The "Smart Predict then Optimise" Framework

Alberto Santini

Fall 2024

Many examples in these lecture notes are adapted from popular books:

- Alexander Schrijver (1998). *Theory of linear and integer programming*. Wiley. ISBN: 0-471-98232-6.
- Vašek Chvátal (1983). *Linear Programming*. W.H. Freeman and Company. ISBN: 0-716-71195-8.
- Laurence Wolsey (2020). *Integer Programming*. 2nd Edition. Wiley. ISBN: 978-1-119-60653-6.
- Silvano Martello and Paolo Toth (1990). *Knapsack Problems: algorithms and computer implementations*. Wiley. ISBN: 978-0-471-92420-3.

1 Introduction

Prediction problems are ubiquitous in statistics and machine learning. An interesting question often neglected is: how will the predictions be used to make real-life decisions? Historically, the two aspects—often called predictive and prescriptive—have been considered separately.

For example, imagine being an operations manager at a large logistics operator serving multiple stores out of a warehouse. You have a machine learning model trained to predict tomorrow's city traffic, i.e., how long it will take to go from one location to the other. Your model takes into account the weather forecast, the time of the day, public transport strikes, etc. If you model the city as a graph, you are assigning to each arc a random variable representing the travel time along the arc. Your machine learning model estimates the travel time values conditional on the features mentioned above. Finally, you use these estimates in your Travelling Salesman Problem (TSP) solver to find the route with the lowest expected travel time or cost.

In the above example, the prediction phase and the optimisation one are completely detached. You train the machine learning model trying to minimise some error (perhaps the squared loss) but completely disregard that the predictions will later be used as input data in an optimisation problem. With this short note, we ask ourselves: is there some way to exploit this knowledge and tune the machine learning loss function to account for the future *use* of the predictions? This is a problem that has been explored many times in the Operational Research literature. Still, it wasn't until recently that a method was developed that combines good theoretical properties and sufficiently good computational performance. The method is the "**Smart «Predict, then Optimise»**" (SPO) of Elmachtoub and Grigas (2021).

Consider an optimisation problem in the following form:

$$\min \quad \vec{c}^{\top} \vec{w} \tag{1a}$$

$$\text{subject to} \quad \vec{w} \in S, \tag{1b}$$

where $(w_1, \ldots, w_d)$ are decision variables, S denotes the feasible region and $\vec{c} \in \mathbb{R}^d$. For example,

in Linear Programming, S is a polytope described by a set of inequalities:

$$S = \left\{\vec{w} \in \mathbb{R}^d \mid A\vec{w} \leq \vec{b} \text{ and } \vec{w} \geq \vec{0}\right\},$$

with $A \in \mathbb{R}^{d \times m}$ and $\vec{b} \in \mathbb{R}^m$. Note that the objective function (1a) is linear, but the feasible region S is an arbitrary set. Although the theory behind the SPO method works for a large variety of problems, we will focus our attention on Linear Programmes (LPs) and Integer Programmes (IPs). These two classes of problems present the best computational characteristics that allow effective real-life use of SPO. We will denote with $w^*(\vec{c}) \in S$ an optimal solution of problem (1a)–(1b), as a function of the objective function coefficients $\vec{c}$.

We will consider the case in which the objective function coefficients $\vec{c}$ are subject to uncertainty. In this case, $\vec{c}$ becomes a vector of random variables and (1a)–(1b) is no longer a *deterministic* optimisation problem, but rather a **stochastic** one.

Consider, then, a set of p input features (also called a *context*) $\vec{x}$ that explain each entry of vector $\vec{c}$. In our previous example, an entry of $\vec{c}$ is the travel time of an arc of the graph, and the set of input features could be tomorrow's rain level forecast, the hour of the day when the vehicle will cross that arc, whether tomorrow is a holiday or a workday, the number of lanes in the road corresponding to the arc, etc. The stochastic version of (1a)–(1b) is:

$$\min \quad \mathbb{E}\left[\vec{c}^\top \vec{w} \mid \vec{x}\right] \tag{2a}$$

$$\text{subject to} \quad \vec{w} \in S. \tag{2b}$$

The above formulation aims to minimise the **expected cost** of a solution to the deterministic optimisation problem (1a)–(1b). By linearity of expectation, and thanks to the fact that we only consider linear objective functions, we can rewrite (2a) as

$$\min \quad \mathbb{E}\left[\vec{c} \mid \vec{x}\right]^\top \vec{w}. \tag{3}$$

Therefore, if we knew $\mathbb{E}\left[\vec{c} \mid \vec{x}\right]$, we could plug it into the deterministic objective function (1a) and solve the deterministic problem. However, in general, we don't know the above statistic, and we try to estimate it from data. Let $(\vec{x}_1, \vec{c}_1), \ldots, (\vec{x}_n, \vec{c}_n)$ be a dataset of size n associating each realised cost vector $\vec{c}_i \in \mathbb{R}^d$ with the corresponding observation $\vec{x}_i \in \mathbb{R}^p$. In a typical machine learning workflow, we define a hypothesis class $\mathcal{H}$ of models of type $\mathbb{R}^p \to \mathbb{R}^d$. For example, the hypothesis class could be: "linear multi-output regression models". Then we look for the member of $\mathcal{H}$ that minimises the empirical loss over our dataset, i.e., for a function f^* defined as:

$$f^* = \operatorname*{argmin}_{f \in \mathcal{H}} \frac{1}{n} \sum_{i=1}^{n} \ell\big(f(\vec{x}_i), \vec{c}_i\big), \tag{4}$$

where $\ell : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}_{\geq 0}$ is the loss function.

