# Machine Intelligence I & II
# Review

Jan Herding

November 4, 2011

## 1 Supervised Learning

In supervised learning, the main gool is to learn associations between a set of inputs and the corresponding known outputs. Then, after having learned the assumed associations between input and output, the trained system should be able to predict the output of novel data samples (generalization).

### 1.1 Perceptron & Multi-Layer Perceptron (MLP)

The simplest idea of a connectionist neuron is the perceptron. And the most common feed-forward network structure is a combination of many of these units, i.e. a MLP.

**Perceptron:** A weighted linear sum of inputs (including a bias/threshold, typically as $w_0$ with $x_0 = 1$) which is then fed into a static nonlinearity:

$$y = f\left(\sum_i w_i x_i\right)$$

The weighted sum of inputs is also referred to as $h$, the input of a single perceptron/unit/neuron.

**MLP:** A layered structure of single perceptrons. Basically, the acrhitecture can be divided into an input layer, hidden layer(s) and an output layer. The output of a neuron in each layer can only be connected to a neuron in its subsequent layer. All neurons in one layer get an input from all neurons in the previous layer, so the architecture is fully connected between layers.

The transfer functions of all neurons within a MLP should be similar and nonlinear. If they weren't nonlinear, the MLP could be replaced by a single perceptron. A proper MLP is then a universal function estimator.

### 1.2 Performance Measure & Model Selection

To assess how good a certain model/network can predict the outcome to a given set of inputs, the deviation of the estimated output $\hat{y}$ from the true value $y$ of the output is evaluated. Since it is done for every input-output pair individually, this concept is called the **individual error** and can be realised in different ways. The most common way is to define it as the squared error $e = \frac{1}{2}(y - \hat{y})^2$, as it can be shown that this is equivalent to MLE with normally distrubuted noise.

**Empirical Risk Minimization (ERM):** A good model/network will yield for any input a low individual error, i.e. the expected value of $e$ is minimal. However, the mathematical value of $< e >$ cannot be determined, as the probability distribution of $e$ is unknown (otherwise there would be no need to learn). Hence, the empirically expected error (empirical average) over a training set of observations is minimized:

$$E^T = \frac{1}{p}\sum_{\alpha=1}^{p} e_{(\alpha)} \stackrel{!}{=} \min$$

where $p$ represents the amount of input-output pairs in the training set.

**Gradient Descent:** The concept of gradient descent is relatively straight forward. Interpreting $E^T$ as an error landscape over the model parameters $\underline{\mathbf{w}}$, an improvement of the model will be achieved by changing $\underline{\mathbf{w}}$ in the direction opposite of the steepest gradient in the error landscape. Hence, the change in single values of $\underline{\mathbf{w}}$ corresponds to:

$$\Delta w_{i,j}^{\nu',\nu} = -\eta \frac{\partial E^T(\underline{\mathbf{w}})}{\partial w_{i,j}^{\nu',\nu}} = -\eta \frac{1}{p} \sum_{\alpha=1}^{p} \frac{\partial e(\underline{\mathbf{w}})_{(\alpha)}}{\partial w_{i,j}^{\nu',\nu}}$$

The partial derivative of the individual error can be split up via applying the chain rule, leaving one term which depends on the cost/error function and a second term which depends on the model class:

$$\frac{\partial e(\underline{\mathbf{w}})_{(\alpha)}}{\partial w_{i,j}^{\nu',\nu}} = \frac{\partial e(\underline{\mathbf{w}})_{(\alpha)}}{\partial y(\underline{\mathbf{x}}_{(\alpha)}, \underline{\mathbf{w}})} \cdot \frac{\partial y(\underline{\mathbf{x}}_{(\alpha)}, \underline{\mathbf{w}})}{\partial w_{i,j}^{\nu',\nu}}$$

For the first (cost-function dependent) term, the partial derivative evaluates to $\hat{y} - y$ (note that $\hat{y} = y(\underline{\mathbf{x}}_{(\alpha)}, \underline{\mathbf{w}})$). The second gradient has to be evaluated depending on the model type.

**Backpropagation:** Efficient way of calculating the gradients in the gradient descent approach. The error is propagated backwards from the output of the model back to the values of the connection weights. Again the chain rule is applied to compose the gradient into sub terms.

**Online vs. batch learning (gradient descent):** In online learning, the weights of an MLP are updated after each sample. The samples are chosen randomly, which introduces some stochastic influence. Therefore, online learning is less affected by the danger of ending up in a local optimum of the error/cost-landscape. On the other hand, there is no formal proof of convergence upto now, although in practise, the algorithm robustly converges to 'good' optima.

In batch learning, all samples are considered for each update of the weights. Only applicable in stationary environments. For batch learning, there are a couple of improvements in the gradient descent method. Using an **impulse term** (smoothing by running average) or an adaptive **step size** are two examples.

**Conjugacy:** Also termed $A$-orthogonal and describes orthogonality between two vectors, rotated by the matrix $A$. If

$$\underline{\mathbf{u}}^T A \underline{\mathbf{v}} = 0$$

the two vectors are said to be conjugate.

**Conjugate Gradient Method:** A gradient descent method with adaptive stepsize and impulse term. The first stepsize is calculated by finding the minimum of the error-landscape on a line (line search) in the direction of the negative gradient. Now, instead of repeating this procedure in a direction which is orthogonal to the previous direction (and thus is the current negative gradient) - which would be steepest descent with optimal stepsize - the direction which is conjugate to the previous direction is chosen for line search.

$$\underline{\mathbf{d}}(t+1) = -\frac{\partial E}{\partial \underline{\mathbf{w}}}|_{\underline{\mathbf{w}}(t+1)} + \beta_t \underline{\mathbf{d}}(t)$$

This procedure is repeated iteratively, such that in each step, one of the conjugate search directions is processed and will not be searched again, since the minimum can be expressed in a basis of conjugate vectors. Hence, it is guaranteed that CG converges in $n$ steps.

CG is the preferred gradient-based method for unconstrained optimization. For constrained optimization, simple gradient descent like backpropagation is used.

**Overfitting:** Small $E^T$ and large $E^G$.

**Cross Validation:** After having trained a model on a training set, it is necessary to validate its performance on a set of unseen data; i.e. its generalization error $E^G$ is informative about the models predictive power. The **test set method** suggests to divide your whole data set prior to training into a training and vaildation set. After the training is done, the model is tested on the validation set and $E^G$ is easily determined.

Another possibility is **n-fold cross-validation**, which suggests to split your data set into $n$ disjunct subsets. Then you train your model $n$ times; in each step the model is trained on the union of all subsets except the $n$th subset, which is used for validation afterwards. In each step you end up with a generalization error of the model. After all $n$ runs, you average the generalization error to get an estimate of $E^G$.

The latter method includes all data samples for training, which might be desirable if the amount of samples is quite low anyways.

**Bias-Variance Decomposition:** From a frequentists perspective, given a scenario of many datasets of equal size and drawn iid from the same underlying model, the expected loss (or mean square error) over all data sets can be decomposed into a bias and a variance term (and noise).

$$\langle(\hat{y} - y)^2\rangle_{\text{all datasets}} = \underbrace{(\langle\hat{y}\rangle - y)^2}_{\text{bias}^2} + \underbrace{\langle(\hat{y} - \langle\hat{y}\rangle)^2\rangle}_{\text{variance}}$$

The above formular describes the mean square error (or ensemble average); the expected loss additionally entails a noise term.

A high model complexity leads to a high variance and a low bias, i.e. the different models, fitted individually to each dataset vary highly among each other, but the average model outcome comes close to the underlying model. A too rigid model on the hand has a low variance among individually fitted models, but a high bias term, i.e. the average model outcome does not approximate the objective function well. These relations are often referred to as **bias-variance trade-off** and the goal of model selection is to find a good balance in this respect.

However, this relationship is only of theoretical interest, since in a practical set-up, one would rather combine all the single datasets to one large datasets and estimate a model.

**Regularization:** Include a penalty (regularization) term to the error function, which prevents the resluting model from overfitting. At the same time, regularization biases the model to certain model types, depending on the regularization term. Hyperparameter $\lambda$ as regularization parameter has to be estimated additionally; it determines how much the model should be regularized.

$$E^T + \lambda E^R \stackrel{!}{=} \min$$

Famous regularization forms are the $L_2$-norm, which is also called weight decay, as it pushes the absolute values of the estimated weights close to 0. It has the advantage that it has a quadratic form and the resulting new optimization problem can be solved in a closed form.

