Homework #1

MTH 9899 Baruch College
DATA SCIENCE II: Machine Learning

Due: April 10, 2019 - 18:00

Notes

- Code for this **MUST** be written in **Python 3.5+**.
- Do NOT use 3^{rd} Party Packages other than those listed and numpy, matplotlib, and pandas (if needed).
- Please include the output of 'pip freeze'.
- Please email your homework to: jayshree.pillai@gmail.com

Problem 1 (10 points) Ignoring more sophisticated algorithms, like the Strassen algorithm, multiplying an $a \times b$ matrix by a $b \times c$ matrix takes $\mathcal{O}(abc)$. Please work out the time complexity of computing a naive K-Fold Cross Validation Ridge Regression on an $N \times F$ input matrix.

Problem 2 (20 points) We can be more efficient. We don't have to compute $(XX^T)^{-1}$ completely each time. In particular, if you break up X into K chunks, there is a faster way.

$$X = \begin{bmatrix} X_1 \\ X_2 \\ \vdots \\ X_K \end{bmatrix}$$

$$X^T X = \begin{bmatrix} X_1^T & X_2^T & \dots & X_K^T \end{bmatrix} \begin{bmatrix} X_1 \\ X_2 \\ \vdots \\ X_K \end{bmatrix}$$

- Define X_{-i} as X with the ith fold omitted. Given these hints, write a description of how you can efficiently compute $X_{-i}^T X_{-i}$ for all K folds.

Data We will use this sample code to generate data for Problems 3 & 4. Please use EXACTLY this data.

```
import numpy as np

def generate_test_data_set1(n, test_sample = False):
    f = 5
    np.random.seed(1 if test_sample else 2)
    true_betas = np.random.randn(f)

    X = np.random.randn(n,f)
    Y = np.random.randn(n) + X.dot(true_betas)

    return (X,Y)

def generate_test_data_set2(n,test_sample = False):
    f = 5
    np.random.seed(1 if test_sample else 2)
    true_betas = np.random.randn(f)
    true_betas[-2:] = 0
    X = np.random.randn(n,f)
    Y = np.random.randn(n) + X.dot(true_betas)

    return (X,Y)
```

Problem 3 (35 points) Use sklearn's implementation of RidgeCV and LassoCV to fit data set 1 and 2 from above on 10,000 rows. If you run a simple LassoCV fit with the default parameters, it will try different values of α, which is their version of a regularization parameter. After it fits, you can see the alpha values it tried (x.alphas_) as well as the value of α that it chose (x.alpha_). Plot the out-of-sample error by generating a new test data set, also of size 10,000, for various values of α. Which data set leads a higher optimal value of α? Why?

Problem 4 (35 points) Implement a simple 1 hidden layer neural network in PyTorch. Let's use tanh as our activation function and a hidden layer of 10 neurons. Use this to fit the 2 data sets from Problem 3. Sample code to start you off is below. Talk about:

- Using the same test data sets as before, calculate the in-sample and out-of-sample error for the network.
- How does this compare to the error from the RidgeCV and LassoCV fits above?
- Just for the 1st data set, roughly how many epochs did it take to converge?
- Just for the 1st data set, roughly how many epochs does it take to converge if we change the activation function to ReLu?

```
## Adopted from https://pytorch.org/tutorials/beginner/pytorch_with_examples.html
import torch
import numpy as np
from gen_data import generate_test_data_set1
# F is num features; H is hidden dimension; out_size is # of output neurons
F, H, out_size = 5, 10, 1


xin, yin = generate_test_data_set1(10000)
x = torch.tensor(xin, dtype=torch.float32)
y = torch.tensor(yin[:,np.newaxis], dtype=torch.float32)

# Use the nn package to define our model and loss function.
model = torch.nn.Sequential(
          torch.nn.Linear(F, H),
```

```
        torch.nn.Tanh(),
        torch.nn.Linear(H, out_size),
    )
loss_fn = torch.nn.MSELoss()

# Use the optim package to define an Optimizer that will update the weights of
# the model for us. Here we will use Adam; the optim package contains many other
# optimization algoriths. The first argument to the Adam constructor tells the
# optimizer which Tensors it should update.
learning_rate = 5e-2
optimizer = torch.optim.Adam(model.parameters(), lr=learning_rate)

for t in range(1500):
  # Forward pass: compute predicted y by passing x to the model.
  y_pred = model(x)

  # Compute and print loss.
  loss = loss_fn(y_pred, y)
  print(t, loss.item())

  # Before the backward pass, use the optimizer object to zero all of the
  # gradients for the Tensors it will update (which are the learnable weights
  # of the model)
  optimizer.zero_grad()

  # Backward pass: compute gradient of the loss with respect to model parameters
  loss.backward()

  # Calling the step function on an Optimizer makes an update to its parameters
  optimizer.step()
```