CMPE 257 Machine Learning

HW#4

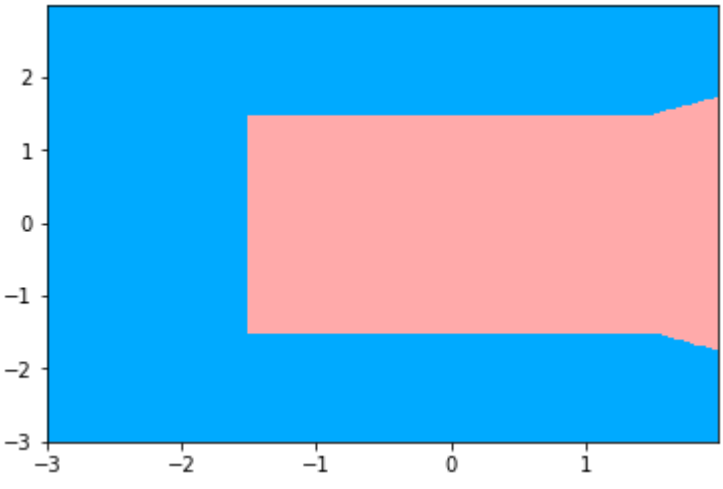
Name: Akshata Deo
Sjsu id: 012565761

Part 1

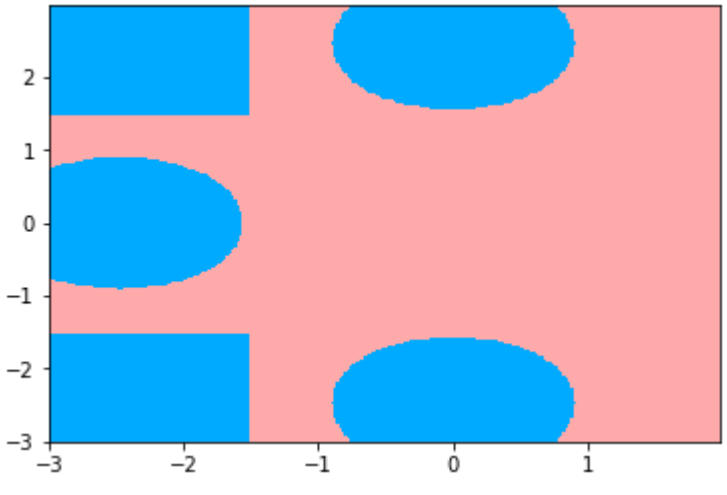
1.

a. For 1-NN rule

X = np.array([[1,0],[0,1],[0,-1],[-1,0],[0,2],[0,-2],[-2,0]])
Y = np.array([-1,-1,-1,-1,1,1,1])
Nearest_neighbours = 1
KNN = neighbors.KNeighborsClassifier(nearest_neighbours, weights='distance')
Decision Region: -



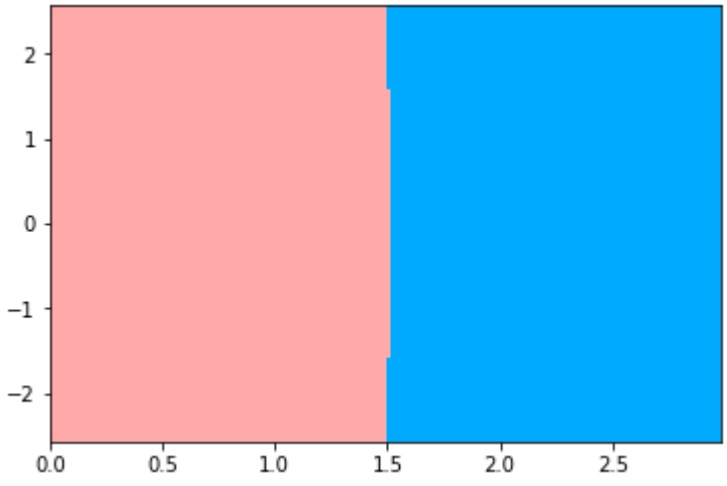
For 3-NN rule
Nearest_neighbours = 3
KNN = neighbors.KNeighborsClassifier(nearest_neighbours, weights='distance')
Decision region: -



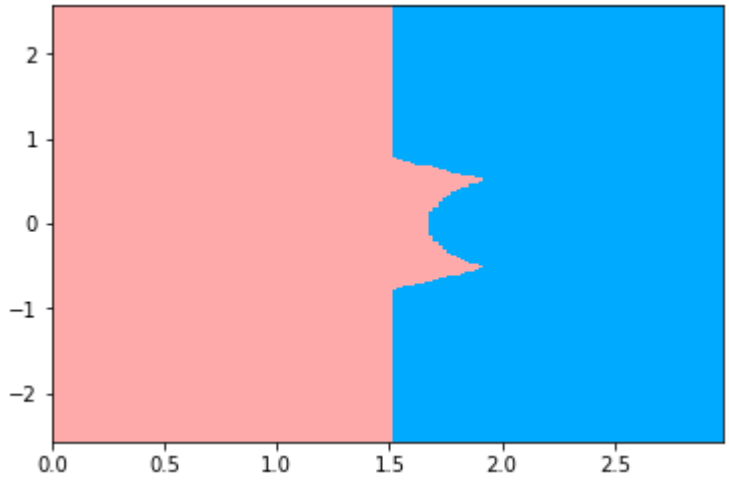
b.

