Optimization Under Uncertainty:
The Scenario Approach and Guaranteed Error Machines

Elliot Pickens
pickense@carleton.edu

June 27, 2020

Abstract

Navigating uncertainty is never easy, and when it rears its head optimization becomes a particularly difficult task. Under such circumstances we cannot simply carry out an optimization regime directly. Instead we must find a way to incorporate the uncertainty into the optimization process. One approach to this problem is the "scenario approach." In this paper I will outline the basics of the "scenario approach" and dive into a few basic applications. In particular, I will show the link between statistical learning and the "scenario approach" through a treatment of guaranteed error machines. Following the introduction to GEMs I present a few potential expansions of GEMs and investigate their efficacy on both real and synthetic data.

1 Introduction

For centuries mathematicians, statisticians, and all persons interested in quantitative problem solving have incorporated some form of optimization into their process. Since the 1900s significant work has been put into formalizing such processes mathematically and searching for methods that apply modern developments in mathematics and computing to optimization.[9] One methodology to arise from the past decade is the "scenario approach." The scenario approach attempts to answer the question of how can we find an optimal value or set of values when outcomes are not guaranteed to stay the same.

Settings where the outcome is not known to be consistent may seem rare, but they pop up in a number of areas. Robotics and control systems are one area where consistent outcomes cannot be depended upon given the potential for disturbances and incomplete knowledge of system dynamics. Finance is another area that is likewise plagued by uncertainty due to rapidly evolving and reactionary market dynamics.[3] The list of examples is boundless and as a result a great deal of effort has been put into figuring out how we can effectively apply an optimization program when uncertainty emerges. This paper centered on one solution to the problem: the scenario approach.

In the following sections I first introduce a several approaches to optimization under uncertainty, before moving onto the specifics of the scenario approach and outlining the theory behind it. In subsequent sections I investigate how the scenario approach can be used in data centric environments and its possible applications within the space. In particular, I hope to show how the lessons learned from the inherently uncertain situations considered by the scenario approach transfer to the world of big data where uncertainty is frequently overlooked.

2 Optimization Under Certainty

The goal of optimization is to the find the best possible solution in a broad solution space. In traditional optimization problems this "best" solution is generally assumed to be unique and consistent. That is to say that if an optimization program returns a set of parameters we assume that those parameters are and will continue to be optimal.

3 Optimization Under Uncertainty

Under uncertainty, a set of uncertain elements denoted Δ is introduced. In this context, each $\delta \in \Delta$ can be interpreted as an uncertain outcome. Naturally, each of these elements will interact slightly differently with each possible ν, where $\nu \in \mathbb{R}^{d-1}$ is a vector of parameters.[3]

These uncertain elements disallow us from simply solving for a true "best" set of parameters, because each δ has a potentially unique "best" ν. In lieu of a true "best" set of parameters, under uncertainty we introduce a loss function $\ell(\nu, \delta)$ that we will optimize by finding the ν that best minimizes (or inversely maximizes) its output.[3]

3.1 The Worst-Case Approach

When handling uncertainty the worst-case approach is possibly the most intuitive. Given that we have to handle some degree of uncertainty, it is natural to consider the effect such unpredictability can have on the set of possible outcomes.

In the worst-case approach we take advantage of our knowledge of Δ in order to mitigate the result of the most destructive uncertain element δ. This approach is naturally fairly conservative in nature, but when stability is required it is not uncommon to adopt such a method.[3]

$$min_{\nu \in \mathbb{R}^{d-1}} \left[max_{\delta \in \Delta} \ell(\nu, \delta) \right] \tag{1}$$

The general form of the worst-case approach is presented above in equation 1. It is important to note going forward that while we will seek a number of different approaches to optimization under the shadow of Δ, the standard form of the combining minimization and maximization functions will continue to show up as the key component of many approaches.

3.2 The Average Approach

The average approach brings a much more probabilistic perspective to the problem. Within the framework of this view, we take Δ to be a random variable supplied with a probability distribution $\mathbb{P}$. Depending on the setup of the optimization being carried out, $\mathbb{P}$ can be setup in a number of different ways. $\mathbb{P}$ can, for example, be used to weight the importance of each uncertain element δ or represent the rate at which they occur. Regardless of setup, the importance of $\mathbb{P}$ for the purposes of the average approach is that it can be used to assign a weight value to each δ and carry out an expectation operation over Δ.[3]

$$min_{\nu \in \mathbb{R}^{d-1}} \mathbb{E}_{\Delta}[\ell(\nu, \delta)] = min_{\nu \in \mathbb{R}^{d-1}} \int_{\Delta} \ell(\nu, \delta) d\mathbb{P} \tag{2}$$

This approach is frequently applied to noisy systems where each burst of noise (δ) can be understood to have been a draw from some sort of distribution (commonly a Gaussian). It should also be noted that while I will not investigate any other optimization methods that make explicit assumptions about the nature of Δ, the scenario approach itself does hold a philosophical connection to the average approach via its probabilistic formulation of the problem.

3.3 The Chance-Constrained Approach

The chance constrained approach is yet another probabilistic path towards optimization. Rather than simply taking an average view of Δ, the chance constrained approach looks to achieve a given level of success with a preset probability.

Like the worst case approach, the chance constrained approach also seeks to minimize a worst case scenario. In fact, the chance constrained approach is effectively a special case of the worst case approach that operates over only a specific portion of Δ.[3]

$$min_{\nu \in \mathbb{R}^{d-1}, \Delta_{\epsilon}} \left[max_{\delta \in \Delta_{\epsilon}} \ell(\nu, \delta) \right] \tag{3}$$

This approach allows us to find the minimum loss value that is directly associated with the risk level we are willing to accept. Tuning ϵ will therefore adjust both the risk level and the parameters of our function.

Compared to the previous two methods mentioned, the chance constrained approach is unique in that it allows the user to input their personal risk tolerance into the optimization program. Although allowing the user to make decisions about the uncertainty of an environment is not always beneficial, it can be incredibly helpful when expert knowledge is available. In finance, for example, each investor has a set of requirements they must meet in order to stay solvent and turn a profit. The chance-constrained approach allows for such requirements to be take into account in a way neither the worst-case or average approach do.

