Evan Nuss

CS 472 Midterm #1

1. Single Layer perceptron.

$\Delta w_i = c(t-z)x_i$

$c = 0.6 \qquad w = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}$

First Epoch:

$net_0 = 0 \quad z_0 = 0$

$\Delta w_0 = 0.6(1-0)1 = 0.6$

$\Delta w_1 = 0.6(1-0)-0.5 = -0.3$

$\Delta w_2 = 0.6(1-0)1 = 0.6$

$\Delta w_b = 0.6(1-0)1 = 0.6$

$net_1 = 0.5(0.6) + 0.7(-0.3) + 0.5(0.6) + 0.6$
$= .99 \quad z_1 = 1$

$\Delta w = \begin{bmatrix} -0.3 \\ -0.42 \\ -0.3 \\ -0.6 \end{bmatrix} \quad w = \begin{bmatrix} 0.3 \\ -0.72 \\ 0.3 \\ 0 \end{bmatrix}$

$net_2 = (0.3) + (-0.5)(-0.72) + 0.5(0.3) + 0 = 0.21 \quad z_2 = 1$

$\Delta w = [0\ 0\ 0\ 0] \quad w = [0.3\ \ -0.72\ \ 0.3\ \ 0]$

$net_3 = 0 + (-0.72) + 0.3^2 = -0.63 \quad z_3 = 0$

$\Delta w = [0\ 0\ 0\ 0] \quad \boxed{w = [0.3\ \ -0.72\ \ 0.3\ \ 0]}$

~~convergence achieved this epoch.~~

Second Epoch

$net_0 = 0.3 + 0.5(0.72) + 0.3 = 0.96 \quad z_0 = 1$

No weight changes necessary.

$net_1 = 0.5(0.3) - 0.7(0.72) + 0.5(0.3) = -0.204 \quad z_1 = 0$

No weight changes necessary. We have gone through all 4 datapoints with no weight changes, therefore convergence has been achieved.

$w = [0.3\ \ -0.72\ \ 0.3\ \ 0]$

2. Feature Reduction / Selection

A. PCA Algorithm:

```
x = dataset
means = mean( for each column of x)   # find mean of each
                                      # feature
x - means          # subtract the mean of the corresponding
                   # feature from each value
covariance(x)      # create the covariance matrix for x.
eigenvectors(x) # find the eigenvalues & eigenvectors for
eigenvalues(x) # each column of x (feature)

p = 2              # how many of the ~~eccs~~ eigenvectors with
                   # the highest eigenvalues we will select
sort (eigenvectors by eigenvalues)
eigenvectors = top p rows

T = fancy_matrix_multiply (eigenvectors, stuff)

bar_graph(T)       # This will display how much information
                   # we get from these particular features
```

B. Greedy Wrapper Algorithm: Backwards search

```
S = set of features
~~for each feature in S:~~ while (S.size > desired_size):
for each feature in S:
    train (feature)
    a = accuracy (feature)

S. remove (feature with lowest accuracy)
```

C. Pros & cons.

The backwards search can be very thorough, but also quite expensive since we are training our model many times.

Principal Component Analysis does not have many cons except that there is no set threshold of information at which a feature should be removed, which could cause us to remove a feature that the model actually did need. A pro is that it does not require training to use this algorithm.

3. Backpropagation.

$w_{21} = -1.5$ $w_{51} = -1$ $w_{53} = -1.2$
$w_{31} = 2$ $w_{62} = 0.4$ $w_{63} = -0.5$ $x = \begin{bmatrix} .1 \\ .6 \end{bmatrix}$ $t = 1$
$w_{41} = 0.5$ $w_{72} = 1.2$ $w_{73} = 1.6$ $c = 0.5$

$\Delta w_{ij} = c\, a_i\, \delta_j$ $a_i = \frac{1}{1+e^{-net_i}}$ $f'(net_i) = a_i(1-a_i)$

output: $\delta_i = (t - a_i)\, f'(net_i)$
hidden: $\delta_i = \sum_k (\delta_k w_{ik})\, f'(net_i)$

$net_2 = -0.1 + 0.6(0.4) + 1.2 = \boxed{1.34}$ $a_2 = \frac{1}{1+e^{-1.34}} = \boxed{0.79}$
$net_3 = -1.2(0.1) - 0.6(0.5) + 1.6 = \boxed{1.18}$ $a_3 = \frac{1}{1+e^{-1.18}} = \boxed{0.77}$
$net_1 = -0.79(1.5) + 0.77(2) + 0.5 = \boxed{0.86}$ $a_1 = \frac{1}{1+e^{-0.86}} = \boxed{0.70}$

