Local Classification

Virgil Pavlu December 4, 2014

1 Intro to similarity functions

Similarities and distances are often the critical aspect of machine learning : knowing these, one can make the prediction/classification problem easy. Trivial examples: kNN, Clustering, Collaborative Filtering

SVM-dual formulation: in the SVM dual form problem, all datapoints x appear only as part of a dot product $< x1 * x2 >$, never alone. Also the prediction function on a test point z, $F(z)$, can be computed as a formula of dot products between z and the support vectors in the training set $< x * z >$.

Non-separable data: SVMs and other kernel machines have two ways to deal with non-separable data:

- by model/representation which is by design, in advance : deciding on a kernel that is likely to make data [more] separable
- by dealing with the actual data, after the model/kernel is decided : use slack variables to allow for "points inside the margins" that cost a regularization penalty.

1.1 data similarities and dot product

- measurement of data similarities : a fundamental problem in ML
- reflects a priori knowledge of the problem/data
- dot product : a natural measure for similarity
  $\langle \mathbf{x} \cdot \mathbf{y} \rangle = \sum_i x_i \cdot y_i$
- dot product amounts to being able to carry all geometric constructions formulated in terms of angles, lengths and distances

$\cos(\mathbf{x}, \mathbf{y}) = \frac{\langle \mathbf{x} \cdot \mathbf{y} \rangle}{\|\mathbf{x}\| \|\mathbf{y}\|}$ $\qquad$ $\|\mathbf{x}\| = \sqrt{\langle \mathbf{x} \cdot \mathbf{x} \rangle}$

1.2 Data similarity and kernels

- general measure for similarity
  $k : X \times X \to \mathbf{R}$, symetric $k(\mathbf{x}, \mathbf{y}) = k(\mathbf{y}, \mathbf{x})$
- symmetry is too general, we want something that feels like dot product
  $\exists \Phi : X \to \mathbf{H}$ mapping function
  $k(\mathbf{x}, \mathbf{y}) = \Phi(\mathbf{x}) \cdot \Phi(\mathbf{y})$
  where $\mathbf{H}$=feature space (Hilbert space, supports dot product)

H= feature space, Φ= map(feature) function

- for which k there exits Φ ?
- given k, if Φ exists, it may be not unique

2 k Nearest-Neighbors

`from Wikipedia` In pattern recognition, the k-Nearest Neighbors algorithm (or k-NN for short) is a non-parametric method used for classification and regression.[1] In both cases, the input consists of the k closest training examples in the feature space. The output depends on whether k-NN is used for classification or regression:

- In k-NN classification, the output is a class membership. An object is classified by a majority vote of its neighbors, with the object being assigned to the class most common among its k nearest neighbors (k is a positive integer, typically small). If k = 1, then the object is simply assigned to the class of that single nearest neighbor.
- In k-NN regression, the output is the property value for the object. This value is the average of the values of its k nearest neighbors. k-NN is a type of instance-based learning, or lazy learning, where the function is only approximated locally and all computation is deferred until classification. The k-NN algorithm is among the simplest of all machine learning algorithms.

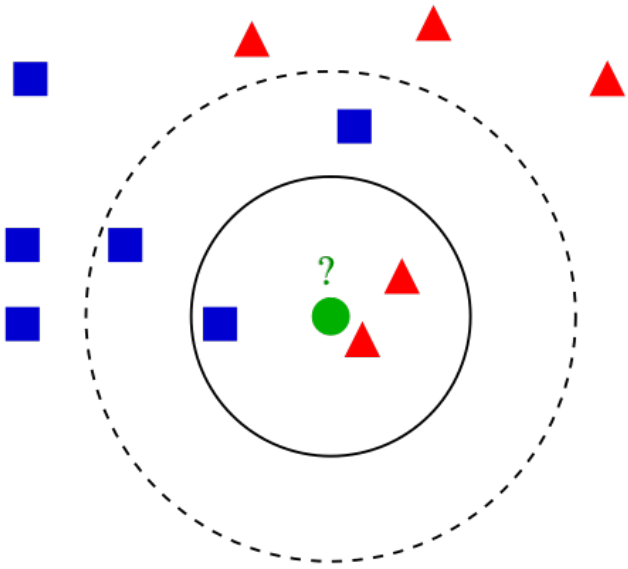

Figure 1: kNN classification: for k=1, prediction is given by majority(1 point)=red;for k=3 the majority is still red; for k=5 the majority is blue.

The neighbors are taken from a set of objects for which the class (for k-NN classification) or the object property value (for k-NN regression) is known. This can be thought of as the training set for the algorithm, though no explicit training step is required. A shortcoming of the k-NN algorithm is that it is sensitive to the local structure of the data.

kNN is easy: predict the label of point z by averaging/majority labels from closest k training points to z.

2.1 can use a kernel as the distance function

However if the kernel in monotonic in the distance/dot-product d, then the closest k neighbors are the same by $k(d)$ as they are by d.

