15780: GRADUATE AI (SPRING 2019)

Homework 2: Machine Learning and Integer Programming

Release: February 14, 2019,
Due: February 28, 2019, 5:00pm

We will use scientific Python for the implementation portions in this course. If you have not used Python before, we recommend downloading the Anaconda distribution (`https://www.continuum.io/downloads`) and looking through introductory resources like Google's Python Class (`https://developers.google.com/edu/python/`) and the Python Beginner's Guide (`https://wiki.python.org/moin/BeginnersGuide`). If you have not used scientific Python before, we recommend following introductions to NumPy (`http://www.numpy.org/`) and matplotlib (`http://matplotlib.org/`).

Please make sure your code submission works with the grading environment, which uses Python 3.7, numpy 1.15.0, and cvxpy 1.0.14 (optional).

1 Linear Regression (Ivan) [20 points]

1.1 What do coefficients mean?

Assume you want to build a model to predict a daily energy consumption at CMU campus, using average daytime temperature in Celsius (`temp_cels`) and number of classes taught on campus that day (`classes`) as features. You have data for every day of the previous year and you decide to fit a linear model using least squares. The resulting model is:

$$\texttt{energy} = \theta_0 + \theta_1 \texttt{temp\_cels} + \theta_2 \texttt{classes}.$$

- **[2 points]** What is the meaning of θ_0 coefficient?
- **[3 points]** What is the meaning of θ_1 and θ_2 coefficients?

  Hint Think about the change in your prediction if you vary one feature keeping another fixed.
- **[3 points]** Now assume that instead of temperature in Celsius you have temperature in Fahrenheit (`temp_fahr`). You refit your model using the data for exactly the same period (the only difference is

that temperature is in Farenheit now) and the new model is

$$\texttt{energy} = \beta_0 + \beta_1 \texttt{temp\_fahr} + \beta_2 \texttt{classes}.$$

What is the relationship between coefficients of old and new models? Express β_0, β_1 and β_2 as some combinations of θ_0, θ_1 and θ_2.

Hint Recall that $\texttt{temp\_fahr} = 9/5 \cdot \texttt{temp\_cels} + 32$.

1.2 Least Squares in High Dimension

Definition 1. The pseudoinverse of X (denoted by X^+) is a matrix satisfying the following properties:

1. $XX^+X = X$
2. $X^+XX^+ = X^+$
3. $(XX^+)^T = XX^+$
4. $(X^+X)^T = X^+X$.

In some applications (i.e. medicine), the number of features n that are available to researcher might be very large, sometimes even larger than the number of points in the dataset m.

In class, we saw that the closed form solution of the least squares regression problem is given as

$$\theta^\star = \left(X^T X\right)^{-1} X^T y.$$

Unfortunately, this solution does not work in case $n > m$, because matrix $X^T X$ is not invertible. However, we can always define a solution in terms of the pseudoinverse X^+ of the matrix X.

1. **[4 points]** Show that
$$\theta^\star = X^+ y$$
is always the minimizer of the least squares regression problem.
2. **[4 points]** The above trick with pseudoinverse has one drawback. Show that if matrix $X^T X$ is not invertible, then there are infinitely many parameter vectors θ that minimize the least squares loss.

   Hint Recall that if a matrix A is not invertible, then there exists some vector $\delta \neq \mathbf{0}$ such that $A\delta = \mathbf{0}$.

Comment Notice that while pseudoinverse matrix does allow us to find a solution that minimizes least squares loss, this solution us not unique. In contrast, there exist infinitely many other solutions that also minimize least squares. While they do not make a difference for least squares loss, predictions for new data can be significantly different depending on what minimizer we choose.

1.3 Overcoming the issue

One way to survive in case $n > m$ is to add some side information to the estimation procedure. One example is regularized least squares with L_2 penalty when we explicitly penalize high values of parameter. Recall

that regularized least squares can be written as the following optimization problem:

$$\underset{\theta}{\text{minimize}}\ \frac{1}{2}\|X\theta - y\|_2^2 + \frac{\lambda}{2}\|\theta\|_2^2.$$

1. **[4 points]** Show that even if matrix $X^T X$ is not invertible, the regularized least squares estimator with $\lambda > 0$ still has a unique solution.

   Hints You may want to use the fact that if for a square symmetric matrix A we have $\delta^T A\delta > 0$ for any $\delta \neq \mathbf{0}$, then the matrix A is invertible.