$\delta_1 = (1 - 0.7)\, 0.7(1-0.7) = \boxed{0.063}$
$\Delta w_{21} = 0.5(0.79)(0.063) = \boxed{\Delta w_{21} = 0.025}$
$\Delta w_{31} = 0.5(0.77)(0.063) = \boxed{\Delta w_{31} = 0.024}$
$\Delta w_{41} = 0.5(1)(0.063) = \boxed{\Delta w_{41} = 0.032}$

$\delta_2 = 0.063(-1.5)\, 0.79(1-0.79) = \boxed{\delta_2 = -0.016}$
$\delta_3 = 0.063(2)\, 0.77(1-0.77) = \boxed{\delta_3 = 0.022}$

$\Delta w_{52} = 0.5(0.1)(-0.016) = \boxed{\Delta w_{52} = -0.0008}$
$\Delta w_{62} = 0.5(0.6)(-0.016) = \boxed{\Delta w_{62} = -0.0048}$
$\Delta w_{72} = 0.5(1)(-0.016) = \boxed{\Delta w_{72} = -0.008}$
$\Delta w_{53} = 0.5(0.1)(0.022) = \boxed{\Delta w_{53} = 0.0011}$
$\Delta w_{63} = 0.5(0.6)(0.022) = \boxed{\Delta w_{63} = 0.066}$
$\Delta w_{73} = 0.5(1)(0.022) = \boxed{\Delta w_{73} = 0.011}$

$w_{21} = -1.48$	$w_{52} = -1.0008$	$w_{53} = -1.19$
$w_{31} = 2.024$	$w_{62} = 0.39$	$w_{63} = -0.43$
$w_{41} = 0.53$	$w_{72} = 1.19$	$w_{73} = 1.61$

4. Quadric Machine.

A. I don't know what you mean by layers, but I think these will be 2, one for the normal inputs & another for the quadratic inputs.

B. We would use the perceptron rule since it is for binary classification.

C. It does well with binary classification problems with data that isn't necessarily linearly separable.

D. $net = w_x X + w_y Y + w_{x^2} X^2 + w_{y^2} Y^2 + w_{xy} XY$

$= 0.1(-0.2) + 0.1(1.1) + 0.1(0.04) + 0.1(1.21) + 0.1(-0.22)$

$= 0.193$

$z = 1$

No weight changes. All weights still $= 0.1$

5. Overfitting

A. Overfitting can happen due to too much training on the training set, or because we've added in duplicates of datapoints to give our set a more normal distribution. It can also be caused by using the same training set throughout all training.

B. 1. N-fold cross-validation. This involves dividing a dataset into N equally sized subsets, then training the model, starting with K=1 & going to N, with dataset minus $subset_K$, then testing the model's accuracy on $subset_K$. This will work for any algorithm.

2. Random split training & validation. Train the model multiple times each time using a different split of the dataset for the training & validation sets. Ex: first training, 90% training set, 10% validation set, second training, 70% training set, 30% validation set. The data should be shuffled each time to reduce overlap across training sessions. This will work for any algorithm.

3. Make sure a feature isn't too highly correlated with the output. For example, if you collected all the data for democrats on a Wednesday, & all the data for republicans on a Tuesday, & included the day of data collection as a feature, the model would overfit a ton & be helpless when presented with real data. This applies for any algorithm.

6. Decision Tree

$Info(S) = \sum_{i=1}^{|S|} \frac{|S_i|}{|S|} Info(S_i)$ $\quad Info(S_i) =$

~~[illegible]~~

$prob_N = 4/7$

$prob_P = 3/7$

$Prob_{single} = 4/7$ $\quad Info_{single} = -\frac{4}{7}\left(\frac{1}{2}\log_2\frac{1}{2} + \frac{1}{2}\log_2\frac{1}{2}\right)$

$Prob_{S,N} = 2/4$ $\quad = 0.571$

$Prob_{S,P} = 2/4$

$Prob_{Married} = 3/7$ $\quad Info_{Married} = -\frac{3}{7}\left(\frac{2}{3}\log_2\frac{2}{3} + \frac{1}{3}\log_2\frac{1}{3}\right)$

