15.095: Machine Learning Under a Modern Optimization Lens **Fall 2019**

Recitation 6: Midterm Review Notes

TA: Ryan Cory-Wright

Please email any typos and/or suggestions to ryancw@mit.edu

In this handout, we provide an (incomplete) review of the material covered in the first half of 15.095, to assist studying for the midterm exam. We recommend recommend reading this handout *and* the relevant lecture slides/ chapters of the textbook[1] before attempting the past exams on Canvas.

6.1 Background on Optimization

6.1.1 Norms and Binary Variables

- You can model $\|\boldsymbol{x}\|_1 \leq 5$ without binary variables. Indeed, it is equivalent to

$$\boldsymbol{e}^\top \boldsymbol{z} \leq 5, \ \boldsymbol{z} \geq \boldsymbol{x}, \boldsymbol{z} \geq -\boldsymbol{x}.$$

- You cannot model $\|\boldsymbol{x}\|_1 \geq 5$ without binary variables (Try it. What goes wrong?).
- An easy way to see whether you can model a norm constraint without binary variables is to draw the feasible region. The feasible region

$$\|\boldsymbol{x}\|_1 \leq 5$$

is convex, so we don't need binary variables. The second feasible region is non-convex, so we need binary variables.

6.1.2 Modeling with Binary Variables

Binary variables let us flexibly model choices between different actions. This is important, because we usually need to make discrete choices when solving optimization problems. For instance:

- $x_i = 1$ if a choice occurs, $x_i = 0$ otherwise.
- At most one event occurs: $\sum_i x_i \leq 1$.
- Exactly one event occurs: $\sum_i x_i = 1$.
- Either zero events or both events occur: $x_1 = x_2$.
- If x_1 occurs then x_2 occurs: $x_1 \leq x_2$.

A (hard!) exercise to try at home:

- Let $y_1, \ldots, y_n$ be continuous decision variables guaranteed to satisfy $L_i \leq y \leq U_i$. Introduce a new decision variable z and model the constraint $z = \max\{y_1, \ldots, y_n\}$.

[1]For instance, material emphasised in lectures but not mentioned in this handout may also be fair game.

6.1.3 The Big-M Method

Many important problems exhibit a logical relationship between continuous variables x and binary variables z of the form "$x = 0$ if $z = 0$". In particular, a cardinality constraint on a set of regressors x can be imposed by requiring that $x_i = 0$ if $z_i = 0$ for each i, and that $\sum_i z_i \leq k$

One way to model logical relations between discrete and continuous variables is to impose a so-called big-M constraint of the form $-Mz \leq x \leq Mz$ for a sufficiently large constant $M > 0$.

When solving problems with big-M constraints, it is important to pick the right value of M (what happens if M is too small? what happens if M is too big?)

6.1.4 Branch-and-Bound

This is an algorithm which can be used to solve any integer optimization problem. At each iteration, we consider a subproblem. We solve a relaxation of a subproblem to obtain a lower bound on its objective value. If the lower bound is greater than the upper bound, we are finished with the subproblem. If the relaxation produced an integer solution, we are finished with the subproblem. Otherwise, we branch into more subproblems. The relaxation can be obtained by solving a continuous convex program. We have a guarantee that the branch-and-bound algorithm will find an optimal solution.

6.1.5 The Cutting-Plane Method

Consider the optimization problem

$$\min_{\boldsymbol{x}\in\mathcal{X}} \ f(\boldsymbol{x})$$

where [if you don't know what this terminology means, take it to mean "the problem is convex, well behaved"]:

- $\mathcal{X}$ is a non-empty compact convex set.
- $f(\cdot)$ is lower semicontinuous in $\boldsymbol{x}$.
- We can cheaply evaluate a subgradient of $f(\cdot)$ at a given $\boldsymbol{x}$.

In a cutting-plane method, we solve this problem by iteratively minimizing:

$$\begin{aligned} z^t &= \min_{\boldsymbol{x}\in\mathcal{X}} \theta \text{ s.t.} \theta \geq f(\boldsymbol{x}^i) + \nabla f(\boldsymbol{x}^i)^\top(\boldsymbol{x} - \boldsymbol{x}^i), \ \forall i \in [t], \\ \boldsymbol{x}^t &= \arg\min_{\boldsymbol{x}\in\mathcal{X}} \theta \text{ s.t.} \theta \geq f(\boldsymbol{x}^i) + \nabla f(\boldsymbol{x}^i)^\top(\boldsymbol{x} - \boldsymbol{x}^i), \ \forall i \in [t], \end{aligned}$$

and obtaining a sequence of lower/upper bounds $(z^t, f(\boldsymbol{x}^t))$.

