Machine Learning (CE 40477)

Fall 2024

Ali Sharifi-Zarchi

CE Department
Sharif University of Technology

November 22, 2025

1. Overview
2. k-Nearest Neighbor
3. Distance Measures
4. kNN Regression
5. References

1 Overview

2 k-Nearest Neighbor

3 Distance Measures

4 kNN Regression

5 References

Parametric vs. Non-Parametric Methods

- **Parametric** methods **learn parameters** from data and then use these inferred parameters to make predictions on new data points.
  - **Learning:** estimating parameters from data.
  - Examples include **Linear Regression** and **Logistic Regression**.
- **Non-parametric** methods:
  - Training examples are used **directly**.
  - A separate **training phase** is **not required**.
  - Example: **k-Nearest Neighbors (kNN)**.
- Both supervised and unsupervised learning methods can be categorized as either parametric or non-parametric.

Outline

Non-Parametric Approach

- **Unsupervised: Non-Parametric Density Estimation**
  - Parzen Windows.
  - k-Nearest Neighbor Density Estimation.
- **Supervised: Instance-Based Learners**
  - Classification:
    - kNN Classification.
    - Weighted (or Kernel) kNN.
  - Regression:
    - kNN Regression.
    - Locally Linear Weighted Regression.

1 Overview

2 k-Nearest Neighbor

3 Distance Measures

4 kNN Regression

5 References

kNN Concept

- "First, tell me who you have lived with, and then I will tell you who you are."

Figure adapted from https://setare.com

k-Nearest Neighbor (kNN)

- **kNN Classifier:** considers the $k \geq 1$ nearest neighbors of a query point.
  - The label of $\mathbf{x}$ is predicted by **majority voting** among its k nearest neighbors.
- Example: $k = 5$, $\mathbf{x} = [x_1, x_2]$

Figure adapted from M. Soleymani, ML slides, Sharif University of Technology.

k-Nearest Neighbor (kNN) Classifier

- **Given:**
  - Training data $\{(\mathbf{x}^{(1)}, \mathbf{y}^{(1)}), \ldots, (\mathbf{x}^{(n)}, y^{(n)})\}$ are stored directly (no model parameters are learned).
- **To classify a new sample $\mathbf{x}$:**
  - Find the k nearest training samples to $\mathbf{x}$.
  - Among these k samples, count how many k_j belong to each class C_j ($j = 1, \ldots, C$).
  - Assign $\mathbf{x}$ to the class C_{j^*}, where
$$j^* = \underset{j=1,\ldots,C}{\arg\max}\, k_j.$$
- The kNN algorithm is considered a **discriminative** method.

k-Nearest Neighbor (kNN) Classifier

- The **kNN classifier** can produce **non-linear decision boundaries**, unlike previous methods such as linear and logistic regression.
- However, this method can be quite sensitive to **outliers** or **noisy data**, particularly when:
  - The dataset is **small**.
  - The data are **low-dimensional**.
  - A **small value of** k is used — for instance, when $k = 1$, the prediction depends only on the single nearest neighbor, which can be misleading in many test cases.

kNN Example

- We aim to classify a new document and assign it to one of three categories by examining its neighboring samples.

kNN Example

1-Nearest Neighbor

2-Nearest Neighbors

3-Nearest Neighbors

5-Nearest Neighbors

Voronoi Tessellation

- **Voronoi Tessellation:**
  - Each cell contains all points that are closer to a given training sample than to any other sample.
  - All points within a cell share the label of the corresponding training sample.

Voronoi Tessellation

- The 1-NN decision regions form a Voronoi tessellation.

Figure adapted from R. Zhu, "Stat 542: Statistical Learning – k-Nearest Neighbor and the Bias–Variance Trade-Off," Lecture Notes.

Reducing Complexity: Editing 1-NN

- If all Voronoi neighbors of a sample belong to the same class, that sample is redundant and can be safely removed.

- The number of stored samples decreases.
- The decision boundaries are guaranteed to remain unchanged.

Nearest Neighbor Classifier: Error Bound

- **Nearest Neighbor:** kNN with $k = 1$.
  - **Decision rule:**

$$\hat{y} = y^{NN(\mathbf{x})} \quad \text{where} \quad NN(x) = \arg\min_{i=1,\dots,N} \left\| \mathbf{x} - \mathbf{x}^{(i)} \right\|.$$

- **Cover & Hart (1967):** The asymptotic risk of the NN classifier satisfies

$$R^* \le R_\infty^{NN} \le 2R^*(1 - R^*) \le 2R^*.$$

R_n: expected risk of the NN classifier with n training examples drawn from $\mathbb{P}(\mathbf{x}, \mathbf{y})$.
$R_\infty^{NN} = \lim_{n\to\infty} R_n^{NN}$.
R^*: the optimal Bayes risk.

