Data Mining and Text Mining

Data mining is the process of identifying valid, novel, potentially useful, understandable patterns in data.
It can be descriptive (models built for gaining insights), predictive (models built for predictions) or prescriptive (descriptive and predictive models are combined and used to recommend an action).

0. Introduction

Data representation

- **Attributes**: numerical, categorical, ordinal, binary
- **Missing values**
  - Missing not at random (MNAR): distribution of missing values depends on missing values.
  - Missing at random (MAR): distribution of missing values depends on observed attributes, but not on missing values (it depends on the *type* of the target, not on the *value*).
  - Missing completely at random (MCAR): distribution of missing values does not depend on observed attributes nor missing values.

  How to deal? Do not impute (DNI) methods are deletion methods: list-wise (delete all the instances with a missing value), pairwise (first decide which attributes will be considered, then delete all the instances with a missing value). Single inputation methods: mean/mode, dummy variable control (to mark that an instance had a missing value), regression imputation. Model-based methods: MLE.
- **Attribute conversions**
  → ***Label encoder***: maps a categorical variable described by n labels into a numerical variable with values from 0 to n-1 (it creates ordinals).
  → ***One-hot encoding***: maps a categorical attribute with n values into n binary variables (each one describing one specific attribute value).
  → ***Discretization***: transforms continuous variables into categorical (ordinal) ones. It returns a set of contiguous intervals that are used to map the original variable range into a sequence of labels corresponding to the intervals containing such values. Approach: unsupervised (does not use infos about the target to set partition boundaries), supervised (uses infos about the target to generate intervals that try to preserve relevant informations about the class). The number of intervals is an hyperparameter. Common unsupervised methods: equal-width (sensitive to outliers), equal-frequency, clustering. Supervised method: decision tree. Supervised might perform better, but not statistically significant.

Data exploration and preparation

- **Summary statistics**: frequency, mode, mean/median (location), percentiles, range/variance (spread)
- **Outliers**: Trimming (eliminate the outlier instance), Winsorizing (outliers under 5th and above 95th perc.)
- **Normalization**
  → Range: $x'_i = \frac{x_i - \min_i x_i}{\max_i x_i - \min_i x_i}$
  → Standard score (z-score): $x'_i = \frac{x_i - \mu}{\sigma}$
- **Visualization**: *bar plot* (horizontal/vertical bars to compare categories), *histogram* (graphical representation of the distribution of the data, they estimate the probability distribution of a continuous variable), *kernel density estimation* (place a kernel function at each observed datapoint and then sum them all), *box plot* (display the distribution of the data: the box is built with 1st (Q_1), 2nd (Q_2) and 3rd (Q_3) quartile, the lines are determined with Q_1-$\frac{3}{2}$IQR and Q_3+$\frac{3}{2}$IQR), *violin plots* (add kernl density estimate to the boxplot), *scatter plot* (used to compare two+ attribtues),*spatial data tools* (dot maps, bubble maps, isocontour maps), *higher dimension tools* (heat maps, spider/radar/star plots, Chemoff faces).
- **Lowering data dimensions**: when we project high-dimensional data into fewer dimensions

  → ***PCA***: linear projection. It's used to reduce the number of dimensions of data (features selection). The goal is to find a projection that captures the largest amount of variation in the data. It works only for numeric data and it's affected by scale.

  1. Normalize input data
  2. Compute k orthonormal vectors (i.e. principal components), which will be sorted in order of decreasing significance
  3. Rewrite each input datapoint as a linear combination of the k principal components

  Data size can be reduced by eliminating the weak components (those with low variance). Using the strongest principal components it is possible to reconstruct a good approximation of the original data.

  → ***t-SNE***: non-linear projection into spaces of 2-3 dimensions. The physical analogy is: we consider a set of points in a high-dim space, we put a strong spring between two close points and a weak spring between two far points, we repeat for all points and we let the points fall down on a 2-dim space. Once the springs have calmed down, the closest points in higher-dim will be close also in the projection.

  1. Define a probability distribution over pairs of high-dim data points: similar data points have a high probability of being picked, dissimilar data points have small probability
  2. Define a similar distribution over the points in the map space
  3. Minimize the Kullback-Leibler divergence between the two distributions w.r.t. the locations of the map points (through gradient descent)

  The mapped points are not linear combination of original attribute values and the axes of mapped space are not linear combination of original axes.

1. Association Rules

- **Association rules**: finding frequent patterns and associations among sets of items in transactions or relational databases. An association rule is an implication of the form $X \Rightarrow Y$, where X, Y are itemsets. There are different scenarios in which we may want to extract association rules. ***Frequent itemset***: we have transactions of itemsets and we look for frequent subsets (we go to a shop and buy A, what do we also buy?). ***Frequent sequences***: we have sequences of items (we have DNA sequences and we look for frequent patterns inside the sequences ($\neq$ from before because a sequence is not just an itemset)).
  Sequential pattern mining: we have sequences of itemsets (every element in this sequence can be a group of elements (itemsets $\neq$ items)) and we're mining frequently occurring ordered events.
- **Evaluation metrics**
  $\rightarrow$ ***Support***: $s = \frac{\sigma(\{X,Y\})}{\#transactions} \sim \mathbb{P}(X,Y)$
  equal to how often X and Y happen together.
  $\rightarrow$ ***Confidence***: $c = \frac{\sigma(\{X,Y\})}{\sigma\{X\}} \sim \mathbb{P}(Y|X)$
  how often Y appears in transactions containing X.
  Tresholds: support must guarantee that X and Y go togheter at least $-\%$ of the times, confidence must guarantee that Y is taken at least $-\%$ of the times when X is taken:
  support $\geq$ minsup treshold
  confidence $\geq$ minconf treshold
  An itemset has a fixed support, its confidence varies based on the association rule: $\{A, B, C\}$ has a fixed support, the confidence of $\{A\} \Rightarrow \{B, C\}$ might be different from the confidence of $\{A, B\} \Rightarrow \{C\}$.
- **Mining association rules**: two-steps approach
  $\rightarrow$ *Frequent itemset generation*: generate all itemsets whose support $\geq$ minsup
  $\rightarrow$ *Rule generation*: generate high confidence rules from frequent itemsets (each rule is a binary partitioning of a frequent itemset)
- **Frequent itemset mining**
  Find, among 2^d itemsets, those with s $\geq$ minsup.
  $\rightarrow$ ***Brute force method***: generation of all possible itemsets. Given d items (overall number of items) there are 2^d possible candidate frequent itemsets. If we have N transactions and the longest (with more items) has w elements, the complexity is $\mathcal{O}(2^d N w)$.
  $\rightarrow$ ***Apriori***. Apriori principle: if an itemset is frequent, then all of its subsets must also be frequent: $\forall X, Y : (X \subseteq Y) \Rightarrow s(X) \geq s(Y)$, the support of an itemset never exceeds the support of its subsets. As soon as we find that an itemset is not frequent, all *bigger*-itemsets that contains it won't be considered (if $\{A\}$ is not frequent, $\{A, B\}$ cannot be frequent).
  1. Set the minimum support (minsup) and k=1
  2. Generate all possible k-dim itemsets: if k>1 consider only frequent itemsets from step k-1
  3. Compute the support for all the k-dim itemsets
  4. Prune the itemsets with support < minsup
  5. If there are no new frequent itemset the procedure ends, otherwise k++ and go to 2.

