

Warming-up to ML, and Some Simple Supervised Learners (Distance-based “Local” Methods)

Piyush Rai

Introduction to Machine Learning (CS771A)

August 2, 2018



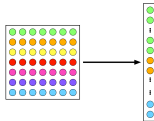
Announcements

- Please sign-up on Piazza if you haven't already
- I'll be clearing all the add-drop requests by tomorrow
- Maths refresher tutorial on Aug 4, 6:00-7:30pm in RM-101
 - Will be mostly on the basics of multivariate calculus, linear algebra, prob/stats, optimization (basically things you are expected to know for this course)



Some Notation/Nomenclature/Convention

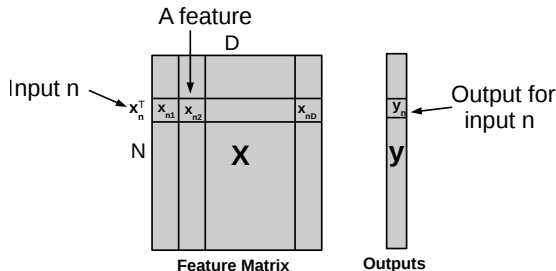
- Supervised Learning requires training data given as a set of input-output pairs $\{(\mathbf{x}_n, y_n)\}_{n=1}^N$
- Unsupervised Learning requires training data given as a set of inputs $\{\mathbf{x}_n\}_{n=1}^N$
- Each input \mathbf{x}_n is (usually) a vector containing the values of the features or attributes or covariates that encode properties of the data it represents, e.g.,
 - Representing a 7×7 image: \mathbf{x}_n can be a 49×1 vector of pixel intensities



- Note: Good features can also be learned from data (feature learning) or extracted using hand-crafted rules defined by a domain expert. **Having a good set of features is half the battle won!**
- Each y_n is the output or response or label associated with input \mathbf{x}_n
 - The output y_n can be a scalar, a vector of numbers, or a structured object (more on this later)

Some Notation/Nomenclature/Convention

- Will assume each input \mathbf{x}_n to be a $D \times 1$ column vector (its transpose \mathbf{x}_n^T will be row vector)
- x_{nd} will denote the d -th feature of the n -th input
- We will use \mathbf{X} ($N \times D$ feature matrix) to collectively denote all the N inputs
- We will use \mathbf{y} ($N \times 1$ output/response/label vector) to collectively denote all the N outputs



- Note: If each y_n itself is a vector (we will see such cases later) then we will use a matrix \mathbf{Y} to collectively denote all the N outputs (with row n containing y_n) and also use boldfaced \mathbf{y}_n

Getting Features from Raw Data: A Simple Example

Consider the feature representation for some text data consisting of the following sentences:

- John likes to watch movies
- Mary likes movies too
- John also likes football

Our feature “vocabulary” consists of 8 unique words

Here is the **bag-of-words** feature vector representation of these 3 sentences

	John	likes	to	watch	movies	Mary	too	also	football
Sentence 1	1	1	1	1	1	0	0	0	0
Sentence 2	0	1	0	0	1	1	1	0	0
Sentence 3	1	1	0	0	0	0	0	1	1

Here the features are binary (presence/absence of each word)

Again, note that this may not necessarily be the best “feature” representation for a given task (which is why other techniques or feature learning may be needed)

Types of Features and Types of Outputs

- Features (in vector \mathbf{x}_n) as well as outputs y_n can be real-valued, binary, categorical, ordinal, etc.
- **Real-valued:** Pixel intensity, house area, house price, rainfall amount, temperature, etc
- **Binary:** Male/female, adult/non-adult, or any yes/no or present/absent type values
- **Categorical/Discrete:** Pincode, bloodgroup, or any “which one from this finite set” type values
- **Ordinal:** Grade (A/B/C etc.) in a course, or any other type where **relative values matters**
- Often, the features can be of mixed types (some real, some categorical, some ordinal, etc.)
- Appropriate handling of different types of features may be very important (even if you algorithm is designed to “learn” good features, given a set of heterogeneous features)
- In Sup. Learning, different types of outputs may require different type of learning models



