Supervised learning. Decision tree learning (DTs)

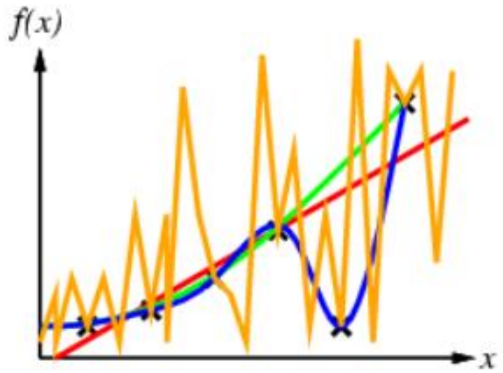
SUMMARY

1. Issues in ML

- what algorithms can approximate functions well (and when)?
- how does the number of training examples influence performance?
- how does complexity of hypothesis representation impact learning performance?
- how does noisy data influence performance?
- what are the theoretical limits of learnability?
- …

2. Supervised learning

- example: Inductive learning
- learn a function from examples
- Occam's razor
  - prefer the simplest hypothesis consistent with data
  - complex models tend not to generalize well



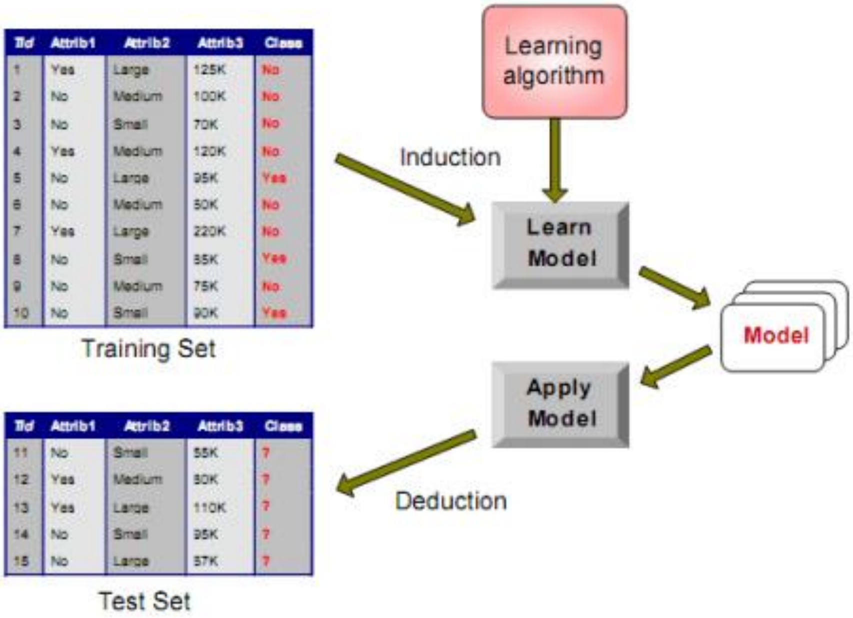
[2]

- **Classification** vs. **regression**
    1. learning a relationship between an input (vector *x*) and an output (*y*) from data
    2. **classification** estimates the discrete output *y* (known as the class)
        - **types**
            - One Class Classification (OCC)
                - Imbalanced data
                - Anomaly/outlier detection, open set recognition (OSR)
                - E.g., OSVM,…
            - Binary classification
            - Multi-class classification
        - predicting tumor cells as benign or malignant
        - categorizing news according to their domain (finance, sports, weather, etc)
        - classifying secondary structures of proteins
        - classifying credit card transactions
        - ….
    3. **regression** estimates the function **f** such that *y*=**f**(*x*) with some confidence measure
        - predicting the exchange rate
        - predicting the price of houses
        - predicting the age of a person
        - …..
- **Offline vs. online** learning
    1. **Offline (batch) learning**
        - generates the best predictor by learning on the entire data set at once
    2. **Online learning**
        - data becomes available in a sequential order and is used to update the best predictor for future data at each step
        - learning incrementally (one instance at time)
        - useful where is computationally unfeasible to train over the entire data set (e.g. stock price prediction)
        - time dependent
- **Eager** vs. **lazy** learning
    1. **Eager learning**
        - an example of offline learning
        - the system tries to construct a general, input independent target function during the training of the system
            - a global model (hypothesis) is built during the training step
            - slow training, fast evaluation
        - builds a global estimate of the target function
        - deal much better with noise
        - generally unable to provide good local approximations of the target function
        - e.g. ANNs, SVMs, DTs, NBC, etc
    2. **Lazy learning**
        - little or no offline processing of the training data
        - motivation
            - online learning
            - data set is continuously updated with new entries