Cover & Hart (1967)

Nearest Neighbor Pattern Classification

T. M. COVER, MEMBER, IEEE, AND P. E. HART, MEMBER, IEEE

Abstract—The nearest neighbor decision rule assigns to an unclassified sample point the classification of the nearest of a set of previously classified points. This rule is independent of the underlying joint distribution on the sample points and their classifications, and hence the probability of error R of such a rule must be at least as great as the Bayes probability of error R^*—the minimum probability of error over all decision rules taking underlying probability structure into account. However, in a large sample analysis, we will show in the M-category case that $R^* \leq R \leq R^*(2 - MR^*/(M-1))$, where these bounds are the tightest possible, for all suitably smooth underlying distributions. Thus for any number of categories, the probability of error of the nearest neighbor rule is bounded above by twice the Bayes probability of error. In this sense, it may be said that half the classification information in an infinite sample set is contained in the nearest neighbor.

I. Introduction

In the classification problem there are two extremes of knowledge which the statistician may possess. Either he may have complete statistical knowledge of the underlying joint distribution of the observation x and the true category θ, or he may have no knowledge of the underlying distribution except that which can be inferred from samples. In the first extreme, a standard Bayes analysis will yield an optimal decision procedure and the corresponding minimum (Bayes) probability of error of classification R^*. In the other extreme, a decision to classify x into category $\hat{\theta}$ is allowed to depend only on a collection of n correctly classified samples $(x_1, \theta_1), (x_2, \theta_2), \cdots, (x_n, \theta_n)$, and the decision procedure is by no means clear. This problem is in the domain of nonparametric statistics and no optimal classification procedure exists with respect to all underlying statistics.

If it is assumed that the classified samples (x_i, θ_i) are *independently identically distributed* according to the distribution of (x, θ), certain heuristic arguments may be made about good decision procedures. For example, it is reasonable to assume that observations which are close together (in some appropriate metric) will have the same classification, or at least will have almost the same posterior probability distributions on their respective classifications. Thus to classify the unknown sample x we may wish to weight the evidence of the nearby x_i's most heavily. Perhaps the simplest nonparametric decision procedure of this form is the *nearest neighbor* (NN) rule, which classifies x in the category of its nearest neighbor. Surprisingly, it will be shown that, in the large sample case, this simple rule has a probability of error which

Manuscript received February 23, 1966; revised April 29, 1966. This work has been supported at Stanford University by U. S. Army Electronics Command under Contract DA28-043-AMC-01764(E) and by USAF under Contract AF49(638)1517; and at the Stanford Research Institute, Menlo Park, Calif., by RAOC under Contract AF30(602)3945.

T. M. Cover is with the Department of Electrical Engineering, Stanford University, Stanford, Calif.

P. E. Hart is with the Stanford Research Institute, Menlo Park, Calif.

isl.stanford.edu/~cover/papers/transIT/0021cove.pdf

Bayes Optimal Classifier

Assume (although this is almost never the case) that we know $\mathbb{P}(y \mid \mathbf{x})$. Then, we can simply predict the most likely label.

$$\text{The Bayes optimal classifier predicts:} \quad y^* = \arg\max_y \mathbb{P}(y \mid \mathbf{x})$$

Although the Bayes optimal classifier performs as well as possible, it can still make mistakes. It is wrong whenever a sample does not belong to the most likely class. We can compute the probability of that happening exactly, which gives the uncertainty rate:

$$\epsilon_{\text{BayesOpt}} = 1 - \mathbb{P}(y^*(\mathbf{x}) \mid \mathbf{x}) = 1 - \mathbb{P}(y^* \mid \mathbf{x})$$

Bayes Optimal Classifier: Example

Assume, for example, that an email $\mathbf{x}$ can be classified as either spam $(+1)$ or ham (-1). For the same email $\mathbf{x}$, the conditional class probabilities are:

$$\mathbb{P}(+1 \mid \mathbf{x}) = 0.8 \qquad \mathbb{P}(-1 \mid \mathbf{x}) = 0.2$$

In this case, the Bayes optimal classifier would predict the label:

$$y^* = +1$$

since it is the most likely one, and its uncertainty rate would be:

$$\epsilon_{\text{BayesOpt}} = 0.2$$

Why is the Bayes optimal classifier interesting if it cannot be used in practice? The reason is that it provides a highly informative lower bound on the uncertainty rate. Given the same feature representation, no classifier can achieve a lower uncertainty. We use this fact to analyze the error rate of the kNN classifier.

