Applied Machine Learning

Data Transformation

Computer Science, Fall 2022

Instructor: Xuhong Zhang

Data Transformation

- Outline
  - Attribute Selection
  - Discretizing numeric attributes
  - Projections
  - Transforming multiple classes to binary ones
  - Calibrating class probabilities

You just want to apply a learner? No

- Scheme/parameter selection
  - Treat selection process as part of the learning process to avoid optimistic performance estimates
- Modifying the input:
  - Data engineering to make learning possible or easier
- Modifying the output
  - Converting multi-class problems into two-class ones
  - Re-calibrating probability estimates

Attribute selection

- Attribute selection is often important in practice
- For example, adding a random (i.e., irrelevant) attribute can significantly degrade a classification tree's performance
  - Problem: the built-in attribute selection is based on smaller and smaller amounts of data
- Instance-based learning is particularly susceptible to irrelevant attributes
  - Number of training instances required increases exponentially with number of irrelevant attributes
- Exception: naïve Bayes can cope well with irrelevant attributes
  - Note that relevant attributes can also be harmful if they mislead the learning algorithm

Shema-independent attribute selection

- *Filter* approach to attribute selection: assess attributes based on general characteristics of the data
- In this approach, the attributes are selected in a manner that is independent of the target machine learning scheme
- One method: find smallest subset of attributes that separates data
- Another method: use a fast learning scheme that is different from the target learning scheme to find relevant attributes
  - E.g., use attributes selected by tree models, or coefficients of linear model, possibly applied recursively (*recursive feature elimination*)

Q: What you need to pay attention to when you use coefficients to select features?

Entropy

- Entropy: a common way to measure impurity
- Entropy $H(X)$ of a random variable X

# of possible values for X

$$H(X) = -\sum_{i=1}^{n} P(X = i) \log_2 P(X = i)$$

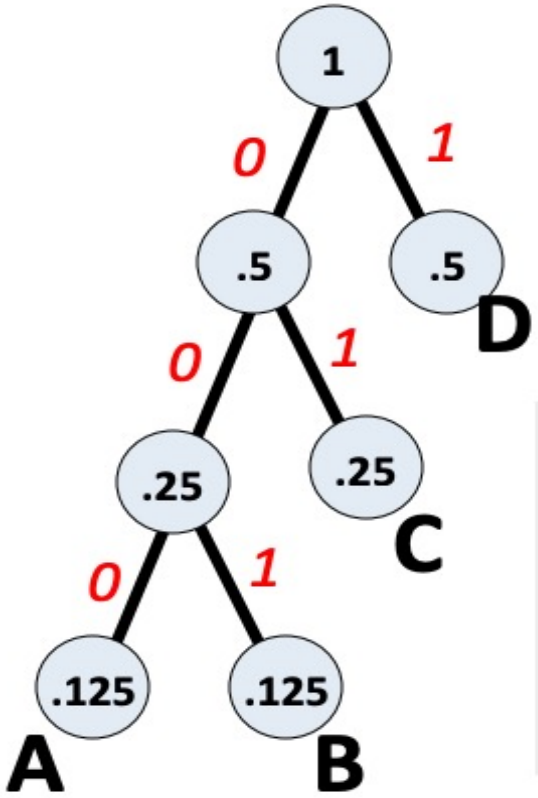
- $H(X)$ is the expected number of bits needed to encode a randomly drawn value of X (under most efficient code) to encode message $X = i$.
- So, expected number of bits to code one random X is $H(X)$.

Example: Huffman Code

- In 1952 MIT student David Huffman devised, in the course of doing a homework assignment, an elegant coding scheme which is optimal in the case where all symbols' probabilities are integral powers of ½.
- A Huffman code can be built in the following manner:
  - Rank all symbols in order of probability of occurrence
  - Successively combine the two symbols of the lowest probability to form a new composite symbol; eventually we will build a binary tree where each node is the probability of all nodes beneath it
  - Trace a path to each leaf, noticing direction at each node

Huffman Code Example

M	P
A	.125
B	.125
C	.25
D	.5



M	code	length	prob	
A	000	3	0.125	0.375
B	001	3	0.125	0.375
C	01	2	0.250	0.500
D	1	1	0.500	0.500
average message length				1.750