- generalization beyond the training set is delayed until a query is made to the system
- simply stores the training examples
    - not really a training phase
    - fast “training”, slow evaluation
- e.g. instance-based learning methods (kNN, LWR, CBR)
    - e.g., kNN used in online recommendation systems (movies, music, items to sale, a.s.o)
- provide a local approximation for the target function, for each new query instance

3. Eager inductive learners

- Most of the inductive learning models are eager ones.
- Building an eager inductive learner



Tid	Attrib1	Attrib2	Attrib3	Class
1	Yes	Large	125K	No
2	No	Medium	100K	No
3	No	Small	70K	No
4	Yes	Medium	120K	No
5	No	Large	95K	Yes
6	No	Medium	60K	No
7	Yes	Large	220K	No
8	No	Small	85K	Yes
9	No	Medium	75K	No
10	No	Small	90K	Yes

Training Set

Tid	Attrib1	Attrib2	Attrib3	Class
11	No	Small	55K	?
12	Yes	Medium	80K	?
13	Yes	Large	110K	?
14	No	Small	95K	?
15	No	Large	67K	?

Test Set

[2]

- train-validation-test sets
- e.g., 60-20-20 split

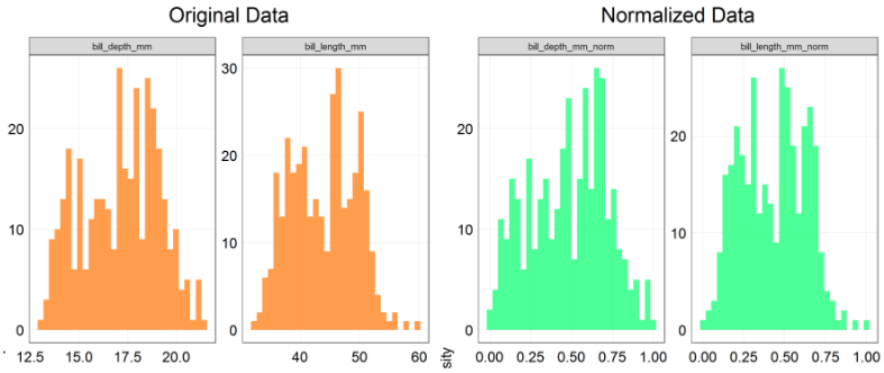
Data pre-processing and analysis

- data cleaning (cleansing)
- **normalization/standardization** (for numerical data), if needed
    - if X is a feature characterizing the data, $X = (X_1, X_2, \dots X_n)$
    - normalization and standardization will not change the distribution of the data
    - **(min-max) normalization**

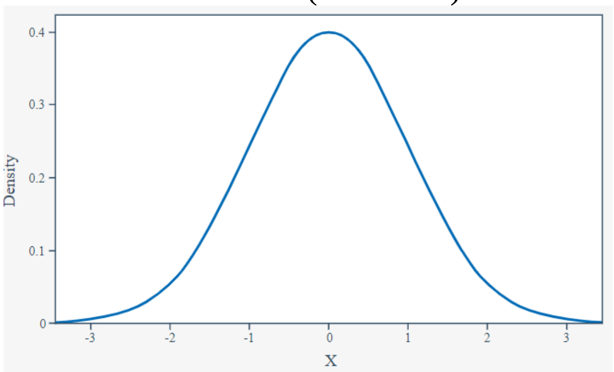
    - changes the range of the data (i.e., the relative distances between individual data points)
        - does not alter the data distribution
    - scale the data between 0 and 1

$$X_i^{'} = \frac{X_i - min}{max - min}$$

        - where *min* and *max* represent the *minimum* and *maximum* values of *X*
        - useful in ML algorithms that do not assume any distribution of data (kNN, neural networks).