1-NN Convergence Proof

Cover and Hart (1967): As $n \to \infty$, the 1-NN error is no more than twice the uncertainty of the Bayes optimal classifier. (Similar guarantees also hold for $k > 1$.)

n small $\qquad$ n large $\qquad$ $n \to \infty$

- As $n \to \infty$, each test point $\mathbf{x}_t$ becomes almost identical to its nearest neighbor $\mathbf{x}_{NN}$, so

$$\mathbb{P}(y^* \mid \mathbf{x}_t) \approx \mathbb{P}(y^* \mid \mathbf{x}_{NN}).$$

- An error occurs only when their labels differ, with probability

$$2\,\mathbb{P}(y^* \mid \mathbf{x}_t)\big(1 - \mathbb{P}(y^* \mid \mathbf{x}_t)\big) \le 2\,\epsilon_{\text{BayesOpt}}.$$

- Thus, the 1-NN error is **at most twice** the Bayes optimal uncertainty.

1-NN Convergence Proof

Let $\mathbf{x}_{NN}$ be the nearest neighbor of the test point $\mathbf{x}_t$. As $n \to \infty$, the distance $\text{dist}(\mathbf{x}_{NN}, \mathbf{x}_t)$ approaches zero.

$$
\begin{aligned}
\epsilon_{NN} &= \mathbb{P}(y^* \mid \mathbf{x}_t)\big(1 - \mathbb{P}(y^* \mid \mathbf{x}_{NN})\big) + \mathbb{P}(y^* \mid \mathbf{x}_{NN})\big(1 - \mathbb{P}(y^* \mid \mathbf{x}_t)\big) \\
&\leq \big(1 - \mathbb{P}(y^* \mid \mathbf{x}_{NN})\big) + \big(1 - \mathbb{P}(y^* \mid \mathbf{x}_t)\big) \\
&= 2\big(1 - \mathbb{P}(y^* \mid \mathbf{x}_t)\big) = 2\,\epsilon_{\text{BayesOpt}}.
\end{aligned}
$$

where the inequality follows from $\mathbb{P}(y^* \mid \mathbf{x}_t) \leq 1$, and we assume that $\mathbb{P}(y^* \mid \mathbf{x}_t) = \mathbb{P}(y^* \mid \mathbf{x}_{NN})$.

1-NN Convergence Proof

Possible labels: Spam, Ham (non-spam email)

In the limiting case, the test point and its nearest neighbor are identical. There are exactly two cases in which a misclassification can occur — when the test point and its nearest neighbor have different labels. The probability of this event is given by the sum of the two red regions:

$$\big(1-\mathbb{P}(s\,|\,\mathbf{x})\big)\mathbb{P}(s\,|\,\mathbf{x})+\mathbb{P}(s\,|\,\mathbf{x})\big(1-\mathbb{P}(s\,|\,\mathbf{x})\big)=2\,\mathbb{P}(s\,|\,\mathbf{x}\big(1-\mathbb{P}(s\,|\,\mathbf{x})\big).$$

Curse of Dimensionality

Good News: As $n \to \infty$, the 1-NN classifier performs at most twice as poorly as the best possible classifier.

Bad News: We are cursed.

Idea: The kNN classifier assumes that similar points have similar labels. However, in high-dimensional spaces, randomly drawn points are rarely close to each other.

Pitfalls: The Curse of Dimensionality

- Low-dimensional visualizations can be misleading. In high-dimensional spaces, most points are far apart.
- If we want the nearest neighbor to be closer than ϵ, how many points are required to guarantee this?
- The volume of a single ball with radius ϵ is $O(\epsilon^d)$.
- The total volume of $[0,1]^d$ is 1.
- Therefore, $O\left(\left(\frac{1}{\epsilon}\right)^d\right)$ balls are needed to cover the entire volume.

Example: Points are uniformly sampled within the unit cube $[0,1]^d$. Let ℓ be the edge length of the smallest hypercube that contains all k-nearest neighbors of a test point.

