ECE 6270, Spring 2021

Homework #5

Due Sunday, April 25, at 11:59pm

1. Prepare a one paragraph summary of what we talked about since the last assignment. I do not want just a bulleted list of topics, I want you to use complete sentences and establish context (Why is what we have learned relevant? How does it connect with other classes?). The more insight you give, the better.

2. You have a large amount of money M that you are going to gamble on a horse race. You want to be smart about it, though.

   There are N horses running in the race. You will divide up your money to place a bet of x_i on each of them. Clearly,

   $$\sum_{i=1}^{N} x_i = M.$$

   As with any parimutuel betting scenario, if horse i wins, the payout to you is proportional to the amount you bet on horse i versus what everybody else (the "public") bet on horse i. If you wager x_i on horse i and the public wagers s_i then

   $$\begin{aligned}\text{payout if horse } i \text{ wins} &= C \cdot (\text{total amount of money bet on all horses}) \cdot \frac{x_i}{x_i + s_i} \\ &= C \cdot \left(M + \sum_{i=1}^{N} s_i\right) \frac{x_i}{x_i + s_i}.\end{aligned}$$

   The constant C above is less than 1, and represents the fact that the track takes a cut of all the bets (the "vigorish" or "vig" is $1 - C$). A typical value of C might be 0.8 or 0.9.

   The reason you are betting is that you have two pieces of key knowledge about this race. First, you know the *actual* probability p_i that horse i will win. Second, you know s_i, the amount that the public will end up placing on horse i.

   (a) With your knowledge of the probabilities $\boldsymbol{p}$ and public money $\boldsymbol{s}_i$, write down a convex optimization program answer will tell you how much to bet on each horse to maximize your expected return. (In the end, you should be maximizing a concave function over a convex set.)

   (b) Using Fenchel duality, show how this expected payout can be computed by solving an optimization program in one variable. (Hint: look at the resource allocation example in the notes.) All of the relevant functions are given to you here, so you can (and should) compute their conjugates explicitly.

   (c) Show how the primal optimal solution (the best x_i) can be recovered from the (single variable) dual solution.

   (d) Here are the track odds right before closing:[1]:

[1]These are the odds from the morning of the 2020 Kentucky Derby, but where I have removed horses with odds of 30-1 or longer to keep things simpler.

1. Tiz the Law	3-5
2. Honor A. P.	7-1
3. Authentic	8-1
4. Ny Traffic	12-1
5. Money Moves	13-1
6. Max player	19-1
7. Enforceable	22-1
8. Storm the Court	27-1

Saying horse i has track odds A-B simply means that a bet of B will yield A in profit (and a total return of $A+B$) should horse i win. From the payout equation above, this tells us that if horse i has odds A-B, then[2]

$$A + B = C\left(\sum_{i=1}^{N} s_i\right)\frac{B}{s_i}.$$

Note that by examining these odds, you can infer what the vig is $(1-C)$. If you know the total amount the public has wagered you can also determine the s_i's. Suppose that the public has wagered a total of \$20 million on this race.

You happen to have some inside information that makes you believe that Tiz the Law is not as strong a favorite as the public seems to believe. Based on your information you think that the true probabilities are

1. Tiz the Law	25%
2. Honor A. P.	25%
3. Authentic	25%
4. Ny Traffic	7%
5. Money Moves	6%
6. Max player	5%
7. Enforceable	4%
8. Storm the Court	3%

You have \$500,000. How much do you bet on each horse? What is your expected return? (You should calculate your return using the value of C and the s_i's determined by your optimization problem.) What is the variance (a simple measure of risk) of your expected return?

(e) Authentic won the race.[3] How much profit did you make?

(f) Explore how your answer changes if you bet larger amounts (in terms of the strategy, expected return, and risk). Can you explain why your betting strategy changes as the amount wagered goes up?

3. In this problem we will explore two alternative approaches to solving a simple variant of the least squares problem where we add the constraint that the solution is non-negative, i.e., we wish to solve

$$\underset{\boldsymbol{x}\in\mathbb{R}^N}{\text{minimize}} \ \frac{1}{2}\|\boldsymbol{y} - \boldsymbol{A}\boldsymbol{x}\|_2^2 \qquad \text{subject to} \qquad \boldsymbol{x} \geq \boldsymbol{0}.$$

[2]Note that this payout equation represents the payoffs that would be made taking into account the bets that have been made up till now, but *before* taking into account any bets you will make at the last minute.

[3]This actually happened.

This is natural in many practical applications where the entries of $\boldsymbol{x}$ have physical interpretations (e.g., light intensity, power, concentration of some physical material, etc.) that don't really make sense as negative quantities. Below we will assume throughout that $\boldsymbol{A}$ is an $M \times N$ matrix with $M > N$ and that $\boldsymbol{A}$ is full rank.

(a) Derive the Lagrangian function for this optimization problem (pay careful attention to the sign of each term).

