Operations Research: MATH 5810-001
2.2. A Polyhedral Neural Network Classifier

Devanshu Agrawal

December 4, 2015

Executive Summary

Consider a data set in which every point belongs to one of two "classes". A classifier is an algorithm that "learns" from the data set and then predicts the classes of new points not in the data set. If the data set is visualized as a scatter plot of points in coordinate space, then a classifier draws a "decision boundary" that best separates the two classes of points. The classifier can then predict the class of a new point not in the data set by simply checking to see on which side of the decision boundary the new point lies.

A polyhedron is a convex region in coordinate space with flat faces. By "convex", we mean that the line segment between any two points in the region is itself contained in the region. Examples of polyhedra in two dimensions include triangles, squares, pentagons, etc. In fact, polyhedra may be unbounded; for example, the region between two parallel lines in two dimensions is a polyhedron.

In this report, we describe the construction and implementation of a polyhedral neural network (PNN). A PNN is a classifier that draws a decision boundary that is also the boundary of a polyhedron. In other words, a PNN learns from a set of data (consisting of two classes of points) and produces a polyhedron so that one class of points is contained inside the polyhedron and the other class of points resides outside the polyhedron. The separation is not necessarily perfect but is instead only a best possible separation.

Our PNN produces a polyhedral decision boundary by minimizing the total cost incurred from data points that are misclassified by the PNN. Our PNN is therefore the solution to a linear optimization problem and takes the form of a linear program. We implemented our PNN algorithm in the programming language Maple.

We applied our PNN to an example data set consisting of twenty points selected at random from a square in the standard coordinate

plane (these twenty points comprise one class) and twenty points selected at random outside the square but contained in a larger square in the coordinate plane (these twenty points comprise the second class). Our PNN returned a triangular decision boundary that appeared to separate the two classes of points very well.

1. Problem Description

Consider a set of data $D = \{x_i \in \mathbb{R}^N : i = 1, \ldots, M\}$ that can be partitioned into two classes $C_1 = \{x_1, \ldots, x_p\}$ and $C_{-1} = \{x_{p+1}, \ldots, x_M\}$. Given an integer $1 \leq K \leq M$, the first part of the problem is to find a polyhedron

$$P = \{x \in \mathbb{R}^N : \alpha_k^\top x \geq \beta_k, k = 1, \ldots, k\}, \quad (\alpha_k \in \mathbb{R}^N, \beta_k \in \mathbb{R})$$

that satisfies $C_1 \subset P$ and $C_{-1} \subset \mathbb{R}^N \setminus P$ to the greatest extent possible. In other words, the problem is to find a polyhedron P that best separates the two classes of data points, with points in C_1 contained in P and points in C_{-1} outside of P. Our solution is a PNN algorithm that returns such a polyhedron P as a function of the data set D.

The second part of the problem is to apply the general solution (i.e., the PNN) to an example data set consisting of 20 points selected at random from the square $[1, 2]^2$ (class C_1) and 20 points selected at random from the region $[0, 3]^2 \setminus [1, 2]^2$ (class C_{-1}). We present the solution to this example as a plot that shows a polyhedral decision boundary that separates C_1 and C_{-1}.

2. Background and Resources

A data point is often a list of values or "features" that characterize the point (e.g., an 8×8-pixel image is a list of 64 pixel intensities). The space of all possible lists of features (e.g., the space of all 8×8-pixel images) is called a *feature space*. A data set is therefore a sample of a feature space. Consider a data set that is partitioned into two classes of points (e.g., each data point is an image of either Alice or Bob). A *classifier* is an algorithm that *learns* from the data set and constructs a *decision boundary* that best separates the two classes of data points. The classifier can then predict the class of a point in the feature space not in the data set based on the side of the decision boundary on which the point is located.

