Weighted Least-Squares

- Least-squares regression problem:
  - Basis functions: $f_1(x), f_2(x), \ldots, f_n(x)$
  - Find coefficients: $w_1, \ldots, w_k$
  - Minimize:
    $$\min_w \sum_j (t_j - \sum_i w_i f_i(x))^2$$

- Some points are more important than others:
  - Weighted least-squares:
    $$\min \sum_j \alpha_j (t_j - \sum_i \alpha_i w_i f_i(x))^2$$
    - $\alpha_j$ weight of point $j$ if care more about $j\text{ than } \alpha_j$ is larger...
Robust Least Squares

- Weighted least squares:
  - Test set distribution may be different from training set!
    - Must reweigh according to likelihood ratio:
      \[ \alpha_j = \frac{\mathcal{L}(x_j)}{P(\mathcal{L}(x_j))} \]
  - But what is the test set distribution???
- Don’t want to commit!
  - Pick worst case weights!

Robust LS:

\[
\max_{\alpha} \min_w \sum_j \alpha_j (t_j - \mathcal{L}_w(f(x_j)))^2 \\
\quad \alpha > 0 \quad \sum_j \alpha_j = 1
\]

Optimization of Robust LS

- Robust LS problem:
  - For each set of weights, must solve weighted least squares:
  - How do we find worst case weights?
    - Option B : guess weights, solve least squares, tweak weights,…
Equivalent optimization problem

- Robust LS:
  - Pushing min \( w \) into constraint:
    - Non-linear constraint, give up!

Minimum over \( w \) as infinite constraints

- Non-linear min constraint:
  - Infinite constraint set:
    - Good \( L^p \)
    - Bad news: infinitely many constraints
    - Great! Had a non-linear constraint, now all I have are infinite constraints, for each \( w \)!
Constraints for one alpha, help with other alphas

- Suppose you have $\alpha_0$, and introduce a constraint for some coefficients $w_0$:
  $$w_0 = \arg\min_w \sum_j c_j (t_j - w_j)^2$$

- Constraint also upper bound for other weights $\alpha_i$:
  $$\sum_j c_j (t_j - w_j)^2 \leq \varepsilon$$

- Linear constraint! Cool!

A geometric view

- We have an infinite number of linear constraints, many are irrelevant
  - Set of constraints forms a convex set

- Linear program with one constraint per $w$
  - Still infinite…
Suppose we use a subset of the constraints

- What if we use a finite number of constraints
  - Set of constraints at a finite set of coefficients $\Omega$

\[
\max \sum_{e \in E} \varepsilon \leq \sum_{j} \delta_j (t_j - w_j f_j) \quad \forall w \in \mathcal{W}
\]
\[
d \geq 0 \quad \sum_{j} \delta_j = 1
\]

- Can solve with any LP solver!
- But, solution with subset of constraints may not be a solution to original problem
  - Fewer constraints, solution may be infeasible, value of LP too high…

Active constraints

- Original LP with infinite constraints:

\[
\max \varepsilon \leq \sum_{j} \delta_j (t_j - w_j f_j) \quad \forall w \in \mathcal{W}
\]
\[
d \geq 0 \quad \sum_{j} \delta_j = 1
\]

- How many variables? $n + 1$
- How many active constraints at optimal solution? $n + 1$

- So, if we knew set of active constraints at optimal solution $\Omega^*$
  - Could discard all other constraints

  \[
  \text{solve only using } \Omega^* \quad \text{ignore all other constraints in } K
  \]
Active Constraints at Optimal Point

- Original problem:
  - If we knew set of active constraints at optimal solution \( \Omega^* \)
    - Could discard all other constraints
    - Solution will be feasible with respect to original problem

Consider some set of constraints \( \Omega \):
- Too few, infeasible solution:
  - \( \exists \) a violated constraint:
    - \( \exists \) \( w \) such that \( x^* \notin \Omega \)
  - \( \exists \) \( w \) in such that \( x^* \notin \Omega \)

