Data Mining and Machine Learning
Fall 2018, Homework 4
(due on Sep 30, 11.59pm EST)

Jean Honorio jhonorio@purdue.edu

The homework is based on a total of 10 points. Your code **should be in Python 2.7**. For clarity, the algorithms presented here will assume zero-based indices for arrays, vectors, matrices, etc. Please read the submission instructions at the end. **Failure to comply to the submission instructions will cause your grade to be reduced.**

In this homework, we will focus on linear regression. You can use the script **createlinregdata.py** to create some synthetic linear regression data:

```
import numpy as np
import numpy.linalg as la
# Input: number of samples n
#        number of features d
# Output: numpy matrix X of features, with n rows (samples), d columns (features)
#             X[i,j] is the j-th feature of the i-th sample
#         numpy vector y of scalar values, with n rows (samples), 1 column
#             y[i] is the scalar value of the i-th sample
# Example on how to call the function:
#     import createlinregdata
#     X, y = createlinregdata.run(10,2)
def run(n,d):
  w = 2*np.random.random((d,1))-1
  w = w/la.norm(w)
  X = np.random.normal(0.0,1.0,(n,d))
  y = np.dot(X,w) + 0.25*np.random.normal(0.0,1.0,(n,1))
  return (X, y)
```

Additionally, you will require a way to solve the linear regression problem, with training data $x_t \in \mathbb{R}^d$, $y_t \in \mathbb{R}$ for $t = 0, \dots, n-1$.

$$\widehat{\theta} \leftarrow \arg\min_{\beta \in \mathbb{R}^d} \frac{1}{2} \sum_{t=0}^{n-1} (y_t - \beta \cdot x_t)^2$$

If you assume that $n > d$, a solution to the above problem is given by the following script **linreg.py**:

```
import numpy as np
import numpy.linalg as la
# Input: numpy matrix X of features, with n rows (samples), d columns (features)
#             X[i,j] is the j-th feature of the i-th sample
#        numpy vector y of scalar values, with n rows (samples), 1 column
#             y[i] is the scalar value of the i-th sample
# Output: numpy vector theta, with d rows, 1 column
# Example on how to call the function:
#      import linreg
#      theta = linreg.run(X,y)
def run(X,y):
  return np.dot(la.pinv(X),y)
```

Here are the questions:

1) [3 points] Let S be a set of features. We define $x_{t,S}$ as a vector corresponding to taking the value of features in S of the t-th sample. (For instance, if $S = \{1, 4, 6\}$ and $t = 20$, then $x_{t,S}$ is a 3-dimensional vector containing the value of the features 1, 4 and 6, of sample 20.) Implement the following *greedy subset selection* algorithm, introduced in Lecture 7.

```
Input: number of features $F$, training data $x_t \in \mathbb{R}^d$, $y_t \in \mathbb{R}$ for $t = 0, \dots, n-1$
Output: feature set $S$, $\theta_S \in \mathbb{R}^F$
$S \leftarrow$ empty set
for $f = 1, \dots, F$ do
  for each $j \notin S$ do
    $\widehat{\theta}_{S \cup j} \leftarrow \arg\min_{\beta \in \mathbb{R}^f} \frac{1}{2} \sum_{t=0}^{n-1} (y_t - \beta \cdot x_{t,S \cup j})^2$
    $J(S \cup j) \leftarrow \frac{1}{2} \sum_{t=0}^{n-1} (y_t - \widehat{\theta}_{S \cup j} \cdot x_{t,S \cup j})^2$
  end for
  $\widehat{j} \leftarrow \arg\min_{j \notin S} J(S \cup j)$
  $S \leftarrow S \cup \{\widehat{j}\}$
end for
$\theta_S \leftarrow \widehat{\theta}_S$
```

The header of your **Python script greedysubset.py** should be:

```
# Input: number of features F
#        numpy matrix X of features, with n rows (samples), d columns (features)
#            X[i,j] is the j-th feature of the i-th sample
#        numpy vector y of scalar values, with n rows (samples), 1 column
#            y[i] is the scalar value of the i-th sample
# Output: numpy vector of selected features S, with F rows, 1 column
#         numpy vector thetaS, with F rows, 1 column
#             thetaS[0] corresponds to the weight of feature S[0]
#             thetaS[1] corresponds to the weight of feature S[1]
#             and so on and so forth
def run(F,X,y):
  # Your code goes here
  return (S, thetaS)
```

You can assume that $n > d > F$.

2) [3 points] Let S be a set of features. We define $x_{t,S}$ as a vector corresponding to taking the value of features in S of the t-th sample. (For instance, if $S = \{1, 4, 6\}$ and $t = 20$, then $x_{t,S}$ is a 3-dimensional vector containing the value of the features 1, 4 and 6, of sample 20.) Implement the following *forward fitting* algorithm, introduced in Lecture 7 (but here we do not allow for choosing the same feature more than once).

```
**Input:** number of features $F$, training data $x_t \in \mathbb{R}^d$, $y_t \in \mathbb{R}$ for $t = 0, \ldots, n-1$
**Output:** feature set $S$, $\theta_S \in \mathbb{R}^F$
$S \leftarrow$ empty set
$\widehat{\theta}_S \leftarrow$ empty array
**for** $f = 1, \ldots, F$ **do**
  **for** $t = 0, \ldots, n-1$ **do**
    $z_t \leftarrow y_t - \widehat{\theta}_S \cdot x_{t,S}$
  **end for**
  **for** each $j \notin S$ **do**
    $\widehat{\theta}_j \leftarrow \arg\min_{\beta \in \mathbb{R}} \frac{1}{2} \sum_{t=0}^{n-1} (z_t - \beta\, x_{t,j})^2$
    $J(\widehat{\theta}_S, j) \leftarrow \frac{1}{2} \sum_{t=0}^{n-1} (z_t - \widehat{\theta}_j\, x_{t,j})^2$
  **end for**
  $\widehat{j} \leftarrow \arg\min_{j \notin S} J(\widehat{\theta}_S, j)$
  $\widehat{\theta}_{S \cup j} \leftarrow (\widehat{\theta}_S, \widehat{\theta}_{\widehat{j}})$
  $S \leftarrow S \cup \{\widehat{j}\}$
**end for**
$\theta_S \leftarrow \widehat{\theta}_S$
```