A simple example of a classifier is the *hyperplane classifier*. We will find it worthwhile to examine the hyperplane classifier in some detail. Our exposition is based on [1]. Let $D = \{x_i \in \mathbb{R}^N : i = 1, \ldots, M\}$ be a data set of points from the feature space $\mathbb{R}^N$, and suppose D is partitioned into two classes $C_1 = \{x_1, \ldots, x_p\}$ and $C_{-1} = \{x_{p+1}, \ldots, x_M\}$. Let $y_i = j$ if $x_i \in C_j$; thus, y_i is the *class label* of x_i. A *half-space* in $\mathbb{R}^N$ is a set of the form

$$H = \{x \in \mathbb{R}^N : \alpha^\top x \geq \beta\}, \quad (\alpha \in \mathbb{R}^N, \beta \in \mathbb{R}).$$

The boundary of a half-space is a hyperplane. A hyperplane classifier applied to the data set D returns a half-space H that satisfies $C_1 \subseteq H$ and $C_{-1} \subseteq$

$\mathbb{R}^N \setminus H$ to the greatest extent possible (so that the boundary of H best separates C_1 and C_{-1}). Ideally, α and β (which define H) satisfy

$$\begin{cases} \alpha^\top x_i > \beta, & \text{if } x_i \in C_1. \\ \alpha^\top x_i < \beta, & \text{if } x_i \in C_{-1}. \end{cases}$$

More concisely, we can write

$$y_i(\alpha^\top x_i - \beta_i) > 0, \quad \forall i = 1, \ldots, M.$$

But often C_1 and C_{-1} cannot be separated perfectly. Therefore, we only require that H separate C_1 and C_{-1} to the greatest extent possible. In particular, we only require α and β to solve the following linear program:

$$\min_{\alpha,\beta,\xi_i} J = \sum_{i=1}^{M} \xi_i \text{ s.t.}$$

$$\begin{aligned} \xi_i \geq 0, \quad \forall i = 1, \ldots, M \\ y_i(\alpha^\top x_i - \beta) \geq 1 - \xi_i, \quad \forall i = 1, \ldots, M, \end{aligned}$$

or equivalently,

$$\min_{\alpha,\beta,\xi_i} J = \sum_{i=1}^{M} \xi_i \text{ s.t.} \tag{LP1}$$

$$\begin{aligned} 0 \leq \xi_i, \quad \forall i = 1, \ldots, M \\ 1 - y_i(\alpha^\top x_i - \beta) \leq \xi_i, \quad \forall i = 1, \ldots, M. \end{aligned}$$

Define the *hinge loss function* $(\cdot)_+ : \mathbb{R} \mapsto \mathbb{R}^{\geq 0}$ by $(z)_+ = \max(0, z)$. Then, the variables ξ_i in LP1 satisfy the constraint

$$(1 - y_i(\alpha^\top x_i - \beta))_+ \leq \xi_i, \quad \forall i = 1, \ldots, M. \tag{1}$$

If the half-space H classifies x_i correctly, then α and β have values such that $(1 - y_i(\alpha^\top x_i - \beta))_+ = 0$. But if H classifies x_i incorrectly, then $(1 - y_i(\alpha^\top x_i - \beta))_+ > 0$. By minimizing the objective J, we minimize each ξ_i. By Equation 1, the hyperplane classifier returns a half-space H that minimizes the total cost incurred by data points x_i that are misclassified by H.

A *polyhedral neural network* (PNN) is a natural extension of a hyperplane classifier. A *polyhedron* P can be defined as the intersection of half-spaces:

Given vectors $\alpha_k \in \mathbb{R}^N$ and scalars $\beta_k \in \mathbb{R}$ for $k = 1, \ldots, K$, define the half-spaces

$$H_k = \{x \in \mathbb{R}^N : \alpha_k^\top x \geq \beta_k\}, \quad \forall k = 1, \ldots, K. \tag{2}$$

Then, the polyhedron P is defined to be the set

$$\begin{aligned} P &= \bigcap_{k=1}^{K} H_k \\ &= \{x \in \mathbb{R}^N : \alpha_k^\top x \geq \beta_k, k = 1, \ldots, K\}. \end{aligned} \tag{3}$$

A PNN is therefore implemented as a "loop" of hyperplane classifiers. This is the approach taken in [2].

