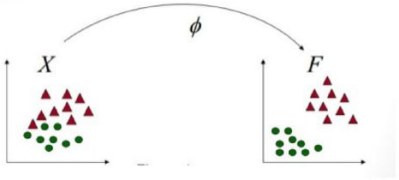
Support Vector Machines for Classification

- Consider binary classification with input space $\mathbb{R}^D$.
- Consider a training set $\{(\boldsymbol{x}_n, t_n)\}$ where $t_n \in \{-1, +1\}$.
- Consider using the linear model

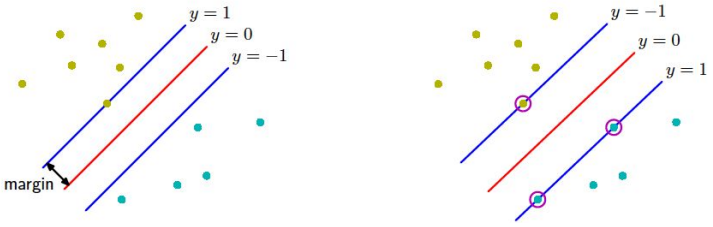
$$y(\boldsymbol{x}) = \boldsymbol{w}^T \phi(\boldsymbol{x}) + b$$

so that a new point $\boldsymbol{x}$ is classified according to the sign of $y(\boldsymbol{x})$.

- Assume that the training set is linearly separable in the feature space (but not necessarily in the input space), i.e. $t_n y(\boldsymbol{x}_n) > 0$ for all n.



- Aim for the separating hyperplane that maximizes the margin (i.e. the smallest perpendicular distance from any point to the hyperplane) so as to minimize the generalization error.



Neural Networks

- Consider binary classification with input space $\mathbb{R}^D$. Consider a training set $\{(\boldsymbol{x}_n, t_n)\}$ where $t_n \in \{-1, +1\}$.
- SVMs classify a new point $\boldsymbol{x}$ according to

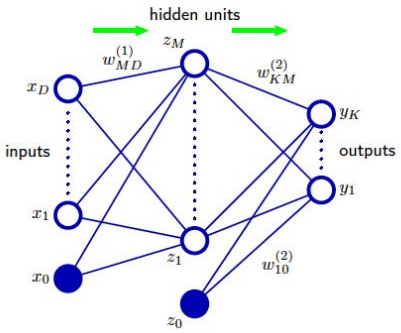
$$y(\boldsymbol{x}) = \text{sgn}\left(\sum_{m \in \mathcal{S}} a_m t_m k(\boldsymbol{x}, \boldsymbol{x}_m) + b\right)$$

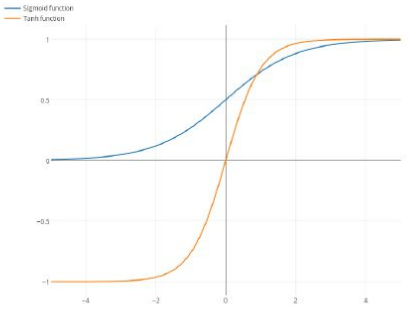
- Consider regressing an unidimensional continuous random variable on a D-dimensional continuous random variable. Consider a training set $\{(\boldsymbol{x}_n, t_n)\}$
- For a new point $\boldsymbol{x}$, SVMs predict

$$y(\boldsymbol{x}) = \sum_{m \in \mathcal{S}} (a_n - \widehat{a}_n) k(\boldsymbol{x}, \boldsymbol{x}_m) + b$$

- SVMs imply data-selected user-defined basis functions.
- NNs imply a user-defined number of data-selected basis functions.

