CS 178: Machine Learning: Winter 2018

Homework 2

Due Date: **Monday, October 22, 2018**

The submission for this homework should be ***a single PDF file*** containing all of the relevant code, figures, and any text explaining your results. When coding your answers, try to write functions to encapsulate and reuse code, instead of copying and pasting the same code multiple times. This will not only reduce your programming efforts, but also make it easier for us to understand and give credit for your work. **Write clearly and show all your work!**

Problem 1: Linear Regression (60 points)

For this problem we will fit linear regression models that minimize the *mean squared error* (MSE).

1. Load the "`data/curve80.txt`" data set, and split it into 75% / 25% training/test. The first column `data[:,0]` is the scalar feature value x; the second column `data[:,1]` is the target value y for each example. For consistency in our results, **do not** reorder (shuffle) the data (they're already in a random order), and use the first 75% of the data for training and the rest for testing:

```
1    X = data[:,0]
2    X = np.atleast_2d(X).T  # code expects shape (M,N) so make sure it's 2-dimensional
3    Y = data[:,1]
4    Xtr,Xte,Ytr,Yte = ml.splitData(X,Y,0.75)  # split data set 75/25
```

   Print the shapes of these four objects. *(5 points)*

2. Use the provided `linearRegress` class to create a linear regression predictor of y given x. You can plot the resulting function by simply evaluating the model at a large number of x values `xs`:

```
1    lr = ml.linear.linearRegress( Xtr, Ytr )  # create and train model
2    xs = np.linspace(0,10,200)      # densely sample possible x-values
3    xs = xs[:,np.newaxis]           # force "xs" to be an Mx1 matrix (expected by our code)
4    ys = lr.predict( xs )           # make predictions at xs
```

   (a) Plot the training data points along with your prediction function in a single plot. *(10 points)*
   (b) Print the linear regression coefficients (`lr.theta`) and verify that they match your plot. *(5 points)*
   (c) What is the mean squared error of the predictions on the training and test data? *(10 points)*

3. Try fitting $y = f(x)$ using a polynomial function $f(x)$ of increasing order. Do this by the trick of adding additional polynomial features before constructing and training the linear regression object. You can do this easily yourself; you can add a quadratic feature of Xtr with

```
1    Xtr2 = np.zeros( (Xtr.shape[0],2) )  # create Mx2 array to store features
2    Xtr2[:,0] = Xtr[:,0]        # place original "x" feature as X1
3    Xtr2[:,1] = Xtr[:,0]**2     # place "x^2" feature as X2
4    # Now, Xtr2 has two features about each data point: "x" and "x^2"
```

   (You can also add the all-ones constant feature in a similar way, but this is currently done automatically within the learner's train function.) A function "`ml.transforms.fpoly`" is also provided to more easily create such features. Note, though, that the resulting features may include extremely large values; if $x \approx 10$, then x^{10} is extremely large. For this reason (as is often the case with features on very different scales) it's a good idea to rescale the features; again, you can do this manually or use the provided `rescale` function:

```
1    # Create polynomial features up to "degree"; don't create constant feature
2    #   (the linear regression learner will add the constant feature automatically)
3    XtrP = ml.transforms.fpoly(Xtr, degree, bias=False)
4
```

```
5   # Rescale the data matrix so that the features have similar ranges / variance
6   XtrP,params = ml.transforms.rescale(XtrP)
7   #  "params" returns the transformation parameters (shift & scale)
8
9   # Then we can train the model on the scaled feature matrix:
10  lr = ml.linear.linearRegress( XtrP, Ytr )     # create and train model
11
12  # Now, apply the same polynomial expansion & scaling transformation to Xtest:
13  XteP,_ = ml.transforms.rescale( ml.transforms.fpoly(Xte,degree,false), params)
```

The transformations used to create features of the training data may depend on properties of that data (such as rescaling the data to have mean zero and variance one). For our learned predictions to be consistent, we need to apply the same transform to new test data, so that it will be represented consistently with the training data. "Feature transform" functions like `rescale` are written to output their parameters (here `params = (mu,sig)`, a tuple containing the mean and standard deviation used to shift and scale the data) so that they can be reused on subsequent data. When evaluating a polynomial regression model, be sure that the same rescaling and polynomial expansion is applied to both the training and test data.

Train polynomial regression models of degree $d = 1, 3, 5, 7, 10, 18$, and:
(a) For each model, plot the learned prediction function $f(x)$. *(15 points)*
(b) Plot the training and test errors on a log scale (`semilogy`) as a function of the model degree. *(10 points)*
(c) What polynomial degree do you recommend? *(5 points)*

When plotting prediction functions in part (a), you should set all plots to have the same vertical axis limits as the $d = 1$ regression model. Otherwise, high-degree polynomials may appear flat due to taking on extremely large values for a subset of inputs. Here is some example code:

```
1   fig, ax = plt.subplots(1, 1, figsize=(10, 8)) # Create axes for single subplot
2   ax.plot(...) # Plot polynomial regression of desired degree
3   ax.set_ylim(..., ...) # Set the minimum and maximum limits
4   plt.show()
```

Problem 2: Cross-validation (35 points)

In the previous problem, you decided what degree of polynomial fit to use based on performance on some test data. Now suppose that you do not have access to the target values of the test data you held out in the previous problem, and want to decide on the best polynomial degree.

One option would be to further split `Xtr` into training and validation datasets, and then assess performance on the validation data to choose the degree. But when training is reasonably efficient (or you have significant computational resources), it can be more effective to use cross-validation to estimate the optimal degree. Cross-validation works by creating many training/validation splits, called folds, and using all of these splits to assess the "out-of-sample" (validation) performance by averaging them. You can do a 5-fold validation test, for example, by:

```
1   nFolds = 5;
2   for iFold in range(nFolds):
3       Xti,Xvi,Yti,Yvi = ml.crossValidate(Xtr,Ytr,nFolds,iFold)  # use ith block as validation
4       learner = ml.linear.linearRegress(...   # TODO: train on Xti, Yti, the data for this fold
5       J[iFold] = ...                # TODO: now compute the MSE on Xvi, Yvi and save it
6   # the overall estimated validation error is the average of the error on each fold
7   print np.mean(J)
```

Using this technique on your training data `Xtr` from the previous problem, find the 5-fold cross-validation MSE of linear regression at the same degrees as before, $d = 1, 3, 5, 7, 10, 18$. To make your code more readable, write a function that takes the degree and number of folds as arguments, and returns the cross-validation error.

1. Plot the *five-fold* cross-validation error (with `semilogy`, as before) as a function of degree. *(10 points)*

2. How do the MSE estimates from five-fold cross-validation compare to the MSEs evaluated on the actual test data (Problem 1)? *(5 points)*

3. Which polynomial degree do you recommend based on five-fold cross-validation error? *(5 points)*

4. For the degree that you picked in step 3, plot (with `semilogy`) the cross-validation error as the number of folds is varied from nFolds $= 2, 3, 4, 5, 6, 10, 12, 15$. What pattern do you observe, and how do you explain why it occurs? *(15 points)*

Problem 3: Statement of Collaboration (5 points)

It is **mandatory** to include a *Statement of Collaboration* in each submission, that follows 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 students to organize (perhaps using Piazza) to discuss the task descriptions, requirements, possible bugs in the support 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 (no photographs of the blackboard, written notes, referring to Piazza, etc.). Especially *after* you have started working on the assignment, try to restrict the discussion to Piazza as much as possible, so that there is no doubt as to the extent of your collaboration.