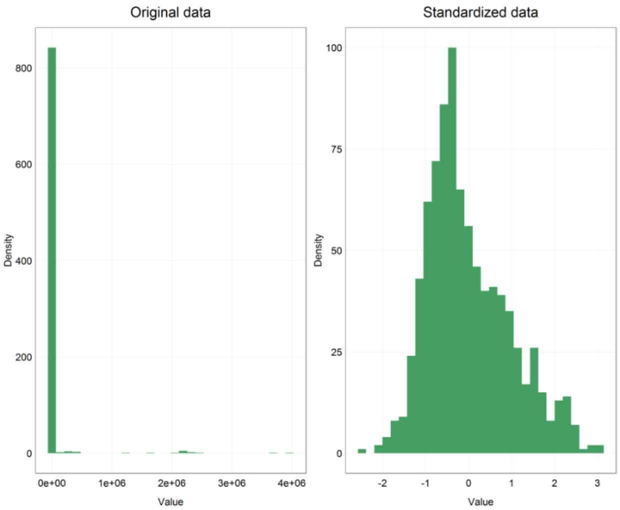
- **standard deviation-based normalization (*standardization*)**
    - standard normal distribution *N*(0,1)
        - 0 mean and 1 stdev (1 variance)



    - after standardization the data will have zero mean and unit variance

$$X_i^{'} = \frac{X_i - \mu}{\sigma}$$

        - μ represents the mean and σ represent the standard deviation of *X*
    - preserves the general data distribution
        - shifts and scales a distribution
        - it does not change the shape of the distribution

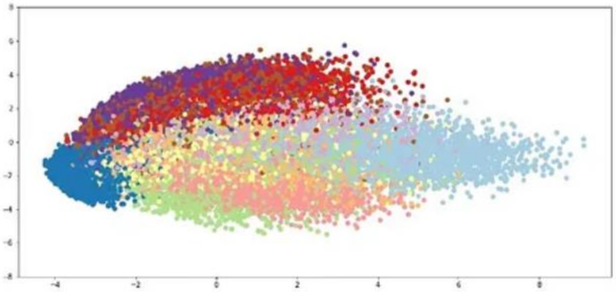


- the transformed data are now centered on 0

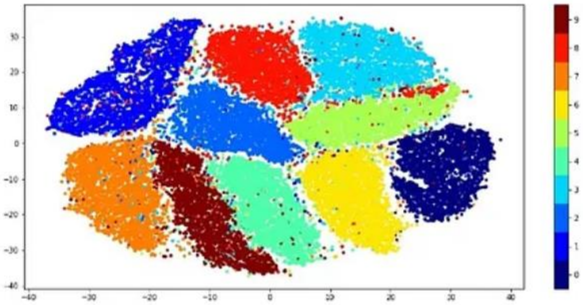
- **feature selection**
  - **select relevant features**
    - correlation
    - independence tests
      - Pearson’s χ^2 test
    - unsupervised learning
      - univariate filter methods
        - information theory, spectral analysis (manifold learning)
    - **univariate feature selection**
      - feature importance – statistical tests
      - select the most important features in a dataset
      - evaluate a feature's relationship with the target variable and select the ones that have the strongest correlation
      - *scikit-learn*
        - F-test for feature scoring
    - algorithms: Relief-based
      - *scikit-learn*
- **component reduction**
  - build new input variables in order to reduce their number
  - no loss of information about data distribution
  - data vizualisation tools
  - **Principal Component Analysis** (PCA)
    - map data into a space of lower dimensionality
    - linear mapping
    - preserving as much of the data's variance
  - **Curvilinear Component Analysis** (CCA)
    - non-linear extension of PCA
    - is a kind of self-organizing map
  - **t-Distributed Stochastic Neighbor Embedding** (t-SNE)
    - non-linear dimensionality reduction

        - **Uniform Manifold Approximation and Projection for Dimension Reduction** (UMAP)

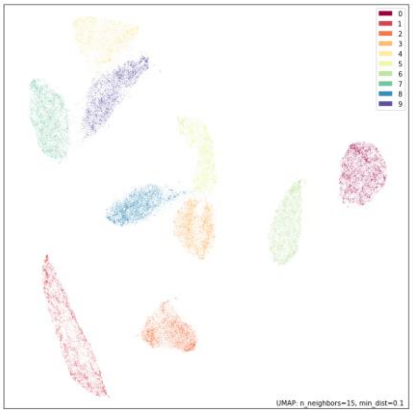
MNIST - PCA



MNIST - TSNE



MNIST-UMAP



    - **others**
        - outliers, isolated instances
        - **imbalanced data**
            - data augmentation
                - oversampling
                    - SMOTE (Synthetic Minority Oversampling TEchnique)
                    - Cluster-based oversampling (CBOS)
                    - Others
                - generative oversampling
                    - GANs, VAE-based
            - undersampling
                - algorithms/rules for deciding the examples to keep from the majority class
                - e.g., Condensed Nearest Neighbors