Curse of Dimensionality

$$\ell^d \approx \frac{k}{n} \quad \Rightarrow \quad \ell \approx \left(\frac{k}{n}\right)^{1/d}.$$

For $n = 1000$ and $k = 10$, we have:

d	ℓ
2	0.10
10	0.63
100	0.955
1000	0.9954

Observation: As d increases, almost the entire space is needed to find the k-nearest neighbors, breaking the fundamental assumption of the kNN classifier.

Curse of Dimensionality

Distance Distributions: The histograms below show the pairwise distances between random points in d-dimensional unit cubes. As d increases, these distances concentrate within a very narrow range, meaning that almost all points are equally far apart.

Curse of Dimensionality

Consequence: In high-dimensional spaces, the "nearest" neighbors are not truly close or similar. This breaks the fundamental assumption underlying the kNN classifier.

Data Growth Requirement: To make ℓ small, we must increase n:

$$\ell = \frac{1}{10} \quad \Rightarrow \quad n = \frac{k}{\ell^d} = k \times 10^d.$$

This quantity grows exponentially with d. For $d > 100$, we would need more data points than there are electrons in the universe.

Effect of k

- Compare the results for $k = 1$ and $k = 15$.
- As k increases, the model becomes smoother and less complex.

Effect of k

Choice of k

- **In practice:**
  - k should be large enough to minimize the overall error rate.
    - A very small k produces noisy and unstable decision boundaries.
  - k should also be small enough to ensure that only nearby samples are taken into account.
    - A very large k produces overly smooth boundaries that may ignore local patterns.
- Balancing these two requirements is not trivial.
  - This is a common challenge in machine learning — we need to smooth data, but not too much.

Choice of k: Example

- For $k = 1, \ldots, 7$, the point x is correctly classified as belonging to the red class.
- For larger values of k, the classification of x becomes incorrect, and it is assigned to the blue class.
- **Bias-Variance Trade-Off in kNN:**
  - Larger $k \Rightarrow$ underfitting (high bias, low variance).
  - Smaller $k \Rightarrow$ overfitting (low bias, high variance).

Validation and Test Sets

- k is an example of a **hyperparameter** — a parameter that cannot be directly learned during the training process.
- We can tune hyperparameters by evaluating model performance on a separate validation set.

- The test set is used only once, at the very end, to evaluate the generalization performance of the final trained model.

Computational Complexity

What is the computational complexity of the kNN algorithm?

- The basic kNN algorithm stores all training examples. Suppose we have n examples, each with d features.
  - Computing the distance to a single example takes $O(d)$ time.
  - Computing distances to all n examples takes $O(nd)$ time.
  - Selecting the k nearest can be done in $O(n \log k)$ time using a **max-heap of size** k.
  - Therefore, the total time complexity of kNN classification is:

$$O(nd + n \log k).$$

  - For fixed small k, this is effectively $O(nd)$.
- This becomes prohibitively expensive when the number of samples and dimensions are both large.

Reducing Complexity: kNN Prototypes

Instead of storing every individual data point, we can group similar samples together and represent each group by a prototype. These prototypes are organized into search trees, where:

- Each node represents a cluster of similar data points.
- The root node summarizes several prototypes or subgroups.

During classification, we only search within the most relevant branches, not the entire dataset.

- **Advantages:** The computational complexity decreases.
- **Disadvantages:**
  - Finding a good search tree is not trivial.
  - It may not always find the true nearest neighbor, and therefore it is **not** guaranteed that the decision boundaries remain unchanged.

kNN Prototypes: Example

- The tree on the left represents class 1, and the tree on the right represents class 2.
- Each gray box (e.g., 147, 253) denotes a prototype node that summarizes several samples.
- The child nodes can be either:
  - Other prototypes representing subgroups, or
  - Individual samples.
- In this way, the data are organized hierarchically to enable efficient search.

Advantages and Disadvantages of kNN

- **Advantages:**
  - Can be applied to data from any distribution.
  - Very simple and intuitive to implement.
  - Provides good classification performance if the number of samples is sufficiently large.
- **Disadvantages:**
  - Selecting the best value of k can be challenging.
  - Computationally expensive, although certain optimizations are possible.
  - Requires a large number of samples to achieve high accuracy.
    - This limitation cannot be overcome without assuming a parametric distribution.

1 Overview

2 k-Nearest Neighbor

3 Distance Measures

4 kNN Regression

5 References

Instance-Based Learner

- The main components required to construct an instance-based learner are:
  - A distance metric to measure similarity between data points.
  - The number of nearest neighbors of the test instance to consider.
  - A weighting function (optional).
  - A method to determine the output based on the neighbors.

