CUTTING-PLANE METHODS WITH APPLICATIONS IN CONVEX OPTIMIZATION AND ACTIVE LEARNING

DAVID WU

1. Introduction

1.1. **Outline.** In this report we present a general cutting-plane algorithm for the problem of finding a point in a target set contained in a bounded convex polyhedron. We then show how this general algorithm can be specialized to (i) solve convex optimization problems, giving the so-called analytic center cutting plane method (ACCPM); and (ii) solve an active learning problem in the context of binary linear classification. Crucial to these algorithms is finding the approximate center of a bounded convex polyhedron. The center of a bounded convex polyhedron has no clear definition, so we present several definitions from the literature: (i) the analytic center (AC); (ii) the Chebyshev center (CC); and (iii) the center of gravity (CG). The AC and CC are used in the algorithms we present while the CG is of theoretical interest. Therefore, we explain how the AC and CC can be computed. Furthermore, we present an experiment involving training logistic regression using the ACCPM and several experiments involving the active learning algorithm on real data. The code and notebooks corresponding to this report can be found in the project's GitHub repository [1].

1.2. **Related and background works.** Our exposition of the material on the general cutting-plane algorithm, computing the AC and ACCPM closely follows notes on the topic by S. Boyd et al. [BVS08] [BV11] [BDV15]. S. Boyd's textbook on convex optimization [BV04] was an invaluable source for background knowledge also. Our exposition of the material on active learning closely follows a paper by U. Louche and L. Ralaivola [LR15].

1.3. **Notation and conventions.** Throughout this paper (unless otherwise stated) we will use the following notation and conventions:

- we use the notation $Ax \preceq b$ to represent a system of m linear inequalities $a_i^T x \leq b, i = 1, \ldots, m$. Therefor the rows of A and entries of b are given by a_i and b_i respectively, for $i = 1, \ldots, m$;
- a polyhedron $\mathcal{P}$ is assumed to be defined by linear inequalities $Ax \preceq b$. That is, $\mathcal{P} \coloneqq \{x \mid Ax \preceq b\}$.

[1]The GitHub repository is `https://github.com/davidjwu/mclass-sky/tree/master/projects/david`

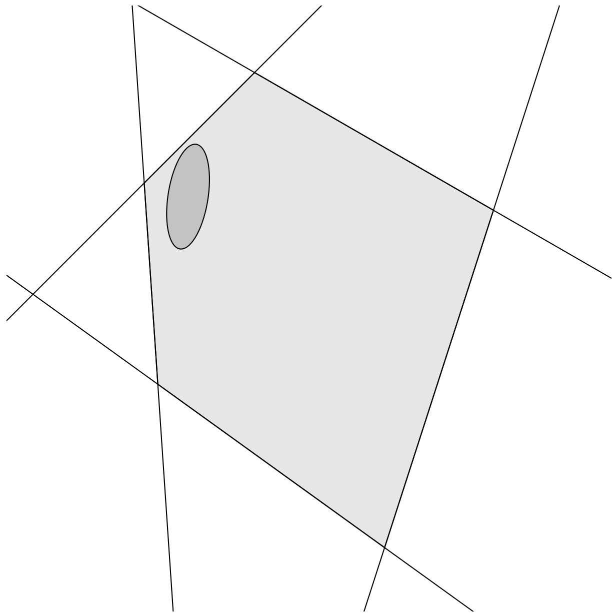


FIGURE 1. The initial polyhedron given by five linear inequality constraints. The oval set is the target set.

- we make no distinction between a system of linear inequalities $Ax \preceq b$ and the polyhedron $\mathcal{P}$ it defines. The two are synonymous; and
- we assume the solution set of $Ax \preceq b$ (and therefore the polyhedron $\mathcal{P}$ it defines) is bounded.

2. The general cutting-plane algorithm

In this section we introduce the concept of cutting-planes and in doing so pave the way to the *general cutting-plane algorithm* (Algorithm 1). This will provide a framework which we can then specialize for the purposes of convex optimization and active learning. Our exposition closely follows S. Boyd and L. Vandenberghe's notes on this topic [BV11, Sections 1-3].

2.1. Introduction to cutting-planes.

Introduction to cutting-planes. The general cutting-plane algorithm address the problem of how to efficiently find a point in a set $X \subset \mathbb{R}^n$, which we call the *target set*, known to be contained in a bounded convex polyhedron $\mathcal{P}_0$ defined by linear inequalities $a_i^T x \leq b, i = 1, \ldots, m$.

The strategy used by this algorithm is to iteratively make *cuts* in the polyhedron $\mathcal{P}_0$ until a point in X is found. Formally, starting at the 0-th iteration we find a point $x^{(1)}$, which we call a *query point*, in the current (or 0-th) iteration's polyhedron $\mathcal{P}_0$. We then check $x^{(1)}$ for membership in the target set X. If $x^{(1)}$ is a member of X then we are done. If $x^{(1)}$ is not a member of X, then we construct a halfspace $a_1^T x \leq b_1$, which we call a *cutting plane*, that separates, or rather *cuts* $x^{(1)}$ from the current iteration's polyhedron $\mathcal{P}_0$. (Cuts are allowed to pass directly through $x^{(1)}$.) We then end the 0-th iteration by defining the polyhedron $\mathcal{P}_1$ of the 1-th iteration to be the union of the 0-th polyhedron and the $1-th$ cutting plane. In the 1-th iteration we then repeat this process with 1-th polyhedron $\mathcal{P}_1$. Formally:

Algorithm 1 (Conceptual cutting-plane algorithm)**.**

```
given a polyhedron $\mathcal{P}_0 = \{x \mid Ax \preceq b\}$ known to contain a target set $X$.
$k := 0$.
repeat
    Choose a query point $x^{(k+1)} \in \mathcal{P}_k$.
    if $x^{(k+1)} \in X$:
        return $x^{(k+1)}$.
    else:
        Construct a cutting-plane $\{x \mid a_{k+1}^T x \le b_{k+1}\}$
        Update $\mathcal{P}_k$ by adding the cutting-plane:
        $\mathcal{P}_{k+1} := \mathcal{P}_k \cap \{x \mid a_{k+1}^T x \le b_{k+1}\}$.
    if $\mathcal{P}_{k+1} = \emptyset$, quit.
    $k := k + 1$.
```

For the purposes of providing a general framework, we employ a *oracle model*, which in our case we simply mean that there is an function that is able to provide us with the necessary information, such as cutting-plane data. We refer the interested reader to S. Boyd and L. Vandenberghe's textbook [BV04, Section 4.1.4].

2.2. **Deep cuts and neutral cuts.** We make a distinction between two types of cuts, deep cuts and shallow cuts. A *deep cut with respect to the current iteration's query point* $x^{(k+1)}$ is a cut that completely separates that point from the next iteration's polyhedron $\mathcal{P}_{k+1}$ while a *neutral cut with respect to the current iteration's query point* $x^{(k+1)}$ is a cut such that that point lies on the boundary of $\mathcal{P}_{k+1}$. (That is, a cut that passes directly through the query point $x^{(k+1)}$.)

