DECISION TREES

EMMANUEL JAKOBOWICZ

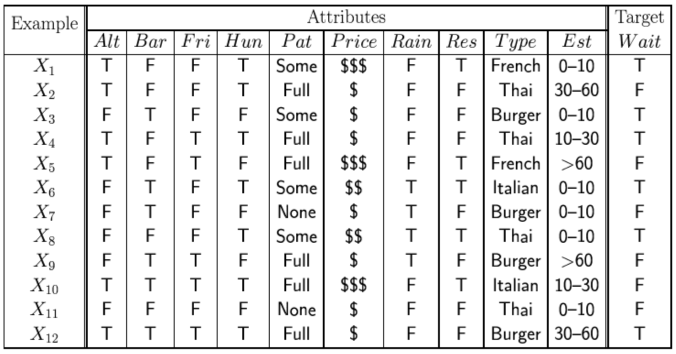
LEARNING DECISION TREES

Example Problem: decide whether to wait for a table at a restaurant, based on the following attributes:

1. **Alternate**: is there an alternative restaurant nearby?
2. **Bar**: is there a comfortable bar area to wait in?
3. **Fri/Sat**: is today Friday or Saturday?
4. **Hungry**: are we hungry?
5. **Patrons**: number of people in the restaurant (None, Some, Full)
6. **Price**: price range ($, $$, $$$)
7. **Raining**: is it raining outside?
8. **Reservation**: have we made a reservation?
9. **Type**: kind of restaurant (French, Italian, Thai, Burger)
10. **WaitEstimate**: estimated waiting time (0-10, 10-30, 30-60, >60)

FEATURE(ATTRIBUTE)-BASED REPRESENTATIONS

- Examples described by feature(attribute) values
  - (Boolean, discrete, continuous)
- E.g., situations where I will/won't wait for a table:



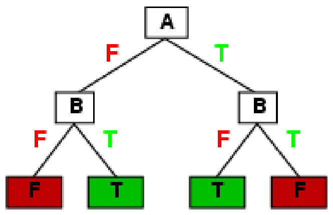
Example	Attributes										Target
	Alt	*Bar*	*Fri*	*Hun*	*Pat*	*Price*	*Rain*	*Res*	*Type*	*Est*	*Wait*
X_1	T	F	F	T	Some	\$\$\$	F	T	French	0–10	T
X_2	T	F	F	T	Full	\$	F	F	Thai	30–60	F
X_3	F	T	F	F	Some	\$	F	F	Burger	0–10	T
X_4	T	F	T	T	Full	\$	F	F	Thai	10–30	T
X_5	T	F	T	F	Full	\$\$\$	F	T	French	>60	F
X_6	F	T	F	T	Some	\$\$	T	T	Italian	0–10	T
X_7	F	T	F	F	None	\$	T	F	Burger	0–10	F
X_8	F	F	F	T	Some	\$\$	T	T	Thai	0–10	T
X_9	F	T	T	F	Full	\$	T	F	Burger	>60	F
X_{10}	T	T	T	T	Full	\$\$\$	F	T	Italian	10–30	F
X_{11}	F	F	F	F	None	\$	F	F	Thai	0–10	F
X_{12}	T	T	T	T	Full	\$	F	F	Burger	30–60	T

- Classification of examples is positive (T) or negative (F)

EXPRESSIVENESS

- Decision trees can express any function of the input attributes.
- E.g., for Boolean functions, truth table row → path to leaf:

A	B	A xor B
F	F	F
F	T	T
T	F	T
T	T	F



- Trivially, there is a consistent decision tree for any training set with one path to leaf for each example (unless *f* nondeterministic in *x*) but it probably won't generalize to new examples
- Prefer to find more compact decision trees