- Just right, feasible solution:
  - \( \forall \) \( w \) such that \( x^* \in \Omega \)
  - \( x^* \) is optimal

Constraint Generation

- Start with some finite set of constraints \( \Omega \)
  - Solve LP, obtain \( \alpha_\Omega, \varepsilon_\Omega \)

- Check is \( (\varepsilon_\Omega, \alpha_\Omega) \) is feasible for infinite constraints:
  - If feasible, done!
    - \( \varepsilon_\Omega = \min \sum_j \alpha_\Omega^j (t_j - w_j) \)
    - \( \varepsilon_\Omega \leq \varepsilon_\Omega \)
  - Otherwise, add a constraint that makes \( (\varepsilon_\Omega, \alpha_\Omega) \) infeasible:
    - \( w_{\text{new}} = \alpha_{\text{new}} \sum_j \alpha_\Omega^j (t_j - w_j) \)
    - Why does this new constraint make \( (\varepsilon_\Omega, \alpha_\Omega) \) no longer feasible?
      - \( \exists \varepsilon_{\text{new}} \) such that \( (\varepsilon_{\text{new}}, \alpha_{\text{new}}) \)

- But how do we find which constraint to add???
Separation Oracle for Robust LS

- Original problem:
  - Is $(\varepsilon, \alpha)$ feasible?
    - infeasibility $\Rightarrow$ $\varepsilon$ too high for this particular $\alpha$
  - What's the smallest possible $\varepsilon$?

- Standard weighted LS!
  - If result is $\varepsilon$, then we are done!
  - Otherwise found a violated constraint

Are we there yet?

- When do we stop?
- Solve with infinite set of constraints:
  - Obtain $(\varepsilon_{OPT}, \alpha_{OPT})$
- Solve with constraints $\Omega$
  - Obtain $(\varepsilon_{OPT}, \alpha_{OPT})$
  - Optimizing subset of constraints, same objective
  - $\varepsilon \leq \frac{2}{\sqrt{3}} \max_i \min_j \left( \varepsilon_j - \omega_j f_j \right)^2 \omega - \omega$
- If we get any feasible point with infinite constraints
  - E.g.,
  - Bound on how far we are from optimal solution:
Constraint Generation: The General Case

- Given an LP with (possibly infinitely) many constraints:
  - Start with some subset of the constraints
  - Solve LP to find a solution with new subset of the constraints:
    - Separation oracle:
      - If \( x \) is feasible:
        - \( x \) is optimal
      - If \( x \) is infeasible:
        - \( x \) is violated constraint
        - Add violated constraint to set
    - (It is also possible to remove (some or all) inactive constraints, in addition to adding violated constraints)
      - Makes LP solver step faster
      - But requires more outer loop iterations
      - Trade-off is application specific

Bound on optimal solution - General case

- Problem with many constraints:
  - Some relaxation:
    - E.g., only subset of constraints
      - A subset \( \mathbf{A} \) of the rows
      - \( a \subset \mathbf{A} \) of the rows
      - If you can obtain some feasible point for the original problem:
        - From \( \mathbf{x} \) you can somehow find \( \mathbf{x}_* \)
          - Such that \( \mathbf{A}\mathbf{x}_* \leq \mathbf{b} \)
      - Bound on the optimal solution:
        - \( \mathbf{c}^T \mathbf{x}_* \geq \mathbf{c}^T \mathbf{x}_0 \geq \mathbf{c}^T \mathbf{x} \)
Why constraint generation converges

- LP with many many constraints: but not infinite
- Solve with subset of constraints: (also called “cutting planes”)
- Relaxed problem, bound on objective:
- If solution $x_{\Omega}$ is feasible wrt all constraints: $c^T x = c^T x_{opt}$
- If solution $x_{\Omega}$ is infeasible wrt all constraints: add a constraint that forces the LP answer $x_n$ to be in $\Omega$