  It's a level-wise search based on: every itemset which contains a non-frequent itemset cannot be frequent.
  $\rightarrow$ ***Eclat algorithm***: still based on the apriori principle but, due to the vertical databse, it results faster.
  $\rightarrow$ ***FP-tree***: usage of a tree structure to summarize all the data without generating candidates. Benefits: the FP-tree structure is complete and compact.
  1. Sort the single items based on their support
  2. Reorder each transaction based on 1.
  3. Add each transaction one by one into the FP-tree and keep track of the count at each node
  4. For each item i
     $\rightarrow$ compute the conditional FP-tree: seach all the paths that end with i, build the conditional FP-tree considering the count of i in the paths
     $\rightarrow$ based on the conditional FP-tree of item i and considering the minimum support, generate all the frequent itemsets containing i
- **Rule mining**
  Find, among $2^k - 2$ possible rules for itemset X (k number of items in X), those with c $\geq$ minconf.
  Confidence is anti-monotone w.r.t. the number of items on the right-side of the rule: it's more difficult to predict the presence of two product instead of one. $c(ABC \Rightarrow D) \geq c(A \Rightarrow BCD)$. If we find a rule with low confidence level, every derived rule won't be considered (if $BCD \Rightarrow A$ has low confidence, $BC \Rightarrow AD$ cannot have high confidence).
  Rule: $head \Rightarrow tail$. The confidence is the percentage of times the tail appears when there is the head.
  Lift: $lift(X \Rightarrow Y) = \frac{conf(X \Rightarrow Y)}{sup(Y)} \sim \frac{\mathbb{P}(Y|X)}{\mathbb{P}(Y)}$.
  It's the ratio of observed joint probability of (X, Y) and expected joint probability of (X, Y) if they were statistically independent. It's a measure of the deviation from stochastic independence. It also measure the surprise of the rule. We tipically look for lift $\gg 1$ (above expectation) or lift $\ll 1$ (below expectation).
- **Summarize itemsets**
  $\rightarrow$ ***Maximal frequent itemsets***: X is a maximal frequent itemset if it has no frequent super-sets. X is the largest frequent itemset containing X's items.
  $\rightarrow$ ***Minimal generators***: X is a minimal generator if it has no subsets with the same support. All the subsets of X have strictly larger support than X's.
  $\rightarrow$ ***Closed frequent itemsets***: X is closed if all the super-sets of X have strictly less support. Every itemset containing X has strictly smaller support.
- **Trawling**: searching for small communities
  1. Graph to database: for every node we create an itemset as all the nodes the node is pointing
  2. Database to frequent itemsets (any method)
  3. Frequent itemsets to subgraphs (fully connected bipartite graphs): for any frequent itemsets put the nodes of the itemset on the right and put on the left only the nodes which are connected to *all* the itemset-nodes

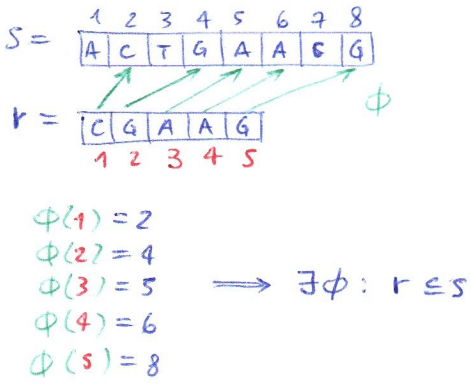
- **Mining frequent sequences**
  A sequence is an ordered list of *items*. We say that $r = r_1, .., r_m$ is a subsequence of $s = s_1, .., s_n$ if there is a strictly increasing function ϕ from $[1, m]$ to $[1, n]$ such that $r[i] = s[\phi(i)]$. Given a database of N sequences s_i, the support of a sequence r is the number of the sequences s_i for which $r \subseteq s_i$. The sequence r is frequent if $\frac{sup(r)}{N} \geq$ minsup.

  GSP algorithm: equivalent to the apriori/Eclat algorithm for frequent itemset miningm, with the difference that we have to consider the order (if A and B are frequent sequences we create both AB and BA).

- **Mining sequential patterns**
  A sequence is now an ordered list of *itemsets*, denoted as: $a =< a_1, .., a_n >$, with a_i itemset: $a_i = x_1, .., x_k$. The size of a sequence is #itemsets (n), the length is #items (k). Not every itemsets must be of length k. A sequence r is contained in s if $\forall r_i \; \exists s_j$: $r_i \subseteq s_j$: $<\{3\}, \{4, 5\}, \{8\}) > \subseteq <\{3, 7\}, \{4, 5, 8\}, \{3, 8\}>$, however $<\{3\}, \{8\}> \nsubseteq <\{3, 8\}>$.

  Sequential patterns: level-wise procedure, the apriori principle still applies (if a sequence is not frequent, then any super-sequence of it cannot be frequent). However, the apriori algorithm leads to complex situations, we prefer the FP-tree approach.

- **Association rules for classification**
  An association rule, $X \Rightarrow Y$ (if we buy X we buy also Y), can be used for classification if we consider X as a set of features and Y the label. To procede we apply one-hot-encoding at the dataset and we focus on finding rules: $X \Rightarrow c_i$, where c_i is the class label.