DECISION TREE CONSTRUCTION ALGORITHM

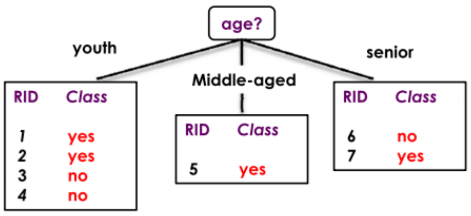
- **Principle**
  - Basic algorithm (adopted by ID3, C4.5 and CART): a **greedy algorithm**
  - Tree is constructed in a *top-down recursive divide-and-conquer* manner
- **Iterations**
  - At start, all the training tuples are at the root
  - Tuples are partitioned recursively based on selected attributes
  - Test attributes are selected on the basis of a heuristic or statistical measure (e.g, information gain)
- **Stopping conditions**
  - All samples for a given node belong to the same class
  - There are no remaining attributes for further partitioning majority **voting** is employed for classifying the leaf
  - There are no samples left

5

This follows an example of Quinlan's ID3

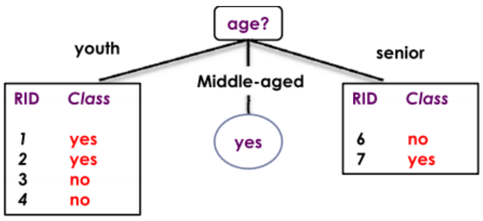
age	income	student	credit_rating	buys_computer
<=30	high	no	fair	no
<=30	high	no	excellent	no
31…40	high	no	fair	yes
>40	medium	no	fair	yes
>40	low	yes	fair	yes
>40	low	yes	excellent	no
31…40	low	yes	excellent	yes
<=30	medium	no	fair	no
<=30	low	yes	fair	yes
>40	medium	yes	fair	yes
<=30	medium	yes	excellent	yes
31…40	medium	no	excellent	yes
31…40	high	yes	fair	yes
>40	medium	no	excellent	no

6



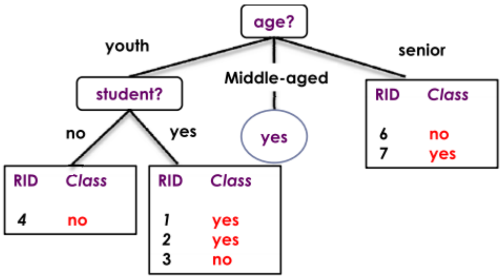
RID	age	student	credit-rating	Class: buys_computer
1	youth	yes	fair	yes
2	youth	yes	fair	yes
3	youth	yes	fair	no
4	youth	no	fair	no
5	middle-aged	no	excellent	yes
6	senior	yes	fair	no
7	senior	yes	excellent	yes

7

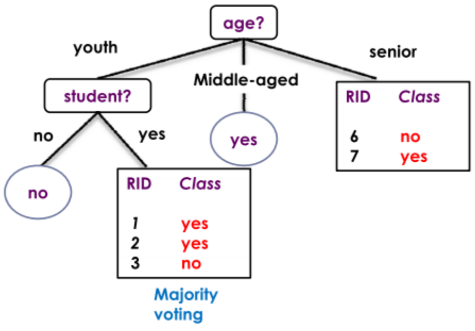


RID	age	student	credit-rating	Class: buys_computer
1	youth	yes	fair	yes
2	youth	yes	fair	yes
3	youth	yes	fair	no
4	youth	no	fair	no
5	middle-aged	no	excellent	yes
6	senior	yes	fair	no
7	senior	yes	excellent	yes

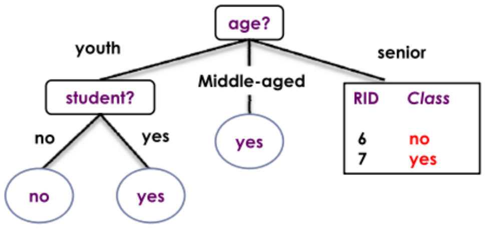
8



RID	age	student	credit-rating	Class: buys_computer
1	youth	yes	fair	yes
2	youth	yes	fair	yes
3	youth	yes	fair	no
4	youth	no	fair	no
5	middle-aged	no	excellent	yes
6	senior	yes	fair	no
7	senior	yes	excellent	yes



