Christopher Swagler
EECE 5644
Prof Yeh
Problem Set 3

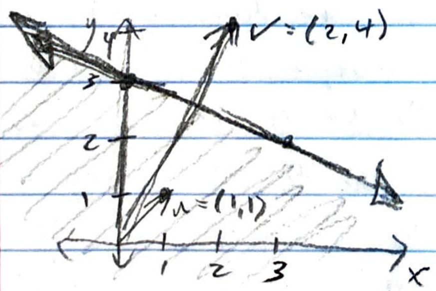
1 partition $\mathbb{R}^d$ into N regions $R_1, \ldots, R_n$ → regions don't overlap except boundary
$g(x) = \hat{y}_k$ for all $x \in R_k$

a $u, v \in \mathbb{R}^d$, $u \neq v$
$S(u,v) = \{x \in \mathbb{R}^d \mid \|x-u\|_2 \leq \|x-v\|_2\}$ → set of points closer to u than v

$\|x-u\|_2 \leq \|x-v\|_2$ holds true iff

$\|x-u\|_2^2 \leq \|x-v\|_2^2$
$(x-u)^T(x-u) \leq (x-v)^T(x-v)$
$\cancel{x^Tx} - 2u^Tx + u^Tu \leq \cancel{x^Tx} - 2v^Tx + v^Tv$
$2v^Tx - 2u^Tx \leq v^Tv - u^Tu$
$2(v-u)^Tx \leq v^Tv - u^Tu$
so in the form $a^Tx = b$,
$a = 2(v-u)$ and $b = v^Tv - u^Tu$

For $u=(1,1)$ and $v=(2,4)$
$a = 2(v-u) = 2(1,3) = (2,6)$
$b = v^Tv - u^Tu = (4+16) - (1+1) = 18$



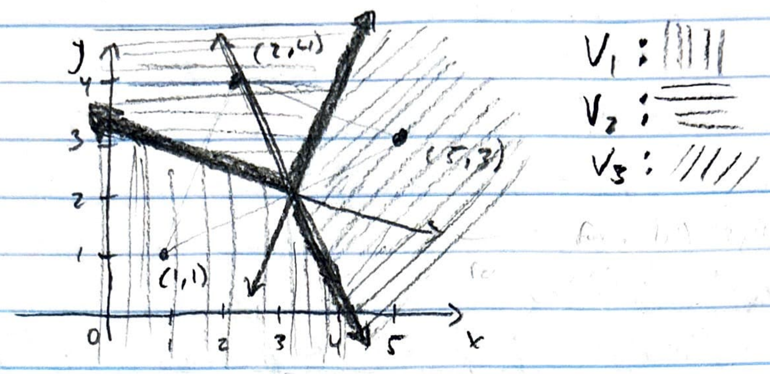
$a^Tx = b$
$(2,6)^Tx = 18$
for (x,y) coordinates → $2x + 6y = 18$
$\Rightarrow y = 3 - \frac{1}{3}x$ is the line/hyperplane

and the set is shaded under the line

b $u^1, \dots, u^m \in \mathbb{R}^d$

$V_i = \bigcap_{j \neq i} S(u^i, u^j)$ → set of points in $\mathbb{R}^d$ closer to x^i

↳ intersection of m-1 halfspaces



for (1,1) (2,4) → $y = 3 - \frac{1}{3}x$

for (1,1) (5,3) → $8x + 4y = 32$ → $y = 8 - 2x$

for (2,4) (5,3) → $6x - 2y = 14$ → $y = -7 + 3x$

The boundary lines for each pair of points was determined and plotted along with the points (1,1) (2,4) (5,3). Then each V_i was determined using the intersection of halfspaces against the other 2 points. This yielded 3 unique regions for each point, and they are denoted using a specific hashing. The 3 lines met at the point (3,2)

c When we consider g as the 1-nearest neighbor predictor on data $x^1, \dots, x^n, y^1, \dots, y^n$, we have $g(x) = y^k$ for $x \in V_k$ where V_k is the Voronoi region for x^k. This is because the voronoi region V_k for x^k is the region in which any point within the region is closer to x^k than any other, therefore the prediction for that point is simply y^k. The corresponding label depends on the Voronoi region the point belongs to since the region consists of points closest to x^k.