A good classifier constructs a decision boundary that respects the general trend of the two classes of data. A classifier that classifies outlier points without respecting the general trend is said to *overfit*. A *regularizer* is a term that is often added to the objective function to reduce overfitting. for example, an L^1 *regularizer* for the hyperplane classifier is a term that would be added to the objective function J and would have the form

$$R(\|\alpha\|_1 + \beta),$$

where R is the *regularization parameter* that controls the influence of the regularizer and $\|\alpha\|_1$ is the L^1-*norm* of α defined by

$$\|\alpha\|_1 = \sum_{j=1}^{N} |\alpha_j|,$$

where α_j is the jth component of α. The PNN described in [2] includes such a regularizer term.

3. Description of General Solution

We maintain the notation introduced in Section 2. The PNN algorithm described in this section is a modification of the algorithm proposed in [2].

The PNN algorithm begins with some initial polyhedron P: For $k = 1, \ldots, K$, select some random $\alpha_k \in \mathbb{R}^N$ and $\beta_k \in \mathbb{R}$, and define P by Equation 3. The PNN algorithm then adjusts each face of the polyhedron P over several cycles until P provides a sufficiently accurate decision boundary. In particular, the values of α_k and β_k (which define the half-space H_k defined in Equation 2) are adjusted from their initial values until the total cost incurred from data points that are misclassified by P is minimized.

Define the additional variables ξ_i for $i = 1, \ldots, M$, and μ_j for $j = 1, \ldots, N+1$. The PNN adjusts the values of α_ℓ and β_ℓ (i.e., the ℓth face of P) to the values that solve the following linear program:

$$\min_{\alpha_\ell, \beta_\ell, \xi_i, \mu_j} J_\ell = R \sum_{j=1}^{N+1} \mu_j + \sum_{i=1}^{p} \xi_i + \sum_{i=p+1}^{M} w_i^\ell \xi_i \text{ s.t.} \tag{LP2}$$

$$\begin{aligned}
G\gamma_i^\ell &\leq \xi_i, \quad \forall i = 1, \ldots, p && \text{(Constraint 1)}\\
0 &\leq \xi_i, \quad \forall i = p+1, \ldots, M && \text{(Constraint 2)}\\
1 - y_i(\alpha_\ell^\top x_i - \beta_\ell) &\leq \xi_i, \quad \forall i = 1, \ldots, M && \text{(Constraint 3)}\\
(\alpha_\ell)_j &\leq \mu_j, \quad \forall j = 1, \ldots, n && \text{(Constraint 4)}\\
-(\alpha_\ell)_j &\leq \mu_j, \quad \forall j = 1, \ldots, N && \text{(Constraint 5)}\\
\beta_\ell &\leq \mu_{N+1} && \text{(Constraint 6)}\\
-\beta_\ell &\leq \mu_{N+1}, && \text{(Constraint 7)}
\end{aligned}$$

where G is the relaxation parameter (see below), R is a regularization parameter, $(\alpha_\ell)_j$ is the jth component of the vector α_ℓ, and γ_i^ℓ and w_i^ℓ are defined by

$$\gamma_i^\ell = \max\left(\{0\} \cup \{1 - y_i(\alpha_k^\top x_i - \beta_k)\}_{k \neq \ell, k=1}^{K}\right)$$
$$w_i^\ell = \prod_{k \neq \ell, k=1}^{K} (1 - y_i(\alpha_k^\top x_i - \beta_k)) + .$$

Since α_k and β_k are constant in the linear program LP2 for $k \neq \ell$, then γ_i^ℓ and w_i^ℓ are constants.

Before elaborating upon the linear program LP2, we summarize the PNN algorithm by the following informal pseudocode:

```
initialize P  # initial polyhedron

for count from 1 to Iterates do  # Iterates is a positive integer
    for l from 1 to K do
        solve LP2  # solve the linear program
    end do
end do

return P
```

The above pseudocode returns a polyhedron that best separates the two classes of data. The parameter “Iterates” is chosen sufficiently large so that further adjustment of the polyhedron P no longer leads to significant improvement.

We now explain why the linear program LP2 is used to adjust the ℓth face of the polyhedron P (or equivalently, the half-spaceH_ℓ):

The first term in the objective J_ℓ and Constraints 4-7 together act as a regularizer. Constraints 4-5 imply

$$|(\alpha_\ell)_j| \leq \mu_j, \quad \forall j = 1, \ldots, N,$$

and Constraints 6-7 imply

$$|\beta_\ell| \leq \mu_{N+1}.$$

Therefore, we have

$$\|\alpha_\ell\|_1 + |\beta_\ell| = \sum_{j=1}^{N} |(\alpha_\ell)_j| + |\beta_\ell| \leq \sum_{j=1}^{N+1} \mu_j.$$

By minimizing the objective J_ℓ, we minimize each μ_j and hence the regularizer $R(\|\alpha_\ell\|_1 + |\beta_\ell|)$.