Neural Networks





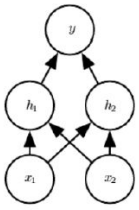
- Activations: $a_j = \sum_i w_{ji}^{(1)} x_i + w_{j0}^{(1)}$
- Hidden units and activation function: $z_j = h(a_j)$
- Output activations: $a_k = \sum_j w_{kj}^{(2)} z_j + w_{k0}^{(2)}$
- Output activation function for regression: $y_k(\boldsymbol{x}) = a_k$
- Output activation function for classification: $y_k(\boldsymbol{x}) = \sigma(a_k)$
- Sigmoid function: $\sigma(a) = \frac{1}{1+\exp(-a)}$
- Two-layer NN:

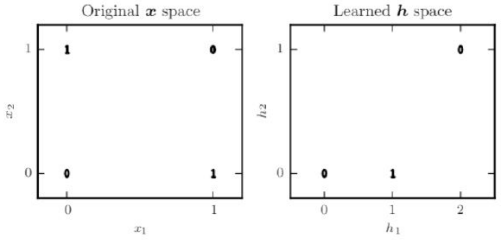
$$y_k(\boldsymbol{x}) = \sigma\Big(\sum_j w_{kj}^{(2)} h\Big(\sum_i w_{ji}^{(1)} x_i + w_{j0}^{(1)}\Big) + w_{k0}^{(2)}\Big)$$

- Evaluating the previous expression is known as forward propagation. The NN is said to have a feed-forward architecture.
- All the previous is, of course, generalizable to more layers.

Neural Networks

- Solving the XOR problem with NNs.
- No line shatters the points in the original space.
- The NN represents a mapping of the input space to an alternative space where a line can shatter the points. Note that the points (0,1) and (1,0) are mapped both to the point (1,0).
- It resembles SVMs.





$w_{11}^{(1)} = w_{12}^{(1)} = w_{21}^{(1)} = w_{22}^{(1)} = 1$
$w_{10}^{(1)} = 0,\ w_{20}^{(1)} = -1$
$h_j = z_j = h(a_j) = \max\{0, a_j\}$
$w_{11}^{(2)} = 1,\ w_{12}^{(2)} = -2$
$w_{10}^{(2)} = 0$
$y = y_k = a_k$

Backpropagation Algorithm

- Consider regressing an K-dimensional continuous random variable on a D-dimensional continuous random variable.
- Consider a training set $\{(\boldsymbol{x}_n, \boldsymbol{t}_n)\}$. Consider minimizing the error function

$$E(\boldsymbol{w}^t) = \sum_n E_n(\boldsymbol{w}^t) = \sum_n \frac{1}{2}\|\boldsymbol{y}(\boldsymbol{x}_n) - \boldsymbol{t}_n\|^2 = \sum_n \sum_k \frac{1}{2}(y_k(\boldsymbol{x}_n) - t_{nk})^2$$

- The weight space is highly multimodal and, thus, we have to resort to approximate iterative methods to minimize the previous expression.
- Batch gradient descent

$$\boldsymbol{w}^{t+1} = \boldsymbol{w}^t - \eta_n \nabla E(\boldsymbol{w}^t)$$

where $\eta_n > 0$ is the learning rate ($\sum_n \eta_n = \infty$ and $\sum_n \eta_n^2 < \infty$ to ensure convergence, e.g. $\eta_n = 1/n$).

- Sequential, stochastic or online gradient descent

$$\boldsymbol{w}^{t+1} = \boldsymbol{w}^t - \eta_n \nabla E_n(\boldsymbol{w}^t)$$

where n is chosen randomly or sequentially.

- Sequential gradient descent is less affected by the multimodality problem, as a local minimum of the whole data will not be generally a local minimum of each individual point.

Backpropagation Algorithm

- Since E_n depends on w_{ji} only via a_j, and $a_j = \sum_i w_{ji} x_i$, then

$$\frac{\partial E_n}{\partial w_{ji}} = \frac{\partial E_n}{\partial a_j}\frac{\partial a_j}{\partial w_{ji}} = \frac{\partial E_n}{\partial a_j} x_i = \delta_j x_i$$

- Since E_n depends on a_j only via a_k, then

$$\delta_j = \frac{\partial E_n}{\partial a_j} = \sum_k \frac{\partial E_n}{\partial a_k}\frac{\partial a_k}{\partial a_j} = \sum_k \delta_k \frac{\partial a_k}{\partial a_j}$$