d For this case, the k-nearest predictor $g(x)$ takes into account the k nearest neighbors and assigns the most common label among the neighbors, Just like for the 1-nearest neighbor case with Voronoi regions, we instead have polyhedra as the regions but the same principle applies that if an input lies within the polyhedra region, it will be assigned that label. For the $k=2$ example, we take the two nearest points for a given input. Then, the decision boundary is determined by the perpendicular bisector between those points. Instead of having n Voronoi regions, it instead considers two regions dictated by the two closest points and so similar to the $k=1$ case, we have g as a piecewise constant.

2 K-class classification

$Y = (Y_1, \ldots, Y_K)^T$ with

$$Y_k = \begin{cases} 1 & \text{if } G = g_k \\ -\frac{1}{K-1} & \text{otherwise} \end{cases}$$

let $f = (f_1, \ldots f_K)^T$ with $\sum_{k=1}^{K} f_k = 0$

$$L(Y, f) = \exp\left(-\frac{1}{K} Y^T f\right)$$

a we want the population minimizer f^* of $E(Y, f)$ subject to $\sum_{k=1}^{K} f_k = 0$

from the definition of Y_k, we could rewrite

$$-\frac{1}{K} Y^T f_i(x) = \sum_{j \neq i} \frac{f_j(x)}{K(K-1)} - \frac{f_i(x)}{K}$$

and accordingly

$$E(Y, f) = \sum_{i=1}^{K} \exp\left(\sum_{j \neq i} \frac{f_j(x)}{K(K-1)} - \frac{f_i(x)}{K}\right) p(G = g_i | x)$$

using the fact that $\sum_{k=1}^{K} f_k = 0$, we have $\sum_{j \neq i} f_j = -f_i$

and so simplifying the above with this

$$E(Y, F) = \sum_{i=1}^{K} \exp\left(-\frac{f_i(x)}{K-1}\right) p(G = g_i | x)$$

then adding in the lagrange multiplier term,

$$\sum_{i=1}^{K} \exp\left(-\frac{f_i(x)}{K-1}\right) p(G = g_i | x) - \lambda \sum_{i=1}^{K} f_i(x)$$

$\frac{\partial}{\partial f_k}$ and $\frac{\partial}{\partial \lambda}$ the above and set $= 0$

$$\Rightarrow -\frac{1}{K-1} \exp\left(-\frac{f_k(x)}{K-1}\right) p(G = g_i | x) - \lambda = 0 \quad \text{for } k = 1 \ldots K$$

$$\sum_{i=1}^{K} f_i(x) = 0$$

(a) for $k = 1, \ldots, K$

$$f^*_k(x) = (K-1)\log p(G = g_k|x) - \frac{K-1}{K}\sum_{k'=1}^{K}\log p(G = g_{k'}|x)$$

and estimating $p(G = g_k|x)$

$$p(G = g_k|x) = \exp\left(\frac{f_k^*(x)}{K-1}\right)\left(\prod_{j=1}^{K} p(G = g_j|x)^{\frac{1}{K}}\right)$$

summing the above for $k = 1, .., K$ will $= 1$ and give

$$\left(\prod_{j=1}^{K} p(G = g_j|x)^{\frac{1}{K}}\right)\left(\sum_{k=1}^{K}\exp\left(\frac{f_k^*(x)}{K-1}\right)\right) = 1$$

$$\Rightarrow \prod_{j=1}^{K} p(G = g_j|x)^{\frac{1}{K}} = \left(\sum_{k=1}^{K}\exp\left(\frac{f_k^*(x)}{K-1}\right)\right)^{-1}$$