(b) Show that the dual optimization problem is itself another nonnegative least squares problem. Do this by deriving the dual function $d(\boldsymbol{\lambda})$ by first finding the $\boldsymbol{x}$ that minimizes the Lagrangian $\mathcal{L}(\boldsymbol{x}, \boldsymbol{\lambda})$, and then plugging this into $\mathcal{L}(\boldsymbol{x}, \boldsymbol{\lambda})$. Simplify the dual optimization problem as much as possible.

(c) Recall from the notes that one of the KKT conditions (KKT4) is that if $\boldsymbol{x}^\star, \boldsymbol{\lambda}^\star$ are primal/dual optimal, then $\nabla_{\boldsymbol{x}} \mathcal{L}(\boldsymbol{x}^\star, \boldsymbol{\lambda}^\star) = \mathbf{0}$. Show that this implies that for the solution

$$\boldsymbol{A}^{\mathrm{T}} \boldsymbol{A} \boldsymbol{x}^\star - \boldsymbol{A}^{\mathrm{T}} \boldsymbol{y} - \boldsymbol{\lambda}^\star = 0.$$

(Note that it is easy to find a $\boldsymbol{x}$ and $\boldsymbol{\lambda}$ that satisfy the condition above – for any $\boldsymbol{x}$ we can form a $\boldsymbol{\lambda}$ that will make this identity true. The trick is that usually the resulting $\boldsymbol{\lambda}$ will not actually satisfy $\boldsymbol{\lambda} \geq 0$.)

(d) Try solving a nonnegative least squares problem using CVXPY. The file `hw06.py` contains some code to set up a nonnegative least squares problem and then solve it using CVXPY. Make sure you understand what the code is doing, and verify that the resulting solution satisfies the optimality condition from part (c).

(e) Implement the projected gradient descent approach described on page 15 of the notes. This should be a minor variation on something you have already implemented before, but there are two important caveats. First, when we solved least squares problems before using gradient descent, we calculated the optimal step size α_k at each iteration. This is much tougher to do in this case because the actual optimal step size would involve figuring out what α is optimal after accounting for the projection step, and this is not easy to do in closed form. You should either use a line search to choose α, or just take a fixed step size. For guidance, the theory guarantees convergence if $\alpha \leq 1/\|\boldsymbol{A}^{\mathrm{T}}\boldsymbol{A}\|_2$.

The second challenge here involves defining a stopping criterion. You cannot expect that the norm of the gradient will be zero at the solution. Instead you could either define a stopping criterion involving $\|\boldsymbol{x}^{(k)} - \boldsymbol{x}^{(k-1)}\|_2$, or alternatively you could do something inspired by part (c) above.

(f) Another way to solve this problem that avoids having to worry about a step size is a primal-dual approach where we take alternating steps over $\boldsymbol{x}$ and $\boldsymbol{\lambda}$ with the goal of satisfying the condition in (c). Specifically, suppose that we are given an estimate $\boldsymbol{\lambda}^{(k)}$. We can then update $\boldsymbol{x}$ by solving the optimality condition from part (c) to obtain

$$\boldsymbol{x}^{(k+1)} = \Big((\boldsymbol{A}^{\mathrm{T}}\boldsymbol{A})^{-1} (\boldsymbol{A}^{\mathrm{T}} \boldsymbol{y} + \boldsymbol{\lambda}^{(k)}) \Big)_+ .$$

Note that we include the projection onto the set of nonnegative $\boldsymbol{x}$ since, in general, $(\boldsymbol{A}^{\mathrm{T}}\boldsymbol{A})^{-1}(\boldsymbol{A}^{\mathrm{T}}\boldsymbol{y} + \boldsymbol{\lambda}^{(k)})$ will lead to negative entries (unless we are already at convergence). Next we can update $\boldsymbol{\lambda}$ by again solving the optimality condition from part (c), which in this case yields

$$\boldsymbol{\lambda}^{(k+1)} = \left(\boldsymbol{A}^{\mathrm{T}}\boldsymbol{A}\boldsymbol{x}^{(k+1)} - \boldsymbol{A}^{\mathrm{T}}\boldsymbol{y}\right)_{+},$$

where again we project the naïve solution to the optimality condition onto the set of nonnegative $\boldsymbol{\lambda}$. Implement this algorithm and compare the results to the projected gradient descent approach from the previous problem.

4. We have presented gradient descent as a basic method for solving smooth unconstrained problems. In this problem we will explore its use in solving nonsmooth constrained problems, specifically linear programs.

   (a) Consider the linear program in $\mathbb{R}^2$,

   $$\min_{\boldsymbol{x}} \ \langle \boldsymbol{x}, \boldsymbol{c}\rangle \quad \text{subject to} \quad \langle \boldsymbol{x}, \boldsymbol{a}_m\rangle \le b_m$$

   where

   $$\boldsymbol{c} = \begin{bmatrix}1\\1\end{bmatrix}, \quad \boldsymbol{a}_m = \begin{bmatrix}\cos(m\pi/3)\\ \sin(m\pi/3)\end{bmatrix}, \quad b_m = 1, \quad m = 1, 2, \ldots, 6.$$

   Sketch the feasible region in the plane – the region where all six linear constraints hold. Using Python, create an image of $\langle \boldsymbol{x}, \boldsymbol{c}\rangle$ over the feasible region. Where is the minimizer $\boldsymbol{x}^{\star}$?