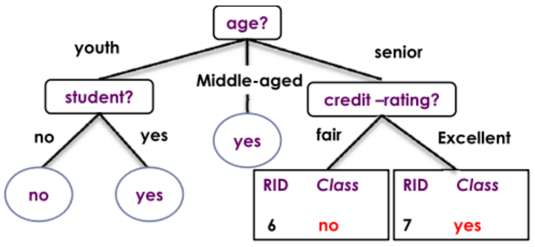
RID	age	student	credit-rating	Class: buys_computer
1	youth	yes	fair	yes
2	youth	yes	fair	yes
3	youth	yes	fair	no
4	youth	no	fair	no
5	middle-aged	no	excellent	yes
6	senior	yes	fair	no
7	senior	yes	excellent	yes



RID	Class
6	no
7	yes

RID	age	student	credit-rating	Class: buys_computer
1	youth	yes	fair	yes
2	youth	yes	fair	yes
3	youth	yes	fair	no
4	youth	no	fair	no
5	middle-aged	no	excellent	yes
6	senior	yes	fair	no
7	senior	yes	excellent	yes

11

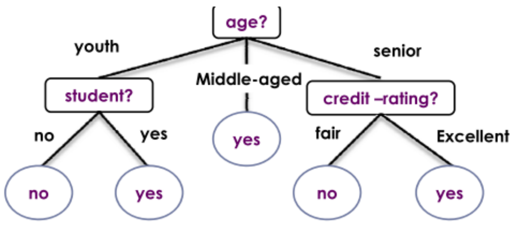


RID	Class
6	no

RID	Class
7	yes

RID	age	student	credit-rating	Class: buys_computer
1	youth	yes	fair	yes
2	youth	yes	fair	yes
3	youth	yes	fair	no
4	youth	no	fair	no
5	middle-aged	no	excellent	yes
6	senior	yes	fair	no
7	senior	yes	excellent	yes

12



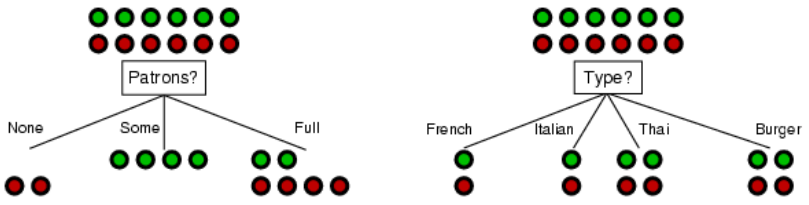
RID	age	student	credit-rating	Class: buys_computer
1	youth	yes	fair	yes
2	youth	yes	fair	yes
3	youth	yes	fair	no
4	youth	no	fair	no
5	middle-aged	no	excellent	yes
6	senior	yes	fair	yes
7	senior	yes	excellent	no

TREE INDUCTION

- Greedy strategy.
  - Split the records based on an attribute test that optimizes certain criterion.
- Issues
  - Determine how to split the records
    - How to specify the attribute test condition?
    - How to determine the best split?
  - Determine when to stop splitting

CHOOSING AN ATTRIBUTE

- Idea: a good attribute splits the examples into subsets that are (ideally) "all positive" or "all negative"



- *Patrons?* is a better choice

15

HOW TO DETERMINE THE BEST SPLIT

- Greedy approach:
  - Nodes with homogeneous class distribution are preferred
- Need a measure of node impurity:

C0: 5	C0: 9
C1: 5	C1: 1
Non-homogeneous, High degree of impurity	Homogeneous, Low degree of impurity

16

MEASURES OF NODE IMPURITY

- Information Gain
- Gini Index
- Misclassification error

Choose attributes to split to achieve minimum impurity

17

INFORMATION GAIN

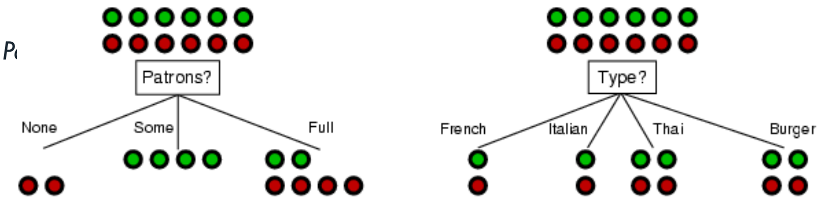
For the training set, $p = n = 6$, $I(6/12, 6/12) = 1$ bit

