Summary of
Data Mining and Text Mining

Manfred Nesti

8 settembre 2020

Indice

1 Data mining and analysis

1.1 Data representation

Dataset:

- **instances**: rows
- **attributes**: columns
- **concept**: target column

Attributes:

- numerical
- categorical (NO order)
- ordinal categorical
- ratio
- binary (dummy)

1.2 Data exploration

Summary statistics:

- Frequency and mode
- Mean and median
- Percentiles
- Trimean and truncated mean (as IQ mean)
- Range, variance, IQR

Type of graphs:

- Bar plots
- Histograms
- Kernel density estimates
- Boxplots
- Violin plots
- Scatter plots
- Maps
- Scalar fields
- Vector fields

Multi-dimensional data: curse of dimensionality

- Heat maps
- Spider plots
- Chernoff faces

1.3 Data preparation

Missing values:

- **MNAR** (Missing Not At Random): depending on value
- **MAR** (Missing At Random): depending on attribute but NOT on value
- **MCAR** (Missing Completely At Random): NOT depending either on attribute nor value

Missing value handling:

- **List-wise deletion**
- **Pair-wise deletion**
- **Single imputation** (mean, mode, dummy control, regression, ...)
- **DNI** (Do Not Impute)

Outliers:

- **Trimming**: eliminate all
- **Winsorizing**: press at range (as 5th and 95th precentiles)

Normalization:

- **Range normalization**: $x_i' = \frac{x_i - \min_i x_i}{\max_i x_i - \min_i x_i}$
- **Standard score normalization** (z-score): $x_i' = \frac{x_i - \mu}{\sigma}$

Aggregation: combining more attributes into a single attribute to reduce data, change of scale or have more stable data
Sampling:

- simple random sampling
- sampling without replacement
- sampling with replacement
- stratified sampling

Dimensionality reduction:

- **PCA** (Principal Component Analysis)
- **t-SNE** (t Distributed Stochastic Neighbor Embedding)
- **SVD** (Singular Value Decomposition)
- **Feature subset selection**
  - Brute-force
  - Embedded approaches (naturally part of data mining algorithm)
  - Filter approach (independent from a specific algorithm)
  - Wrapper approach

Feature creation: use features to create another more useful
Discretization:

- **supervised**: attributes are discretized using the class information
- **unsupervised**: attributes are discretized solely based on their values

2 Frequent pattern mining

Implication between itemsets: $X \Rightarrow Y$
Rule evaluation metrics:

- **support:** $s(X \Rightarrow Y) = \frac{|X \cup Y|}{\#\text{ transaction}}$
- **confidence:** $c(X \Rightarrow Y) = \frac{|X \cup Y|}{|X|}$

2.1 Frequent itemset mining

We look, though 2^d possibly itemsets (d number of items, at which have $s \geqslant s_{\min}$
Brute-force method: $\mathcal{O}(2^d N d)$

- d: number of elements
- N: number of transaction
- w: maximum length of transactions

A-priori principle: $(X \subset Y) \Rightarrow s(X) \geqslant s(Y)$
A-priori algorithm (Level-wise approach): thanks to a-priori principle uses pruning to limitate the investigated cases 2^d
Vertical database: $t(C)$ is the vector of transactions identifiers (tids) in which C appears

- $t(XY) = t(X) \cap t(Y)$
- $s(X) = |t(XY)|$

Eclat algorithm (Tidset intersection approach): thanks to vertical database avoid to visit the whole list of transaction many times
FP-Growth algorithm (Frequent pattern tree approach): avoid to compute all candidates and repeated scans of entire database

2.2 Rule mining

We look, though $2^k - 2$ possibly rules form the itemset X (k number of items in X), at which have $c \geqslant c_{\min}$
We can use the same algorithms than before
Anti-monotonicity of confidence w.r.t. second itemest: $c(XY \Rightarrow X) \geqslant c(XY \Rightarrow X)$
Lift ratio: $\text{lift}(X \Rightarrow Y) = \frac{s(X \cup Y)}{s(X)s(Y)} = \frac{c(X \Rightarrow Y)}{s(Y)}$ measure of the expectations of the rule
Summarizing itemsets:

- Set of **maximal frequent itemsets:**
  $M = \{X \in F : \nexists Y \supset X, Y \in F\}$
- Set of **closed frequent itemsets:**
  $C = \{X \in F : \forall Y \supset X \; s(Y) < s(X)\}$
- Set of **minimal generators:**
  $G = \{X \in F : \nexists Y \subset X : s(X) = s(Y)\}$

2.3 Mining frequent sequences

Sequence: $s = s_1, \cdots, s_k$, $s(i) = s_i$
Subsequence: $r \subseteq s$ if $\exists \phi : [1, N_r] \to [1, N_s]$ strictly increasing s.t. $r_i = s(\phi(i))$
Consecutive subsequence: $r \subseteq s$ consecutively if
$r(i) = s(j) \; \forall i \in [1, N_r], \; j = i + k, \; k < N_s - N_r$

- Prefix

- Suffix

Support: $s(r) = |\{s_i : r \subseteq s_i\}|$
GSP algorithm: as Eclat algorithm

3 Clustering

Data matrix and **similarity matrix** (symmetric or lower triangular)
Distance measures:

- L^p−norm: $d(x,y) = (\sum_{i=1}^{n} |x_i - y_i|)^{\frac{1}{p}}$
  - $p = 2$: Euclidean distance:
    $d(x,y) = \sqrt{\sum_{i=1}^{n} (x_i - y_i)^2}$
  - $p = 1$: Manhattan distance:
    $d(x,y) = \sum_{i=1}^{n} |x_i - y_i|$
  - $p = \infty$: $d(x,y) = (\sum_{i=1}^{n} |x_i - y_i|)^{\frac{1}{p}}$
- Jaccard distance: $d(x,y) = 1 - SIM(x,y)$ where $SIM(x,y) = \frac{x \cap y}{x \cup y}$
- Cosine distance: $d(x,y) = \arccos \frac{\sum_i^n x_i y_i}{\sqrt{\sum_i^n x_i^2}\sqrt{\sum_i^n y_i^2}}$
- Cosine similarity: $\frac{\sum_i^n x_i y_i}{\sqrt{\sum_i^n x_i^2}\sqrt{\sum_i^n y_i^2}}$
- Edit distance: $d(x,y) = |x| + |y| - 2|LCS(x,y)|$ where LCS is the Longest Common Subsequence
  It is also the smallest number of insertions and deletions off single characters to transform x in y
- Hamming distance: $d(x,y) = \frac{l-m}{m}$ where m is the number of matching components and l the length of x and y

Tools for custom distances:

- **Label Encoder**: target labels with values between 0 and $n-1$
- **One Hot Encoder**: maps each $n-$ value categorical attribute with n different binary attributes
- **Ordinal Encoder**: performs an ordinal (integer) encoding of the categorical features

Requisites for clustering algorithms:

- Scalability
- Adaptivity to different attributes
- Adaptivity to dynamic data
- Discoverability of cluster with arbitrary shape
- Bit requirements for domain knowledge to determine input parameters
- Adaptivity to noise and outliers
- Insensitive to order of input records
- Managing high dimensionality
- Ability to incorporate user-specified constraints
- Intepretability and usability

Curse of dimensionality

3.1 Hierarchical clustering

With a hierarchy shown by a dendogram, it produces (agglomeratively or divisively) some sets of partitions in which we can cluster our set

Computational complexity:

- Time: $\mathcal{O}\left(N^3\right) \rightarrow \mathcal{O}\left(N^2 \ln N\right)$
- Space: $\mathcal{O}\left(N^2\right)$

Agglomerative/divisive algorithm

Centroid: $\mu_i = \frac{1}{|C_i|} \sum_{x_i \in C_i} x_i$

Distances between clusters:

- **Single link** (or MIN):
  $d\left(C_i, C_j\right) = \min_{p,q} d\left(x_{i,p}, x_{j,q}\right)$
- **Complete link** (or MAX):
  $d\left(C_i, C_j\right) = \max_{p,q} d\left(x_{i,p}, x_{j,q}\right)$
- **Mean distance**: $d\left(C_i, C_j\right) = d\left(\mu_i, \mu_j\right)$
- **Group average**: $d\left(C_i, C_j\right) = \text{avg}_{p,q}\, d\left(x_{i,p}, x_{j,q}\right)$