Considering the usefulness of a tune-able risk parameter it is natural to wonder whether this approach has seen widespread adoption. At the moment it has not seen much use (in spite of significant theoretical work) due in large part to the complexity of actually solving chance constrained optimization. This difficulty (and the general question of how we can tune our risk level) is something to keep in mind as we move into the discussion of the scenario approach.

4 The Scenario Approach

Similar the other methods previously mentioned, the scenario approach uses a probabilistic angle of attack. Unlike the average and chance constrained approaches, however, the scenario approach does not require complete knowledge of the distribution $\mathbb{P}$ associated with Δ. Instead, the scenario approach seeks to approximate Δ by independently sampling N *scenarios* $\delta_1, \delta_2, ..., \delta_N$ from $\mathbb{P}$. Approximating $\mathbb{P}$ with N scenarios may seem

somewhat difficult, but we will see that it is possible to develop a very precise definition of our approximation. The use of sampling also has the added benefit of increasing the computational tractability of algorithms built onto of the set used to approximate Δ.[3]

Once a sample set of scenarios $\delta_1, \delta_2, ..., \delta_N$ has been collected, we can consider the simplest possible scenario program.[3]

$$min_{\nu \in \mathbb{R}^{d-1}} [max_{i=1,...,N} \ell(\nu, \delta_i)] \tag{4}$$

This program, which is referred to as SP_N (a scenario program with N scenarios) is at its core an application of the worst-case approach to our sample of Δ. Solving this program will return an optimal set of parameters ν^* with an associated loss value ℓ^*.[3]

Key Assumption: Convexity

The scenario approach works under the assumption that the loss function $\ell(\nu, \delta)$ is convex in ν. Convexity in ν inherently sets a limit on the application of the scenario approach, but given that many problems are convex (or can be restructured to be convex) this limitation is by no means a heavy cap on the scenario approach.[a]

Convexity of a function can be formalized to state that a function $f(\nu)$ is convex if $\forall\, v', v''$ and $\alpha \in [0, 1]$:[3]

$$f(\alpha\nu' + (1-\alpha)\nu'') \leq f(\nu') + (1-\alpha)f(\nu'') \tag{5}$$

This definition of convexity plainly states that if a line is drawn between any two points ν_1, ν_2 on the graph of $f(\nu)$ it will sit above $f(\nu_3)\ \forall\, \nu_1 < \nu_3 < \nu_2$.[3]

[a] Some work has been done to push the scenario approach past the limitation of convexity, but it is still an open area of research.[3]

4.1 Example: Polynomial Fitting

Imagine we are in possession of some data set that contains a series of points $(x_i, y_i) \in \mathbb{R}^2$. How could we use the scenario approach along with such a data set to fit a polynomial to our data?

In this example we have N independently sampled data points that we can treat as scenarios. Thus, assuming we believe the polynomial that best suits our data is quadratic we can setup a simple implementation of the scenario approach.

$$min_{\nu_1,\nu_2,\nu_3} = \left[max_{i=1,...,N} |y_i - [\nu_1 + \nu_2 x_i + \nu_3 x_i^2]|\right] \tag{6}$$

Equation (6) follows the standard setup of the scenario approach shown in equation (4) to select the set of parameters $\nu = \{\nu_1, \nu_2, \nu_3\}$ that best fit the data. In this particular example, this process is akin to finding

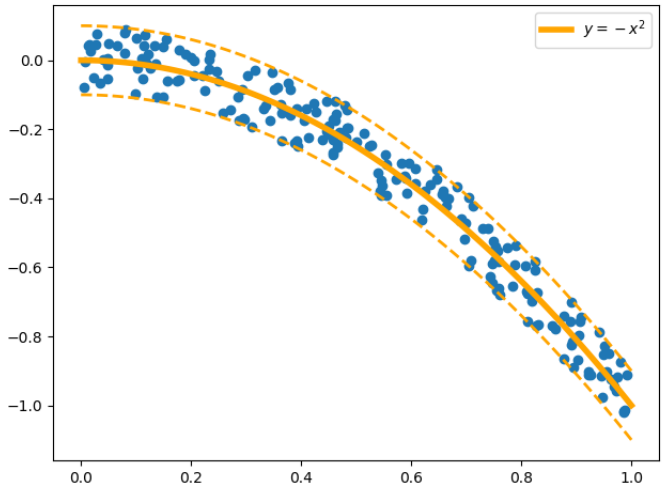


Figure 1: Example data against the line we are trying to fit[1]
[1] Graphic based on an example presented in [3]

the center of the band, which best encapsulates all of our data.[1][3]

4.2 How Good is the Scenario Approach?

As previously mentioned, the scenario approach uses an approximation of Δ to solve for our set of solution parameters ν^*. Computationally a sample based approximation of Δ is incredibly useful, but if we cannot determine the accuracy of our approximation its unclear whether or not it is possible for a scenario program to return a valid result. To do this we need to consider the worst case loss ℓ^* that can result from a given scenario program. ℓ^* is the upper bound given by applying a worst-case optimization to a set of scenarios $\delta_1, \delta_2, ..., \delta_N$ via SP_N, but it is by no means an upper bound for all of Δ. This discrepancy brings about the question of just how large the difference between ℓ^* and $\ell_{worst-case}$ is.[3]

Theorem 4.1. *For any $\epsilon \in (0,1)$ (violation parameter) and $\beta \in (0,1)$ (confidence parameter), if the number of scenarios N satisfies $N \geq \frac{2}{\epsilon}(ln\frac{1}{\beta}+d-1)$, then with probability $\geq 1-\beta$, it holds that $\mathbb{P}\{\delta \in \Delta : \ell(\nu^*, \delta) > \ell^*\} \leq \epsilon$*