The second term in the objective J_ℓ and Constraints 1 and 3 are included to adjust the polyhedron P so that $C_1 \subseteq H_\ell$ is satisfied to a greater extent. Constraints 1 and 3 together imply

$$(1 - y_i(\alpha_\ell^\top x_i - \beta_\ell))_+ = \max\{0, 1 - y_i(\alpha_\ell^\top x_i - \beta_\ell)\} \leq \xi_i, \quad \forall i = 1, \ldots, p.$$

By minimizing J_ℓ, we minimize each ξ_i for $i = 1, \ldots, p$. Recalling the hyperplane classifier and in particular Equation 1, this implies that H_ℓ is adjusted so that points in C_1 are correctly classified by H_ℓ. Since $LP2$ is solved for every $\ell = 1, \ldots, K$, then all faces of P are adjusted in this way.

If a point $\xi_i \in C_1$ is classified incorrectly by at least one half-space H_k, then $1 - y_i(\alpha_k^\top x_i - \beta_k) > 0$ and hence $\gamma_i^\ell > 0$. By Constraints 1 and 3, it follows that the upper bound on $1 - y_i(\alpha_\ell^\top x_i - \beta_\ell)$ is relaxed. This means that if a point $\xi_i \in C_1$ is difficult to classify, then the PNN will “relax” its attempt to classify ξ_i correctly. This “relaxation” scheme is used to reduce overfitting. The parameter G in Constraint 2 controls the amount of relaxation and is restricted to $0 \leq G \leq 1$.

The third term in the objective J_ℓ and Constraints 2-3 are included to adjust P so that $C_{-1} \subseteq \mathbb{R}^N \setminus P$ is satisfied to a greater extent. Constraints 2-3 together imply

$$(1 - y_i(\alpha_\ell^\top x_i - \beta_\ell))_+ = \max\{0, 1 - y_i(\alpha_\ell^\top x_i - \beta_\ell)\} \leq \xi_i, \quad \forall i = p+1, \ldots, M.$$

Suppose $w_i^\ell > 0$. Then, by minimizing J_ℓ, we minimize ξ_i. By Equation 1, it follows that the half-space H_ℓ is adjusted so that $x_i \in \mathbb{R}^N \setminus H_\ell$ and hence $x_i \notin P$. But suppose $x_i \in \mathbb{R}^N \setminus H_k$ for some $k \neq \ell$. Then, $(1 - y_i(\alpha_k^\top x_i - \beta_k))_+ = 0$ and hence $w_i^\ell = 0$. In this case, the objective J_ℓ is independent of ξ_i. The overall effect is that if a point $x_i \in C_{-1}$ is excluded by at least one face of P, then there is no need to adjust the other faces to exclude the same point.

We implemented the above algorithm in the programming language Maple. For the Maple code, see Section A1.

4. Application to the Specific Context

We construct an example data set D as follows: Randomly select points $x_1, \ldots, x_{20} \in [1,2]^2$ and $x_{21}, \ldots, x_{40} \in [0,3]^2 \setminus [1,2]^2$. The points $x_{21}, \ldots, x_{40}$ can be generated by randomly selecting points in $[1,2]^2$ and then randomly selecting and adding a point from the finite set $\{(a,b)\}_{a,b=-1,0,1} - \{(0,0)\}$. Our data set is $D = \{x_1, \ldots, x_{40}\}$. The two classes of points in D are $C_1 = \{x_1, \ldots, x_{20}\}$ and $C_{-1} = \{x_{21}, \ldots, x_{40}\}$.

We applied a PNN on the data set D with parameter values $R = 0.01$ (regularization parameter), $G = 0.5$ (relaxation parameter), $K = 5$ (maximum number of faces possible for the polyhedron P), and "Iterates" $= 10$ (number of cycles over $k = 1, \ldots, K$, or equivalently the number of times each face of P is adjusted). The PNN returned a triangle as the polyhedron P and hence a triangular decision boundary (Figure 1).