$Prob_{M,N} = 2/3$ $\quad = 0.394$

$Prob_{M,P} = 1/3$

$Info_{Marital\ Status} = 0.965$

$Prob_T = 4/7$ $\quad Info_T = 0.571$

$Prob_{T,N} = 2/4$

$Prob_{T,P} = 2/4$

$Prob_{Short} = 3/7$ $\quad Info_{Short} = 0.394$

$Prob_{S,N} = 2/3$

$Prob_{S,P} = 1/3$

$Info_{Height} = 0.965$

$Prob_{Hi} = 3/7$
$Prob_{Hi,N} = 1/3$
$Prob_{Hi,P} = 2/3$

$Info_{Hi} = 0.394$

$Prob_{Med} = 2/7$
$Prob_{M,N} = 1/2$
$Prob_{M,P} = 1/2$

$Info_{Med} = -\frac{2}{7}\left(\frac{1}{2}\log_2\left(\frac{1}{2}\right) + \frac{1}{2}\log_2\left(\frac{1}{2}\right)\right)$
$= 0.286$

$Prob_{Lo} = 2/7$
$Prob_{L,N} = 2/2$
$Prob_{L,P} = 0/2$

$Info_{Lo} = -\frac{2}{7}\left(\log_2(1)\right)$
$= 0$

$Info_{GPA} = 0.68$

Due to having the least information (entropy), we should split on GPA first.

7

~~6~~. Dealing with unknown/missing data.

1. Set the missing value equal to the mean or mode of the rest of the datapoints for that feature.
2. Simply include "missing" as another possible value for the feature. Works best for nominal data.
3. Train another model without that feature, & when the original model encounters a datapoint with that missing value, it will send that datapoint to the new model to be classified.

perceptron rule: $y=1$ if net >0, bias input always 1. $\Delta w_i = c(t-y)x_i$, output, good for binary classification

Delta rule: $\Delta w_i = c(t-net)x_i$. Not good for classification, but good for regression.

Stochastic update/gradient descent: update after each iteration

Batch update/gradient descent: update after each epoch.

Quadric Machine: uses perceptron rule to update weights but also includes quadratics of inputs multiplied together. x^2, y^2, xy

Decision Boundary equation: $x_2 = (-w_1/w_2)x_1 + \theta/w_2$

$w_1 x_1 + w_2 x_2 = \theta \rightarrow x_2 + \frac{w_1 x_1}{w_2} = \frac{\theta}{w_2}$

Delta rules aims to minimize the SSE of a regression line.

Perceptron rule aims to maximize correct classification.

Backprop: $\Delta w_{ij} = c\, o_i\, \delta_j$ • "output" of input nodes is just their input value

$o_i = \frac{1}{1+e^{-net_i}}$

$f'(net_i) = o_i(1-o_i)$

$\delta_i = (t_i - o_i) f'(net_i)$ * output node

$\delta_i = \sum_k (\delta_k w_{ik}) f'(net_i)$ * hidden node

• find nets of hidden nodes, then outputs of hidden nodes, then nets of next layer, outputs of next layer, then δ if output layer for each node, then weight changes for weights leading to output nodes.

k: all previous layer nodes.

momentum in backprop: $\Delta w_{ij}(t+1) = c\alpha_i \delta_j + \alpha \Delta w_{ij}(t)$

α usually = .9

Grid search: choose potential hyperparameters & try all combinations. Only useful if not a lot of hyperparameter options.

N-Fold Cross-validation: use all data for training & validation.

k=1 to N, M_k uses $D - S_k$ to train, then tested on S_k

a_k = accuracy of M_k on S_k. total accuracy: $\frac{(a_1 + a_2 + \ldots + a_n)}{N}$

Random search: user chooses distribution over a range of values for each hyperparameter. Ex. # of hidden nodes = uniform (5, 50)

L1: $\sum |t_i - z_i|$

SSE/L2: $\sum (t_i - z_i)^2$

MSE: $\frac{SSE}{\#\text{ of datapoints}}$

RMSE: $\sqrt{MSE}$

If multiple outputs & targets for each datapoint, L1, L2, & MSE for dataset will be sums across values for each target, output node, then RMSE will still be $\sqrt{MSE_{total}}$

$Info(S) = \sum_{i=1}^{n} \frac{|S_i|}{|S|} Info(S_i)$ data with specific attribute / all datapoints in current set

1. calculate baseline entropy of S
2. calculate entropy of each feature.
3. split on feature with lowest entropy.
4. split S based on feature.
5. repeat with new S.