In a classical approach, we train model f^* and deploy it in production. At some point, we will obtain a new observation $\vec{x}$. For example, $\vec{x}$ could encode information on tomorrow's weather, etc. We then predict $\hat{\vec{c}} = f^*(\vec{x})$ and use it in the deterministic optimisation problem, solving

$$\min \quad \hat{\vec{c}}^\top \vec{w} \tag{5a}$$

$$\text{subject to} \quad \vec{w} \in S. \tag{5b}$$

2 Predictive vs. prescriptive accuracy and regret

As we already mentioned, there is no direct link between the machine learning training problem (4) and the subsequent deterministic optimisation problem that we solve. During training, we minimise the empirical loss, which we use as a reasonable proxy for model accuracy and, therefore, for **prediction accuracy**. During the combinatorial optimisation phase, however, we are not directly concerned with prediction accuracy. We care more about another concept, which is sometimes called **regret**. When implementing the solution $w^*(\hat{\vec{c}})$ obtained solving (5a)–(5b), we will be able to observe the *real* costs $\vec{c}$. Therefore, implementing our solution will cost us

$$\vec{c}^\top w^*(\hat{\vec{c}}).$$

However, if we had an oracle that told us the real value of $\vec{c}$ beforehand, we could have solved the "real" optimisation problem, and we would have obtained solution $w^*(\vec{c})$. Such a solution has the following cost:

$$\vec{c}^\top w^*(\vec{c}).$$

The difference between these two quantities is the regret:

$$\text{regret}(\hat{\vec{c}}, \vec{c}) = \vec{c}^\top w^*(\hat{\vec{c}}) - \vec{c}^\top w^*(\vec{c}). \tag{6}$$

The regret quantifies how much we lose by solving the optimisation problem with the predicted costs $\hat{\vec{c}}$, compared to solving the same problem with the real (but, unfortunately, unknown) costs $\vec{c}$.

We can now identify the reason for the "uncoupling" between the predictive and the prescriptive analyses. For predictions, we train a machine learning model that maximises the *accuracy* of the predicted costs. For the prescriptive phase, instead, we want predicted costs that minimise *regret*. The solution, in a broad sense, is clear: we should train our machine learning model optimising for low regret in the subsequent prescriptive phase rather than for high accuracy. This is what the SPO framework will help us accomplish.

To give an extreme example of why prediction accuracy and regret are not aligned, imagine having a machine learning model f that, given an input vector $\vec{x}$ and a real (but unknown) associated cost vector $\vec{c}$, always outputs prediction $f(\vec{x}) = \hat{\vec{c}} = 10^9\vec{c}$. In other words, the prediction is simply one billion times the real cost. Such a model will be terrible from the point of view of prediction accuracy, and it will have a huge Mean Squared Error (MSE). When plugging $\hat{\vec{c}}$ instead of $\vec{c}$ into the optimisation problem, however, we are just performing a rescaling of the objective function. Therefore, problems (1a)–(1b) and (5a)–(5b) will have the same optimal solution ($w^*(\vec{c}) = w^*(\hat{\vec{c}})$) and the regret will be zero:

$$\text{regret}(\hat{\vec{c}}, \vec{c}) = \vec{c}^\top w^*(\hat{\vec{c}}) - \vec{c}^\top w^*(\vec{c}) = 0.$$

Therefore, a model with bad predictive properties can actually be great when using the predicted costs in an optimisation problem. On the other hand, a model with good (but not perfect) predictive properties is not necessarily the best one when it comes to minimising regret.

3 Examples

To understand how the regret differs, e.g., from the MSE, consider the following basic example. You are given the network in Figure 2 in which two parallel arcs link two vertices, and you must choose the shortest path from the left to the right node. In other words, the optimisation

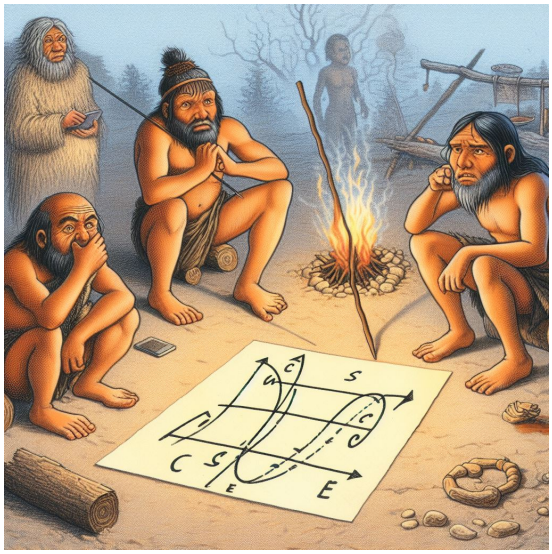

Figure 1: Left: statisticians try to figure out if they can minimise something other than the MSE. Right: operational researchers using the "Smart «Predict, then Optimise»" framework. *Credits:* images by DALL·E 3.