Supervised Learning



Supervised Learning

- Supervised Learning comes in many flavors. The flavor depends on the type of each output y_n
- Regression:** $y_n \in \mathbb{R}$ (real-valued scalar)
- Multi-Output Regression:** $\mathbf{y}_n \in \mathbb{R}^M$ (real-valued vector containing M outputs)

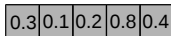


Illustration of a 5-dim output vector
for a multi-output regression problem

- Binary Classification:** $y_n \in \{-1, +1\}$ or $\{0, 1\}$ (output in classification is also called “label”)
- Multi-class Classification:** $y_n \in \{1, 2, \dots, M\}$ or $\{0, 1, \dots, M-1\}$ (one of M classes is correct label)

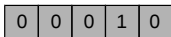


Illustration of a 5-dim **one-hot** label vector
for a multi-class classification problem

- Multi-label Classification:** $y_n \in \{-1, +1\}^M$ or $\{0, 1\}^M$ (a subset of M labels are correct)

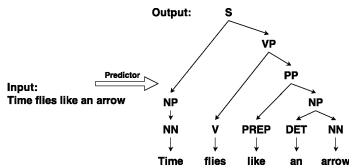


Illustration of a 5-dim binary label vector
for a multi-label classification problem
(unlike one-hot, there can be multiple 1s)

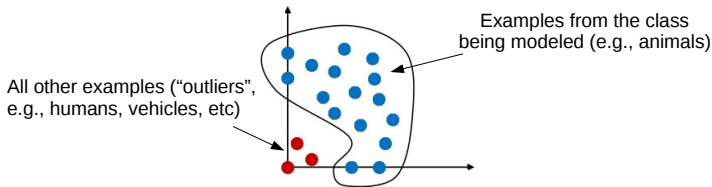
- Note: Multi-label classification is also informally called “**tagging**” (especially in Computer Vision)

Supervised Learning (Contd.)

- **Structured-Prediction** (a.k.a. Structured Output Learning): Each y_n is a structured object



- **One-Class Classification** (a.k.a. outlier/anomaly/novelty detection): y_n is “1” or “everything else”



- **Ranking**: Each y_n is a ranked list of relevant stuff for a given input/query x



Background: Computing Distances/Similarities

- Assuming all real-valued features, an input $\mathbf{x}_n \in \mathbb{R}^{D \times 1}$ is a point in a D dim. vector space of reals
- Standard rules of vector algebra apply on such representations, e.g.,
 - Euclidean distance b/w two points (say two images or two documents) $\mathbf{x}_n \in \mathbb{R}^D$ and $\mathbf{x}_m \in \mathbb{R}^D$

$$d(\mathbf{x}_n, \mathbf{x}_m) = \|\mathbf{x}_n - \mathbf{x}_m\| = \sqrt{(\mathbf{x}_n - \mathbf{x}_m)^\top (\mathbf{x}_n - \mathbf{x}_m)} = \sqrt{\sum_{d=1}^D (x_{nd} - x_{md})^2}$$

- Inner-product similarity b/w \mathbf{x}_n and \mathbf{x}_m (cosine, $\mathbf{x}_n, \mathbf{x}_m$ are unit-length vectors)

$$s(\mathbf{x}_n, \mathbf{x}_m) = \langle \mathbf{x}_n, \mathbf{x}_m \rangle = \mathbf{x}_n^\top \mathbf{x}_m = \sum_{d=1}^D x_{nd} x_{md}$$

- ℓ_1 distance between two points \mathbf{x}_n and \mathbf{x}_m

$$d_1(\mathbf{x}_n, \mathbf{x}_m) = \|\mathbf{x}_n - \mathbf{x}_m\|_1 = \sum_{d=1}^D |x_{nd} - x_{md}|$$

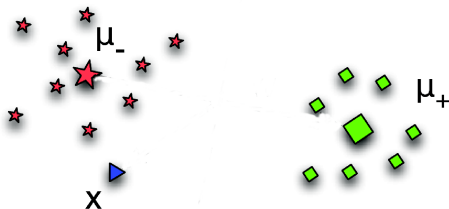


Our First (Supervised) Learning Algorithm
(need to know nothing except how to
compute distances/similarities between points!)