5. Interpretations and Conclusions

The triangular decision boundary obtained for the example data set in Section 4. separates the two classes of data points very well (Figure 1). Our implementation of a PNN classifier was therefore successful in the example application and might be extensible to other data sets as well.

Because the data in the example application was generated inside square regions, then we might have expected the PNN to return a square decision boundary (in particular, the boundary of the square polyhedron $[1,2]^2$). Even though we set the maximum possible number of faces to $K = 5$, the PNN returned a triangular decision boundary. The triangle is a special polyhedron because it is the simplest polyhedron in two dimensions and is therefore also known as a 2-simplex (at least three sides are needed in two dimensions to enclose nonzero and finite area). It appears then that our PNN might remain faithful to the "simplicial topology" of the feature space (i.e., the structure

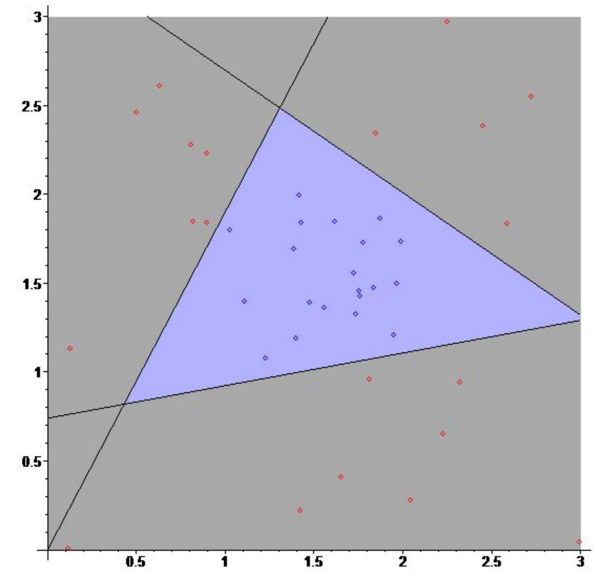

Figure 1: Plot of the polyhedron (light blue) returned by the PNN when applied to the two classes of points C_1 (blue) and C_{-1} (red). The polyhedron is a triangle and therefore defines a triangular decision boundary.

of space as a complex of triangles glued together) [3].

Upon further testing, we found that the relaxation parameter should be set to $G = 1$ for a low number of faces ($K = 3, 4$) and $G = 0.5$) for a higher number of faces ($K = 5, 6, 7$). The relaxation parameter therefore plays an important role. We also found that for $K = 6, 7$, the decision boundary is quadrilateral (although this could change if we modify G or the number of data points). Future work will therefore include further testing and investigation into the role of topology in the PNN.

Bibliography

[1] Linear binary classification. http://inst.eecs.berkeley.edu/~ee127a/book/login/l_lqp_apps_class.html. Accessed: 2015-11-19.

[2] M Murat Dundar, Matthias Wolf, Sarang Lakare, Marcos Salganicoff, and Vikas C Raykar. Polyhedral classifier for target detection: a case study: colorectal cancer. In *Proceedings of the 25th international conference on Machine learning*, pages 288–295. ACM, 2008.

[3] Jeff Knisley. Unpublished notes. Accessed: 2015-11-20.