Build the model (training) – *induction*

- **training set**
    - used to train the model

- **validation set** (extracted from the training data) can be used for optimizing the model
    - used to validate the model
        - on data unseen during training
    - unbiased evaluation of a model fit on the training dataset while tuning the model's hyperparameters.
    - **cross-validation** may be used
    - "best" model

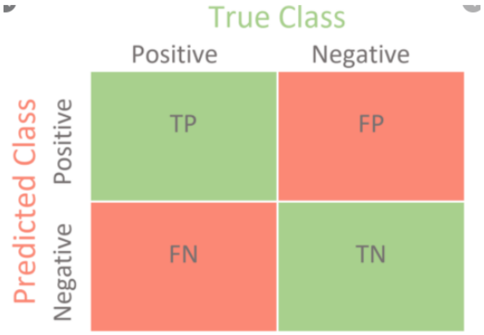
Evaluate/test the performance of the model

- **test set**
    - unbiased evaluation of a final model fit on the training data set
- apply the model – *deduction*
- for evaluating the performance of the model, a cross-validation (**CV**) testing methodology should be used
- **CV**
    - helps in detecting overfitting
    - a way to test robustness
    - a statistical analysis of the results obtained during the CV (e.g., **confidence intervals - CIs**)
        - **Central Limit Theorem** for Cross-validation
            - given a sufficiently large sample size, the sampling distribution of the mean for a variable will approximate a normal distribution
        - perform multiple independent (or close to independent) evaluations of a model performance to result in a population of estimates
            - the mean of these estimates will be an estimate (with certain error) of the true underlying estimate of the model skill on the problem
        - around 30–50 repetitions for computing **CI**s
    - multiple (training-validation-testing) splits are used
        - *k*-fold cross-validation
            - *k* buckets
                - *k*-1 used for training, the remaining for testing
                - repeat *k* times
                    - *k* performance measures $P_1, P_2, \dots, P_k$
                - μ - the mean performance
                    - $\mu = \frac{\sum_{i=1}^{k} P_i}{k}$
                - compute the **CI** of μ
                    - $[\mu - \alpha, \mu + \alpha\,]$ (or $\mu \pm \alpha$)
                    - α is the *confidence value*
                    - e.g., for 95% CI

$$\alpha = 1.96 \cdot \frac{\sigma}{\sqrt{k}}$$

            - 10 is usually used for *k*
        - leave-one out cross-validation
            - *n*-1 instances for training, the remaining for testing
            - repeat *n* times
            - only one performance measure is provided
                - *accuracy*
            - computationally expensive
        - “2/3-1/3” split
            - randomly select 2/3 from the data for training, the remaining for testing
            - repeat *k* times
            - CI of the mean performance

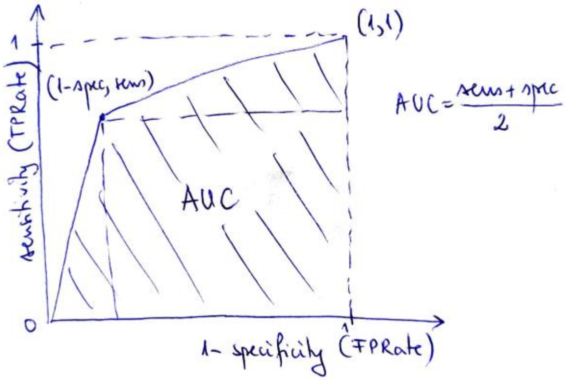
- performance measures used for assessing the performance of a ML model:
    - **for classification**:
        - accuracy, precision, recall (sensitivity), f-measure (f-score), specificity, AUC (Area under the ROC curve), AUPRC (Area under the precision/recall curve)
            - [0,1]
            - binary classification
                - confusion matrix (computed on a testing data set)
                    - TP - # of true positives
                    - TN - # of true negatives
                    - FP - # of false positives
                    - FN - # of false negatives



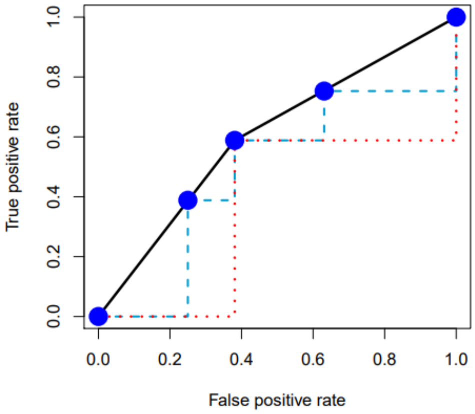
		True Class	
		Positive	Negative
Predicted Class	Positive	TP	FP
	Negative	FN	TN