The theorem above points to a couple of key results. Theorem 4.1 states that the probability that some new (and unseen δ) results in a loss value $\ell(\nu^*, \delta)$ that is greater than ℓ^* can be held to a given value of ϵ. Conversely, a given ϵ value may or may not be satisfied depending on the set of scenarios at hand. Each sample of Δ is different and each resulting ν^* and ℓ^* are unique. Since ν^* and ℓ^* inform $\mathbb{P}\{\delta \in \Delta : \ell(\nu^*, \delta) > \ell^*\} \leq \epsilon$ depends on the sample. The theorem accounts for this by including a second level of probability β as a way of reflecting the rate at which different samples occur. When put together, the ϵ bound is said of hold true with probability $1 - \beta$.[3]

The pairing of ϵ and β does not benefit the readability of theorem 4.1, but it does allow for a number of straight forward computations. With this theorem we can quickly solve for N, ϵ, and β assuming we either know or can assume something about the other two values.[2] Such calculations are particularly useful during

[1] We can formalize the process of fitting a "band" around our data through the use of a Chebyshev layer[6]
[2] I do not mention d alongside the other parameters in this case because I assume the number of optimization variables can easily be calculated

the early stages of solving a scenario program. In particular, we can select a pair of ϵ and β values that we are willing to accept and use them to inform our sampling process.[3]

It is also interesting to note that knowledge of $\mathbb{P}$ is not required when using theorem 4.1. We are solely dependent on our sample and the scenarios within it. This is a slightly odd, but powerful result given that it implies a forceful level of universality to the scenario approach. That being said, knowledge of $\mathbb{P}$ is not worthless. When setting up a scenario program we can find a better version of ν by using our prior knowledge to restrict its domain. This could take the form of zeroing out a number of parameters, bounding their range, or otherwise altering the space they take on.[3]

4.3 The Risk Return Trade-off

In the previous sections we have seen how the scenario approach can identify a set of parameters that limit the chance of throwing back a loss value worse than the worst case seen in the scenario sample. It is, however, possible that the solution to the scenario program does not provide an acceptable return (success rate/performance level). In such cases its reasonable to consider the how we might be able to tune our risk level in order to settle on a new ν^*, which meets out performance requirements.

Conveniently, the scenario approach is perfectly suited for such *a posteriori* tweaks. Considering the set of scenarios $\delta_1, \delta_2, ..., \delta_N$ are already in hand there are a few simple operations we can carryout during the improvement process. The foremost being a straightforward search. Using our ν^* we can find all scenarios that result in the worst case loss ℓ^* by plainly plugging them into ℓ. A plug in and test everything method is easily accomplished via $\ell(\nu^*, \delta_i) = \ell^*$ for all $i \leq N, i \in \mathbb{Z}$.[3] Once all such scenarios have been found we can discard them, leaving us with a reduced sample. With the remaining scenarios in the sample we can then rerun the scenario program to get a new ν^* and ℓ^*. If the new values still fail to meet the user requirements the process outlined above can be repeated until such a level is achieved.[3]

At first glance such a tuning algorithm may seem like a violation of previously stated theorems, and perhaps an infringement on the heavily sample dependent scenario approach as a whole. Fortunately, the worries mentioned above are only partly true, and with a few alterations to the previous theorems we can snugly fit our tuning algorithm within the theoretical underpinning of the scenario approach.

The following two theorems are versions of theorem 4.1 that have been adapted to account for k discarded scenarios. These theorems both use with ν_k^* and ℓ_k^* to represent the set of parameters and performance level that arise from the scenario program after k scenarios have been discarded.[3]

Theorem 4.2. *ℓ_k^* is ϵ_k-risk guaranteed with probability $\geq 1 - \beta$, where*

$$\epsilon_k = \frac{k}{N} + \left[\frac{\sqrt{k}}{N} + \frac{\sqrt{k}+1}{N} \left((d-1)ln(K+d-1) + \frac{d-1}{\sqrt{k}} + ln\frac{1}{\beta} \right) \right] \tag{7}$$

Theorem 4.3. *ℓ_k^* is **approximately** ϵ_k-risk guaranteed with probability $\geq 1 - \beta$, where*

$$\epsilon_k = \frac{k}{N} + O(\frac{N}{\sqrt{N}}) \tag{8}$$

The two theorems above can each be divided into two terms: a $\frac{k}{N}$ term and as theorem 4.2 makes clear a slightly more complicated one. In theorem 4.2 that second term looks quite daunting, but with a little bit of simplification we can arrive at the much more readable theorem 4.3. We reach theorem 4.3 by recognizing that the bracketed term in theorem 4.2 approaches zero as $O(lnN/\sqrt{N})$, which is only slightly slower than $O(1/\sqrt{N})$.

The beauty of theorem 4.3 and $O(1/\sqrt{N})$ convergence is that if we hold the ratio $\frac{k}{N}$ constant then the second term in theorem 4.3 will go to zero as $O(1/\sqrt{N})$ meaning that will a large number of scenarios tuning risk by discarding scenarios will only slightly alter our overall confidence.[3]

It also happens to be true that theorem 4.2 holds true regardless of the way in which the scenarios are discarded. Whether we discard scenarios based on the degree to which they improve performance or at random the result is the same. The lack of direct connection to the specific scenarios being discarded allows for a significant degree of flexibility in applying theorem 4.2. We can even use it to create a scree like risk-performance plot to inform us roughly how many scenarios we can throw away without taking on unnecessary risk.[3]

4.4 General Form of the Scenario Approach

Throughout the previous sections the scenario approach was a stated as a program that focused on the minimization of a loss function ℓ. A formulation of the scenario approach centered on ℓ is effective, but not necessarily the most efficient statement of the optimization program. In this section we will cover a reformulation of the problem couched in the language of linear programming.