If we use this code to many messages (A,B,C or D) with this probability distribution, then, over time, the average bits/message should approach 1.75

Based on slide from M. DesJardians & T. Finin

Scheme-independent attribute selection

- Attribute weighting techniques based on instance-based learning can also be used for filtering

- Correlation-based Feature Selection (CFS):
  - Correlation between attributes measured by symmetric uncertainty:

$$U(A,B) = 2\frac{H(A)+H(B)-H(A,B)}{H(A)+H(B)} \in [0,1],\ H(A,B) = -\sum_i \sum_j P(A,B)\log_2 P(A,B)$$

  - Goodness of subset of attributes measured by

$$\sum_j U(A_j,C) / \sqrt{\sum_i \sum_j U(A_i,A_j)}$$

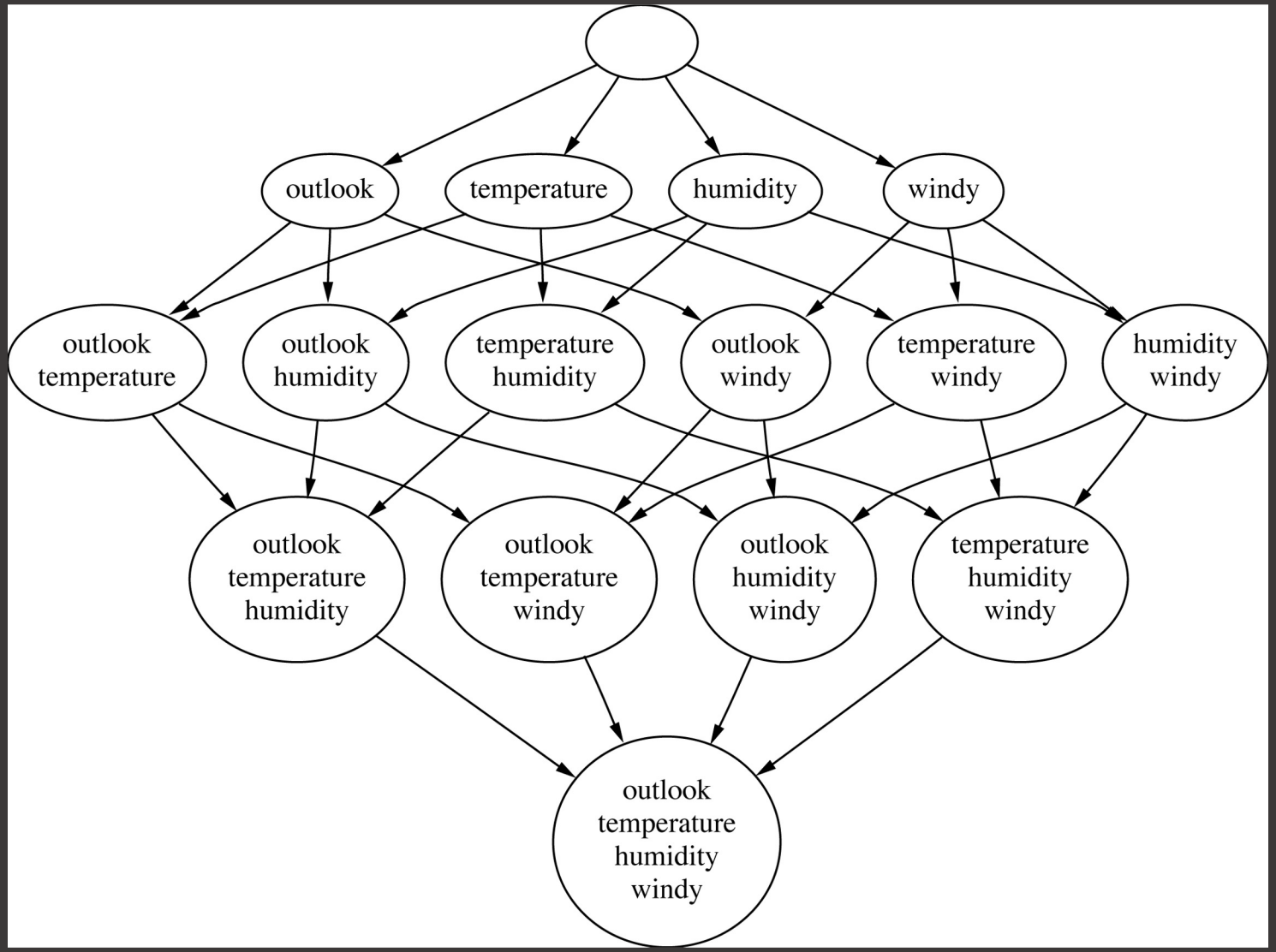
    breaking ties in favour of smaller subsets