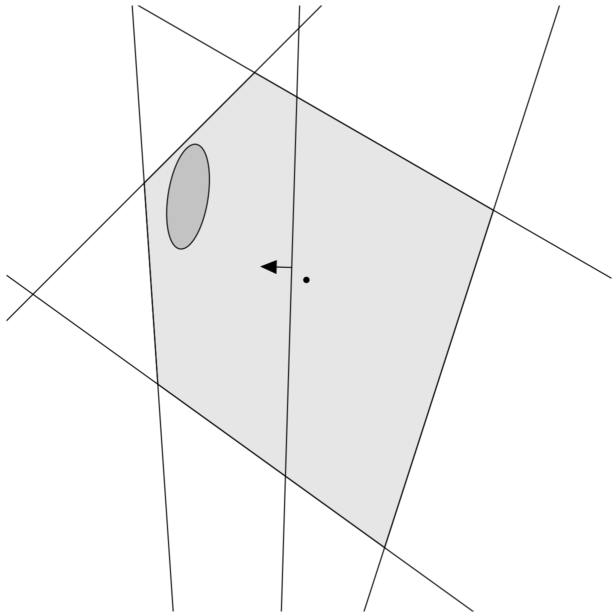


Figure 2. A deep cut.

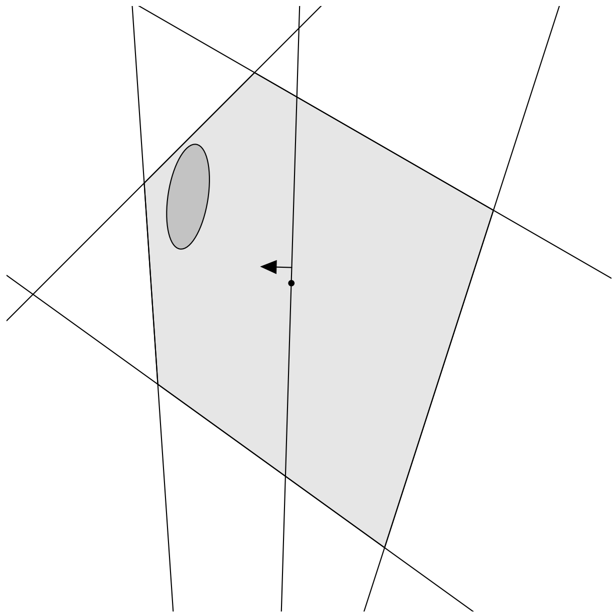


Figure 3. A neutral cut.

2.3. **Adapting the general cutting-plane algorithm for applications.**

To specialize the general cutting-plane algorithm we must specify

1. the target set X;
2. how query points are chosen;
3. how cutting-planes are constructed; and
4. stopping criterion for the algorithm,

in addition to any details the problem at hand demands.

By choosing each of these components appropriately we will be able to adapt the general cutting-plane algorithm for convex optimization and active learning. For instance, for convex optimization problems X is typically taken to be some ϵ-suboptimal set while for active learning we take X to be the version space, that is, the set of classification vectors for which the corresponding linear predictors make no mistake on the training data set.

Moreover, choosing each of these components also effects the efficiency of the algorithm. For instance, in the applications we consider we will choose the query point to be an approximation of the current polyhedron's center, for which there are several choices of varying computational difficulty.

For specializing the general algorithm to both convex optimization and active learning we will apply the *heuristic* that at each iteration we should try to choose the query point and construct the cutting-plane that enable us to reduce the volume of the current polyhedron as much as possible. Intuitively, if the chosen query point lies close to the approximate center of the current iteration's polyhedron then the resultant cut should significantly reduce the current iteration's polyhedron. We follow this strategy.

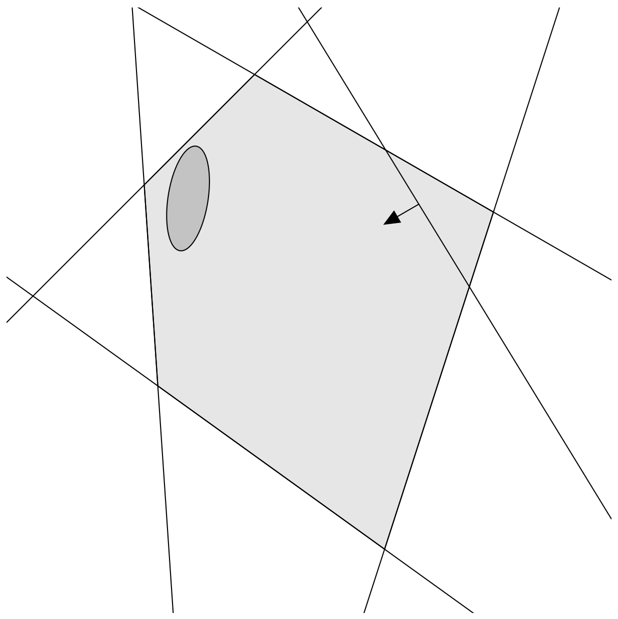


Figure 4. A cutting-plane with undesirable volume reduction. Figure 2.2 shows a cutting-plane with desirable volume reduction.

3. Approximating the center of a convex polyhedron

In this section we will make the notion of approximate center precise. We present the concepts of (i) the analytic center (AC); (ii) the Chebyshev center (CC) and (iii) center of gravity (CG) to do this. Our exposition closely follows S. Boyd and L. Vandenberghe's lecture notes on this topic [BV11, Section 4].

3.1. **The center of gravity.**

The *center of gravity* (CG) of a bounded subset C of $\mathbb{R}^n$ with non-empty interior is defined to be

$$\mathbf{CG}(C) = \frac{\int_C x dx}{\int_C dx}.$$

The CG is a very elegant formulation of the center of a polyhedron and this is reflected by the fact [BV11, Section 4.2] that for a polyhedron $\mathcal{P}_k$ if we take any cutting-plane passing through the center of gravity of the current polyhedron $\mathbf{CG}(\mathcal{P}_k)$ then

$$\frac{\mathbf{vol}(\mathcal{P}_{k+1})}{\mathbf{vol}(\mathcal{P}_k)} \leq 1 - \frac{1}{e} \approx 0.63,$$

so each iteration the volume is reduced by roughly 37%. This would arguably be the best choice for the query point each iteration were it not for the downside that the CG is extremely difficult to compute [BV11, Section 4.2]. Although the CG is not practically useful, it can be used to theoretically justify using approximations of the CG, such as the CC, to choose query points [LR15, Section II.C]. We will say no more about the CG and refer the interested reader to S. Boyd and L. Vandenberghe's notes [BV11, Section 4.2] and a paper by U. Louche and L. Ralaivola [LR15, Section II.C].

3.2. **The analytic center and Chebyshev center.** In this section we discuss the concepts of the analytic center (AC) and Chebyshev center (CC). In Section 5 we will use the AC with the method of constructing cutting-planes outlined in Section 5.2 to specialize the general algorithm to attain the ACCPM. In Section 6 the active learning algorithm we present will be compatible with both the AC and CC.