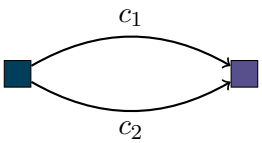

Figure 2: Simple multi-graph with two nodes and two arcs with stochastic cost.

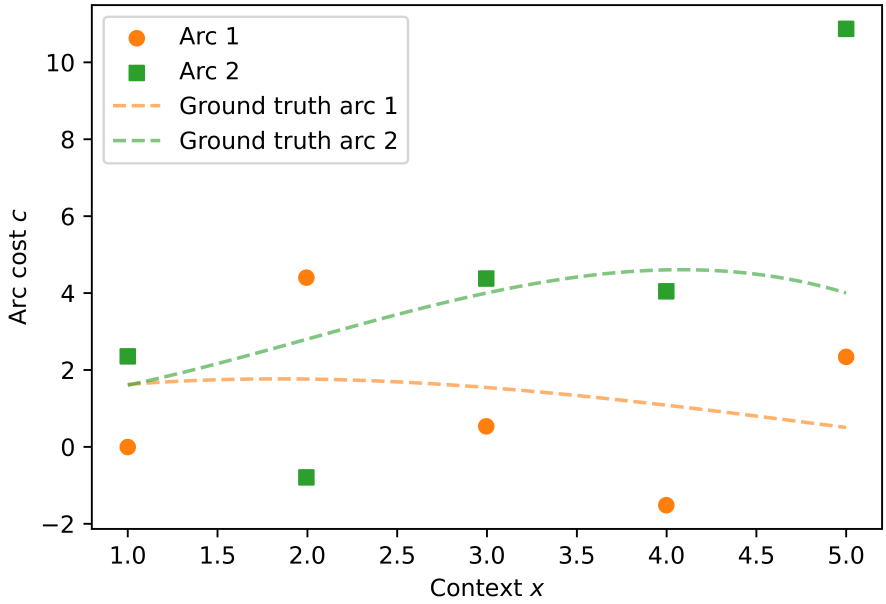

Figure 3: Dataset for the two-arcs example.

problem must select the arc with the lowest cost. The two arcs have stochastic costs c_1 and c_2 that depend on a univariate independent variable x.

Figure 3 shows a dataset with five (x, c) points for each arc, together with the ground truth function $f : \mathbb{R} \to \mathbb{R}$ used to produce them. The points are obtained by adding a Gaussian noise to the ground truth function. Figure 4 shows two linear models trained on the dataset using Ordinary Least Squares, i.e., minimising the MSE. Next, Figure 5 depicts the training prediction error made by the two models. In particular, the MSE is the sum of the squares of the lengths of the dashed vertical segments.

What would be, instead, the regret when using these models? In other words, we keep the models as they are, the result of an OLS estimation obtained minimising the MSE and not the regret, but we now compute the regret a posteriori. Our optimisation problem is trivial enough that we can immediately see its output: given the two cost estimates, $\hat{c}_1$ for arc 1 and $\hat{c}_2$ for arc 2, the model chooses to use arc 1 if $\hat{c}_1 \leq \hat{c}_2$ and arc 2 otherwise. In the first case, the real incurred cost is c_1 while it is c_2 in the second case. Therefore, if $\hat{c}_1 \leq \hat{c}_2$ when $c_1 \leq c_2$ and $\hat{c}_1 \geq \hat{c}_2$ when $c_1 \geq c_2$, the optimisation model will always select the best arc and the regret will be zero. On the other hand, if $\hat{c}_1 \leq \hat{c}_2$ for some data point with $c_1 > c_2$ or $\hat{c}_1 \geq \hat{c}_2$ for some data point with $c_1 < c_2$, then the optimisation model will choose the wrong arc and the regret will be $|c_1 - c_2|$. As Figure 6 shows, we are in the first case (no regret) for three out of five points and in the second case (positive regret) for the other two points.

Let's see, then, what happens when we use the regret as the loss function during training. To this end, we will consider a slightly larger training set with twenty observations, and we will remove the noise to make visualisations clearer. Figure 7 depicts this dataset. While both functions to generate arc costs c_1 and c_2 from x are deterministic in this example (indeed, $c_1 = 0.015x^3 + 0.1x^2 - 0.2x + 1$ and $c_2 = 0.02x^2 + 0.1x + 3.5$), they are not linear. Therefore, the linear models we use are misspecified and must show some bias when predicting $\mathbb{E}[c\,|x]$. The figure also marks with a dashed red line the threshold for the independent variable x when arc 1 becomes more expensive than arc 2.