C: the class attribute
Note: if all m attributes in the subset correlate perfectly with the class and with one another, the numerator becomes ?
And the denominator becomes ?

Searching the attribute space

- Number of attribute subsets is exponential in the number of attributes
- Common greedy approaches:
  - forward selection
  - backward elimination
- More sophisticated strategies:
  - Bidirectional search
  - Best-first search: can find optimum solution
  - Beam search: approximation to best-first search
  - Genetic algorithms

Attribute subsets for weather data



Scheme-specific selection

- *Wrapper* approach to attribute selection: attributes are selected with target scheme in the loop
- Implement "wrapper" around learning scheme
  - Evaluation criterion: cross-validation performance
- Time consuming in general
- Can use significance test to stop cross-validation for a subset early if it is unlikely to "win"
  - Can be used with forward, backward selection, prior ranking, or special-purpose *schemata search*
- Scheme-specific attribute selection is essential for learning decision tables
- Efficient for decision tables and Naïve Bayes

Attribute discretization

- Discretization can be useful even if a learning algorithm can be run on numeric attributes directly
- Avoids normality assumption in Naïve Bayes and clustering
- Examples of discretization we have already encountered:
  - Decision tree performs local discretization
- Global discretization can be advantageous because it is based on more data
- Apply learner to
  - k - valued discretized attribute or to
  - k – 1 binary attributes that code the cut points
- The latter approach often works better when learning decision trees or rule sets

Discretization: unsupervised

- *Unsupervised discretization*: determine intervals without knowing class labels
  - When clustering, the only possible way!
- Two well-known strategies:
  - *Equal-interval binning*
  - *Equal-frequency binning* (also called *histogram equalization*)
- Unsupervised discretization is normally inferior to supervised schemes when applied in classification tasks
  - But equal-frequency binning works well with naïve Bayes if the number of intervals is set to the square root of the size of dataset (*proportional k-interval discretization)*

Discretization: supervised

- Classic approach to *supervised* discretization is entropy-based
- This method builds a decision tree with pre-pruning on the attribute being discretized
  - Uses entropy as splitting criterion
  - Uses the minimum description length principle as the stopping criterion for pre-pruning
- Works well: still the state of the art

Formula for stopping criterion

- Can be formulated in terms of the information gain
- Assume we have *N* instances
  - Original set: *k* classes, entropy *E*
  - First subset: k_1 classes, entropy E_1
  - Second subset: k_2 classes, entropy E_2

$$gain > \frac{\log_2(N-1)}{N} + \frac{\log_2\left(3^k - 2\right) - kE + k_1E_1 + k_2E_2}{N}$$

- If the information gain is greater than the expression on the right, we continue splitting
- Results in *no* discretization intervals for the temperature attribute in the weather data