Another important regularization form is the $L_1$-norm, which is known as the lasso. It prefers sparse models, as it tends to set weight values of the estimated models to zero. The lasso can not be solved in closed form and must be optimized by using lagrange multipliers, e.g. IRLS algorithm.

In general, regularizations known as the $L_q$-norm can be stated as

$$E^R = \frac{1}{2}\sum_{j=1}^{M} |w_j|^q$$

for $q = 2$ yielding weight decay (ridge regression) and $q = 1$ yielding the Lasso.

**(Multi-Class) Classification:** With gradient-based methods, the prediction of class labels is not (really) possible. However, the class probabilities can be calculated by normalizing the raw output of a network to reflect a probability distribution over the possible classes. The observed data must be labeled and coded in binary vectors with one non-zero entry to specify the class of the observation.

Then the KL-divergence between the true class probability distribution and the predicted distribution can be used as the cost function to estimate the model. Eventually, the cross-entropy has to be optimized (minimized)

$$E^G = -\sum_C \int d\underline{\mathbf{x}} P(C|\underline{\mathbf{x}}) P(\underline{\mathbf{x}}) \ln P(C|\underline{\mathbf{x}}; \underline{\mathbf{w}}).$$

Since the true distribution of $P(C, \underline{\mathbf{x}}) = P(C|\underline{\mathbf{x}})P(\underline{\mathbf{x}})$ is not known, the empirical average over a training set is used.

**RBF-networks:** Position RBFs as basis functions in the input domain and use the RBF-values of the inputs instead of the raw input data. Usually, use a 2-step procedure: First fit centroids of RBFs to the given input data (unsupervised - e.g. K-means) and then learn the weights for each basis function to approximate the desired output by applying simply least squares.

Advantages of using localized basis-functions in general, are their local influence and in the case of RBFs their nonlinear transformation of the data. However, they are prone to the curse of dimensionality, as one needs a certain amount of basis functions (lets call it $M$) for each dimension $d$ to fully cover the input space. So, $d^M$ basis functions are required.

Regularization is also possible; for example it is possible to regularize a model in the fourier space by using the high pass filtered signal as a penalty term, since high frequencies in the estimated model correspond to a 'rough' output function of the model. Using this regularization implicitly constrains the model to be smooth. Minimizing such a regularized model results in a change of basis functions; namely, the model is essentially based on the inverse fourier transform of the filter applied in the fourier space.

Not applying the 2-step procedure and simply assigning a RBF with the centroid located at each input data is also possible, but yields most probably a considerably overfitted model. Note the similarity to the parzen-windows, which is just a sum of gaussians located at the data points to estimate a cluster structure (or density).

# 2 Statistical Learning Theory & Support Vector Machines

In contrast to learning by empirical risk minimization (ERM), there is also the possibility of learning by structural risk minimization (SRM). Both approaches can be related to each other via the statistical learning theory. Support Vector Machines (SVM) are the most famous algorithm which is based on the latter.

## 2.1 Statistical Learning Theory

Try to assess how good ERM works; and under which conditions it works well. The ERM procedure should converge to the optimal predictor in the limit of an infinite number of samples. Therfore, conditions which hold for

$$\lim_{p\to\infty} P\left[|R(\underline{\mathbf{w}}_p) - R(\underline{\mathbf{w}}_0)| \geq \eta\right] = 0 \text{ for all } \eta > 0$$

have to be found.

**Key Theorem:** The ERM is strictly consistent for all $\underline{\mathbf{w}} \in \Lambda$ and $P(\underline{\mathbf{z}}) \in \Pi$. Strict consistency is defined as

$$\lim_{p\to\infty} P\left[\left|\inf_{\underline{\mathbf{w}}\in\Lambda_{(c)}} R^{(p)}_{\text{emp}}(\underline{\mathbf{w}}) - \inf_{\underline{\mathbf{w}}\in\Lambda_{(c)}} R(\underline{\mathbf{w}})\right| \geq \eta\right] = 0$$

where $\Lambda_{(c)}$ is the set of all 'bad' predictors with $R(\underline{\mathbf{w}}) \geq c$. Hence, in plain words, the difference between the empirical risk (in the limit of a number of infinite samples) and the true risk is 0 for the best, 'bad' models of $\underline{\mathbf{w}} \in \Lambda_{(c)}$.

For ERM it even holds that the set of 'bad' models expands to the set of all models $\underline{\mathbf{w}} \in \Lambda$. Learning via induction through ERM is thus, closely linked to the convergence of the empirical average $R^{(p)}_{\text{emp}}(\underline{\mathbf{w}})$ to the mathematical expectation $R(\underline{\mathbf{w}})$ in the limit of infinite samples.

From the definition of the key theorem it follows that

$$\lim_{p\to\infty} P\left[R_{\text{emp}}^{(p)}(\underline{\mathbf{w}}_p) \geq R(\underline{\mathbf{w}}_0) + \frac{\epsilon}{2}\right] = 0, \forall\epsilon > 0.$$

If we now assume a set of models

$$\Lambda_{(R(\underline{\mathbf{w}}_0)+\epsilon)} = \{\underline{\mathbf{w}} : R(\underline{\mathbf{w}}) \geq R(\underline{\mathbf{w}}_0) + \epsilon\}$$

we see that

$$\lim_{p\to\infty} P\left[\inf_{\underline{\mathbf{w}}\in\Lambda_{(R(\underline{\mathbf{w}}_0)+\epsilon)}} R_{\text{emp}}^{(p)}(\underline{\mathbf{w}}) \geq R(\underline{\mathbf{w}}_0) + \frac{\epsilon}{2}\right] = 1$$

and hence

$$\lim_{p\to\infty} P\left[\underline{\mathbf{w}}_p \in \Lambda_{(R(\underline{\mathbf{w}}_0)+\epsilon)}\right] = 0.$$

Therefore we can conclude that $R(\underline{\mathbf{w}}_0) \leq R(\underline{\mathbf{w}}_p) \leq R(\underline{\mathbf{w}}_0) + \epsilon$ for $\underline{\mathbf{w}}_p \notin \Lambda_{(R(\underline{\mathbf{w}}_0)+\epsilon)}$.

**Statistics of linear seperability:** The fraction of linearly seperable assignments in a classification task with 2 classes ($p = 0.5$) for a given sample size $N$ corresponds to a cumulative binomial distribution (cumulated over all possible seperations $k$ in a $D$-dimensional space):

$$\Pi = \sum_{k=0}^{D-1} \binom{N-1}{k} \left(\frac{1}{2}\right)^k \left(\frac{1}{2}\right)^{N-1-k}$$

The difference of linearly seperable assignments, when one new data point is added to the training set is proportional to the binomial distribution with respect to the dimension $D$ and the number of samples $N$:

$$\Delta\Pi = \binom{N-1}{D-1} \left(\frac{1}{2}\right)^{D-1} \left(\frac{1}{2}\right)^{N-1-D-1}$$

Since the binomial distribution can be seen as the sum over independent bernoulli trials, for a large sample size $N$, the binomial distribution approximates the normal distribution with $\mu = Np$ and $\sigma^2 = Np(1-p)$.

In the case of two-class classification, a normal distribution with mean $N/2$ emerges. Yielding the result that all $2^N$ label configurations are reliably seperable below $N/D = 2$, i.e. if you don't have at least twice as many data points as dimensions, there's a high probability that all possible label configurations are seperable which means that nothing can be learned by inductive learning. For $N > 2D$ on the other hand, almost none of the label configurations are seperable.

**VC Dimension:** Describes the capacity of a given model(-class) and corresponds to the maximum number of samples which can be classified correctly by the model, no matter how the points are labeled. For example, a model with VC dimension of 3 is capable of seperating any possible (two-class) labeling of the 3 data points, i.e. $2^3$ possible label assignments can be seperated by a model with $d_{VC} = 3$.

The growth function is based on the VC dimension, actually it is just the logarithm of all possible label assignments $2^p$ for $p \leq d_{VC}$ and $d_{VC}$ plus a small term else. Anyways, it provides an upper bound of the test error of the according model(-class) $\Lambda$:

$$G_{(N)}^{\Lambda} \begin{cases} = N\ln 2 & \text{for } N \leq d_{VC} \\ \leq d_{VC}(1 + \ln\frac{N}{d_{VC}}) & \text{for } N > d_{VC} \end{cases}$$

The VC dimension introduces a new way of stating if a classification task is learnable: If $d_{VC}$ is finite, a problem is learnable.