To begin we will restate $\ell(\nu, \delta)$ as a linear cost function $c^T\theta$ where $\theta \in \Theta \in \mathbb{R}^d$. When expressed as a linear cost function we can easily rewrite the scenario program as:

$$SP_n : \ min_{\theta \in \Theta} c^T\theta \ s.t. \ \theta \in \cap_{i=1,\dots,N} \Theta_{\delta_i} \tag{9}$$

When converted to a linear cost function it is not initially clear how each scenario δ plays a role in the optimization. Within linear programming δ shifts to merge into Θ. θ is used to represent (ν, ℓ) and each constraint Θ_δ becomes indexed by some δ. Here we can see that both ℓ and δ have camouflaged themselves, but not truly disappeared. In fact, we are still minimizing ℓ where $\ell \geq \ell(\nu, \delta)$ through θ. Both variables also continue to exist in each Θ_δ which is defined as the set where $\ell \geq \ell(\nu, \delta)$.[3]

It is also important to note that stating the scenario program as a linear program will not harm the assumption of convexity, although some convex functions may need to be rewritten to fit within equation (9).[3]

Since this more general form of the scenario approach does not change any of the assumption previously present in theorem 4.1 we can amend it to the more general setting with a few basic substitutions as:

Theorem 4.4. *$\forall \epsilon \in (0,1)$ and $\beta \in (0,1)$, if N satisfies $N \geq \frac{2}{\epsilon}(ln\frac{1}{\beta} + d - 1)$, then with probability $\geq 1 - \beta$, it holds that $\mathbb{P}\{\delta \in \Delta : \theta^* \notin \Theta_\delta\} \leq \epsilon$*

The theorem above is nearly identical to theorem 4.1 with the sole exception being the way in which we define a violation of the ϵ bound. In this version, we use the constraint $\theta \in \Theta$ in place of the old ℓ based constraint.[3]

4.5 A Deeper Look: Additional Theoretical Results

Following the generalization in the previous section in this section we will cover a few expansions of the theorems we have encountered with a greater degree of detail. In particular, we will cover theorems 4.1 and 4.4 in greater detail.

Definition: Violation Probability

Violation probability is defined as the probability a of the violation set θ occurs, where the violation of $\theta \in \Theta$ is defined as $\{\delta \in \Delta : \theta \notin \Theta_\delta\}$. Using the definition of the violation set, we can write the probability of its occurrence as $V(\theta) := \mathbb{P}\{\delta \in \Delta : \theta \notin \Theta_\delta\}$.[4]

Note: $\{\delta \in \Delta : \theta \notin \Theta_\delta\} \subset \Delta$ while the violation is a probabilistic measure over another distribution $\mathbb{P}\{\delta \in \Delta : \theta \notin \Theta_\delta\}$. It should also be mentioned that these concepts exist independent of optimization.[4]

With the definition of violation probability in hand we will introduce a few important assumptions.

Assumption 1. *Convexity: Θ and Θ_δ, $\delta \in \Delta$, are convex and closed sets[4]*

Assumption 2. *Existence and Uniqueness: $\forall\, m$ and $\forall$ samples $(\delta_1, \delta_2, ..., \delta_m)$, the solution to the scenario program defined in equation* (9) *exists and is unique[4]*

Using assumptions 1 and 2 we can now introduce the generalized theorem of the scenario approach.

Theorem 4.5. *Let $N \geq d$. Then assumptions 1 and 2 imply that [4]*

$$\mathbb{P}^N\{V(\theta^*) > \epsilon\} \leq \sum_{i=0}^{d-1} \binom{N}{i} \epsilon^i (1-\epsilon)^{N-i} \tag{10}$$

An interesting side effect of theorem 4.5 is that it allows us to bound the violation probability $V(\theta^*)$ with a flip of the inequalities in equation (11) using:[4]

$$F_V(\epsilon) := \mathbb{P}^N\{V(\theta^*) \leq \epsilon\} \geq 1 - \sum_{i=0}^{d-1} \binom{N}{i} \epsilon^i (1-\epsilon)^{N-i} \tag{11}$$

Which can be further refined by noticing that the sum present in equations (10) and (11) is equivalent to the cumulative distribution function of a beta distribution. Thus, we restate it as a beta distribution with shape parameters d and $N - d + 1$ as follows: [4][1]

$$f_B(\epsilon) = d\binom{N}{d} \epsilon^{d-1} (1-\epsilon)^{N-d} \tag{12}$$

The form of this beta distribution tied to equation (12) has a compelling impact on the on probability that violation probability exceeds the ϵ bound present in both equations (10) and (11). Beta distributions

embody the property that the bulk of their density can be shifted significantly by altering their parameters. [1] Since equation (12) operates over ϵ, the entire distribution exists over $[0, 1]$. Thus, by invoking the nature of the beta distribution we can push the majority of the distributions density closer to either 0 or 1 by changing the shape parameters through N and d.

Shifting the density of the beta distribution is effectively equivalent to shifting the ϵ bound where a move towards 1 raises the violation probability and a move to 0 does the opposite. Key to these shifts is the relationship between N and d. Increasing N will push the distribution closer to 0, while upping d results in an inverse movement.[4] Such a result is not unexpected given what we already know from theorem 4.1. At their core the paring of theorem 4.5 with equation (12) act together to state that a larger sample is a better approximation of Δ than a small one, and that as the number of parameters present in an optimization increases so does the number of scenarios required.[4]

4.5.1 A Quick Derivation

Having spent the time to describe the generalized theorem of the scenario approach we can show the derivation of theorem 4.3 using just algebraic manipulation.

The process of taking theorem 4.5 to 4.3 can be done in three steps. First we will find a bound for equation (10). Second will we rearrange the bound found in the first step to approach the form of theorem 4.3. Third we will introduce a bound for the form found during the second step.[4]