The *analytic center* (AC) of a polyhedron $\mathcal{P}$ defined by linear inequalities $Ax \preceq b$, denoted by $\mathbf{AC}(\mathcal{P})$, is the solution of the minimization problem

$$\min_{\mathbf{dom}\,\phi} \phi(x) := -\sum_{i=1}^{m} \log\,(b_i - a_i^T x), \tag{1}$$

where

$$\mathbf{dom}\,\phi = \{x \mid a_i^T x < b_i, i = 1, \ldots, m\}.$$

We call ϕ the *logarithmic barrier* (or *log barrier*) *for the linear inequalities* $Ax \preceq b$. The AC can be computed using a infeasible start Newton method, which we describe in Section 4.1.

The *Chebyshev center* (CC) of a polyhedron $\mathcal{P}$ defined by linear inequalities $Ax \preceq b$, denoted by $\mathbf{CG}(\mathcal{P})$, is defined to be the center of the largest Euclidean ball that is contained in $\mathcal{P}$. The CC can be computed by formulating it as a linear program (LP), which we do in Section 4.5.

4. Computing the analytic center and Chebyshev center

In this section we describe how the AC and CC can be computed.

4.1. **The analytic center.** Suppose we have a polyhedron $\mathcal{P}$ defined by inequalities $Ax \preceq b$ and want to solve for the analytic center $\mathbf{AC}(\mathcal{P})$ of $\mathcal{P}$. We observe that the log barrier function ϕ of $\mathcal{P}$ has the properties that

(1) it is differentiable;
(2) it is convex;
(3) it has bounded domain, as we assume that $\mathcal{P}$ is bounded; and therefore
(4) it is bounded below,

meaning that it has a unique minimizer exists which we may attain by solving

$$\nabla\phi(x) := -\sum_{i=1}^{m} \frac{1}{b_i - a_i^T x} a_i = 0.$$

Solving this analytically is not practical. (We refer the interested reader to the notebook `test_accpm_toy.ipynb` for sample calculations for a simple problem that make this difficulty apparent.) Therefore we instead choose to solve for the analytic center $\mathbf{AC}(\mathcal{P})$ of $\mathcal{P}$ using a infeasible start Newton method.

4.2. **An alternative approach for computing the analytic center.** The infeasible start Newton method is one of two approaches suggested by S. Boyd, L. Vandenberghe and J. Skaf in their notes on the ACCPM [BVS08, p. 3]. It has the advantage that the starting point need not be contained in $\mathbf{dom}\,\phi$. The other approach they suggest is to (i) use a phase I optimization method [BV04, Section 11.4] to find a point in $\mathbf{dom}\,\phi$ (or determine that $\mathbf{dom}\,\phi$ is empty); and then (ii) use this as a starting point for a standard Newton method.

In this project we initially tried to solve for the AC using this approach. After failing to do so over a number of weeks we decided to use the infeasible start Newton method instead. As this was successful we did not resume work on this approach. It is unclear whether or not this failure was due to theoretical or technical issues. In hindsight, after implementing the infeasible start Newton method correctly, we have the following suggestions for the reader that might be interested in doing so:

- when initializing the algorithm and after each iteration ensure the linear inequality constraints have been normalized. That is, if we have the linear inequality constraints $a_i^T x \leq b_i, i = 1, \dots, m, m+1, \dots, m+k$ corresponding to the m linear inequalities that define $\mathcal{P}_0$ and having completed k iterations, then $\frac{a_i}{\|a_i\|}^T x \leq \frac{b_i}{\|a_i\|}, i = 1, \dots, m, m+1, \dots, m+k$ define the same linear inequalities;
- consider first making the substitution $y = b - Ax$;
- be wary of numerical issues occurring, particularly those that involve numbers in a neighborhood of 0 where rounding to 0 may be necessary or where values that should be positive become negative and vice-versa.

4.3. **The infeasible start Newton method.** To use the infeasible start Newton method to compute the AC we first reformulate the problem (1) as

$$\begin{array}{ll} \text{minimize} & -\sum_{i=1}^{m} \log y_i \\ \text{subject to} & y = b - Ax, \end{array} \tag{2}$$

in the variables $x \in \mathbb{R}^n$ and $y \in \mathbb{R}^m$. The advantage of the infeasible start Newton method is that we can start it from any $x \in \mathbb{R}^n$ and any $y \succ 0$. S. Boyd, L. Vandenberghe and J. Skaf [BVS08, Section 2] recommend $x := x_{\text{previous}}$, the query point from the previous iteration and y as

$$y_i := \begin{cases} b_i - a_i^T x & \text{if } b_i - a_i^T x > 0 \\ 1 & \text{otherwise.} \end{cases}$$

Before proceeding, we define

$$g \coloneqq \nabla\left(-\sum_{i=1}^{m}\log y_i\right) = -\begin{bmatrix}\frac{1}{y_1}\\ \vdots \\ \frac{1}{y_m}\end{bmatrix}$$

$$H \coloneqq \nabla^2\left(-\sum_{i=1}^{m}\log y_i\right) = \mathbf{diag}\left(\begin{bmatrix}\frac{1}{y_1^2}\\ \vdots \\ \frac{1}{y_m^2}\end{bmatrix}\right).$$

Now, we define the *dual residual* and *primal residual* for the problem (2) as

$$r_d \coloneqq \begin{bmatrix}A^T v\\ g+v\end{bmatrix} \qquad r_p \coloneqq y + Ax - b,$$

and define $r(x,y,v) \coloneqq (r_d(x,y,v), r_p(x,y,v))$ also.

The *Newton step at a point* (x,y,v) is given by the expressions

$$\begin{aligned}\Delta x &= (A^T H A)^{-1}(A^T g - A^T H r_p)\\ \Delta y &= -A\Delta x - r_p\\ \Delta v &= -H\Delta y - g - v.\end{aligned}$$

In our implementation we computed Δx using a *Cholesky factorization* [BV04, Appendix C.3].

The infeasible start Newton method is then

Algorithm 2 (Infeasible start Newton method)**.**

```
given a starting point $x, y \succ 0$, tolerance $\epsilon > 0, \alpha \in (0, \frac{1}{2}), \beta \in (0,1)$.
$v \coloneqq 0$.
Compute residuals $r \coloneqq (r_d, r_p)$.
repeat
    1. Compute the Newton step $(\Delta x, \Delta y, \Delta v)$.
    2. Backtracking line search on $\|r\|$.
       $t \coloneqq 1$.
       while  $y + t\Delta y \not\succ 0$:
           $t \coloneqq \beta t$.
       while $\|r(x + t\Delta x, y + t\delta y, v + t\Delta v\| > (1 - \alpha t)\|r(x,y,v)\|$:
           $t \coloneqq \beta t$.
    3. Update $x \coloneqq x + t\Delta x, y \coloneqq y + t\Delta y, v \coloneqq v + t\Delta v$.
until $y = b - Ax$ and $\|r(x,y,v)\| \le \epsilon$.
```

4.4. **Implementation details.** Our implementation can be found in the repository of this project with file name `accpm.py` and tests of this can be found in the notebooks `test_accpm_log_regression.ipynb` and `test_accpm_toy.ipynb`.

As suggested by S. Boyd, L. Vandenberghe, and J. Skaf in their lecture notes [BVS08, Section 2] we found $\alpha \coloneqq 0.01$ and $\beta \coloneqq 0.7$ to work quite effectively.

We computed the Cholesky factorization using the function `scipy.linalg.cholesky`. We found that if the polyhedron was too large, often `scipy.linalg.cholesky` would encounter numerical difficulty.

