All of Machine Learning

A summary under eternal construction

Moritz Gück
github.com/MoritzGuck/All_of_ML-under_construction
Last changes: April 7, 2020

Abstract

This is a reference for machine learning approaches and methods. The topics range from basic statistics to complex machine learning models and explanation methods. For each method and model, I have provided the underlying formulas (objective functions, prediction functions, etc.) as well as code snippets from the respective python libraries. I made this reference to quickly look up things I have studied already. I published it to give data scientists a catalogue to find methods for their problem, refresh their knowledge and give references for further reading. If you find errors or unclear explanations in this text, please file an issue under: `github.com/MoritzGuck/All_of_ML-under_construction` [?]

Contents

1 Probability Theory

A probability is a measure of how frequent or likely an event will take place.

Probability interpretations

Frequentist: Fraction of positive samples, if we measured infinitely many samples.

Objectivist: Probabilities are due to inherent uncertainty properties.

Subjectivist: An agent's degree of belief (not external).

Bayesian: (Building on subjectivism) A reasonable expectation on the basis of a state of knowledge/evidence.

! $\rightarrow$ Also the frequentist view is subjective since you need to compare events on otherwise similar objects. Usually there are no completely similar objects, so you need to define them.

Q $\rightarrow$ The Bayesian view allows to give certainties to events, where we don't have samples on (e.g. disappearance of the south pole until 2030).

Probability Space The probability space is a triplet space containing a sample/outcome space Ω (containing all possible atomic events), a collection of events S (containing a subset of Ω to which we want to assign probabilities) and the mapping P between Ω and S.

Axioms of Probability The mapping P must fulfill the axioms of probability:

1. $P(a) \geq 0$
2. $P(\Omega) = 1$
3. $a, b \in S$ and $a \cap b = \{\} \Rightarrow P(a \cup b) = P(a) + P(b)$

Random Variable (RV) A RV is a function that maps points from the sample space Ω to some range (e.g. Real numbers or booleans). They are characterized by their distribution function. E.g. for a dice roll:

$$X(\omega) = \begin{cases} 0, & \text{if } \omega = heads \\ 1, & \text{if } \omega = tails. \end{cases}$$

Proposition A Proposition is a conclusion of a statistical inference that can be true or false (e.g. a classification of a datapoint). More formally: A disjunction of events where the logic model holds. An event can be written as a **propositional logic model**:
$A = true, B = false \Rightarrow a \wedge \neg b$. Propositions can be continuous, discrete or boolean.

1.1 Probability distributions(PDF)

Probability distributions assign probabilities to to all possible points in Ω (e.g. $P(Weather) = \langle 0.3, 0.4, 0.2, 0.1 \rangle$, representing Rain, sunshine, clouds and snow). Joint probability distributions give you a probability for each atomic event of the RVs (e.g. $P(weather, accident)$ gives you a 2×4 matrix.)

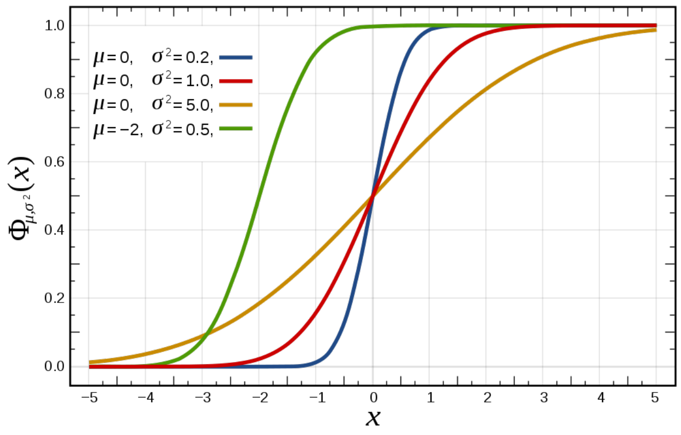

Fig. 1: Cumulative distribution function of a normal distribution for different mean (μ) and variance (σ). *Source: user Inductiveload on wikimedia.org.*

Cumulative Distribution Function (CDF) The CDF is defined as $F_X(x) = P(X \leq x)$ (See figure 8).

Probability Density Function (PDF) For continuous functions the PDF is defined by

$$p(x) = \frac{d}{dx}p(X \leq x). \tag{1}$$

The probability of x being in a finite interval is

$$P(a < X \leq b) = \int_a^b p(x)dx \tag{2}$$

A PDF is shown in figure

Properties of Distributions

The **expected value** (E) or **mean** (μ) is given by $E[X] = \sum_{x \in X} x * p(x)$ for discrete RVs and $E[X] = \int_X x * p(x)dx$ for continuous RVs.

The **variance** measures the spread of a distribution: $var[X] = \sigma^2 = E[(X-\mu)^2] = E[X]^2 - \mu^2$.

The **standard deviation** is given by: $\sqrt{var[X]} = \sigma$.

The **mode** is the value with the highest probability (or the point in the PDF with the highest value):

The **median** is the point at which all point less than the median and all points greater than the median have the same probability (0.5).

The **quantiles** (Q) divide the datapoints into sets of equal number. The Q_1 qua**r**tile has 25% of the values below it.