3 kNN visualization of decision boundary

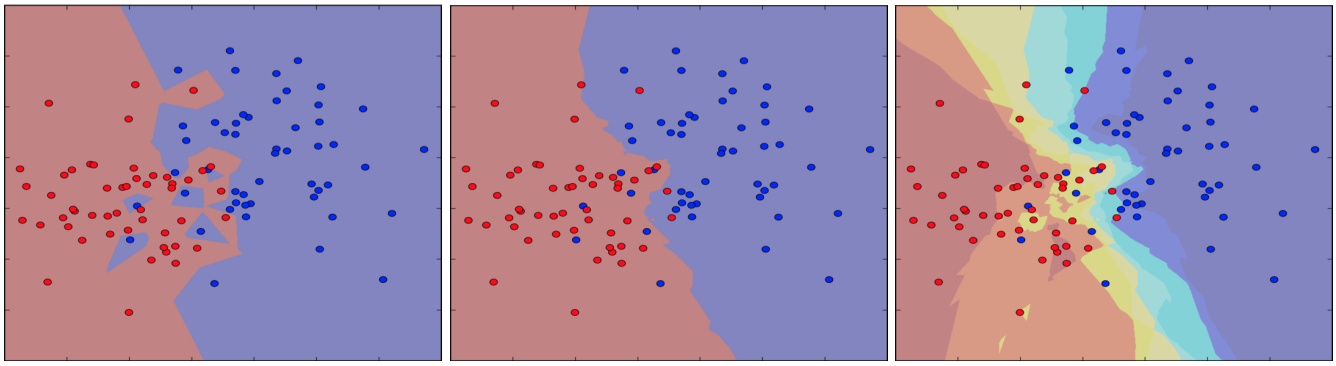

Figure 2: 1-NN classification (left), 7-NN classification (middle) and 7-NN regression (right)

4 kNN in window/range - not a fix k

Instead of fixing k, will fix a window around test point z (typically a radius for the distance to z). Then we look for majority/average in this window, which might vary in number of neighbors for different points z. Any kernel/similarity or distance can be used to define the neighborhoods.

- set window/range radius R
- for each point z construct the neighborhood $NN_z = \{\text{training } x | dist(x, z) \leq R\}$
- predict as with kNN, the majority or average label in the neighborhood.

5 kNN for feature selection

We can use kNN idea (the closest points) to do feature selection: for each training point x, we consider how much feature j helps discriminate the classes among points close to x:

- for close points from the same class z_{same}, feature j is useful if the different in feature values $|x^j - z^j_{same}|$ is small
- for close points from the opposite class z_{opp}, feature j is useful if the different in feature values $|x^j - z^j_{same}|$ is large

Here is a heuristic that measures feature j quality Q^j with respect to a point x. The close neighbors are the points z in the neighborhood of x $NN(x) = \{z \text{ close to } x\}$ — either the closest k points or the points inside a radius $dist(x, z) \leq R$.

We then separate the points in the neighborhood into points from the class (label) as x, z_{same}, and points with the opposite label, z_{opp}. The quality is updated iteratively over training points x and close neighbors z:

$Q^j_{new} = Q^j_{old} - (x^j - z^j_{same})^2 + (x^j - z^j_{opp})^2$

This is called the RELIEF algorithm for feature selection.

6 Kernel density estimation / Parzen Windows

Both for classification and regression, it can be useful to weight the contributions of the neighbors, so that the nearer neighbors contribute more to the average than the more distant ones.

Instead of fixing k or fixing a window around test point z, we will count all the training points, but weight them appropriately by the distance/similarity function (or kernel). We still compute majority or average, which is now weighting the entire training set.

Assuming the function k is normalized to 1, we can interpret the average below as a probability that x is "generated" from class C using Bayesian inference:

$$
\begin{aligned}
P(x|C) &= \frac{1}{m_C} \sum_{label(z)=C} k(x,z) \\
P(C) &= \frac{m_c}{m} \\
P(x) &= \sum_C P(x|C) * P(C) = \sum_C \frac{1}{m} \sum_{label(z)=C} k(x,z) = \frac{1}{m} \sum_z k(x,z) \\
\text{prediction on class C: } P(C|x) &= P(x|C) * P(C)/P(x) \\
&= \frac{m_c}{m} \frac{1}{P(x)} \frac{1}{m_c} \sum_{label(z)=C} k(x,z) \\
&= \frac{1}{mP(x)} \sum_{label(z)=C} k(x,z) \\
&= \frac{\sum\limits_{label(z)=C} k(x,z)}{\sum_T \sum\limits_{label(z)=T} k(x,z)} = \frac{\sum\limits_{label(z)=C} k(x,z)}{\sum\limits_z k(x,z)}
\end{aligned}
$$

Put it in different words, a soft version on kNN is to count each neighbor not as 1, but as $k(x,z)$. The result is normalized among label classes.

