Homework #1

MTH 9899 Baruch College
DATA SCIENCE II: Machine Learning

Due: April 5, 2017 - 18:00

Notes

- Code for this **MUST** be written in **Python 3.x**.
- Do NOT use 3^{rd} Party Packages for the regression functions.

Problem 1 In class, we spoke about the time complexity for multiplying matrices. Ignoring more sophisticated algorithms, like the Strassen algorithm, multiplying an $a \times b$ matrix by a $b \times c$ matrix takes $\mathcal{O}(abc)$. As we did in class, 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 We can be more efficient. In particular, 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.

$$\begin{aligned} X &= 2 \\ X &= \begin{bmatrix} X_1 \\ X_2 \\ \vdots \\ X_K \end{bmatrix} \\ X^T X &= \begin{bmatrix} X_1^T & X_2^T & \ldots & X_K^T \end{bmatrix} \begin{bmatrix} X_1 \\ X_2 \\ \vdots \\ X_K \end{bmatrix} \end{aligned}$$

- 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.

Problem 3 Implement the algoirthms discussed in a Jupyter Notebook.

- Using what you learned in Problem 2, implement 2 versions of Ridge Regression in the Python template shown below. One should be the slower naive algorithm, the other should be the faster version derived in Problem 2. Don't use any external math packages (other than NumPy).

- Test them. Generate random datasets with varying numbers of rows (anything from 1000 rows to 1,000,000) for 5 and 50 columns. Test both algos with 10 reasonable lambda values, and plot the time it takes to compute both versions as a function of N.

```
import numpy as np

def generate_test_data(n, f):
    np.random.seed(1)
    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 naive_ridge_cv(X, Y, num_folds, lambdas):
    """ Implements a naive (ie slow) Ridge Regression of X against Y.  It
    will take in a list of suggested lambda values and return back the
    lambda and betas that generates minimum mean squared error.

    Parameters
    ----------
    X : numpy ndarray
        The independent variables, structured as (samples x features)
    Y : numpy ndarray
        The dependent variable, (samples x 1)
    num_folds : int
        The number of folds to use for cross validation
    lambdas : numpy ndarray
        An array of lambda values to test

    Returns
    -------
    lambda_star : float
            The lambda value that represents the min MSE
    beta_star : numpy ndarray
            The optimal betas
    """





    return (lambdas[0], np.repeat(0, X.shape[1]))


def fast_ridge_cv(X, Y, num_folds, lambdas):
    """ Implements a fast Ridge Regression  of X against Y.  It
    will take in a list of suggested lambda values and return back the
    lambda and betas that generates minimum mean squared error.

    Parameters
    ----------
    X : numpy ndarray
        The independent variables, structured as (samples x features)
    Y : numpy ndarray
        The dependent variable, (samples x 1)
    num_folds : int
        The number of folds to use for cross validation
    lambdas : numpy ndarray
        An array of lambda values to test

    Returns
    -------
    lambda_star : float
```

```
        The lambda value that represents the min MSE
beta_star : numpy ndarray
        The optimal betas
"""





return (lambdas[0], np.repeat(0, X.shape[1]))
```