   (b) Now consider the smooth, unconstrained problem

   $$\min_{\boldsymbol{x}} \langle \boldsymbol{x}, \boldsymbol{c}\rangle - \frac{1}{\tau}\sum_{m=1}^{6} \log(b_m - \langle \boldsymbol{x}, \boldsymbol{a}_m\rangle).$$

   Using Python, create an image of the functional over the feasible region for $\tau = 1$. What is happening near the boundary?

   (c) Solve the program above using gradient descent (with backtracking – you can set $c_1 = 0.001$ and $\rho = 0.8$) for $\tau = 1, 10, 100, 500, 1000$ and starting at the origin, $\boldsymbol{x}_0 = \boldsymbol{0}$. Make a figure of the feasible region, then put an 'x' where your solution landed for these different τ. Note how many iterations it took for each τ.

   (d) Do the same, but use the solution for $\tau = 1$ as the starting point for $\tau = 10$ then use that solution for $\tau = 100$, etc. How many total iterations does it take to get the answer for $\tau = 1000$?

5. Consider the general quadratic programming problem with linear constraints:

   $$\min_{\boldsymbol{x}} \ \frac{1}{2}\boldsymbol{x}^{\mathrm{T}}\boldsymbol{P}\boldsymbol{x} + \langle \boldsymbol{x}, \boldsymbol{q}\rangle \quad \text{subject to} \quad \boldsymbol{A}\boldsymbol{x} \le \boldsymbol{b}.$$

   We will assume that $\boldsymbol{P}$ is symmetric positive definite (and so has full rank).

   (a) Find the dual.

(b) Show how the primal optimal solution $\boldsymbol{x}^\star$ can be computed from the dual optimal solution $\boldsymbol{\lambda}^\star$. (You should be able to convince yourself that both programs have unique solutions.)

(c) Describe how we can use ADMM to solve the dual problem (in a non-distributed manner, to start).

(d) Describe how we can write the dual as

$$\min_{\boldsymbol{\alpha},\boldsymbol{\beta}} g(\boldsymbol{\alpha}) + h(\boldsymbol{\beta}),$$

where $g(\cdot)$ is **separable** (a sum of individual components of $\boldsymbol{\alpha}$) and $h(\cdot)$ is an indicator of a convex set which we can easily project onto. Write down explicitly how this projection operator works.

(e) Using your results from part (d), describe how we can use distributed ADMM to solve the dual program.

6. Consider the optimization problem

$$\underset{\boldsymbol{z}}{\text{minimize}} \ \ \|\boldsymbol{z}\|_1 + \frac{1}{2\alpha}\|\boldsymbol{z} - \boldsymbol{v}\|_2^2, \tag{1}$$

where $\boldsymbol{v}$ is a fixed vector. This is the optimization problem that defines the prox operator of the ℓ_1 norm, and it arises in many approaches to solving the LASSO (e.g., ADMM). We claimed in class that the solution to this optimization problem is given by $\boldsymbol{z}^\star = T_\alpha(\boldsymbol{v})$ where

$$[T_\alpha(\boldsymbol{v})]_i = \begin{cases} v_i - \alpha, & v_i > \alpha \\ 0, & |v_i| \le \alpha \\ v_i + \alpha, & v_i < -\alpha. \end{cases}$$

Prove that this is indeed the case.

7. (Optional.) In this problem you will solve the LASSO using a distributed implementation of ADMM.

(a) Begin by creating an instance of your problem by creating a matrix $\boldsymbol{A}$ which is $M \times N$ where $N = 10M$ with standard normal random variables as entries. I will let you experiment with the size M and N. The goal is to make these large, but not so large that you cannot possibly solve the problem given the constraints of your personal machine. Crease a vector of observations $\boldsymbol{b}$ by choosing a sparse vector $\boldsymbol{x}_0$ that has at most $M/4$ nonzero entries and computing $\boldsymbol{b} = \boldsymbol{A}\boldsymbol{x}_0$ and then adding a small amount of noise.

(b) Implement a non-distributed version of ADMM and apply it to your problem. You can use whatever method you prefer to solve the least squares problem at each iteration. You will need to tune the parameter τ to ensure that you are in a regime where you are doing a good job of estimating $\boldsymbol{x}_0$.

(c) Using www.cvxpy.org/examples/applications/consensus_opt.html as a guide, create a distributed implementation of ADMM for your problem (you can use the same basic structure, but do not use CVXPY to solve the subproblems, instead solve the least squares problem and implement soft-thresholding explicitly.

(d) Compare the run times for the two approaches (the difference here will depend significantly on the specifics of your hardware). Try this comparison for a few different sizes of problems and comment on what you observe.