Clustering quality measures

- **Internal validation measures**:
  - **Within-cluster Sum os Squares**:
    $WSS\left(C\right) = \sum_{i=1}^{k} \sum_{x_j \in C_j} d\left(x_j, \mu_i\right)^2$
  - **Between-cluster Sum os Squares**:
    $BSS\left(C\right) = \sum_{i=1}^{k} |C_j| d\left(\mu, \mu_i\right)^2$ with μ centroid of the entire dataset
- **External validation measures**: use a prior or expert-specified knowledge about the clusters or criteria NOT inherent to the dataset
- **Relative validation measures**: are obtained via different parameter settings for the same algorithm and directly compare different solutions through

Silhouette coefficients: $x_i \in C_i$ we have $s\left(x_i\right) = \frac{b(x_i) - a(x_i)}{\max(b(x_i), a(x_i))} \in [-1, 1]$

- $a\left(x_i\right) = \frac{1}{|C_i| - 1} \sum_{j \in C_i, j \neq i} d\left(x_i, x_j\right)$
  mean distance between x_i and all other data points in the same cluster
- $b\left(x_i\right) = \min_{k \neq i} \frac{1}{|C_k|} \sum_{j \in C_k} d\left(x_i, x_j\right)$
  smallest mean distance between x_i and all points in any other clusters

Knee/elbow analysis plotting WSS and BSS for every possible partition

Cluster representation:

- Euclidean spaces: centroid or convex hull
- NON-Euclidean spaces: suitable chosen distance

Strengths:

- NO need assumptions on number of clusters, we can cut the dendogram when we want

Limitations:

- Every decision can NOT be undone
- NO objective function to directly minimize
- Can be sensitive to noise and outliers
- Can difficult handling different sized clusters and convex shapes
- Can break large clusters
- Bad scalability because of $\mathcal{O}\left(N^2\right)$

3.2 Representative-based clustering

3.2.1 k-means

Given the number k of desired clusters, it generate a partition of k clusters in greedy way, minimizing the WSS
Greedy algorithm: every decision can NOT be undone, can converge to a local maximum
Computational complexity: Time: $\mathcal{O}\left(tnkd\right)$

- t: iterations
- n: points
- k: wished cluster
- d: dimensions

Centroid initialization:

- points far away from one another
- hierarchical clustering and pick a point for each cluster
- multiple runs can help
- select more than k centroids and then pick k among the resulting centroids after k-means

Update centroid incrementally: but is more expensive, introduces an order dependency
Pre-processing:

- normalize data
- eliminate outliers

Post-processing:

- eliminate small clusters
- split clusters with high WSS
- merge clusters close and with low WSS

Variants:

- different selection of initial centroids
- different dissimilarity calculations
- different way to compute cluster means
- k-modes for categorical data (with suitable dissimilarity measures and frequency-based method to update modes)
- k-prototype method per mixture categorical-numerical data

Advantages:

- efficiency and fast
- often terminates at a local optimum
- simple and explanable

Limitations:

- need a mean
- need to specify k in advance
- unable to handle noisy and outliers
- unable to handle clusters of different sizes
- unable to handle clusters of different densities
- unable to discover cluster with NON-globular shapes

3.2.2 BFR algorithm

Variant of k-means which assumes that clusters are normally distributed around a centroid and are axists-alligned ellipses
Classes of points:

- **DS** (Discard set): points close enough to a centroid to be summarized
- **CS** (Compression set): groups of points close together but NOT close to any existing centroid, they are summarized bu NOT assigned to a cluster
- **RS** (Retained set): isolated points waiting to be assigned to a compression set

Summarizing DS:

- number of points N
- vector SUM whose component $SUM\,(i)$ is the sum of the coordinates of the points in the $i-$th dimension
- vector $SUMSQ$ whose component $SUMSQ\,(i)$ is the sum of squares of the coordinates of the points in the $i-$th dimension

Advantages:

- desigend to handle large data sets
- points are read from disk one chunck at the time so to it into main memory
- only $2d+1$ values to represent any size cluster
- centroid easy computable as $\frac{SUM(i)}{N}$