To begin the first step we note that the basic form of equation (11) is quite similar to theorem 4.3. Given the resemblances we can show theorem 4.1 by bounding the right hand side of equation (11) below with the confidence parameter β. Thus, we want to show $1 - \sum_{i=0}^{d-1} \binom{N}{i}\epsilon^i(1-\epsilon)^{N-i} \geq 1 - \beta$. Alternatively, by canceling the ones and swapping the inequality we can show $\sum_{i=0}^{d-1} \binom{N}{i}\epsilon^i(1-\epsilon)^{N-i} \leq \beta$. With that goal in mind we can begin expanding $\sum_{i=0}^{d-1} \binom{N}{i}\epsilon^i(1-\epsilon)^{N-i}$ using the following sequence[3]:[4]

$$\sum_{i=0}^{d-1} \binom{N}{i}\epsilon^i(1-\epsilon)^{N-i} = 2^{d-1}\sum_{i=0}^{d-1} \binom{N}{i}\frac{\epsilon^i}{2^{d-1}}(1-\epsilon)^{N-i} \tag{13}$$

$$\leq 2^{d-1}\sum_{i=0}^{d-1} \binom{N}{i}(\frac{\epsilon}{2})^i(1-\epsilon)^{N-i} \tag{14}$$

$$\leq 2^{d-1}\sum_{i=0}^{N} \binom{N}{i}(\frac{\epsilon}{2})^i(1-\epsilon)^{N-i} \tag{15}$$

$$= 2^{d-1}(\frac{\epsilon}{2} + (1-\epsilon))^N \tag{16}$$

$$= 2^{d-1}(1 - \frac{\epsilon}{2})^N \tag{17}$$

$$\leq 2^{d-1}e^{-\frac{\epsilon}{2}N} \tag{18}$$

For the second step, we will plug in our β bound and then apply the natural logarithm to the inequality to obtain:[4]

[3]The final inequality in the sequence holds because e^{-x} lies above $1 - x$[4]

$$2^{d-1}e^{-\frac{\epsilon}{2}N} \leq \beta \tag{19}$$

$$(d-1)ln2 - \frac{\epsilon}{2}N \leq ln(\beta) \tag{20}$$

In the third and final step, we rearrange the previous equation to place N on the left side and introduce a final bound which happens to be our desired result[4].[4]

$$N \geq \frac{2}{\epsilon}(ln\frac{1}{\beta} + ln2(d-1)) \tag{21}$$

$$N \geq \frac{2}{\epsilon}(ln\frac{1}{\beta} + d - 1) \tag{22}$$

The algebra present in this transformation may not be of particular importance, but it does help to elucidate the connection between the more formal definitions used in section 3.5 and the more intuitive theorems given earlier on.

5 Connection to Statistical Learning

Previously we glanced at a data driven application of the scenario approach, where we used it to fit a polynomial to a data set. In this section we will expand upon the use of the scenario approach within a data centric environment. More specifically we will show how the scenario approach can be leveraged for statistical learning applications.

5.1 A Brief Introduction to Statistical Learning and Classification

Statistical learning is a broad and very active area of study with a number of different sub-fields, so for the purposes of this paper we will focus on one specific topic: classification. In particular, we will focus on supervised classification.

When studying classification we are presented with the problem of how we can assign a *label* y to an *object* x. Each label y is discrete quantity and each object $x \in \mathbb{R}^p$ is a vector of attributes (or features) that will be used to estimate y. Although the label y can take on many different values in the general classification setting, in the following sections we will assume that we are dealing with binary classification where $y \in \{0, 1\}$.[7]

For the purposes of classification we assume that y is determined by x, but we have no way of knowing the map $y = y(x)$. Therefore, we have to find a way of estimating it via some other function. Such an estimation function is known as a *classifier* and denoted $\hat{y} = \hat{y}(x)$. When creating a classifier our goal is to minimize the number of instances where $y(x) \neq \hat{y}(x)$ in order to construct a classifier with the lowest possible error rate.[7]

[4]We are able to drop the $ln2$ term because $ln2 < 1$[4]

Definition: Probability of Error

Within the statistical learning framework we assume that x occurs according to some probability denoted μ. The degree to which a classifier approximates $y(x)$ is then calculated by checking the probability of x where the classifier is incorrect, or $\mu\{x : y(x) \neq \hat{y}(x)\}$. This probability is known as the probability of error and commonly denoted $PE(\hat{y})$.[7]

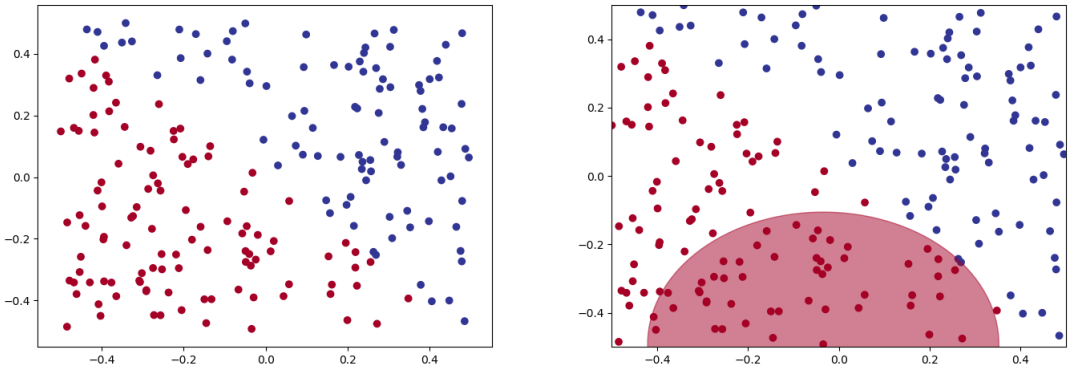
Note: we cannot calculate $PE(\hat{y})$ without knowledge of μ and $y(x)$.[7]

Considering our goal when building a classifier is to minimize our error rate when assigning labels it is tempting to attempt to minimize $PE(\hat{y})$ directly. Sadly, such an approach is impossible due to our lack of knowledge about the underlying data generation process. That being said, it is possible to alter our conception of $PE(\hat{y})$.[7]

One way to adjust $PE(\hat{y})$ is to use a ternary-valued classifier. Ternary-valued classifiers can return a value of "unknown" in addition to the standard $\{0, 1\}$. Allowing a label of "unknown" effectively allows the classifier to abstain from making a decision about a data point. A side effect of an occasional refusal to classify is that we can redefine $PE(\hat{y})$ as $PE(\hat{y_N}) = \mu\{x : \hat{y_N}(x) = 0 \, or \, 1, \, and \, y(x) \neq \hat{y}(x)\}$.[7]

5.2 Guaranteed Error Machines

Guaranteed error machines or GEMs are ternary-valued classifiers that employ the scenario approach to achieve clearly bounded error rates.[2] The complete version of the GEM algorithm can be found Campi's original paper of on the subject, but during the following sections I will present a simplified version of the algorithm.