Prototype based Classification

- Given: N labeled training examples $\{\mathbf{x}_n, y_n\}_{n=1}^N$ from two classes
 - Assume green is positive and red is negative class
 - N_+ examples from positive class, N_- examples from negative class
- Our goal: **Learn a model** to predict label (class) y for a new **test example** \mathbf{x}



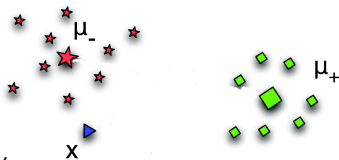
- A simple “**distance from means**” model: **predict the class that has a closer mean**
- Note: The basic idea easily generalizes to more than 2 classes as well

Pic credit: Modified from the original by Alex Smola and Vishy Vishwanathan



Prototype based Classification: More Formally

- What does the decision rule look like, mathematically ?



- The mean of each class is given by

$$\mu_- = \frac{1}{N_-} \sum_{y_n=-1} \mathbf{x}_n \quad \text{and} \quad \mu_+ = \frac{1}{N_+} \sum_{y_n=+1} \mathbf{x}_n$$

- Euclidean Distances** from each mean are given by

$$\|\mu_- - \mathbf{x}\|^2 = \|\mu_-\|^2 + \|\mathbf{x}\|^2 - 2\langle \mu_-, \mathbf{x} \rangle$$

$$\|\mu_+ - \mathbf{x}\|^2 = \|\mu_+\|^2 + \|\mathbf{x}\|^2 - 2\langle \mu_+, \mathbf{x} \rangle$$

- Decision Rule:** If $f(\mathbf{x}) := \|\mu_- - \mathbf{x}\|^2 - \|\mu_+ - \mathbf{x}\|^2 > 0$ then predict +1, otherwise predict -1

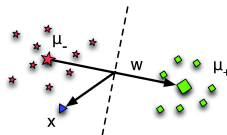


Prototype based Classification: The Decision Rule

- We saw that our decision rule was

$$f(\mathbf{x}) := \|\boldsymbol{\mu}_- - \mathbf{x}\|^2 - \|\boldsymbol{\mu}_+ - \mathbf{x}\|^2 = 2\langle \boldsymbol{\mu}_+ - \boldsymbol{\mu}_-, \mathbf{x} \rangle + \|\boldsymbol{\mu}_-\|^2 - \|\boldsymbol{\mu}_+\|^2$$

- **Imp.:** $f(\mathbf{x})$ effectively denotes a **hyperplane based classification rule** $f(\mathbf{x}) = \mathbf{w}^\top \mathbf{x} + b$ with the vector $\mathbf{w} = \boldsymbol{\mu}_+ - \boldsymbol{\mu}_-$ representing the direction normal to the hyperplane



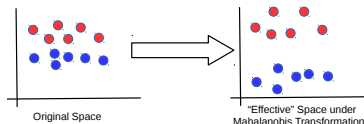
- **Imp.:** Can show that the rule is equivalent to $f(\mathbf{x}) = \sum_{n=1}^N \alpha_n \langle \mathbf{x}_n, \mathbf{x} \rangle + b$, where α 's and b can be estimated from training data (**try this as an exercise**)
 - This form of the decision rule is very important. Decision rules for many (in fact most) supervised learning algorithms can be written like this (**weighted sum of similarities with all the training inputs**)

