A. Experimental details

A.1. Data

Across all experiments we use four real datasets:

- **`Spam`**[12]. This dataset includes features of authentic users and spammers from a large social network, and was used in Hardt et al. (2016). The data includes $n = 7,076$ examples and $d = 15$ features. Features include number of words in the post, number of phone numbers in the post and number of followers of the user. The data is balanced, i.e., there is an equal number of positive ($y = 1$) and negative ($y = -1$) examples.

- **`Credit`**[13]. This dataset includes features describing credit card spending patterns, along with labels indicating default on payment. We use the same version used in the recourse paper by Ustun et al. (2019), and adopt their preprocessing procedure (see link). The original dataset includes $n = 30,000$ examples and $d = 11$ Features. Features include age, amount of bill statement and history of past payments. For training time purposes, in our experiments we used a random balanced subset of 3,000 examples, as we did not notice any changes in performance for larger sample set sizes (which is plausible given that f is linear and d here is small).

- **`Fraud`**[14] (Dal Pozzolo et al., 2015). This dataset includes credit card transactions that are either genuine or fraudulent. The original data includes $n = 284k$ examples and $d = 29$ features. Due to confidentiality issues, feature information is not provided. The original dataset is highly imbalanced, and for our experiments we take all available negative examples and uniformly sample a matching number of positive examples, giving a total of $n = 984$ balanced examples.

- **`Finance`**[15]. This dataset includes time-series data describing firms along with an indication of their level of financial distress (denoted `fd`). Each series ends either at the maximal time step of $t = 14$ or earlier if the firm has gone bankrupt. Bankrupcy is declared when $\texttt{fd} < -0.5$, and we use this definition to determine labeles y. The data includes $n = 422$ time-series examples, with each time step within each example described using $d = 83$ anonymized numerical features.[16] The ratio of positive (i.e., non-bankrupt) examples is 67.7%. Time-series lengths are in the range $t \in \{0, \ldots, 14\}$ (inclusive), with mean=8.7 and median=4.9.

Features in all datasets were standardized (i.e., scaled to obtain a mean of zero and standard deviation of one) on the train set. To ensure consistent behavior in term of the effect of the cost function on movement, features in each datasets were further divided by $\sqrt{d}$ where d is the (per-dataset) number of features.

A.2. Training and tuning

For all experiments we use a 60-20-20 split into train, validation, and test sets, respectively, and all results are averaged over multiple random splits. For optimization we use ADAM, where within each epoch use randomize batches of size 24 for `financial distress` and `fraud` 64 for `credit` and 128 for `spam` chosen according to their respective number of examples, and early stop w.r.t. accuracy on the validation set. For all methods, learning rates were set according to the validation set. For σ we set $\tau = 1$ for training and $\tau = 0.2$ for testing, and set the CCP tolerance to 0.001, but note results are robust to variations in these.

A.3. CCP procedure

We use our own implementation of CCP, based on Shen et al. (2016), and using the publicly-available SCS solver[17] for solving the concave sub-problems appearing in each iteration. Algorithm 1 includes pseudocode for our procedure. We use the notation:

$$\sigma_{\cup}(x; f) = \frac{1}{2}\sqrt{\left(\tau^{-1} f(x) + 1\right)^2 + 1}, \qquad \sigma_{\cap}(x; f) = -\frac{1}{2}\sqrt{\left(\tau^{-1} f(x) - 1\right)^2 + 1},$$

[12] Data can be obtained from the authors of Costa et al. (2014).
[13] https://github.com/ustunb/actionable-recourse
[14] https://www.kaggle.com/mlg-ulb/creditcardfraud
[15] https://www.kaggle.com/shebrahimi/financial-distress
[16] The data includes a single categorical feature, which we discard.
[17] https://web.stanford.edu/~boyd/papers/scs.html

for the convex and concave parts of σ, respectively.