Distance Measures

- **Minkowski Distance:**

$$d(\mathbf{x}, \mathbf{x}') = \left(\sum_{i=1}^{n} |x_i - x_i'|^p \right)^{\frac{1}{p}}$$

  - For $p \geq 1$, this is a valid distance metric.
  - When $p = 2$, the Minkowski distance is equivalent to the Euclidean distance.
  - The Minkowski distance is also equal to the L^p norm of $(\mathbf{x} - \mathbf{x}')$.
- Recall the definition of the L^p norm from linear algebra:

$$\|\mathbf{x}\|_p = \sqrt[p]{|x_1|^p + \cdots + |x_n|^p}.$$

$$\text{Some common } L^p \text{ norms: } \begin{cases} \|\mathbf{x}\|_1 & = \sum_{i=1}^{n} |x_i|, \\ \|\mathbf{x}\|_2 & = \sqrt{x_1^2 + \cdots + x_n^2}, \\ \|\mathbf{x}\|_\infty & = \max\{|x_1|, |x_2|, \ldots, |x_n|\}. \end{cases}$$

Distance Measures

- **Euclidean Distance:**

$$d(\mathbf{x}, \mathbf{x}') = \sqrt{(x_1 - x_1')^2 + \cdots + (x_d - x_d')^2}.$$

- **Distance Learning Methods:**
  - **Weighted Euclidean Distance:**

$$d_w(\mathbf{x}, \mathbf{x}') = \sqrt{w_1(x_1 - x_1')^2 + \cdots + w_d(x_d - x_d')^2}.$$

  - Large weight $w_i \Rightarrow$ attribute i is more important.
  - Small weight $w_i \Rightarrow$ attribute i is less important.
  - Zero weight $w_i \Rightarrow$ attribute i does not contribute to the distance.

Distance Measures

L_1 (Manhattan) Distance:

$$d_1(\mathbf{I}_1, \mathbf{I}_2) = \sum_p \left| \mathbf{I}_1^p - \mathbf{I}_2^p \right|.$$

L_2 (Euclidean) Distance:

$$d_2(\mathbf{I}_1, \mathbf{I}_2) = \left(\sum_p (\mathbf{I}_1^p - \mathbf{I}_2^p)^2 \right)^{\frac{1}{2}}.$$

Distance Measures ($k = 1$)

L_1 (Manhattan) Distance:

$$d_1(\mathbf{I}_1, \mathbf{I}_2) = \sum_p \left| \mathbf{I}_1^p - \mathbf{I}_2^p \right| .$$

L_2 (Euclidean) Distance:

$$d_2(\mathbf{I}_1, \mathbf{I}_2) = \left(\sum_p (\mathbf{I}_1^p - \mathbf{I}_2^p)^2 \right)^{\frac{1}{2}} .$$

Distance Measures

Cosine Distance (Angle-Based Similarity)

- Measures the angle between vectors; scale-invariant.
- Often used for text or high-dimensional sparse data.
- Note: **Not a true metric** (triangle inequality may fail).

$$d(\mathbf{x}, \mathbf{x}') = 1 - \frac{\mathbf{x} \cdot \mathbf{x}'}{\|\mathbf{x}\|_2 \, \|\mathbf{x}'\|_2}$$

Effect of Distance Measure

$$d(\boldsymbol{x}, \boldsymbol{x}') = \sqrt{(x_1 - x_1')^2 + (x_2 - x_2')^2}$$

$$d(\boldsymbol{x}, \boldsymbol{x}') = \sqrt{(x_1 - x_1')^2 + 3(x_2 - x_2')^2}$$

Effect of different distance measures on classification boundaries.

Figures adapted from M. Soleymani, *Machine Learning Slides*, Sharif University of Technology.

Effect of Distance Measure

You can be quite creative in choosing the distance function. For example, one can use *bending energy*, which measures how much transformation is required to make one example resemble another.

Fig. 1. Examples of two handwritten digits. In terms of pixel-to-pixel comparisons, these two images are quite different, but to the human observer, the shapes appear to be similar.

Fig. 4. Illustration of the matching process applied to the example of Fig. 1. Top row: 1st iteration. Bottom row: 5th iteration. Left column: estimated correspondences shown relative to the transformed model, with tangent vectors shown. Middle column: estimated correspondences shown relative to the untransformed model. Right column: result of transforming the model based on the current correspondences; this is the input to the next iteration. The grid points illustrate the interpolated transformation over $\mathbb{R}^2$. Here, we have used a regularized TPS model with $\lambda_o = 1$.

