Data Mining

Lesson 5

Instance-based Learning

MSc in Computer Science

University of New York Tirana

Assoc. Prof. Dr. Marenglen Biba

Data Mining: Content

- Introduction to data mining and machine learning
- Inductive learning
- Decision trees
- Rule induction
- Instance-based learning
- Bayesian learning
- Neural networks
- Support vector machines
- Other machine learning models
- Engineering data mining tasks

Lesson Outline

- Instance-based Learning
- Clustering

- Lab session
  - IBL in Weka
  - Clustering in Weka
  - Clustering in Oracle

Introduction

- Instance-based learning methods such as nearest neighbor and locally weighted regression are conceptually straightforward approaches to approximating real-valued or discrete-valued target functions.

- Learning in these algorithms consists of simply storing the presented training data.

- When a new query instance is encountered, a set of similar related instances is retrieved from memory and used to classify the new query instance.

Approximation of the target function

- One key difference between these approaches and the methods discussed before is that instance-based approaches can construct a different approximation to the target function for each distinct query instance that must be classified.

- In fact, many techniques construct only a local approximation to the target function that applies in the neighborhood of the new query instance, and never construct an approximation designed to perform well over the entire instance space.

- This has significant advantages when the target function is very complex, but can still be described by a collection of less complex local approximations.

Disadvantages of instance-based approaches

- One disadvantage of instance-based approaches is that the cost of classifying new instances can be high.
  - This is due to the fact that nearly all computation takes place at classification time rather than when the training examples are first encountered.
  - Therefore, techniques for efficiently indexing training examples are a significant practical issue in reducing the computation required at query time.

Disadvantages of instance-based approaches

- A second disadvantage to many instance-based approaches, especially nearest-neighbor approaches, is that they typically consider all attributes of the instances when attempting to retrieve similar training examples from memory.
  - If the target concept depends on only a few of the many available attributes, then the instances that are truly most "similar" may well be a large distance apart.

k-Nearest Neighbor Learning

- This is the most basic instance-based method.
- This algorithm assumes all instances correspond to points in the n-dimensional space R^n.
- The nearest neighbors of an instance are defined in terms of the standard Euclidean distance.
- More precisely, let an arbitrary instance x be described by the feature vector:

$$(a_1(x), a_2(x), \ldots, a_n(x))$$

- where $a_r(x)$ denotes the value of the rth attribute of instance x.
- Then the distance between two instances x_i, and x_j is defined to be $d(x_i, x_j)$ where:

$$d(x_i, x_j) \equiv \sqrt{\sum_{r=1}^{n} (a_r(x_i) - a_r(x_j))^2}$$

The *k*-nearest-neighbor algorithm

- In nearest-neighbor learning the target function may be either discrete-valued or real-valued.
- Let us first consider learning discrete-valued target functions of the form f: $R^n \rightarrow V$, where V is the finite set $\{v_1, \ldots v_s\}$.

Training algorithm:

- For each training example $\langle x, f(x) \rangle$, add the example to the list *training_examples*

Classification algorithm:

- Given a query instance x_q to be classified,
  - Let $x_1 \ldots x_k$ denote the k instances from *training_examples* that are nearest to x_q
  - Return

$$\hat{f}(x_q) \leftarrow \underset{v \in V}{\operatorname{argmax}} \sum_{i=1}^{k} \delta(v, f(x_i))$$

where $\delta(a, b) = 1$ if $a = b$ and where $\delta(a, b) = 0$ otherwise.

TABLE 8.1
The *k*-NEAREST NEIGHBOR algorithm for approximating a discrete-valued function $f : \Re^n \rightarrow V$.

The *k*-nearest-neighbor algorithm

- As shown, the value $\hat{f}(x_q)$ returned by this algorithm as its estimate of $f(x_q)$ is just the most common value of *f* among the k training examples nearest to x_q.

- If we choose k=1 then the 1-Nearest Neighbor algorithm assigns to $\hat{f}(x_q)$ the value $f(x_i)$ where x_i is the training instance nearest to x_q.