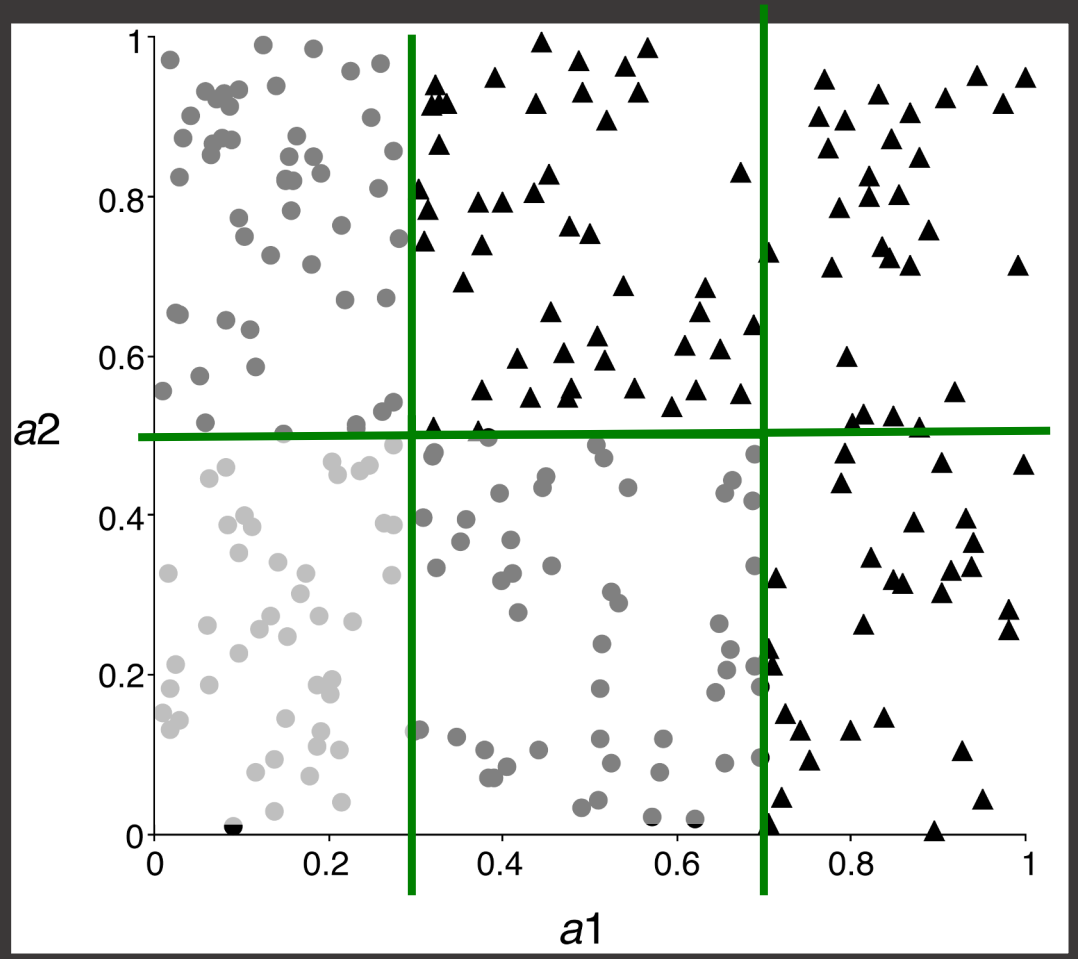
Error-based vs. entropy-based

- Question:
  could the best discretization ever have two adjacent intervals with the same class?
- Error-based: No. For if so,
  - Collapse the two
  - Free up an interval
  - Use it somewhere else
  - (This is what error-based discretization will do)
- Entropy-based: Surprisingly, yes.
  - (and entropy-based discretization can do it)

Error-based vs. entropy-based

A 2-class, 2-attribute problem

- The point is that what changes as the value of a1 crosses the boundary at 0.3 is not the majority class but the class distribution.
- The majority class remains dot.
- The distribution, however, changes markedly, from 100% to just over 50%.
- And the distribution changes again as the boundary at 0.7 is crossed, from 50% to 0%.



Entropy-based discretization can detect change of ***class distribution***

Supervised discretization: other methods

- Can replace top-down procedure by bottom-up method
- This bottom-up method has been applied in conjunction with the *chi-squared* test
  - Continue to merge intervals until they become significantly different
- Can use dynamic programming to find optimum *k*-way split for given additive criterion
  - Requires time quadratic in the number of instances
  - But can be done in linear time if error rate is used instead of entropy
  - However, using error rate is generally not a good idea when discretizing an attribute as we will see