Consider the attributes *Patrons* and *Type* (and others too):

$$IG(Patrons) = 1 - [\frac{2}{12} I(0,1) + \frac{4}{12} I(1,0) + \frac{6}{12} I(\frac{2}{6}, \frac{4}{6})] = .0541 \text{ bits}$$

$$IG(Type) = 1 - [\frac{2}{12} I(\frac{1}{2}, \frac{1}{2}) + \frac{2}{12} I(\frac{1}{2}, \frac{1}{2}) + \frac{4}{12} I(\frac{2}{4}, \frac{2}{4}) + \frac{4}{12} I(\frac{2}{4}, \frac{2}{4})] = 0 \text{ bits}$$

P



18

MEASURE OF IMPURITY: GINI (CART, IBM INTELLIGENTMINER)

- Gini Index for a given node t :

$$GINI(t) = 1 - \sum_j [p(j \mid t)]^2$$

(NOTE: $p(j \mid t)$ is the relative frequency of class j at node t).

- Maximum ($1 - 1/n_c$) when records are equally distributed among all classes, implying least interesting information
- **Minimum (0.0)** when all records belong to one class, **implying most interesting information**

C1	**0**
C2	**6**
Gini=0.000	

C1	**1**
C2	**5**
Gini=0.278	

C1	**2**
C2	**4**
Gini=0.444	

C1	**3**
C2	**3**
Gini=0.500	

SPLITTING BASED ON GINI

- Used in CART, SLIQ, SPRINT.
- When a node p is split into k partitions (children), the quality of split is computed as,

$$GINI_{split} = \sum_{i=1}^{k} \frac{n_i}{n} GINI(i)$$

where, n_i = number of records at child i,

n = number of records at node p.

COMPARISON OF ATTRIBUTE SELECTION METHODS

- The three measures return good results but
  - Information gain:
    - biased towards multivalued attributes
  - Gain ratio:
    - tends to prefer unbalanced splits in which one partition is much smaller than the others
  - Gini index:
    - biased to multivalued attributes
    - has difficulty when # of classes is large
    - tends to favor tests that result in equal-sized partitions and purity in both partitions

EXAMPLE ALGORITHM: C4.5

- Simple depth-first construction
- Uses Information Gain
- Sorts Continuous Attributes at each node
- Needs entire data to fit in memory
- Unsuitable for Large Datasets

DECISION TREE BASED CLASSIFICATION

- Advantages:
  - **Easy** to construct/implement
  - Extremely **fast** at classifying unknown records
  - Models are **easy to interpret for small-sized trees**
  - **Accuracy is comparable** to other classification techniques for many simple data sets
  - Tree models make no assumptions about the distribution of the underlying data : **nonparametric**
  - Have a **built-in feature selection** method that makes them immune to the presence of useless variables

23

DECISION TREE BASED CLASSIFICATION

- Disadvantages
  - Computationally **expensive to train**
  - **Some decision trees can be overly complex** that do not generalise the data well.
  - **Less expressivity**:There may be concepts that are hard to learn with limited decision trees

24

OVERFITTING AND TREE PRUNING

- **Overfitting:** An induced tree may overfit the training data
  - Too many branches, some may reflect anomalies due to noise or outliers
  - Poor accuracy for unseen samples
- Two approaches to avoid overfitting
  - **Prepruning:** Halt tree construction early - do not split a node if this would result in the goodness measure falling below a threshold
    - Difficult to choose an appropriate threshold
  - **Postpruning:** Remove branches from a "fully grown" tree - get a sequence of progressively pruned trees
    - Use a set of data different from the training data to decide which is the "best pruned tree"

25

DECISION TREE WITH PYTHON

- Scikit learn can handle decision trees classification and regression
- We will focus on classification
- Use the tree.DecisionTreeClassifier() function of sklearn on iris data

- Represent the tree in your notebook

NAIVE BAYES

BAYESIAN CLASSIFICATION: WHY?