- For larger values of k, the algorithm assigns the most common value among the k nearest training examples.

k-nearest-neighbor classification

- Illustration the operation of the k-Nearest Neighbor algorithm for the case where the instances are points in a two-dimensional space and where the target function is boolean valued.

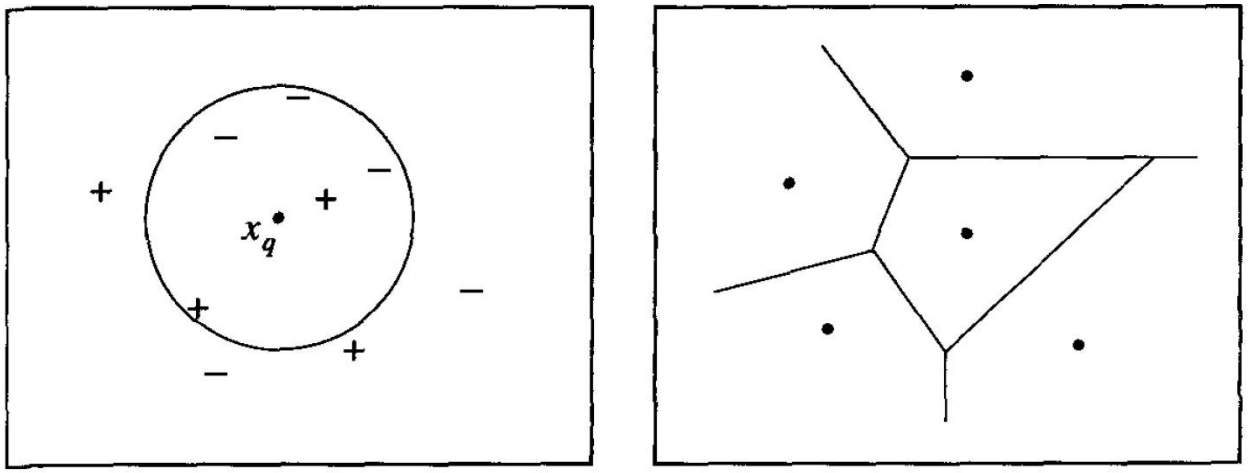


FIGURE 8.1
k-NEAREST NEIGHBOR. A set of positive and negative training examples is shown on the left, along with a query instance x_q to be classified. The 1-NEAREST NEIGHBOR algorithm classifies x_q positive, whereas 5-NEAREST NEIGHBOR classifies it as negative. On the right is the decision surface induced by the 1-NEAREST NEIGHBOR algorithm for a typical set of training examples. The convex polygon surrounding each training example indicates the region of instance space closest to that point (i.e., the instances for which the 1-NEAREST NEIGHBOR algorithm will assign the classification belonging to that training example).

Nature of the hypothesis space

- What is the nature of the hypothesis space H implicitly considered by the *k*-Nearest Neighbor algorithm?
  - Note the *k*-Nearest Neighbor algorithm never forms an explicit general hypothesis f^ regarding the target function f.
  - It simply computes the classification of each new query instance as needed.

- Nevertheless, we can still ask what the implicit general function is, or what classifications would be assigned if we were to hold the training examples constant and query the algorithm with every possible instance in X.

Voronoi diagram

- The diagram on the right side (Figure 8.1) shows the shape of this decision surface induced by 1-Nearest Neighbor over the entire instance space.
- The decision surface is a combination of convex polyhedra surrounding each of the training examples.
- For every training example, the polyhedron indicates the set of query points whose classification will be completely determined by that training example.
- Query points outside the polyhedron are closer to some other training example.
- This kind of diagram is often called the Voronoi diagram of the set of training examples.

Continuous-valued target functions.