2 Integer Programming (Ivan) [30 points]

In class we have shown that the problem of assigning papers to referees in peer review can be written as an Integer Programming (IP) problem. Recall that we have d papers $\{1, 2, \ldots, d\} = [d]$, n reviewers $\{1, 2, \ldots, n\} = [n]$ and similarity matrix $S \in [0, 1]^{d\times n}$ where each component S_{ij} represents a similarity between paper i and reviewer j. The goal is to assign papers to reviewers such that the sum of similarities of all assigned (paper, reviewer) pairs is maximized subject to the following constraints:

- **Paper load constraints.** Each paper gets *at least* $\lambda \in \mathbb{N}$ reviewers
- **Reviewer load constraints.** Each reviewer gets *at most* $\mu \in \mathbb{N}$ papers

To cast this problem as an IP problem, we introduce a binary matrix $X \in \{0, 1\}^{d\times n}$ that can represent any assignment of d papers to n reviewers: component X_{ij} equals 1 if and only if paper i is assigned to reviewer j. In this notation, the assignment problem corresponds to the following IP:

$$\begin{aligned}
\underset{X\in\mathbb{Z}^{d\times n}}{\text{maximize}} \quad & \sum_{i\in[d]}\sum_{j\in[n]} S_{ij}X_{ij} \\
\text{subject to} \quad & \sum_{i\in[d]} X_{ij} \leq \mu \quad \forall j \\
& \sum_{j\in[n]} X_{ij} \geq \lambda \quad \forall i \\
& X_{ij} \in \{0, 1\} \quad \forall i, j
\end{aligned}$$

The size of modern conferences is huge (NeurIPS received more that 5000 submissions last year) and general IP problems of that scale are prohibitively large even for state of the art solvers. Fortunately, the IP problem we formulated features some special structure which allows us to relax it (allow X to be real rather than integer), apply LP solver and be sure that resulted solution will be integer! In this problem we will formally prove this claim.

Definition 2. A matrix is called a Totally Unimodular Matrix (TUM) if every square submatrix has a determinant $-1, 0$, or 1.

1. **[5 points]** Rewrite the IP above as an IP of the following form:

$$\begin{aligned}
\underset{x\in\mathbb{Z}^{\mathbf{dim}}}{\text{maximize}} \quad & \langle c, x\rangle \\
\text{subject to} \quad & Ax \leq b,
\end{aligned}$$

where x is a vector that is constructed from matrix X by concatenating its rows. Specify A, c, b and **dim**.

Hint: Make sure you account for all the constraints in original LP formulation, including $X_{ij} \in \{0, 1\}$.

2. **[20 points]** Prove that matrix A is a Totally Unimodular Matrix (TUM).

   Here is a suggested path you can take:

   Sketch of the proof:

   - First, use the fact that if a matrix $M' \in \{-1, 0, 1\}^{k_1 \times k_2}$ is TUM, then matrix M defined by

   $$M = \left[\frac{M'}{\pm I}\right],$$

   where I is a square identity matrix of size k_2, is also TUM.

   Now look at your matrix A. Do you see that it contains I and $-I$? If so, then apply the above fact and now look only at the corresponding submatrix A'.

   - Next, proceed by induction:

   Base of induction

   Check that for every 1x1 submatrix of A' TUM condition holds.

   Inductive step

   Make inductive assumption that for every square submatrix of size k TUM condition holds. Now you need to show that it holds for each square submatrix of size $k+1$. To this end, consider arbitrary square submatrix M of size $k+1$. Notice that there are several possible options:

   – *Case 1:* Some column of M is all 0s. This case is easy, right?

   – *Case 2:* Some column of M contains single 1 and all other entries of this column are 0s. Notice that in this case rows of M can be permuted (recall that permutation preserves determinant) and we can write $M^{\text{permuted}} = \left[\begin{array}{c|c} 1 & \mathbf{a} \\ \hline \mathbf{0} & M' \end{array}\right]$, where $\mathbf{a}$ and M' are some submatrices. What is the size of M'? Can you apply inductive assumption in this case?

   – *Case 3:* Every column of M contains exactly two 1s. In this case, sum rows of matrix M. Do you observe something that helps you to compute the determinant. Try to show that rows are linearly dependent in this case. Recall that if rows of matrix are linearly dependent, then its determinant equals 0.