A1. Code for the General Solution

```
restart;
with(Optimization):

# "PNN" is the polyhedral neural network procedure.
# "PNN" takes as inputs two lists "Positives" and "
   Negatives", an integer "K" (K >= 1),
# real parameters "R" (R >= 0) and "G" (0 <= G <= 1)
   , and a positive integer "Iterates".
# "Positives" is a list of data points in Class 1 (i
   .e, positive class).
# "Negatives" is a list of data points in Class -1 (
   i.e., negative class).
# "PNN" returns a list "PolyhedronBounds" of
   inequalities that together define a polyhedral
   region
# whose boundary separates "Positives" and "
   Negatives", with "Positives" inside the region.
# "K" is the maximum number of faces that the
   polyhedral region may have.
```

```
# "R" is the regularization parameter for the normal
    vectors and off-sets defining the faces of the
   polyhedron.
# "R" is the regularization parameter.
# "G" is the relaxation parameter.
# "Iterates" is the number of iterations in the
   training of the PNN.

PNN := proc(Positives, Negatives, K, R, G, Iterates)
local x,y,alpha,beta,p,M,N,i,j,k,l,count,e,w,sol,J,
   Constraints,PolyhedronBounds:

p := nops(Positives):  # number of data points in
   Class 1
M := p+nops(Negatives):  # total number of data
   points

x := [op(Positives), op(Negatives)]:  # list of data
    points
y := [seq(1, i=1..p), seq(-1, i=1..M-p)]:  # list of
    class labels for each data point
i:='i':

N := nops(x[1]):  # dimension of feature space

alpha := [seq([0, 1], i=1..K)]:  # initialize normal
    vectors to polyhedron faces
beta := [seq(0, i=1..K)]:  # initialize off-sets of
   polyhedron faces

# train the neural network

for count from 1 to Iterates do
for l from 1 to K do

for k from 1 to K do
for i from 1 to M do
e[k,i] := max(0, 1-y[i]*(add(alpha[k][j]*x[i][j], j
   =1..N)-beta[k])):
end do:
end do:
```

```
k:='k': i := 'i':

for i from 1 to M do
w[i] := mul(e[k,i], k in {seq(1..K)} minus {l}):
end do:
i:='i': j:='j': k:='k':

J := R*sum(mu[i], i=1..N+1) + sum(w[i]*xi[i], i=p
   +1..M) + sum(xi[i], i=1..p):  # objective
   function

Constraints := {
seq( xi[i] >= 0, i=1..M),
seq( xi[i] >= 1-y[i]*(sum(alpha_new[j]*x[i][j], j
   =1..N)-beta_new), i=1..M),
seq( xi[i] >= G*max(seq(e[k,i], k in {seq(1..K)}
   minus {l})), i=1..p),
seq(alpha_new[i] <= mu[i], i=1..N),
seq(-alpha_new[i] <= mu[i], i=1..N),
beta_new <= mu[N+1],
-beta_new <= mu[N+1]
}:

sol := LPSolve(J, Constraints):  # solve linear
   program
alpha[l] := [seq(rhs(sol[2][j]), j=-N..-1)]:
beta[l] := rhs(sol[2][1]):

end do:
end do:


# alpha and beta are now lists of normal vectors and
    off-sets (respectively) of the polyhedron faces
# create set of nontrivial inequalities that define
   the polyhedron

PolyhedronBounds := {}:
for k from 1 to K do
if add(abs(alpha[k][j]), j=1..N)+abs(beta[k]) >= 1e
   -5 then
```

```
PolyhedronBounds := {op(PolyhedronBounds), sum(alpha
   [k][j]*u[j], j=1..N) >= beta[k]}:
end if:
end do:

return(PolyhedronBounds):
end proc:

# Generate random data in the square [0, 3]^2.
# Positive points are chosen in [1, 2]^2.
# Negative points are chosen outside [1, 2]^2.

with(stats[random]):
rand8 := rand(1..8):
Blocks :=
   [[-1,1],[0,1],[1,1],[-1,0],[1,0],[-1,-1],[0,-1],[1,-1]]:

Positives := [seq([uniform[1,2](2)], i=1..20)]:  #
   points in Class 1
Negatives := [seq([uniform[1,2](2)]+Blocks[rand8()],
    i=1..20)]:  # points in Class -1

# Train the PNN to obtain a decision boundary.

PolyhedronBounds := PNN(Positives, Negatives, 5,
   0.01, 0.5, 10):

# List the inequalities defining the decision
   boundary.

for i from 1 to nops(PolyhedronBounds) do
print(evalf(PolyhedronBounds[i], 4));
end do:

# Plot the data points and the decision boundary.
# Note: The code below only works in a N=2-
   dimensional feature space.

with(plots):
P_1 := inequal(PolyhedronBounds, u[1]=0..3, u
   [2]=0..3):  # decision boundary
```

```
P_2 := pointplot(Positives, color=blue):  # data
   points in Class 1
P_3 := pointplot(Negatives, color=red):  # data
   points in Class -1
display(P_1, P_2, P_3, scaling=constrained);
```