$$\Rightarrow P(G = G_k|x) = \exp\left(\frac{f_k^*(x)}{K-1}\right)\cdot\left(\sum_{k=1}^{K}\exp\left(\frac{f_k^*(x)}{K-1}\right)\right)^{-1}$$

b Adaboost uses the traditional exponential loss function of $L(Y, f) = \exp(-Y^T f)$ whereas we used $L(Y, f) = \exp(-\frac{1}{K}Y^T F)$. Applying our loss function to multiclass boosting will yield a similarly structured reweighting algorithm as adaboost since adaboost is also multiclass boosting, but there will only be slight differences in the actual weighting computation, not the overall algorithmic structure. A difference between the two lies in the computation of the weak classifier's weight (α) at each iteration. Traditional adaboost uses $\alpha = \log\frac{1-err}{err}$ for each iteration's error whereas using this loss function will also incorporate $K-1$ (as $+\log(K-1)$) since this is factored into the probability computation in $P(G = g_k|x)$.

3 Note: I referenced the Elements of Statistical Learning chapter 11 on neural networks, specifically 11.4 on fitting neural networks to understand the derivation on a similar derivation for squared error.

cross entropy

$$R(\theta) = -\sum_{i=1}^{N}\sum_{k=1}^{K} y_{ik} \log f_k(x_i)$$

classifier: $G(x) = \arg\max_k f_k(x)$

we'll let $z_{mi} = \sigma(\alpha_{0m} + \alpha_m^T x_i)$

derived feature, activation function, intercept

and let $z_i = (z_{1i}, z_{2i}, \ldots, z_{Mi})$

then

$$R(\theta) = \sum_{i=1}^{N} R_i = \sum_{i=1}^{N}\sum_{k=1}^{K}(-y_{ik} \log f_k(x_i))$$

we'll derivate with respect to β_{km} and $\alpha_{m\ell}$

$$\frac{\partial R_i}{\partial \beta_{km}} = -\frac{y_{ik}}{f_k(x_i)} g_k'(\beta_k^T z_i) z_{mi}$$

$$\frac{\partial R_i}{\partial \alpha_{m\ell}} = -\sum_{k=1}^{K} \frac{y_{ik}}{f_k(x_i)} g_k'(\beta_k^T z_i)\beta_{km}\sigma'(\alpha_m^T x_i) x_{i\ell}$$

gradient descent update step at $(r+1)$ iteration

$$\beta_{km}^{(r+1)} = \beta_{km}^{(r)} - \gamma_r \sum_{i=1}^{N} \frac{\partial R_i}{\partial \beta_{km}^{(r)}}$$

$$\alpha_{m\ell}^{(r+1)} = \alpha_{m\ell}^{(r)} - \gamma_r \sum_{i=1}^{N} \frac{\partial R_i}{\partial \alpha_{m\ell}^{(r)}}$$

where γ_r is the learning rate for batch learning and is typically a constant

(3) we can rewrite our equation in the form

$$\frac{\partial R_i}{\partial \beta_{km}} = \delta_{ki} z_{mi}$$

$$\frac{\partial R_i}{\partial \alpha_{ml}} = s_{mi} x_{il}$$

$$\Rightarrow \delta_{ki} = -\frac{y_{ik}}{f_k(x_i)} g'_k(\beta_k^T z_i)$$

$$s_{mi} = \sigma'(\alpha_m^T x_i) \sum_{k=1}^{K} \beta_{km} \delta_{ki}$$

the update steps derived above can be implemented with a forward pass and backward pass.
For the forward pass, the current weights are fixed and the predicted values from

$$Z_m = \sigma(\alpha_{0m} + \alpha_m^T X) \quad m = 1,..,M$$
$$T_k = \beta_{0k} + \beta_k^T Z \quad k = 1...K$$
$$f_k(x) = g_k(T) \quad k = 1..K$$

to get the predicted value $\hat{f}_k(x_i)$.
For the backward pass, the errors δ_{ki} are computed then used in the s_{mi} equation. The errors can then be used to compute the gradients for the update step