- A statistical classifier: performs *probabilistic prediction, i.e.,* predicts class membership probabilities
- Foundation: Based on Bayes' Theorem.
- Performance: A simple Bayesian classifier, *naïve Bayesian classifier*, has comparable performance with decision tree and selected neural network classifiers
- Incremental: Each training example can incrementally increase/decrease the probability that a hypothesis is correct — prior knowledge can be combined with observed data

WHAT IS NAIVE BAYES

- a classification technique based on Bayes' Theorem with an assumption of independence among predictors.
- a Naive Bayes classifier assumes that the presence of a particular feature in a class is unrelated to the presence of any other feature.
- a fruit may be considered to be an apple if it is red, round, and about 3 inches in diameter. Even if these features depend on each other or upon the existence of the other features, all of these properties independently contribute to the probability that this fruit is an apple and that is why it is known as 'Naive'.
- Naive Bayes model is easy to build and particularly useful for very large data sets.
- Along with simplicity, Naive Bayes is known to outperform even highly sophisticated classification methods.
- Bayes theorem provides a way of calculating posterior probability P(c|x) from P(c), P(x) and P(x|c). Look at the equation below:

•*P*(*c*|*x*) is the posterior probability of *class* (c, *target*) given *predictor* (x, *attributes*).
•*P*(*c*) is the prior probability of *class*.
•*P*(*x*|*c*) is the likelihood which is the probability of *predictor* given *class*.
•*P*(*x*) is the prior probability of *predictor*.

$$\underbrace{P(c\mid x)}_{\text{Posterior Probability}}=\frac{\overbrace{P(x\mid c)}^{\text{Likelihood}}\,\overbrace{P(c)}^{\text{Class Prior Probability}}}{\underbrace{P(x)}_{\text{Predictor Prior Probability}}}$$

$$P(c\mid X)=P(x_1\mid c)\times P(x_2\mid c)\times\cdots\times P(x_n\mid c)\times P(c)$$

BAYES' RULE

$$\boxed{p(h\mid d)=\frac{P(d\mid h)P(h)}{P(d)}}$$

Understanding Bayes' rule
d = data
h = hypothesis (model)
- rearranging
$p(h\mid d)P(d)=P(d\mid h)P(h)$
$P(d,h)=P(d,h)$
the same joint probability
on both sides

Who is who in Bayes' rule

$P(h)$: prior belief (probability of hypothesis *h* before seeing any data)
$P(d\mid h)$: likelihood (probability of the data if the hypothesis *h* is true)
$P(d)=\sum_h P(d\mid h)P(h)$: data evidence (marginal probability of the data)
$P(h\mid d)$: posterior (probability of hypothesis *h* after having seen the data *d*)

EXAMPLE OF BAYES THEOREM

- Given:
  - A doctor knows that meningitis causes stiff neck 50% of the time
  - Prior probability of any patient having meningitis is 1/50,000
  - Prior probability of any patient having stiff neck is 1/20
- If a patient has stiff neck, what's the probability he/she has meningitis?

$$P(M \mid S) = \frac{P(S \mid M)P(M)}{P(S)} = \frac{0.5 \times 1/50000}{1/20} = 0.0002$$

HOW IT WORKS

- Let's understand it using an example.
- I have a training data set of weather and corresponding target variable 'Play' (suggesting possibilities of playing).
- Now, we need to classify whether players will play or not based on weather condition. Let's follow the below steps to perform it.
  - Step 1: Convert the data set into a frequency table
  - Step 2: Create Likelihood table by finding the probabilities like Overcast probability = 0.29 and probability of playing is 0.64.
  - Step 3: Now, use Naive Bayesian equation to calculate the posterior probability for each class. The class with the highest posterior probability is the outcome of prediction.
- **Problem:** Players will play if weather is sunny. Is this statement is correct?
  - We can solve it using above discussed method of posterior probability.
  - P(Yes | Sunny) = P(Sunny | Yes) * P(Yes) / P (Sunny)
  - Here we have P (Sunny |Yes) = 3/9 = 0.33, P(Sunny) = 5/14 = 0.36, P(Yes)= 9/14 = 0.64
  - Now, P (Yes | Sunny) = 0.33 * 0.64 / 0.36 = 0.60, which has higher probability.
- Naive Bayes uses a similar method to predict the probability of different class based on various attributes. This algorithm is mostly used in text classification and with problems having multiple classes.