   Important: You should formally prove that only these 3 cases are possible for the given problem.

   Clarification:[1] In your proof you may end up having matrix with both positive and negative entries. In this case, you can still use the suggestions for inductive step, but instead of 1s in your case you might have ± 1s. For example, *Case 2* can be read as "Some column of M contains single non-zero entry and all other entries of this column are 0s."

[1] Clarification added

3. **[5 points]** Apply theorem from the lecture (see slides of `IP II: Applications` lecture) and write down the LP problem that is equivalent to the original IP problem. Conclude that the problem of assigning papers to referees can be efficiently solved by simplex algorithm.

3 Classification Programming (Junjue) [20 points]

In this section, you will develop a multi-class classifier to classify digits from the MNIST data set. We will extend the notation used in class a bit, and use a loss function that *directly* captures a k-class classification tasks, called the softmax or cross-entropy loss. In our new setting, we have a training set of the form $(x^{(i)}, y^{(i)})$, $i = 1, \ldots, m$, with $y^{(i)} \in \{0, 1\}^k$ (remember that k is the number of classes we're trying to predict), where $y_j^{(i)} = 1$ when j is the target class, and 0 otherwise. That is, if output values can take on one of 10 classes (as will be the case in the digit classification task), and the target class for this example is class 4, then corresponding $y^{(i)}$ is simply

$$y^{(i)} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}.$$

This is sometimes called a "one-hot" encoding of the output class.

Under the model, our hypothesis function $\hat{y} = h_\theta(x)$ will now output *vectors* in $\mathbb{R}^k$, where the relative size of the $\hat{y}_j$ corresponds roughly to how likely we believe that the output is really class j (this will become more concrete when we formally define the class function). For instance, the (hypothetical) output

$$\hat{y} = h_\theta(x^{(i)}) = \begin{bmatrix} 0.1 \\ -0.2 \\ 2.0 \\ 5.0 \\ 0.1 \\ -1.0 \\ -5.0 \\ 1.0 \\ 0.4 \\ 0.2 \end{bmatrix}$$

would correspond to a predict that the example $x^{(i)}$ is probably from class 4 (the element with the largest entry). Analogous to binary classification (where, if we wanted binary prediction we would simply take the sign of the hypothesis function), if we want the to predict a single class label for the output, we simply predict class class j for which $\hat{y}_j$ takes on the largest value.

Our loss function is now defined as a function $\ell : \mathbb{R}^k \times \{0,1\}^k \to \mathbb{R}_+$ that quantifies how good a prediction is. In the "best" case, the predictions $\hat{y}_j$ would be $+\infty$ for the true class (i.e. for the element where $y_j^{(i)} = 1$) and $-\infty$ otherwise (of course, we usually won't make infinite predictions, because we would then suffer very high loss if we ever made a mistake, and we won't get such predictions with most classifiers if we include a regularization term).

In below, you'll implement a linear classification models to classify digits. That is, our hypothesis function will be

$$h_\theta(x^{(i)}) = \Theta \begin{bmatrix} x^{(i)} \\ 1 \end{bmatrix}$$

where $\Theta \in \mathbb{R}^{10 \times 785}$ is our vector of parameters. Using a simple application of the chain rule (similar to that applied to derive backpropagation), we can compute the gradient of our loss function for this hypothesis class

$$\nabla_\Theta \ell(h_\theta(x^{(i)}), y) = \nabla_{\hat{y}} \ell(\hat{y}, y) \begin{bmatrix} {x^{(i)}}^T & 1 \end{bmatrix}.$$

where $\hat{y} \equiv h_\theta(x^{(i)})$ and where the gradient $\nabla_{\hat{y}} \ell(\hat{y}, y)$ will be given.

3.1 Softmax Classifier with Cross-Entropy Loss

The classifier you are going to implement is a Softmax Classifier. A softmax classifier interprets the output of $h_\theta(x)$ as unnormalized log probabilities for each class. The typical loss function of a Softmax classifier is cross-entropy loss, given by[2]

$$\ell(\hat{y}, y) = \log \left(\sum_{j=1}^{k} e^{\hat{y}_j} \right) - \hat{y}^T y.$$

This loss function has the gradient

$$\nabla_{\hat{y}} \ell(\hat{y}, y) = \frac{e^{\hat{y}}}{\sum_{j=1}^{k} e^{\hat{y}_j}} - y \tag{1}$$

where the exponent $e^{\hat{y}}$ is taken elementwise.