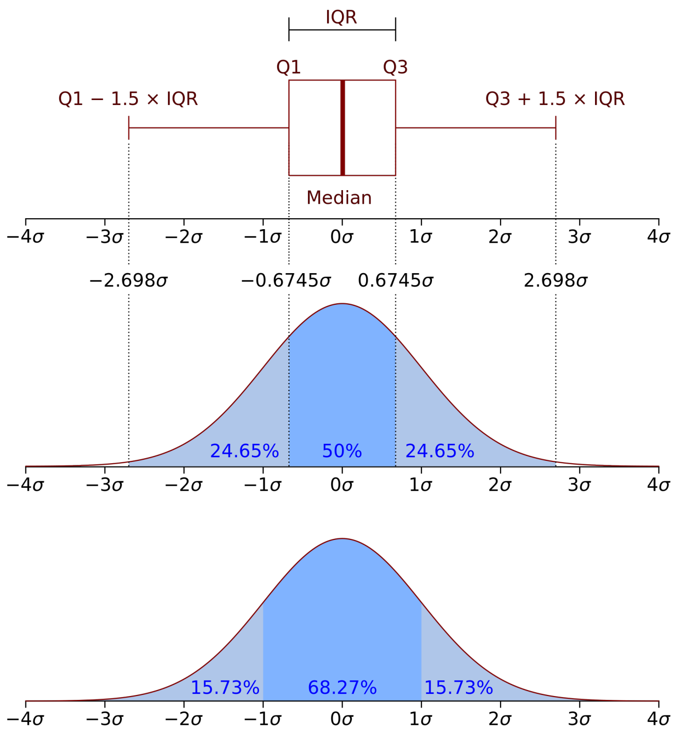

Fig. 2: Probability density function of a normal distribution with variance (σ). In red a range from a Box-plot is shown with quartiles (Q1, Q3) and interquartile range (IQR). For the cutoffs (borders to darker blue regions) the IQR (on top) and σ are chosen. Another common cutoff is the confidence interval with light blue regions having a probability mass of $2 * \alpha/2$. *Source: user Jhguch on wikimedia.org.*

Dirac delta function is a function that is infinite at one point and 0 everywhere else:

$$\delta(x) = \begin{cases} \infty, & \text{if } x = 0 \\ 0, & \text{if } x \neq 0 \end{cases} \qquad \text{and} \quad \int_{-\infty}^{\infty} \delta(x) dx = 1$$

1.1.1 Uniform distribution

The uniform distribution has the same probability throughout a specific interval:

$$\text{Unif}(a, b) = \frac{1}{b-a} \mathbb{1}(a < x \leq b) = \begin{cases} \frac{1}{b-a}, & \text{if } x \in [a, b] \\ 0, & \text{else} \end{cases}$$

$\mathbb{1}$ is a vector of ones.

1.1.2 Discrete distributions

Used for random variables that have discrete states.

Binomial distribution Used for experiments with two outcomes (e.g. coin flips).

$$X \sim \text{Bin}(n, \theta) \quad , \text{Bin}(k|n, \theta) = \binom{n}{k} \theta^k (1-\theta)^{n-k} \quad , \binom{n}{k} = \frac{n!}{k!(n-k)!},$$

where n is the number of total experiments, k is the number of sucessful experiments and θ is the probability of success of an experiment.

Bernoulli distribution Is a special case of the binomial distribution with $n = 1$.

$$X \sim \text{Ber}(\theta) \quad , \text{Ber}(x|\theta) = \theta^{\mathbb{1}(x=1)} (1-\theta)^{\mathbb{1}(x=0)} = \begin{cases} \theta, & \text{if } x = 1 \\ 1-\theta, & \text{if } x = 0 \end{cases}$$

Multinomial distribution Used for experiments with k different outcomes (e.g. dice rolls).

$$\text{Mu}(x|n, \theta) = \binom{n}{x_1, ..., x_K} \prod_{i=1}^{K} \theta_j^{x_j} = \frac{n!}{x_1!, ..., x_k!} \prod_{i=1}^{K} \theta_j^{x_j},$$

where k is the number of outcomes, x_j is the number times that outcome j happens. $X = (X_1, ..., X_K)$ is the *random vector*.

Multinoulli distribution Is a special case of the multinomial distribution with $n = 1$. The random vector is then represented in *dummy-* or *one-hot-encoding* (e.g. $(0, 0, 1, 0, 0, 0)$ if outcome 3 takes place).

$$\text{Mu}(x|1, \theta) = \prod_{j=0}^{K} \theta_j^{\mathbb{1}(x_j=1)}$$

Empirical distribution

$$\mathsf{p}_{\mathsf{emp}}(A) = \frac{1}{N}\sum_{i=1}^{N}\delta_{x_i}(A), \quad \delta_{x_i} = \begin{cases} 1, & \text{if } x \in A \\ 0, & \text{if } x \notin A \end{cases},$$

w where $x_1, ..., x_N$ is a data set with N points. The points can also be weighted:

$$p(x) = \sum_{i=1}^{N} w_i \delta_{x_i}(x)$$

1.1.3 Continuous distributions

Used for random variables that have continuous states.