4.5. **The Chebyshev center.** Our strategy for computing the CC is to formulate it as a linear program (LP). Formally, let $\mathcal{B}(x_c, r)$ be the ball centered at x_c with radius $r > 0$. Then the CC of a polyhedron $\mathcal{P}$ given by the linear inequalities $a_i^T x \leq b_i, i = 1, \dots, m$ is given by the maximization problem

$$\max_{\mathcal{B}(x_c,r)\subset\mathcal{P}} r$$

in the variables x_c and r with the constraint that $r > 0$.

To see how we can formulate this as a LP suppose there exists a ball $\mathcal{B}(x_c, r)$ that is contained in the single half-space $a_i^T x \leq b_i$. Then this is equivalent to having the existance of some x_c that satisfies

$$a_i^T(x_c + u) \leq b_i.$$

for all u with $\|u\| \leq r$. If this is true, then this implies x_c satisfies

$$a_i^T(x_c + u) = a_i^T x_c + a_i^T u \leq a_i^T x_c + r\|a_i\| \leq b_i$$

since at most for all u that satisfy $\|u\| \leq r$ we have $a_i^T u \leq \|a_i\|\|u\| \leq r\|a_i\|$. (This is achieved for $\frac{r \cdot a}{\|a\|}$.) Conversely, if we have a x_c and $r > 0$ that satisfies $a_i^T x_c + r\|a_i\| \leq b_i$, then it is clear that the circle $\mathcal{B}(x_c, r)$ lies in the half-space $a_i^T x \leq b_i$.

Therefore, $\mathcal{B}(x_c, r) \subset \mathcal{P}$ if and only if

$$a_i^T x_c + r\|a_i\| \leq b_i$$

for $i = 1, \dots, m$ meaning we can formulate the problem of finding a CC as the LP

$$\begin{aligned} &\text{maximize} && r \\ &\text{subject to} && a_i^T x_c + r\|a_i\| \leq b_i, \end{aligned} \tag{3}$$

4.6. **Implementation details.** The implementation of the *CC* computation can be found in `active.py` with tests in the note book `test_chebyshev.ipynb`. We computed it using the function `scipy.optimize.linprog`.

5. The analytic center cutting plane method (ACCPM) for solving convex optimization problems

In this section we specialize the general cutting-plane method to solve convex optimization problems. This involves specifying (i) the target set X; (ii) how query points are chosen; (iii) how cutting-planes are constructed; and (iv) stopping criterion for the algorithm. The specialized algorithm is referred to as the ACCPM

5.1. **Convex optimization problems.** We consider inequality constrained convex optimization problems of the form

$$\begin{array}{ll} \text{minimize} & f_0(x) \\ \text{subject to} & f_i(x) \leq 0, \quad i = 1, \ldots, m \end{array} \tag{4}$$

where $f_0, \ldots, f_m$ are convex differentiable functions. We remark that the exposition here holds if the $f_1, \ldots, f_m$ are non-differentiable functions without much change given instead of considering gradients we consider subgradients. We refer the interested reader to [BDV15].

5.2. **Specifying the general cutting-plane method.**

5.2.1. *The target set.* We typically take the target set X to be the ϵ-suboptimal set for some fixed $\epsilon > 0$. The important property is that there must exist some bounded polyhedron $\mathcal{P}$ that contains X.

5.2.2. *Choosing query points and constructing cutting-planes.* Suppose we are at iteration k with polyhedron $\mathcal{P}_k$. Then we choose the query point $x^{(k)}$ to be $\mathbf{AC}(\mathcal{P}_k)$, the analytic center of the current iteration's polyhedron $\mathcal{P}_k$. If we have determined that $x^{(k)}$ is not in the target set X then we check $x^{(k)}$ for feasibility, that is, whether or not it satisfies all inequality constraints $f_i(x) \leq 0, \;\; i = 1, \ldots, m$. Feasibility determines how we construct the cutting-plane for $x^{(k)}$.

The construct of cutting-planes relies on the following result [BV04, Section 3.1], and particularly, the inequality: suppose f is differentiable. Then f is convex if and only if $\mathbf{dom}\, f$ is convex and

$$f(y) \geq f(x) + \nabla f(x)^T (y - x),$$

for all $x, y \in \mathbf{dom}\, f$.

Suppose $x^{(k)}$ is not a feasible point. Then there exists some index $j \in \{1, \ldots, m\}$ with $f_j(x) > 0$, that is, the j-th constraint has been violated. The cutting plane we will define comes from the following discussion. As f_j is convex and differentiable we have

$$f_j(x) \geq f_j(x^{(k)}) + \nabla f_j(x^{(k)})^T (z - x)$$

for all x in the problem's domain. Therefore,

$$f_j(x^{(k)}) + \nabla f_j(x^{(k)})^T (z - x) > 0 \implies f_j(x) > 0,$$

or equivalenty,

$$f_j(x) \leq 0 \implies f_j(x) + \nabla f_j(x^{(k)})^T (z - x) \leq 0$$

for all x in the problem's domain. That is all feasible points must satisfies $f_j(x) + g^T(z - x) \leq 0$. So the cutting plane is defined to be

$$\{x \mid f_j(x) + \nabla f_j(x^{(k)})^T (x - x^{(k)}) \leq 0\}.$$

Observe also that substituting $x^{(k)}$ into this inequality reduces to $f_j(x^{(k)}) \leq 0$ but in actuality $f_j(x^{(k)}) > 0$, so therefore this is a deep cut with respect to $x^{(k)}$, which is what we want. To summarize, using the notation of the general cutting-plane algorithm we have

$$a := \nabla f_j(x^{(k)}), \quad b := \nabla f_j(x^{(k)})^T x^{(k)} - f_j(x^{(k)}). \tag{5}$$

On the other hand, suppose $x^{(k)}$ is a feasible point. Observe that if a point x satisfies $\nabla f_0(x^{(k)})^T(x-x^{(k)}) > 0$, then this implies $f_0(x) \geq f_0(x^{(k)})$. That is, x is not optimal. Therefore, we might define the cutting-plane as

$$\{x \mid \nabla f_0(x^{(k)})^T(x - x^{(k)}) \leq 0\},$$

but this is not ideal as this is a neutral cut with respect to $x^{(k)}$.

To attain a deep cut we define f_{best} as

$$f_{\text{best}} := \min\{f_0(x)\}$$

where $x \in \{x^{(1)}, \ldots, x^{(k)}\} \cap \{\text{feasible points}\}$ and define the cutting-plane to be

$$\{x \mid \nabla f_0(x^{(k)})^T(x - x^{(k)}) + f_0(x^{(k)}) - f_{\text{best}} \leq 0\}.$$

To summarize, using the notation of the general cutting-plane algorithm we have

$$a := \nabla f_0(x^{(k)}), \quad b := \nabla f_0(x^{(k)})^T x^{(k)} - f_0(x^{(k)} + f_{\text{best}}. \tag{6}$$

Stopping criterion. In regards to the stopping criterion there a number of options. S. Boyd and L. Vandenberghe describe a way to compute the lower-bound of f_0, which can therefore be used to determine whether or not a query point is contained in the ϵ-suboptimal target set [BV11, Section 7]. If it is known that the objective function has a unique minimizer (and no maximizers), then another option is to check whether or not $\left\|\nabla f_0(x^{(k)})\right\| \leq \epsilon$ for some given tolerance $\epsilon > 0$.