1 Overview

2 k-Nearest Neighbor

3 Distance Measures

4 kNN Regression

5 References

kNN Regression

- Let $\mathbf{x}'^{(1)}, \ldots, \mathbf{x}'^{(k)}$ be the k-nearest neighbors of the query point $\mathbf{x}$, and let $y'^{(1)}, \ldots, y'^{(k)}$ be their corresponding target values:

$$\hat{y} = \frac{1}{k} \sum_{j=1}^{k} y'^{(j)}.$$

- Some challenges of kNN regression when fitting functions:
  - Discontinuities in the estimated function.
  - For 1-NN: highly sensitive to noise (overfitting).
  - For $k > 1$: averaging reduces noise sensitivity but may lead to excessive smoothing, especially near the boundaries.

kNN Regression

Dataset 1 Dataset 2 Dataset 3

$k = 1$

$k = 9$

Figures adapted from Andrew Moore's tutorial on "Instance-Based Learning"

kNN Regression: Example

- Suppose we have a dataset with a single feature uniformly distributed over $[0, 2\pi]$.
- The true model is defined as:

$$Y = 2\sin(X) + \epsilon,$$

  where ϵ is a standard normal error term.
- We simulate 200 observations and visualize the fitted model for $k = 1$.

Example of kNN regression with $k = 1$.

kNN Regression: Example

- Results for $k = 2$ and $k = 10$.

Figures adapted from Andrew Moore's tutorial on *Instance-Based Learning*.

kNN Regression: Example

- As k increases, the model becomes smoother. However, if k is too large, it starts to deviate from the true underlying function.

kNN: Universal Approximation

As the number of training samples approaches infinity, the nearest neighbor method can approximate any function*!

The kNN regression fits a piecewise constant function (green), which approximates the true curve (blue) by averaging the nearby training samples (black).

* Subject to several technical conditions: only continuous functions on a compact domain, with assumptions on the spacing of training points, etc.

Locally Weighted Linear Regression

- For each test sample, the model produces a linear approximation of the target function within a local region.
- Instead of computing the output via weighted averaging (as in kernel regression), we fit a parametric function locally:

$$\hat{y} = f\left(\mathbf{x}, \mathbf{x}^{(1)}, y^{(1)}, \ldots, \mathbf{x}^{(n)}, y^{(n)}\right)$$

$$\hat{y} = f(\mathbf{x}; \mathbf{w}) = w_0 + w_1 x_1 + \cdots + w_d x_d$$

$$J(\mathbf{w}) = \sum_{i \in \mathcal{N}_k(\mathbf{x})} \left(y^{(i)} - \mathbf{w}^T \mathbf{x}^{(i)}\right)^2$$

The weight vector $\mathbf{w}$ is computed separately for each test sample.

Locally Weighted Linear Regression: Example

$\sigma = \frac{1}{32}$ of x-axis width $\sigma = \frac{1}{16}$ of x-axis width $\sigma = \frac{1}{8}$ of x-axis width

- Produces smoother and more accurate results than weighted kNN regression.

Vizier

Vizier: Old but fast locally weighted regression for Windows

Authors: Artur Dubrawski, Mike Duggan, Mary Soon Lee, Andrew Moore, Jeff Schneider, David Stott

Vizier was written in the mid-90's and was a combination of fast algorithms for the following operations, with a Graphical User Interface driver for Windows:

- K Nearest neighbor classification
- K Nearest neighbor regression
- Kernel regression (aka Shepherd's Interpolation)
- Kernel classification
- Locally weighted linear regression
- Locally weighted polynomial regression
- Locally weighted Bayesian polynomial regression
- Automatic determination of best model and distance metric

These days we have faster algorithms, and they run under Windows and Unix, but the functionality is not quite the same as the wild world of Vizier, so we are puttin
might find it interesting or useful. Useful Links:

- Download Vizier
- A tutorial on using Vizier.
- A paper on some of the fast algorithms in Vizier
- Tutorial slides on locally weighted learning algorithms
- More recent, faster, Auton Lab local regression code: Bug Andrew About This! It's ready for applicification but just needs a final piece of work by Andrew!
- If you have questions or comments, please contact Andrew Moore

www.cs.cmu.edu/~awm/vizier/

1 Overview

2 k-Nearest Neighbor

3 Distance Measures

4 kNN Regression

5 References

Contributions

- **These slides are authored by:**
  - Danial Gharib
  - Mahan Bayhaghi
  - Erfan Jafari
  - Mahdi Aghaei