Fundamentals of Machine Learning:

Nonparametric Methods: Nearest Neighbors and Kernel Density Estimation

Prof. Andrew D. Bagdanov (andrew.bagdanov AT unifi.it)

UNIVERSITÀ
DEGLI STUDI
FIRENZE

DINFO
DIPARTIMENTO DI
INGEGNERIA
DELL'INFORMAZIONE

Outline

Introduction

Motivations

- We have seen a variety of methods for classification and regression.
- So far, all of them are what is known as parametric methods.
- These methods concentrate on learning by estimating the optimal model parameters (in some way).
- The resulting model is typically simple (at least so far):

$$f(\mathbf{x}; \mathbf{w}, b) = \mathbf{w}^T\mathbf{x} + b$$

- A characteristic of these techniques is that the "knowledge" learned from the data is completely encapsulated in the parameters.
- This is sometimes called a bottleneck and can help generalization.

Lecture objectives

After this lecture you will:

- Understand how to evaluate classifier performance.
- Understand the basics of kernel density estimators and how to use them to model probability density functions.
- Understand how kernel density estimation approach can be applied to regression problems (e.g. Nadaraya-Watson).
- Understand how the Nearest-neighbor Algorithm works in theory and in practice.

Leftovers: Classifier Evaluation

Evaluating Classifiers: Balanced Problems

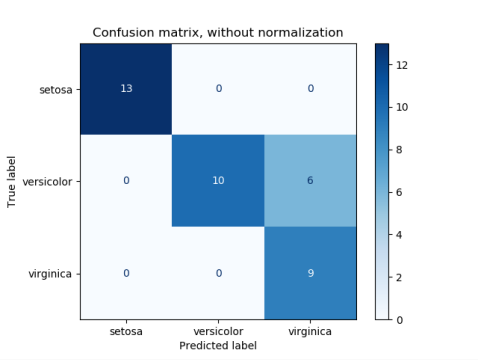
- How to we measure the performance of trained classifiers?
- If you have a balanced classification problem (like most we will see in the labs), use the classifier accuracy:

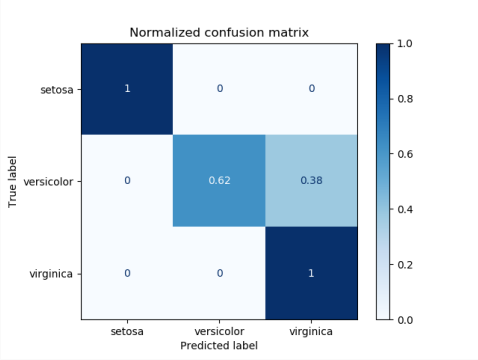
$$\text{accuracy} = \frac{\text{\# correctly classified test samples}}{\text{\# test samples}}$$

- Why might this be a bad metric if the problem is unbalanced?

Evaluating Classifiers: Confusion Matrices

- A good tool to get an overview of the types of errors a classifier is making is the confusion matrix:





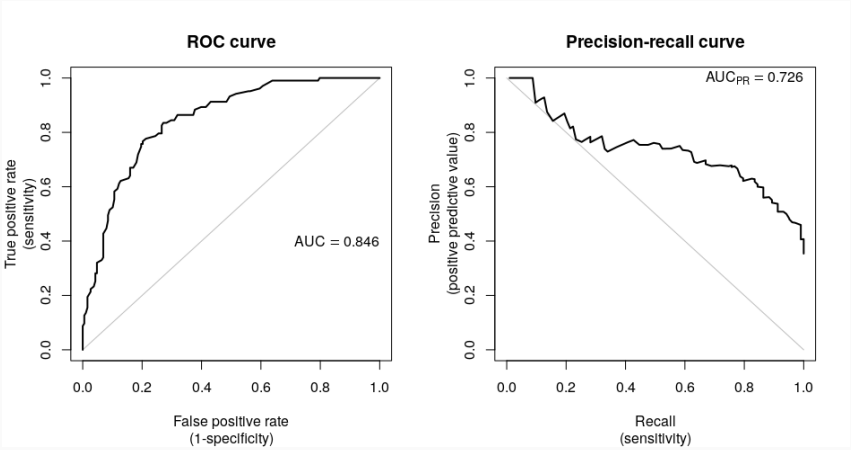
Evaluating Classifiers: Unbalanced Problems