ASPECTS

- Pros:
  - It is easy and fast to predict class of test data set. It also perform well in multi class prediction
  - When assumption of independence holds, a Naive Bayes classifier performs better compare to other models like logistic regression and you need less training data.
  - It perform well in case of categorical input variables compared to numerical variable(s). For numerical variable, normal distribution is assumed (bell curve, which is a strong assumption).
- Cons:
  - If categorical variable has a category (in test data set), which was not observed in training data set, then model will assign a 0 (zero) probability and will be unable to make a prediction. This is often known as "Zero Frequency". To solve this, we can use the smoothing technique. One of the simplest smoothing techniques is called Laplace estimation.
  - On the other side naive Bayes is also known as a bad estimator, so the probability outputs from predict_proba are not to be taken too seriously.
  - Another limitation of Naive Bayes is the assumption of independent predictors. In real life, it is almost impossible that we get a set of predictors which are completely independent.

HOW TO ESTIMATE PROBABILITIES FROM DATA?

- For continuous attributes:
  - Discretize the range into bins
    - one ordinal attribute per bin
    - violates independence assumption
  - Two-way split: $(A < v)$ or $(A > v)$
    - choose only one of the two splits as new attribute
  - Probability density estimation:
    - Assume attribute follows a normal distribution
    - Use data to estimate parameters of distribution
      (e.g., mean and standard deviation)
    - Once probability distribution is known, can use it to estimate the conditional probability $P(A_i|c)$

APPLICATIONS

- **Real time Prediction:** Naive Bayes is an eager learning classifier and it is sure fast. Thus, it could be used for making predictions in real time.
- **Multi class Prediction:** This algorithm is also well known for multi class prediction feature. Here we can predict the probability of multiple classes of target variable.
- **Text classification/ Spam Filtering/ Sentiment Analysis:** Naive Bayes classifiers mostly used in text classification (due to better result in multi class problems and independence rule) have higher success rate as compared to other algorithms. As a result, it is widely used in Spam filtering (identify spam e-mail) and Sentiment Analysis (in social media analysis, to identify positive and negative customer sentiments)
- **Recommendation System:** Naive Bayes Classifier and Collaborative Filtering together builds a Recommendation System that uses machine learning and data mining techniques to filter unseen information and predict whether a user would like a given resource or not

NAIVE BAYES WITH PYTHON

- Again, scikit learn (python library) will help here to build a Naive Bayes model in Python. There are three types of Naive Bayes model under scikit learn library:
  - **Gaussian:** It is used in classification and it assumes that features follow a normal distribution.
  - **Multinomial:** It is used for discrete counts. For example, let's say, we have a text classification problem. Here we can consider bernoulli trials which is one step further and instead of "word occurring in the document", we have "count how often word occurs in the document", you can think of it as "number of times outcome number x_i is observed over the n trials".
  - **Bernoulli:** The binomial model is useful if your feature vectors are binary (i.e. zeros and ones). One application would be text classification with 'bag of words' model where the 1s & 0s are "word occurs in the document" and "word does not occur in the document" respectively.
- Based on your data set, you can choose any of above discussed model.
- You can use from sklearn.naive_bayes import GaussianNB to apply a Naïve Bayes classifier

NAÏVE BAYES (SUMMARY)

- *Advantage*
  - Robust to isolated noise points
  - Handle missing values by ignoring the instance during probability estimate calculations
  - Robust to irrelevant attributes
- *Disadvantage*
  - Assumption: class conditional independence, which may cause loss of accuracy
  - Independence assumption may not hold for some attribute. Practically, dependencies exist among variables
    - Use other techniques such as Bayesian Belief Networks (BBN)

IMPROVING YOUR NAIVE BAYES CLASSIFIER

- If continuous features do not have normal distribution, we should use transformation or different methods to convert it in normal distribution.
- Remove correlated features, as the highly correlated features are voted twice in the model and it can lead to over inflating importance.
- Naive Bayes classifiers has limited options for parameter tuning like alpha=1 for smoothing, fit_prior=[True|False] to learn class prior probabilities or not and some other options. I would recommend to focus on your pre-processing of data and the feature selection.