Convergence properties:

- In practice, the cutting-plane method often converges very fast.
- If $\mathcal{X}$ is a binary set (e.g. in sparse linear regression) then we converge in a finite number of iterations, since we never visit a point twice and there are finitely many binary points.

- In general (e.g. in the robust linear regression case), the cutting-plane method converges to within ϵ of *an* optimal solution in a finite number of iterations[2].
  The proof of this is quite technical and beyond the scope of this class[3].

6.1.6 Heuristics

These are frameworks for obtaining high-quality feasible solutions to optimization problems. One example of a heuristic is local search, in which we start with some feasible integer solution, and then search the neighboring feasible points of the current integer solution and see if any of them have a lower cost. The term "neighbouring feasible points" must be defined for the particular problem. Heuristics converges to a suboptimal solution (although, if we are lucky, it may converge to an optimal solution). We have no guarantees on how close a heuristic solution is to an optimal solution.

Heuristics can be used to speed-up exact methods like the cutting-plane method or branch-and-bound. They accelerate exact methods for two reasons. First, they improve the quality of the incumbent solution, which means that the exact method doesn't have to identify a good solution. Second, they allow exact methods to prune vectors of partial solutions which are provably worse than the warm-start, which in turn improves the exact method's bound quality, by reducing the set of feasible solutions which can be selected at each subsequent iteration.

[2]Be careful with the "an" here! There could be more than one optimal solution.

[3]See Mutapcic and Boyd (2009). "Cutting-set methods for robust convex optimization with pessimizing oracles"; Bertsimas and Cory-Wright (2019). "On Polyhedral and Second-Order-Cone Decompositions of Semidefinite Optimization Problems".

6.2 Machine Learning Concepts

6.2.1 Classification Loss Functions

Let there be n data points $(\boldsymbol{x}_i, y_i)$, where y_i is one of $1, \cdots, K$ classes. Let f_i be the predicted label for the ith point, and p_{ij} be the predicted probability that ith point is in the jth class.

Then the common loss functions for classification can be written as:

- Misclassification Loss
$$\frac{1}{n}\sum_{i=1}^{n} I(f_i \neq y_i)$$
  - Pros: easy to understand, simple to compute.
  - Cons: performs poorly if the classes are highly unbalanced, or if we have more than 2 classes.
- Gini Impurity
$$\frac{1}{n}\sum_{i=1}^{n}(1 - p_{iy_i})$$
- Entropy
$$\frac{1}{n}\sum_{i=1}^{n} -\log(p_{iy_i})$$
- Gini and Entropy perform similar in practice, so the better one to use depends on the application (try both and see which one performs better out of sample).
- Entropy is slightly slower to compute than Gini.

6.2.2 Area Under Receiver-Operator Characteristic Curve (AUC)

AUC is an evaluation metric that accounts for the "overall" diagnostic ability of a binary predictive algorithm $f : \mathbf{X} \to \{0, 1\}$. One can show that AUC is the probability that a random positive (1) data point would be assigned a higher probability of 1 than a random negative (0) example. An AUC of roughly 0.7 and above is an "acceptable" figure for ML models, but the model would not be very good at individual predictions. An AUC of 0.9 or above usually translates to good individual predictions.

- In-sample: $0.5 \leq AUC \leq 1$.
- Out-of-sample: $0 \leq AUC \leq 1$.

6.2.3 R^2

Coefficient of Determination (R^2):

$$R^2 = 1 - \frac{\sum_{i=1}^{n}(\hat{y}_i - y_i)^2}{\sum_{i=1}^{n}(\bar{y} - y_i)^2}.$$

- y_i's are true values, $\hat{y}_i$'s are predicted values, $\bar{y}$ is mean of training set.

- Measures how well the model fits the data compared to a simple baseline model that predicts $\bar{y}$.
- Can evaluate any **regression** model, not just linear regression.
- In-sample: $0 \leq R^2 \leq 1$.
- Out-of-sample: $-\infty < R^2 \leq 1$.

6.3 Machine Learning Techniques

A common task in machine learning is to fit a linear model $\boldsymbol{w}$ to some training data $(\boldsymbol{X}, \boldsymbol{y}) \in (\mathbb{R}^{n \times p}, \mathbb{R}^n)$ to minimize a loss function ℓ. In this model, we assume that the relationship between the input and response variables is given by $y_i = \boldsymbol{w}^\top \boldsymbol{X}_i + \epsilon$, and ϵ is noise. This task can be modelled as solving:

$$\min_{\boldsymbol{w}} \quad \sum_{i=1}^{n} \ell(y_i, \boldsymbol{X}_i^\top \boldsymbol{w}).$$

6.3.1 Regularization and Robustness

In the machine learning community, many people have observed that $\boldsymbol{w}$ typically overfits the training data, and proposed combating overfitting by augmenting the objective with a regularizer. This leads to the optimization problem:

$$\min_{\boldsymbol{w}} \quad \sum_{i=1}^{n} \ell(y_i, \boldsymbol{X}_i^\top \boldsymbol{w}) + \Omega(\boldsymbol{w}),$$

where $\Omega(\boldsymbol{w})$ is a regularizer. Common choices of $\Omega(\cdot)$ include:

- A lasso regularizer: $\Omega(\boldsymbol{w}) = \|\boldsymbol{w}\|_1$.
- A ridge regularizer: $\Omega(\boldsymbol{w}) = \|\boldsymbol{w}\|_2^2$.