Normal/Gaussian distribution Often chosen for random noise because it is simple and needs few assumptions (see sect. 1.1.4). The PDF is given by:

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

where μ is the mean and σ^2 is the variance. The CDF is given by:

$$\Phi(x) = \frac{1}{\sqrt{2\pi}} \int_{\infty}^{x} e^{\frac{-t^2}{2}dt}$$

Multivariate normal/Gaussian distribution For T datapoints with k dimensions (features). The pdf is:

$$p(x|\mu, \Sigma) = \frac{1}{\sqrt{(2\pi)^k |\Sigma|}} \exp\left[-\frac{1}{2}(x-\mu)^\top \Sigma^{-1}(x-\mu)\right],$$

where x now has multiple dimension $(x_1, x_2, ..., x_k)$ and Σ is the $k \times k$ covariance matrix: $\Sigma = \mathsf{E}[(X-\mu)(X-\mu)]$. The covariance between features is: $\mathsf{Cov}[X_i, X_j] = \mathsf{E}[(X_i - \mu_i)(X_j - \mu_j)]$

Beta distribution defined for $0 \leq x \leq 1$ (see figure 3). The pdf is:

$$f(x|\alpha, \beta) = \frac{1}{B(\alpha, \beta)} x^{\alpha-1}(1-x)^{\beta-1}$$

The beta function is there to normalize and ensure that the total probability is 1.

Dirichlet distribution The multivariate version of the Beta distribution (see fig. 4). The PDF is:

$$\mathsf{Dir}(x|\alpha) \triangleq \frac{1}{B(\alpha)} \prod_{i=1}^{K} x_i^{\alpha_i - 1}, \quad \sum_{i=1}^{K} x_i = 1, \quad x_i \geq 0 \; \forall i$$

Marginal distributions Are the probability distributions of subsets of the original distribution. Marginal distributions of normal distributions are also normal distributions (see fig. 5).

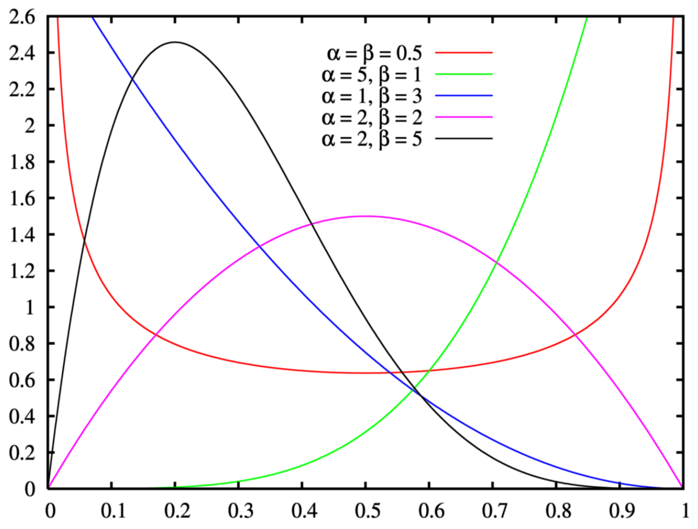

Fig. 3: Probability density function of a beta-distribution with different parameter values. *Source: user MarkSweep on wikimedia.org.*

1.1.4 Central limit theorem

In many cases the sum of random variables will follow a normal distribution as n goes to infinity.

1.2 Conditional/Posterior Probability

Expresses the probability of one event (Y) under the condition that another event (E) has occurred. (e.g. $C =$ "gets cancer", $S =$ "is a smoker" $\rightarrow p(C|S) = 0.2$, meaning: "given the *sole information* that someone is a smoker, their probability of getting cancer is 20%.")

The conditional probability can be calculated like this:

$$P(A \mid B) = \frac{P(A \cap B)}{P(B)} = \frac{P(A, B)}{P(B)} = \alpha P(A, B),$$

where α is used as a normalization constant. If you have hidden variables (confounding factors) you need to sum them out like so:

$$P(Y|E = e) = \alpha P(Y, E = e) = \alpha \sum_h P(Y, E = e, H = h)$$

where X contains all variables, Y is called *query variable*, E is called *evidence variable*, $H = X - Y - E$ is called *hidden variable*. You get the joint probabilities by summing out the

! $\rightarrow$ Usually $p(A|B) \neq p(B|A)$

! $\rightarrow$ Priors are often forgotten: E.g. P("COVID-19") is confused with P("COVID-19"|"Person is getting tested") (because only people with symptoms go to the testing station).

! $\rightarrow$ Base rate neglect: Under-representing prior probability. E.g. You have a test with a 5% false positive rate and a incidence of disease of 2% in the population. If you are tested positive in a population screening your probability of having the disease is only 29%.

🔍 $\rightarrow$ Conditional distributions of Gaussian distributions are Gaussian distributions themselves.