- Since $a_k = \sum_j w_{kj} z_j$ and $z_j = h(a_j)$, then

$$\frac{\partial a_k}{\partial a_j} = h'(a_j) w_{kj}$$

- Putting all together, we have that

$$\delta_j = h'(a_j) \sum_k \delta_k w_{kj}$$

- Since $y_k = a_k$ for regression, then

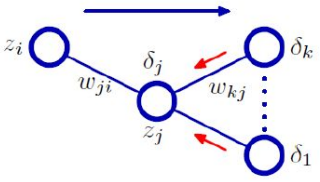
$$\delta_k = \frac{\partial E_n}{\partial a_k} = y_k - t_k$$

- Backpropagation algorithm:
  1. Forward propagate to compute activations, and hidden and output units.
  2. Compute δ_k for the output units.
  3. Backpropagate the δ's, i.e. evaluate δ_j for the hidden units recursively.
  4. Compute the required derivatives.

Backpropagation Algorithm

- Backpropagation algorithm:
  1. Forward propagate to compute activations, and hidden and output units.
  2. Compute δ_k for the output units.
  3. Backpropagate the δ's, i.e. evaluate δ_j for the hidden units recursively.
  4. Compute the required derivatives.

Figure 5.7 Illustration of the calculation of δ_j for hidden unit j by backpropagation of the δ's from those units k to which unit j sends connections. The blue arrow denotes the direction of information flow during forward propagation, and the red arrows indicate the backward propagation of error information.

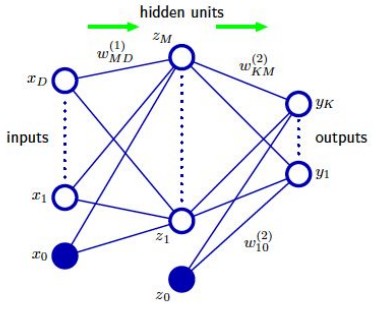


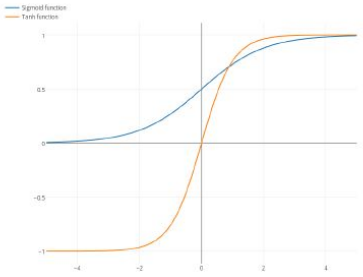
- Since $y_k = \sigma(a_k)$ for classification, then

$$\delta_k = \frac{\partial E_n}{\partial a_k} = \sigma(a_k)(1 - \sigma(a_k))$$

- This is an example of embarrassingly parallel algorithm.

Backpropagation Algorithm





- Example: $y_k = a_k$, and $z_j = h(a_j) = \tanh(a_j)$ where $\tanh(a) = \frac{\exp(a)-\exp(-a)}{\exp(a)+\exp(-a)}$.
- Note that $h'(a) = 1 - h(a)^2$.
- Backpropagation:
  1. Forward propagation, i.e. compute
  $$a_j = \sum_i w_{ji} x_i \text{ and } z_j = h(a_j) \text{ and } y_k = \sum_j w_{kj} z_j$$
  2. Compute
  $$\delta_k = y_k - t_k$$
  3. Backpropagate, i.e. compute
  $$\delta_j = (1 - z_j^2) \sum_k w_{kj} \delta_k$$
  4. Compute
  $$\frac{\partial E_n}{\partial w_{kj}} = \delta_k z_j \text{ and } \frac{\partial E_n}{\partial w_{ji}} = \delta_j x_i$$

Backpropagation Algorithm

- The weight space is non-convex and has many symmetries, plateaus and local minima. So, the initialization of the weights in the backpropagation algorithm is crucial.
- Hints based on experimental rather than theoretical analysis:
  - Initialize the weights to different values, otherwise they would be updated in the same way because the algorithm is deterministic, and so creating redundant hidden units.
  - Initialize the weights at random, but
    - too small magnitude values may cause losing signal in the forward or backward passes, and
    - too big magnitude values may cause the activation function to saturate and lose gradient.
  - Initialize the weights according to prior knowledge: Almost-zero for hidden units that are unlikely to interact, and bigger magnitude values for the rest.
  - Initialize the weights to almost-zero values so that the initial model is almost-linear, i.e. the sigmoid function is almost-linear around the zero. Let the algorithm to introduce non-linearities where needed.
    - Note however that this initialization makes the sigmoid function take a value around half its saturation level. That is why the hyperbolic tangent function is sometimes preferred in practice.