- The k-Nearest Neighbor algorithm is easily adapted to approximating continuous-valued target functions.
- To accomplish this, we have the algorithm calculate the mean value of the k nearest training examples rather than calculate their most common value.
- More precisely, to approximate a real-valued target function f: $R^n \rightarrow R$ we replace the final line of the above algorithm by the line:

$$\hat{f}(x_q) \leftarrow \frac{\sum_{i=1}^{k} f(x_i)}{k}$$

Distance-Weighted Nearest Neighbor Algorithm

- One obvious refinement to the *k*-Nearest Neighbor algorithm is to weight the contribution of each of the *k* neighbors according to their distance to the query point x_q, giving greater weight to closer neighbors.
- For example, in the algorithm, which approximates discrete-valued target functions, we might weight the vote of each neighbor according to the inverse square of its distance from x_q.
- This can be accomplished by replacing the final line of the algorithm by:

$$\hat{f}(x_q) \leftarrow \underset{v \in V}{\operatorname{argmax}} \sum_{i=1}^{k} w_i \delta(v, f(x_i))$$

- where

$$w_i \equiv \frac{1}{d(x_q, x_i)^2} \qquad (8.3)$$

Distance-weight the instances for real-valued target functions

- We can distance-weight the instances for real-valued target functions in a similar fashion, replacing the final line of the algorithm in this case by:

$$\hat{f}(x_q) \leftarrow \frac{\sum_{i=1}^{k} w_i f(x_i)}{\sum_{i=1}^{k} w_i} \qquad (8.4)$$

- where w_i is as defined in Equation (8.3).

Issues in instance-based learning

Local vs. global methods

- All of the above variants of the *k*-Nearest Neighbor algorithm consider only the *k* nearest neighbors to classify the query point.

- Once we add distance weighting, there is really no harm in allowing all training examples to have an influence on the classification of the x_q, because very distant examples will have very little effect on $\hat{f}(x_q)$.

- The only disadvantage of considering all examples is that our classifier will run more slowly.

Local vs. global methods

- If all training examples are considered when classifying a new query instance, we call the algorithm a *global* method.

- If only the nearest training examples are considered, we call it a *local* method.

- When the rule in the equation in the previous slide is applied as a *global* method, using all training examples, it is known as Shepard's method (Shepard, 1968).

Remarks on *k*-Nearest Neighbor Algorithm

- The distance-weighted *k*-Nearest Neighbor algorithm is a highly effective inductive inference method for many practical problems.

- It is robust to noisy training data and quite effective when it is provided a sufficiently large set of training data.

- Note that by taking the weighted average of the k neighbors nearest to the query point, it can smooth out the impact of isolated noisy training examples.

Inductive bias of *k*-Nearest Neighbor?

- What is the inductive bias of *k*-Nearest Neighbor?

- *The inductive bias corresponds to an assumption that the classification of an instance x_q will be most similar to the classification of other instances that are nearby in Euclidean distance.*

Locally Weighted Regression

A Note on Terminology

- Much of the literature on nearest-neighbor methods and weighted local regression uses a terminology that has arisen from the field of statistical pattern recognition.
- In reading that literature, it is useful to know the following terms:
  - Regression means approximating a real-valued target function.
  - Residual is the error $\hat{f}(x) - f(x)$ in approximating the target function.
  - Kernel function is the function of distance that is used to determine the weight of each training example.
  - In other words, the kernel function is the function K such that $w_i = K(d(x_i, xq))$.

Locally Weighted Regression

- The nearest-neighbor approaches described previously can be thought of as approximating the target function *f(x)* at the single query point $x = x_q$.
- Locally weighted regression is a generalization of this approach.
- It constructs an explicit approximation to *f* over a local region surrounding x_q.
- Locally weighted regression uses nearby or distance-weighted training examples to form this local approximation to *f*.
- For example, we might approximate the target function in the neighborhood surrounding x_q using a linear function, a quadratic function, a multilayer neural network, or some other functional form.

Locally Weighted Regression

- The phrase "locally weighted regression“ (LWR) is called:
  - local because the function is approximated based only on data near the query point,
  - weighted because the contribution of each training example is weighted by its distance from the query point, and
  - regression because this is the term used widely in the statistical learning community for the problem of approximating real-valued functions.

Locally Weighted Regression

- Given a new query instance x_q the general approach in locally weighted regression is to construct an approximation $\hat{f}$ that fits the training examples in the neighborhood surrounding x_q.