In summary we have

Algorithm 3 (Analytic center cutting-plane method (ACCPM))**.**

```
given an initial polyhedron $\mathcal{P}_0 = \{z \mid Cz \preceq d\}$ known to contain $X$.
$k := 0$.
repeat
    Compute $x^{(k+1)}$, the analytic center of $\mathcal{P}_k$.
    if $x^{(k+1)} \in X$, quit.
    else:
        Check whether or not $x^{(k+1)}$ is feasible.
        Update $\mathcal{P}_k$ by adding the corresponding cutting-plane:
            $\mathcal{P}_{k+1} := \mathcal{P}_k \cap \{z \mid a_{k+1}^T z \leq b_{k+1}\}$.
    if $\mathcal{P}_{k+1} = \emptyset$, quit.
    $k := k + 1$.
```

5.3. **Implementation details.** Our implementation can be found in the file `accpm.py` in the GitHub repository.

6. Cutting plane methods for active learning

In this section we introduce active learning in the context of a binary linear classification problem. We then specialize the general cutting plane algorithm to attain a active learning algorithm for these types of problems. Our treatment here and the algorithm we present closely follows Section D of U. Louche and L. Ralaivola's paper [LR15]. We extend their work to include the AC.

6.1. Active learning.

Suppose a learner is given a pool of data $\mathcal{D}$, where each data point consists of a input vector and corresponding label pair. In the linear classification active learning problem we further suppose that all labels are initially unknown to the learner and individual labels can be obtained at some cost. Given a restriction on the number of labels the learner can obtain and that the learner can only label one label at a time, the aim is to find a classifier that will accurately map input vectors to labels [Das11].

S. Dasgupta's 2011 paper [Das11] proposes two common heuristics for solving this problem:

(1) label data points that reduce the space of candidate classifiers (or equivalently, in our case, classification vectors) as efficiently as possible; and
(2) exploit clusters in the pool of data.

We will pursue intuition (1) as it is similar to the approach taken by cutting-plane algorithm.

6.2. Linear classification problems.

In a linear classification problem we are given a training set $\mathcal{D} = \{(x_n, y_n)\}_{n=1}^{N}$ with input vectors $x_n \in \mathcal{X} := \mathbb{R}^d$ and target variables $y_n \in \mathcal{Y} := \{+1, -1\}$. The aim is to find a classification vector $w \in \mathcal{X}$ whose corresponding linear predictor, defined by

$$f_w(x) = \mathbf{sign}(w^T x)$$

for all $x \in \mathcal{X}$, makes no mistakes on $\mathcal{D}$. That is, we want to find an element of the version space

$$\mathcal{W} := \{w \in \mathcal{X} \mid y_n(w^T x_n) \geq 0 \text{ for } n = 1, \ldots, n\}.$$

In the context of active learning, the problem we consider is this: if $\mathcal{D}$ is the unlabeled pool of data where each data point has equal labeling cost then how should we decide which data point to label next (or first)?

To address this, suppose we have identified that a subset W' of W is contained in a bounded set $\mathcal{C}$ of candidate classification vectors. Applying heuristic (1) from Section 6.1 we would want to label a data point (x, y) that allows us to obtain a set $\mathcal{C}'$ contained within $\mathcal{C}$ of reduced volume. Cutting plane methods are well suited for this task.

6.3. **From general cutting plane algorithm to active learning algorithm.** As observed in Section 2.3 to adapt the general cutting plane algorithm some work is required. Particularly, in this case we must specify

(1) the target set;
(2) the initial polyhedron containing the target set;
(3) how to choose a query point;
(4) how to choose a point to label;
(5) how cutting-planes are constructed; and
(6) stopping criterion for the algorithm,

which we undertake in this section.

The target set and the initial polyhedron containing the target set. The naive approach for this is to take $\mathcal{W}$. However, this is problematic as if $w \in \mathcal{W}$ then $cw \in \mathcal{W}$ for all $c > 0$ since $y_n(w^T x_n) \geq 0$ implies $y_n(cw^T x_n) \geq 0$ for all $n = 1, \ldots, N$. Therefore, most of the time $\mathcal{W}$ will be unbounded when we require it to be contained in some bounded initial polyhedron.

The solution is in the problem, however, as if $w \in W$ then $\frac{w}{\|w\|} \in W$ also. Therefore, without loss of generality we may take the target set to be

$$\mathcal{W}^\star := \{w \in \mathcal{W} \mid \|w\| \leq 1\},$$

which we refer to as a version space also.

The initial polyhedron, which we refer to as the initial candidate space $\mathcal{C}_0$ is then straightforwardly chosen to be the d-dimensional hyper-cube centered at the origin. U. Louche and L. Ralaivola take $\mathcal{C}_0$ to be the unit sphere centered at the origin [LR15, Section II.B]. Our choice is motivated by the AC being defined *only* for polyhedrons and the fact that the CC of a polyhedron can be formulated as a linear program.

Choosing query points. In their paper U. Louche and L. Ralaivola consider both the case where the query point is the CC and CG [LR15, Section III.D]. For purposes of comparison, we attempt to replicate the case where the query point is the CC case. In this report we consider the AC case also.

Choosing a point to label. We follow the approach taken by U. Louche and L. Ralaivola's paper [LR15, Section III.D] which is summarized by their Query function. (To be consistent with our terminology, we will call this function Label.)

This involves uniformly sampling M points from the current iteration's candidate space. U. Louche and L. Ralaivola provide a number of references for uniform sampling methods [LR15, Section III.D]. In our implementation we chose to use rejection sampling to achieve this, which is known to be very inefficient.

Overall, for the purposes of this project, and partially because the U. Louche and L. Ralaivola's paper [LR15, Section III.D] commented little on this function, we took the Query function for granted.

Constructing cutting-planes and stopping criterion. Suppose we have so far labeled points $(x_{n_1}, y_{n_1}), (x_{n_2}, y_{n_2}), \ldots, (x_{n_k}, y_{n_k})$ and have computed $w^{(k+1)}$ and labeled $(x_{n_{k+1}}, y_{n_{k+1}})$. Then, we first check if $w^{(k+1)}$ is able to correctly predict the label of the point $(x_{n_{k+1}}, y_{n_{k+1}})$. If it is, then we finish the current iteration of the algorithm without returning a cutting-plane. If it is incorrect, that is, if $y_{n_{k+1}}({w^{(k+1)}}^T x_{n_{k+1}}) < 0$, then we are able to make a *deep* cut with respect to $w^{(k+1)}$ through the cutting-plane

$$\{w \mid y_{n_{k+1}}(w^T x_{n_{k+1}}) \geq 0\}.$$

There are several choices for the stopping criterion of this algorithm, not necessarily mutually exclusive. U. Louche and L. Ralaivola's [LR15, Section III.D] propose doing so when the volume of $\mathcal{C}_k$ is small enough in volume. Other options are to check each iteration if $w^{(k+1)}$ is in the version space $\mathcal{W}^\star$, check each iteration if the linear predictor $f_{w^{(k)}}$ has achieved a threshold accuracy, or check if a maximum number of points have been labeled.