3.2.3 Mean shift clustering

Iterative, NON-parametric and versatile algorithm which look for the point of highest density of a data distribution
We assume that the data points are sampled form an underlying PDF
Computational complexity: Time: $\mathcal{O}\left(tN^2\right) \rightarrow \mathcal{O}\left(tN \ln N\right)$

- t: iterations
- N: data

Bandwith: h to be chosen
Kernel density estimation:

- **Uniform kernel**: $K_U(x) = c\ \chi_{\{\|x\| \leq 1\}}$
- **Epanechnikov kernel**: $K_E(x) = c\left(1 - \|x\|^2\right) \chi_{\{\|x\| \leq 1\}}$
- **Normal kernel**: $K_N(x) = c \cdot e^{-\frac{\|x\|^2}{2}}$
- **Gaussian kernel**: $K_G\left(\frac{x - x_i}{h}\right) = \frac{1}{\sqrt{2\pi}} e^{-\frac{(x - x_i)^2}{2h^2}}$

Advantages:

- NOT need to specify a-priori the number of clusters
- handle arbitrarily shaped clusters
- robust to initialization
- less sensitive to outliers than k-means

Limitations:

- h is NOT trivial to choose and the choose is important
- computational expensive in time w.r.t. k-means
- NOT suitable for high-dimensional features

3.2.4 EM (Expectation-Maximization) algorithm

Cluster identification: $\theta = (\mu_i, \Sigma_i, P(C_i))$

- μ_i: mean vector
- Σ_i: covariance matrix
- $P(C_i)$: prior probability of all the clusters C_i

We want to finde θ^* which maximizes the likelihood

3.3 Density-based clustering

Density-based clustering can handle NON-convex clusters
Advantages:

- NOT need to specify a-priori the number of clusters
- can mining NON-convex clusters
- handle noise
- only one scan

Disadvantages:

- need density parameters as termination condition
- need density

3.3.1 DBSCAN algorithm

$\varepsilon-$**neighborhood**: $N_\varepsilon(x) = \{y : d(x,y) \leqslant \varepsilon\}$
Core object: $\varepsilon-$neighborhood with at least n points
Directly density-reachability: Y is directly density-reachable from X if it is within the $\varepsilon-$neighborhood of X and the latter is a core object
Density-reachability: Y is density-reachable from X if there is a chain of objects directly density-reachable from X to Y
Desity-connectivity: Y is density-connected to X if there exists an object Z s.t. both X and Y are density-reachable from Z
Density-based cluster: Maximal set of density-connected points
Basic concepts:

- Density corresponds to have at least n points within a specified radius ε
- A **border point** has fewer than n points within ε, but is in the neighborhood of a core point
- A **noise point** is any point that is NOT a core point NOR a border point

4 Regression

4.1 Normal and multiple regression

Linear regression: $y_i = w_0 + w_1 x_i + \varepsilon_i$
Residual Sum of Squares: $RSS(w_0, w_1) = \sum_{i=1}^{N} \varepsilon_i^2$ to minimize, with N number of samples
Gradient descent: $w^{t+1} = w^t - \eta \nabla RSS(w^t)$
with $\nu = \nu(t) = \frac{\alpha}{t}, \frac{\alpha}{\sqrt{t}}$
Multiple linear regression: $y_i = w_0 + \sum_{j=}^{D} w_j x_{ij} + \varepsilon_i$, with D number of features
Residual Sum of Squares: $RSS(w_0, w_1) = \sum_{i=1}^{N} \varepsilon_i^2$ to minimize
Total Sum of Squares: $TSS = \sum_{i=1}^{N} (y_i - \overline{y})^2$
Coefficient of determination: $R^2 = 1 - \frac{RSS}{TSS} = \frac{\sum_{i=1}^{N}(y_i - \hat{y}_i)^2}{\sum_{i=1}^{N}(y_i - \overline{y})^2}$
Underfitting and overfitting: we can penalize large weights to avoid overfitting
Stabilized regression:

- **Ride regression** (L^2-regularization):
  $RSS(w) = \sum_{i=1}^{N} \varepsilon_i^2 + \alpha \sum_{j=0}^{D} w_j^2$
- **Lasso regression** (L^1-regularization):
  $RSS(w) = \sum_{i=1}^{N} \varepsilon_i^2 + \alpha \sum_{j=0}^{D} |w_j|$
  Lasso regression has sparsity effect, namely performs also **feature selection** setting to zero some weights