- This approximation is then used to calculate the value $f(x_q)$, which is output as the estimated target value for the query instance.

- The description of f may then be deleted, because a different local approximation will be calculated for each distinct query instance.

Locally Weighted Linear Regression

- Let us consider the case of locally weighted regression in which the target function *f* is approximated near x_q using a linear function of the form

$$\hat{f}(x) = w_0 + w_1 a_1(x) + \ldots + w_n a_n(x)$$

- As before, $a_i(x)$ denotes the value of the *ith* attribute of the instance x.

- There are methods such as gradient descent, that find the coefficients $w_0 \ldots w_n$ to minimize the error in fitting such linear functions to a given set of training examples.

Clustering

Clusters

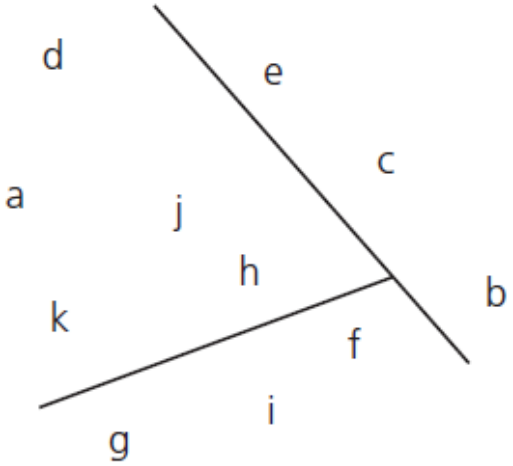
- When clusters rather than a classifier is learned, the output takes the form of a diagram that shows how the instances fall into clusters.

- In the simplest case this involves associating a cluster number with each instance, which might be depicted by laying the instances out in two dimensions and partitioning the space to show each cluster.

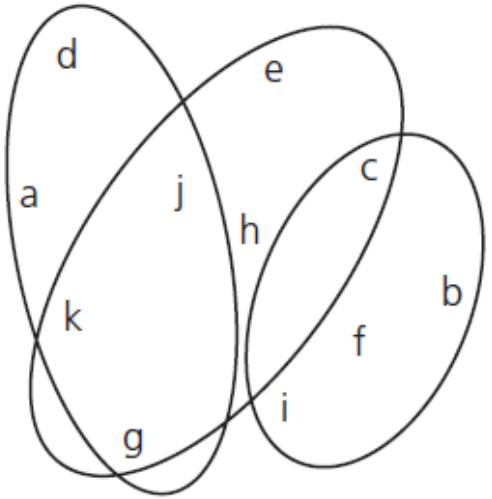
Clusters

- Some algorithms associate instances with clusters probabilistically rather than categorically.
- In this case, for every instance there is a probability or degree of membership with which it belongs to each of the clusters.
- This particular association is meant to be a probabilistic one, so the numbers for each example sum to one — although that is not always the case.
- Other algorithms produce a hierarchical structure of clusters so that at the top level the instance space divides into just a few clusters, each of which divides into its own subclusters at the next level down, and so on.

Representing clusters



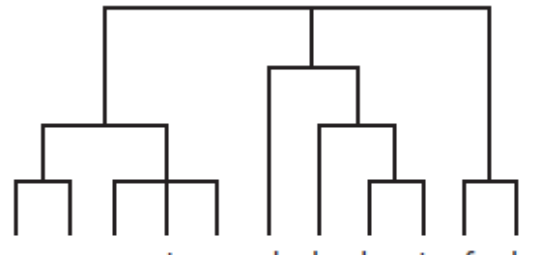
(a)



(b)

	1	2	3
a	0.4	0.1	0.5
b	0.1	0.8	0.1
c	0.3	0.3	0.4
d	0.1	0.1	0.8
e	0.4	0.2	0.4
f	0.1	0.4	0.5
g	0.7	0.2	0.1
h	0.5	0.4	0.1

(c)



(d)

Figure 3.9 Different ways of representing clusters.