Be Careful when Computing Distances

- Euclidean distance $d(\mathbf{x}_n, \mathbf{x}_m) = \sqrt{(\mathbf{x}_n - \mathbf{x}_m)^\top (\mathbf{x}_n - \mathbf{x}_m)}$ may not always be appropriate
- Another alternative (still Euclidean-like) can be to use the Mahalanobis distance

$$d_M(\mathbf{x}_n, \mathbf{x}_m) = \sqrt{(\mathbf{x}_n - \mathbf{x}_m)^\top \mathbf{M} (\mathbf{x}_n - \mathbf{x}_m)}$$

- Shown below is an illustration of what $\mathbf{M} = \begin{bmatrix} 1 & 0 \\ 0 & 2 \end{bmatrix}$ will do (note: figure not to scale)

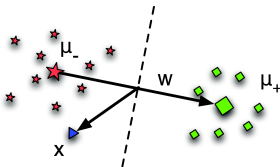


- How do I know what's the right \mathbf{M} for my data? Some options
 - Set it based on some knowledge of what your data looks like
 - [Learn it](#) from data (called [Distance Metric Learning](#)¹ - a whole research area in itself)
- Distance Metric Learning is one of the many approaches for [feature learning](#) from data

¹Distance Metric Learning. See "A Survey on Metric Learning for Feature Vectors and Structured Data" by Ballet *et al*



Prototype based Classification: Some Comments



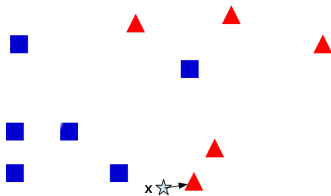
- A very simple supervised learner. Only works for binary/multi-class classifn (**can't do regression**)
- The basic approach can only learn **linear decision boundaries**, unless Euclidean distance is replaced by a nonlinear distance function. Also assumes **roughly equal “spread” of each class** (think why)
- Usually require plenty of training data for each class (to estimate the mean of each class reliably)
 - But with a good feature learner, even ONE (or very few) example per class may be enough (a state-of-the-art “Few-Shot Learning” model actually uses Prototype based classification)
- Can also be made more rigorous by modeling each class by a more sophisticated **class probability distribution** (e.g., a multivariate Gaussian) and computing distances of test points from these class distributions (rather than means)

Another Simple Supervised Learner: Nearest Neighbors



Nearest Neighbor

- Another classic distance-based supervised learning method

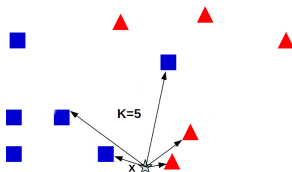


- The label y for $\mathbf{x} \in \mathbb{R}^D$ will be the label of its **nearest neighbor in training data**. Also known as **one-nearest-neighbor (1-NN)**
- Euclidean/Mahalanobis distance can be used to find the nearest neighbor (or can use a learned distance metric)
- We typically use more ($K > 1$) neighbors in practice
- Note: The method is widely applicable - works for both classification and regression problems



K-Nearest Neighbors (K-NN)

- Makes one-nearest-neighbor more robust by using more than one neighbor
- Test time simply does a majority vote (or average) of the labels of K closest training inputs



- For a test input x , the averaging version of the prediction rule for K -nearest neighbors

$$\mathbf{y} = \frac{1}{K} \sum_{n \in \mathcal{N}_K(x)} \mathbf{y}_n$$

.. where $\mathcal{N}_K(x)$ is the set of K closest training inputs for x

- Above assumes the K neighbors have equal $(1/K)$ weights. Can also use distance-based weights
- Note: The rule works for **multi-label classification** too where each $\mathbf{y}_n \in \{0, 1\}^M$ is a binary vector
 - Averaging will give a real-valued “label score vector” $\mathbf{y} \in \mathbb{R}^M$ using which we can find the best label(s)