**Summary and Link to SRM:** Note that now, the number of samples is denoted by $p$ again! By using the results of statistical learning theory and the VC dimensions we can state the upper bound on the generalization error for ERM:

$$P\left[\sup_{\underline{\mathbf{w}}\in\Lambda}\left|R(\underline{\mathbf{w}})-R_{\text{emp}}^{(p)}(\underline{\mathbf{w}})\right|>\eta\right]<\underbrace{4\exp\left(G_{(2p)}^{\Lambda}-p\left(\eta-\frac{1}{p}\right)^2\right)}_{=\epsilon}$$

In plain words, with probability of $1-\epsilon$, the relationship of generalization error and empirical error can be stated as

$$R(\underline{\mathbf{w}})<R_{\text{emp}}^{(p)}(\underline{\mathbf{w}})+\underbrace{\left(\frac{G_{(2p)}^{\Lambda}-\ln\frac{\epsilon}{4}}{p}\right)^{\frac{1}{2}}+\frac{1}{p}}_{\eta\text{: complexity term, depending on model class}}$$

In ERM the empirical risk is minimized to keep the right hand side as close as possible to the left hand side. SRM in contrast tries to minimize the complexity term while keeping the empirical risk at an acceptable level.

## 2.2 Support Vector Machines (SVM)

The general principle of SVMs for a classification task can be described in plain words as taking the most ambiguous examples and finding a decision boundary. In contrast, a RBF-network classifier choses a decision boundary based on the prototypes - the most typical examples - of each class.

**Canonical Hyperplane:** In a classification problem, a decision boundary of the form

$$\underline{\mathbf{w}}^T\underline{\mathbf{x}}+b=0$$

is tried to be found. However, it makes no difference to normalize $\underline{\mathbf{w}}$ and $b$ in such a way that

$$\min_{\alpha=1,\ldots,p}|\underline{\mathbf{w}}^T\underline{\mathbf{x}}_\alpha+b|\stackrel{!}{=}1.$$

In other words, in a classification task with targets $t_n\in\{-1,+1\}$, for the data points which are closest to the decision boundary we set

$$t_n(\underline{\mathbf{w}}^T\underline{\mathbf{x}}_n+b)=t_n y(\underline{\mathbf{x}}_n)=1.$$

Then for all datapoints $n=1,\ldots,N$ it holds that $t_n y(\underline{\mathbf{x}}_n)\geq 1$ (assuming that all datapoints are linearly seperable and correctly classified), which is called canonical hyperplane.

**Margin:** The area around the decision boundary in which no data points are located (at least in a linearly seperable problem) and which is spanned by the most ambiguous data points from both classes, i.e. the data points of each class which are closest to the decision boundary.

The decision boundary is placed in such a way that the (orthogonal) distance to the closest data points from each class is the same, i.e. the decision boundray is placed in the middle of the most ambigous points. This distance is called margin and the data points which define the margin are called support vectors. The goal of optimizing SVMs is to maximize the margin while keeping the number of misclassified training samples as low as possible (or even zero in case of non-overlapping classes). The margin of a SVM also imposes an upper bound of the VC-dimension of a model (The larger the margin, the lower is the according VC-dimension and hence the growth function).

The distance of any point to the decision boundary is given by $t_n y(\underline{\mathbf{x}}_n)/||\underline{\mathbf{w}}||$. However, for all support vectors the distance - and hence the margin - is defined as

$$d_w=\frac{1}{||\underline{\mathbf{w}}||}$$

since $t_n y(\underline{\mathbf{x}}_n)=1$ for all support vectors.

**Constrained Optimization via Lagrangian:** To maximize the margin $d_w = \frac{1}{||\underline{\mathbf{w}}||}$ it is equivalent to minimize $||\underline{\mathbf{w}}||^2$ with respect to the earlier defined constraint $t_n y(\underline{\mathbf{x}}_n) \geq 1$. Thus, yielding a Lagrangian of the following form:

$$L(\underline{\mathbf{w}}, b, \underline{\lambda}) = \frac{1}{2}||\underline{\mathbf{w}}||^2 - \sum_{n=1}^{N} \lambda_n \{t_n(\underline{\mathbf{w}}^T \underline{\mathbf{x}}_n + b) - 1\}$$

The factor of 0.5 is introduced for convenience, as we have to calculate the derivatives of $L(\underline{\mathbf{w}}, b, \underline{\lambda})$ w.r.t $\underline{\mathbf{w}}$ and $b$. Re-inserting the results of these derivations then yields the dual representation of the problem, only depending on the Lagrange multipliers $\lambda_n$ and the dot-product of different data points (here's where the kernel-trick can be applied):

$$\tilde{L}(\underline{\lambda}) = \sum_{n=1}^{N} \lambda_n - \frac{1}{2} \sum_{n,m=1}^{N} \lambda_n \lambda_m t_n t_m \underline{\mathbf{x}}_n^T \underline{\mathbf{x}}_m$$

Optimizing this dual representation under the following constraints

$$\lambda_n \geq 0, n = 1, \ldots, N \qquad\qquad \sum_{n=1}^{N} \lambda_n t_n = 0$$

gives the values for all the Lagrange multipliers. This so called dual optimization problem is often solved, using the sequential minimal optimizations (SMO) method. Note that $\lambda_n > 0$ only holds for the support vectors and thus, if one wants to predict the class labels of new data points and expresses $y(\underline{\mathbf{x}})$ in terms of the Lagrange multipliers, it is only necessary to sum over all support vectors:

$$y(\underline{\mathbf{x}}) = \sum_{n \in \text{SuppVec}} \lambda_n t_n \underline{\mathbf{x}}^T \underline{\mathbf{x}}_n + b$$

Taking the sign of the result, gives the class label. However, the offset parameter $b$ still has to be determined. This is done by taking the constraint $t_n y(\underline{\mathbf{x}}_n) \geq 1$ and inserting the expression of $\underline{\mathbf{w}}$ which is fomulated in terms of $\lambda_n$. This constraint is active (=) only for support vectors and solving it for $b$ as an average over all support vectors gives a numerical stable solution:

$$b = \frac{1}{N_{SV}} \sum_{n \in SV} \left( t_n - \sum_{m \in SV} \lambda_m t_m \underline{\mathbf{x}}_n^T \underline{\mathbf{x}}_m \right)$$

**Comments on SVMs:**

- SVMs are only based on dot products. Therefore it is possible to replace the dot products by any valid Kernel function, which implicitly represents a dot product in some other metric feature space (every positive definite Kernel is valid says Mercer's Theorem). However, no explicit transformation into that feature space is necessary (**Kernel Trick**).
- A classification model operating directly in feature space has as many parameters as the feature space has dimensions. Transforming the problem into a dual representation, the model only requires as many parameters as there are data points. Hence, SVMs can work for feature spaces with infinite dimension via solving their dual representation.
- SVMs are based on Karush-Kuhn-Tucker (KKT) conditions: For each constraint, the Lagrange multiplier and the constraint must be larger or equal to zero and the product of multiplier and constraint must be exactly zero for each data point. Thus, either a constraint is active (= 0) or the Lagrange multiplier is 0.
- SVMs are also called sparse-kernel machines. Because their predictions are only based on a sparse subest (support vectors) of all datapoints.

**C-SVM ($\nu$-SVM):** Note that up to now, we assumed that the SVM classifies every training data point correctly via imposing the constraint

$$t_n y(\underline{\mathbf{x}}_n) \geq 1, n = 1, \ldots, N.$$

Now, we allow for errors in the classification and introduce so called slack variables $\xi_n \geq 0$ for each training data point. If the data point is lying on or inside the correct margin boundary, $\xi_n = 0$, else, $\xi_n = |t_n - y(\underline{\mathbf{x}}_n)|$.