In practice, you would probably want to implemented regularized loss minimization, but for the sake of this problem set, we'll just consider minimizing loss without any regularization (at the expense of overfitting a little bit).

[2] It won't be relevant to the implementation in this problem, but for those curious where this loss function comes from, softmax regression can be interpreted as a probabilistic model where

$$p(y = j | \hat{y}) = \frac{e^{\hat{y}_j}}{\sum_{l=1}^{k} e^{\hat{y}_l}}.$$

If we look at maximum likelihood estimation under this true model, we get the optimization problem

$$\text{minimize}_\theta - \sum_{i=1}^{m} \log p(y^{(i)} | x^{(i)}) \;\equiv\; \text{minimize}_\theta \sum_{i=1}^{m} \log \left(\sum_{j=1}^{k} e^{h_\theta(x^{(i)})_j} \right) - h_\theta(x^{(i)})^T y^{(i)}.$$

3.1.1 Gradient descent (10 points)

Implement the gradient descent algorithm to solve this optimization problem. Recall from the slides that the gradient descent algorithm is given by:

function $\theta = \text{Gradient-Descent}(\{(x^{(i)}, y^{(i)})\}, h_\theta, \ell, \alpha)$
  Initialize: $\theta \leftarrow 0$
  For $t = 1, \ldots, T$:
    $g \leftarrow 0$
    For $i = 1, \ldots, m$:
      $g \leftarrow g + \frac{1}{m} \nabla_\theta \ell(h_\theta(x^{(i)}), y^{(i)})$
    $\theta \leftarrow \theta - \alpha g$
  return θ

i.e., we compute the gradient with respect to the loss function for all the examples, divide it by the total number of examples, and take a small step in this direction (you can leave all the step sizes α at their default values for the programming part). Each pass over the whole dataset is referred to as an *epoch*.

Specifically, you'll implement the `softmax_gd` function:

```
def softmax_gd(X, y, Xt, yt, epochs=10, alpha = 0.5):
"""
Run gradient descent to solve linear softmax regression.

Inputs:
X: numpy array of training inputs
y: numpy array of training outputs
Xt: numpy array of testing inputs
yt: numpy array of testing outputs
epochs: number of passes to make over the whole training set
alpha: step size

Outputs:
Theta: 10 x 785 numpy array of trained weights
"""
```

In addition to outputing the trained parameters, your function should print the error (computed by the included `error` function) on the test and training sets, and the softmax loss averaged on the test and training sets (again using the provided function, which will help you in computing the gradient). In your PDF, you should report all these values computed at each iteration (epoch) *before* adjusting the parameters for that iteration. All the four values are averaged over the train and test sets and must be reported in the PDF. For example, our implementation of gradient descent outputs the following:

```
Test Err  |Train Err |Test Loss |Train Loss
0.9020|    0.9013|    2.3026|    2.3026
0.3192|    0.3276|    1.8234|    1.8318
0.2101|    0.2228|    1.4983|    1.5141
```

```
0.2142|    0.2246|    1.2830|    1.3018
0.1844|    0.1935|    1.1341|    1.1555
0.1816|    0.1924|    1.0260|    1.0490
0.1702|    0.1785|    0.9463|    0.9697
0.1670|    0.1754|    0.8826|    0.9072
0.1613|    0.1681|    0.8334|    0.8576
0.1559|    0.1653|    0.7909|    0.8160
```

3.1.2 Stochastic gradient descent (10 points)

Implement the stochastic gradient descent algorithm for linear softmax regression. Recall from the notes that the SGD algorithm is:

function $\theta = \text{SGD}(\{(x^{(i)}, y^{(i)})\}, h_\theta, \ell, \alpha)$
  Initialize: $\theta \leftarrow 0$
  For $t = 1, \ldots, T$:
    For $i = 1, \ldots, m$:
      $\theta \leftarrow \theta - \alpha \nabla_\theta \ell(h_\theta(x^{(i)}), y^{(i)})$
  return θ

That is, we take small gradient steps on *each* example, rather than computing the average gradient over all the examples.