Regularization

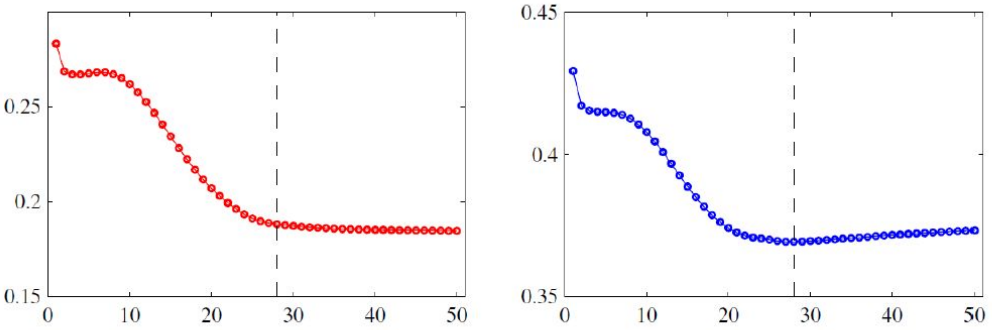


Figure 5.12 An illustration of the behaviour of training set error (left) and validation set error (right) during a typical training session, as a function of the iteration step, for the sinusoidal data set. The goal of achieving the best generalization performance suggests that training should be stopped at the point shown by the vertical dashed lines, corresponding to the minimum of the validation set error.

- Regularization when learning the parameters: Early stopping the backpropagation algorithm according to the error on some validation data.
- Regularization when learning the structure:
  - Cross-validation.
  - Penalizing complexity according to

$$E(\boldsymbol{w}) + \frac{\lambda_1}{2}||\boldsymbol{w}^{(1)}||^2 + \frac{\lambda_2}{2}||\boldsymbol{w}^{(2)}||^2$$

instead of the classical

$$E(\boldsymbol{w}) + \frac{\lambda}{2}||\boldsymbol{w}||^2$$

and choose λ_1 and λ_2 by cross-validation.

- To see why, let $z_j = h(\sum_i w_{ji} x_i + w_{j0})$ and $y_k = \sum_j w_{kj} z_j + w_{k0}$.
- Consider the linear transformation $x_i \to a x_i + b$.
- Transform $w_{j0} \to w_{j0} - \frac{b}{a} \sum_i w_{ji}$ and $w_{ji} \to \frac{1}{a} w_{ji}$.
- Both NNs define the same mapping, but they may receive different penalty for complexity $\|\boldsymbol{w}\|^2$.
- Solution: Penalize according to $\frac{\lambda_1}{2}\|\boldsymbol{w}^{(1)}\|^2 + \frac{\lambda_2}{2}\|\boldsymbol{w}^{(2)}\|^2$ and let $\lambda_1 \to a^{1/2}\lambda_1$.
- Finally, note that the effect of the penalty is simply to add $\lambda_1 w_{ji}$ and $\lambda_2 w_{kj}$ to the appropriate derivatives.

Limitations of Neural Networks

Theorem (Universal approximation theorem)

For every continuous function $f : [a, b]^D \to \mathbb{R}$ and for every $\epsilon > 0$, there exists a NN with one hidden layer such that

$$\sup_{\mathbf{x} \in [a,b]^D} |f(\mathbf{x}) - y(\mathbf{x})| < \epsilon$$

Theorem (Universal classification theorem)

Let $\mathcal{C}^{(k)}$ contain all classifiers defined by NNs of one hidden layer with k hidden units and the sigmoid activation function. Then, for any distribution $p(\mathbf{x}, t)$,

$$\lim_{k \to \infty} \inf_{y \in \mathcal{C}^{(k)}} E_X[y(\mathbf{x})] - E_X[p(t|\mathbf{x})] = 0$$

- How many hidden units has such a NN ?
- How much data do we need to learn such a NN (and avoid overfitting) via the backpropagation algorithm ?
- How fast does the backpropagation algorithm converge to such a NN ? Assuming that it does not get trapped in a local minimum...
- The answer to the last two questions depends on the first: More hidden units implies more training time and higher generalization error.

