CS 273P: Machine Learning and Data Mining

Homework 4

Due date: **Tuesday, May 28, 2019**

Instructor: Xiaohui Xie

The submission for this homework, as for others so far, should be **one stand-alone PDF file** containing all of the relevant code, figures, and any text explaining your results. The code is not as important as your explanations of the trends and conclusions, so use the Markdown cells in Jupyter quite liberally.

Points: This homework adds up to a total of **100 points**, as follows:

Problem 1: Warm-up: Decision Tree	5 points
Problem 2: Setting up the data	10 points
Problem 3: Linear Classifiers	20 points
Problem 4: Nearest Neighbor	20 points
Problem 5: Decision Trees	20 points
Problem 6: Neural Networks	20 points
Statement of Collaboration	5 points

Be sure to re-download and replace the `mltools` package; it contains a number of new classifiers for you.

1 Warm-up: Decision Tree (5 points)

In order to reduce my email load, I decide to implement a machine learning algorithm to decide whether or not I should read an email, or simply file it away instead. To train my model, I obtain the following data set of binary-valued features about each email, including whether I know the author or not, whether the email is long or short, and whether it has any of several key words, along with my final decision about whether to read it ($y = +1$ for "read", $y = -1$ for "discard").

x_1 know author?	x_2 is long?	x_3 has 'research'	x_4 has 'grade'	x_5 has 'lottery'	y ⇒ read?
0	0	1	1	0	-1
1	1	0	1	0	-1
0	1	1	1	1	-1
1	1	1	1	0	-1
0	1	0	0	0	-1
1	0	1	1	1	1
0	0	1	0	0	1
1	0	0	0	0	1
1	0	1	1	0	1
1	1	1	1	1	-1

In the case of any ties, we will prefer to predict class $+1$.

1. Calculate the entropy of the class variable y
2. Calculate the information gain for each feature x_i. Which feature should I split on first?
3. Draw the complete decision tree that will be learned from these data.

2 Setting up the data (10 points)

The following is the snippet of code to load the datasets, and split it into train and validation data:

```python
# Data Loading
X = np.genfromtxt('data/X_train.txt', delimiter=None)
Y = np.genfromtxt('data/Y_train.txt', delimiter=None)
X,Y = ml.shuffleData(X,Y)
```

1. Print the minimum, maximum, mean, and the variance of all of the features. `5 points`
2. Split the dataset, and rescale each into training and validation, as:

   ```python
   Xtr, Xva, Ytr, Yva = ml.splitData(X, Y)
   Xt, Yt = Xtr[:5000], Ytr[:5000]   # subsample for efficiency (you can go higher)
   XtS, params  = ml.rescale(Xt)    # Normalize the features
   XvS, _  = ml.rescale(Xva, params) # Normalize the features

   ```

   Print the min, maximum, mean, and the variance of the rescaled features. `5 points`

3 Linear Classifiers (20 points)

In this problem, you will use an existing implementation of logistic regression, from the last homework, to analyze its performance on the Kaggle dataset.

```python
learner = mltools.linearC.linearClassify()
learner.train(XtS, Yt, reg=0.0, initStep=0.5, stopTol=1e-6, stopIter=100)
learner.auc(XtS, Yt) # train AUC
```

1. One of the important aspects of using linear classifiers is the regularization. Vary the amount of regularization, `reg`, in a wide enough range, and plot the training and validation AUC as the regularization weight is varied. Show the plot. `10 points`
2. We have also studied the use of polynomial features to make linear classifiers more complex. Add degree 2 polynomial features, print out the number of features, why it is what it is. `5 points`
3. Reuse your code that varied regularization to compute the training and validation performance (AUC) for this transformed data. Show the plot. `5 points`

4 Nearest Neighbors (20 points)

In this problem, you will analyze an existing implementation of K-Nearest-neighbor classification for the Kaggle dataset. The K-nearest neighbor classifier implementation supports two hyperparameters: the size of the neighborhood, K, and how much to weigh the distance to the point, a (0 means no unweighted average, and the higher the α, the higher the closer ones are weighted[1]). Note, you might have to subsample a lot for KNN to be efficient.

```python
learner = mltools.knn.knnClassify()
learner.train(XtS, Yt, K=1, alpha=0.0)
learner.auc(XtS, Yt) # train AUC
```

