CSCI567 Machine Learning (Fall 2017)

Prof. Fei Sha

U of Southern California

Lecture on Sept. 28, 2017

Outline

Outline

Administrative stuff on Quiz 1

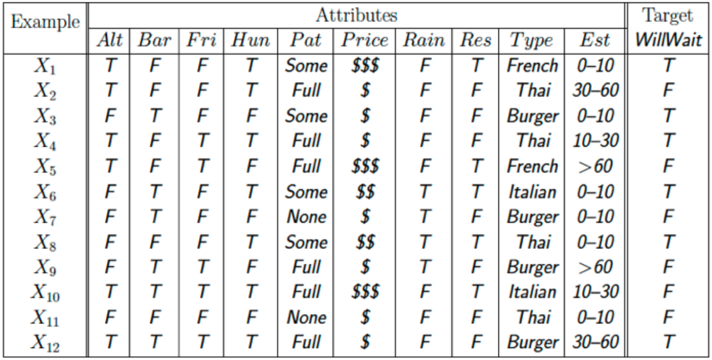
- Closed-book, no online search, no cheat sheet
- Time: 5:15pm - 6:15pm but do *arrive early*
- Location: SGM 114 (ie, here!)

Outline

A tree model for deciding where to eat

Choosing a restaurant

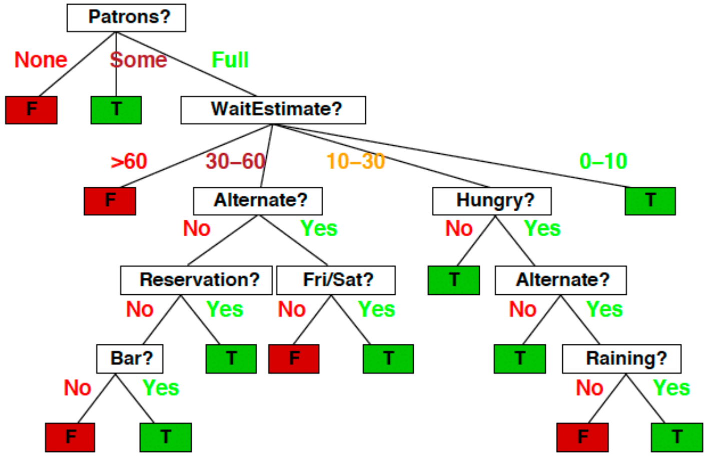
(Example from Russell & Norvig, AIMA)



Example	Attributes										Target
	Alt	*Bar*	*Fri*	*Hun*	*Pat*	*Price*	*Rain*	*Res*	*Type*	*Est*	*WillWait*
X_1	T	F	F	T	Some	\$\$\$	F	T	French	0–10	T
X_2	T	F	F	T	Full	\$	F	F	Thai	30–60	F
X_3	F	T	F	F	Some	\$	F	F	Burger	0–10	T
X_4	T	F	T	T	Full	\$	F	F	Thai	10–30	T
X_5	T	F	T	F	Full	\$\$\$	F	T	French	>60	F
X_6	F	T	F	T	Some	\$\$	T	T	Italian	0–10	T
X_7	F	T	F	F	None	\$	T	F	Burger	0–10	F
X_8	F	F	F	T	Some	\$\$	T	T	Thai	0–10	T
X_9	F	T	T	F	Full	\$	T	F	Burger	>60	F
X_{10}	T	T	T	T	Full	\$\$\$	F	T	Italian	10–30	F
X_{11}	F	F	F	F	None	\$	F	F	Thai	0–10	F
X_{12}	T	T	T	T	Full	\$	F	F	Burger	30–60	T

Classification of examples is positive (T) or negative (F)

Greedily we build the tree and get this



Outline

Boosting

High-level idea: combine a lot of classifiers

- Sequentially construct those classifiers one at a time
- Use *weak* classifiers to arrive at complex decision boundaries

Our plan

- Describe AdaBoost algorithm
- Derive the algorithm

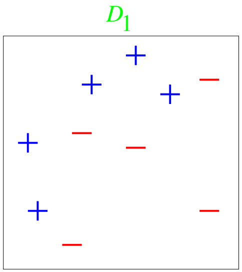
How Boosting algorithm works?

