CS 540 (Shavlik) HW 4 –ANNs and SVMs

Assigned:	11/8/16
Due:	11/22/16 at 11:55pm (not accepted after 11:55pm on 12/1/16)
Points:	100

Be sure to show your work for all of the questions. You need to turn in `HW4.pdf` (your solutions to Problems 1-4 plus your lab report on training the perceptron on WINE) and `HW4.java`.

Problem 1 – Perceptrons (15 points)

Consider the following examples:

```
ex1:  F1 = 1   F2 = 1   F3 = 0   category = 1

ex2:  F1 = 1   F2 = 0   F3 = 1   category = 0

ex3:  F1 = 0   F2 = 1   F3 = 1   category = 1
```

Create a perceptron (i.e., an artificial neural network with *no hidden units*) for this task. Do not forget about the input unit whose value is always '-1' and whose weight represents the threshold for the output unit. Initially set all the weights to 0.1, then show the resulting network after each of the following scenarios. Assume that if the weighted input equals or exceeds the threshold the perceptron outputs 1 otherwise it outputs 0 (see top of page 724 of the textbook). Use the *perceptron learning rule* (Equation 18.7 of the textbook) with the learning rate $\alpha = 0.1$.

i. training on ex1

ii. training on ex1 followed by training on ex2

iii. training on ex1 followed by training on ex2 followed by training on ex3

Each of the above parts picks up where the last one left off so you only need to show your work for the final step in each part, but do show THREE network states, that is, one for each of the above three questions.

iv. What output does your trained perceptron produce after Part *iii* above for this example?

```
exNew:  F1 = 0  F2 = 1 F3 = 0 category = ?
```

Problem 2 – Hidden Units (15 points)

i. Using the dataset in Problem 1, draw a neural network with <u>two</u> hidden units. Connect every input unit to every hidden unit and every hidden unit to the output unit. The only connection from the input layer to the output unit should be from the '-1' input unit. Initially set all weights and thresholds to 0.1. As done in Problem 1, set $\alpha = 0.1$.

   The activation function for the *hidden* units for this question should be the Rectified Linear Unit (ReLU), where *x* is the weighted sum and includes the '-1' unit's contribution. The activation function for the *output* unit should be 'purely' linear, i.e. $f(x) = x$.

ii. Show, in your answer to Part *i*, the activations of the hidden and output units for *ex1* from Problem 1. Show your work below the network.

iii. Apply the back-propagation algorithm (Figure 18.24 of the text) to your answer from Part *ii*. (However, rather than assigning random weights, assign all weights to 0.1 to simplify grading.) Draw the resulting neural network and show your work above it (i.e., re-draw your answer to Part *i*, but use the new weights). Note: the line "for l = L-1 to 1 do" in Fig 18.24 should be "for l = L-1 to 2 do" since Layer 1 contains the input units.

   The derivative of the ReLU (*g'(in)* in Figure 18.24) is:

   0 if $in \leq 0$ and 1 otherwise.

   The derivative for the linear output unit is 1.

Problem 3 – Weight Space (10 points)

Assume we have a perceptron with one input unit and one output unit, where the output unit's activation function is the *sigmoid*, i.e. $output = 1 / (1 + e^{-x})$ where x is the weighted sum and includes the '-1' unit's contribution. For simplicity, in this problem we will fix the threshold to be 1.5. Hence, we have only <u>one</u> free parameter, the weight between the input unit and the output unit.

Also assume we have this dataset:

```
ex1:  F1 = 1   category = 1
ex2:  F1 = 2   category = 0
ex3:  F1 = 3   category = 0
```

We will define the error of the network on ex_i, to be the square of the difference between the category of ex_i and the predicted output of the network for that example.

Draw a graph where the *x*-axis is the weight (<u>not</u> the weighted sum) and the *y*-axis is the total error over the three examples above. Create this graph by letting the weight be one of the values in this set and then 'connect the dots' to create a curve: {-4, -2, -1, 0, 1, 2, 4}.

Hint: I recommend writing some Java code to compute the errors (but no need to turn in the code).