(a) Example data set where the two regions can be divided by $y = 1/2 - x$

(b) First step of the GEM algorithm

Figure 2

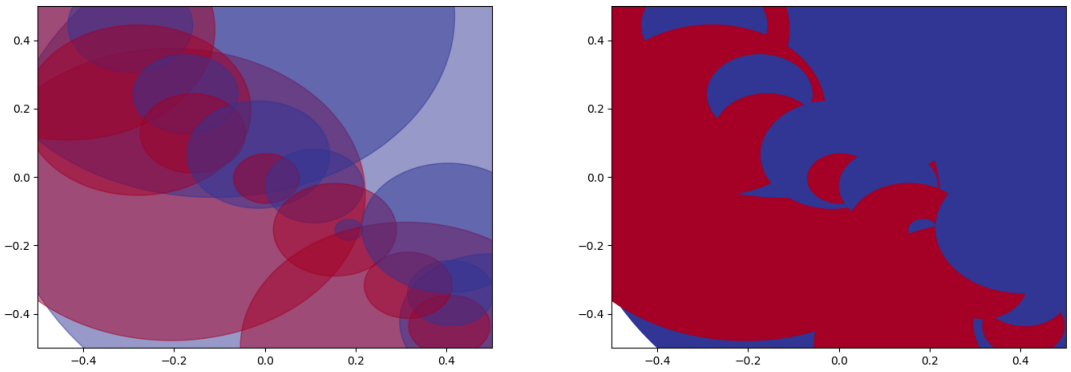
Suppose we are given the data set shown in figure 2.a. We want to build a classifier that correctly divides the two regions by labeling points above the line $y = \frac{1}{2} - x$ as blue and those below the line as red. To construct such a classifier we are going to repeatedly fit a series of disks to the data.[5] Combined, these disks

[5]The full GEM algorithm allows for us to use more than just disks[2]

will define the ternary-valued GEM classifier.

To begin building a GEM classifier we must first select a value d which will define the frequency with which we return the label "unknown."[2][7] In the context of the disk (or in higher dimensions, hyper-sphere) based version of the GEM algorithm, d will define the maximum number of disks that can be fit to the data.[7] Once this cap has been set we can begin our application of the GEM algorithm.

As is illustrated in figure 2.b, the initial disk is fit by choosing a center point and then finding the nearest point of the opposite label. This opposite label point is know as the support point and will be used to define the limit of the disk emanating from a given center. Once the support point has been found and the disk has been fit, all points within the disk are discarded from the data set and the current support point is taken to be the new center point. The process of is then repeated for the new center in order to create another disk (with the opposite label). This alternating sequence continues until we have fit d disks, at which point any areas within the region that are left uncovered will return an "unknown" label.[2][7]



(a) Disks fit by the GEM algorithm

(b) Label regions defined by the GEM algorithm

Figure 3

Once all disks have been fit we can combine them to divide up the region among our three labels. In figure 3.b we can see that the disks ultimately create a region that while imperfect does succeed in roughly representing the divide between the two labels. It should also be mentioned that the GEM classifier used to fit the regions shown in figure 3.b was not given a high tolerance for "unknown" labels, and as a result only returns "unknown" for a small region in the lower left hand corner.

Although it may not be initially obvious, the fitting of each individual disk is a scenario procedure. Thus, the GEM algorithm can be restated as a sequence of optimizations that each seek to maximize the number of points correctly classified before hitting an incorrect classification. As a byproduct of our ability to state the GEM algorithm through the lens of the scenario approach we can apply theorem 4.5 to produce the following theorem:[7][5]

Theorem 5.1. *For a given probability μ with density, the probability distribution of the probability of error*

$PE(\hat{y_N})$ of the GEM algorithm has the relation:

$$F_{PE}(\epsilon) := \mu^N\{PE(\hat{y_N}) \leq \epsilon\} \geq 1 - \sum_{i=0}^{d-1} \binom{N}{i} \epsilon^i (1-\epsilon)^{N-i} \tag{23}$$

Theorem 5.1 shows that the GEM algorithm has an error bound that is identical to the bound shown in theorem 4.5. This result is striking because just as we saw in the previous sections, the bound is not tied to any distribution and in this case holds true regardless of $y(x)$ and μ.[7]

5.3 Extending the GEM Algorithm

After spending some time working with my implementation of the GEM algorithm I began to wonder how it could possibly be extended. Like all classifiers, the GEM algorithm is not perfect, but I wondered whether it would be possible to increase effectiveness of the algorithm by incorporating ideas found elsewhere in statistical learning.

5.3.1 Ensembles

The first extension of the GEM algorithm I implemented was an ensemble version that allowed an arbitrary number of GEM classifiers to be fit to data set. The choice to begin with an ensemble was driven by a curiosity as to whether it is possible to reduce the rate at which "unknown" labels are returned without over fitting to the training data.

Classifier ensembles can be created in a number of different ways, but regardless of architecture all ensembles look to improve performance by combining the strengths of a number of models in order to create a single more powerful model.[10] The ensemble implementation I employ is a random forest type committee. Committee style ensembles allow each classifier to vote on the predicted label. The votes are then tallied and the label with the most votes is returned. [10]

5.3.2 Bagging

With the baseline ensemble setup in place we can begin to add a few complications to the creation of the ensemble with the hope of raise the overall performance level. The first augmentation I considered was bagging.