You'll implement the following function:

```
def softmax_sgd(X, y, Xt, yt, epochs=10, alpha = 0.5):
"""
Run stochastic gradient descent to solve linear softmax
regression.

Inputs:
X: numpy array of training inputs
y: numpy array of training outputs
Xt: numpy array of testing inputs
yt: numpy array of testing outputs
epochs: number of passes to make over the whole training set
alpha: step size

Outputs:
Theta: 10 x 785 numpy array of trained weights
"""
```

You should print the same information (train/test error/loss) as for the above function, but importantly only print this information *once* per outer iteration, before you iterate over all the examples. Report these values in your PDF.

4 Integer Programming (Chun Kai) [30 points]

In this task you will develop an integer programming algorithm, based upon branch and bound, to solve Sudoku puzzles. Specifically, implement the function `solve_sudoku` in `sudoku.py`.

The input of the function is a Sudoku puzzle, represented as a 9×9 "list of lists" of integers, e.g.,

```
puzzle = [[4, 8, 0, 3, 0, 0, 0, 0, 0],
          [0, 0, 0, 0, 0, 0, 0, 7, 1],
          [0, 2, 0, 0, 0, 0, 0, 0, 0],
          [7, 0, 5, 0, 0, 0, 0, 6, 0],
          [0, 0, 0, 2, 0, 0, 8, 0, 0],
          [0, 0, 0, 0, 0, 0, 0, 0, 0],
          [0, 0, 1, 0, 7, 6, 0, 0, 0],
          [3, 0, 0, 0, 0, 0, 4, 0, 0],
          [0, 0, 0, 0, 5, 0, 0, 0, 0]],
```

where zero entries indicate missing entries that should be assigned by your algorithm, and all other positive integers (between 1 and 9) indicate known assignments. The function `solve_sudoku` should return a tuple `(solved_puzzle, constraints)`, where `solved_puzzle` is the input puzzle with all the missing entries assigned to their correct values. For instance, for the above puzzle, `solved_puzzle` should be

```
solved_puzzle = [[4, 8, 7, 3, 1, 2, 6, 9, 5],
                 [5, 9, 3, 6, 8, 4, 2, 7, 1],
                 [1, 2, 6, 5, 9, 7, 3, 8, 4],
                 [7, 3, 5, 8, 4, 9, 1, 6, 2],
                 [9, 1, 4, 2, 6, 5, 8, 3, 7],
                 [2, 6, 8, 7, 3, 1, 5, 4, 9],
                 [8, 5, 1, 4, 7, 6, 9, 2, 3],
                 [3, 7, 9, 1, 2, 8, 4, 5, 6],
                 [6, 4, 2, 9, 5, 3, 7, 1, 8]]
```

and the `constraints` value could be `constraints = [(8, 5, 2, 1), (5, 3, 4, 1)]`.

This implies that if we add two constraints $(x_{8,5})_2 = 1$ and $(x_{5,3})_4 = 1$ to the linear program for Sudoku (described in detail below) then the solution has only integer values and returns the solution above. Note that, since these values are all indexed starting at 1, it may be necessary to add/subtract 1 to convert to and from Python indices. Also note that there can be more than one set of constraints that lead to the same integer solution. Thus, we will check your solution by directly plugging the constraints your algorithm returns in, not by comparing with our solution. Specifically, you need to do two things to solve this problem:

1. Write code to solve the linear programming relaxation of a Sudoku puzzle. Let $x_{i,j} \in [0,1]^9$ be the indicator of the (i,j)-th square in a Sudoku board. Then, solve the following optimization problem:

   minimize $\sum_{i,j}^{9} \max_k (x_{i,j})_k$

   subject to $x_{i,j} \in [0,1]^9 \;\; i,j = 1...9$ (i.e., all variables must be between zero and one)

$$\sum_{k=1}^{9}(x_{i,j})_k = 1, \;\; i,j = 1...9 \qquad \text{(i.e., each grid must have only one assigned number)}$$

$$\sum_{j=1}^{9} x_{i,j} = \mathbf{1}, \;\; i = 1...9 \qquad \text{(i.e., each row must contain each number)}$$

$$\sum_{i=1}^{9} x_{i,j} = \mathbf{1}, \;\; j = 1...9 \qquad \text{(i.e., each column must contain each number)}$$

$$\sum_{m,l=1}^{3} x_{i+m,j+l} = \mathbf{1}, \;\; i,j \in \{0,3,6\} \qquad \text{(i.e., each } 3 \times 3 \text{ box must contain each number)}$$

$$(x_{ij})_k = 1 \text{ if } (i,j)\text{-th square is } k \geq 1 \qquad \text{(i.e., assignments of the entries fixed by the puzzle).}$$

You should write code to solve this problem using `cvxpy`. You can find additional documentation about `cvxpy` at its website `http://cvxpy.org`. Installation instructions are included at the end of the question. Please read the instructions if you decide to use `cvxpy`.