2. Clustering

It is the search for grouping/structure in un-labeled data. Clustering can be applied to any type of data. Clustering algorithms group a collection of data points into clusters according to some distance measures. A good clustering consists in high intra-class and low inter-class similarities.

- **Data matrix, disssimilarity matrix**
- **Distance measures**: Euclidean, Manhattan (city-block), Jaccard, Hamming, cosine, Edit
- **Normalization (!)**: attributes are measured on different scales, always normalize.
- **Curse of dimensionality**: in high dimensions almost all pairs of points are equally far away.

Hierarchical clustering

It doens't provide a solution, it provides a set of solutions.

- **Agglomerative/divisive hierarchical clustering**: produces a hierarchy (dendrogram) of nested clusters that can be analyzed and visualized.

  1. Compute proximity matrix of pairwise distances between all the points
  2. Let each data point be a cluster
  3. Merge the two closest clusters
  4. Update proximity matrix
  5. If there is a single cluster the procedure ends, otherwise go to 3.

- **Distance between clusters**: single linkage, complete linkage, mean/centroid distance, group average. Different distances lead to different solutions.
- **Number of clusters**: data points in the same cluster should have a small distance from one another (cohesion), data points in different clusters should be at large distance from one another (separation).

  To decide we rely on internal validation measures, which are criteria derived from data.

  → ***Cohesion***: $WSS(C) = \sum_k \sum_i d(x_i, \mu_k)^2$
  where μ_k is the centroid of cluster C_k.
  It measures how closely related are object in a cluster.

  → ***Separation***: $BSS(C) = \sum_k |C_k| d(\mu_k, \mu)^2$
  where μ is the centroid of the whole dataset.
  It measures how well separated are clusters. from other clusters

  Elbow/knee analysis: plot the WSS and BSS for every clustering (change the number of clusters but maintain the other settings) and look for a knee/elbow.
- [N] Number of clusters: inconsistency, acceleration
- **Conclusions**. A plus is that we don't have to choose the number of clusters in advance. However: every decision cannot be undone, no objective function is minimized, it can be sensitive to noise/outliers, it cannot handle different sized or non-convex shapes.

Representative-based clustering

Given a dataset of N instances and a desired number of clusters k, this class of algorithms generates a partition C of N in k clusters. For each cluster there is a point that summarizes it: the centroid. We want to avoid the brute-force approach (generating all possible clusterings and selecting the best one).

- **k-means**
  Greedy iterative approach which aims to find a clustering that minimizes the SSE objective:
  $SSE(C) = \sum_k \sum_i ||x_i - \mu_k||^2$.

  1. (Given k) Randomly initialize k centroids
  2. Assign each point to the closest centroid
  3. Update centroids (as mean of their *new* clusters)
  4. If stopping conditions are met the procedure ends, otherwise go back to 2.

  It converges to a local optimal instead of a global optimal. The algorithm is very sensitive w.r.t. the initialization of the centroids. Moreover, it is shaped-biased (biased towards classes that are n-dim spheres).

  Centroid initialization. We may pick points that are as far away from one another as possible. Or, we can rely on hierarchical clustering: we cut at k and we consider as centroids the mean of the clusters.

  Bisecting k-means (centroids initialization): start from 2 clusters, take the worst (the less cohese) and add a seed. The procedure goes on until k centroids.

  Pre/post-processing. It is very important to normalize the data and eliminate outliers. Moreover we should eliminate small clusters (they may represent outliers) merging them with the closest ones.

  Conclusions. It is very efficient and fast. However: applicable only when mean is defined, need to specify k (we may perform an eblow-knee analysis using an internal measure of performance), not suitable for noisy data and for non-convex shape clusters.
- [N] Demonstration of k-means assumptions
- **Mean shift clustering**
  Iterative nonparametric algorithm that searches for the modes (points of highest density) of a data distribution. It's not really a clustering procedure, it finds points with maximal density. However, it can be adapted to obtain clusters: trajectories that lead to the same mode should be in the same cluster.

  1. Choose a search window size (bandwidth)
  2. For each point of the dataset:
     a. center the search window at the point
     b. compute the mean of the search window
     c. center the search window at the mean (*b.*)
     d. if stoppings conditions are met (the shift was small) go to the next point, otherwise go to *b.*
  3. Clustering step: assign points that lead to nearby modes to the same cluster

  When we evaluate the mean (search window's center of mass) we can use a kernel density estimation.

  Conclusions. No assumptions about the number of clusters. Moreover, since mean shift is based on density estimation, arbitrary shaped clusters can be found. Another plus is that the window size can have a physical meaning. However: the window size selection is not trivial, inappropriate window size can cause modes to be merged or generate additional "false" modes, it is not suitable for high-dim features.

- **Expectation maximization (EM) clustering**
  It considers point's membership to a cluster as a probability. Each cluster is characterized by a multivariate normal distribution: $\vec{\theta_i} = \{\vec{\mu_i}, \Sigma_i, \mathbb{P}(C_i)\}$, where $\mathbb{P}(C_i)$ is the prior probability of the cluster i.

  1. Initialize the estimate of the parameter vectors
  2. For each cluster, use the current estimate of the parameters to compute the posterior probabilities: $\mathbb{P}(C_i|\vec{x}_j) = \mathbb{P}(\vec{x}_j \in C_i)$
  3. Update for all i: $\vec{\mu}_i, \Sigma_i, \mathbb{P}(C_i)$
  4. If stopping conditions are met the procedure ends, otherwise go back to 2.

  EM clustering returns probabilities, it does not assign points to clusters. We can assign points to the cluster with the highest posterior probability.

- **Density-based methods**
  The previous clustering methods (except for mean shift clustering) are suitable for finding ellipsoidal-shaped clusters, at best convex. Density-based methods can mine non-convex clusters. These methods can discover arbitrary shapes, handle noise and perform just one scan. The user defines the of density.