- Given: N samples $\{\boldsymbol{x}_n, y_n\}$, where $y_n \in \{+1, -1\}$, and some ways of constructing weak (or base) classifiers
- Initialize weights $w_1(n) = \frac{1}{N}$ for every training sample.
- For t= 1 to T
  1. Train a weak classifier $h_t(\boldsymbol{x})$ based on the current weight $w_t(n)$, by minimizing the *weighted* classification error
  $$\epsilon_t = \sum_n w_t(n) \mathbb{I}[y_n \neq h_t(\boldsymbol{x}_n)]$$
  2. Calculate weights for combining classifiers $\beta_t = \frac{1}{2} \log \frac{1-\epsilon_t}{\epsilon_t}$
  3. Update weights
  $$w_{t+1}(n) \propto w_t(n) e^{-\beta_t y_n h_t(\boldsymbol{x}_n)}$$
  and normalize them such that $\sum_n w_{t+1}(n) = 1$.
- Output the final classifier
$$h[\boldsymbol{x}] = \text{sign}\left[\sum_{t=1}^{T} \beta_t h_t(\boldsymbol{x})\right]$$

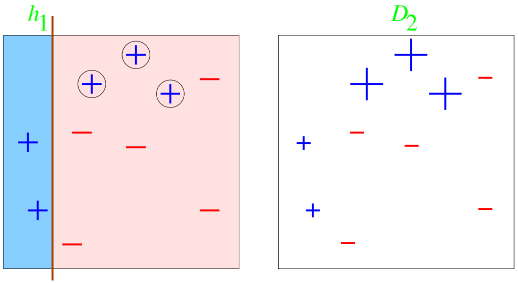
Example

10 data points

- Base classifier $h(\cdot)$: either horizontal or vertical lines (these are called *decision stumps*, classifying data based on a single attribute)
- The data points are clearly not linear separable.
- In the beginning, all data points have equal weights (the size of the data markers "$+$" or "-")

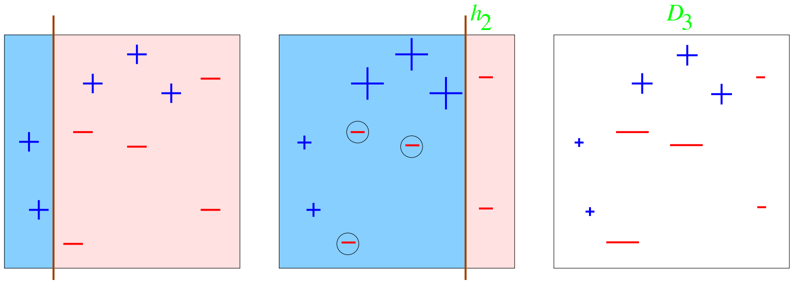


Round 1: $t = 1$



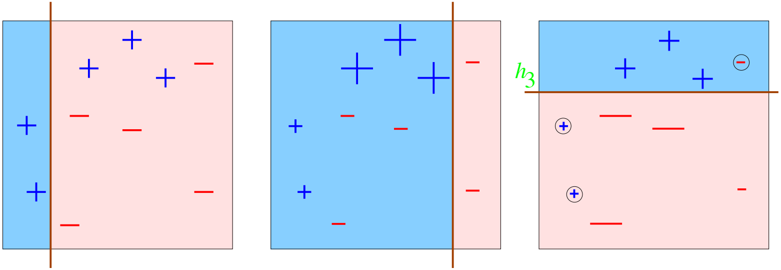
- 3 misclassified (with circles): $\epsilon_1 = 0.3 \rightarrow \beta_1 = 0.42$.
- Weights recomputed; the 3 misclassified data points receive larger weights