The header of your **Python script forwardfitting.py** should be:

```
# Input: number of features F
#        numpy matrix X of features, with n rows (samples), d columns (features)
#            X[i,j] is the j-th feature of the i-th sample
#        numpy vector y of scalar values, with n rows (samples), 1 column
#            y[i] is the scalar value of the i-th sample
# Output: numpy vector of selected features S, with F rows, 1 column
#         numpy vector thetaS, with F rows, 1 column
#             thetaS[0] corresponds to the weight of feature S[0]
#             thetaS[1] corresponds to the weight of feature S[1]
#             and so on and so forth
def run(F,X,y):
  # Your code goes here
  return (S, thetaS)
```

You can assume that $n > d > F$.

3) [4 points] Let S be a set of features. We define $x_{t,S}$ as a vector corresponding to taking the value of features in S of the t-th sample. (For instance, if $S = \{1, 4, 6\}$ and $t = 20$, then $x_{t,S}$ is a 3-dimensional vector containing the value of the features 1, 4 and 6, of sample 20.) Implement the following *myopic forward fitting* algorithm, introduced in Lecture 7 (but here we do not allow for choosing the same feature more than once).

```
**Input:** number of features $F$, training data $x_t \in \mathbb{R}^d$, $y_t \in \mathbb{R}$ for $t = 0, \ldots, n-1$
**Output:** feature set $S$, $\theta_S \in \mathbb{R}^F$
$S \leftarrow$ empty set
$\widehat{\theta}_S \leftarrow$ empty array
**for** $f = 1, \ldots, F$ **do**
  **for** $t = 0, \ldots, n-1$ **do**
    $z_t \leftarrow y_t - \widehat{\theta}_S \cdot x_{t,S}$
  **end for**
  **for** each $j \notin S$ **do**
    $DJ(\widehat{\theta}_S, j) \leftarrow -\sum_{t=0}^{n-1} z_t \, x_{t,j}$
  **end for**
  $\widehat{j} \leftarrow \arg\max_{j \notin S} |DJ(\widehat{\theta}_S, j)|$
  $\widehat{\theta}_{\widehat{j}} \leftarrow \arg\min_{\beta \in \mathbb{R}} \frac{1}{2} \sum_{t=0}^{n-1} (z_t - \beta \, x_{t,\widehat{j}})^2$
  $\widehat{\theta}_{S \cup j} \leftarrow (\widehat{\theta}_S, \widehat{\theta}_{\widehat{j}})$
  $S \leftarrow S \cup \{\widehat{j}\}$
**end for**
$\theta_S \leftarrow \widehat{\theta}_S$
```

The header of your **Python script myopicfitting.py** should be:

```
# Input: number of features F
#        numpy matrix X of features, with n rows (samples), d columns (features)
#            X[i,j] is the j-th feature of the i-th sample
#        numpy vector y of scalar values, with n rows (samples), 1 column
#            y[i] is the scalar value of the i-th sample
# Output: numpy vector of selected features S, with F rows, 1 column
#         numpy vector thetaS, with F rows, 1 column
#             thetaS[0] corresponds to the weight of feature S[0]
#             thetaS[1] corresponds to the weight of feature S[1]
#             and so on and so forth
def run(F,X,y):
  # Your code goes here
  return (S, thetaS)
```

You can assume that $n > d > F$.

SOME POSSIBLY USEFUL THINGS.
Python 2.7 is available at the servers antor and data. From the terminal, you can use your Career account to start a ssh session:

```
  ssh username@data.cs.purdue.edu
  OR
  ssh username@antor.cs.purdue.edu
```

From the terminal, to start Python:

```
  python
```

Inside Python, to check whether you have **Python 2.7**:

```
  import sys
  print (sys.version)
```

Inside Python, to check whether you have the package **cvxopt**:

```
  import cvxopt
```

From the terminal, to install the Python package **cvxopt**:

```
  pip install --user cvxopt
  OR
  python -m pip install --user cvxopt
```

More information at `https://cvxopt.org/install/index.html`

SUBMISSION INSTRUCTIONS.
Your code **should be in Python 2.7**. We **only need** the Python scripts (.py files). We **do not need** the Python (compiled) bytecodes (.pyc files). You will get 0 points if your code does not run. You will get 0 points in you fail to include the Python scripts (.py files) even if you mistakingly include the bytecodes (.pyc files). We will deduct points, if you do not use the right name for the Python scripts (.py) as described on each question, or if the input/output matrices/vectors/scalars have a different type/size from what is described on each question. Homeworks are to be solved individually. We will run plagiarism detection software.
Please, submit a single ZIP file **through Blackboard**. Your Python scripts (**greedysubset.py**, **forwardfitting.py**, **myopicfitting.py**) should be directly inside the ZIP file. **There should not be any folder inside the ZIP file**, just Python scripts. The ZIP file should be named according to your Career account. For instance, if my Career account is jhonorio, the ZIP file should be named **jhonorio.zip**