THE K-NEAREST NEIGHBORS

CLASSIFICATION PARADIGMS

- We can categorize three fundamental approaches to classification:
  - Generative models: Model $p(x|C_k)$ and $P(C_k)$ separately and use the Bayes theorem to find the posterior probabilities $P(C_k|x)$
    - E.g. Naive Bayes, Gaussian Mixture Models, Hidden Markov Models,…
  - Discriminative models:
    - Determine $P(C_k|x)$ directly and use in decision
    - E.g. Linear discriminant analysis, SVMs, NNs,…
  - Find a discriminant function f that maps x onto a class label directly without calculating probabilities

LAZY LEARNERS

- The classification algorithms presented before are **eager learners**
  - Construct a model before receiving new tuples to classify
  - Learned models are ready and eager to classify previously unseen tuples
- **Lazy learners**
  - The learner waits till the last minute before doing any model construction
  - In order to classify a given test tuple
    - Store training tuples
    - Wait for test tuples
    - Perform generalization based on similarity between test and the stored training tuples

Eager Learners	**Lazy Learners**
• Do lot of work on training data	• Do less work on training data
• Do less work when test tuples are presented	• Do more work when test tuples are presented

WHEN DO WE USE KNN ALGORITHM?

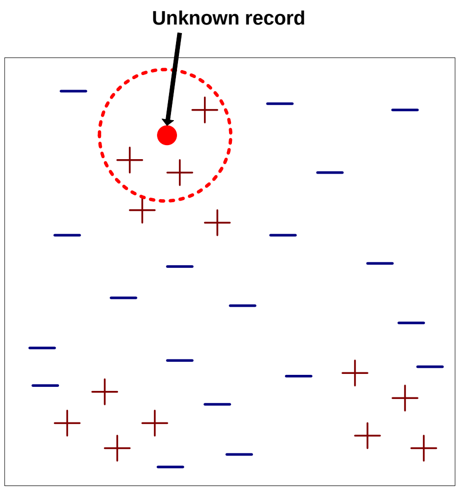
- KNN can be used for both classification and regression predictive problems. However, it is more widely used in classification problems in the industry. To evaluate any technique we generally look at 3 important aspects:
  1. Ease to interpret output
  2. Calculation time
  3. Predictive Power

BASIC K-NEAREST NEIGHBOR CLASSIFICATION

- Given training data $(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_N, y_N)$
- Define a distance metric between points in input space ***D(x$_1$,x$_i$)***
  - E.g., Eucledian distance, Weighted Eucledian, Mahalanobis distance, TFIDF, etc.

- Training method:
  - Save the training examples
- At prediction time:
  - Find the ***k*** training examples $(x_1,y_1),\ldots(x_k,y_k)$ that are **closest** to the test example x given the distance $D(x_1,x_i)$
  - Predict the most frequent class among those y_i's.

NEAREST-NEIGHBOR CLASSIFIERS



- Requires three things
  - The set of stored records
  - Distance Metric to compute distance between records
  - The value of *k*, the number of nearest neighbors to retrieve
- To classify an unknown record:
  - Compute distance to other training records
  - Identify *k* nearest neighbors
  - Use class labels of nearest neighbors to determine the class label of unknown record (e.g., by taking majority vote)

K-NEAREST NEIGHBOR MODEL

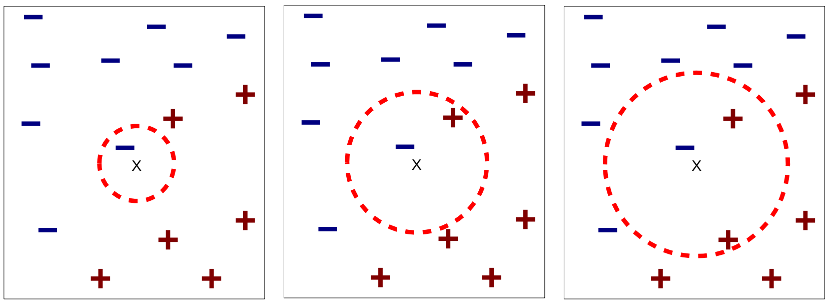
- **Classification:** $\hat{y} = \text{ most common class in set } \{y_1, \ldots, y_K\}$
- **Regression:** $\hat{y} = \frac{1}{K} \sum_{k=1}^{K} y_k$

