15-780: Problem Set #2

February 19, 2014

1. **Constraint satisfaction problem (CSP) [20pts]** A common problem at universities is to schedule rooms for exams. The basic structure of this problem is to divide the exam period into predefined time slots and then assign courses to the available rooms at particular time slots. The assignments must be consistent with a set of constraints. For this question, you will describe how you would design various elements of a CSP program that would solve a restricted version of this problem.

   For this version of the problem, a course is defined as having a name, instructor(s) and students. A room has a name and a capacity. Time slots are positive integers, and we will assume that there are three time slots that are represented as 0, 1, and 2. The constraints that must be observed are the following:

   - All courses must be assigned to a room and time slot.
   - No assigned room can have less capacity than the number of students in the assigned course.
   - No room can be double-booked with more than one course per time slot.
   - No instructor or student can be in two different rooms during the same time slot.

   Using the backtracking-search algorithm presented in class as a framework for your CSP solver, provide concise answers to the following:

   (a) Describe the basic data structures you would use to represent the domain objects and the constraints.

   (b) What data structure would you use as a variable and what would you use as a value?

   (c) Describe a variable-ordering heuristic that could improve performance. This heuristic would be used in SELECT-UNASSIGNED-VARIABLE.

   (d) Describe how you would compute the possible values for a variable in ORDER-DOMAIN-VALUES.

   (e) Given a variable and a value assignment, describe how you would check that the assignment is consistent with the constraints. Note that this question may be straightforward, depending on how you generate the values.

(f) How many representations of state do you need in your CSP solver? Remember that backtracking-search is a depth-first search that undoes its assignments when back tracking.

2. **Minimum error classification as mixed integer programming [10 pts]** In class, we mentioned that we cannot efficiently find a linear classifier that minimizes 0/1 loss. That is, we cannot easily solve the optimization problem

$$\underset{\theta}{\text{minimize}} \sum_{i=1}^{m} \mathbf{1}\{y_i \cdot h_\theta(x_i) < 0\} \tag{1}$$

with optimization variables θ, input data (x_i, y_i), $i = 1, \ldots, m$ and where $\mathbf{1}$ denotes the indicator function (one if the argment is true and zero otherwise).

Show that we can, however, solve this problem as a binary mixed integer program. That is, write an optimization problem over the variables θ and z with whatever (convex) objective and constraints you want, plus the additional constraint that $z_i \in \{0, 1\}$, such that the solution of this optimization problem gives the θ the minimizes 0/1 loss.

3. **Machine Learning Diagnostics [10pts]** In an effort to improve your grade in 15-780 you are designing a machine learning algorithm that attempts to classify web pages as helpful for learning about AI versus wastes of time. You have painstakingly manually annotated 1000 web sites of each type, as well as designed 500 custom features based upon your own knowledge of the subject which you believe will be helpful in this classification task. You split your data into training and testing sets, run regularized logistic regression on the problem, and find that you correctly classify 90% of the examples in the training set (acceptable performance) but only 60% of the examples on the test set (not good enough). Which of the following would be reasonable next steps to try? In each case, write a short (≤ 2 sentence) description of why this would be a good idea or not. If you need to run an additional experiment to tell whether this is a good idea, briefly mention what experiment that would be and how the results would influence what you decide.

   - Collect more data (i.e., label more websites manually)
   - Design more features to help you classify the website
   - Try using fewer features (selecting only the ones that look the most relevant out of 500)
   - Lower the regularization parameter λ
   - Increase the regularization parameter λ
   - Try using a more complicated algorithm (i.e., a neural network with more parameters than logistic regression)

4. **Mixed integer programming [30pt]** In this problem you will develop a mixed integer programming algorithm, based upon branch and bound, to solve Sudoku puzzles as described in class.

   In particular, you need to implement the function

```
def solve_sudoku(puzzle):
  # ...
  return (solved_puzzle, constraints)
```

The function takes as input a Sudoku puzzle as a 9x9 "list of lists" of integers, i.e.,

```
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 zeros represent missing entries that must be assigned by your algorithm, and all other integers represent a known assignment. The function needs to return a tuple `(solved_puzzle, constraints)`, where `solved_puzzle` contains the input puzzle with all the zeros assigned to their correct values. For instance, for the above puzzle this would 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 augment the Sudoku linear program (described in more detail below) with the constraints $(z_{8,5})_2 = 1$ and $(z_{5,3})_4 = 1$, then the solution will have integer values and give the solution above (these values are all indexed staring at 1, so you'll need to add/subtract one to convert to and from Python indices). Note that there can be more than one set of constraints that will lead to the same integer solution, and we'll check your solution by directly plugging these values in, not by comparing with our solution.

To solve this this problem, you will need to do two things:

(a) Write code to solve the linear programming relaxation of a Sudoku puzzle, using the representation mentioned in class. In particular, if we let $\bar{z}_{i,j} \in \mathbb{R}^9$ denote the indicator of the (i,j) square in a Sudoku board, then we want to solve the optimization problem

$$\begin{aligned}
\text{minimize} \quad & \sum_{i,j=1}^{9} \max_k (z_{i,j})_k \\
\text{subject to} \quad & z_{i,j} \in [0,1]^9, \quad i,j = 1,\ldots,9 \\
& \sum_{k=1}^{9} (z_{i,j})_k = 1, \quad i,j = 1,\ldots,9 \\
& \sum_{j=1}^{9} z_{i,j} = \mathbf{1}, \quad i = 1,\ldots,9 \\
& \sum_{i=1}^{9} z_{i,j} = \mathbf{1}, \quad j = 1,\ldots,9 \\
& \sum_{k,\ell=1}^{3} z_{i+k,j+\ell} = \mathbf{1}, \quad i,j \in \{0,3,6\} \\
& (z_{i,j})_k = 1, \quad \text{if puzzle}_{i,j} = k
\end{aligned}$$

where the first constraint encodes the fact that all variables must be between zero and one (a relaxation of the constraint that they be zero or one); the second constraint encodes the fact that each grid must have only one assigned number; the third that each column must contain each number; the fourth that each row must contain each number; the fifth that each 3x3 box must contain each number; and the sixth specifies the assignment of those elements that are fixed by the puzzle. You should write code to solve this problem using cvxpy.

To test this portion of the assignment, you can use the following puzzle, where the linear programming relaxation happens to be exact (that is, you will get integer solutions if you solve the above optimization problem 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]]
```