Projections

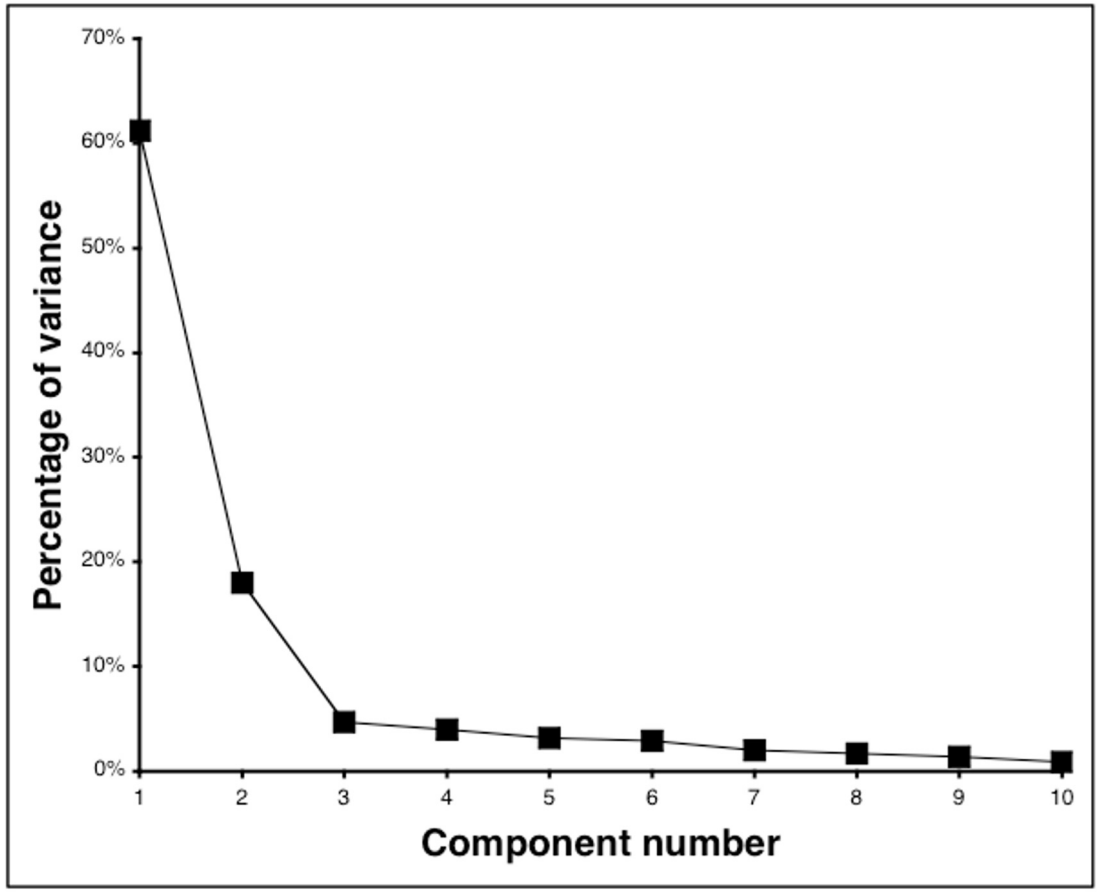
- Simple transformations can often make a large difference in performance
- Example transformations (not necessarily for performance improvement):
  - Difference of two date attributes
  - Ratio of two numeric (ratio-scale) attributes
  - Concatenating the values of nominal attributes
  - Encoding cluster membership
  - Adding noise to data
  - Removing data randomly or selectively
  - Obfuscating the data

Principle component analysis

- Unsupervised method for identifying the important directions in a dataset
- We can then rotate the data into the (reduced) coordinate system that is given by those directions
- PCA is a method for *dimensionality reduction*
- Algorithm:
  1. Find direction (axis) of greatest variance
  2. Find direction of greatest variance that is perpendicular to previous direction and repeat
- Implementation: find eigenvectors of the covariance matrix of the data
  - Eigenvectors (sorted by eigenvalues) are the directions

Example: 10-dimensional data

Axis	Variance	Cumulative
1	61.2%	61.2%
2	18.0%	79.2%
3	4.7%	83.9%
4	4.0%	87.9%
5	3.2%	91.1%
6	2.9%	94.0%
7	2.0%	96.0%
8	1.7%	97.7%
9	1.4%	99.1%
10	0.9%	100.0%