  → ***DBSCAN***. We define the ***ϵ-neighborhood*** of x as $N_\epsilon(x) = \{y|d(x,y) \leq \epsilon\}$. Moreover we define:

  - ***core point***: x core point if $|N_\epsilon(x)| \geq$ minpts. These are points in a dense area.
  - ***border point***: x border point if $|N_\epsilon(x)| <$ minpts, but x is inside the neighborhood of a core point. These are points nearby a dense area.
  - ***noise point***: x noise point if $|N_\epsilon(x)| <$ minpts and it is not in the neighborhood of a core point. These are isolated points.

  Given this classification, we connect dense areas:

  - x is *directly density-reachable* from y if y is a core point and $x \in N_\epsilon(y)$
  - x is *density-reachable* from y if there is a chain of points $x_1(=x), .., x_n(=y)$ such that x_{i+1} is directly density-reachable from x_i
  - x is *density-connected* to y w.r..t. ϵ and minpts if there is a point z such that both x and y are density reachable from z
  - a ***density-based cluster*** is defined as a maximal set of density connected points

  The algorithm needs to compute the ϵ-neighborhood for each point. Once it has the neighborhoods, the algorithm needs only a single pass over all the points to find the denisty-based clusters. However, this method fails when applied to data of varying density.

  → ***HDBSCAN***. We define the ***core distance*** of x as the maximum distance of a x to its k-th nearest neighbor, and we denote it by $core_k(x)$. Points in high density areas will have low core distance, points in low density areas will have high core distance.
  The goal of the algorithm is to remap the points in another space where dense points are kept close to each other, while sparse points are pushed away.

  1. Transform the space
  The distance between x, y in the new space will be: $d_{new}(x,y) = \max\{core_k(x), core_k(y), d(x,y)\}$, where $d(\cdot,\cdot)$ is the original distance metrics. This new distance is called mutual reachability distance.

  2. Build the minimum spannin tree
  Based on d_{new} (evaluated for every couple of points) we build the minimum spanning tree: data points are vertices, an edge between x, y has weight equal to $d_{new}(x,y)$, we build the spanning tree one edge at the time by adding the lowest weight edge that connects the current tree to a point not yet in the tree.

  3. Build the cluster hierarchy
  The algorithm converts the minimum spanning tree to a hierarchy of connected components. It sorts the edges of the tree by weight in increasing order (the weight of and edge represents the distance between the vertex of the edge), and then iterates creating a new merged cluster for each edge.

  4. Extracting clusters
  We condense the dendrogram to highlight the difference between mergings. We procede iteratively starting from the biggest cluster. At each split we check the size of the two new clusters: if a cluster has fewer points than a treshold then it's eliminated, otherwise both clusters are maintained. Given the condensed dendrogram we select clusters that last long.

- [N] Connectivity constraints
- [N] Silhouette analysis

3. Linear Regression

Can we predict the value of y from $\vec{x}$?

- **Baseline performance**: mean of y.
- **Simple linear regression**: $y = w_0 + w_1 x$

  The goal of the model is the prediction, not description. We have to introduce two levels of performance: performance while we *build* and performance while we *use* the model. Given N pairs x, y we evaluate the model by computing the ***residual sum of squares***:

  $$RSS(w_0, w_1) = \sum_{i=1}^{N} (y_i - (w_0 + w_1 x_i))^2.$$

  The goal is to find w_0, w_1 that minimize RSS. To do so, we use the ***gradient descent*** method:

  $$\vec{w}^{(t+1)} = \vec{w}^{(t)} - \eta \nabla RSS(\vec{w}^{(t)})$$

  where η is the learning rate (usually $\eta = \frac{1}{t}$ / $\eta = \frac{1}{\sqrt{t}}$).

- **Multiple linear regression**: $y = w_0 + \sum_{j=1}^{D} w_j h_j(\vec{x})$

  The goal is unchanged, but now we consider D input variables. The model cost is computed again as:

  $$RSS(\vec{w}) = \sum_{i=1}^{N} (y_i - w_0 - \sum_{j=1}^{D} w_j h_j(\vec{x}_i))^2$$

  where $h_j(\vec{x}_i)$ represents the preprocessed input.
  To minimize RSS we apply ***gradient descent***.
  To measure to goodness of fit of the regression line we define the ***coefficient of determination***:

  $$R^2 = 1 - \frac{RSS}{\sum_{i=1}^{N} (y_i - \bar{y})^2} := 1 - \frac{RSS}{TSS}$$

  where $\bar{y}$ is the average of all y_i and TSS is the total sum of squares. TSS corresponds to RSS in the case of the baseline (average). R^2 measures how well the regression line approximates the real data points. If R^2=1 then RSS=0, the regression perfectly fits the data. If R^2=0 then RSS=TSS, which means that the model performs like the average. R^2 can also be negative: in that case RSS>TSS, which means that the model performs worse than the simple average.

- **Model evaluation**
  Model should be evaluated using data that have not been used to build the model itself. We divide the dataset in training data, used to build the model, and test data, used to evaluate the model performance.

  → ***Hold-out evaluation*** reserves a certain amount of data for testing and uses the remainder for training. Trade-off: small training leads to poor weight estimation, small test leads to poor estimation in the future performance. Usually is $\frac{2}{3}$ for training and $\frac{1}{3}$ for testing, or half-half. This approach is still biased by the initial division of the dataset.

  → ***Cross-validation*** uses every point for train and test. Data is split into k subsets of equal size. Each subset in turn is used for testing and the remainder for training. Often the subsets are stratified before cross-validation is performed (we want each fold tobe as representative of the whole as possible). The overall error is the average of the k errors.

  Note: cross-validation provides an evaluation of model performance on unknown data, the output is the evaluation, not the model. The model should be obtained by running the algorithm on the entire dataset.

- **Overfitting and regularization**
  We overfit when we have a good performance on the training set but terrible performance on the test. In regression overfitting is associated to large weights estimates. To avoid this we can add to the usual cost (RSS) a term that penalizes large weights.

  → ***Ridge regression*** *(L_2 reg.)*

  $$cost(\vec{w}) = RSS(\vec{w}) + \alpha \sum_{j=1}^{D} w_j^2$$

  → ***Lasso regression*** *(L_1 reg.)*

  $$cost(\vec{w}) = RSS(\vec{w}) + \alpha \sum_{j=1} |w_j|$$

  If α is zero, the cost is the same as before. If α is infinite the only solution is to have all the weights null. As α increases ($\neq \infty$): for ridge none of the weights go to zero, for lasso instead almost all the weights are forced to go to zero. Lasso tends to zero-out less important features and produces sparser solutions (by penalizing weights it also performs feature selection).