Solving the linear programming relaxation for this problem should result in the 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 cvxpy will not be exactly integer valued, due to numerical approximation. For the purposes of this assignment, you can assume that any value within 0.005 of 0 or 1 is integer valued, and just round these entries in the `cvxpy` solution the nearest integer.

(b) Next, write a branch and bound algorithm that will solve these problems even in the case where the original linear programming relaxation is not tight. That is, implement the algorithm on slide 19 of the mixed integer programming slides (this is the "simpler" algorithm; you do not need to implement the version that generates feasible upper bounds).

One element we did not discuss in class is how you select which variables (in this case, which variable $(z_{i,j})_k$) to split on. A simple rule is to pick the variable with solution closest to 0.5 (i.e., the "most undetermined") and split on this variable, solving the two subproblems that constrain the variable to be either 0 or 1.

5. **Digit classification [30pts]** You've seen the example in class (and probably just about anywhere else you've seen a presentation about machine learning) but now you're going to actually do it: write a program that can classify images as digits. For this program, you will create the following single Python function in the `digits.py` file that classifies these images:

```
def classify_digits(images):
  # ...
  return labels
```

where `images` is a list of lists, with each entry being a list containing 784 floating point values between zero and one that represent the intensity of pixels in a 28x28 grid. Your function needs to output a list of labels, where each entry in this list is the integer corresponding to the digit in the respective image.

This evaluation function is the only one that you will actually submit, and your submission will be ranked according to the percentage of images in a held-out test set (that you do not have access to) that your function classifies correctly. If you are able to classify more than 90% of the images (we mention one way to do this below), you will receive full credit on the problem, but we will also give a small amount of extra credit to the students with the top 5 best performing functions as ranked on Autolab.

You can use whatever method you like to classify the images, but your method needs to run on the Autolab servers and note that there is also a 1MB upload limit for any submission. We believe that the genuine best way to do this will be to use a machine learning approach, though feel free to use whatever method you like if you think it can do better. In order to facilitate a machine learning approach to this problem, we have included files containing training data for this problem (as well as a separate validation set, which you are free to include in your training as well, but which can also serve as a separate way of evaluating your algorithm). In paricular, you can download the `train.pickle.gz` and `validation.pickle.gz` files from the class website. Uncompress them using the `gunzip` utility (or any variety of other compression utilities), and load the corresponding .pickle files in Python using the following command

```
import pickle
X,y = pickle.load(open("train.pickle","rb"))
```

After loading this data, the `X` variable will be a 10000-element list of 784-element lists, each corresponding to the 28x28 pixel images mentioned above, and `y` will be a list of the 10000 corresponding image labels. These are the same formats that your `classify_images` function should take as input and return respectively. You can use this data to build a system that will classify digits based upon their pixel values, and then test the performance of this function on the `validation.pickle` file. We also include a script `check_digits.py` that runs the `classify_digits` function above on the validation set and outputs the error; the Autolab grader is exactly the same, just using a separate set of data that you won't have access to.

Clearly, there are a number of different machine learning approaches you can use for this task, and you are free to use any of them (including external libraries like scikit.learn, but remember that this library is *not* installed on the autolab machines, so if you use a classifier from that library, you'll need to put it into a form that can be run as a stand-alone script). Having said that, here is a method that will get you *full credit* on the assignment:

- Implement regularized logistic regression, trained via Newton's method, using the equations on slide 40 from the class machine learning slides.
- Build 10 different logistic regression classifiers, using the 10000 training examples above and a regularization parameter of $\lambda = 1$ to classify each digit against all others (i.e., build a binary classifier to classify digits as 0 or anything other than 0, 1 or anything other than 1, etc; this is a standard way to transform a multi-class to binary classifier). Now, given 10 of these classifiers (really, 10 θ vectors), predict the class of a new image correponding to the highest inner product $\theta^T x_i$.
- (Be sure to augment the raw images with a constant term 1 at the end of the vectors x_i).