The goal of bagging is to improve predictive accuracy by reducing the variance of a classifier. We can achieve this aim by averaging our set of observations. This can be done by using a number of different training sets to build separate classifiers whose predictions can be averaged. Under perfect circumstances we could obtain our classifiers by using a multitude of training sets, but considering it is unlikely that we will be in possession of an abundance of training data we can instead turn to the bootstrap.[11] By bootstrapping we can expand a single training set into many by sampling from it. The bootstrapped training sets can then used to fit classifier that can be averaged. We can formalize this process as:[11]

$$\hat{f}_{bag}(x) = \frac{1}{B} \sum_{b=1}^{B} \hat{f}^b(x) \tag{24}$$

5.3.3 Random Forests

The next adaption I chose to add was ability to create a random forest composed of GEM classifiers. The goal of using a random forest in place of a bagging is to hopefully "decorrelate" the classifiers being added to the ensemble.[11]

Much like bagging, random forests are centered around the use of the bootstrap. Just as was the case during the creation of the bagged ensemble, the first step in creating a random forest classifier is the creation of a large number of bootstrapped training sets. We then fit individual classifiers to each of the training sets just as we did when bagging, with one important caveat. When fitting the classifiers we limit the maximum predictors any given classifier can take into account, such that no single classifier is able to use the full set of predictors when being fit. This restriction is put in place to diversify the classifiers being placed in the ensemble.[11]

When applied to decision trees, random forests ensure that not every tree placed in the ensemble is utilizing the same splits. When applied to the GEM algorithm, random forests instead alter the way in which the distance between the center and nearest support point is calculated.

5.4 Methodology

After implementing both bagging and random forest adaptations of the GEM algorithm I tested both methods in addition to the standard form of the GEM algorithm on both synthetically generated data and real world data set.

The synthetic data set was created by generating five features independently of one another by taking random draws on the interval $[-0.5, 0.5]$. A label is then assigned to each data point based on whether or not the sum of the five features was greater than zero. This data generation process was chosen as a baseline to represent an unambiguous classification problem where the two classes are separable using a hyperplane.

Each method was tested on the synthetic data by fitting 100 copies of each classifier to 100 different randomly generated data sets. True positive, true negative, false positive, and false negative numbers were then calculated for each classifier by testing their predictions on a unique test sets.

To test the methods on a less clear cut "real world" data set I selected the famous Wisconsin breast cancer detection data set. The Wisconsin breast cancer data set is well known within the machine learning community and is commonly used for both teaching and method evaluation. The data set is comprised of an sample code (ID), nine numeric features, and a binary class value (representing benign and malignant cancers).[8] Before use the ID number was dropped and the data was scaled to prevent distance calculations being skewed or distorted, but the data was otherwise left in its original form. A single observation (out of 700) was also dropped due to the presence of an NA value in "Bare Nuclei" feature.

Each method was then tested on the cancer data by running 250 simulations for each method, where a portion of the data set was randomly drawn and used as the training set and the remaining portion was used for testing. True positive, true negative, false positive, and false negative numbers were calculated for each

pairing of classifier to random data set, as was the case with the synthetic data.

5.5 Results

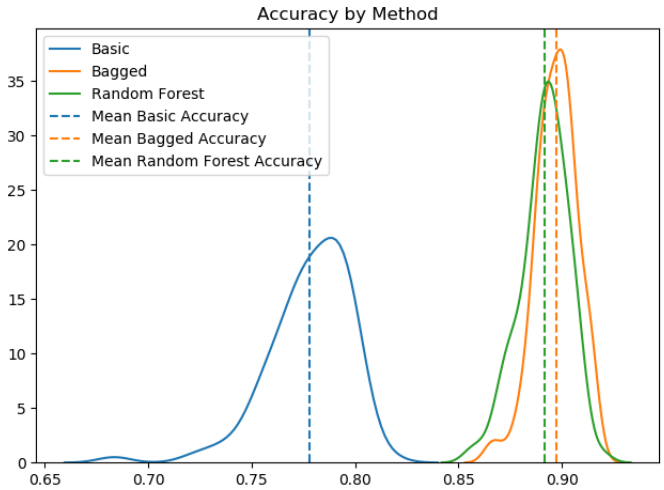


Figure 4: Distribution of accuracy scores for each method using synthetic data

Looking at the results from the tests carried out on synthetic data we can clearly see that the two ensemble methods out performance the standard GEM algorithm by a wide margin. It is, however, intriguing to note that in this case the bagged ensemble just slightly exceeds the random forest ensemble in terms of accuracy. Initially this appears to be an unexpected result, but recall that the label of each data point equally weighs all five features. Since each feature is of equal importance, restricting the set of features that can be used when fitting the classifiers is not only unhelpful, but outright harmful.

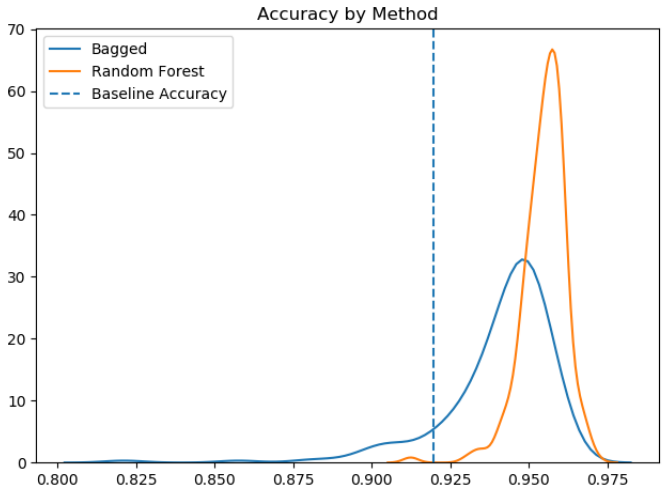
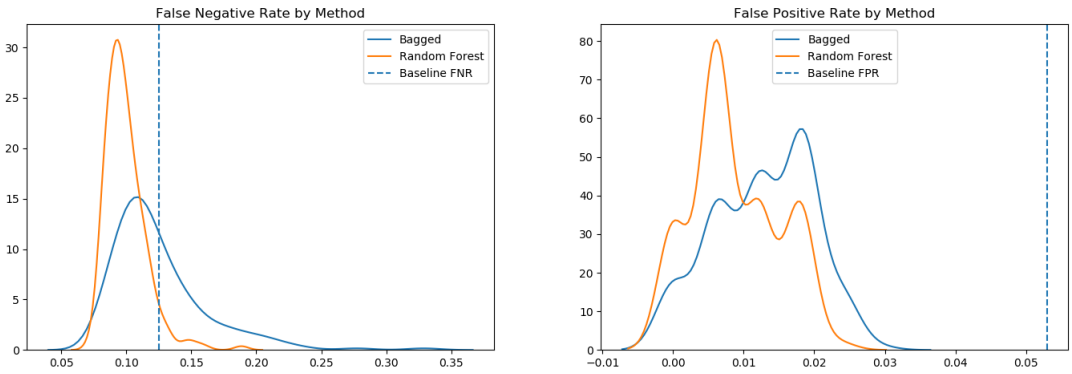


Figure 5: Distribution of accuracy scores for each method using cancer data

Across the simulations carried out on the cancer data set, we can see a more expected pattern occur as

the random forest beats out the bagged ensemble in each metric shown. This is likely caused by interactions between features that are causing unhelpful correlations between classifiers. In this case the random forest is able to avoid the pitfall of such interactions slightly better than the bagged approach and post higher (and more concentrated) scores as a result.