- **Hyperparameter optimization**
  To select the hyperparameters we cannot use the test set since it is going to be used for evaluating the final model. We need to reserve a part of the training data to evaluate possible candidates for the hyperparams and to select the best ones. If we have ***enough data*** we can extract a validation set from the training data and use it to determine hyperparams. If instead data are ***not enough*** we can apply a k-fold cross-validation on the training data and at the end we set the hyperparams as the average of the k optimal ones. There is also an *overkill* option: nested cross-validation (cross-validation of cross-validation). This is an extremely expensive procedure. We divide the whole dataset in k_1 folds. At each step: (1) we consider one fold as test set, (2) the remaining folds are merged and we perform an inner cross-validation (we split in k_2 folds, we use one as validation set and the others as training).

- [N] Features selection: reduced variance, univariate feature selection, PCA, embedded approaches, recursive reature selection, random forest, wrapper approach (hill climbing, genetic algorithms)

  wrapper vs. filter ?

4. Classification

Given a point $\vec{x}$, can we predict its label?

- **Baseline performance**: majority voting.
- **Classification**
  The target to predict is a label/class. We have to compute the model using known data and perform on unseen data.
- **Logistic regression**
  Logistic regression starts from the linear classifier. The idea is to build an hyperplane to separate points based on their labels. We define a ***score function***:

  $$Score(\vec{x}_i) = \sum_{j=1}^{D} w_j\, h_j(\vec{x}_j)$$

  where $h_j(\vec{x}_i)$ is the preprocessed input.
  In the case of linear classifier, the label of a point is determined by the sign of the score value:

  $$\hat{y}_i = sign(Score(\vec{x}_i))$$

  Instead of deterministically computing the label, logistic regression computes the probability of assigning a class to the point:

  $$\mathbb{P}(\hat{y}_i = +1|\vec{x}_i) = \frac{1}{1+e^{-Score(\vec{x}_i)}} = \frac{1}{1+e^{-\vec{w}\,\vec{h}(\vec{x}_i)}}$$

  To optimize, logistic regression searches for weights that correspond to the highst likelihood (it aims to maximize the product of all the probabilities):

  $$l(\vec{w}) = \prod_{i=1}^{N} \mathbb{P}(y_i|\vec{x}_i, \vec{w}_i)$$

  For this purpose, it is convenient to use the log likelihood and to apply the ***gradient ascent***.

  Considering that $\mathbb{P}(\hat{y}_i{=}-1|\vec{x}_i) = 1 - \mathbb{P}(\hat{y}_i{=}+1|\vec{x}_i)$, to classify a point we select the class with the highest probability. Equivalently, we can check the ratio of the probabilities and assign +1 if:

  $$\frac{\mathbb{P}(\hat{y}_i=+1|\vec{x}_i)}{\mathbb{P}(\hat{y}_i=-1|\vec{x}_i)} = e^{\vec{w}\,\vec{h}(\vec{x}_i)} > 1$$

  Or, we can check the log ratio and assign +1 if:

  $$\log\left(\frac{\mathbb{P}(\hat{y}_i=+1|\vec{x}_i)}{\mathbb{P}(\hat{y}_i=-1|\vec{x}_i)}\right) = \vec{w}\,\vec{h}(\vec{x}_i) > 0$$

  We're back to a linear classification rule, meaning that logistic regression is a linear classifier.

  Overfitting and regularization. Also logistic regression has to deal with overfitting problems and also here we can apply L_1/L_2 regularization. This time the optimization problem is a maximization problem (in fact, we use gradient ascent), so the penalization will have a negative sign. The L_1 regularization changes the objective in:

  $$l(\vec{w}) - \alpha \sum_{j=1}^{D} |w|$$

  while L_2 regularization changes the objective in:

  $$l(\vec{w}) - \alpha \sum_{j=1}^{D} w_j^2$$

- **Multiclass classification**
  Logistic regression assumes that there are only two class values. If we have more class values we can either use a one-versus-others approach, or a multilclass model. In the one-versus-others, for each class we create one classifier that predicts the target class against all the others. In the multiclass logistic regression we use the softmax function:

  $$\mathbb{P}(y_i = k|\vec{x}_i) = \frac{e^{\vec{w}_k\,\vec{h}_k(\vec{x}_i)}}{\sum_{k=1}^{K} e^{\vec{w}_k\,\vec{h}(\vec{x}_i)}}$$

  using a weight vector for each class.
- **Classification metrics**: confusion matrix, accuracy, cost of classification, precision, recall, F1-measure, sensitivity, specificity.
- **Models comparison**
  To choose one model over another we should have a statistically significant difference in the performance, for example in accuracy. We can apply the t-test and compute the p-value.

  1. Generate k folds for each model: $\theta_1^A, .., \theta_k^A, \theta_1^B, .., \theta_k^B$
  2. Compute differences, mean and std deviation:
     $\delta_i = \theta_i^A - \theta_i^B$, $\mu_\delta = \frac{1}{k}\sum_i \delta_i$, $\sigma_\delta = \sqrt{\frac{1}{k}\sum_i (\delta_i - \mu_\delta)^2}$
  3. Set a confidence level α and test:
     $H_0 : \mu_\delta = 0$ vs. $H_1 : \mu_\delta \neq 0$
     if p-value $> \alpha$ the difference in performance is not statistically significant (we do not reject H_0)

  The p-value represents the probability that the reported difference is due to chance. The procedure is valid if the two models share the same data (paired t-test) and also if they don't (unpaired t-test).

  Note: if we run different tests we have to apply the Bonferroni correction. Assuming that we perform n independent tests, to have an overall confidence of level α, for each individual test we have to use $\frac{\alpha}{n}$.
- **Probabilistic classifiers and tresholds**
  Logistic regression returns probabilities $\mathbb{P}(y_i|\vec{x}_i)$. To classify a point we can select the class with the highest probability. This is equivalent to use a treshold of 0.5. What if use other tresholds?