This imposes a slightly different constraint

$$t_n y(\underline{\mathbf{x}}_n) \geq 1 - \xi_n, n = 1, \ldots, N$$

as well as a minimization problem which includes a penalty term for large errors:

$$\min_{\underline{\mathbf{w}}, b, \underline{\xi}} \frac{1}{2}||\underline{\mathbf{w}}||^2 + C \sum_{n=1}^{N} \xi_n.$$

The corresponding Lagrangian is given by

$$L(\underline{\mathbf{w}}, b, \underline{\xi}, \underline{\lambda}, \underline{\mu}) = \frac{1}{2}||\underline{\mathbf{w}}||^2 + C \sum_{n=1}^{N} \xi_n - \sum_{n=1}^{N} \lambda_n \{t_n(\underline{\mathbf{w}}^T \underline{\mathbf{x}}_n + b) - 1 + \xi_n\} - \sum_{n=1}^{N} \mu_n \xi_n$$

along with the usual KKT conditions

$$\begin{aligned} \lambda_n &\geq 0 \\ t_n y(\underline{\mathbf{x}}_n) - 1 + \xi_n &\geq 0 \\ \lambda_n(t_n y(\underline{\mathbf{x}}_n) - 1 + \xi_n) &= 0 \\ \mu_n &\geq 0 \\ \xi_n &\geq 0 \\ \mu_n \xi_n &= 0. \end{aligned}$$

Taking the derivatives w.r.t $\underline{\mathbf{w}}$, $b$ and $\underline{\xi}$, solving them and re-inserting them into the Lagrangian, yields the dual representation of the problem, which is in fact identical to the case of linearly seperable data. There is only one difference in the constraints and now it is necessary that $0 \leq \lambda_n \leq C$.

The predictive model and the offset parameter can be determined as described in the linearly separable case. Additionally, it is now possible to interpret the datapoints which have a non-zero Lagrangian multiplier $\lambda_n > 0$ in some more detail. Again, all of those points are called support vectors and for $\lambda_n < C$, they lie again on the margin boundary. However, support vectors with $\lambda_n = C$ can lie within the margin and can be either classified correctly ($\xi_n \leq 1$) or incorrectly ($\xi_n > 1$).

**Gram Matrix:** The Kernel matrix holding the values of applying the kernel function to all possible combinations of datapoints:

$$K_{\alpha\beta} = k(\underline{\mathbf{x}}_\alpha, \underline{\mathbf{x}}_\beta)$$

**SMO:** The sequential minimal optimization method is the preferred method to train SVMs. Roughly, it operates on pairwise data in each step and optimizes 2 Lagrangian multipliers at a time. It loops over all $\lambda_n$ which violate the KKT conditions and selects a corresponding second Lagrange multiplier which promises to make a large step towards the optimum.

**Dyadic Data:** Data in matrix form which describes the relationship between certain 'column' objects, which should be described (feature vectors) and 'row' objects which are used for their description (features). A special case is pairwise data, in which the objects, which should be described and the objects which are used to describe are the same (symmetric square-matrix). This kind of data arises naturally as the consequence of relative measurements and can be interpreted as the result of a similarity measure (dot product) in a higher dimensional feature space.

**P-SVM (Potential Support Vector Machnine):** An extension of classical SVMs which works on dyadic data (see dyadic data) and uses 'support features' as basis for decision boundary.

The motivation for P-SVM arose from the fact that not all kind of data can be naturally represented in form of vectors (cf. dyadic data), that the resulting decision boundary of a SVM is not scale-invariant (how to scale data before training) and that only positive definite Gram matrices are valid for SVMs.

It is proposed to scale the data in such a way that the margin of the SVM is kept constant, while the radius of a sphere which can include all data points $R$ is minimized. The new radius of the scaled data is then termed $\tilde{R}$. Note that the VC-dimensions are bound by the equation $d_{VC} \leq \min(\lceil \frac{R^2}{d_w^2} \rceil, N) + 1$.

The new objective function is then formulated to minimize this upper bound with the new radius $\tilde{R}$:

$$\frac{\tilde{R}^2}{d_w^2} = \tilde{R}^2 ||\underline{\mathbf{w}}||^2 \leq \max_n (\underline{\mathbf{w}}^T \underline{\mathbf{x}}_n)^2 \leq \sum_n (\underline{\mathbf{w}}^T \underline{\mathbf{x}}_n)^2 = ||X^T \underline{\mathbf{w}}||^2$$

Note that if $XX^T = I$, i.e. if the data has already been whitened/sphered (zero mean, unit variance), we get the classical objective function.

As a constraint, we use the squared error function along a selected feature. Assuming that the Gram matrix (dyadic data) already contains the measurements of each feature ($K_{\alpha\beta} = (\underline{\mathbf{z}}_\alpha)^T \underline{\mathbf{x}}_\beta$), where $\underline{\mathbf{z}}_\alpha$ denotes one selected feature, we get:

$$\begin{aligned} (\underline{\mathbf{z}}_\beta)^T \frac{\partial R_{\text{emp}}^{(p)}}{\partial \underline{\mathbf{w}}} &\stackrel{!}{=} 0 \\ \Leftrightarrow \frac{1}{p} \sum_{\alpha=1}^{p} K_{\beta\alpha} (\underline{\mathbf{w}}^T \underline{\mathbf{x}}_n + b - t_\alpha) &\stackrel{!}{=} 0 \\ \Leftrightarrow K^T (X^T \underline{\mathbf{w}} - \underline{\mathbf{t}}) &\stackrel{!}{=} 0 \end{aligned}$$

Hence the P-SVM primal problem is stated as:

$$\begin{aligned} \min_{\underline{\mathbf{w}}} \quad & \frac{1}{2} ||X^T \underline{\mathbf{w}}||^2 \\ \text{subject to} \quad & K^T (X^T \underline{\mathbf{w}} - \underline{\mathbf{t}}) = 0 \end{aligned}$$

However, as soon as $K$ has a rank $\geq p$, i.e. at least as many features as data points, the constraint is always true and overfitting occurs.

Therefore, it is necessary to solve a regularized version of the above stated primal problem, similar to that one in the classical SVM derivation:

$$\min_{\underline{\mathbf{w}}, \underline{\xi}^+, \underline{\xi}^-} \quad \frac{1}{2} ||X^T \underline{\mathbf{w}}||^2 + C \underline{\mathbf{1}}^T (\underline{\xi}^+ + \underline{\xi}^-)$$

subject to some constraints involving the slack variables being positive and punishing large errors.

In general, P-SVM can also be used for feature selection as only support features influence the positioning of the decision boundary.

## 2.3 Bayesian Inference

First order logic fails to establish rule-based systems which can account for all possible situation, because outcomes are often governed by stochastic mechanisms and only incomplete observations are available for the creation of a rule base etc. Therefore, it is handy to treat the outcome of certain event as probabilities, or rather state the 'degree of belief' in the particular event.

**Product Rule:**

$$P(X, Y) = P(X|Y)P(Y) = P(Y|X)P(X)$$

**Sum Rule:**

$$P(X) = \sum_Y P(X,Y) = \sum_Y P(X|Y)P(Y)$$

**Probabilistic Inference Using Joint Probabilities:** Store degrees of belief for every atomic event in the knowledgebase: $P(x, y, \dots)$. Use product rule to infer from given evidence (observation) what the probability for certain events are.

However, for a large set of random variables, the knowledgebase explodes. And to calculate the infered probability, it is necessary to marginalize over all non-observed probabilities. Very expensive calculations and storage.

**Conditional Independence:** $X$ and $Y$ are conditionally independent, given $Z$:

$$P(X,Y|Z) = P(X|Z)P(Y|Z)$$

Store 'right' conditional probabilities rather than probabilities of all atomic events. Reduces the size of the knowledgebase dramatically.

If you can visualize the 'world' in which inferences should be made as a DAG, just store the conditional probabilities which arise from the graph and you're good to go.

**Directed Acyclic Graph (DAG):** Set of random variables are the nodes, and directed links between nodes indicate causal relationships. The nodes are then annotated with conditional probabilities.

$$P(x_1, \dots, x_n) = \prod_{i=1}^{n} P(x_i|\text{parents}(x_i))$$

given that the nodes are well-ordered. The conditional independence is encoded in the graph structure:

- A node is conditionally independent of its non-descendants, given its parents
- A node is conditionally independent of all other nodes in the graph, given its parents, childrens and children's parents (uncles). This is called a 'Markov Blanket'.

A DAG is an efficient representation of knowledge about causal relationships, including in its topology information such as conditional independence. Additionally, annotation of the nodes with probability tables of pdfs gives convenient information.

However, DAGs are not good representations for inference.

**Complete Graph:** A graph in which each node is connected with each other note, i.e. a fully connected graph.

**Proper decomposition:** $A, B, C$ are subsets of nodes of a given graph $G$. It is said to be a proper decomposition, if e.g. $C$ is a complete graph which seperates $B$ and $C$. A seperator is a node which is a hub between to other graphs, i.e. all connections go through the seperator.