Round 2: $t = 2$



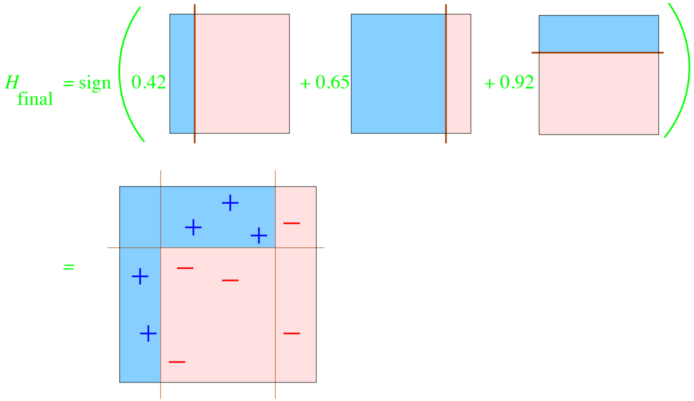
- 3 misclassified (with circles): $\epsilon_2 = 0.21 \rightarrow \beta_2 = 0.65$.
  Note that $\epsilon_2 \neq 0.3$ as those 3 data points have weights less than $1/10$
- Weights recomputed; the 3 misclassified data points receive larger weights. Note that the data points classified correctly on round $t = 1$ receive much smaller weights as they have been consistently classified correctly

Round 3: $t = 3$



- 3 misclassified (with circles): $\epsilon_3 = 0.14 \rightarrow \beta_3 = 0.92$.
- Note that those previously correctly classified data points are now misclassified — however, we might be lucky on this as if they have been consistently classified correctly, then this round's mistake is probably not a big deal.

Final classifier: combining 3 classifiers



- all data points are now classified correctly!

Why AdaBoost works?

We will show next that it minimizes a loss function related to classification error.

Classification loss

- Suppose we want to have a classifier

$$h(\boldsymbol{x}) = \text{sign}[f(\boldsymbol{x})] = \begin{cases} 1 & \text{if } f(\boldsymbol{x}) > 0 \\ -1 & \text{if } f(\boldsymbol{x}) < 0 \end{cases}$$

- our loss function is thus

$$\ell(h(\boldsymbol{x}), y) = \begin{cases} 0 & \text{if } yf(\boldsymbol{x}) > 0 \\ 1 & \text{if } yf(\boldsymbol{x}) < 0 \end{cases}$$

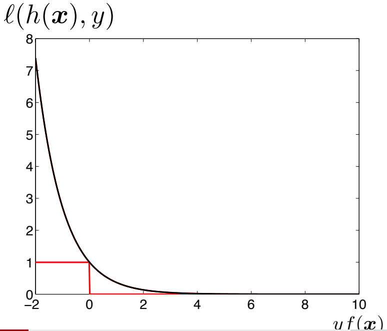
Namely, the function $f(\boldsymbol{x})$ and the target label y should have the same sign to avoid a loss of 1.

Exponential loss

The previous loss function $\ell(h(\boldsymbol{x}), y)$ is difficult to optimize. Instead, we will use the following loss function

$$\ell^{\mathrm{EXP}}(h(\boldsymbol{x}), y) = e^{-yf(\boldsymbol{x})}$$

This loss function will function as a surrogate to the true loss function $\ell(h(\boldsymbol{x}, y)$. However, $\ell^{\mathrm{EXP}}(h(\boldsymbol{x}), y)$ is easier to handle numerically as it is differentiable, see below the contrast between the red and black curves



Choosing the t-th classifier

Suppose we have built a classifier $f_{t-1}(\boldsymbol{x})$, and we want to improve it by adding a new classifier $h_t(\boldsymbol{x})$ to construct a new classifier

$$f(\boldsymbol{x}) = f_{t-1}(\boldsymbol{x}) + \beta_t h_t(\boldsymbol{x})$$

how can we choose optimally the new classifier $h_t(\boldsymbol{x})$ and the combination coefficient β_t? The strategy we will use is to greedily *minimize the exponential loss function*.