In lecture 2, we saw that imposing a regularizer is actually equivalent to solving a robust optimization problem. In particular, it is equivalent to assuming that nature has perturbed each column of $\boldsymbol{X}$ in the worst way possible, but nature is bounded in its power (see lecture 2 for the specifics).

Takeaways:

- Lasso is a robust method, but not necessarily a sparse method.
- To ensure sparsity without making assumptions on the data, we need to use mixed-integer optimization.

6.3.2 Sparsity

In lecture 3, we saw a second way to combat overfitting in machine learning models which also enhances interpretability. Namely, imposing a sparsity constraint. This leads to the following optimization problem:

$$\min_{\boldsymbol{w}} \quad \sum_{i=1}^{n} \ell(y_i, \boldsymbol{X}_i^\top \boldsymbol{w}) \text{ s.t. } \|\boldsymbol{w}\|_0 \leq k,$$

where $\|\cdot\|_0$ counts the number of non-zero entries in $\boldsymbol{w}$.

- When we have more data than features, the sparsity constraint is desirable, because it enhances interpretability.
- When we have more features than data, the sparsity constraint is necessary, because there are infinitely many different models which perfectly fit the training data.

We have seen two ways to model sparsity constraints. Either way, we first rewrite the sparsity constraint as:

$$w_i = 0 \text{ if } z_i = 0, \ \forall i \in [n],$$

where:

$$\boldsymbol{z} \in \{0,1\}^n, \ \boldsymbol{e}^\top \boldsymbol{z} \le k,$$

and then reformulate the logical constraint $w_i = 0$ if $z_i = 0$ in one of two different ways.

1. Use the big-M method to rewrite the logical constraints as $-Mz_i \le w_i \le Mz_i$.
2. Impose a ridge regularizer (i.e., $\frac{1}{2\gamma}\|\boldsymbol{w}\|_2^2$) in the objective, replace w_i with $w_i z_i$ everywhere in the problem *except the regularizer*, take the dual with respect to $\boldsymbol{w}$ and run a cutting-plane method.

6.3.3 CART

- Classification and Regression Trees (Brieman et al. 1984)
- Greedily pick the parallel split which reduces the loss function by the largest amount.
- We build a deep tree that overfits, then prune back to a smaller tree that performs well on a hold-out validation set.
- Loss functions (for classification):
  - Misclassification
  - Gini
  - Entropy
- For each loss function, we can define the loss for both a **single leaf** and the **overall tree**.

6.3.4 Optimal Trees

The key idea is that classification trees (CART) are interpretable, so let's try to optimize and find the best possible interpretable model for a problem. Optimal Trees achieves this via a local search method.

For each random start:

1. Construct a (deep) tree using a random subset of the variables and a random sample of the training observations.
2. Go to a random node: If it is a leaf node, consider adding a split. If it is not a leaf node, consider: Pruning the tree so that this becomes a leaf node, Changing the variable, threshold, and/or direction of the split, while preserving the structure of the tree under the left and right daughter nodes.
3. Choose the option from step 2 which decreases the loss function (misclassification, gini impurity, or entropy) by the largest amount, and update the tree accordingly.
4. Once we have visited all nodes of the tree without updating further, terminate. Otherwise, return to step 2 and try improving another node in the tree.

The complete method can then be stated as follows:

- Input: Training/Validation data, number of random starts N.
- Output: Optimal Classification Tree (OCT) model.

Method:

1. Run the local search algorithm to construct N locally optimal trees on the training data set.
2. For each tree, find the optimal cp value using the validation set.
3. Take the average/median of the cp values to get the optimal cp value.
4. From this cp value, determine the coefficient α of the regularizer term in the overall Optimal Trees objective function.
5. Retrain the trees on the entire Training + Validation data set. Select the tree that minimizes the overall objective function as the final model.

Characteristics of Optimal Trees:

- **Interpretability:** Trees are among the most interpretable machine learning algorithms out there.
- **Accuracy:** Optimal Classification Trees (OCT) performs better than CART. OCT with Hyperplane splits matches the accuracy of Random Forest and Gradient Boosted Trees (XGBOOST).
- **Scalability:** Optimal Classification Trees scales to successfully solve problems with millions of samples and thousands of features.