**Decomposable Undirected Graphs:** A grapgh is a decomposable graph if it is a complete graph, or if there existst a proper decomposition in such a way that both subgraphs are both proper decmpositions themselves.

This leads to a decomposition into maximally complete subgraphs (**cliques**) seperated by **seperators**.

A decomposistion of a graph into its cliques and seperators, corresponds to the statement that 2 cliques are conditionally independent from each other, given their seperator.

The probability distribution of the whole graph can then be stated as a relation of the factorized probabilities over the cliques and graphs

$$P(\underline{\mathbf{x}}) = \frac{\prod_{\text{cliques C}} P_C(\underline{\mathbf{x}}_C)}{\prod_{\text{seperators S}} P_S(\underline{\mathbf{x}}_S)}$$

where $P_C$ and $P_S$ are the marginal probabilities over the clique and seperator variables, respectively. In general, this relation also holds/still is proportional if you don't have the exact pdfs, but only potentials.

**Moralized Graph:** All parents in a directed graph are connected as well and directed graphs are replaced by undirected graphs.

**Chordal Graph:** Introduce chords / shortcuts into cycles of undirected links which are of length 4 or larger (only triangle cycles in graph - only cliques). A chordal graph is not unique and it is NP hard to construct a chordal graph with minimimal clique size.

However, a chordal graph is always decomposable and therefore the basis for Bayesian inferences. Its decomposition is put into the form of a junction tree and then via message passing, the influence of observed evidence is propagated through the whole network to make some inferences.

**Junction Tree:** The junction tree is constructed by taking the cliques and seperators of the chordal graph, and thus turns the graph into a chain. The clique potentials are then initialized with a matching probability of the according directed graph, i.e. a clique involving nodes $A, B, C$ is initialized with a probability which includes all of the nodes. Note that each probability which is known from the database is only used for one clique. Obviously, such an initialization is not unique!

The seperators can be initialized by 1, as they are just normalization factors and the cliques are initialized with probabilities per se.

**Message Passing:** To do some inference, now it is necessary to insert some evidence into the juntion tree and propagate it through the whole chain. It is basically a three step procedure:

1. Select a root node and request information from all other nodes
2. Pass the information from the other side of the chain to the root node
3. When the information reached the root node, integrate all the given evidence and pass it back

If the root node is selected in such a way that it is at one end of the junction tree and entails the event which probability is queried, there is actually no need for the third step.

**Bayes Theorem:**

$$\underbrace{p(M|E)}_{\text{posterior}} = \frac{\overbrace{p(E|M)}^{\text{likelihood}}\overbrace{p(M)}^{\text{prior}}}{\underbrace{p(E)}_{\text{normalization}}}$$

A commomn way to maximize the posterior probability, is to maximize the likelihood (MLE). Then, no prior knowledge is used or known for the estimation of the model. Inclusion of a prior leads to MAP estimation, which can also be interpreted as MLE with a regularization term.

Another way for selecting a prior is using a maximum entropy method to find least informative prior beliefs (exponential function).

**Bayesian Reression:** Take datapoints point by point. Start with a normally ditributed prior and calculate the posterior of the model. In the next step take the 'old' posterior as the 'new' prior and calculate a 'new' posterior based on the new datapoint.

**Conjugate Prior:** The posterior distribution will have the same functional form as the prior, if it is a conjugate prior.

**Predcitive Distribution:** Eventually, a procedure like Bayesian regression will give you not only a single model, as MLE would do, but gives you a (posterior) distribution of the model parameters based on all observations. This distribution can then be used to predict the model outcome for unseen datapoints. Since the outcome is a distribution again, its confidence bounds can also be calculated.

The predictive distribution is then a convolution of the likelihood of a new observation with the posterior based on all previously seen observations $D$:

$$P(t|\alpha, \beta, D) = \int P(t|\underline{\mathbf{w}}, \beta) P(\underline{\mathbf{w}}|\alpha, \beta, D) d\underline{\mathbf{w}}$$

In the limit of infinite observations, the predictive distribution will have zero uncertainty, i.e. with growing amount of observations, the predictions get better and better.

The predictive distribution of an MLE solution is simply the outcome of the assumed model with the maximum-likelihood estimators plugged in.

**Bayesian Model Comparison:** Instead of using the above explained method to get a distribution over probable model parameters, it can also be used to compute a probability distribution over different model classes. With this information, a kind of commitee of different models could be set up and they all would contribute to the overall outcome. This would correspond to averaging the predictive distributions of different model classes (weighted by their posterior distributions) for a given novel input:

$$p(t|\underline{\mathbf{x}}, D) = \sum_{i=1}^{L} p(t|\underline{\mathbf{x}}, M_i, D) p(M_i|D)$$

If none of the model-classes is preferred over another, the normalized marginal likelihood (or evidence) of the model class can be used as posterior as it corresponds to the posterior with uniform prior:

$$p(D|M_i) = \int p(D|\underline{\mathbf{w}}, M_i) p(\underline{\mathbf{w}}|M_i) d\underline{\mathbf{w}}$$

in wich the parameters have been marginalized out.

# 3 Unsupervised Learning

In contrast to supervised learning techniques, in unsupervised learning the given data is not labeled, i.e. in general, there is no input-output association to be learnt, but only some (input) observations are given. The task is to find patterns (clustering) in these observations or estimate the underlying distribution (density estimation) of the data solely by looking at the given observations.

## 3.1 Probability Density Estimation

In a density estimation, the task is to find a model which describes the underlying density of a given set of observations. This can be done by a parametric model (e.g. MLP) in finding an appropriate set of parameters $\underline{\mathbf{w}}$ for a predetermined distribution class, or by non-parametric methods which don't assume anything about the ditribution in advance.

**Histogram:** Simply partition the input space into bins and count the datapoints within each bin. Dividing the counts by the binwidth and the total number of datapoints ensures that you end up with a pdf.

$$p_i = \frac{n_i}{N\Delta_i}$$

The problem with this approach is that you will get discontinuities in your estimated density and that the positioning of the bins influences the rsulting pdf as well.

**Kernel Density Estimation (Parzen Estimator):** Here, you center a kernel over each datapoint and sum them up to get a density. Or in other words, you slide a kernel (window) function along your data space and convolve your kernel with the data. This technique eliminates the problem of the arbitrary positioning of the bins and also allows for continous kernel-functions, yielding continous pdfs. The general formlular for a kernel density estimation looks like:

$$p(\underline{\mathbf{x}}) = \frac{1}{N}\frac{1}{h^D}\sum_{n=1}^{N} k\left(\frac{\underline{\mathbf{x}} - \underline{\mathbf{x}}_n}{h}\right)$$

where $h$ controls the kernel-width and $h^D$ accordingly represents the volume in which the datapoints are 'counted'. A kernel-function can be any non-negative function which integrates to 1 over $\underline{\mathbf{x}}$, e.g. a gaussian.

For this kind of density estimation, there is no training necessary. Only the hyperparameter $h$ hast to be adjusted to get a good result. The problem with this is that $h$ will be the same for the whole input space and so in regions with fine differences in the underlying distribution a smaller $h$ is necessary than in other regions, to capture all the information of the density.

**Parametrized Density Estimation:** Chose a parametrized family of pdfs and try to create a generative model which is as close as possible to the true unknown distribution.

Note that in contrast to regression or classification, here an unconditional probability $p(\underline{\mathbf{x}})$ is estimated instead of $p(t|\underline{\mathbf{x}})$.

One ansatz to find the parameters is to minimize the KL-divergence

$$D_{KL} = \int P(\underline{\mathbf{x}}) \ln \frac{P(\underline{\mathbf{x}})}{\hat{P}(\underline{\mathbf{x}}; \Theta)} d\underline{\mathbf{x}}$$

between the true distribution and the parametrized distribution family with respect to $\Theta$. This will brake down to minimizing the cross-entropy

$$\hat{\Theta} = \arg\min_{\Theta} \left( - \int P(\underline{\mathbf{x}}) \ln \hat{P}(\underline{\mathbf{x}}; \Theta) d\underline{\mathbf{x}} \right).$$

which represents the 'generalization cost' $E^G$. However, since the true distribution $P(\underline{\mathbf{x}})$ is not known, this is not feasible and an estimation to $E^G$ is minimized, i.e. the empirical average, which is also known as training cost:

$$E^T = -\frac{1}{N}\sum_{n=1}^{N} \ln \hat{P}(\underline{\mathbf{x}}_n; \Theta).$$

Optimization and validation is done as discussed before in the chapter on supervised learning (e.g. gradient based methods + cross-validation). Note that this ansatz yields the exact same fomulation as the MLE.