- We begin by dissecting the classifications:
  - True positives (TP): we predicted yes, and sample does belong to class.
  - True negatives (TN): we predicted no, and the sample does not belong to class.
  - False positives (FP): we predicted yes, but the sample does not belong to class – also known as a Type I error.
  - False negatives (FN): we predicted no, but sample does belong to the class – also known as a Type II error.
- We can then define some useful metrics for unbalanced problems:

$$\begin{aligned} \text{Precision(c)} &= \frac{\text{TP}}{\text{TP} + \text{FP}} \\ \text{Recall(c)} &= \frac{\text{TP}}{\text{TP} + \text{FN}} \\ \text{F1(c)} &= 2\frac{\text{Precision(c)} * \text{Recall(c)}}{\text{Precision(c)} + \text{Recall(c)}} \end{aligned}$$

Evaluating Classifiers: Unbalanced Problems

- We usually understand precision and recall using Precision-recall (PR) and/or Receiver-Operating Characteristic (ROC) curves:



Evaluating Classifiers: Analysis

- The evaluation metric to use is usually clear given the type of problem (classification, regression, retrieval) and the distribution of training data (balanced, unbalanced).
- Getting a reliable estimate of a classifier can be tricky as it clearly depends on your train/test split.
- This becomes the central issue when performing model selection via cross-validation..
- In sklearn: `sklearn.metrics`

Density estimation: Histograms Revisited

The problem with parametric models

- The probabilistic models we have studied thus far are based on parametric models.
- We estimate a small number of distribution parameters (e.g. mean and covariance of Gaussians).
- Such a prior assumption of a distributional form is a limitation of these approaches.
- Especially when these assumptions turn out to be inaccurate.

Idea: can we dispense with the parametric assumption, and just let the data speak for itself?

Histograms: piecewise constant density estimators

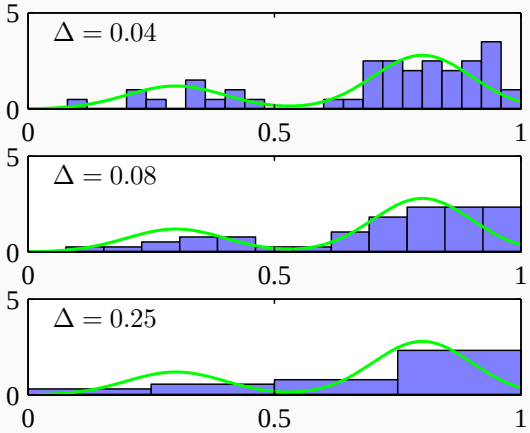
- A histogram partitions the domain into fixed-width bins.
- We then count the number of observations that fall into each discrete bin.
- We can turn this representation into a probability density by dividing by the total number of observations N and the bin width Δ_i:

$$p_i = \frac{n_i}{N\Delta_i}$$

- This yields a density $p(x)$ that is constant over the width of each bin (usually $\Delta_i = \Delta$).

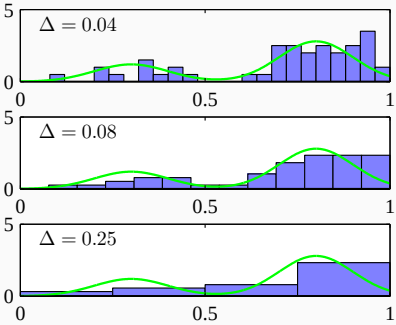
Histograms: piecewise constant density estimators

- This gives a density estimator with a single hyperparameter:



Histograms: piecewise constant density estimators

- Selection of Δ is critical – too small and the estimator has high variance.
- Too large and the curve is too smooth.
- The model is discontinuous at bin boundaries.
- The data are not needed at test time, but histograms scale very poorly with dimension (M^D).