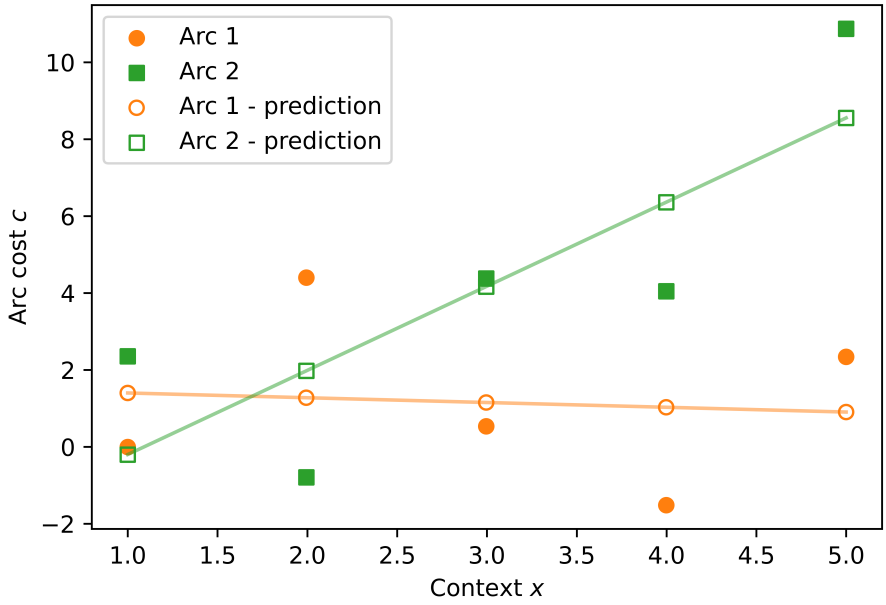

Figure 4: OLS models obtained minimising the MSE for the two-arcs example.

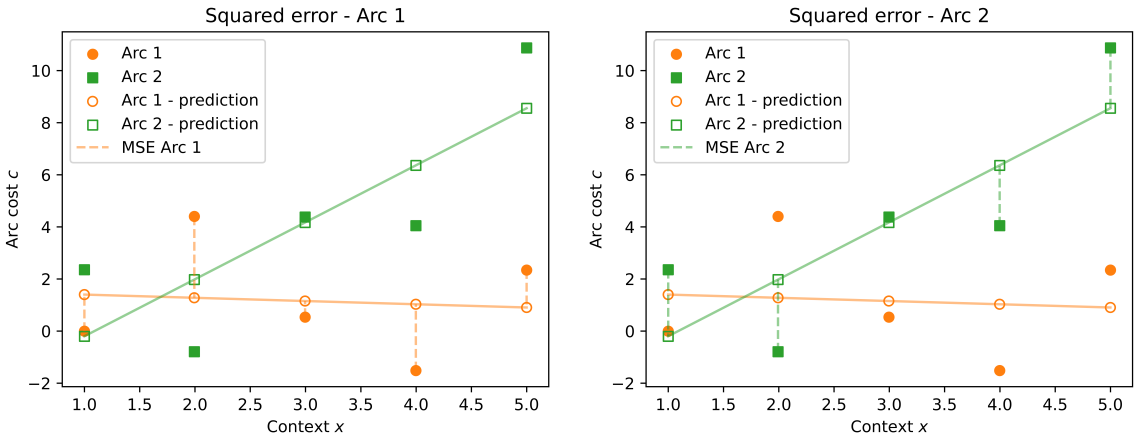

Figure 5: Prediction errors made by the linear models when predicting the costs of the two arcs.

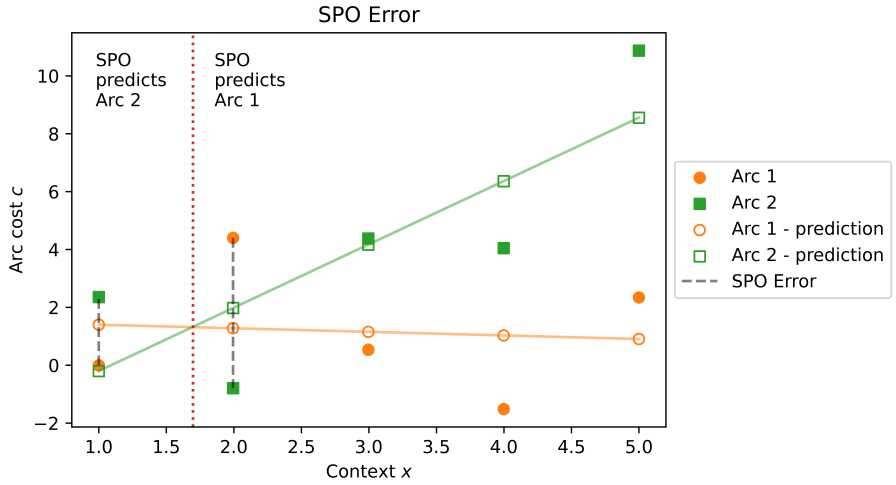

Figure 6: SPO errors made by the linear models trained using the MSE when used in the subsequent optimisation phase.

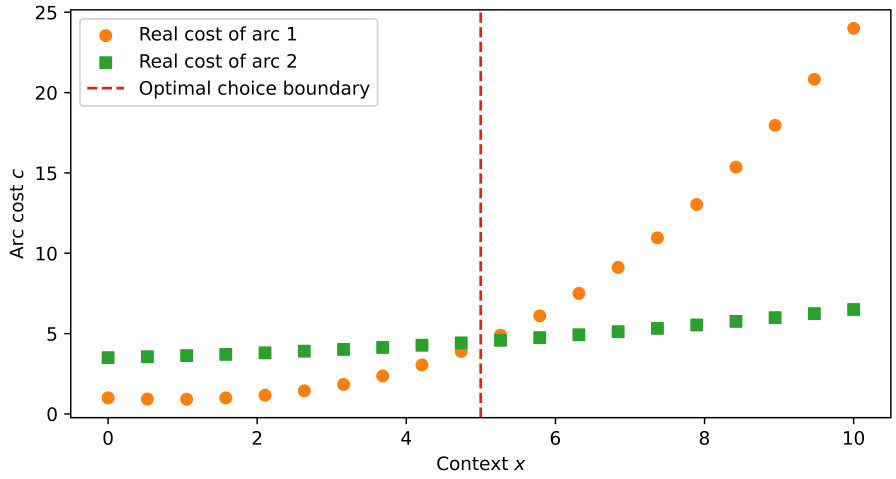

Figure 7: Dataset for the expanded two-arcs example.