In summary, we have

Algorithm 4 (Active learning algorithm)**.**

```
given a pool of data $\mathcal{D}_{\text{testing}} \coloneqq \{(x_n, y_n)\}_{n=1}^N$.
$\mathcal{W}^\star \coloneqq \{w \in \mathcal{W} \mid \|w\| \leq 1\}$.
$\mathcal{C}_0 \coloneqq [-\frac{1}{2}, \frac{1}{2}]^d$.
$\mathcal{D}_0 \coloneqq \mathcal{D}_{\text{testing}}$.
$k \coloneqq 0$.
repeat
    Compute the query point $w^{(k+1)} \coloneqq \textbf{center}(P_k)$.
    if $w^{(k+1)} \in \mathcal{W}^\star$:
        return $w^{(k+1)}$.
    else:
        Label a data point $(x_{n_{k+1}}, y_{n_{k+1}}) \coloneqq \textsc{Label}(\mathcal{C}_k, \mathcal{D}_k)$.
        if $y_{n_{k+1}}({w^{(k+1)}}^T x_{n_{k+1}}) < 0$ :
            Update $\mathcal{C}_k$ by adding the new cutting-plane:
                $\mathcal{C}_{k+1} \coloneqq \mathcal{C}_k \cap \{w \mid y_{n_k}(w^T x_{n_k}) \geq 0\}$.
    $k \coloneqq k + 1$.
    $\mathcal{D}_{k+1} \coloneqq \mathcal{D}_k - \{(x_{n_{k+1}}, y_{n_{k+1}})\}$.
    quit if $\mathcal{C}_{k+1}$ is small enough

function $\textsc{Label}(\mathcal{C}, \mathcal{D})$
    Uniformly sample $M$ points $w_1, \ldots, w_M$ from $\mathcal{D}$.
    $w_{\text{approx}} \coloneqq \frac{1}{M}\sum_{i=1}^M s_i$.
    $x \coloneqq \arg\min_{x_i \in \mathcal{D}}\{w_{\text{approx}}^T x_i\}$
    Obtain the label $y$ of $x$.
return $(x, y)$
```

6.4. **Implementation details.** Our implementation can be found in the GitHub repository in the file `active.py`.

7. Experiments

In this section we present one experiment using the ACCPM to train logistic regression and two experiments that compare the performance of the active learning algorithm with query point chosen as (i) the CC; (ii) the AC; and (iii) a randomly chosen point of the current iteration's candidate space.

7.1. **Computational set-up.**

We ran the experiments on a mid-2012, 13-inch MacBook Air with 1.8 GHz Intel Core i5 processor, 8 GB 1600 MHz DDR3 memory and Intel HD Graphics 4000 1536 MB graphics.

7.2. **The data sets.**

We work with two data sets and produce an experiment for each data set. In both cases we assess the ability of our implementation of Algorithm 4 to find an accurate linear predictor for the data set.

Specifically, we work with the Iris flower data set [2] and the diabetes data set [3]. The Iris flower data set consists of 150 data points split evenly between three classes, corresponding to three different species of Iris flower. Each data point has four features and a class label. It is well known that the class corresponding to the species *Iris setosa* can be linearly separated from the other two classes. Therefore, we consider the problem of linearly separating the class *Iris setosa* from the other two classes. In addition to this, we work only with the first two of four features. The diabetes data set consists of 768 data points split between two classes, corresponding to testing positive (268) for and *not* testing positive (500) for diabetes. Each data point has eight features and a class label. This data set is not linearly separable. For both the Iris flower data set and diabetes data we add a bias variable fixed at one to each input vector therefore giving us two plus one features for each data point and eight plus one features for each data point in each data set, respectively.

In our experiment we random uniformly evenly split the data set into a training set $\mathcal{D}_{\text{training}}$ and testing set $\mathcal{D}_{\text{testing}}$. We take the training set $\mathcal{D}_{\text{training}}$ as the pool of unlabeled data.

7.3. **ACCPM experiment.**

We do binary classification using logistic regression on the diabetes data set. Our model has nine features $x := x^{[0]}, \ldots, x^{[8]}$ where $x^{[0]}$ is the dummy input variable fixed at 1. (The fixed dummy input variable could easily be $x^{[5]}$ or $x^{[8]}$, it's index is unimportant.) We set the basis functions to the simplest choice $\phi_0(x) = x^{[0]}, \ldots, \phi_8(x) = x^{[8]}$. Our model then has the form

$$y(x) = \sigma(w^T x.)$$

[2]Found at `http://archive.ics.uci.edu/ml/datasets/Iris`.

[3]See `https://archive.ics.uci.edu/ml/datasets/Pima+Indians+Diabetes` for details. We use a normalized version found at `http://mldata.org/repository/data/download/csv/diabetes_scale/`

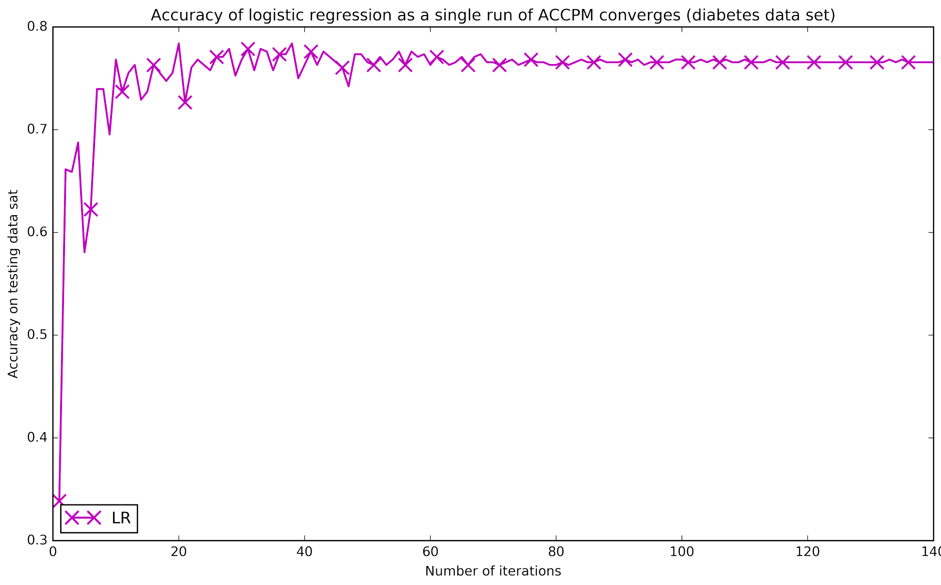
We train our model by finding the parameter vector w which minimizes the (data-dependent) cross-entropy error function [Bis06, Section 4.3.2]

$$E_D(w) = -\sum_{n=1}^{N}\{y_n \ln \sigma(w^T x_n) + (1-y_n)\ln\big(1-\sigma(w^T x_n)\big)\}.$$

The gradient of this function is given by

$$\nabla E(w) = \sum_{i=1}^{N}(\sigma(w^T x_n) - y_n)x_n.$$

We take the initial polyhedron $\mathcal{P}_0$ to be the 9-dimensional hypercube centered at the origin with sides of length 20. We run the algorithm once until the stopping criterion is satisfied. Each iteration we remember the query point, which in the case of the ACCPM corresponds to a parameter vector for logistic regression. Therefore, comparing against $\mathcal{D}_{\text{testing}}$ we can compute an associated accuracy for each iteration. This allows us to measure the performance of the algorithm.