  - Treshold close to 1 $\rightarrow$ pessimistic classifier.
    We will classify as positive only points for which we are very confident. In this case the precision will be high, recall will be low.
  - Treshold close to 0 $\rightarrow$ optimistic classifier.
    We will classify as positive almost everything. Precision will be low, recall will be high.

  Given a model we can vary its treshold and collect the pair [*recall, precision*] for every treshold. We can plot the pairs, obtaining the precision-recall curve. To compare models we can look at the the areas under the respective precision-recall curves: the higher the area the better the mode.

  ROC and AUC. For every treshold we can collect pairs of [*false positive, sensitivity*]. We can plot the pairs, obtaining the ROC curve. The ideal point is [0,1]. If we choose the treshold that corresponds to the point [0,0] we're declearing everything to be negative, viceversa if we choose the treshold corresponding to [1,1] everything will be positive. If a point is under the diagonal means that by switching all the labels in the classification we obtain a better classifier. To compare models we look for the area under the curve (AUC). The ideal area is 1.
- **Other classificatin methods/tools**

  Support vector machines. SVM works by searching the hyperplane that maximizes the margin between classes: largest γ such that $y_i(wx_i + b) \geq \gamma$.

  Neural networks. It passes the sum of weighted inputs to an activation function. Labels assignment depends on the output of the activation function.

Naive Bayes

- **Naive Bayes classifiers: $\vec{x}$ categorical**
  Given $\vec{x}$ we look for the class with highest probability:

  $$class = \arg\max_y \mathbb{P}(y|\vec{x})$$

  Given the targett y and the point $\vec{x}$ described by n attributes, the Bayes theorem says that:

  $$\mathbb{P}(y|\vec{x}) = \frac{\mathbb{P}(\vec{x}|y)\,\mathbb{P}(y)}{\mathbb{P}(\vec{x})}$$

  Naive Bayes classifiers assume that attributes are statistically independent, and so we can write:

  $$\mathbb{P}(y|\vec{x}) = \frac{\mathbb{P}(x_1|y)\,..\mathbb{P}(x_n|y)\,\mathbb{P}(y)}{\mathbb{P}(\vec{x})}$$

  To perform the ***training*** we count the frequency of touples (x_i, y) for each attribute value x_i and each class value y in the dataset. Then we use counts to compute estimates for the class probability $\mathbb{P}(y)$ and the conditional probability $\mathbb{P}(x_i|y)$. To ***test***, given a new point $\vec{x}$ we compute the most likely class as:

  $$\begin{aligned} class &= \arg\max_y \mathbb{P}(y|\vec{x}) \\ &= \arg\max_y \mathbb{P}(x_1|y)\,..\,\mathbb{P}(x_n|y)\,\mathbb{P}(y) \end{aligned}$$

  We are making two strong assumptions: attributes are equally important and attributes are statistically independent (knowing the value of the attribute x_j says nothing about the value of the attribute x_i if the class y is known: $\mathbb{P}(x_i|x_j, y) = \mathbb{P}(x_i|y)$).

- **Zero-frequency problem: Laplace estimator**
  If an attribute value does not occur with every class value the aposteriori probability of that class will be zero. To avoid, the Laplace estimator method adds 1 everywhere (on variables, not targets).

- **Missing values**
  Naive Bayes has a way to deal with missing values. In the training phase, instances with missing values are not included in frequency counts for attribute value-class combination. In the testing phase, the attribute with missing value is omitted from calculation.

  Note: in logistic regression we cannot deal with missing values, we can work on the instances with missing value before the algorithm, but not during.

- **Naive Bayes classifiers: $\vec{x}$ numerical**
  If the j-th attribute of $\vec{x}$ is numerical we can discretize it. Or, we can compute a probability density for each class, assuming a parametric form for the distribution and estimate its parameters. Tipically, we assume the attributes to come from a gaussian:

  $$\sim \mathcal{N}(\mu, \sigma^2) = N(\tfrac{1}{n}\textstyle\sum_i x_i,\ \tfrac{1}{n-1}\sum_i (x_i - \mu)^2)$$

  For a new point $\vec{x}$, the most likely class is:

  $$\begin{aligned} class &= \arg\max_y \mathbb{P}(y|\vec{x}) \\ &= \arg\max_y \mathbb{P}(x_1|y)\,..\,f(x_j|y)\,..\,\mathbb{P}(x_n|y)\,\mathbb{P}(y) \end{aligned}$$

- **Bayesian belief networks**
  The assumption of independence among variables is really strong: attributes are often correlated. Bayesian belief networks allow us to specify which pair of attributes are conditionally (in)dependent.

  The key elements are: a ***direct acyclic graph*** which encodes the dependences among variables, a set of ***probability tables*** which associates nodes to their immediate parents nodes.

  Each node is associated with a probability table: if a node X has no parents then its probability table contains only the prior probability $\mathbb{P}(X)$, if a node has parents $Y_1, .., Y_n$ then its probability table contains the conditional probability $\mathbb{P}(X|Y_1, .., Y_n)$.

  The goal is to estimate the probabilites of unknown variables given the known.

  Note: there is not a real concept of class, we're trying to model the entire distribution of the data.

k-Nearest neighbors

- **k-Nearest neighbors classifier**
  To classify a point $\vec{x}$: select the k most similar points to $\vec{x}$ in the training set and assign to $\vec{x}$ the most frequent class among those. No actual model is computed: we use the training records to predict an unknown class label. The elements are: the training dataset, the similarity function and the value of k. To classify a point:

  1. Compute distance to other training records
  2. Identify the k nearest neighbors
  3. Use labels of nearest neighbors to determine the label of the point (majority voting/others)

  How many neighbors? Hyperparameter (cross-validation changing k at every iteration or nested crossvalidation). If k is too small the classification might be too sensitive. If k is too large the neighborhood may include quite dissimilar points.

  Similarity measures? Clustering ones.

- **Efficient nearest neighbors**

  → ***KD-trees***: randomly partitionate the space (generate a random point and from it split in one direction) and derive the partitioning tree. To find the neighbor of a specific instance navigate the tree to reach the specific leaf, then go back to check nearby regions until k nearest neighbors are found.
  A limitation of this method are the corners.

  → ***Ball trees***: variation of KD-trees where regions are balls (hyperspheres) instead of hyperrectangles. A ball-tree organizes data into a tree of k-dim hyperspheres. Balls can overlap.