1. Plot of the training and validation performance for an appropriately wide range of K, with $\alpha = 0$. `5 points`
2. Do the same with unscaled/original data, and show the plots. `5 points`
3. Since we need to select both the value of K and α, we need to vary both, and see how the performance changes. For a range of both K and α, compute the training and validation AUC (for unscaled or scaled data, whichever you think would be a better choice), and plot them in a two dimensional plot like so:

[1]Specifically, weight of point x_i to the average is $\exp^{-\alpha||x-x_i||_2^2}$

```python
K = range(1,10,1) # Or something else
A = range(0,5,1) # Or something else
tr_auc = np.zeros((len(K),len(A)))
va_auc = np.zeros((len(K),len(A)))
for i,k in enumerate(K):
  for j,a in enumerate(A):
    tr_auc[i][j] = ... # train learner using k and a
    va_auc[i][j] = ...
# Now plot it
f, ax = plt.subplots(1, 1, figsize=(8, 5))
cax = ax.matshow(mat, interpolation='nearest')
f.colorbar(cax)
ax.set_xticklabels(['']+A)
ax.set_yticklabels(['']+K)
plt.show()

```

Show both the plots, and recommend a choice of K and α based on these results. `10 points`

5 Decision Trees (20 points)

For this problem, you will be using a similar analysis of hyper-parameters for the decision tree implementation. There are three hyper-parameters in this implementation that become relevant to its performance; `maxDepth`, `minParent`, and `minLeaf`, where the latter two specify the minimum number of data points necessary to split a node and form a node, respectively.

```python
learner = ml.dtree.treeClassify(Xt, Yt, maxDepth=15)
```

1. Keeping `minParent=2` and `minLeaf=1`, vary `maxDepth` to a range of your choosing, and plot the training and validation AUC. `5 points`

2. Plot the number of nodes in the tree as `maxDepth` is varied (using `learner.sz`). Plot another line in this plot by increasing either `minParent` or `minLeaf` (choose either, and by how much). `5 points`

3. Set `maxDepth` to a fixed value, and plot the training and validation performance of the other two hyper-parameters in an appropriate range, using the same 2D plot we used for nearest-neighbors. Show the plots, and recommend a choice for `minParent` and `minLeaf` based on these results. `10 points`

6 Neural Networks (20 points)

Last we will explore the use of neural networks for the same Kaggle dataset. The neural networks contain many possible hyper-parameters, such as the number of layers, the number of hidden nodes in each layer, the activation function the hidden units, etc. These don't even take into account the different hyper-parameters of the optimization algorithm.

```python
nn = ml.nnet.nnetClassify()
nn.init_weights([[XtS.shape[1], 5, 2], 'random', XtS, Yt) # as many layers nodes you want
nn.train(XtS, Yt, stopTol=1e-8, stepsize=.25, stopIter=300)
```

1. Vary the number of hidden layers and the nodes in each layer (we will assume each layer has the same number of nodes), and compute the training and validation performance. Show 2D plots, like for decision trees and K-NN classifiers, and recommend a network size based on the above.

2. Implement a new activation function of your choosing, and introduce it as below:

```python
def sig(z): return np.atleast_2d(z)
def dsig(z): return np.atleast_2d(1)
nn.setActivation('custom', sig, dsig)

```

Compare the performance of this activation function with `logistic` and `htangent`, in terms of the training and validation performance.

Statement of Collaboration (5 points)

It is **mandatory** to include a `Statement of Collaboration` in each submission, with respect to the guidelines below. Include the names of everyone involved in the discussions (especially in-person ones), and what was discussed.

All students are required to follow the academic honesty guidelines posted on the course website. For programming assignments, in particular, I encourage the students to organize (perhaps using Campuswire) to discuss the task descriptions, requirements, bugs in my code, and the relevant technical content `before` they start working on it. However, you should not discuss the specific solutions, and, as a guiding principle, you are not allowed to take anything written or drawn away from these discussions (i.e. no photographs of the blackboard, written notes, referring to Campuswire, etc.). Especially `after` you have started working on the assignment, try to restrict the discussion to Campuswire as much as possible, so that there is no doubt as to the extent of your collaboration.

Acknowledgements

This homework is adapted (with minor changes) from notes made available by Alex Ihler and Sameer Singh.