K-NEAREST NEIGHBOR MODEL:WEIGHTED BY DISTANCE

- **Classification**: $\hat{y} =$ most common class in wieghted set
$$\{D(\mathbf{x},\mathbf{x}_1)y_1,\ldots,D(\mathbf{x},\mathbf{x}_K)y_K\}$$

- **Regression**:
$$\hat{y} = \frac{\sum_{k=1}^{K} D(x,x_k)y_k}{\sum_{k=1}^{K} D(x,x_k)}$$

DEFINITION OF NEAREST NEIGHBOR

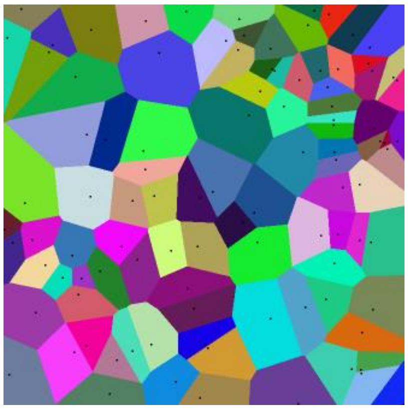


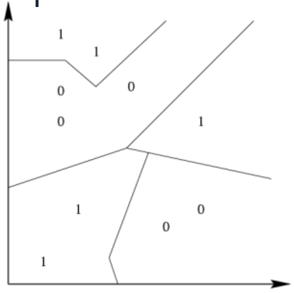
(a) 1-nearest neighbor (b) 2-nearest neighbor (c) 3-nearest neighbor

K-nearest neighbors of a record x are data points that have the k smallest distance to x

VORONOI DIAGRAM

Decision surface formed by the training examples

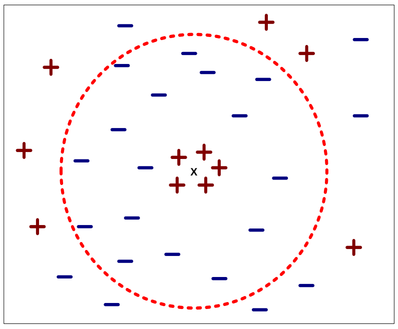




- Each line segment is equidistance between points in opposite classes.
- The more points, the more complex the boundaries.

NEAREST NEIGHBOR CLASSIFICATION…

- Choosing the value of k:
  - If k is too small, sensitive to noise points
  - If k is too large, neighborhood may include points from other classes



DETERMINING THE VALUE OF K

- In typical applications k is in units or tens rather than in hundreds or thousands
- Higher values of k provide smoothing that reduces the risk of overfitting due to noise in the training data
- Value of k can be chosen based on error rate measures
- We should also avoid over-smoothing by choosing k=n, where n is the total number of tuples in the training data set

DETERMINING THE VALUE OF K

- Given training examples $(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_N, y_N)$
- Use N fold cross validation
  - Search over K = (1,2,3,…,*Kmax*). Choose search size *Kmax* based on compute constraints
  - Calculated the average error for each K:
    - Calculate predicted class $\hat{y}_i$ for each training point $(\mathbf{x}_i, y_i), \quad i = 1, \ldots, N$

    (using all other points to build the model)
    - Average over all training examples
- Pick K to minimize the cross validation error

NEAREST NEIGHBOR CLASSIFICATION…

- k-NN classifiers are lazy learners
  - It does not build models explicitly
  - Unlike eager learners such as decision tree induction and rule-based systems
- Adv: No training time
- Disadv:
  - Testing time can be long, classifying unknown records are relatively expensive
  - Curse of Dimensionality : Can be easily fooled in high dimensional spaces
    - Dimensionality reduction techniques are often used

K-NN WITH PYTHON

- K-NN to predict a continuous outcome
- Working on NBA players statistics for season 2013-1014
- Lets try to predict pts.Values for a test set (equal to 20%) of the dataset
- Prepare your dataset and apply k-nearest neighbors using the scikit-learn function
  - `KNeighborsRegressor(n_neighbors=?)`