Iterative distance-based clustering

- The classic clustering technique is called *k-means.*
- First, you specify in advance how many clusters are being sought: this is the parameter *k*.
- Then *k* points are chosen at random as cluster centers.
- All instances are assigned to their closest cluster center according to the ordinary Euclidean distance metric.
- Next the centroid, or mean, of the instances in each cluster is calculated — this is the “means” part.
- These centroids are taken to be new center values for their respective clusters.
- Finally, the whole process is repeated with the new cluster centers.
- Iteration continues until the same points are assigned to each cluster in consecutive rounds, at which stage the cluster centers have stabilized and will remain the same forever.

Iterative distance-based clustering

- This clustering method is simple and effective.
- *It is easy to prove that choosing the cluster center to be the centroid minimizes the total squared distance from each of the cluster's points to its center.*
- Once the iteration has stabilized, each point is assigned to its nearest cluster center, so the overall effect is to minimize the total squared distance from all points to their cluster centers.
- But the minimum is a local one; there is no guarantee that it is the global minimum.
- The final clusters are quite sensitive to the initial cluster centers.
- To increase the chance of finding a global minimum people often run the algorithm several times with different initial choices and choose the best final result — the one with the smallest total squared distance.

Choosing the number of clusters

- Suppose you are using *k*-means but do not know the number of clusters in advance.
- One solution is to try out different possibilities and see which is best — that is, which one minimizes the total squared distance of all points to their cluster center.
- A simple strategy is to start from a given minimum, perhaps *k* = 1, and work up to a small fixed maximum, using cross-validation to find the best value.
- Because *k*-means is slow, and cross-validation makes it even slower, it will probably not be feasible to try many possible values for *k*.
- Note that on the training data the “best” clustering according to the total squared distance criterion will always be to choose as many clusters as there are data points!

Choosing the number of clusters

- Another possibility is to begin by finding a few clusters and determining whether it is worth splitting them.

- You could choose $k = 2$, perform k-means clustering until it terminates, and then consider splitting each cluster.

- Computation time will be reduced considerably if the initial two-way clustering is considered irrevocable and splitting is investigated for each component independently.

Readings for this part

- Machine Learning. T. Mitchell
  - Chapter 8

- Data Mining. Witten and Frank
  - Sections 4.8, 6.6.

Lab Session

- Instance-based learning in Weka
- Clustering in Weka
- Clustering in Oracle

Housing problem

- **Data Set Information:**
  - Concerns housing values in suburbs of Boston.

  Attribute Information:
  - 1. CRIM: per capita crime rate by town
    2. ZN: proportion of residential land zoned for lots over 25,000 sq.ft.
    3. INDUS: proportion of non-retail business acres per town
    4. CHAS: Charles River dummy variable (= 1 if tract bounds river; 0 otherwise)
    5. NOX: nitric oxides concentration (parts per 10 million)
    6. RM: average number of rooms per dwelling
    7. AGE: proportion of owner-occupied units built prior to 1940
    8. DIS: weighted distances to five Boston employment centres
    9. RAD: index of accessibility to radial highways
    10. TAX: full-value property-tax rate per $10,000
    11. PTRATIO: pupil-teacher ratio by town
    12. B: 1000(Bk - 0.63)^2 where Bk is the proportion of blacks by town
    13. LSTAT: % lower status of the population
    14. MEDV: Median value of owner-occupied homes in $1000's

- Try different parameters for the algorithm!

Housing problem

IBK in Weka

- Try with default parameters
  - Correlation coefficient 0.8677
  - Mean absolute error 2.9794
  - Root mean squared error 4.6961
  - Relative absolute error 44.687 %
  - Root relative squared error 50.9388 %

- Try the following optimizations
  - KNN = 1
  - Cross-validate = true
  - Weight by 1/distance
  - Mean-squared = true

  - Correlation coefficient 0.8677
  - Mean absolute error 2.9794
  - Root mean squared error 4.6961
  - Relative absolute error 44.687 %
  - Root relative squared error 50.9388 %