Limitations of Neural Networks

- How many hidden units does the NN need ?
- Any Boolean function can be written in disjunctive normal form (OR of ANDs) or conjunctive normal form (AND of ORs). This is a depth-two logical circuit.
- For most Boolean functions, the size of the circuit is exponential in the size of the input.
- However, there are Boolean functions that have a polynomial-size circuit of depth k and an exponential-size circuit of depth $k-1$.
- Then, there is no universally right depth. Ideally, we should let the data determine the right depth.

Theorem (No free lunch theorem)

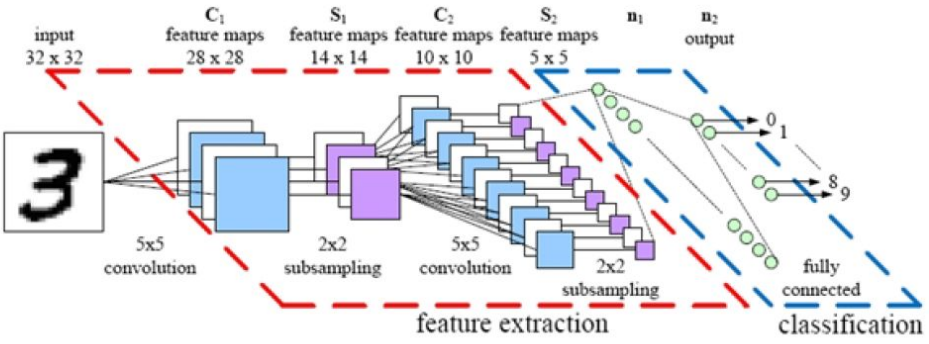
For any algorithm, good performance on some problems comes at the expense of bad performance on some others.

- A deep NN is a function that maps input to output.
- The mapping is formed by composing many simpler functions.
- Each layer provides a new representation of the input, i.e. complex concepts are built from simpler ones.
- The representation is learned automatically from data.

Deep Neural Networks

- Training DNNs is difficult:
  - Typically, poorer generalization than (shallow) NNs.
  - The gradient may vanish/explode as we move away from the output layer, due to multiplying small/big quantities. E.g. the gradient of σ and tanh is in $[0, 1)$. So, they may only suffer the gradient vanishing problem. Other activations functions may suffer the gradient exploding problem.
  - There may be larger plateaus and many more local minima than with NNs.
- Training DNNs is doable:
  - Convolutional networks, particularly suitable for image processing.
  - Rectifier activation function, a new activation function.
  - Layer-wise pre-training, to find a good starting point for training.
- In addition to performance, the computational demands of the training must be considered, e.g. CPU, GPU, memory, parallelism, etc.
  - The authors state that GoogLeNet was trained "using modest amount of model and data-parallelism. Although we used a CPU based implementation only, a rough estimate suggests that the GoogLeNet network could be trained to convergence using few high-end GPUs within a week, the main limitation being the memory usage".

Convolutional Networks



- DNNs suitable for image recognition, since they exhibit invariance to translation, scaling, rotations, and warping.
- Convolution: Detection of local features, e.g. a_j is computed from a 5x5 pixel patch of the image.
- To achieve invariance, the units in the convolution layer share the same activation function and weights.
- Subsampling: Combination of local features into higher-order features, e.g. a_k is compute from a 2x2 pixel patch of the convoluted image.
- There are several feature maps in each layer, to compensate the reduction in resolution by increasing in the number of features being detected.
- The final layer is a regular NN for classification.