$acc = \frac{TP+TN}{TP+TN+FP+FN}$	$prec_+ = \frac{TP}{TP+FP}$ (Positive predictive value, PPV)	$sens = recall_+ = \frac{TP}{TP+FN}$ (TPRate, probability of detection - POD)
$f\text{-}score_+ = \frac{2}{\frac{1}{prec_+}+\frac{1}{recall_+}}$	$prec\_ = \frac{TN}{TN+FN}$ (Negative predictive value, NPV)	$spec = recall\_ = \frac{TN}{TN+FP}$ (TNRate)
$f\text{-}score\_ = \frac{2}{\frac{1}{prec\_}+\frac{1}{recall\_}}$	$f\text{-}score = \frac{f\text{-}score_+ + f\text{-}score\_}{2}$	Weigthed f-score

- higher values are better
- accuracy (*acc*)
    - *n* is the dimensionality of the testing data
    - $acc = \frac{1 + 0 + \cdots + 1}{n}$
    - 95% CI of *acc* (the mean of a population parameter)
        - $\alpha = 1.96 \cdot \sqrt{\frac{acc \cdot (1 - acc)}{n}}$
- **AUC**
    - imbalanced data
    - only one point on the ROC (Receiver Operating Characteristic) curve

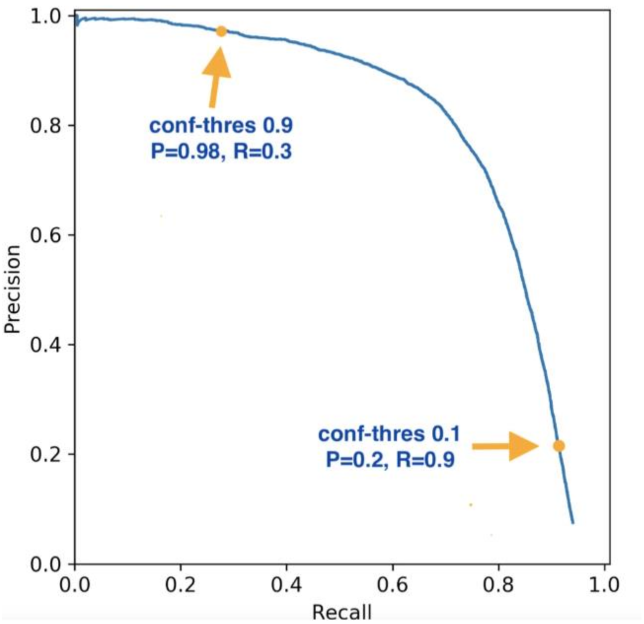


    - multiple points



    - performance of the classifier
        - AUC=0.5 random classifier
        - AUC ∈(0.5,0.6] poor
        - AUC ∈(0.6,0.7] fair
        - AUC ∈(0.7,0.8] good
        - AUC ∈(0.8,0.9] very good
        - AUC ∈(0.9,1] excellent

    - **AUPRC**
        - Area under the Precision-Recall curve



    - **other**
        - e.g., (weather) forecasting
            - critical success index (CSI)
            - false alarm rate (FAR)
            - BIAS
            - HSS (Heidke Skill Score)
            - etc

    - multi-class classification
        - generalization from the binary case
        - $M=(m_{i,j})_{i,j=1,n}$ where n is the number of classes
            - generalization from the binary case
            - $m_{i,j}$= number of instances having the actual class j, predicted as belonging to class i
            - the evaluation measures are computed for each class i (e.g., $prec_{i,}$ etc)
                - an aggregated measure is provided
- **for regression**: MAE (Mean of Absolute Errors), RMSE (Root Mean Squared Error), NRMSE (Normalized Root Mean Squared Error), R^2 (R-squared - coefficient of determination)
    - RMSE is more sensitive to outliers, whereas MAE treats all errors equally
    - MAPE (Mean absolute percentage error)
        - forecasting
    - MPE (Mean percentage error)