- Data is normally standardized or mean-centered for PCA

Random projections

- PCA is nice but expensive: cubic in number of attributes
- Alternative: use random directions instead of principal components
- Surprising: random projections preserve distance relationships quite well (on average)
  - Can use them to apply *k*D-trees to high-dimensional data
  - Can improve stability by using ensemble of models based on different projections
- Different methods for generating random projection matrices have been proposed

Correlation vs. statistical independence

- PCA is sometimes thought of as a method that seeks to transform correlated variables into linearly uncorrelated ones
- Important: correlation and statistical independence are two different criteria
  - Uncorrelated variables have correlation coefficients equal to zero – entries in a covariance matrix
  - Two variables *A* and *B* are considered independent when their joint probability is equal to the product of their *marginal* probabilities:

$$P(A, B) = P(A)P(B)$$

Automatic data cleansing

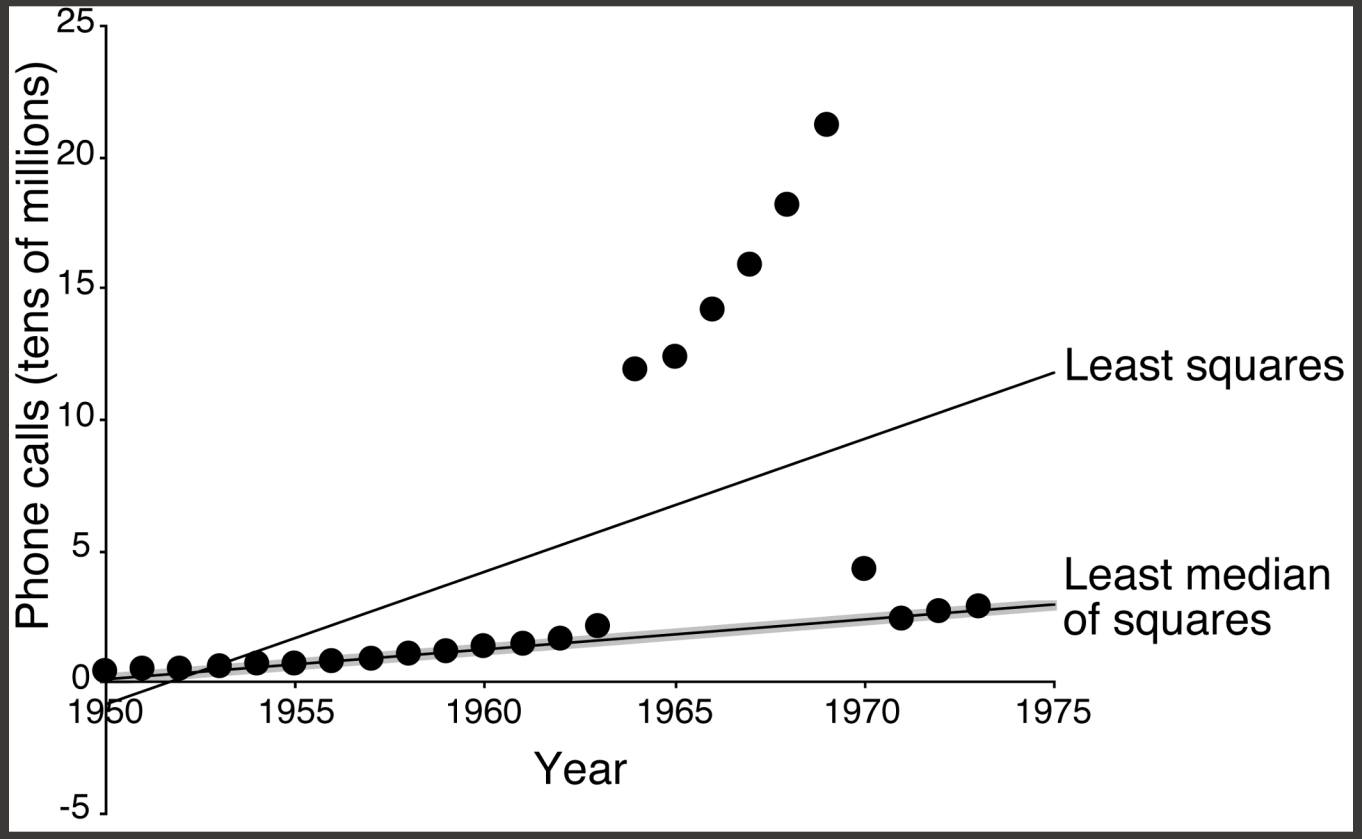
- Attribute noise vs. class noise
  - Attribute noise should be left in the training set (i.e., do not train on clean set and test on dirty one)
  - Systematic class noise (e.g., one class substituted for another): leave in training set
  - Unsystematic class noise: eliminate from training set, if possible