- DNNs allow increased depth because
  - they are sparse, which allows the gradient to propagate further, and
  - they have relatively few weights to fit due to feature locality and weight sharing.
- The backpropagation algorithm needs to be adapted, by modifying the derivatives with respect to the weights in each convolution layer m.
- Since E_n depends on $w_i^{(m)}$ only via $a_j^{(m)}$, and $a_j^{(m)} = \sum_{i \in L_j^{(m)}} w_i^{(m)} z_i^{(m-1)}$ where $L_j^{(m)}$ is the set of indexes of the input units, then

$$\frac{\partial E_n}{\partial w_i^{(m)}} = \sum_j \frac{\partial E_n}{\partial a_j^{(m)}} \frac{\partial a_j^{(m)}}{\partial w_i^{(m)}} = \sum_j \delta_j^{(m)} z_i^{(m-1)}$$

- Note that $w_i^{(m)}$ does not depend on j by weight sharing, whereas $i \in L_j^{(m)}$ by feature locality.

Rectifier Activation Function

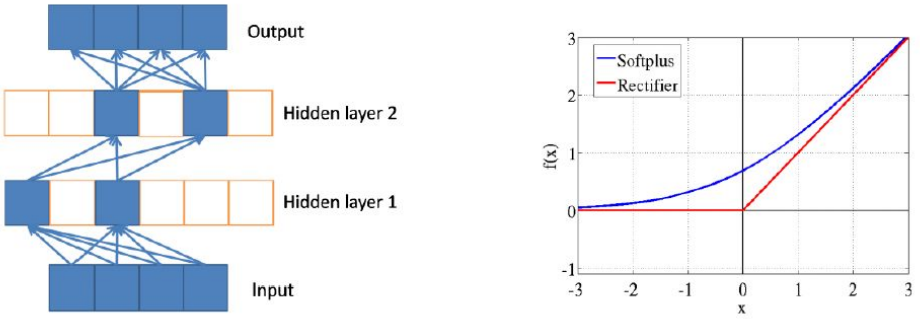


Figure 2: *Left:* **Sparse propagation of activations and gradients in a network of rectifier units.** The input selects a subset of active neurons and computation is linear in this subset. *Right:* **Rectifier and softplus activation functions.** The second one is a smooth version of the first.

- $rectifier(x) = \max\{0, x\}$, i.e. hidden units are off or operating in a linear regime.
- The most popular choice nowadays.
- Sparsity promoting: Uniform initialization of the weights implies that around 50 % of the hidden units are off.
- Piece-wise linear mapping: The input selects which hidden units are active, and the output is a liner function of the input in the selected hidden units.

- It simplifies the backpropagation algorithm as $h'(a_j) = 1$ for the selected units. So, there is no gradient vanishing on the paths of selected units.
- Note that $h'(0)$ does not exist since $h'_+(0) \neq h'_-(0)$. We can get around this problem by simply returning one of two one-sided derivatives. Or using a generalization of the rectifier function.
- Regularization is typically added to prevent numerical problems due to the activation being unbounded, e.g. when forward propagating.

Layer-Wise Pre-Training

  - Supervised version:
    1. Train each layer of the DNN as if it was the hidden layer in a depth-two NN. As input, use the output of the last of the previously trained layers.
    2. Run the backpropagation algorithm to fine-tune the weights.
  - The pre-training aims to find a good starting point for the subsequent run of the backpropagation algorithm.
  - Unsupervised version: Similar to the supervised one but the hidden layers (except the last one) are trained to learn an encoding of the output of the previous layer, instead of the original classification or regression function.
- Direct application of the backpropagation algorithm to DNNs produces poor results.
- Convolutional networks: It makes the backpropagation algorithm more efficient by using local features and weight sharing. This also achieves invariance, which is particularly important for image processing.
- Rectifier activation function: Free of gradient vanishing problem and it simplifies the backpropagation algorithm.
- Layer-wise pre-training: Heuristic weight initialization to alleviate the local optima problem.