**Maximum Likelihood Estimation (MLE):** One tries to find a model which has the highest probability given a set of iid data points. Since the data is assumed to be iid, the joint probability factorizes and can be assessed. However, this factorization will yield in general very small probability values (wich might lead to numerical errors), so the logarithm of this probability is taken instead.

After having chosen a parametrized model familiy, the log-likelihood of the observed data can be maximized (or the negative log-likelihood is minimized) with respect to the parameter vector $\Theta$. Note that the parameter vector is different for this kind of density estimation than for the earlier discussed regression and classification cases, in which the input (possibly transformed by basis functions) was weighted by the parameters (weight vector). Here, the parameter vector contains the estimated moments ($\Theta = (\mu, \sigma^2)^T$ for gaussian data) of the distribution.

Assuming a gaussian distribution over the observed samples, the sample mean and sample variance will be the MLE of the observations.

**Quality of Estimators:** Estimators in the given context are random variables themselves, as they are estimated from a set of random variables. Therefore, their quality can be assessed by calculating its bias and variance:

$$\text{bias}(\hat{\theta}) = E[\hat{\theta}] - \theta \qquad \text{var}(\hat{\theta}) = E[(\hat{\theta} - E[\theta])^2]$$

Note that $\hat{\theta}$ represents a single estimated parameter out of the parameter vector $\hat{\Theta}$ and $\theta$ corresponds to its true value. For the ML estimates of the mean and the variance, it holds that $\mu_{ML}$ is unbiased whereas $\sigma^2_{ML}$ is not. However, it can be shown that the MLE is asymptotically efficient (unbiased & approaches Cramer-Rao bound (inverse of Fisher Information Matrix)).

## 3.2 Projection Methods

In general projection methods are useful for visualisation and dimension reduction for high-dimensional data. Projecting the data onto 'interesting' (or discriminant) features, may help as a preprocessing step for later classification.

**Principal Component Analysis (PCA):** Basically, applying PCA is nothing but solving the eigenvalue problem for the covariance matrix of zero-centered data:

$$C\underline{\mathbf{e}}_a = \lambda \underline{\mathbf{e}}_a$$

This can be derived by maximizing the variance with respect to any 'complex' direction $\underline{\mathbf{e}}_a$ of the data, subject to the constraint that the length of the vector $\underline{\mathbf{e}}_a$ should by 1.

Then, the eigenvectors, corresponding to the largest eigenvalues, point into the direction of the largest variance in the data. The eigenvalues themselves correspond to the variance of the data in that direction. Obviously, this makes PCA vulnerable to outliers in the data.

The eigenvectors of the covariance matrix are called its principal components and can be used to project the data onto a subspace which is spanned by the PCs with largest variance. When the PCs are sorted in an descending order with respect to their eigenvalues, then a scree plot (eigenvalue vs. no. of PC) can be used to select a subset of reasonable PCs for projection. Note that the first PC points in the direction of largest variance of the 'full' space, but the second PC points in the direction of largest variance in the space spanned by all other PCs except the first one etc.

The projection of datapoints onto a set of eigenvectors of the covariance matrix

$$\underline{\mathbf{x}} = \underbrace{a_1}_{\underline{\mathbf{e}}_1^T\underline{\mathbf{x}}}\underline{\mathbf{e}}_1 + \underbrace{a_2}_{\underline{\mathbf{e}}_2^T\underline{\mathbf{x}}}\underline{\mathbf{e}}_2 + \cdots + \underbrace{a_N}_{\underline{\mathbf{e}}_N^T\underline{\mathbf{x}}}\underline{\mathbf{e}}_N$$

can then be computed for $N$ PCs. If $N$ equals the number of dimensions of the data, then the above stated projection is simply a decomposition of the original data (and the orignal 'full' data is preserved).

A projection onto the first $M$ PCs of the data exhibits the minimal approximation error to the original data with respect to all possible projections onto an $M$-dimensional subspace.

Some facts on PCA:

- $C$ is covariance matrix of zero-centered data: $C_{ij} = \frac{1}{N}\sum_{n=1}^{N} x_i^{(n)} x_j^{(n)}$
- zero-centering is necessary to get a basis of PCs which minimize the mean square error of the approximation in a subspace. If no zero-centering is done, first PC might be something like the mean of data.
- $C$ is symmetric, real-valued and positive-semidefinite ($\forall i : \lambda_i \geq 0$, as they are covariances)
- PCs form an orthonormal basis (which follows from the previous point)
- PCs are uncorrelated
- whitening/sphering: Project (zero-meaned) data onto PCs and scale it with squareroot of inverse eigennvalues

**Hebbian Learning:** Based on Hebb's rule: *What fires together wires together.*

$$\Delta w_j = \epsilon \cdot y x_j$$

A learning rule for connectionist neurons where $y = \underline{\mathbf{w}}^T \underline{\mathbf{x}}$, $\epsilon$ is the learning rate/step and $\underline{\mathbf{x}}$ is the input vector. The correlation-basis of this method can be easily seen by considering the product between input and output of the neuron. Eventually, the learned weight vector will converge to the first principal component of the data. Hence, hebbian learning can be seen as something like an on-line version of PCA, which you can watch converging to the first PC.

However, nothing prevents the weight vector to grow infinitely in length. Therefore, Oja suggested a rule for normalizing the vector length to 1, by introducing a 'forgetting term':

$$\Delta w_j = \epsilon(x_j y - y^2 w_j)$$

**Kernel PCA:** Standard PCA works only for linearly decorrelatable data. However, it would be nice to find the directions of non-linear manifolds which correspond to the directio of largest variance in non-linear data.

Since PCA is based on solving the eigenvalue problem for the covariance matrix of zero-centered data, it is expressed only terms of a similarity measure (dot product) between data points. Moreover, the eigenvectors can also be expressed as projections onto the data point. Therefore, it is possible to apply the Kernel trick and replace the dot product by a valid kernel function (remember: every positive definte kernel corresponds to a dot product in some metric feature space). Introducing the according Gram matrix

$$K_{\alpha\beta} = k(\underline{\mathbf{x}}_\alpha, \underline{\mathbf{x}}_\beta)$$

and solving the eigenvalue problem for the zero-centered Gram matrix (note that the zero centering has to be done on the Gram matrix and not on the riginal data)

$$K\underline{\mathbf{e}}_k = p\lambda_k \underline{\mathbf{e}}_k$$

basically summarizes the technique of kernel PCA. After having extracted the PCs / eigenvectors of the Gram matrix, they still have to be normalized to unit length:

$$\underline{\mathbf{e}}_k^{\text{normalized}} = \frac{1}{\sqrt{p}\sqrt{\lambda_k}||\underline{\mathbf{e}}_k||}\underline{\mathbf{e}}_k.$$

Kernel PCA is basically normal PCA in the feature space. But, number of PCs can now exceed number of dimensions in original space and an expansion of PCs into data points is not sparse.

## 3.3 Source Seperation Techniques

Techniques which try to uncover the source-signals of an observed, mixed signal. Source seperation Techniques differ in what kind of prior knowledge they exploit. It is assumed that the observed signal is the result of an unknwon mixing matrix being multiplied with the sources:

$$\underbrace{\underline{\mathbf{x}}}_{\text{observations}} = \underbrace{\underline{\mathbf{A}}}_{\text{mixing matrix}} \underbrace{\underline{\mathbf{s}}}_{\text{sources}}$$

The aim is to find an unmixing matrix with $\underline{\mathbf{W}} = \underline{\mathbf{A}}^{-1}$ from which the sources can be recovered:

$$\underbrace{\hat{\underline{\mathbf{s}}}}_{\text{estimated sources}} = \underline{\mathbf{W}}\underline{\mathbf{x}}.$$

**Independent Component Analysis (ICA):** ICA makes the assumption that the sources are statistically independent, i.e. that their density factorize:

$$P(\underline{\hat{\mathbf{s}}}) = \prod_{i=1}^{N} P(\hat{s}_i).$$

Note that PCA only decorrelates and hence is of no use here. However, also ICA bears some limits in the recovery of sources:

- permutation ambiguity - order of sources is not revealed
- amplitude of sources cannot be revealed
- ICA can only work for non-gaussian source distributions as the possibility of factorising gaussians is infinite
- the number of observations must be at least as large as the number of sources $\#x_i \geq \#s_i$

There are different ways of solving this problem: It can be solved by minimizing the mutual Information/distance between the density of all sources and the factorization of the desnities of the single sources ($D_{KL}(P(\underline{\mathbf{x}})||\prod_i P(x_i))$), which is called the **infomax** principle. Another possibility is given by maximizing the non-gaussanity (**kurtosis**) of the source-distributions. A rather new method requires the joint **diagonalization** of time shifted cross correlation matrices.