Housing

Try the following optimizations

- KNN = 5
- Cross-validate = false
- Weight by 1/distance
- Mean-squared = true
  - Correlation coefficient 0.8917
  - Mean absolute error 2.7268
  - Root mean squared error 4.2732
  - Relative absolute error 40.8973 %
  - Root relative squared error 46.3523 %

- KNN = 10
- Cross-validate = false
- Weight by 1/distance
- Mean-squared = true
  - Correlation coefficient 0.8703
  - Mean absolute error 3.0125
  - Root mean squared error 4.7113
  - Relative absolute error 45.183 %
  - Root relative squared error 51.1041 %

Housing

- KNN = 10
- Cross-validate = false
- Weight by 1 - distance
- Mean-squared = true
  - Correlation coefficient 0.8274
  - Mean absolute error 3.3396
  - Root mean squared error 5.3186
  - Relative absolute error 50.0881 %
  - Root relative squared error 57.6913 %

Housing

- LWL in Weka
  - Classifier: DecisionStump
  - Weighting function:
    - [0 = Linear, 1 = Epanechnikov,2 = Tricube, 3 = Inverse, 4 = Gaussian and 5 = Constant. (default 0 = Linear)].
  - Correlation coefficient 0.8016
  - Mean absolute error 4.1837
  - Root mean squared error 5.5604
  - Relative absolute error 62.7492 %
  - Root relative squared error 60.3141 %

  - Classifier: RepTree
    - Correlation coefficient 0.8759
    - Mean absolute error 2.8832
    - Root mean squared error 4.4416
    - Relative absolute error 43.2442 %
    - Root relative squared error 48.1787 %

CPU performance problem

Data Set Information:

- The estimated relative performance values were estimated by the authors using a linear regression method. See their article (pp 308-313) for more details on how the relative performance values were set.

Attribute Information:

- 1. vendor name: 30
  (adviser, amdahl,apollo, basf, bti, burroughs, c.r.d, cambex, cdc, dec, dg, formation, four-phase, gould, honeywell, hp, ibm, ipl, magnuson, microdata, nas, ncr, nixdorf, perkin-elmer, prime, siemens, sperry, sratus, wang)
  2. Model Name: many unique symbols
  3. MYCT: machine cycle time in nanoseconds (integer)
  4. MMIN: minimum main memory in kilobytes (integer)
  5. MMAX: maximum main memory in kilobytes (integer)
  6. CACH: cache memory in kilobytes (integer)
  7. CHMIN: minimum channels in units (integer)
  8. CHMAX: maximum channels in units (integer)
  9. PRP: published relative performance (integer)
  10. ERP: estimated relative performance from the original article (integer)

CPU performance problem

IBK in Weka

- Try with default parameters
  - Correlation coefficient 0.9467
  - Mean absolute error 20.8278
  - Root mean squared error 53.6354
  - Relative absolute error 23.7602 %
  - Root relative squared error 34.6563 %

- Some optimizations
  - The same results are obtained

Auto: Miles per gallon problem

- The data concerns city-cycle fuel consumption in miles per gallon, to be predicted in terms of 3 multivalued discrete and 5 continuous attributes.
- Attributes
    1. mpg: continuous
    2. cylinders: multi-valued discrete
    3. displacement: continuous
    4. horsepower: continuous
    5. weight: continuous
    6. acceleration: continuous
    7. model year: multi-valued discrete
    8. origin: multi-valued discrete
    9. car name: string (unique for each instance)

Auto Mpg

- Correlation coefficient 0.8964
- Mean absolute error 2.505
- Root mean squared error 3.5384
- Relative absolute error 38.2516 %
- Root relative squared error 45.1882 %

- Some optimizations
  - KNN = 1
  - Cross-validate = true
    - IB1 instance-based classifier using 4 inverse-distance-weighted nearest neighbour(s) for classification
  - Weight by 1/distance
  - Mean-squared = true

  - Correlation coefficient 0.9106
  - Mean absolute error 2.2708
  - Root mean squared error 3.2278
  - Relative absolute error 34.6756 %
  - Root relative squared error 41.221 %