Practicalities of Constraint Generation

- Constraint generation converges in a finite number of iterations if the original set is finite
  - Can’t guarantee fast rate in general, similar to simplex algorithm (there are special cases with good rates)
  - Infinite case: will get arbitrarily close, but not necessarily to the optimum
- Idea of using relaxations to obtain bounds is very useful in general
  - E.g., useful in duality (more later in the semester)
- Separation oracle:
  - Must find some violated constraint
  - If we find most violated constraint, usually faster
  - Also very useful for proving that LPs can be solved in polytime (ellipsoid algorithm, more later)
- Constraint generation is extremely useful in practice
  - Often, e.g., robust LS, we have a poly-time separation oracle, even if there are exponentially or infinitely many constraints
  - Even if polynomially many constraints, a fast oracle can make constraint generation faster than using a standard solver
- Constraint generation can be useful for solving general convex problems, not just LP
- Remember: most LP solvers allow you to start from previous solution (the one found with fewer constraints)
- Make sure you do this, otherwise approach will be much much much slower
Constraint generation and duality

- Primal problem with many constraints:
  \[ \max_x \sum_j b_j x_j \]
  \[ \text{s.t. } \sum_i a_{ij} x_j \leq c_i, \forall i \in I \]
- Constraint generation: find most important constraints
- What's the dual equivalent?

- Dual:
  \[ \min_y c^T y \]
  \[ \text{s.t. } a_{ij} y_i = b_j, \forall i \in I \]
  \[ y \geq 0 \]
Column generation (aka variable generation)

- Dual problem:
  \[
  \min_y \sum_{i \in I} c_i y_i \\
  \text{s.t.} \sum_{i \in I} a_{ij} y_i = b_j, \ \forall j \in 1, \ldots, m \\
  y_i \geq 0, \ \forall i \in I
  \]

  - Many many variables!!
  - At optimal basic feasible solution
    - Most variables are zero

- Idea:
  - Set most variables to zero
  - Solve problem with other variables
  - Incrementally increase sets of non-zero variables

Solving problem with subset of variables

- Solve problem with subset of variables
  \[
  \min_y \sum_{i \in \Omega} c_i y_i \\
  \text{s.t.} \sum_{i \in \Omega} a_{ij} y_i = b_j, \ \forall j \in 1, \ldots, m \\
  y_i \geq 0, \ \forall i \in \Omega
  \]

  - Rest of variables set to zero

Questions:
- How do we decide what variables to use?
  - Reduced costs, just like Simplex
- How do we decide when we are done?
What variables should we add?

- Same as simplex
  - Solve problem with variables $\Omega$
    - At optimal basic feasible solution, set of basic variables $B$
      - Find submatrix corresponding to basic variables $A_B$
        - Cost of these variables $c_B$
          $$y_B = A_B^{-1} b$$
          $$x_B = \begin{pmatrix} c_B \\ \vdots \\ \vdots \end{pmatrix}$$
          $$A = \begin{pmatrix} a_{1,1} & \cdots & a_{1,n} \\ \vdots & & \vdots \\ a_{m,1} & \cdots & a_{m,n} \end{pmatrix}$$
          - Reduced cost for each potential new variable $y_i$, for $i \in I$:
            - If all are positive?
            - Otherwise:
              - Add var $i$ such that $\bar{c}_i < 0$, usually $\arg \min_i \bar{c}_i$.
              - Guaranteed to converge to optimal solution

Column generation summary

- Dual of constraint generation
- Also useful for problems with infinitely many variables
- Some problems
  - Have efficient separation oracles (In these, constraint generation is useful)
  - Have efficient variable generation oracles (In these, column generation is useful)
- Both methods can be useful in polynomially large problems
  - E.g., when constraint matrix is too large to fit in memory
    - By incrementally solving the problem, bound amount of memory needed at each iteration
- If you have many many variables and constraints
  - Can use a combination of constraint and column generation