Histograms: piecewise constant density estimators

- Despite their limitations, histograms tell us something important about density estimation.
- To estimate the density at some point x in the domain, we should look at data points that lie near x.
- This use of locality assumes we have some meaningful distance metric in the domain (e.g. Euclidean).
- Let's look at some common nonparametric methods used in Machine Learning.

Density Estimation: Kernel Density Estimators

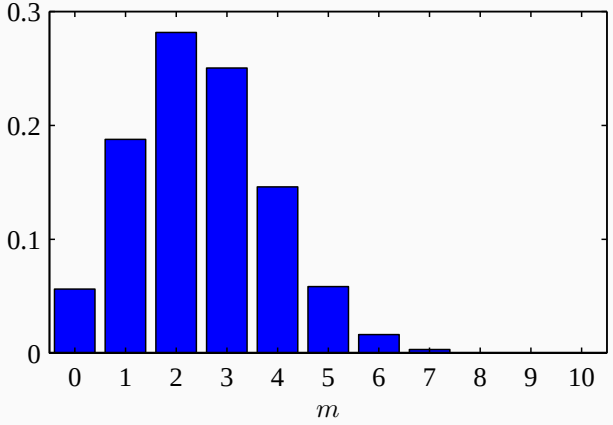
A binomial aside

- To motivate how we can extend our histogram idea to a purely local estimator, we need to consider the binomial distribution.
- The binomial distribution represents the probability of seeing m successes in a sequence of N Bernoulli trials:

$$\begin{aligned} \mathrm{Bin}(m|N,\mu) &= \binom{N}{m}\mu^m(1-\mu)^{(N-m)} \\ \mathbb{E}(m) &= \sum_{m=0}^{N} m\mathrm{Bin}(m|N,\mu) \\ &= N\mu \end{aligned}$$

A binomial aside

- The binomial is unimodal and somewhat Gaussian-like:



Generalizing

- Assume we have observations drawn from some unknown density $p(\mathbf{x})$ in a D-dimensional Euclidean space.
- Similar to a histogram, we consider a small region $\mathcal{R}$ around $\mathbf{x}$.
- The probability mass associated with $\mathcal{R}$ is:

$$P = \int_{\mathcal{R}} p(\mathbf{x}) d\mathbf{x}$$

.

- If we have N observations drawn from $p(\mathbf{x})$, each one has probability P of falling in $\mathcal{R}$.
- So, the total number of points K falling in $\mathcal{R}$ will be binomially distributed:

$$\text{Bin}(K|N, P) = \binom{N}{K} P^K (1-P)^{(N-K)}$$

Generalizing

- Using the formulas for mean and variance of the Binomial we can analyze the fraction of points falling in $\mathcal{R}$:

$$\begin{aligned} \mathbb{E}(K/N) &= P \\ \text{var}(K/N) &= \frac{P(1-P)}{N} \end{aligned}$$

- For large N this distribution will be concentrated around the P and so:

$$K \approx NP$$

Generalizing

- For large N this distribution will be concentrated around the P and so:

$$K \approx NP$$

- On the other hand, we can assume $\mathcal{R}$ with volume V is sufficiently small (and hence $p(\mathbf{x})$ is constant) and so:

$$P \approx p(\mathbf{x})V$$

- Leading us to:

$$p(\mathbf{x}) \approx \frac{K}{NV}$$

Generalizing

- Note that this estimate of $p(\mathbf{x})$ is based on two contradictory assumptions:
  - That $\mathcal{R}$ is sufficiently small and thus $p(\mathbf{x})$ is constant in it.
  - That $\mathcal{R}$ is sufficiently large so that the number of points falling into it are enough to peak the binomial.
- This leaves us with two choices:
  1. Fix K and let the data determine what V should be (Nearest Neighbors).
  2. Fix V and let the data determine what K should be (Kernel Density Estimation).