Decision trees

- **Splitting criteria**
  A decision tree is a structure that allows to make a decision. At each node one attribute is chosen to split training instances into distinct classes. A new point is classified following a matching path to a leaf node.

  A top-down approach creates the tree (initially all training instances are at the root, then they're recursively partitioned by choosing one attribute at the time). A bottom-up approach prunes the tree to generalize the structure and avoid overfitting.

  Splitting attribute. We want to separate points to create areas that are pure and contain mainly points of one class. At each node available attributes are evaluated based on separating the classes using a purity or impurity measure. Typical measures used are the entropy, the information gain (ID3), information gain ratio (C4.5), gini index (CART).

- **Overfitting**: pre-pruning, post-pruning
  If we have too many branches, some may reflect anomalies due to noise or outliers, which leads to poor accuracy for unseen samples. To avoid over complex/ overfitted trees, we can pre-prune (stop before) or post-prune (remove branches from a fully grown tree).

  Post-prune. Once the tree is fully grown, we go-bottom up and we reconsider our choices: was a split worth to create? If no, we merge together again the groups. We determine if a split is worth statistically. We consider one split. For the father and the childrend nodes we calculate: the error f on training data (percentage of missclassified in the node) and the upper bound for the error estimate e of the node:

  $$e = (f + \frac{z^2}{2N} + z\sqrt{\frac{f}{N} - \frac{f^2}{N} + \frac{z^2}{4N^2}}) \ / \ (1 + \frac{z^2}{N})$$

  we choose the confidence level $c = 25\%$, so $z = 0.69$. We prune only if it reduces the estimated error: if the weighted average of children's e's is higher than father's e we prune.

- **Decision trees for regression**
  Decision trees can also be used to predict the value of a numerical target variable. Prediction is computed as the average of numerical target variable in the subspace (regression trees). Alternatively, leafs can contain a linear model to predict the target value in the corresponding subspace (model trees). An impurity measure is the standard deviation reduction:

  $$SDR = \sigma(D) - \sum_i \frac{|D_i|}{|D|}\sigma(D_i)$$

  where D is the original data, D_i the partitions and $\sigma(\cdot)$ the standard deviation of the target in the set.

Algorithm/model selection

- **Comparison**
  If the most effective algorithm is:
  - *logistic regression* $\rightarrow$ points are separable
  - *k-NN* $\rightarrow$ locality is more important
  - *naive bayes* $\rightarrow$ the hypothesis that variables are $\perp\!\!\!\perp$ is not bad, also, the probabilistic view of the data is more effective than the geometrical one (logistic regression) or locality (k-NN)

- **No free lunch theorem**: there are no context/usage independent reasons to favor one method over anothe, if one algorithm outperforms another in a certain situation it is a consequence of a particular problem, not the general superiority of the algorithm.
- **Occam's Razor**: the best theory is the smallest one that describes all the facts.

5. Ensemble Methods

- **Baseline performances**
  We generate a set of models from the training data and we predict the labels of unseen cases by aggregating predictions: majority voting for classification, averaging for regression.

- **Bagging (bootrsap aggregation)**
  How can we generate different models using the same data and the same approach? Models need to be independent, so it makes no sense to split the data. Bagging method samples with replacement for every new model. It works because it reduces the variance by voting/averaging. The more models the better.

  1. Consider a dataset of D touples
  2. At iteration i sample with raplcement from D a training set D_i of d touples (bootstrap)
  3. Learn a model M_i for each training set D_i
  4. Return the target prediction for each model M_i
  5. The bagged model M^* returns as final result the majority in case of classification or the average in case of regression

  Bagging helps with *unstable* algorithms (algorithms for which small changes in the training set cause large changes in the learned models). Bagging in case of *stable* algorithms is not a good idea.

- **Random forests**
  Random forest are ensembles of unpruned decision tree learners with randomized selection of features at each split. The generalization error of a forest depends on the strength of the individual trees and the correlation between them.

  1. Consider a dataset of D touples
  2. At iteration i sample with replacement from D a training set D_i of d touples (bootstrap)
  3. Learn tree T_i from D_i using at each node only a subset of the n variables, without pruning
  4. Return the target prediction for each tree T_i
  5. The output is computed as the majority voting for classification or average for prediction

  Out-of-bag error. For each observation construct the random forest predictor by considering only the trees corresponding to bootrap samples in which the observation did not appear. The OOB error is almost equal to the error obtained by n-fold cross-validation.

- **Boosting** – *focus on missclassified points*
  The idea of boosting is to iteratively create models that try to solve mistakes made by the previous models. We obtain a sequence of models.

  Adaboost. It applies to classification with two classes $(-1/+1)$. Adaboost computes strong classifiers as a combination of weak classifiers.

  1. Assign uniform weights to each training sample
  2. At iteration i learn a weak classifier h_i
  3. Compute the error $\epsilon_i = \sum_j w_j \mathbb{I}_{\{h_i(x_j) \neq y_j\}}$
  4. Compute $\alpha_i = \frac{1}{2} \ln(\frac{1-\epsilon_i}{\epsilon_i})$
  5. Update the weights:
     $$w_{i+1} = w_i e^{-\alpha_i} \quad \text{correctly classified}$$
     $$= w_i e^{\alpha_i} \quad \text{incorrectly classified}$$
     and normalize them
  6. The final model is:
     $$H(x) = sign(\sum_{t=1}^{T} \alpha_t h_t(x)) \in \{-1, +1\}$$

  Note. In bootsting by ***sampling*** the weights are used to sample the data for training, in boosting by ***weighting*** the weights are used by the learning algorithm

- **Gradient boosting** – *focus on target and residuals*
  We build a model to predict the target (regression or classification) and then we look at the ***residuals***, i.e. difference between the target function and the prediction. A second model is focusing on predicting the residual. Then, the final prediction is computed as the sum of the predictions of the two models.

  1. Learn a basic (even simple) predictor
  2. Compute the gradient of a loss function w.r.t. the predictor, for instance the mean square error
     $$MSE(y, \hat{y}) = \frac{1}{N} \sum_i (y_i - \hat{y})^2$$
  3. Compute a model to predict the residuals
  4. Update the predictor with a new model
     $$\hat{y}_i = \hat{y}_i + \alpha \nabla MSE(y, \hat{y})$$
     where large α means larger steps
  5. If stopping conditions are met the procedure ends, otherwise go to 2.