You can also try to use the two-phase simplex algorithm you implemented instead of using `cvxpy`; however, in that case you will have to deal with the A matrix possibly not having full row rank (which requires some additional steps of removing linearly dependent rows of the matrix), possibly deal with degenerate initial solutions, and encode the $\max$ function above using linear equalities in standard form. To test this part of the assignment, you can use the following puzzle, where the linear programming relaxation happens to give an integer solution without any additional constraints:

```
puzzle = [[8, 5, 0, 0, 0, 2, 4, 0, 0],
          [7, 2, 0, 0, 0, 0, 0, 0, 9],
          [0, 0, 4, 0, 0, 0, 0, 0, 0],
          [0, 0, 0, 1, 0, 7, 0, 0, 2],
          [3, 0, 5, 0, 0, 0, 9, 0, 0],
          [0, 4, 0, 0, 0, 0, 0, 0, 0],
          [0, 0, 0, 0, 8, 0, 0, 7, 0],
          [0, 1, 7, 0, 0, 0, 0, 0, 0],
          [0, 0, 0, 0, 3, 6, 0, 4, 0]],
```

whose linear programming relaxation should give the following solution:

```
solved_puzzle = [[8, 5, 9, 6, 1, 2, 4, 3, 7],
                 [7, 2, 3, 8, 5, 4, 1, 6, 9],
                 [1, 6, 4, 3, 7, 9, 5, 2, 8],
                 [9, 8, 6, 1, 4, 7, 3, 5, 2],
                 [3, 7, 5, 2, 6, 8, 9, 1, 4],
                 [2, 4, 1, 5, 9, 3, 7, 8, 6],
                 [4, 3, 2, 9, 8, 1, 6, 7, 5],
                 [6, 1, 7, 4, 2, 5, 8, 9, 3],
                 [5, 9, 8, 7, 3, 6, 2, 4, 1]].
```

Important: The results of simplex may not be exactly integers because of numerical approximation. For this part of the assignment, you can simply round any value within 0.005 of 0 or 1 to the nearest integer.

2. Next, write a branch and bound algorithm solving a Sudoku puzzle even when its linear programming relaxation is not tight. That is, implement the algorithm in the integer programming slides (implement the "simpler" algorithm instead of the version generating feasible upper bounds). To select variables to split on (in this case, $\{(x_{(i,j)})_k : i, j, k = 1, ..., 9\}$), a simple rule is to pick the variable with the "most undetermined" value closest to 0.5 and split on this variable. Then, solve the two subproblems where we constrain the variable to be either 0 or 1.

Grading. There will be 10 test cases. Each test is weighted according to its difficulty. You are given 60 seconds to complete each test. This should be sufficient for the purpose of solving sudoku.

Installing `cvxpy`

We will be using `cvxpy` version 1.0.14 for grading. Different versions of `cvxpy` may have different function signatures and cause errors. Installation may be done using

```
$ conda install -c conda-forge cvxpy=1.0.14
```

or if using pip,

```
$ pip install cvxpy==1.0.14
```

We suggest running the tests recommended.

```
$ conda install nose
$ nosetests cvxpy
```

Important: It is possible that OSQP, the default solver for `cvxpy` fails to solve your linear program. We suggest using the ECOS solver instead.

```
# Use this,
prob.solve(solver='ECOS')
# instead of this.
prob.solve()
```

5 Submitting to Diderot

Create a tar file containing your writeup for the first two problems and the completed `cls.py` and `sudoku.py` modules for the programming problems. Make sure that your tar has these files at the root and not in a subdirectory. Use the following commands from a directory with your files to create a `handin.tgz` file for submission.

```
$ ls
cls.py  sudoku.py  writeup.pdf
```

```
$ tar cvzf handin.tgz writeup.pdf sudoku.py cls.py
a writeup.pdf
a sudoku.py
a cls.py
$ ls
cls.py  handin.tgz  sudoku.py  writeup.pdf
```