$$\begin{aligned}(h_t^*(\boldsymbol{x}), \beta_t^*) &= \arg\min_{(h_t(\boldsymbol{x}),\beta_t)} \sum_n e^{-y_n f(\boldsymbol{x}_n)} \\ &= \arg\min_{(h_t(\boldsymbol{x}),\beta_t)} \sum_n e^{-y_n[f_{t-1}(\boldsymbol{x}_n)+\beta_t h_t(\boldsymbol{x}_n)]}\end{aligned}$$

Choosing the t-th classifier

Suppose we have built a classifier $f_{t-1}(\boldsymbol{x})$, and we want to improve it by adding a new classifier $h_t(\boldsymbol{x})$ to construct a new classifier

$$f(\boldsymbol{x}) = f_{t-1}(\boldsymbol{x}) + \beta_t h_t(\boldsymbol{x})$$

how can we choose optimally the new classifier $h_t(\boldsymbol{x})$ and the combination coefficient β_t? The strategy we will use is to greedily *minimize the exponential loss function*.

$$\begin{aligned}(h_t^*(\boldsymbol{x}), \beta_t^*) &= \arg\min_{(h_t(\boldsymbol{x}),\beta_t)} \sum_n e^{-y_n f(\boldsymbol{x}_n)} \\ &= \arg\min_{(h_t(\boldsymbol{x}),\beta_t)} \sum_n e^{-y_n [f_{t-1}(\boldsymbol{x}_n)+\beta_t h_t(\boldsymbol{x}_n)]} \\ &= \arg\min_{(h_t(\boldsymbol{x}),\beta_t)} \sum_n w_t(n) e^{-y_n \beta_t h_t(\boldsymbol{x}_n)}\end{aligned}$$

where we have used $w_t(n)$ as a shorthand for $e^{-y_n f_{t-1}(\boldsymbol{x}_n)}$

The new classifier

We decompose the *weighted* loss function (by $w_t(n)$) into two parts

$$\begin{aligned}
&\sum_n w_t(n) e^{-y_n \beta_t h_t(\boldsymbol{x}_n)} \\
&= \sum_n w_t(n) e^{\beta_t} \mathbb{I}[y_n \neq h_t(\boldsymbol{x}_n)] + \sum_n w_t(n) e^{-\beta_t} \mathbb{I}[y_n = h_t(\boldsymbol{x}_n)]
\end{aligned}$$

The new classifier

We decompose the *weighted* loss function (by $w_t(n)$) into two parts

$$\begin{aligned}
&\sum_n w_t(n) e^{-y_n \beta_t h_t(\boldsymbol{x}_n)} \\
&= \sum_n w_t(n) e^{\beta_t} \mathbb{I}[y_n \neq h_t(\boldsymbol{x}_n)] + \sum_n w_t(n) e^{-\beta_t} \mathbb{I}[y_n = h_t(\boldsymbol{x}_n)] \\
&= \sum_n w_t(n) e^{\beta_t} \mathbb{I}[y_n \neq h_t(\boldsymbol{x}_n)] + \sum_n w_t(n) e^{-\beta_t} (1 - \mathbb{I}[y_n \neq h_t(\boldsymbol{x}_n)])
\end{aligned}$$

The new classifier

We decompose the *weighted* loss function (by $w_t(n)$) into two parts

$$\begin{aligned}
&\sum_n w_t(n) e^{-y_n \beta_t h_t(\boldsymbol{x}_n)} \\
&= \sum_n w_t(n) e^{\beta_t} \mathbb{I}[y_n \neq h_t(\boldsymbol{x}_n)] + \sum_n w_t(n) e^{-\beta_t} \mathbb{I}[y_n = h_t(\boldsymbol{x}_n)] \\
&= \sum_n w_t(n) e^{\beta_t} \mathbb{I}[y_n \neq h_t(\boldsymbol{x}_n)] + \sum_n w_t(n) e^{-\beta_t} (1 - \mathbb{I}[y_n \neq h_t(\boldsymbol{x}_n)]) \\
&= (e^{\beta_t} - e^{-\beta_t}) \sum_n w_t(n) \mathbb{I}[y_n \neq h_t(\boldsymbol{x}_n)] + e^{-\beta_t} \sum_n w_t(n)
\end{aligned}$$

We have used the following properties to derive the above

- $y_n h_t(\boldsymbol{x}_n)$ is either 1 or -1 as $h_t(\boldsymbol{x}_n)$ is the output of a binary classifier.
- The indicator function $\mathbb{I}[y_n = h_t(\boldsymbol{x}_n)]$ is binary, either 0 or 1. Thus, it equals to $1 - \mathbb{I}[y_n \neq h_t(\boldsymbol{x}_n)]$.