K-NN for Multi-Label Learning: Pictorial Illustration

- Suppose $K = 3$. The label averaging for a multi-label learning problem will look like

$$\mathbf{y} = \frac{1}{3} * \begin{bmatrix} 1 & 0 & 0 & 1 & 0 \end{bmatrix} + \frac{1}{3} * \begin{bmatrix} 1 & 0 & 1 & 1 & 0 \end{bmatrix} + \frac{1}{3} * \begin{bmatrix} 1 & 0 & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0.33 & 0.66 & 0.33 \end{bmatrix}$$

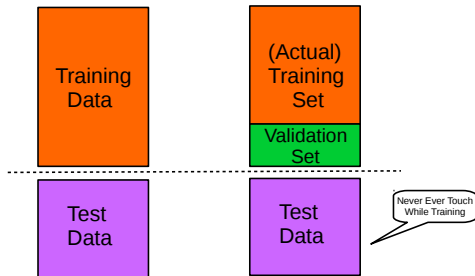
#1 label #4 label #3 label #2 label #3 label

- Note that we can use the final \mathbf{y} to rank the labels based on the real-valued scores
 - Can use it to predict the best, best-2, best-3, and so on..
 - Note: This is why multi-label learning is often used in some ranking problems where we wish to predict a ranking of the possible labels an input can have



How to Select K : Cross-Validation

- We can use cross-validation to select the “optimal” value of K
- Cross-validation - Divide the training data into two parts: actual training set and a **validation set**



- Try different values of K and look at the accuracies on the validation set
 - Note: For each K , we typically try multiple splits of train and validation sets
- Select the K that gives the best accuracy on the validation set
- **Never touch the test set (even if you have access to it) during training to choose the best K**



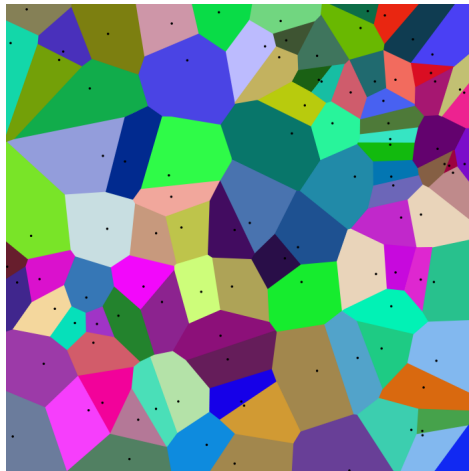
Some Aspects about Nearest Neighbor

- A simple yet very effective method in practice (if given lots of training data)
 - *Provably* has an error-rate that is no worse than twice of the “Bayes optimal” classifier which assumes knowledge of the true data distribution for each class
- Also called a **memory-based** or **instance-based** or **non-parametric** method
- No “model” is learned here. Prediction step uses all the training data
- Requires lots of storage (need to keep all the training data at test time)
- Prediction can be slow at test time
 - For each test point, need to compute its distance from all the training points
 - Clever data-structures or data-summarization techniques can provide speed-ups
- Need to be careful in **choosing the distance function** to compute distances (especially when the data dimension D is very large)
- The 1-NN **can suffer if data contains outliers** (we will soon see a geometric illustration), or **if amount of training data is small**. Using more neighbors ($K > 1$) is usually more robust



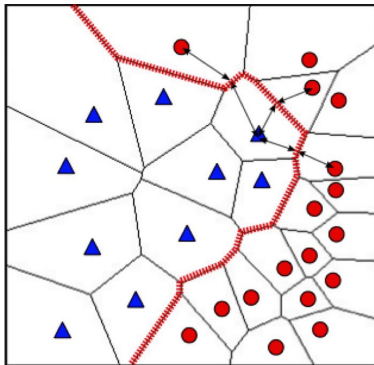
Geometry of 1-NN

- 1-NN induces a Voronoi tessellation of the input space



The Decision Boundary of 1-NN (for binary classification)