Kernel Density Estimation

- Let's take $\mathcal{R}$ to be a unit hypercube centered at the origin.
- This is conveniently described by the kernel function:

$$k(\mathbf{u}) = \begin{cases} 1 & \text{if } |u_i| \leq 1/2, \text{ for } i = 1, \ldots, D \\ 0 & \text{otherwise} \end{cases}$$

- So, for any point $\mathbf{x}$, $k((\mathbf{x} - \mathbf{x}_n)/h) = 1$ if the point $\mathbf{x}_n$ lies within a hypercube with size h centered on $\mathbf{x}$.
- The total number of points K lying in such a hypercube around $\mathbf{x}$ is then:

$$K = \sum_{n=1}^{N} k\left(\frac{\mathbf{x} - \mathbf{x}_n}{h}\right)$$

Kernel Density Estimation

- Going back to our equation for $p(\mathbf{x})$, we have:

$$\begin{aligned} p(\mathbf{x}) &= \frac{K}{NV} \\ &= \frac{1}{N}\sum_{n=1}^{N}\frac{1}{h^D}k\left(\frac{\mathbf{x}-\mathbf{x}_n}{h}\right) \end{aligned}$$

- This still suffers from discontinuities at the hypercube boundaries, so a more common kernel is the Gaussian:

$$p(\mathbf{x}) = \frac{1}{N}\sum_{n=1}^{N}\frac{1}{(2\pi h^2)^{D/2}}\exp\left\{\frac{||\mathbf{x}-\mathbf{x}_n)||^2}{2h^2}\right\}$$

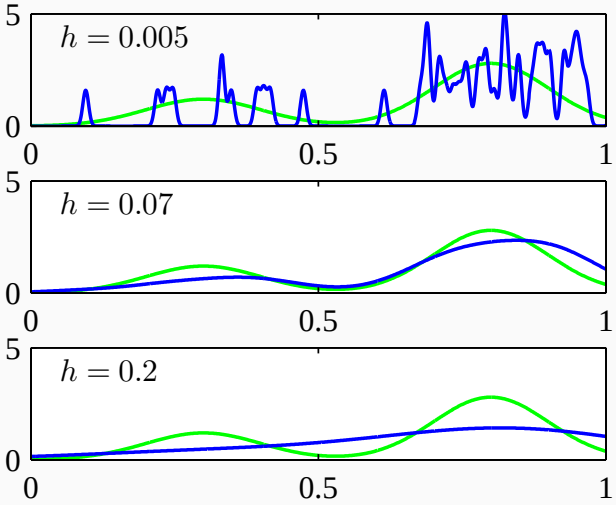
Kernel Density Estimation

- This is called a Kernel (or *Parzen*) Density Estimator.
- The hyperparameter h is called the kernel bandwidth that controls the amount of smoothing that is performed.
- Kernels functions should satisfy:

$$k(\mathbf{u}) \geq 0, \int \mathbf{u}k(\mathbf{u})d\mathbf{u} = \mathbf{0}, \text{ and } \int k(\mathbf{u})d\mathbf{u} = 1$$

- These type of approaches are often called local methods because to estimate $p(\mathbf{x})$ they look at data close to $\mathbf{x}$.

Kernel Density Estimation



Nadaraya-Watson Kernel Regression

Locality in x *and* y

- We can apply this type of local method to regression problems.
- Recall that an optimal predictor for regression is:

$$\begin{aligned}\mathbb{E}(t \mid \mathbf{x}) &= \int t p(t \mid \mathbf{x}) dt \\ &= \int t \frac{p(\mathbf{x}, t)}{p(\mathbf{x})} dt\end{aligned}$$

- Now we use Kernel Density Estimates as approximations:

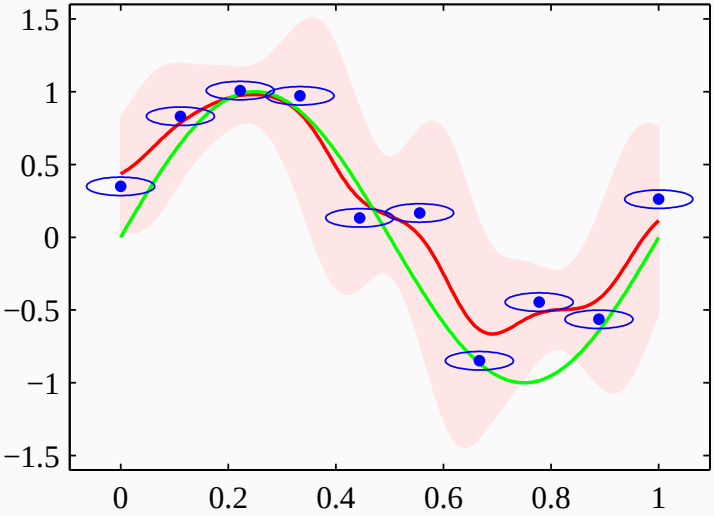
$$\hat{p}(\mathbf{x}, t) = \frac{1}{N} \sum_{n=1}^{N} k_h(\mathbf{x} - \mathbf{x}_n) k_h(t - t_n) \qquad \hat{p}(\mathbf{x}) = \frac{1}{N} \sum_{n=1}^{N} k_h(\mathbf{x} - \mathbf{x}_n)$$

Nadaraya-Watson: locally weighted regression

- Our approximation of the optimal estimator is now:

$$
\begin{aligned}
\mathbb{E}(t \mid \mathbf{x}) &= \int t \frac{p(\mathbf{x}, t)}{p(\mathbf{x})} dt \\
&= \int t \frac{\frac{1}{N} \sum_{n=1}^{N} k_h(\mathbf{x} - \mathbf{x}_n) k_h(t - t_n)}{\frac{1}{N} \sum_{n=1}^{N} k_h(\mathbf{x} - \mathbf{x}_n)} dt \\
&= \frac{\sum_{n=1}^{N} k_h(\mathbf{x} - \mathbf{x}_n) \int t k_h(t - t_n) dt}{\sum_{n=1}^{N} k_h(\mathbf{x} - \mathbf{x}_n)} \\
&= \frac{\sum_{n=1}^{N} k_h(\mathbf{x} - \mathbf{x}_n) t_n}{\sum_{n=1}^{N} k_h(\mathbf{x} - \mathbf{x}_n)}
\end{aligned}
$$

An approximately optimal estimator



K-Nearest Neighbors

The problem of h

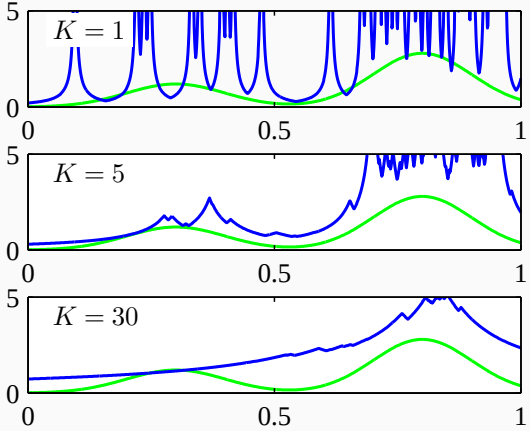
- A problem with Kernel Density Estimation is that the bandwidth h is fixed for all kernels.
- If we have parts of the space with a lot of samples, this can lead to oversmoothing and loss of detail if h is too large.
- If we reduce h, in low-density regions we will have noisy estimates.
- What if, instead of fixing V (via the bandwidth) and deriving an expression for K, we fix K and go in the other direction.

K-Nearest Neighbors

- The K-Nearest Neighbors approach is algorithmic:
  - We fix K and consider a hypersphere around a point $\mathbf{x}$ where we want to estimate $p(\mathbf{x})$.
  - We then increase the radius of this hypersphere until exactly K points from the training set are inside it.
  - We then compute $p(\mathbf{x}) = \frac{K}{NV}$ for V equal to the volume of the resulting hypersphere.