Algorithm 1 CCP forward pass

Initialize $t = 0, \delta = \infty$
$x^0 = x$
repeat
  $t = t + 1$
  $g = \nabla_x \sigma_\cup(x^{t-1}; f)$ {linearized convex term at x^{t-1}}
  $x^t = \text{argmax}_{x'}\, g^\top x' + \sigma_\cap(x'; f) - c(x, x')$ {get argmax of current concave proxy}
until convergence
$g^t = \nabla_x \sigma_\cup(x^t; f)$ {final linearized term, to be used for backward pass}
return x^t, g^t

In practice we terminate when $\|x^t - x^{t-1}\|_2 \leq \texttt{tol}$ or after at most $t = 100$ iterations. In our experiments we used $\texttt{tol} = 0.001$, but the vast majority of instances terminated after $t = 5$ iterations. The algorithm returns two objects: the approximate argmax x^t used in the forward pass, and the linearization g^t (of which x^t is an exact argmax) used to parameterize the convex optimization layer to enable a backwards pass.

A.4. Runtime evaluation

In this section we present results on the runtime of our method. Presumably the main overhead in our approach is computing the CCP solution and surrogate in the forward pass. Note however that while each call to solver may be expensive, jointly solving for all examples in a batch (i.e., solving k independent problems in one call, where k is the batch size) can greatly reduce this overhead. All experiments were run on a single laptop (Intel(R) Core(TM) i7-7500U CPU @ 2.70GHz 2.90 GHz, 16.0 GB RAM).

To evaluate this, we run an experiment on synthetic data and for varying batch sizes. We use scikit-learn's `make_classification` function, using 750 train samples, 250 validation samples, $d = 5$ features, balanced classes, and 0.01 label noise. 4 (two left-most plots) show runtime results for increasing batch size k and training for 5 epochs. We consider two alternatives to setting the number of CCP iterations: (i) convergence to tolerance 0.001 on average across examples in the batch (left), and (ii) a slowly increasing, uniform number of iterations, with the number of iterations initialized at one and increased by one after each epoch (right). As can be seen, increasing the batch size greatly improves overall runtime, as well as the the relative runtime of CCP.

We also report runtime on the real datasets used in Sec. 4.1. For equalized comparison we set the number of epochs to 7. Batch sizes were set per dataset as described in Sec. A.2 (as noted, these were not chosen to optimize for speed). Figure 4 (two right-most plots) show total train times and relative CCP times for a all datasets for both stopping criteria.

Code. Our code can be obtained at: `https://github.com/SagiLevanon1/scmp`.

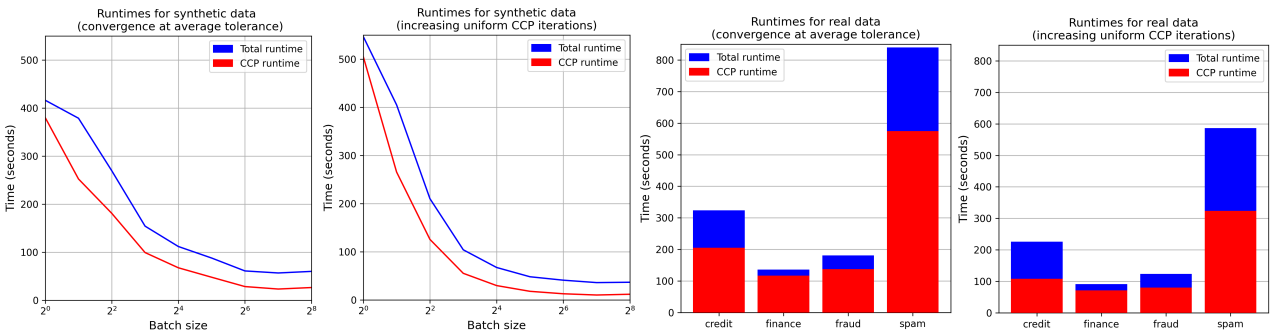

Figure 4. Runtime on synthetic data (varying CCP batch size) and real data (fixed batch size).