In general, it is worth mentioning that PCA is useful to be applied before doing ICA for sphering and dimension reduction.

**Infomax (Information Maximization):** Under the above stated assumption that the unknown source disributions are independent and the joint distribution will factorize, a minimization of the distance between the joint distribution and its factorization yields the ansatz for the infomax principle:

$$D_{KL}(P(\underline{\hat{\mathbf{s}}})||\prod_i P(\hat{s_i})) = \int P(\underline{\hat{\mathbf{s}}}) \ln \frac{P(\underline{\hat{\mathbf{s}}})}{\prod_{i=1}^{N} P(\hat{s_i})} d\underline{\hat{\mathbf{s}}} \stackrel{!}{=} \min$$

which can as well be stated as the maximization of mutual information for a network with an input $\underline{\mathbf{x}}$ and an output $\underline{\mathbf{u}}$:

$$I(\underline{\mathbf{u}}, \underline{\mathbf{x}}) = \underbrace{H(\underline{\mathbf{u}})}_{\text{entropy}} - \underbrace{H(\underline{\mathbf{u}}|\underline{\mathbf{x}})}_{\text{conditional entropy/noise entropy}} \stackrel{!}{=} \max$$

Since the noise is assumed to be constant, eventually, the technique breaks down to maximizing the entropy of the network output.

In this network representation, where $\underline{\mathbf{u}} = f(\underline{\mathbf{W}}\underline{\mathbf{x}})$ with $f(\cdot)$ being a differentiable and invertable function (e.g. tanh, logisticfunction) the optimal weights can be found using gradient ascend methods like in empirical risk minimization.

To get there, a density transformation from $\underline{\mathbf{u}}$ to $\underline{\mathbf{x}}$ has to be done for which it holds that

$$P(\underline{\mathbf{u}})d\underline{\mathbf{u}} = P(\underline{\mathbf{x}})d\underline{\mathbf{x}}$$

due to conservation of probabilities. The transformed density $P(\underline{\mathbf{u}}) = |\frac{d\underline{\mathbf{x}}}{d\underline{\mathbf{u}}}|P(\underline{\mathbf{x}})$ is then substituted into the entropy of the network output and maximized with respect to $\underline{\mathbf{W}}$:

$$\begin{aligned} H(\underline{\mathbf{u}}) &= -\int d\underline{\mathbf{u}} P(\underline{\mathbf{u}}) \ln P(\underline{\mathbf{u}}) \\ &= -\int d\underline{\mathbf{x}} P(\underline{\mathbf{x}}) \ln \left( \left| \frac{d\underline{\mathbf{x}}}{d\underline{\mathbf{u}}} \right| P(\underline{\mathbf{x}}) \right) \\ &= -\int d\underline{\mathbf{x}} P(\underline{\mathbf{x}}) \ln P(\underline{\mathbf{x}}) + \int d\underline{\mathbf{x}} P(\underline{\mathbf{x}}) \ln \left| \frac{d\underline{\mathbf{u}}}{d\underline{\mathbf{x}}} \right| \stackrel{!}{=} \max_{\underline{\mathbf{W}}} \end{aligned}$$

where $\underline{\mathbf{W}}$ is hidden within $\underline{\mathbf{u}}$ and the mathematical expectation is again replaced by the empirical average.

Note that using vanilla gradient ascent methods will involve a matrix inversion during every weight update step, which is computationally very expensive. Using the natural gradient (steepest ascent under normalized step size) instead avoids this problem.

Some comments on ICA:

- Undetermined source amplitudes might lead to convergence problems. Solution: e.g. $w_{ii} = 1$ and don't change
- ICA is relatively robust to wrong choices of $f(\cdot)$, but probability density with one maximum is very likely
- true (independent) source signals are always a fixpoint of natural gradient ascent, independent of $f(\cdot)$

**Diagonalization of Crosscorrelation Matrices:** Here the seperation idea is to find an unmixing matrix $\underline{\mathbf{W}}$ in such a way that all time-shifted cross-correlation functions (between the observations $\underline{\mathbf{x}}(t)$) vanish. This is a valid assumption, as statistical independence implies uncorrelation among the signals.

The eigenvalue problem of the form

$$\underline{\mathbf{C}}_u^{(\tau)} \underline{\mathbf{e}}_k = \lambda_k \underline{\mathbf{e}}_k \text{ with } \left[\underline{\mathbf{C}}_u^{(\tau)}\right]_{ij} = \langle u_{i(t)} u_{j(t-\tau)} \rangle_t$$

has to be solved. In this case by diagonalization of $\underline{\mathbf{C}}_u^{(\tau)}$ for a set of $K$ shifts simultaneously.

However, in real world examples, this is only approximately possible (since data is only approximately independent) and therefore a suitable cost function is necessary to be minimized:

$$E^T = \sum_\tau \alpha_\tau \sum_{i \neq j} (\underline{\mathbf{W}} \underline{\mathbf{C}}_x^{(\tau)} \underline{\mathbf{W}}^T)_{ij}^2$$

where

$$\left[\underline{\mathbf{C}}_x^{(\tau)}\right]_{ij} = \frac{1}{T} \sum_t x_{i(t)} x_{j(t-\tau)}.$$

**Maximization of Kurtosis:** The kurtosis is a measure for non-gaussanity. If the kurtosis of a random variable is 0, the variable is said to be gaussian. If the kurtosis $> 0$, it is super-gaussian (peak, long-tailed) and sub-gaussian (bulky, no outliers) if $< 0$.

The idea is then to maximize the kurtosis of the estimated sources for sphered data, as the extrema of the kurtosis correspond to independent sources. In practise, this is done by maximizing the negentropy:

$$J(p_x) = \underbrace{H(\phi_x)}_{\text{entropy of Gaussian}} - \underbrace{H(p_x)}_{\text{entropy of true distribution}}$$

which measures the difference in entropy between the actual distribution to a gaussian with the same mean and variance as the true distribution $p_x$. The negentropy is always non-negative. However, in algorithms like `fastICA` an approximation to the negentropy is maximized as it is computationally expensive to optimize the true entity.

In general, this method can be seen as projection pursuit method, as it projects into the direction of maximum non-gaussanity.

## 3.4 Discrete State Optimization

Up to now, almost all learning methods were based on a differentiable cost function $E^T$ which was minmized or maximized. This required real valued arguments to apply some form of gradient based optimization. In this section, methods are discussed which can be applied for discrete arguments.

**Simualetd Annealing:** This technique is inspired from the crystallization of certain materials under slow decrease of the temperature. Actually, it mimicks this slow cooling procedure and tries to converge to a minimum of a cost function. Given a cost function $E(\underline{\mathbf{s}})$ which takes a discrete state and a randomly chosen initial state, one starts picking a new state randomly, evaluates its energy and compares it to the old state. The new state is kept as the current state with a probability which is given by

$$P(\underline{\mathbf{s}}(t-1) \to \underline{\mathbf{s}}(t)) = \frac{1}{1+\exp(\beta_t \Delta E)}$$

where $\beta = \frac{1}{T}$, the inverse of the 'temperature' and $\Delta E = E(\underline{\mathbf{s}}(t)) - E(\underline{\mathbf{s}}(t-1))$ (i.e. $\Delta E > 0$: new state is worse, $\Delta E < 0$: new state is better). Hence, the probability of changing the current state depends on the temperature of the annealing procedure and the improvement by changing to the new state w.r.t the cost function. For high temperatures, it is likely that the state changes often, as if the particals of some material still have a lot of energy and wiggle around. Thus, in this stage of the annealing process, it is possible to escape from local optima in the cost landscape. In fact, a convergence to a global optima is guaranteed if $\beta_t \sim \ln t$.

However, in practise another annealing scheme is often chosen, as the optimal one is simply to slow (e.g. $\beta_t \sim \tau \beta_{t-1}$).