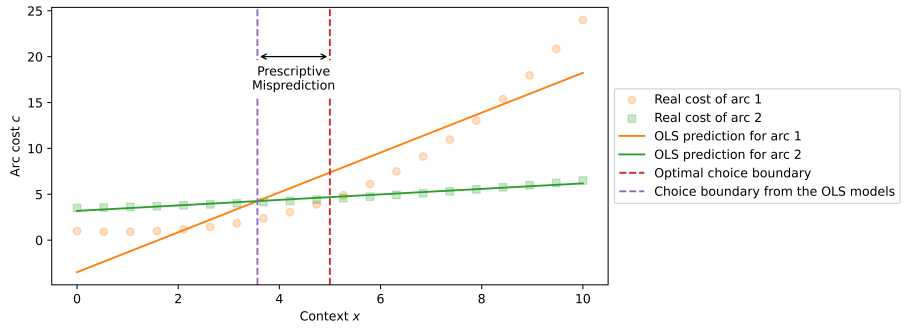

Figure 8: OLS models for the expanded two-arcs example.

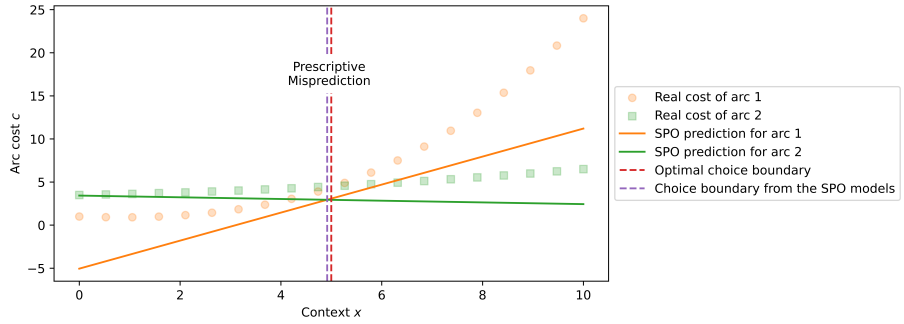

Figure 9: Linear models trained using the regret loss function for the expanded two-arcs example.

If we train two linear models using OLS, we end up with the two regression lines depicted in Figure 8. Note that we should change our prediction of which arc is the cheapest starting where the two regression lines intersect. Therefore, the decision threshold based on the OLS predictions and the real decision threshold based on knowing the true distribution of $c \mid x$ delimit an area where the prescriptive optimisation model will choose the wrong arc. Left of the purple dashed line, $\hat{c}_1 \leq \hat{c}_2$ and $c_1 \leq c_2$; therefore, the predicted costs will lead to the correct prescriptive decision. Analogously, right of the red dashed line, $\hat{c}_1 \geq \hat{c}_2$ and $c_1 \geq c_2$. However, between the two dashed lines, $\hat{c}_1 \leq \hat{c}_2$, but $c_1 > c_2$ and the optimisation model will choose the wrong arc.

We then decide to apply the SPO framework and train the two linear models using the SPO loss function. Figure 9 depicts the corresponding regression lines. The lines do not fit the training data as well as the previous two if we look at them through the lens of the MSE. However, the area where the downstream optimisation model would choose the wrong arc (and, therefore, have positive regret) is now basically empty. The reason is that the relative order of $\hat{c}_1$ and $\hat{c}_2$ now coincides everywhere with the order of c_1 and c_2. We have succeeded at training a prediction model that is useful for the subsequent optimisation task rather than just training a model with high predictive accuracy.

4 Regret and its surrogate: the SPO+ loss

Given the previous considerations, we have a general plan: using the loss function

$$\ell\left(\hat{\vec{c}}, \vec{c}\right) = \text{regret}(\hat{\vec{c}}, \vec{c})$$

during model training. In a practical implementation, however, we must take care of some important numerical details. First, we must be careful with ties and multiple optima of the

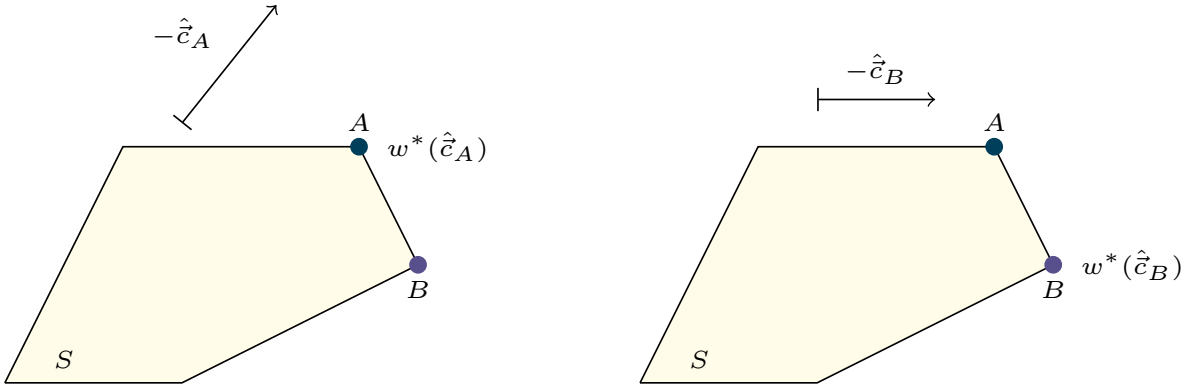

Figure 10: Two estimates $\hat{\vec{c}}_A$ and $\hat{\vec{c}}_B$ of the same real cost vector $\vec{c}$ (not depicted) lead to different optimal solutions of the LP over polytope S.