- The decision boundary is composed of hyperplanes that form perpendicular bisectors of pairs of points from different classes

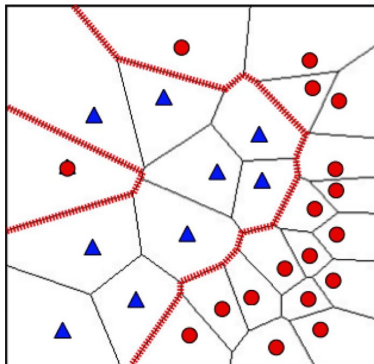


Pic credit: Victor Lavrenko



Effect of Outliers on 1-NN

- How the decision boundary can drastically change when the data contains some outliers

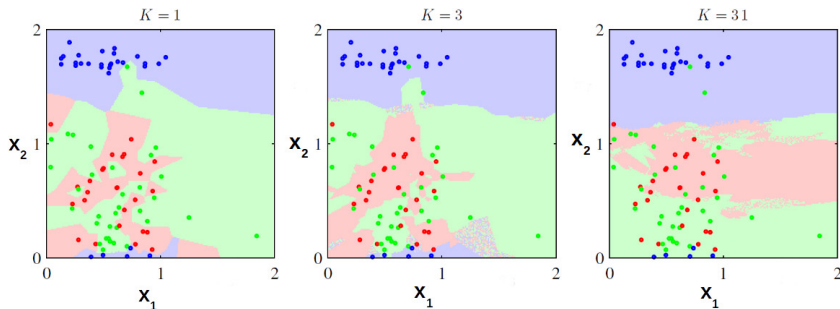


Pic credit: Victor Lavrenko



Effect of Varying K

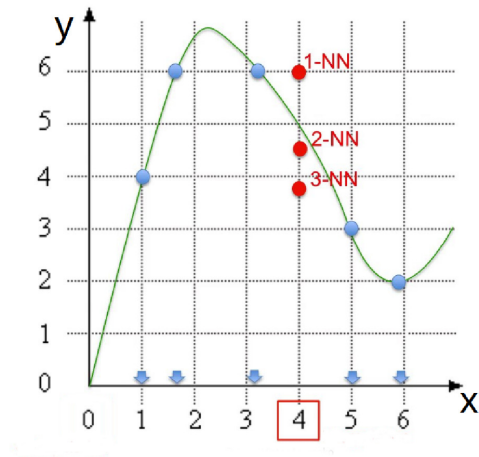
- Larger K leads to smoother decision boundaries



Too small K (e.g., $K = 1$) can lead to **overfitting**, too large K can lead to **underfitting**



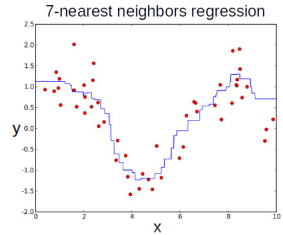
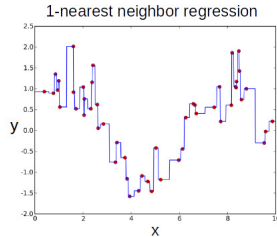
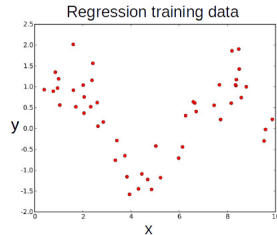
K-NN Behavior for Regression



Pic credit: Victor Lavrenko



K-NN Behavior for Regression



Pic credit: Alex Smola and Vishy Vishwanathan



Summary

- Looked at two distance-based methods for classification/regression
 - A “Distance from Means” Method
 - Nearest Neighbors Method
- Both are essentially “local” methods (look at local neighborhood of the test point)
- Both are simple to understand and only require knowledge of basic geometry
- Have connections to other more advanced methods (as we will see)
- Need to be careful when computing the distances (learned Mahalanobis distance metrics, or “learned features” + Euclidean distance can often do wonders!)