Robust regression

- "Robust" statistical method -- one that addresses problem of outliers
- Possible ways to make regression more robust:
  - Minimize absolute error, not squared error
  - Remove outliers (e.g., 10% of points farthest from the regression plane)
  - Minimize median instead of mean of squares (copes with outliers in x and y direction)
- *Least median of squares regression* finds the narrowest strip covering half the observations
  - Expensive to compute

Example: least median of squares

Number of international phone calls from Belgium, 1950–1973



Detecting anomalies

- Visualization can help to detect anomalies
- Automatic approach: apply committee of different learning schemes, e.g.,
  - decision tree
  - nearest-neighbor learner
- Conservative *consensus* approach: delete instances incorrectly classified by all of them
  - Problem: might sacrifice instances of small classes

One-Class Learning

- Usually training data is available for all classes
- Some problems exhibit only a single class at training time
- Test instances may belong to this class or a new class not present at training time
- This the problem of *one-class classification*
  - Predict either *target* or *unknown*

Outlier detection

- One-class classification is often used for *outlier/anomaly/novelty* detection
- First, a one-class models is built from the dataset
- Then, outliers are defined as instances that are classified as *unknown*
- Another method: identify outliers as instances that lie beyond distance *d* from percentage *p* of training data
- Density estimation is a very useful approach for one-class classification and outlier detection
  - Estimate density of the target class and mark low probability test instances as outliers
  - Threshold can be adjusted to calibrate sensitivity

Transforming multiple classes to binary ones

- Some learning algorithms only work with two class problems
- Sophisticated multi-class variants exist in many cases but can be very slow or difficult to implement
- A common alternative is to transform multi-class problems into multiple two-class ones
- Simple methods:
  - Discriminate each class against the union of the others – *one-vs.-rest*
  - Build a classifier for every pair of classes – *pairwise classification*

Error-correcting output codes

- Multiclass problem vs. multiple binary problems
  - Simple one-vs.rest scheme: One-per-class coding
- Idea: use *error-correcting codes* instead
  - base classifiers predict 1011111, true class = ??
- Use bit vectors (codes) sot that we have large *Hamming distance* between any pair of bit vectors:
  - Can correct up to $(d - 1)/2$ single-bit errors

class	class vector
a	1000
b	0100
c	0010
d	0001

class	class vector
a	1111111
b	0000111
c	0011001
d	0101010

Exhaustive ECOCs

- *Exhaustive* code for *k* classes:
  - Columns comprise every possible *k*-string …
  - … except for complements and all-zero/one strings
  - Each code word contains $2^{k-1} - 1$ bits
- Class 1: code word is all ones
- Class 2: 2^{k-2} zeroes followed by $2^{k-2} - 1$ ones
- Class *i* : alternating runs of 2^{k-i} 0s and 1s
  - last run is one short