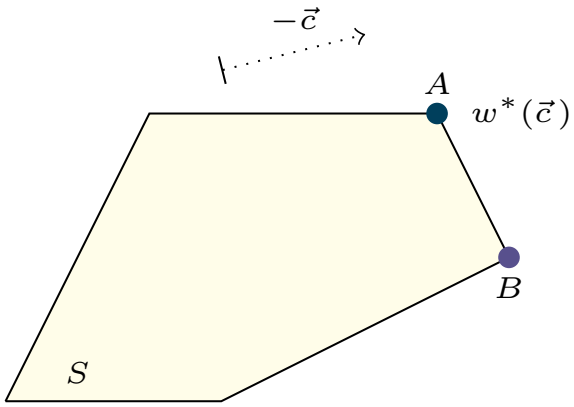

Figure 11: Optimal solution of the LP when using the real costs $\vec{c}$.

optimisation problem. To see why this is important, consider a machine learning model f that always predicts $\hat{\vec{c}} = \vec{0}$. Such a model would "remove" the objective function from the optimisation problem and would make any feasible solution optimal. In this case, the value of $w^*(\hat{\vec{c}})$ would be pretty meaningless. Second, we must be wary of the numerical properties of our regret loss function. In general, this function is non-convex and, depending on the nature of the feasible region S, even discontinuous.

To see how "bad" the regret function can behave, let's consider a familiar case: when S is a polytope and (1a)–(1b) is an LP. Figure 10 shows the same polytope S and two estimates $\hat{\vec{c}}_A$ and $\hat{\vec{c}}_B$ of an unknown cost vector $\vec{c}$. When solving the deterministic LP using $\hat{\vec{c}}_A$, the optimum is vertex A; Solving the LP using $\hat{\vec{c}}_B$, the optimum is vertex B. Figure 11 reveals the real cost vector $\vec{c}$. If we could have solved the LP using $\vec{c}$, the optimum would have been A. Therefore, the regret associated with estimate $\hat{\vec{c}}_A$ is zero, while the regret associated with estimate $\hat{\vec{c}}_B$ is strictly positive.

Note that small perturbations of $\hat{\vec{c}}_B$ will still produce B as the optimal solution of the LP. Therefore, the gradient of the regret function at $\hat{\vec{c}}_B$ is zero. Indeed, the regret function for an LP has zero gradients almost everywhere because small changes in the cost vector do not immediately change the optimal vertex. Moreover, the regret function itself is discontinuous: regret stays the same while $w^*(\hat{\vec{c}})$ does not change and then suddenly jumps as soon as $w^*(\hat{\vec{c}})$ moves to another vertex of the polytope.

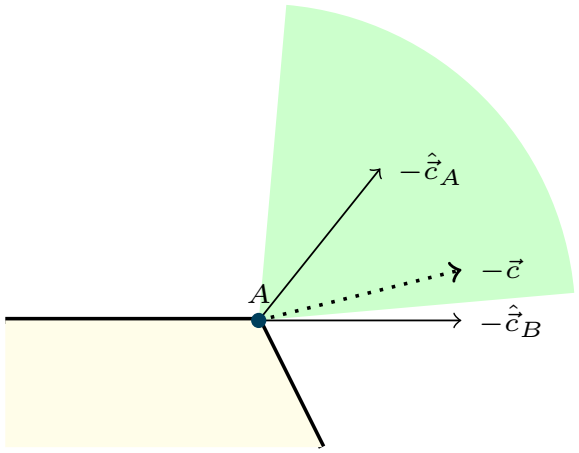

Figure 12: Relationship between $\vec{c}$, $\hat{\vec{c}}_A$ and $\hat{\vec{c}}_B$.

In Figure 12, $\vec{c}$, $\hat{\vec{c}}_A$ and $\hat{\vec{c}}_B$ are drawn with a common origin at vertex A. The figure shows that $\hat{\vec{c}}_B$ is "closer" to $\vec{c}$ than $\hat{\vec{c}}_A$ is. Still, $\text{regret}\big(\hat{\vec{c}}_B, \vec{c}\big) > 0$ while $\text{regret}\big(\hat{\vec{c}}_A, \vec{c}\big) = 0$, i.e., $\hat{\vec{c}}_B$ is a more accurate prediction of $\hat{c}$ (from a "myopic" machine learning point of view), but a worse prediction to use in practice during the prescriptive phase. In the figure, all vectors $\hat{\vec{c}}$ falling within the green area would lead to optimal solution A when solving the prescriptive LP.