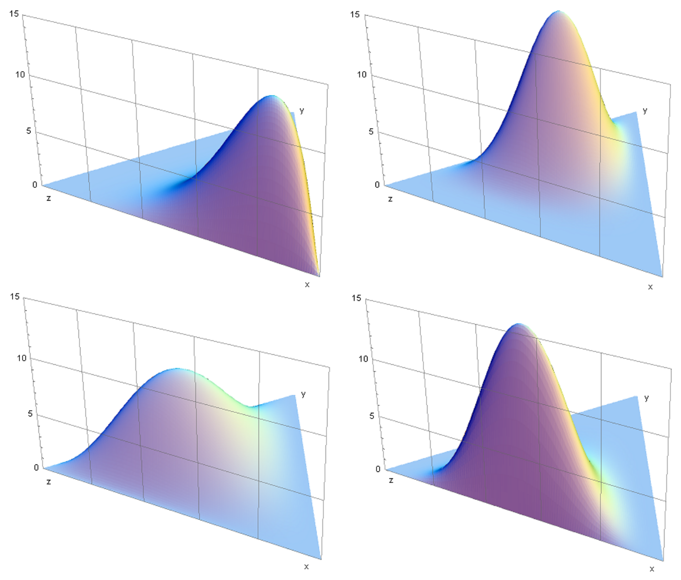

Fig. 4: Probability density function of a Dirichlet-distribution on a 2-simplex (triangle) with different parameter values. Clockwise from top left: $\alpha =$ (6,2,2), (3,7,5), (6,2,6), (2,3,4). *Source: user ThG on wikimedia.org.*

Independence For independent variables it holds: $P(A|B) = P(A)$ or $P(B|A) = P(B)$

Conditional independence Two events A and B are independent, given C: $P(A|B,C) = P(A|C)$. A and B must not have any information on each other, given the information on C. E.g. for children: $P(\text{"vocabulary"}|\text{"height"}, \text{"age"}) = P(\text{"vocabulary"}|\text{"age"})$.

1.2.1 Bayes Rule

Bayes rule:

$$P(\text{hypothesis}|\text{evidence}) = \frac{P(\text{evidence}|\text{hypothesis})P(\text{hypothesis})}{P(\text{evidence})}$$

often used as:

$$P(\text{model}|\text{data}) = \frac{P(\text{data}|\text{model})P(\text{model})}{P(\text{data})}$$

Terminology:

- $P(\text{hypothesis}|\text{evidence})$ = Posterior
- $P(\text{evidence}|\text{hypothesis})$ = Likelihood
- $P(\text{hypothesis})$ = Prior (How probable hypothesis was before seeing evidence)
- $P(\text{evidence})$ = Marginal (How probable evidence is under all possible hypotheses)
- $\frac{P(\text{evidence}|\text{hypothesis})}{P(\text{evidence})}$ = Support B provides for A

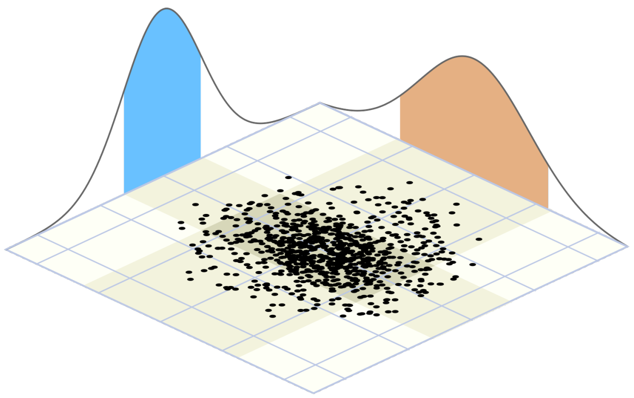

Fig. 5: Data following a 2D-Gaussian distribution. Marginal distributions are shown on the sides in blue and orange. *Source: user Auguel on wikimedia.org.*

- $P(\text{data}|\text{model})P(\text{model})$ = joint probability $(P(A, B))$

The proof (see above): $P(A|B)P(B) = P(A, B) = P(B|A)P(A)$

Example for Bayes Rule using COVID-19 Diagnostics

$$P(\text{COVID-19}|\text{cough}) = \frac{P(\text{cough}|\text{COVID-19})P(\text{COVID-19})}{P(\text{cough})} = \frac{0.7 * 0.01}{0.1} = 0.07$$

Estimating $P(\text{COVID-19}|\text{cough})$ is difficult, because there can be an outbreak and the number changes. However, $P(\text{cough}|\text{COVID-19})$ stays stable, $P(\text{COVID-19})$ and $P(\text{cough})$ can be easily determined.

1.3 Further Concepts

Convergence in Probability of Random Variables You expect your random variables (X_i) to converge to an expected random variable X. I.e. after looking at infinite samples, the probability that your random variable X_n differs more than a threshold ϵ from your target X should be zero.

$$\lim_{n\to\infty} P(|X_n - X| > \epsilon) = 0$$

Bernoulli's Theorem / Weak Law of Large Numbers

$$\lim_{n\to\infty} P(|\frac{\sum_{i=1}^{n} X_i}{n} - \mu| > \epsilon) = 0,$$

where $X_1, ..., X_n$ are independent & identically distributed (i.i.d.) RVs. $\Rightarrow$ With enough samples, the sample mean will approach the true mean. The **strong law of large numbers** states that $|\frac{\sum_{i=1}^{n} X_i}{n} - \mu| < \epsilon$ for any $\epsilon > 0$.