        - lower values are better
            - e.g., NRMSE below 5%

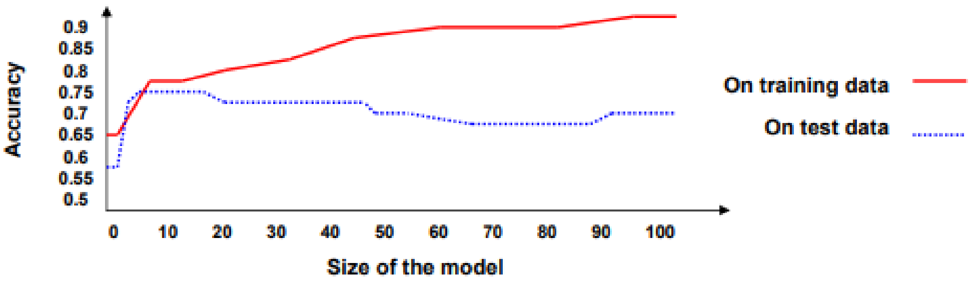
$MAE = \frac{\sum_{i=1}^{n} \lvert y_i - f(x_i) \rvert}{n}$ y_i - predicted value for x_i $f(x_i)$ - true value (observation – in forecasting) n - # test instances	$RMSE = \sqrt{\frac{\sum_{i=1}^{n} (y_i - f(x_i))^2}{n}}$
$MAPE = \frac{100\%}{n} \cdot \sum_{i=1}^{n} \left\lvert \frac{f(x_i) - x_i}{f(x_i)} \right\rvert$	$R^2 = 1 - \frac{SS_{RES}}{SS_{TOT}} = 1 - \frac{\sum_i (y_i - \hat{y}_i)^2}{\sum_i (y_i - \bar{y})^2}$ • y – observations • $\hat{y}$ - predictions

- comparing the performance of two classifiers/regressors
    - **confidence intervals**
    - **statistical tests**
        - is there a statistically significant difference between the performances of 2 models?
            - one/two-tailed Wilcoxon signed rank test for paired data, etc

Generalization

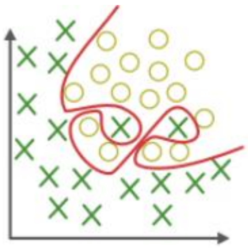
1. overfitting

- overfitting is finding complex functions due to
    - small number of training instances
    - noise or irrelevant data
- a hypothesis overfits the training data if a hypothesis that fits the training examples less well performs better on the entire data distribution
- an overfit model performs **well** on the **training set**, but usually has poor generalization capabilities



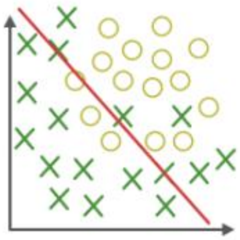
[2]

- causes
  - the model is too complex
    - models the training data two well
    - does not generalize
  - the training data is relatively small and is an insufficient representation of the underlying distribution that it is sampled from
    - the model fails to learn a generalizable mapping.



- good performance on training data but poor performance on testing data
- **techniques** to reduce overfitting:
  - Increase training data.
  - Reduce model complexity.
  - Early stopping during the training phase.
    - stop training once the model performance stops improving on a hold-out validation data set
  - Use regularization techniques (L2, L1 - Ridge Regularization, Lasso Regularization)
    - reducing the generalization error without affecting the training error much
  - Use dropout for neural networks.

2. **underfitting**
   - the model is too simple

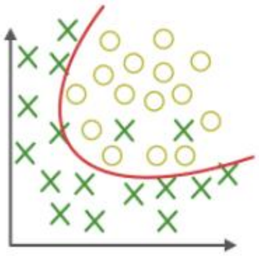


- a model that can neither model the training data, nor generalize to new data
- poor performance both on training data and on testing data