Auto Mpg

- KNN = 10
  - Cross-validate = false
  - Weight by 1/distance
  - Mean-squared = true
    - Correlation coefficient 0.9097
    - Mean absolute error 2.2979
    - Root mean squared error 3.2425
    - Relative absolute error 35.0895 %
    - Root relative squared error 41.4087 %
- KNN = 30
  - Cross-validate = false
  - Weight by 1/distance
  - Mean-squared = true
    - Relative absolute error 36.3831 %
    - Root relative squared error 43.5612 %
  - KNN=100
    - Relative absolute error 43.7969 %
    - Root relative squared error 50.298 %

Auto price problem

- **Attribute Information:**
- Attribute: Attribute Range

1. symboling: -3, -2, -1, 0, 1, 2, 3.
2. normalized-losses: continuous from 65 to 256.
3. make:
alfa-romero, audi, bmw, chevrolet, dodge, honda,
isuzu, jaguar, mazda, mercedes-benz, mercury,
mitsubishi, nissan, peugot, plymouth, porsche,
renault, saab, subaru, toyota, volkswagen, volvo

4. fuel-type: diesel, gas.
5. aspiration: std, turbo.
6. num-of-doors: four, two.
7. body-style: hardtop, wagon, sedan, hatchback, convertible.
8. drive-wheels: 4wd, fwd, rwd.
9. engine-location: front, rear.
10. wheel-base: continuous from 86.6 120.9.
11. length: continuous from 141.1 to 208.1.
12. width: continuous from 60.3 to 72.3.
13. height: continuous from 47.8 to 59.8.
14. curb-weight: continuous from 1488 to 4066.
15. engine-type: dohc, dohcv, l, ohc, ohcf, ohcv, rotor.
16. num-of-cylinders: eight, five, four, six, three, twelve, two.
17. engine-size: continuous from 61 to 326.
18. fuel-system: 1bbl, 2bbl, 4bbl, idi, mfi, mpfi, spdi, spfi.
19. bore: continuous from 2.54 to 3.94.
20. stroke: continuous from 2.07 to 4.17.
21. compression-ratio: continuous from 7 to 23.
22. horsepower: continuous from 48 to 288.
23. peak-rpm: continuous from 4150 to 6600.
24. city-mpg: continuous from 13 to 49.
25. highway-mpg: continuous from 16 to 54.
26. price: continuous from 5118 to 45400.

Auto price problem

IBK

- Default parameters
  - Correlation coefficient 0.8738
  - Mean absolute error 1609.8648
  - Root mean squared error 2902.5515
  - Relative absolute error 34.8291 %
  - Root relative squared error 49.0919 %

Parameter optimization

- KNN = 1
- Cross-validate = true
  - IB1 instance-based classifier using 1 inverse-distance-weighted nearest neighbour(s) for classification
- Weight by 1/distance
- Mean-squared = true
  - Correlation coefficient 0.8738
  - Mean absolute error 1609.8648
  - Root mean squared error 2902.5515
  - Relative absolute error 34.8291 %
  - Root relative squared error 49.0919 %

Auto price problem

Parameter optimization

- KNN = 5
- Cross-validate = false
- Weight by 1/distance
- Mean-squared = true
  - Correlation coefficient 0.878
  - Mean absolute error 1666.5529
  - Root mean squared error 2953.8493
  - Relative absolute error 36.0555 %
  - Root relative squared error 49.9595 %

Auto horse power problem

- Horsepower treated as the class attribute.

IBK

- Default parameters
  - Correlation coefficient 0.8759
  - Mean absolute error 8.4089
  - Root mean squared error 19.6074
  - Relative absolute error 27.4415 %
  - Root relative squared error 49.293 %

Clustering in Weka

- Parameters
  - Type of algorithm
- Clustering
  - Iris
    - Simple K-means
      - 2 clusters (sum of squared errors: 62.143)
      - 3 clusters (sum of squared errors: 7.81.
    - EM
      - Log likelihood: -2.03504
    - DensityBasedCluster
      - Log likelihood: -3.06315

- Credit
- Hepatitis

Clustering in Oracle

- Oracle Tutorial on clustering
  - Customer segmentation on Insurance data

End of class