Exhaustive code, *k* = 4

class	class vector
a	1111111
b	0000111
c	0011001
d	0101010

Calibrating class probabilities

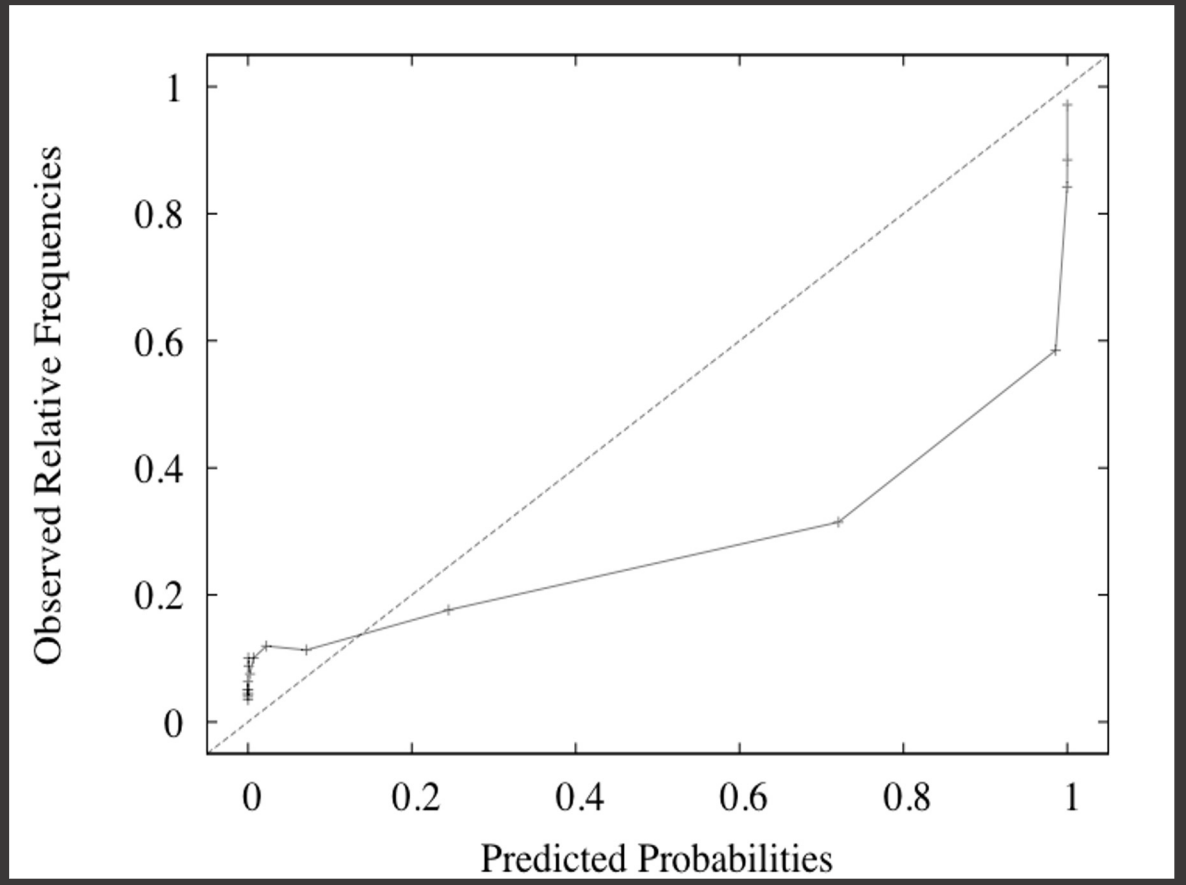
- Class probability estimation is harder than classification:
  - Classification error is minimized as long as the correct class is predicted with maximum probability
  - Estimates that yield correct classification may be quite poor with respect to quadratic or informational loss

- But: it is often important to have accurate class probabilities
  - E.g. cost-sensitive prediction using the minimum expected cost method

Visualizing inaccurate probability estimates

- Consider a two class problem. Probabilities that are correct for classification may be:
  - Too optimistic – too close to either 0 or 1
  - Too pessimistic – not close enough to 0 or 1

Reliability diagram showing overoptimistic probability estimation for a two-class problem



Calibrating class probabilities

- Can view calibration as a function estimation problem
  - One input – estimated class probability – and one output – the calibrated probability
- Reasonable assumption in many cases: the function is piecewise constant and monotonically increasing
- Can use *isotonic regression,* which estimates a monotonically increasing piece-wise constant function: Minimizes squared error between observed class "probabilities" (0/1) and resulting calibrated class probabilities
- Alternatively, can use *logistic regression* to estimate the calibration function
  - Note: must use the *log-odds* of the estimated class probabilities as input
- Advantage: multiclass logistic regression can be used for calibration in the multiclass case