Overfitting and underfitting

- causes for poor performance of ML models

3. **appropriate fitting**

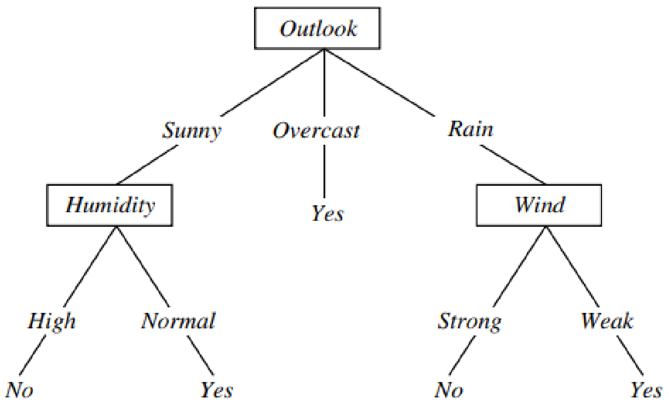


4. Decision tree (DT) learning

- **recent advances**
  - Deep DTs (**Decision Streams**)
    - deep directed acyclic graph of decision rules
  - Deep neural DTs
  - Deep neural decision forests
  - „**Awesome DT research papers**“ - https://github.com/benedekrozemberczki/awesome-decision-tree-papers
    - classification and regression tree research papers (before 2022)
    - prestigious conferences
      - *Machine Learning*: NeurIPS, ICML, ICLR
      - *Computer Vision*: CVPR, ICCV
      - *Natural Language Processing*: ACL, NAACL
      - *Artificial Intelligence*: AAAI, ICANN, IJCAI
      - ….

- used in Data Mining and Machine Learning, uses a decision tree as a predictive model
  - one of the practical methods for inductive inference
- it can be used both for **classification** and **regression**
- is an **eager** inductive learning model
- DT algorithms
  - ID3 – offline algorithm (Iterative Dichotomiser 3, Ross Quinlan in 1986)
  - ID4 is the online variant of ID3
  - C4.5 (C5.0) extensions of ID3
  - J48 is an open source Java implementation of C4.5 in Weka
  - CART (Classification And Regression Trees)
  - aso
- follows the principle of **Occam's razor** in attempting to create the smallest DT possible
- **Limitations**
  - the problem of learning an optimal DT is NP-complete ⇒ the greedy algorithms lead to a local optimum
    - *n* boolean attributes/features, binary classification ⇒ how many distinct trees?
  - overfitting (complex trees)
  - the DT model is **unstable (**small fluctuations in data can make a large difference)
    - bagging, boosting

- **Advantages**
    - simple to understand and interpret
        - people are able to understand DT models
            - interpretable/explainable ML (XML)
                - making ML models and their decisions interpretable
        - uses a white box model
    - require little data preparation (other techniques often require data normalization, missing values to be removed)
    - able to handle both numerical and categorical data
        - **categorical variables**
            - fixed number of possible values
            - information can be sorted in categories
            - types of categorical variables
                - *ordinal* (some intrinsic order between the values, e.g. **rating**: excellent, good, fair, poor)
                - *nominal* – without any intrinsic order (e.g. **sex**: male, female)
                - *binary* (true, false)
        - **continuous variables**
            - can take an infinite number of possible values **(**e.g. real numbers)
            - e.g. temperature, blood pressure
    - perform well with large data sets
- **Decision tree representation**
    - each internal node tests an attribute
    - each branch corresponds to an attribute value
    - each leaf node assigns a classification
    - in DT learning, a hypothesis is a decision tree
    - DTs classify instances by sorting them down the tree from the root to some leaf node, which provides the classification of the instance

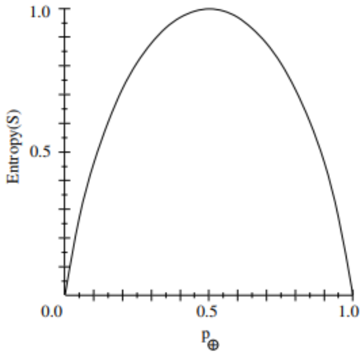


[1]

    - DT – disjunctive expression
        - $(\mathit{Outlook}=\mathit{Sunny} \land \mathit{Humidity}=\mathit{Normal}) \lor (\mathit{Outlook}=\mathit{Overcast}) \lor (\mathit{Outlook}=\mathit{Rain} \land \mathit{Wind}=\mathit{Week})$

- **When to consider DTs**
  - instances describable by attribute-value pairs
  - target function is usually discrete valued (ID3/C4.5)
  - disjunctive hypotheses may be required
  - possibly noisy training data
    - training data may contain errors
    - missing attribute values in the training data

- **ID3 algorithm**
  - performs a greedy search through the space of possible decision trees ⇒ local optimum instead of the global one
  - recursive construction
    - select the "best" decision attribute for the current node
      - the attribute that is the most useful for classifying the examples
    - create descendants of the current node and build the subtrees recursively
  - for selecting the "best" attribute the **information gain** (IG) measure is used
    - measures the expected reduction in **impurity** due to sorting on the attribute
    - measures for **impurity**
      - **entropy**