7 Local classification based on heat equation

`follows John Lafferty "Semi-Supervised Learning Using Gaussian Fields and Harmonic Functions"`

In this section we demonstrate local classification with kernel (similarity), this time minimizing a specific locality condition: the analogous of heat equation energy from physics.

The idea is that heat propagates in a building following this analogy between the learning problem and heat propagation:

- various locations in the building are the datapoints.
- each point is represented only by its similarity/distance with the neighbors by the kernel function $k(x,z)$

- the temperature f at each point is target/label in machine learning sense
- there are cold and hot sources that have a fixed temperature. These points are the training set.
- there are unlabeled points (testing set) where we want to infer the temperature values

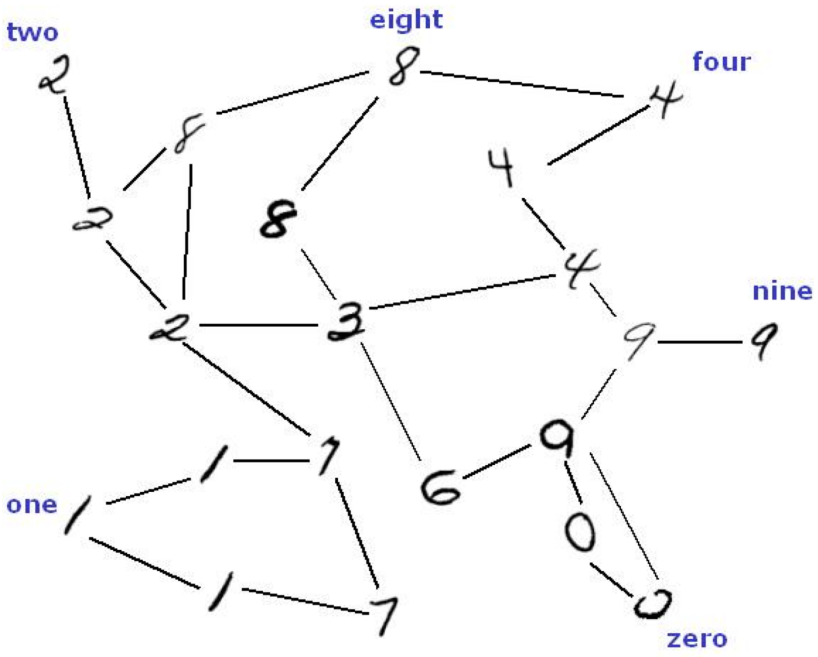

Figure 3: A graph view of the kernel/similarity. The nodes are the datapoints (exemplified by digits), the edges are similarity/kernel values k_{ij} : missing edges are $k_{ij} = 0$. Labeled (training) points have the label in blue text.

Informally we say $k_{it} = k(x_i, x_t)$ is the kernel matrix. For example k might be an RBF/Gaussian-like function

$k_{it} = \exp(-\sum_{j=1}^{d} \frac{(x_i^j - x_t^j)^2}{\sigma_j^2})$.

where j iterates through all dimensions/features and σ_j^2 is the variance on feature j.

We want the prediction/label function f (analogous to temperature) on unlabeled points; for simplicity we might right $f(x_i) = f(i) = f_i$ The objective is the equilibrium condition realized by the heat once it is distributed and conditions (sources) remain unchanged; the heat energy equation below ensures *temperature smoothness*:

$E(f) = \frac{1}{2} \sum_{i,t} k_{it}(f(x_i) - f(x_t))^2$

minimizing this energy equations implies that "close" points (k_{it} large) must have similar "temperatures", while diferent "temperatures" are allowed for distant points (small k_{it}).

Theorem. The function f that realizes the minimum energy $E(f)$ satisfies:

- f is harmonic on unlabeled points, i.e. f_i is the weighted average of f at neighbors of x_i $f(x_i) = \frac{1}{d_i} \sum_{t:k_{it}>0} k_{it} f_t$ where d_i is the number of neighbors for x_i with nonzero similarity (the number of edges incident on node x_i).

- $f = Pf$ where $P = D^{-1}K, D = \text{diagonal}(d_i)$ and $K = K_{ij}$ is the kernel matrix. Equivalently we can write $(D - K)f = 0$.
- f is unique

We can compute f by separating the train and test points into u = test points (unlabeled) and l = train points (labeled) and reorganizing the kernel K and array f by these groups:

$$K = \begin{bmatrix} K_{ll} & K_{lu} \\ K_{ul} & K_{uu} \end{bmatrix} \qquad f = \begin{bmatrix} f_l \\ f_u \end{bmatrix}$$

then

$$f_u = (D_{uu} - Kuu)^{-1} K_{ul} f_l = (I - P_{uu})^{-1} P_{ul} f_l$$

It turns out, like with regression, that the computation while exact, it is often impractical due to the matrix inversion. One can use instead a numerical optimization to minimize $E(f)$, see "Semi-Supervised Learning Using Gaussian Fields and Harmonic Functions" by John Lafferty, Xiaojin Zhu, and Zoubin Ghahramani.