**The Gibbs Distribution:** We want a probability distribution across states, independent of the previos state, i.e. the probability for the transition $\underline{\mathbf{s}} \to \underline{\mathbf{s}}'$ should be the same as for the transition in the opposite direction $\underline{\mathbf{s}}' \to \underline{\mathbf{s}}$. In plain words we want $P(\underline{\mathbf{s}})$, which is given by the Gibbs- or Boltzmann distribution:

$$P(\underline{\mathbf{s}}) = \frac{1}{Z}\exp(-\beta E) \qquad \text{with } Z = \sum_{\underline{\mathbf{s}}} \exp(-\beta E).$$

The distribution is kind of the opposite to the cost function. For states with low costs, the probability is high and vice versa. However, it is as hard to obtain the maxima of $P(\underline{\mathbf{s}})$ as to get the minima of $E(\underline{\mathbf{s}})$.

**Mean Field Annealing:** In this technique, an approximation to the Gibbs distribution is calculated by searching for a factorizing distribution over so called 'mean-fields':

$$P(\underline{\mathbf{s}}) \approx Q(\underline{\mathbf{s}}) = \frac{1}{Z_Q}\exp(-\beta \sum_k \underbrace{e_k}_{\text{mean-fields}} s_k)$$

The first-order moment (expected value) of $Q(\underline{\mathbf{s}})$ factorizes which means that the state-variables are uncorrelated. Using the KL divergence again as an ansatz to minimize the distance between the true distribution $P(\underline{\mathbf{s}})$ and the familiy of factorizing distributions $Q(\underline{\mathbf{s}})$ w.r.t the mean fields:

$$D_{KL} = \sum_{\underline{\mathbf{s}}} Q(\underline{\mathbf{s}}) \ln \frac{Q(\underline{\mathbf{s}})}{P(\underline{\mathbf{s}})} \overset{!}{=} \min_{e_i}$$

The algorithm than works as follows: In each step, first the mean-fields are calculated, which are then used to calculate the moments $\langle s_k \rangle_Q$ (expected values) for each state variable. Additionally, in each step the temperature is decreased ($\beta$ is increased). In the limit (for $\beta \to \infty$), $\langle s_k \rangle_Q \to s_k^{\text{opt}}$, as $P(\underline{\mathbf{s}})$ is becoming singular at the state of optimal cost.

Mean field annealing is no longer a (slow) stochastic optimization procedure, but a fast determined procedure, given that the mean fields are easily computable.

## 3.5 Clustering

This technique obvioulsy is concerned with the task to finding clusters within a given set of datapoints. Ideally, these clusters represent different objects, which can be discriminated with respect to certain features.

Basically, there are two different kinds of clustering approaches, central clustering and pairwise clustering.

**Center Clustering (K-means):** K-means serves as an example of center clustering and is probably the most popular clustering algorithm, especially for clustering of vectorial data. Each cluster is described by a so called prototype (vector), which is located at the center of all the examples which belong to the cluster.

The batch-algorithm is actually a two step procedure. Given $k$ prototypes (initially, random vectors)

1. assign all datapoints to its closest prototype
2. move the prototypes to the center of mass of the cluster

until no change in assignment occurs. Technically, this corresponds to minimizing the average quadratic distance between the datapoints and the prototypes

$$E^T = \frac{1}{2p} \sum_{q,n} m_q^{(n)} (\underline{\mathbf{x}}_n - \underline{\mathbf{w}}_q)^2$$

where $m_q^{(n)}$ codes the membership of the datapoint $n$ to the cluster $q$ and $\sum_q m_q^{(n)} = 1$, i.e. each data point can only belong to a single cluster. Optimizing this error function with respect to a cluster $\underline{\mathbf{w}}_q$ yields the above stated result: The center of mass of a cluster is the (locally) optimal solution. Additionally, it can be shown that it is always a minimum.

The online version of $k$-means increases the chances of avoiding local minima. In this version, a datapoint is selected randomly at a time, assigned to its closest prototype and the prototypes' position is updated towards the new datapoint (scaled by a learning step $\eta$). The learning step should be small $\eta < 0 << 1$ and it is decreased exponentially after a certain amount of time $(T/4)$.

A small extension on the algorithm allows to select the number of clusters automatically, by specifying in advance what the resolution of the clustered data should be. This is done by defining a value for $(E^T_{\min})*$ (large = few clusters, small = many clusters) which represents the desired 'variance' (average quadratic distance between prototypes and data) of the data. As soon as $E^T_{\min}$ is smaller than that value, it's done. Otherwise, the cluster with largest variance will be selected and an additional prototype is added close to the position of the already existing prototype. Then $k$-means is repeated.

To get the 'right' amount of clusters, one can increase the number of $k$ until overfitting occurs.

**Pairwise Clustering:** The data which has to be clustered is not necessarily in vectorial form (or transformable into it); sometimes it is given as dyadic data (distance matrix) which describes the pairwise distances of the objects $d_{\alpha\beta}$. However, with this distance measure it is again possible to define a cost function comparable to that one defined for $k$-means clustering for $M$ clusters:

$$E^T = \frac{1}{M} \sum_q \frac{\sum_{n,m} m_q^{(n)} m_q^{(m)} d_{nm}}{\sum_n m_q^{(n)}}$$

which is the average distance of two objects which are in the same cluster, averaged over all clusters. This cost function now has to be minimized w.r.t the set of membership assignments $\{m_q^{(n)}\}$. In fact, assuming a squared euclidean distance measure, this cost function is equivalent to the one defined for $k$-means clustering.

However, if we have any other distance measure the problem appears to be a discrete optimization problem, which can be solved by applying stochastic optimization (simulated annealing) or mean-field annealing.

For the mean-field annealing, the same reasoning as before applies: One is interested in the probability distribution accross all discrete states $P(\underline{\mathbf{m}}_q^{(n)})$ (which is given by the Gibbs distribution) and tries to approximate it with a factorization of a familiy of distributions which involve the 'mean-fields'. This is done by minimzing the KL-divergence between the two, giving rise to a formulation of the mean-fields. They are in turn used to calcuate the moments (expected value) of each state variable, which then give assignment probabilities for each cluster. In the limit $\beta \to \infty$, the expected values become hard assignments, as the probability becomes singular in the optimal state.

A metric interpretation of a mean-field $e_p^{(n)}$ yields that they correspond to the average distance of the object $n$ to (all) the datapoints in cluster $p$, corrected by the size of cluster $p$. As they only depend

on the average distance of objects towards other data objects from a certain cluster, even uncomplete distance matrices can be clustered, by calculating the average over the available data.

$\beta$ is now the parameter which determines the complexity of the clustering solution. A decrease of $\beta$ increases, the average cost and the cluster sizes, and decreases the resolution.

Using the squared euclidean distance and still applying the mean-field annealing, will yield a 'fuzzy' version of $k$-means clustering ('soft' $k$-means).

Some common assumptions for pairwise clusering:

- symmetric distance matrix
- zero diagonal

**Self-Organizing Maps (SOM):** These maps for data visualisation, also called Kohonen networks, basically do clustering with some extra information. The clusters in data space (i.e. their prototypes) are mapped onto a euclidean grid of nodes in form of a WTA principle, i.e. each data point can only belong to a single node.

During the learning phase of the algorithm, the prototype which is closest to the given data sample adjusts its position as well as the prototypes which are in the neighbourhood of the node corresponding to the closest prototype. The change of neighbouring prototypes is determined by a so called neighborhood function.

Eventually, this procedure will create a lower dimensional 'map' of the data, including the information that neighboring nodes represent similar data clusters. Additionally, the amount of data in each cluster/node can be colorcoded in the map to get a nice visualisation of the data.

The online update rule for each prototype in a SOM is defined as following:

$$\underline{\mathbf{w}}_i(t+1) = \underline{\mathbf{w}}(t) + \eta h_{ci}(t)(\underline{\mathbf{x}}_n - \underline{\mathbf{w}}_i)$$

where $h_{ci}(t)$ represents the neighborhood function (in grid space!) and $c$ corresponds to the 'winning' prototype with smallest distance to the data sample $\underline{\mathbf{x}}_n$. A common example for the neighborhood function is a gaussian. Note that the learning rate $\eta$ as well as the width parameter $\sigma$ (e.g. std of gaussian) should be annealed.

Again, this concept of SOM can be transferred to pairwise data, yielding a discrete optimization problem which can be optimized by applying simulated or mean-field annealing.