Bias of an estimator You have a model with a parameter $\hat{\theta}$ that is an estimator for the true θ. You want to know whether your model over- or underestimates the true θ systematically.

$$\text{Bias}[\hat{\theta}] = \text{E}_{X|\mathcal{D}}[\hat{\theta}] - \theta$$

2 Classification Methods

Classification is the assignment of objects (data points) to categories (classes). It requires a data set (i.e. training set) of points with known class labels. If the class labels are not known you can instead group the data using clustering algorithms (chapter 3).

2.1 Evaluation of Classifiers

2.1.1 Basic Quality Measures

Accuracy / Success Rate

Precision

Recall

Specificity

Sensitivity

Support

F-score

2.1.2 Area under the Precision-Recall Curve

2.1.3 Handling Unbalanced Data

Having many more samples in one class than the others during training can lead to high accuracy values event hough the classifier performs poorly on the small classes. You can handle the unbalance by:

- up-sampling the smaller data set (creating more artificial samples for that class)
- giving more weight to the samples in the smaller data set

Implementations for Handling Unbalanced Data Sets

Oversampling using imbalanced-learn (see: documentation)

```
from imblearn.over_sampling import RandomOverSampler
ros = RandomOverSampler(random_state=0)
features_resampled, labels_resampled = ros.fit_resample(df[feature_cols],
    df[label_col])
```

Sensitivity, specificity, precision, recall, support and F-score

```
y_true = df[label_col]
y_pred = classifier.predict(df[feature_cols])

from imblearn.metrics import sensitivity_specificity_support
sensitivity, specificity, support = sensitivity_specificity_support(y_true,
    y_pred)

from sklearn.metrics import precision_recall_fscore_support
precision, recall, fscore, support =
    precision_recall_fscore_support(y_true, y_pred)
```

2.2 Linear Classifiers

Linear classifiers use linear decision boundaries to classify points to a respective class.

2.2.1 Support Vector Classifier (SVC)

SVCs use hyperplanes to separate data points according to their class label with a maximum margin (M) between the separating hyperplane ($x^T\beta + \beta_0 = 0$) and the points. If points cannot be perfectly separated by the decision boundary, a soft margin SVM is used with a slack variable ξ that punishes points in the margin or on the wrong side of the hyperplane. The optimization problem is given by [?] :

$$\begin{aligned} \max_{\beta,\beta_0,\beta=1} M, \\ \text{subject to } y_i(x_i^T\beta + \beta_0) \geq 1 - \xi_i, \quad \forall i, \\ \xi_i \geq 0, \quad \sum \xi_i \leq constant, \quad i = 1, ..., N, \end{aligned} \tag{3}$$

where β are the coefficients and x are the N data points. The support vectors are the points that determine the orientation of the hyperplane (i.e. the closest points). The classification function is given by:

$$G(x) = \text{sign}[x^T\beta + \beta_0] \tag{4}$$

If you only calculate the inner part of the function you can get the distance of a point to your hyperplane (in SKlearn you need to divide by the norm vector w of your hyperplane to get the true distance). To get the probability of a point being in a class, you can use Platt's algorithm [?]. SVMs are sensitive to the scaling of the features. Therefore, the data should be normalized before classification.

Implementation of SVCs

```
from sklearn import svm
# train the model
svc_model = svm.SVC()
svc_model.fit(train_df[feautre_cols], train_df[label_col])
# test the model
y_predict = svc_model.predict(test_df[feature_cols])
```

3 Clustering Methods

Clustering methods are used to group data when no class labels are present. You thereby want to learn an intrinsic structure of the data.

3.1 K-Means Clustering

Goal: Divide data into K clusters so that the variance within the clusters is minimized. The objective function:

$$V(D) = \sum_{i=1}^{k} \sum x_j \in C_i (x_j - \mu_i)^2, \tag{5}$$

where V is the variance, C_i is a cluster, μ_i is a cluster mean, x_j is a datapoint. The algorithm works as follows:

1. Assign the data to k initial clusters
2. Calculate the mean of each cluster
3. Assign the data points to the closest cluster mean
4. If a point changed its cluster, repeat from step 2

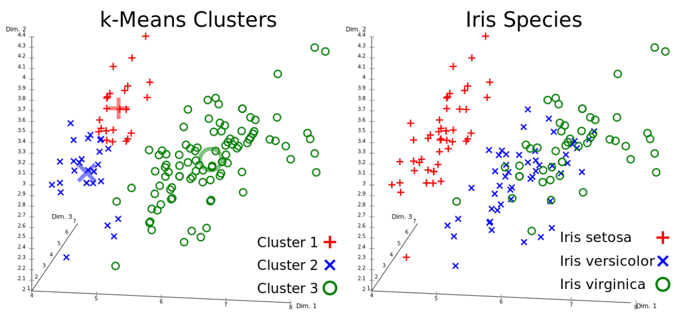

Fig. 6: Data from the *Iris flower data set* clustered into 3 clusters using k-Means. On the right the data points have been assigned to their actual species. *Figure from user Chire on wikimedia.org.*

3.2 Graph-Based Clustering

You represent data set D as a graph $G = (V, E)$ and divide it up in connected sub-graphs that represent your clusters. Each edge e_{ij} (between nodes v_i and v_j) has a weight w_{ij} (which is commonly a similarity or distance measure).