To overcome the numerical difficulties involved with the use of regret (which, Elmachtoub and Grigas (2021) call the "SPO Loss"), the authors devise a surrogate "SPO+" loss function:

$$\ell_{\text{SPO+}}\big(\hat{\vec{c}}, \vec{c}\big) = \max_{\vec{w} \in S} \left\{ \big(\vec{c} - 2\hat{\vec{c}}\big)^\top \vec{w} \right\} + 2\hat{\vec{c}}^\top w^*(\hat{\vec{c}}) - \vec{c}^\top w^*(\vec{c}). \tag{7}$$

The derivation is developed and justified in the paper. Here, we focus on the interesting properties enjoyed by the SPO+ loss. First, we should verify that it takes value zero for a correct prediction:

$$\begin{aligned} \ell_{\text{SPO+}}\big(\vec{c}, \vec{c}\big) &= \max_{\vec{w} \in S} \big\{ -\vec{c}^\top \vec{w} \big\} + 2\vec{c}^\top w^*(\vec{c}) - \vec{c}^\top w^*(\vec{c}) = \\ &= -\vec{c}^\top w^*(\vec{c}) + 2\vec{c}^\top w^*(\vec{c}) - \vec{c}^\top w^*(\vec{c}) = 0. \end{aligned}$$

Next, the authors prove that:

- **Upper bounding.** $\ell_{\text{SPO+}}$ always bounds the regret from above.
- **Convexity.** $\ell_{\text{SPO+}}$ is convex in $\hat{\vec{c}}$.
- **Existence of a subgradient.** $\ell_{\text{SPO+}}$ admits a subgradient, $2\big(w^*(\vec{c}) - w^*(2\hat{\vec{c}} - \vec{c})\big)$, everywhere. The last two properties are crucial for the computational efficiency of any training algorithm using the SPO+ loss.
- **Special case of linear models.** The training problem of a linear regression model with the SPO+ loss is a Linear Programme when S is a polytope.
- **Relation with classification.** When $S = [-1/2, 1/2]$ and $c \in \{-1, 1\}$, the SPO loss becomes the 0–1 classification loss and the SPO+ loss becomes the hinge loss.
- **Consistency.** Under some hypotheses, the SPO+ loss is consistent with the SPO loss, i.e., if we could have access to the entire population instead of our sample $(\vec{x}_1, \vec{c}_1), \ldots, (\vec{x}_n, \vec{c}_n)$, both training problem would have the same minimiser. The most constraining hypothesis is that the distribution of $\vec{c}|\vec{x}$ is continuous and symmetric around its mean.

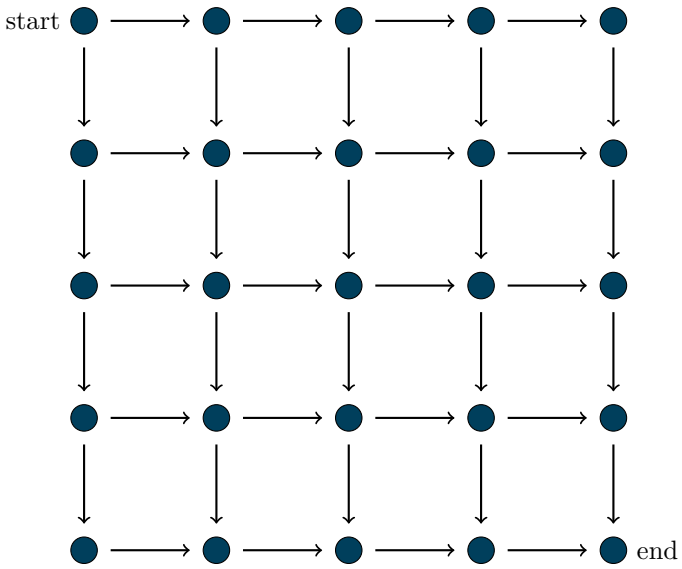

Figure 13: Grid-like shortest path problem evaluated by Elmachtoub and Grigas (2021).

- **Consistent estimator.** Furthermore, the SPO (and, therefore, the SPO+) is a consistent estimator of $\mathbb{E}\left[\vec{c}|\vec{x}\right]$.
- **Calibration.** Later, Ho-Nguyen and Kılınç-Karzan (2022) have shown that SPO+ is calibrated to SPO, i.e., it enjoys non-asymptotic consistency properties.

Finally, the authors show through extensive computational experiments that the SPO+ loss leads to lower prescriptive error (i.e., regret) compared with other consistent estimators of $\mathbb{E}[\vec{c}|\vec{x}]$ such as linear OLS models, linear models trained minimising the Mean Absolute Error, and random forests. For example, they observed the results in Figure 14 on instances of the shortest path problem. This problem is defined over a directed graph with 25 vertices arranged in a 5×5 grid, with arcs going rightwards and downwards, and the goal is to find the shortest path from the top-left to the bottom-right corner. The ground truth function linking $\vec{x} \in \mathbb{R}^5$ with $\vec{c} \in \mathbb{R}^{40}$ is a polynomial whose degree appears on the x-axis of the plot. The y-axis represents the normalised regret, i.e., the ratio between regret and the optimal objective function value of the optimisation problem. The authors test four models using training sets of different sizes (100, 1000 and 5000), different variances of the noise added to the ground truth function (0 and 0.5), and performing 50 experiments for each parameter combination. The results in Figure 14 show that the SPO+ loss outperforms the other models, especially for small training sets and for higher model misspecification (i.e., higher degree of the polynomial). Random forests are universal approximators, and therefore, their relative performance improves when the training set size is large. Still, the SPO+ loss is competitive and relies on a much more interpretable linear model.