K-Nearest Neighbors

- This approach is discontinuous and noisy for density estimation:



K-Nearest Neighbors Classification

- The K-Nearest Neighbors (KNN) approach is a very convenient method for classification, however.
- Intuitively, we make a KNN density estimate for each class conditional density, and then apply Bayes' Rule.
- Suppose we have N_k examples from each class k (so $\sum_k N_k = N$).
- To classify a new point $\mathbf{x}$ we do the hypersphere trick assuming the volume needed around $\mathbf{x}$ to include K points is $V_{\mathbf{x}}$.
- If in this hypersphere we have K_k points from class k:

$$p(\mathbf{x} \mid \mathcal{C}_k) = \frac{K_k}{N_k V_{\mathbf{x}}}$$

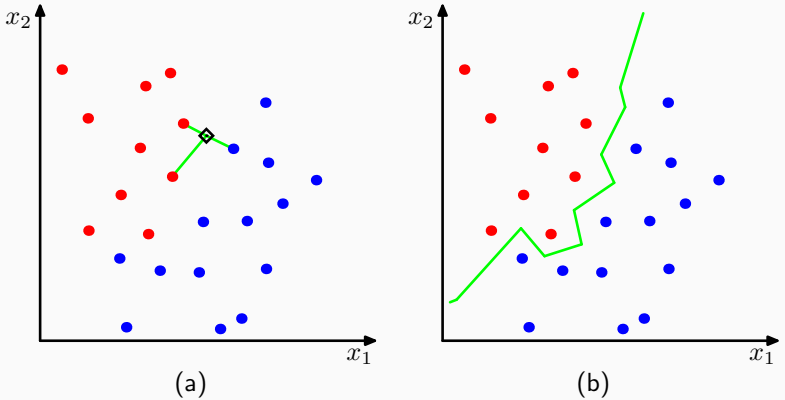
- The marginal density and class priors are similarly estimated:

$$p(\mathbf{x}) = \frac{K}{N V_{\mathbf{x}}} \quad p(\mathcal{C}_k) = \frac{N_k}{N}$$

K-Nearest Neighbors Classification

- Enter Bayes:

$$p(\mathcal{C}_k \mid \mathbf{x}) = \frac{p(\mathbf{x} \mid \mathcal{C}_k)p(\mathcal{C}_k)}{p(\mathbf{x})} = \frac{K_k}{K}$$



(a) (b)

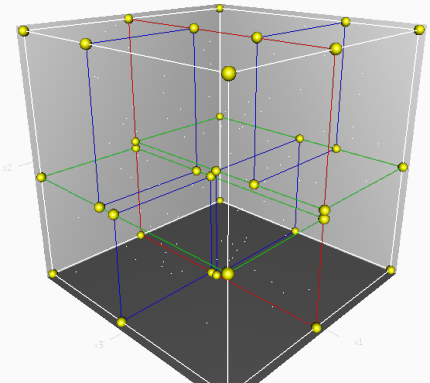
Concluding Remarks

Local methods

- In this lecture we have seen several nonparametric and local methods for:
  1. Estimating densities (Kernel Regression).
  2. Regression (Nadaraya-Watson)
  3. Classification (K-Nearest Neighbors).
- These techniques are simple and intuitive and explainable.
- They have a small (but important!) number of hyperparameters: the kernel bandwidth h (often more than one), or the number of neighbors K.
- These methods have nice convergence properties (e.g. K-Nearest Neighbors is near-optimal for $K = 1$ as $N \to \infty$).
- These methods however have a *huge* practical drawback?
  Has anyone seen it yet?

Space inefficiency

- To make predictions, all these methods *require that all training data be available at inference time*.
- In practice, we typically use a data structure (e.g. a KD-tree or Ball-Tree) to efficiently and approximately retrieve the K points in $\mathcal{D}$ closest to a point $\mathbf{x}$.



Reading and Homework Assignments

Reading Assignment:

- Bishop: Chapter 2 (2.5), Chapter 6 (6.3)
- Scikit-learn User Guide (sections 2.8, 1.3, 1.6)

Homework:

1. Try a K-Nearest Neighbor classifier on the classification problems (synthetic and real) from the lab on Classification. Visualize the decision boundary and observe the qualitative differences (with varying K between it and that of an SVM or a Bayes generative classifier.
2. Implement the Nadaraya-Watson Kernel Regressor with a Gaussian Kernel. Use it to estimate a regressor for the example in the Linear Regression laboratory. How should the bandwidth parameter h be tuned?