7.4. **Discussion of ACCPM experiment.** The ACCPM successfully converged after 140 iterations. It is re-assuring to know that the ACCPM performed as well as using the model trained using `scipy.optimize`, in both cases achieving an upper-bound of 75% accuracy. Indeed, we see that convergence in accuracy was rapid, which matches our intuition that as the polyhedron becomes smaller, the points within it should be more accurate parameter vectors.

It is interesting to observe that accuracy reached its upperbound well before the stopping criterion was satisfied.

7.5. **Active learning experiments.** Our experimental procedure for both data sets is almost the same. Therefore, we explain it using the general set-up introduced in Section 6.2 where in total we have N data points. (This is useful also for the reader that might want to replicate these experiments.)

The aim of our experimental procedure is to approximate the average change in accuracy from labeling an additional point. To see how we will achive this, we consider a single run of Algorithm 4 on the training set $\mathcal{D}_{\text{training}}$ with the stopping criterion being to label no more than p points. Now, suppose in this single run we label the points such as to produce the sequence $(x_{n_1}, y_{n_1}), (x_{n_1}, y_{n_1}), \dots, (x_{n_p}, y_{n_p})$. Then at each iteration $i = 0, \dots, p-1$ we compute a classification vector $w^{(i+1)}$ which is associated with the sequence of labeled points $(x_{n_1}, y_{n_1}), (x_{n_2}, y_{n_2}), \dots, (x_{n_i}, y_{n_i})$. For each $i = 1, \dots, p$ we can then test the predictions of $f_{w^{(i)}}$ against $\mathcal{D}_{\text{testing}}$ to compute an accuracy $r_i^{(1)} \in [0,1]$. (The reason for the upper-index will be made clear shortly.) Now, as choosing the point to be labeled involves a random process, each time we run the algorithm we may produce a different sequence of weights $w_1, \dots, w_p$. Therefore, we run the algorithm q times and the run we are up to is indicated by the upper-index. Doing this we produce a $q \times p$ matrix of accuracies

$$R := \begin{bmatrix} r_1^{(1)} & r_2^{(1)} & \cdots & r_p^{(1)} \\ r_1^{(2)} & r_2^{(2)} & \cdots & r_p^{(2)} \\ \vdots & \vdots & \ddots & \vdots \\ r_1^{(q)} & r_2^{(q)} & \cdots & r_p^{(q)} \end{bmatrix}.$$

The average accuracy r_i of the algorithm when $i = 1, \dots, p$ points have been labeled can then be computed as

$$r_i := \frac{r_i^{(1)} + \cdots + r_i^{(q)}}{q}.$$

(That is, the average computed over the i-th column.)

We complete this procedure of running the Algorithm 4 10 times, labeling 25 points each time, and computing $r_1, \dots, r_p$ three times in total; one each for the query point each iteration being computed as (i) the CC; (ii) the AC; and (iii) a randomly chosen point of the current iteration's candidate space.

We also analogously complete this process for logistic regression, for purposes of comparison, where for a single run (out of 10) at the 1-th iteration we randomly select a data point (x_{n_1}, y_{n_1}) from $\mathcal{D}_{\text{training}}$ without replacement. We then train logistic regression on the data point (x_{n_1}, y_{n_1}) and use the resultant model and testing data set $\mathcal{D}_{\text{testing}}$ to produce a accuracy $r_0^{(1)}$. On the 2-th iteration we again randomly select another data point (x_{n_1}, y_{n_1}) from $\mathcal{D}_{\text{training}}$ without replacement. We then train logistic regression on the data points $(x_{n_1}, y_{n_1}), (x_{n_2}, y_{n_2})$ and use the resultant model and testing data set $\mathcal{D}_{\text{testing}}$ to produce an accuracy $r_1^{(1)}$. We do this for 24 iterations. We then repeat this another 9 times to produce the matrix R from which $r_1, \dots, r_p$ can be computed.

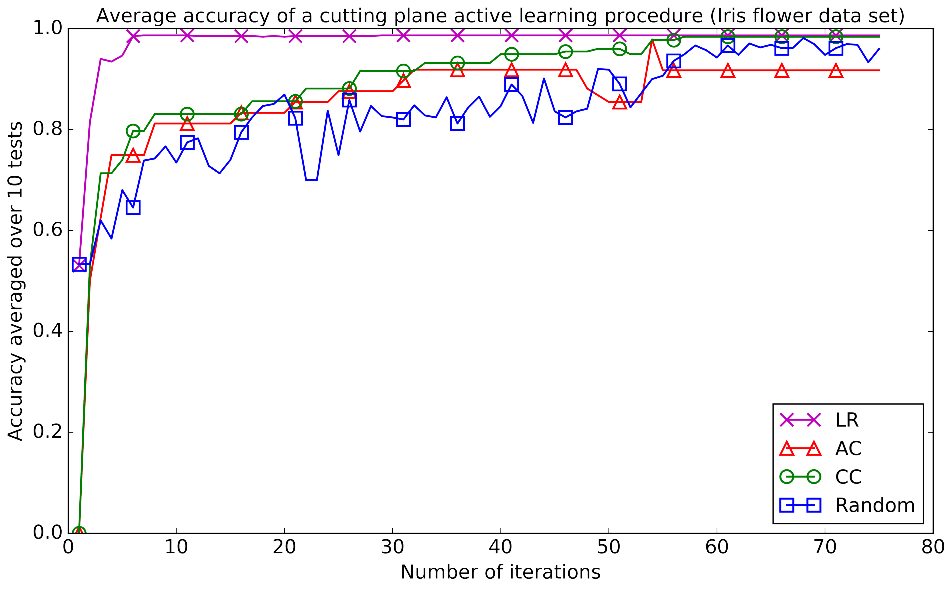
In the experiment using the Iris flower data set we label $p = |\mathcal{D}_{\text{training}}| = 75$ data points each run and run the algorithm $q = 10$ times.

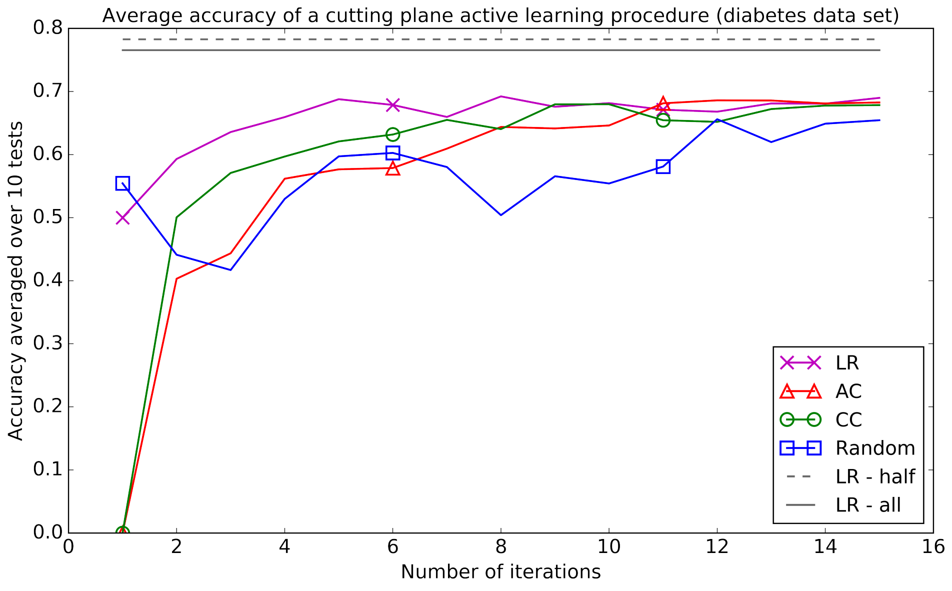
In the experiment using the diabetes data set we label $p = 15$ data points each run and run the algorithm $q = 10$ times.