References

Chvátal, Vašek (1983). *Linear Programming*. W.H. Freeman and Company. ISBN: 0-716-71195-8.

Elmachtoub, Adam and Paul Grigas (2021). "Smart «Predict, then Optimize»". In: *Management Science* 68.1, pp. 9–26. DOI: 10.1287/mnsc.2020.3922.

Martello, Silvano and Paolo Toth (1990). *Knapsack Problems: algorithms and computer implementations*. Wiley. ISBN: 978-0-471-92420-3.

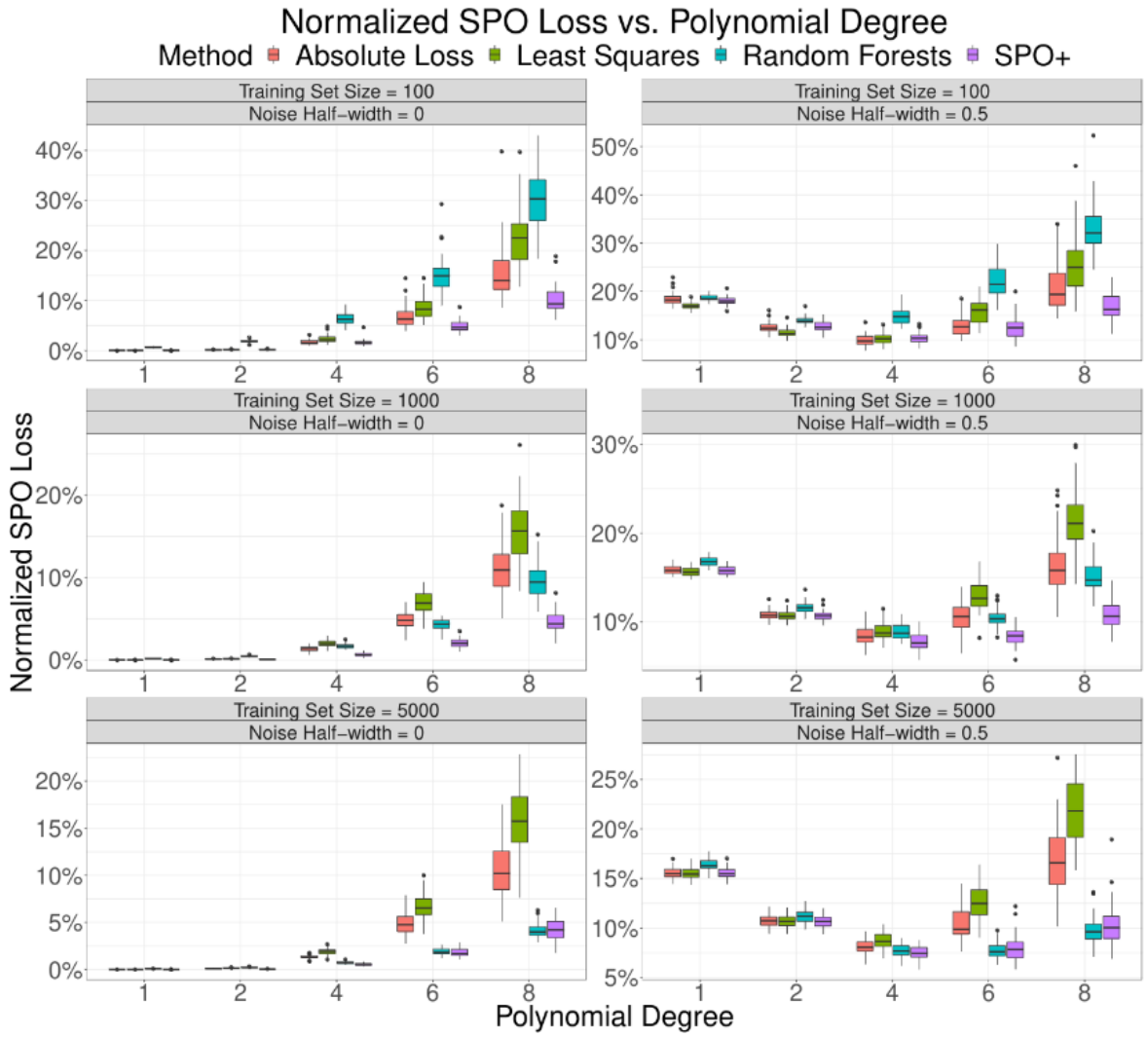

Figure 14: Results on a shortest-path problem from Elmachtoub and Grigas (2021).

Ho-Nguyen, Nam and Fatma Kılınç-Karzan (2022). "Risk Guarantees for End-to-End Prediction and Optimization Processes". In: *Management Science* 68 (12), pp. 8680–8698. DOI: 10.1287/mnsc.2022.4321.

Schrijver, Alexander (1998). *Theory of linear and integer programming*. Wiley. ISBN: 0-471-98232-6.

Wolsey, Laurence (2020). *Integer Programming*. 2nd Edition. Wiley. ISBN: 978-1-119-60653-6.