Basic Graph-Based Clustering The basic algorithm works like this:

1. Define a weight-threshold θ
2. For all edges: if $w_{ij} > \theta$: remove e_{ij}
3. If nodes are connected by a path (via *depth first search*): Assign are them to the same cluster

DBScan *Density-Based Spatial Clustering of Applications with Noise* is a more noise robust version of graph-based clustering.

Cut-Based Clustering You introduce a **adjacency/similarity** matrix W (measures similarity between data points) and define the number of clusters k. You now try to minimize the weight of edges between the clusters (equal to cutting edges between nodes that are least similar):

$$\begin{aligned} &\min \frac{1}{2} \sum_{a=1}^{k} \sum_{b=1}^{k} \kappa(C_a, C_b) \\ \text{where } \kappa(C_a, C_b) = &\sum_{v_i \in C_a, v_j \in C_b, a \neq b} W_{ij} \\ &\text{and } \kappa(C_a, C_a) = 0 \end{aligned}$$

$\rightarrow$ You only add up the similarities/edge-weights between your clusters (but not within your clusters).
$\rightarrow$ For constructing the similarity matrix, different kernels can be used (commonly the linear kernel or the Gaussian kernel).

3.3 Spectral Clustering

The goal is again, to cluster the points, such that the similarity of points of different clusters is minimal. Spectral clustering employs three steps: Preprocessing, decomposition and grouping.

Preprocessing We create a Laplacian matrix L (laplacian operator in matrix form, measuring how strongly a vertex differs from nearby vertices (because the edges are similarity measures)):

$$L = D - W$$

$$D_{ij} = \begin{cases} \sum_{j=1}^{N} W_{ij} \\ 0 \text{ if } i \neq j \end{cases}$$

where D is the degree matrix (the degree of each node is on the diagonal).

Decomposition The class assignment is encoded in vectors c_k. We normalize them and get e.g.:

$$u_a = \frac{c_a(i)}{||c_a||} = \begin{cases} \frac{1}{||c_a||} \text{ if } v_i \in C_a \\ \frac{0}{||c_a||} \text{ if } v_i \notin C_a \end{cases}$$

As in cut-based clustering we search for a minimum k-cut, but also add a constraint that $u_a^T u_a = 1$

$$\min \frac{1}{2} \sum_{a=1}^{k} u_a^T L u_a + \lambda_a \sum_{a=1}^{k} 1 - ||u_a||^2$$

We find the optimal u_a by eigenvalue decomposition:

$$L u_a = \lambda u_a$$

The final data is now represented as a matrix of k eigenvectors (of the least dominant eigenvalues) as column-vectors.

Grouping You get the final cluster assignments by normalizing the now k-dimensional data and applying k-means clustering to it.

3.3.1 Sparse Subspace Clustering (SSP)

The underlying assumption of SSP is that the different clusters reside in different subspaces of the data. Clusters are therefore perpendicular to each other and points in a cluster can only be reconstructed by combinations of points in the same cluster ($\rightarrow$ self-expressiveness, the reconstruction vectors ought to be sparse). For each point you try to find other points that can be used to recreate that point - these then form the same cluster. Doing that for all points gives you a data matrix X and a matrix of reconstruction vectors V:

$$X = X * V \text{ s.t. } \text{diag}(V) = 0.$$

You now try to minimize the V-matrix according to the L1-norm (giving you a sparse matrix). This matrix can then be (multiplied with its transpose and) used for e.g. spectral clustering.

3.4 Soft-assignment Clustering

3.4.1 Expectation Maximization (EM) Clustering

[Coming soon]

3.4.2 Gaussian Mixture Models

[Coming soon]

3.5 Artificial Neural Networks for Clustering

See chapter *Neural Networks* (5)

4 Generative models

4.1 Generative Models for Discrete Data

4.1.1 Bayesian Concept Learning

can learn a concept $c \in C$ from positive examples alone. For that define the posterior: $p(c|\mathcal{D})$. To get to learn a concept you need a hypothesis space $\mathcal{H}$ and a version space (a subset of $\mathcal{H}$) that is consistent with $\mathcal{D}$. You choose a hypothesis h by assuming that samples are randomly chosen from the true concept and calculate $p(\mathcal{D}|h) = [\frac{1}{|h|}]^N$ (sampling the N data points from h). You than choose the hypothesis that has the highest probability (thereby you choose suspicious coincidences of too broad models). The priors can be chosen e.g. by giving lower priority to concepts with complex rules (e.g. "all powers of 2 below 100 but not 64."). This is subjective, however often beneficial for rapid learning. Using Bayes rule, we can calculate the posterior:

$$p(h|\mathcal{D}) = \frac{p(\mathcal{D}|h)p(h)}{p(\mathcal{D})} = \frac{p(\mathcal{D}|h)p(h)}{\sum_{h' \in \mathcal{H}} p(\mathcal{D}|h')p(h')} = \frac{\mathbb{I}(\mathcal{D} \in h)p(h)}{\sum_{h' \in \mathcal{H}} \mathbb{I}(\mathcal{D} \in h')p(h')},$$

where $\mathbb{I}(\mathcal{D} \in h)p(h) = 1$ if the data adhere to the h. The maximum of $p(h|\mathcal{D})$ is the **MAP estimate**.