For the experiment using the diabetes data set we also computed (i) the average accuracy of logistic regression trained on half of the points from $\mathcal{D}_{\text{training}}$, computed by averaging over 10 runs of training logistic regression on a randomly selecting subset of $\mathcal{D}_{\text{training}}$ that is half the size of $\mathcal{D}_{\text{training}}$; and (ii) the accuracy from training logistic regression on the entire training data set $\mathcal{D}_{\text{training}}$

We used our own implementation of logistic regression where the training is completed using the `scipy.optimize.minimize(method='BFGS')` function. It is worth noting that when computing the classification vector for logistic regression the minimization function often failed to converge, although it would still produce a classification vector. We went ahead and used these classification vectors.

7.6. **Implementation details.** The notebooks corresponding to these experiments can be found in the GitHub repository in `experiment_active_iris.ipynb` and `experiment_active_diabetes.ipynb`.


Average accuracy of a cutting plane active learning procedure (Iris flower data set)
Accuracy averaged over 10 tests
Number of iterations
0.0
0.2
0.4
0.6
0.8
1.0
0
10
20
30
40
50
60
70
80
LR
AC
CC
Random


Average accuracy of a cutting plane active learning procedure (diabetes data set)
Accuracy averaged over 10 tests
Number of iterations
0.0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0
2
4
6
8
10
12
14
16
LR
AC
CC
Random
LR - half
LR - all

7.7. **Discussion of active learning experiments.** In both the experiment on the Iris flower data set and the diabetes data set we see that our implementation of logistic regression (LR) trained on a random sample outperformed the active learning algorithm with query point as the AC (AC), CC (CC) and random center (Random). In Section 8 we will give suggestions on how the performance of the active learning algorithm could be improved.

In the experiment with the Iris flower data set we see that overall CC outperformed AC and Random and performed on par with LR after approximately 55 iterations. AC performed similarly to CC, and outperformed Random with the exception that after aproximately 45 iterations the average accuracy of AC dropped signficantly.

Although we have not investigated this, it is possible that this can be attributed to a numerical computing error occurring once the candidate space $\mathcal{C}_k$ becomes very small. This seems a reasonable cause given that the implementation of the AC computation was our own and numerical computing issues did arise during its development. Consistent with this, the smoother behavior of CC could be in part due to relying on the robust `scipy.optimize.linprog` function to compute the CC.

In the experiment with the diabetes data set we first observe that on average logistic regression trained on half the training data set (LR– half) and the whole training data set (LR–all) performed the best but still with average accuracy less than 80%. Indeed, it is interested to observe that LR–half outperformed LR–all, which could possibly be attributed to the data set not being linearly separable. Like the experiment on the Iris flower data set we see that AC and CC outperformed random and CC outperformed AC.

LR significantly outperformed AC and CC on the Iris flower data set and while it also outperformed AC and CC on the diabetes data set this was not as significantly and after roughly 9 iterations, LR, AC and CC performed approximately equally.

In regards to the experiment on the diabetes data set, it is worth noting that we only included data up to labeling at most 15 points as the computation time grew exponentially with the maximum number of points to be labeled. Having said this, there is partial data for labeling at most 30 points, which took approximately 36 hours to compute. For the interested reader, this can be found in the GitHub repository. This running time growth can be attributed to using rejection sampling to sample the points in the Query function. Specifically, random vectors were drawn from a nine-dimensional unit hypercube centered at the origin and were rejected if they did not lie in the current candidate space $\mathcal{C}_k$. We remark that it is known that as the dimension increases, rejection sampling tends to perform exponentially worse. We see this in how computing each of AC, CC and Random in the case of labeling at most 15 points involved generating at least $1,000,000$ random vectors while the case of labeling at most 30 points involved generating at least $1,000,000,000$ random vectors.

8. Conclusion

In this report we showed how a general cutting-plane algorithm can be specialized for the purposes of convex optimization and an active learning problem. This specialization involves specifying (i) the target set; (ii) how query points are chosen; (iii) how cutting-planes are constructed; and (iv) stopping criterion. And as we saw in the case of the active learning problem, these choices effect the performance of the algorithm. Now, the experiments on both the Iris flower data set and diabetes data set LR outperformed both the CC and AC based cutting-plane algorithms. This raises the general question of what options there are to improve the performance of a cutting-plane algorithm? (Specifically, the question I am interested in is what can be done to improve the performance of the active learning algorithm.) In their 2011 notes S. Boyd and L. Vandenberghe [BV11, Section 5] provide a number of suggestions for extending the general cutting-plane algorithm:

(1) each iteration instead of returning a single cutting-plane, we can return *multiple* cutting-planes;
(2) each iteration we can remove (or in other words *drop* or *prune*) inequality constraints that may be redundant;
(3) we can limit the number of inequality constraints; and
(4) we can allow *non-linear* cuts.

In our implementation we did not include any of these extensions. Of particular promise is the suggestion to limit the number of inequality constraints as it is suggested that this can lead to improved accuracy iteration-by-iteration [BV11, Section 5]. Therefore, we see that the general cutting-plane algorithm is a highly adaptable algorithm.

Bibliography

[Bis06] C. Bishop. *Pattern Recognition and Machine Learning.* Springer, 2006.

[BV04] S. Boyd and L. Vandenberghe. *Convex Optimization.* Cambridge University Press, 2004.

[BVS08] S. Boyd, L. Vandenberghe and J. Skaf. *Analytic Center Cutting-Plane Method.* 2008. https://see.stanford.edu/materials/lsocoee364b/06-accpm_notes.pdf.

[BV11] S. Boyd and L. Vandenberghe. *Localization and Cutting-Plane Methods.* 2011. http://web.stanford.edu/class/ee364b/lectures/localization_methods_notes.pdf.

[BDV15] S. Boyd, J. Duchi and L. Vandenberghe. *Subgradients.* 2015. http://web.stanford.edu/class/ee364b/lectures/subgradients_notes.pdf.

[Das11] S. Dasgupta. (2011). Two faces of active learning. *Theoretical Computer Science*, Volume 412, Issue 19, pp. 1767-1781.

[LR15] U. Louche and L. Ralaivola. (2015). From Cutting Planes Algorithms to Compression Schemes and Active Learning. *2015 International Joint Conference on Neural Networks*, pp. 1-8.