- S is a sample of training examples
- $p_\oplus$ is the proportion of positive examples in S
- $p_\ominus$ is the proportion of negative examples in S
- Entropy measures the impurity of S

$$Entropy(S) \equiv -p_\oplus \log_2 p_\oplus - p_\ominus \log_2 p_\ominus$$

[1]

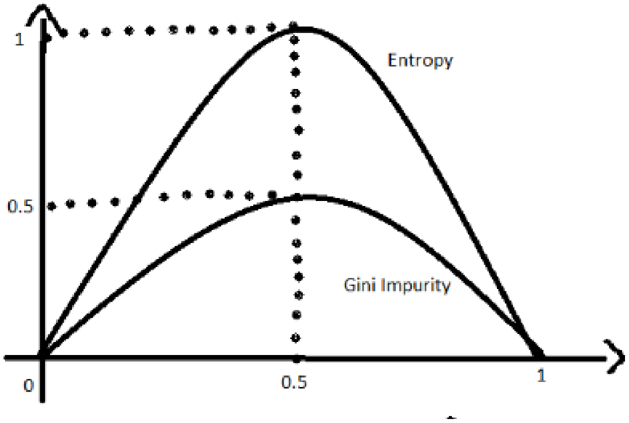
- c classes

$$E(S) = \sum_{i=1}^{c} -p_i log_2 p_i$$

$$Gain(S, A) \equiv Entropy(S) - \sum_{v \in Values(A)} \frac{|S_v|}{|S|} Entropy(S_v)$$

      - **Gini index**

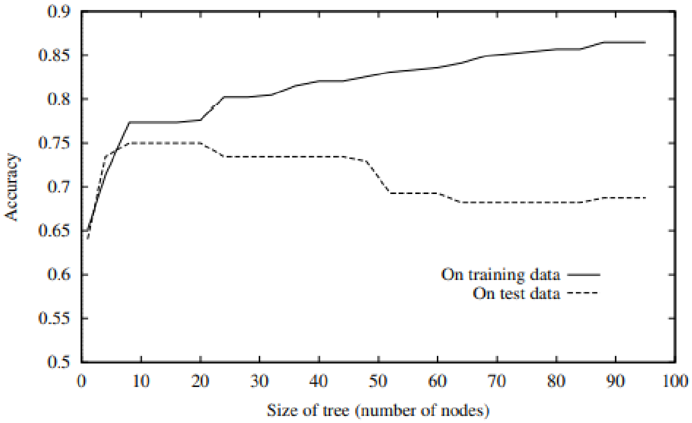
$$Gini = 1 - \sum_{i=1}^{n}(p_i)^2$$



      - about the same performance using entropy and Gini index

- **Inductive bias in ID3**
  - preference for short trees, and for those with **high information** gain near the root

- **Overfitting in DT learning**
  - long hypotheses (trees) may be obtained by coincidence



[1]

    - avoiding overfitting by **pruning**
      - stop growing the tree when data split is not statistically significant
        - use χ^2 test to decide, when building the DT, if pruning should be applied at a certain node

            - if the feature (attribute) at a certain node is uncorrelated with the decision of splitting the node, then prune
    - grow full tree, then post prune [1]
        - ***reduced error pruning***
            - use a validation set
            - evaluate the impact on the validation set of pruning each possible node (plus those below it)
        - ***rule post-pruning***
            - convert tree into rules
            - prune each rule independent of others
            - more successful in practice
            - used by C4.5

- **Issues in DT learning [1]** – handled by C4.5 algorithm
    - incorporating continuous valued attributes
    - alternative measures for selecting attributes
    - handling training examples with missing attribute values
    - handling attributes with costs

5. DT related research topics

- ensemble learning
    - Boosted Decision Trees
    - Random Forests (bagging)
- Fuzzy Decision Trees
- Lazy Decision Trees
- Hybrid models
    - DT + ANN (Artificial Neural Networks)
    - DT + SVM (Support Vector Machines)
    - DT + HMM (Hidden Markov Models)

[SLIDES]

– Inductive learning and decision tree learning (F. Leon) [2]

– DTs (T. Mitchell) [1]

[READING]

– Decision tree learning (T. Mitchell) [1]

– Decision trees (N. Nillson) [2]

Bibliography

[1] Mitchell, T., *Machine Learning*, McGraw Hill, 1997 (available at www.cs.ubbcluj.ro/~gabis/ml/ml-books)

[2] Nillson, N., *Introduction to Machine Learning*, Stanford University, 1996 (available at www.cs.ubbcluj.ro/~gabis/ml/ml-books)