Summary:

- Parameters: minbucket, max_depth, *cp*
- Loss functions: misclassification, gini, entropy
- CART: Greedy splits
- Optimal Classification / Regression Trees
  - Find high-quality solutions using local search procedure with random restarts.

6.3.5 Prescriptive Methods

Fundamental Problem of OR: How can we solve an optimization problem with uncertain data?

Suppose that we are given data $(\mathbf{x}_i, y_i, z_i), i \in [n]$, where $\mathbf{x}_i \in \mathbb{R}^p$ are covariates, $z_i \in \mathbb{R}^q$ are decisions, and $y_i \in \mathbb{R}$ are possibly unknown outcomes which depend upon $\mathbf{x}_i$ and z_i. The purpose of "Predictive Methods" is to predict the outcomes for some unknown y_i's given the $\mathbf{x}_i$'s as inputs. We have seen many examples of predictive methods in this class, including Sparse Regression, Convex Regression, and Optimal Trees.

On the other hand, the purpose of "Prescriptive Methods" is to solve the problem:

$$\begin{aligned} \min_{\mathbf{z}} \quad & \mathbb{E}_{\mathbf{y}} \sum_{i=1}^{n} c(\mathbf{x}_i, y_i, z_i) \\ \text{s.t.} \quad & \mathbf{z} \in \mathcal{Z}, \end{aligned}$$

where $c(\mathbf{x}_i, y_i, z_i)$ is the cost of making decision z_i in scenario i, and $\mathcal{Z}$ is the set of all feasible decisions. For instance, $\mathbf{x}_i$ could represent the clinical covariates of patient i, y_i could be the health outcome, and $z_i \in \{1, \dots, L\}$ could be the drug to prescribe from L possible options.

We now discuss two methods for solving this problem:

Sample Average Approximation: Given a finite amount of data, we can make the assumption that the y_i's are random, but there is no good function to predict them. In this case, we apply the "Sample Average Approximation", and assume that the y_i's are drawn uniformly at random from some fixed but unknown distribution. In this case, we simply average over the y_i's, assuming that they are equally probable:

$$\begin{aligned} \min_{\mathbf{z}} \quad & \sum_{i=1}^{n} \left(\frac{1}{n} \sum_{j=1}^{n} c(\mathbf{x}_i, y_j, z_i) \right) \\ \text{s.t.} \quad & \mathbf{z} \in \mathcal{Z}. \end{aligned}$$

If our decision z does not affect y and we have enough data, then the SAA approach is optimal. However, if we have a small amount of data, or z affects y, then SAA isn't very good, and we need a different approach, which places more weight on y_i's which we predict are more likely to be similar to our uncertain outcome.

Prescriptive Analytics: This approach uses machine learning to generate smarter weights than those given by the Uniform Random distribution. We consider the following problem:

$$\begin{aligned} \min_{\mathbf{z}} \quad & \sum_{i=1}^{n} \left(\sum_{j=1}^{n} w_{i,j} c(\mathbf{x}_i, y_j, z_i) \right) \\ \text{s.t.} \quad & \mathbf{z} \in \mathcal{Z}. \end{aligned}$$

We use a machine learning model to generate the weights $w_{i,j}$, with the property that $\sum_{j=1}^{n} w_{i,j} = 1$ for all i. For the SAA model, we have $w_{i,j} = \frac{1}{n}$ for all i, j. For the k-nearest neighbors model, we have:

$$w_{i,j} = \begin{cases} \frac{1}{k} & \text{if } \mathbf{x}_j \text{ is among the } k\text{-nearest neighbors of } \mathbf{x}_i, \\ 0 & \text{otherwise.} \end{cases}$$

For a decision tree model such as CART or Optimal Trees, we could use the weights:

$$w_{i,j} = \begin{cases} \frac{1}{N_{\ell(i)}} & \text{if } \mathbf{x}_j \text{ is in the same leaf node as } \mathbf{x}_i, \\ 0 & \text{otherwise,} \end{cases}$$

where $N_{\ell(i)}$ is the number of points in the same leaf node as $\mathbf{x}_i$. There are more complicated machine learning models that we could use as well, such as kernel methods. For many of these predictive models, we can prove that the solution is asymptotically optimal.

6.4 Misc

Try to practice good exam technique! For instance:

- In general, not every exam question worth the same number of marks is equally difficult. It is usually a good idea to try to answer easier questions first.
- If you don't know the answer to a question, you can usually obtain partial credit by either talking about something related or showing the first steps which you would take to obtain an answer. Similarly, if you run out of time, you can usually obtain some credit by sketching how you would answer the question.
- It takes three good nights to get over a bad night's sleep. Try to sleep well the week before the midterm.

If you have questions, please let the teaching team know via email. Happy studying and good luck!