4.2 Model evaluation

Datasets:

- **Training set**: $\frac{1}{2}$ or $\frac{1}{2}$ of data
- **Tests set**: $\frac{1}{2}$ or $\frac{2}{3}$ of data

Cross validation: often $k-$fold with $k = 10$ repeated 10 time averaging the results
Model selection: cross validation's output is the evaluation, NOT the model, so we need to build the model running the algorithm on the entire training set
Hyper-parameter optimization (choice of α): we reserve a validation set in training one or we perform both training and selection of α in the whole training set

5 Classification

We have data with label and we want to build a model to assign label to new data (classification) or make a numerical prediction (prediction or regression)
Evaluation criteria:

- accuracy
- speed for both training and usage
- rubstness in handling noise
- scalability
- simplicity and explanability

5.1 Logistic regression

Score: $\text{score}(\vec{x}_i) = \sum_{j=0}^{D} w_j h_j(\vec{x}_i)$
Label: $\hat{y}_i = \text{sign}(\text{score}(\vec{x}_i))$
Probability of assignment:

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

Likelihood: $L(\vec{w}) = \prod_{i=0}^{N} P(y_i|\vec{x}_i, \vec{w})$
Log-likelihood: $l(\vec{w}) = \ln L(\vec{w})$ to maximize
Logistic regression: $\ln\left(\frac{P(\hat{y}_i=+1|\vec{x}_i)}{P(\hat{y}_i=-1|\vec{x}_i)}\right) = \vec{w}h(\vec{x}) = \sum_{j=0}^{D} w_y h_j(\vec{x}) > 0$
Linear VS Logistic regression:

- Logistic curve fits better 0/1 data and results in better decision boundary
- Error is monotonic in distance from boundary for logistic curve
- Logistic curve is less sensitive to outliers

Regularization:

- L^1−regularization: $l(\vec{w}) - \alpha\|\vec{w}\|_{L^1}$ to minimize
- L^2−regularization: $l(\vec{w}) - \alpha\|\vec{w}\|_{L^2}^2$ to minimize

Multi-class classification:

- **One VS the Rest**
- **Changing model to be multi-class**, i.e. One Hot Encoding

5.2 Support Vector Machines

It works by searching for the hyperplane that maximizes the margin or the largest γ s.t. $\forall i\ y_i(wx_i + w_i) \geqslant \gamma$

5.3 Neural Networks

Perceptron: binary classifier which maps its real input in output computing a weighted sum
Activation functions:

- **Linear**:

  $g(a) = a$

  $g'(a) = 1$

- **Sigmoid**:

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

  $g'(a) = g(a)(1 - g(a))$

- **Tanh**:

  $g(a) = \tanh(a)$

  $g'(a) = 1 - g(a)^2$

5.4 k-nearest neighbor

Characteristics:

- It decides the label observing only the k examples nearest to the new data
- need NO training, the training dataset is the model itself
- k has to be chosen properly to avoid much sensitivity to noise points (k too small) or to include dissimilar examples (k too large)
- For every new data it requires to scan the entire database so it is slow to use
- It is the basic form of learning, it is simple and well understandable

KD-Trees: methods that allow us to use $k-$nearest neighbor algorithm as it is, but scanning in a smart way the database
Ball Trees: better the KD-Trees because using hyper-spheres instead of hyper-rectangles we avoid the problem of data near corners
k-Nearest Neighbor Regression: the target of a new observation is based (mean, regression, ...) on the targets of its k nearest examples

5.5 Naive Bayes

Bayes formula: $P(y|\vec{x}) = \frac{P(\vec{x}|y)P(y)}{P(\vec{x})}$
Assumptions:

- attributes statistically independent
- attributes equally important

Bayes formula using independence: $P(y|\vec{x}) = \frac{P(x_1|y)\cdots P(x_n|y)P(y)}{P(\vec{x})}$
Class: we want to assign the class which is $\arg\max_y P(y|\vec{x})$
Computing probabilities: using independence we can compute the $P(x_i|y)$ easily as frequency counting the case labeled with y in feature x_i
Zero-frequency problem:

- we sum 1 to every computing
- in general when we have k observations on n case we compute the frequency as $\frac{k+\mu p_i}{n+\mu}$ where $\sum_{i=1}^{m} p_i = 1$ with m number of labels (often $p_i = \frac{1}{m}$)

Missing values:

- **Training set**: record NOT included in computing of frequency
- **Test set**: frequency of missing attribute omitted in calculation of posterior probability

Numeric attributes: we assume gaussian distribution, calculate μ and σ and use the gaussian density function to compute the $P(x_i|y)$
Missing values (numerical attributes):

- **Training set**: record NOT included in computing of μ and σ
- **Test set**: frequency of missing attribute omitted in calculation of posterior probability

Advantages: works well even if independence assumption is clearly violated, since it NOT requires accurate probability estimates as long as maximum probability is assigned to correct class
Disadvantages: adding too many redundant attributes will cause problems and also gaussian assumption may be poor

5.5.1 Bayesian Belief Networks

Types of nodes:

- X dose NOT have parents: prior probability $P(X)$
- X has only 1 parent Y: conditional probability $P(X|Y)$
- X has multiple parents $Y_1, \ldots, Y_k$: conditional probability $P(X|Y_i, \ldots, Y_k)$

Types of variables:

- observed
- unobserved
- to be predicted

Advantages:

- econde dependencies among variables
- well suited to dealing with incomplete data
- robust to overfitting
- can be used to include domain knowledge
- well understandable thank to graphical network representation

Disadvantages:

- involves searching a huge space of possible conditional independence relationship
- adding variable to an existing network is difficult

5.5.2 Decision trees

Entropy: $entropy\left(p_i, \ldots, p_n\right) = -\sum_{i=1}^{n} p_i \log_n p_i$
Information: $info\left([n,m]\right) = entropy\left(\frac{n}{n+m}, \frac{m}{n+m}\right)$
Gain: $gain\left(A\right) = info\left(D\right) - info_A\left(D\right)$
$info_A\left(D\right) = \sum_{i=1}^{n} \frac{|D_i|}{|D|} info\left(D_i\right)$
We choose the attributes with **highest** gain
Stopping criteria:

- all samples for a given node belong to the same class (maximum purity)
- NO remaining attributes for further partitioning
- too few samples in a division
- nothing to gain in splitting

Overfitting: tree with 100% purity in all nodes, but useless to classify new item
Intrinsic information: $IntrinsicInfo\left(S,A\right) = -\sum_i \frac{|S_i|}{|S|} \log_2 \frac{|S_i|}{|S|}$
Gain ratio: $GainRatio\left(S,A\right) = \frac{gain(S,A)}{IntrinsicInfo(S,A)}$
We choose the attributes with **highest** gain ratio
Numerical attributes:

- pre-discretization
- multi-way splits

Gini index: $gini\left(T\right) = 1 - \sum_{j=1}^{n} p_j^2$ with often $n = 2$
Gini: $gini_A\left(D\right) = \sum_{i=1}^{n} \frac{|D_i|}{|D|} gini\left(D_i\right)$
Gini gain: $\Delta gini\left(A\right) = gini\left(D\right) - gini_A\left(D\right)$
We divide attributes with many values into One VS the Rest
Overfitting:

- **pre-pruning**: we stop to build the tree where we do NOT gain enough, based on a chi-squared test
- **post-pruning**: we replace subtree if it reduces the estimating error

Error estimate: $e = \left(f + \frac{z^2}{2N} + z\sqrt{\frac{f}{N} - \frac{f^2}{N} + \frac{z^2}{4N}}\right) / \left(1 + \frac{z^2}{N}\right)$ with $c = 0.25, z = 0.69$
Regression and model trees: we compute the standard Deviation Reduction $SDR = \sigma\left(D\right) - \sum_i \frac{|D_i|}{|D|} \sigma\left(D_i\right)$
Decision stumps: one level decision trees

6 Ensembles methods

Steps:

- Create multiple data sets from training data
- Build multiple classifiers
- Combine the obtained classifiers

Advantages:

- can use different classifiers to capture different aspects of the phenomenon
- often improves predictive performance

Disadvantages:

- usually produces output that is very hard to analyze
- difficult and difficult to explain

Bagging (Bootstrap Aggregation): reduces the variance (needs to be high) without increasing the bias

1. take a training set D
2. (Bootstrap) make at each iteration i a training set $D_i \subset D$ sampled with replacement
3. learn a classifier M_i on each D_i
4. each classifier M_i returns its prediction
5. the bagged classifier M^* counts the votes and assigns the class with the most votes

Stable VS Unstable classifiers: if the learning algorithm is stable, the procedure does NOT work well since we are learning from similar D_i and so we learn every time a similar model
Error rate for N classifiers: $err = \sum_{i=\frac{N}{2}+1}^{N} C_{N,i}\varepsilon^i (1-\varepsilon)^{N-i}$
Random Forests: ensembles of unpruned decision tree learners with randomized selection of features at each split

1. take a training set D
2. (Bootstrap) make at each iteration i a training set $D_i \subset D$ sampled with replacement
3. learn a tree T_i on each D_i using at each node a percentage f of the n variables, without pruning
4. each classifier T_i returns its prediction
5. the bagged classifier T^* counts the votes and assigns the class with the most votes

OOB evaluation (Out Of Bag): we evaluate each tree T_i on the records that are NOT in its training dataset D_i
Advantages:

- easy to use
- very high accuracy
- NO overfitting if selecting large number o trees
- insensitive to choice of split percentage

Boosting: reduces the bias without increasing the (too much) the variance

1. weights are assigned
2. a series of classifiers is iteratively learned
3. after a classifier M_i is learned. the weights are updated to allow the subsequent classifier M_{i+1} to pay more attention to the training tuples that were misclassified by M_i
4. the final classifier M^* combines the votes where the weight of each classifier's vote is a function of its accuracy

AdaBoost:

1. we assign to each data uniform weights $w_i = \frac{1}{N}$ with N number of data
2. at each iteration t we generate a weak classifier h_t
3. we compute the error $\varepsilon_t = \frac{1}{n}\sum_{j=1}^{N} w_j \chi_{\{h_t(x_j)\neq y_j\}}$
4. we compute $\alpha_t = \frac{1}{2}\ln\left(\frac{1-\varepsilon_t}{\varepsilon_t}\right)$
5. we update the weights as

- $w_{t+1} = w_t e^{-\alpha_t}$ for correctly classified data
- $w_{t+1} = w_t e^{\alpha_t}$ for incorrectly classified data

6. we normalize the weights
7. the final model is $H(x) = \text{sign}\left(\sum_{t=1}^{T} \alpha_t h_t(x)\right)$

Gradient Boosting: build a sequence of predictors learning a regression predictor, computing the error residual and learning to predict the residual, iteratively
Stacking: trains a learning algorithm to combine the predictions of an heterogeneous set of learning algorithms

7 Evaluation and credibility

Properties:

- **completeness**: algorithm that produces all the interesting patterns contained in a dataset
- **optimization**: algorithm that produces only the interesting patterns contained in a dataset

Model evaluation:

- metrics for performance evaluation
- methods for performance evaluation
- methods for model comparison
- model selection

Confusion matrix:

- **Accuracy**: $acc = \frac{TP+TN}{TP+TN+FP+FN}$
- **Precision**: $pre = \frac{TP}{TP+FP}$
- **Recall**: $rec = \frac{TP}{TP+FN}$
- **F1 measure**: $F1 = \frac{2\ TP}{2\ TP+FP+FN}$
- **Sensitivity**: $ratio_{TP} = \frac{TP}{TP+FN}$
- **Specificity**: $ratio_{TN} = \frac{TN}{TN+FP}$

Methods of estimation:

- **Holdout** (also repeated)
- **Random subsampling**
- **Cross validation**
- **Stratified sampling**
- **Boostrap**: training sets has 63.2% of data so we compute $\varepsilon = 0.632\ \varepsilon_{test} + 0.368\ \varepsilon_{training}$

Model comparison:

- $\theta_1^A, \cdots, \theta_k^A, \theta_1^B, \cdots, \theta_k^B$
- $\delta_i = \theta_i^A - \theta_i^B$
- $\mu_\delta = \frac{1}{k}\sum_i \delta_i$
- $\sigma_\delta = \sqrt{\frac{1}{k}\sum_i (\delta_i - \mu_\delta)^2}$
- H_0: $\mu_\delta = 0$ VS H_1: $\mu_\delta \neq 0$ with t-test

Bonferroni correction: $p-$value is corrected with $\frac{p}{n}$ with n number of test performed
ROC curves
No free-lunch theorem

8 Time series

Constitutive sections:

- **trend**: asymptotic behavior of the data
- **pattern**: seasonal behavior of the data with pattern length time-invariant
- **cyclic**: cyclic behavior of the data with pattern length NOT constant

Build-up:

- **additive**: time series is the sum of trend, pattern and cyclic parts
- **multiplicative**: time series is scaled by trend, pattern and cyclic parts

Percentage of change (pct_change): $\frac{x_t - x_{t-1}}{x_t}$
Autocorrelation

- **normal** (ACF): $r_k = Corr(x_t, x_{t+k})$ for lag k
- **partial** (PACF): regression of the series against its past lags

Random walk normal test:

- **without drift**: $P_t = P_{t-1} + \varepsilon_t$
- **with drift**: $P_t = \mu + P_{t-1} + \varepsilon_t$

Statistical test for random walk: $P_t = \alpha + \beta P_{t-1} + \varepsilon_t$

- $\beta = 1$: random walk
- $\beta < 1$: NOT random walk

Dickey-Fuller test for random walk: $P_t - P_{t-1} = \alpha + \beta P_{t-1} + \varepsilon_t$

- $\beta = 0$: random walk
- $\beta < 1$: NOT random walk

Stationarity of time series:

- **stationary**: statistical properties such as mean, variance and autocorrelation, do NOT change over time
- **NON-stationary**: statistical properties/parameters change over time, like seasonal series which we can make stationary taking its difference

AR (AutoRegressive model):
$AR(1) : R_t = \mu + \Phi R_{t-1} + \varepsilon_t$

- $\Phi = 1$: random walk
- $\Phi = 0$: white noise
- $\Phi \in (-1, 1)$: stationary process

MA (MovingAverage model):
$MA(1) : R_t = \mu + \varepsilon_t + \theta \varepsilon_{t-1}$
ARMA (AutoRegressive - MovingAverage model):
$ARMA(1,1) : R_t = \mu + \Phi R_{t-1} + \varepsilon_t + \theta \varepsilon_{t-1}$

- $ARIMA(p, 0) = ARMA(p, 0)$

- $ARIMA(0,q) = ARMA(0,q)$

ARMA (AutoRegressive - MovingAverage with eXogenous inputs model):
$ARMAX(1,1) : R_t = \mu + \Phi R_{t-1} + \varepsilon_t + \theta\varepsilon_{t-1} + \eta d_{t-1}$
where d_{t-1} is an exogenous variable which contains extra information
ARIMA (AutoRegressive Integrated - MovingAverage with eXogenous inputs model) only for stationary processes:
$ARIMA(1,1,1) : \Delta R_t = \mu + \Phi\Delta R_{t-1} + \varepsilon_t + \theta\varepsilon_{t-1}$ where $\Delta R_t = R_t - R_{t-1}$
$ARIMA(p,0,q) = ARMA(p,q)$
Model identification:

- ACF tails off and PACF cuts off after lag $p \Rightarrow AR\,(p)$
- ACF cuts off after lag q and PACF tails off $\Rightarrow MA\,(q)$
- ACF and PACF tails off $\Rightarrow$ ARMA model, NO information about order

Information criteria:

- **Aikike Information Criterion**: for more explanatory models
- **Bayesian Information criterion**: $BIC = T\ln\left(\frac{SSE}{T}\right) + (k+2)\ln T$ for simpler models

Time series cross-validation:
Error: $e_y = e_y - \hat{y}_t$

- $MAE = \frac{1}{N}\sum_t |e_t|$
- $RMSE = \sqrt{\frac{1}{N}\sum_t e_t^2}$
- $MAPE = \frac{1}{N}\sum_t |p_t|$ where $p_t = \frac{e_t}{y_t}$