Minimizing the weighted classification error

Thus, we would want to choose $h_t(\boldsymbol{x}_n)$ such that

$$h_t^*(\boldsymbol{x}) = \arg\min_{h_t(\boldsymbol{x})} \epsilon_t = \sum_n w_t(n) \mathbb{I}[y_n \neq h_t(\boldsymbol{x}_n)]$$

Namely, the weighted classification error is minimized — precisely *train a weak classifier based on the current weight $w_t(n)$ on the slide* **How Boosting algorithm works?**.

Minimizing the weighted classification error

Thus, we would want to choose $h_t(\boldsymbol{x}_n)$ such that

$$h_t^*(\boldsymbol{x}) = \arg\min_{h_t(\boldsymbol{x})} \epsilon_t = \sum_n w_t(n)\mathbb{I}[y_n \neq h_t(\boldsymbol{x}_n)]$$

Namely, the weighted classification error is minimized — precisely *train a weak classifier based on the current weight $w_t(n)$ on the slide* **How Boosting algorithm works?**.

Remarks We can safely assume that $w_t(\boldsymbol{x}_n)$ is normalized so that $\sum_n w_t(\boldsymbol{x}_n) = 1$. This normalization requirement can be easily maintained by changing the weights to

$$w_t(\boldsymbol{x}_n) \leftarrow \frac{w_t(\boldsymbol{x}_n)}{\sum_{n'} w_t(\boldsymbol{x}_{n'})}$$

This change *does not* affect how to choose $h_t^*(\boldsymbol{x})$, as the term $\sum_{n'} w_t(\boldsymbol{x}_{n'})$ is a constant with respect to n.

How to choose β_t?

We will select β_t to minimize

$$(e^{\beta_t} - e^{-\beta_t}) \sum_n w_t(n) \mathbb{I}[y_n \neq h_t(\boldsymbol{x}_n)] + e^{-\beta_t} \sum_n w_t(n)$$

We assume $\sum_n w_t(n)$ is now 1 (cf. the previous slide's Remarks). We take derivative with respect to β_t and set to zero, and derive the optimal β_t as

$$\beta_t^* = \frac{1}{2} \log \frac{1 - \epsilon_t}{\epsilon_t}$$

which is precisely what is on the slide **How Boosting algorithm works?**

Take-home exercise. Verify the solution

Updating the weights

Now that we have improved our classifier into

$$f(\boldsymbol{x}) = f_{t-1}(\boldsymbol{x}) + \beta_t^* h_t^*(\boldsymbol{x})$$

At the t-th iteration, we will need to compute the weights for the above classifier, which is,

$$w_{t+1}(n) = e^{-y_n f(\boldsymbol{x}_n)} = e^{-y_n[f_{t-1}(\boldsymbol{x}) + \beta_t^* h_t^*(\boldsymbol{x}_n)]}$$

Updating the weights

Now that we have improved our classifier into

$$f(\boldsymbol{x}) = f_{t-1}(\boldsymbol{x}) + \beta_t^* h_t^*(\boldsymbol{x})$$

At the t-th iteration, we will need to compute the weights for the above classifier, which is,

$$w_{t+1}(n) = e^{-y_n f(\boldsymbol{x}_n)} = e^{-y_n [f_{t-1}(\boldsymbol{x}) + \beta_t^* h_t^*(\boldsymbol{x}_n)]}$$

$$= w_t(n) e^{-y_n \beta_t^* h_t^*(\boldsymbol{x}_n)}$$

Updating the weights

Now that we have improved our classifier into

$$f(\boldsymbol{x}) = f_{t-1}(\boldsymbol{x}) + \beta_t^* h_t^*(\boldsymbol{x})$$