With more data the MAP-estimate converges to the MLE. If the true hypothesis is in $\mathcal{H}$ then MLE and MAP will converge to it ($\rightarrow$ consistent estimators). If you take the entire distribution of the hypotheses you get a distribution for the estimate (and not a point prediction) $\rightarrow$ **posterior predictive distribution**.

$$p(\tilde{x}|\mathcal{D}) = \sum_h p(\tilde{x}|h)p(h|\mathcal{D})$$

This weighting of hypotheses is called **Bayes model averaging**. For small data sets you get a vague posterior and broad predictive distribution. You can replace the posteriors with their delta-function:

$$p(\tilde{x}|\mathcal{D}) = \sum_h p(\tilde{x}|h)\delta_{\hat{h}_{\text{MAP}}}(h)$$

$\rightarrow$ **plug-in approximation** (under-represents uncertainty).

4.1.2 Beta-binomial model

This is a distribution that uses a binomial distribution as its likelihood and a beta-distribution over it's θ parameter as its prior.

Likelihood

$$p(\mathcal{D}|\theta) = \text{Bin}(k|n, \theta) \propto \theta^k(1-\theta)^{n-k},$$

where k are the successful trials, n are the total trials and θ are the success-probabilities of the single experiments. k and $n-k$ are *sufficient statistics of the data*: $p(\theta|\mathcal{D} = p(\theta|k, n-k))$.

Prior

$$\text{Beta}(\theta|\alpha, \beta) \propto \theta^{\alpha-1}(1-\theta)^{\beta-1}$$

the parameters α and β are used as hyper parameters. The prior has the same form as the likelihood $\rightarrow$ *conjugate prior*.

Posterior

$$p(\theta|\mathcal{D}) \propto \mathsf{Bin}(k|n,\theta)\mathsf{Beta}(\theta|\alpha,\beta) = \mathsf{Beta}(\theta|k+\alpha, n-k+\beta)$$

The posterior predictive distribution is:

$$\begin{aligned} p(\tilde{x}=1|\mathcal{D}) &= \int_0^1 p(\tilde{x}=1|\theta)p(\theta|\mathcal{D})\mathrm{d}\theta = \int_0^1 \theta\mathsf{Beta}(\theta|\alpha,\beta)\mathrm{d}\theta & (6) \\ &= \mathsf{E}[\theta|\mathcal{D}] = \frac{N-k+\alpha}{N\cancel{-k+k}+\alpha+\beta} & (7) \end{aligned}$$

5 Neural Networks

5.1 Introduction

On the most basic level neural networks consist of many simple models (e.g. linear and logistic models) that are chained together in a directed network. The models sit on the neurons (nodes) of the network. The most important components of neurons are:

1. **Activation**: $a = Wx + b$ (W = weights and b = bias)
2. **Non-linearity**: $f(x, \theta) = \sigma(a)$ (e.g. a sigmoid function for logistic regression, giving you a probability output. θ is a threshold)

The neurons (nodes) in the first layer uses as its input the sample values and feeds its output into the activation function of the next nodes in the next layer, a.s.o. The later layers should thereby learn more and more complicated concepts or structures.

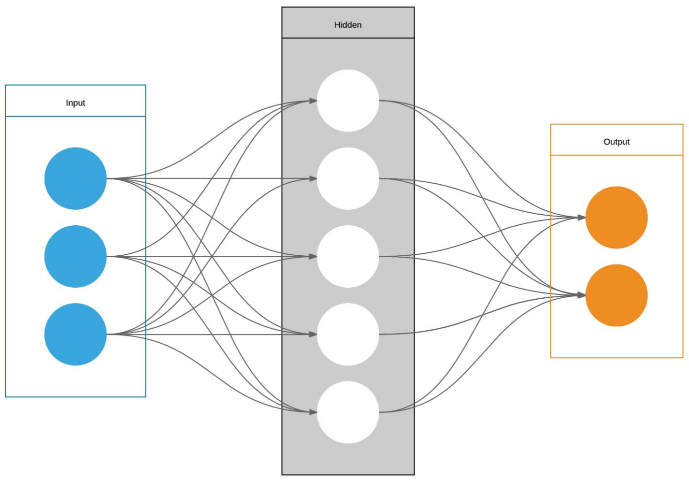

Fig. 7: Model of an artificial neural network with one hidden layer. *Figure from user LearnDataSci on wikimedia.org.*

5.1.1 Non-Linearities

Different non-linear functions can be used to generate the output of the neurons.

Sigmoid/Logistic Functions

Tanh Functions

Rectifiers/ReLU

Terminology

Input layer/visible layer: Input variables

Hidden layer: Layers of nodes between input and output layer

Output layer: Layer of nodes that produce output variables

Size: Number of nodes in the network

Width: Number of nodes in a layer

Depth: Number of layers

Capacity: The type of functions that can be learned by the network

Architecture: The arrangement of layers and nodes in the network

5.2 Feedforward Neural Network / Multi-Layer Perceptron

This is the simplest type of proper neural networks. Each neuron of a layer is connected to each neuron of the next layer and there are no cycles. The outputs of the previous layer corresponds to the x in the activation function. Each output (x_i) of the previous layer gets it's own weight (w_i) in each node and a bias (b) is added to each node. Neurons with a very high output are "active" neurons, those with negative outputs are "inactive". The result is mapped to the probability range by (commonly) a sigmoid function. The output is then again given to the next layer.