  The predictor is increasingly accurate and complex.

  XGBoost: efficient and scalable implementation of gradient boosting applied to classification and regression trees. It only deals with numerical variables.

- **Stacking (stacked generalization)**
  The previous ensambles considered homogeneous models (they combined all trees/ all logistic regressions/..). We can train a learning algorithm to combine the predictions of a heterogeneous set of models.

  First, a set of base learners are trained over the data. These learners are generated from different learning schemes (in this way we capture different regularities). Then, a meta learner is trained using the predictions of base models as additional inputs.

- [N] Comparing classifiers using statistical tests
- [N] Variable importance
- [N] Voting classifier

6. Text mining

Mine informations from unstructured text documents.

- **Informations retrieval**
  We have a set of documents, the corpus, and we want to find the ones related to a particular topic. Typical issues are: management of unstructured documents, approximate search, measure of relevance.

  Text retrieval methods. We can use a document selection approach, where a query defines a set of requisites and only documents which satisfy the query are returned. Alternatively, we can use a document ranking method, where documents are ranked on the basis of their releveance w.r.t. the user query.

  $$precision = \frac{|\{Relevant\} \cap \{Retrieved\}|}{\{Retrieved\}}$$

  $$recall = \frac{|\{Relevant\} \cap \{Retrieved\}|}{\{Relevant\}}$$

  $$F - score = \frac{2 \cdot precision \cdot recall}{precision + recall}$$

- **Documents representation**
  Documents are vectors in high-dim spaces corresponding to all the keywords (bag-of-words representation). Relevance is measured with an appropriate similarity measure defined over the vector space.

  Given two documents $q = q_1, .., q_N$ and $d = d_1, .., d_N$, where q_i and d_i represent words, we use a similarity function $f(q, d)$, which returns a real value, to rank most similar documents.

  Basic similarity function:

  $$f(q, d) = \sum_i q_i \, d_i$$

  where $q_i, d_i \in \{0, 1\}$ if we only consider the presence/absence of a keyword, or $q_i, d_i \in \mathbb{R}$ if we count also the repetition of the matching words.

  Inverse document frequency (IDF):

  $$IDF(w) = \log \frac{M}{k} = \log \frac{M}{df(w)}$$

  the higher the IDF the more important a word (because it appears in fewer docs)

  where M is the overall number of documents and k is the number of documents containing the word w (document frequency of the word w ($df(w)$).

  Combination of term frequency (TF) and IDF:

  $$d_i = c(w_i, d) \log \frac{M}{df(w_i)}$$

  $$q_i = c(w_i, q) \log \frac{M}{df(w_i)}$$

  $$f(q, d) = \sum_{i=1}^{N} q_i \, d_i$$
  $$= \sum_{w \in q \cap d} c(w, q) \, c(w, d) \, (\log \frac{M}{df(w)})^2$$

  where $c(w_i, d)$ is the frequence of the word w_i in the document d (and q in $c(w_i, q)$).
  This similarity function accounts both the frequency of a word and the importance of a frequent word.

  Pivoted length normalization (VSM):

  $$d_i = \frac{\ln(1 + \ln(1 + c(w_i, d)))}{1 - b + b \frac{|d|}{avdl}} \log \frac{M}{df(w_i)}$$

  This similarity function avoids frequency problems considering a non-linear transformation of $c(w_i, d)$ (TF term: $\ln(1 + \ln(..))$), penalizes words that frequently occur in many documents (IDF term) and penalizes also the document lenght ($1 - b + b \frac{|d|}{avdl}$ is a length normalizer ($|d|$ is the doc length and $avdl$ is the average doc length: documents are penalized/rewarded if their length is above/under the average)).

  BM25/Okapi:

  $$d_i = \frac{(k+1) \, c(w_i, d)}{(c(w_i, d) + k) \, (1 - b + b \frac{|d|}{avdl})} \log \frac{M}{df(w_i)}$$

  We have a non-linear transformation of $c(w_i, d)$ that avoids frequency problems, a penalization for words that frequently occur in many documents (IDF), and a lenght penalization.

- **Pre-processing**
  Some useful preliminary steps: remove non-context related informations, lowercase the text, remove *stop words* ("the", "a", "always"), simplify words that share a common prefix.

7. Genetic Algorithms

Given a certain problem and a target function over a domain we want to find the maximum/minimum. If we don't know the analytic form, we have to perform a black-box optimization. The idea of genetic algorithms is to evolve a population of candidate solutions using the concepts of survival of the fitness, variation and inheritance.

Genetic algorithms' procedure is to generate an initial population, select promising solutions from the population, create new solutions applying variations to the selected ones, incorporate the new solutions into the original population and repeat until a stop criterion is met.

1. Initial population
2. Evaluation (given a fitness function)
3. Tournament selection: select a random subset of k solutions from the original population and then select the best solution out of the subset
4. Variation (given two solutions)
   - → One-point crossover
   - → Two-points crossover
   - → Uniform crossover
   - → Bit-flip mutation
5. Replace: replace all, replace worst (elitism)

8. Time Series

- Constitutive sections: trend, pattern, cyclic
- Additive and multiplicative time series
- Percentage of change
- Filling missing data: pad/ffill, backfill/bfil, fill_value
- Forward/backward shift

Modeling

- Correlation between pairs of series
- Autocorrelation function (ACF) + partial autocorrelation
- Baseline performance: random walk (no/with drift)
- Test for random walk: statistical test, Dickey-Fuller
- Stationary and non-stationary series
- Models
  - Autoregressive model: AR(p)
  - Moving average model: MA(q)
  - Combination of AR(p), MA(q): ARMA(p,q)
  - ARMA with independent variables: ARMAX(p,q)
  - AR integrated MA: ARIMA(p,d,q)
- Model selection: partial autocorrelation function (PACF) with ACF, bayesian information criterion (BIC)
- Box-Jenkins method

Forecasting

- Training, test, cross-validation with time series
- Evaluation metrics: single forecast error, mean absolute error (MAE), root mean square error (RMSE), mean absolute percentage error (MAPE)
- Linear regression (or other simple models)