At the t-th iteration, we will need to compute the weights for the above classifier, which is,

$$\begin{aligned} w_{t+1}(n) &= e^{-y_n f(\boldsymbol{x}_n)} = e^{-y_n[f_{t-1}(\boldsymbol{x}) + \beta_t^* h_t^*(\boldsymbol{x}_n)]} \\ &= w_t(n) e^{-y_n \beta_t^* h_t^*(\boldsymbol{x}_n)} = \begin{cases} w_t(n) e^{\beta_t^*} & \text{if } y_n \neq h_t^*(\boldsymbol{x}_n) \\ w_t(n) e^{-\beta_t^*} & \text{if } y_n = h_t^*(\boldsymbol{x}_n) \end{cases} \end{aligned}$$

Updating the weights

Now that we have improved our classifier into

$$f(\boldsymbol{x}) = f_{t-1}(\boldsymbol{x}) + \beta_t^* h_t^*(\boldsymbol{x})$$

At the t-th iteration, we will need to compute the weights for the above classifier, which is,

$$\begin{aligned} w_{t+1}(n) &= e^{-y_n f(\boldsymbol{x}_n)} = e^{-y_n[f_{t-1}(\boldsymbol{x})+\beta_t^* h_t^*(\boldsymbol{x}_n)]} \\ &= w_t(n)e^{-y_n\beta_t^* h_t^*(\boldsymbol{x}_n)} = \begin{cases} w_t(n)e^{\beta_t^*} & \text{if } y_n \neq h_t^*(\boldsymbol{x}_n) \\ w_t(n)e^{-\beta_t^*} & \text{if } y_n = h_t^*(\boldsymbol{x}_n) \end{cases} \end{aligned}$$

Remarks The key point is that the misclassified data point will get its weight increased, while the correctly data point will get its weight decreased.

Remarks

Note that the AdaBoost algorithm itself never specifies how we would get $h_t^*(\boldsymbol{x})$ as long as it minimizes the weighted classification error

$$\epsilon_t = \sum_n w_t(n)\mathbb{I}[y_n \neq h_t^*(\boldsymbol{x}_n)]$$

In this aspect, the AdaBoost algorithm is a meta-algorithm and can be used with any classifier where we can do the above.

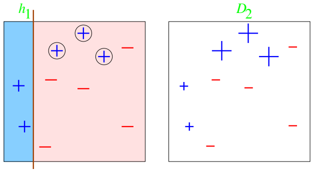
Remarks

Note that the AdaBoost algorithm itself never specifies how we would get $h_t^*(\boldsymbol{x})$ as long as it minimizes the weighted classification error

$$\epsilon_t = \sum_n w_t(n)\mathbb{I}[y_n \neq h_t^*(\boldsymbol{x}_n)]$$

In this aspect, the AdaBoost algorithm is a meta-algorithm and can be used with any classifier where we can do the above.

Ex. How do we choose the decision stump classifier given the weights at the second round of the following distribution?



We can simply enumerate all possible ways of putting vertical and horizontal lines to separate the data points into two classes and find the one with the smallest weighted classification error!

Nonlinear basis learned by boosting

Two-stage process

- Get $\textsc{sign}[f_1(\boldsymbol{x})]$, $\textsc{sign}[f_2(\boldsymbol{x})]$,$\cdots$,
- Combine into a linear classification model

$$y = \textsc{sign}\left\{\sum_t \beta_t \textsc{sign}[f_t(\boldsymbol{x})]\right\}$$

Equivalently, each stage learns a nonlinear basis $\phi_t(\boldsymbol{x}) = \textsc{sign}[f_t(\boldsymbol{x})]$.

One thought is then, why not learning the basis functions and the classifier at the same time?

