CS 273A Homework 2

Machine Learning, CS 273A, Winter 2021

Due Date: 11:59pm Tuesday January 26, 2021 (Pacific Time)

The submission for this homework, as for the first one, should be **one stand-alone PDF file** containing all of the relevant code, figures, and any text explaining your results. When coding the answers, try to write functions to encapsulate and reuse the code, instead of copy pasting the same code multiple times. This will not only reduce your programming efforts, but also make it easier to debug, and for us to grade your work.

Write neatly (or type) and show all your work!

Points: This homework adds up to a total of **100 points + 10 extra points**, as follows:

Problem 1: Linear Regression	60 points + 10 extra points
Problem 2: Cross-Validation	35 points
Statement of Collaboration	5 points

Problem 1: Linear Regression (60 points + 10 extra points)

For this problem we will explore linear regression, the creation of additional features, and cross-validation.

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 (x) values; the second column `data[:,1]` is the target value y for each example. For consistency in our results, **don't** 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:

```
X = data[:,0]
X = np.atleast_2d(X).T  # code expects shape (M,N) so make sure it's 2-dimensional
Y = data[:,1]           # doesn't matter for Y
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`:

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

```
Xtr2 = np.zeros( (Xtr.shape[0],2) )  # create Mx2 array to store features
Xtr2[:,0] = Xtr[:,0]        # place original "x" feature as X1
Xtr2[:,1] = Xtr[:,0]**2     # place "x^2" feature as X2
# 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 e.g., 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 a provided `rescale` function:

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

# Rescale the data matrix so that the features have similar ranges / variance
XtrP,params = ml.transforms.rescale(XtrP)
#  "params" returns the transformation parameters (shift & scale)

# Then we can train the model on the scaled feature matrix:
lr = ml.linear.linearRegress( XtrP, Ytr )    # create and train model

# Now, apply the same polynomial expansion & scaling transformation to Xtest:
XteP,_ = ml.transforms.rescale( ml.transforms.fpoly(Xte,degree,false), params)
```

This snippet also shows a useful feature transformation framework – often we wish to apply some transformation to the features; in many cases the desired transformation depends on the data (such as rescaling the data to unit variance). Ideally, we should then be able to apply this same transform to new test data when it arrives, so that it will be treated in exactly the same way as the training data. "Feature transform" functions like rescale are written to output their settings, (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. You should create a function here that takes the degree as an argument, and returns the test set performance.

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

For (a), remember that your learner has now been trained on the polynomially expanded features, and so is expecting `degree` features (columns) to be input. So, don't forget to also expand and scale the features of `xs` using `fpoly` and `rescale`. You can do this manually as in the code snippet above, or you can think of this as a "feature transform" function Phi, eg.,

```
# Define a function "Phi(X)" which outputs the expanded and scaled feature
   matrix:
def Phi(X):
  return ml.transforms.rescale( ml.transforms.fpoly(X, degree,False), params)[0]
# the parameters "degree" and "params" are memorized at the function definition

# Now, Phi will do the required feature expansion and rescaling:
YhatTrain = lr.predict( Phi(Xtr) )    # predict on training data
YhatTest  = lr.predict( Phi(Xte) )    # predict on test data
# etc.
```

Also, you may want to save the original axes of your plot and re-apply them to each subsequent plot for consistency. (Otherwise, high-degree polynomials may look "flat" due to some extremely large values.) You can do this by, for example:

```
# Creating subplots with just one subplot so basically a single figure.
fig, ax = plt.subplots(1, 1, figsize=(10, 8))
ax.plot(...) # Plot for each polynomial degree
ax.plot(...) # like so
ax.set_ylim(..., ...) # Set the minimum and maximum limits
plt.show()
```

4. (Extra Credit, 10pts) Instead of expanding using polynomial features, try using Fourier features, i.e.,

```
XtrF = np.zeros( (Xtr.shape[0],5) ) # create Mx5 array to store features
XtrF[:,0] = Xtr[:,0] # place original "x" feature as X1
XtrF[:,1] = np.sin(Xtr[:,0]/2.) # place "sin(x)" feature as X2 (approx. scaled to
#  X's range)
XtrF[:,2] = np.cos(Xtr[:,0]/2.) # place "cos(x)" feature as X3
XtrF[:,3] = np.sin(Xtr[:,0]*2./2.) # place "sin(2*x)" feature as X4
XtrF[:,4] = np.cos(Xtr[:,0]*2./2.) # place "cos(2*x)" feature as X5
# Now, XtrF has five features about each data point: "x" and four Fourier
    features
```

Try expanding the number of Fourier features (sin(3x), etc.) and plot the training and validation curves for this feature set. Plot your results, and discuss.

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[1]. Let's now imagine that you did not have access to the target values of the test data you held out in the previous problem, and wanted to decide on the best polynomial degree.

Of course, we could simply repeat the exercise, further splitting `Xtr` into a training and validation split, and then assessing performance on the validation data to decide on a degree. But when training is reasonably fast, it can be more effective to use cross-validation to estimate the optimal degree. Cross-validation works by creating many such 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:

```
nFolds = 5;
for iFold in range(nFolds):
  Xti,Xvi,Yti,Yvi = ml.crossValidate(Xtr,Ytr,nFolds,iFold)  # use ith block as
   validation
  learner = ml.linear.linearRegress(...   # TODO: train on Xti, Yti, the data for
   this fold
  J[iFold] = ...                # TODO: now compute the MSE on Xvi, Yvi and save it
# the overall estimated validation error is the average of the error on each fold
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, $d = 1, 3, 5, 7, 10, 18$ (or more densely, if you prefer). Again, a function that has degree and number of folds as arguments, and returns cross-validation error, will be useful.

1. Plot the *five-fold* cross-validation error and test 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 the cross-validation error as the number of folds is varied (nFolds $= 2, 3, 4, 5, 6, 10, 12, 15$), again with `semilogy`. What pattern do you observe, and how do you explain it? *(15 points)*

[1] Technically, since you knew the answers to these data's targets and could use them to evaluate performance at different degrees, I would probably call them validation data instead.

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 one made available by Alex Ihler's machine learning course. Thanks, Alex!