! $\rightarrow$ If your input layer has 6400 features (80*80 image), a network with 2 hidden layers of 16 nodes will have $6400 * 16 + 16 * 16 + 16 * 10 + 16 + 16 + 10 = 102'858$ parameters. This is a very high number of degrees of freedom and requires a lot of training samples.

Implementation of Feedforward Neural Nets

```
from torch import nn

class CustomNet(nn.Module):
    def __init__(self):
        super(CustomNet, self).__init__()
        self.lin_layer_1 = nn.Linear(in_features=10, out_features=10)
        self.relu = nn.ReLU()
        self.lin_layer_2 = nn.Linear(in_features=10, out_features=10)

    def forward(self, x):
        x = self.lin_layer_1(x)
        x = self.relu
        x = self.lin_layer_2(x)
        return x

    def num_flat_features(self, x):
        size = x.size()[1:] # Use all but the batch dimension
        num = 1
        for i in size:
            num *= i
        return num

new_net = CustomNet()
```

5.2.1 Bekpropageshn

This is the method by which neural networks learn the optimal weights and biases of the nodes. The components are a cost function and a gradient descent method.
The cost function analyses the difference between the designated activation in the output layer (according to the label of the data) and the actual activation of that layer. Commonly a residual sum of squares is used.
You get the direction of the next best parameter-combination by using a *stochastic gradient descent* algorithm using the gradient for your cost function:

1. We use a "mini-batch" of images for each round/step of the gradient descent.
2. We calculate squared residual of each feature of the output layer for each sample.
3. From that we calculate what the bias or weights from the output layer and the activation from the last hidden layer must have been to get this result. We average that out for all images in our mini-batch.
4. From that we calculate the weights, biases and activations of the upstream layers $\rightarrow$ we *backpropagate*.

5.3 Convolutional Neural Networks

5.4 Autoencoders

Contrary to the other architectures, autoencoders are used for unsupervised learning. Their goal is to compress and decompress data to learn the most important structures of the data. The layers therefore become smaller for the encoding step and the later layers get bigger again, up to the original representation of the data. The optimization problem is now:

$$\min_{W,b} \frac{1}{N} * \sum_{i=1}^{N} ||x_i - \hat{x}_i||^2 \tag{8}$$

with x_i being the original datapoint and $\hat{x}_i$ the reconstructed datapoint.

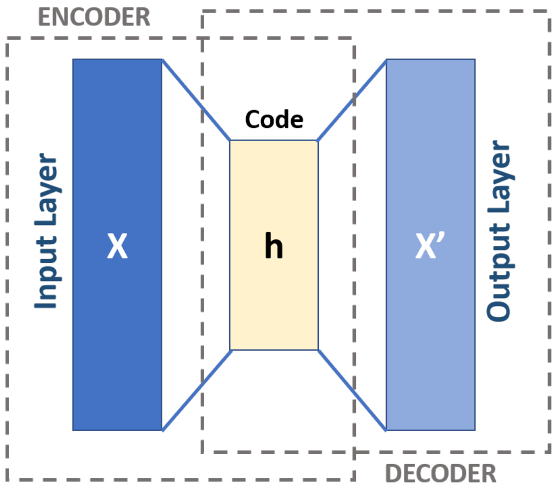

Fig. 8: Model of an autoencoder. The encoder layers compress the data towards the code layer, the decoder layers decompress the data again. *Figure from Michela Massi on wikimedia.org.*

5.4.1 Autoencoders for clustering

You can look at layers of a NN as ways to represent data in different form of complexity and compactness. The code layers of autoencoders are a very compact way to represent the data. You can then use the compressed representation of the code layer and do clustering on that data. Because the code layer is however not optimized for that task XXXX combined the cost function of the **autoencoder and k-means clustering**:

$$\min_{W,b} \frac{1}{N} * \sum_{i=1}^{N} ||x_i - \hat{x}_i||^2 - \lambda \sum_{i=1}^{N} ||f(x_i) - c_i||^2 \tag{9}$$

with $f(x_i)$ being the non-linearity of the code layer and λ is a weight constant.

XXXX adapted spectral clustering (section 3.3) using autoencoders by replacing the (linear) eigen-decomposition with the (non-linear) decomposition by the encoder. As in spectral clustering the Lapla-

cian matrix is used as the the input to the decomposition step (encoder) and the compressed representation (code-layer) is fed into k-means clustering.
Deep subspace clustering by XXXX employs autoencoders combined with sparse subspace clustering 3.3.1. They used autoencoders and optimized for a compact representation of the code layer:

$$\begin{gathered}\min_{W,b} \frac{1}{N} * \sum_{i=1}^{N} ||x_i - \hat{x}_i||^2 - \lambda ||V||_1 \\ \text{s.t.} F(X) = F(X) * V \text{ and } \operatorname{diag}(V) = 0\end{gathered} \tag{10}$$

with V being the sparse representation of the code layer $(F(X))$.

5.5 Generative Adversarial Networks

5.6 Recurrent Neural Networks

Index