Problem 4 – Support Vector Machines (20 points)

Consider the following new features (F1 through F3), where *ex1* through *ex3* are from Prob. 1:

F1(ex_i) = similarity between *ex1* and ex_i features (i.e., number of bits in common)

F2(ex_i) = similarity between *ex2* and ex_i

F3(ex_i) = similarity between *ex3* and ex_i

i. Convert Problem 1's dataset into this representation, i.e. a table with 3 rows and 4 columns, where the 4th column is the output category (which is not used in the above similarity functions and hence is unchanged). Write out the dataset in a manner similar to that used at the start of Problem 1, but instead using F1(ex_i), F2(ex_i), and F3(ex_i) as the features.

ii. Repeat Problem 1i through Problem 1iii, but this time use the weight-update rule below (called "weight decay"), with $\alpha = 0.1$ and $\lambda = 0.2$ (i.e., draw THREE network states). Do not forget about the 'dummy' input unit whose weight represents the output unit's threshold and again initialize all weights to 0.1.

$$w_i \leftarrow w_i + [\ \alpha \times (\text{category}(ex) - \text{perceptronOutput}(ex)) \times \text{feature}_i(ex)\ ] - \alpha \times \lambda \times w_i$$

This is essentially what a Support Vector Machine (SVM) does, though here we are using gradient descent rather than linear or quadratic programming to find good weights.

iii. Repeat Problem 1*iv* using the final network state from Problem 4*ii*.

Problem 5 – Implementing the Perceptron (40 points)

In this part of HW4 you will implement the perceptron algorithm and run it on the WINE dataset. I.e., you will largely repeat Problem 1 of this HW, but in Java and using WINE. In addition, you will use the TUNE set examples to choose the network state that will be applied to the TEST examples (i.e., 'early stopping').

Here is what you need to do.

1) Copy the TRAIN set from

   http://pages.cs.wisc.edu/~shavlik/cs540/HWs/HW0/red-wine-quality-train.data

2) Copy the TUNE set from

   http://pages.cs.wisc.edu/~shavlik/cs540/HWs/HW0/red-wine-quality-tune.data .

3) Copy the TEST set from

   http://pages.cs.wisc.edu/~shavlik/cs540/HWs/HW0/red-wine-quality-test.data

4) Create a file HW4.java that takes as inputs these three strings:

   ```
   nameOfTrainSetFile  nameOfTuneSetFile  nameOfTestSetFile
   ```

5) Your code should create one input unit per binary feature, where 0 represents the first value of that feature and 1 represents the second value. (There will also be the '-1' input unit.) The output should be the class label, where 0 represents the first output class in the dataset files and 1 represents the second.

   Initialize all weights to 0. Use a learning rate (α) of 0.1.

   Recall that an *epoch* is one pass through each of the examples. Use the train set to adjust weight for 1000 epochs, where at the start of each epoch your code permutes the order the training examples are processed. (One way you can permute a list is to assign a random number to each element, in a new list of elements of a class that has two values: the original item and a double. Sort this new list, then create a new list of examples in that sorted order.)

   After every 50 epochs, measure the accuracy of the current perceptron on the train, tune, and test sets. Have your code print out these values, like this (these numbers are made up):

   ```
   Epoch   50:  train = 73.6%   tune = 67.5%   test = 65.8%
   Epoch  100:  train = 76.9%   tune = 71.3%   test = 70.4%
   Epoch  150:  train = 80.1%   tune = 72.2%   test = 72.7%
   …
   ```

   After 1000 epochs, before exiting your code should print out the epoch where the TUNE set was highest, and the TEST set accuracy at that epoch, like this (numbers also made up):

   ```
   The tune set was highest (74.0% accuracy) at Epoch 250.  Test set = 73.8% here.
   ```

Your code should also print, after learning is completed, the weights on each feature and the threshold, e.g. (again, numbers made up):

```
Wgt =  0.1 fixedAcidityGt47
Wgt = -2.4 volatileAcidityGt17
Wgt =  3.9 volatileAcidityGt29
…
Threshold = 2.8
```

6) Turn in a short lab report where you plot (by hand is ok) the train, tune, and test accuracies as a function of the epoch. (The lecture notes have a graph of this form, illustrating 'early stopping' as a method for reducing overfitting.) Briefly discuss your results. Do you see overfitting? Did the tune set do a reasonable job of choosing a good stopping point? Be sure to label your axes.

   Also compare the test set results for the perceptron to the ensemble of decision trees of HW2. You only need to discuss the TUNE and TEST set results at the chosen *L* (of HW2) and epoch (in this HW) – i.e., no need to compare the entire graphs in these two HWs.

   Finally, include in your lab report the weights on each feature plus the threshold. Highlight (i.e., mark somehow) which feature has the largest positive weight (if any), which has the largest negative weight (if any), and which feature weight is closest to zero. Since you are probably not an expert on wine acidity, etc., you do not need to discuss if these weights make sense.

Note that we are likely to test your code on datasets different than WINE, but all will only involve binary-valued features and output.