Applying the non-linear transform

$Z1 = \text{sqrt}((X[:,0])^2 + (X[:,1])^2)$
$Z2 = \arctan(X[:,1]/X[:,0])$
For 1-NN, the decision region is



For 3-NN, the decision region is



---

2. Using graph representation-
$h1(x) = \text{sign}(w1^T x)$ & $h2(x) = \text{sign}(w2^T x)$
At first layer we have these three signals: 1, h1(x), h2(x).
At second layer we have following three signals-
1st signal = 1

2nd signal= sign(w0 + w1h1(x) + w2h2(x))
= sign((-1.5)(1) + (h1(x))(1) + (h2(x))(-1))
= sign(-1.5 + h1(x) – h2(x))

3rd signal = sign(w0 + w1h1(x) + w2h2(x))
= sign((-1.5)(1) + (h1(x))(-1) + (h2(x))(1))
= sign(-1.5 - h1(x) + h2(x))

So, final equation for the target function f(x) will be the addition of all the three signals from previous layer.
f(x) = sign(1.5 + sign(-1.5 + h1(x) – h2(x)) + sign(-1.5 - h1(x) + h2(x)))
Hence Proved

---

3. Given in sample error is-
$Ein(w) = (1/N) \sum_{n=1toN} (\tanh(w^T x_n) - y_n)^2$
Differentiate above equation

$d(Ein(w)) = (2/N) \sum_{n=1toN} (\tanh(w^T x_n) - y_n) *$ differentiation of $(\tanh(w^T x_n) - y_n)$
$d(Ein(w)) = (2/N) \sum_{n=1toN} (\tanh(w^T x_n) - y_n) (1-\tanh^2(w^T x_n)) *$ differentiation of $(w^T x_n)$
$d(Ein(w)) = (2/N) \sum_{n=1toN} (\tanh(w^T x_n) - y_n) (1-\tanh^2(w^T x_n)) x_n$
Hence Proved

If w goes to infinity $\tanh^2(w^T x_n)$ will go to 1. Therefore, the term $(1-\tanh^2(w^T x_n))$ will go to 0. This creates a problem because the minimum is achieved without minimizing the error. So, that is why it is hard to optimize perceptron.

---

4. There is a single input, and weight matrices are-
W1 = [[0.1,0.2],[0.3,0.4]]; W2 = [[0.2],[1],[-3]]; W3 = [[1],[2]]
For the data points X=2, and Y=1, forward propagation gives-
X0 = [[1],[2]]
$S1 = W1^T X0$
= [[0.1,0.3],[0.2,0.4]] [[1],[2]]
= [[0.7],[1.0]]
X1 = [[1.0],[0.7],[1.0]]
$S2 = W2^T X1$
= [0.2,1,-3] [[1.0],[0.7],[1.0]]
= [-2.1]
X2 = [[1.0], [-2.1]]
$S3 = W3^T X2$
= [1,2] [[1.0], [-2.1]]
= [-3.2]
X3 = [-3.2]

Now back-propagation gives-
d3 = [2(XL - Y)]
= [2(-3.2 - 1)]
= [-8.4]
$d2 = \Theta'(s^2) * W3\ d3$
= [2] [-8.4]
= [-16.8]
$d1 = \Theta'(s^1) * W2\ d2$
= [[1],[-3]] [-16.8]
= [[-16.8],[50.4]]

$de/dW1 = X0\ (d1)^T$
= [[1],[2]] * [-16.8,50.4]
= [[-16.8,50.4],[-33.6, 100.8]]
$de/dW2 = X1\ (d2)^T$
= [[1.0],[0.7],[1.0]] * [16.8]
= [[-16.8], [-11.823], [-16.8]]
$de/dW3 = X2\ (d3)^T$
= [[1.0], [-2.1]] * [-8.4]

= [[-8.4], [17.64]]

Part 2

1. File attached.
2. File attached.

3. Variable rate gradient descent: - If we implement variable rate gradient descent with conservative incremental parameters (alpha 1.05 - 1.1 and beta 05 - 0.8) in our code, then the algorithm will be faster. However, if the parameters are too big there is a chance of missing minima.
Steepest Descent: - convergence or minima would be attained faster (less iteration) than batch gradient descent or variable rate gradient descent. However, It is hard to know when the actual convergence is achieved. It is also hard to differentiate between flat surface and local minima.
Conjugate descent: - Fastest algorithm. Minima or convergence will be attained by running even less example.

REFRENCE:

1. book forum: http://book.caltech.edu/bookforum/