Outline

KNN: design an algorithm

In nearest neighbor classification, we use the default Euclidean distance

$$\|\boldsymbol{x} - \boldsymbol{x}_n\|_2^2 = \sum_d (x_d - x_{nd})^2$$

One drawback is that the Euclidean distance is sensitive to the unit used for each dimension (for example, if we change the feature "height" from being measured in feet to being measured in inches, the values would be increased by a factor of 12). A simple way to fix this is to use the weighted distance

$$d(\alpha, \beta) = \alpha(x_1 - x_{n1})^2 + \beta(x_2 - x_{n2})^2$$

where we have constrained to two-dimensional features. α and β need to be nonnegative (i.e., $\alpha \geq 0$ and $\beta \geq 0$). For simplicity, we further assume α and β are integers and they are smaller than a known constant B. In other words, you could enumerate all possible α and β.

Please present a viable strategy of identifying the optimal α and β such that the expected classification error of using the distance $d(\alpha, \beta)$ in nearest neighbor classification is minimized.

KNN: analyze an algorithm

Given training data $\{(\boldsymbol{x}_n, y_n)_{n=1}^N\}$, for a new data point $\boldsymbol{x}$, the label y of $\boldsymbol{x}$ is determined by the nearest neighbors of $\boldsymbol{x}$. In the lecture, we discuss the strategy of using K such neighbors, each of which gives a prediction. Then we combine these predictions using the majority vote. We consider a simple extension. Suppose $\boldsymbol{x}'$ is a neighbor of $\boldsymbol{x}$, we give it a soft vote

$$\text{vote}(\boldsymbol{x}') = \exp\{-\|\boldsymbol{x} - \boldsymbol{x}'\|_2^2/\sigma^2\}$$

where σ is a constant to be determined. Note that the farther $\boldsymbol{x}'$ is from $\boldsymbol{x}$, the smaller the vote is. Now, instead of using K neighbors, we use all training data instances. For each instance $\boldsymbol{x}_n$, we compute the vote and add the vote to a bucket corresponding to y_n. Formally,

$$\text{vote}(c) = \sum_n \mathbb{I}[y_n = c]\text{vote}(\boldsymbol{x}_n)$$

where c is one of the class labels and $\mathbb{I}$ is the indicator function (taking value of 1 if its argument is true, or 0 otherwise). We then choose the class label with the largest vote as the label for $\boldsymbol{x}$.

To help you to answer the following questions, we will consider the vote's equivalent form

$$\text{vote}(c) = \sum_n \mathbb{I}[y_n = c] \frac{\exp\{-\|\boldsymbol{x} - \boldsymbol{x}_n\|_2^2/\sigma^2\}}{\sum_{m=1}^M \exp\{-\|\boldsymbol{x} - \boldsymbol{x}_m\|_2^2/\sigma^2\}}$$

where the denominator "normalizes" the vote.

- Consider the case $\sigma \to 0$, of which you should interpret that σ becomes very small. Please show that this corresponds to the studied case $K = 1$.
- Consider the case $\sigma \to +\infty$. Please derive what this corresponds to (i.e., what the classification rule would become).

Regression

Sometimes, we have to consider models where noise is not homogeneous. Specifically, in standard linear regression, we have assumed the following model

$$p(y|\boldsymbol{x}) = \mathcal{N}(y|\boldsymbol{w}^{\mathrm{T}}\boldsymbol{x}, \sigma^2)$$

where σ^2 is the variance of the noise, and is the same for all $\boldsymbol{x}$ (Note that we have absorbed the constant feature 1 into $\boldsymbol{x}$). Let us consider the situation where noise is not homogeneous. For training data $\{(\boldsymbol{x}_n, y_n)_{n=1}^{N}\}$, we will assume a new model

$$p(y_n|\boldsymbol{x}_n) = \mathcal{N}(y|\boldsymbol{w}^{\mathrm{T}}\boldsymbol{x}_n, \sigma_n^2)$$

where σ_n^2 is a noise term that depends on $\boldsymbol{x}_n$. Please derive the maximum likelihood solution of $\boldsymbol{w}$ under the new model.