(a) FNR by method on cancer data

(b) FPR by method on cancer data

Figure 6

Perhaps the most intriguing results are those shown in the error plots in figure 6. Both ensemble methods dramatically outshine the standard GEM algorithm. Although, no research has truly explored this phenomenon, theorem 5.1 appears to provide at least some insight into the matter. With theorem 5.1 we can place error bounds on each of the classifiers within our ensembles, where N remains constant across all involved classifiers and d is same in the bagged and standard cases, while being reduced in the random forest version. Thus, theorem 5.1 would seem to imply the possibility of the random forest having a lower error rate (assuming that it is possible to treat an ensemble of scenario programs as a single grand program - a result not yet proven). The empirical results of these simulations appear to show this may be the case (especially given the striking resemblance the distributions shown in figure 6.a have to the Beta distribution described in theorem 5.1), but further research is needed to reach a strong conclusion on the matter.

Aside from the connection between the empirical results and theory outlined in previous sections it is exciting to see that the results indicate the potential efficacy of the scenario approach in the domain of statistical learning. While this section is focused on GEMs it can also be seen as a proof of concept for the application of the scenario approach to statistical methods. From a production standpoint it is also crucial to highlight that even when using the ensembled versions of the GEM algorithm the results shown in figure 6 imply a clear connect to the theoretical bounds allowing for readily explainable worse case performance levels. Such lower bounds could be invaluable when presenting a predictable model.

5.6 Future Improvements

The promise shown by the bagged and random forest versions of the GEM algorithm begs the question of how other statistical learning techniques could impact the power of the GEM algorithm. In particular, I think boosting could further improve upon the ensembles I investigated by more intelligently weighting the individual classifiers within the ensemble by building upon one another to better classify difficult examples using the following equation:[10]

$$\hat{f}(x) = \sum_{b=1}^{B} \lambda \hat{f}^b(x) \tag{25}$$

To find such λ weights we need to in case our classifiers within an algorithmic shell. In the context of GEMs we can imagine adapting the well know "AdaBoost" algorithm to work with GEMs. We could do this by using the algorithm outlined below.[6][10]

AdaBoost for GEMs

1. Initialize a set of equal weights $w_i = 1/N$ such that each of the N observations has an initially equivalent weight
2. Then for each of the M classifiers to be included in the ensemble:
   (a) Fit a weak GEM classifier to the weighted training set
   (b) Compute the error of the classifier where $err_m = \frac{\sum_{i=1}^{N} w_i I(y_i \neq G_m(x_i))}{\sum_{i=1}^{N} w_i}$
   (c) Use the error value to compute $\alpha_m = log((1 - err_m)/err_m)$
   (d) Update the weights using the alpha value as $w_i \leftarrow w_i * exp(\alpha_m * I(y_i \neq G_m(x_i))) \, \forall \, w_i$
3. Return the sign of the sum $\sum_{m=1}^{M} \alpha_m G_m(x)$ or 0 if the sum is 0

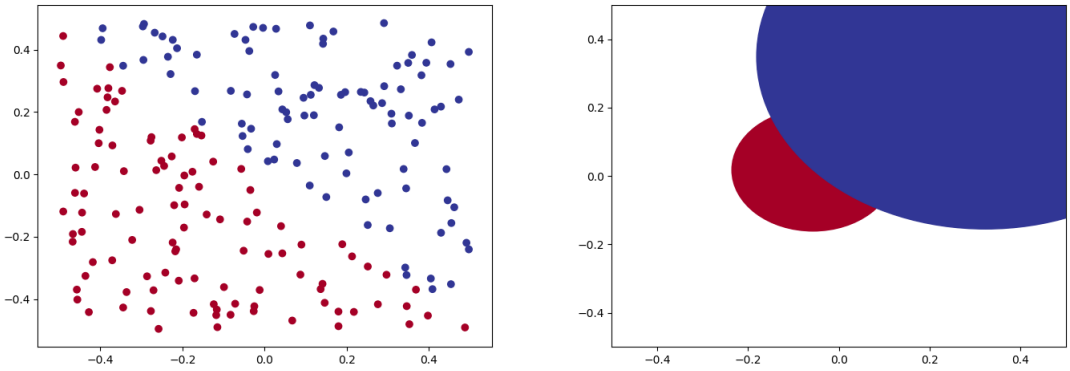
The reasoning behind the adaptation of AdaBoost for use with GEM algorithms stems from the success that AdaBoost has shown when used with tree based classifiers. Given the process of fitting a GEM is not too dissimilar to that of fitting a decision tree it seems reasonable to believe AdaBoost is prime candidate for boosting GEMs.[7] In order to meld the two methods we do, however, need to include one significant alternation to each of GEM classifiers in the ensemble: make them weak. A weak classifier is simply a classifier that is not very good and is likely only slightly better than randomly guessing.[10] For the tree based version of AdaBoost we create weak tree classifiers by heavily pruning them and reducing them to "stubs" that include just a single split. For GEMs we can achieve an analogous result by fitting just two regions and allowing the remaining space to be considered uncertain. The figures below show the results of an example weak GEM classifier on a basic synthetic data set.

Initial prototypes of the AdaBoost adaptation have shown signs of similar levels of improvement as the bagged and random forest ensembles of GEMs, but further testing before we can say with confidence that this form of boosting fits well with GEMs.[8]

[6]The algorithm below is equivalent to the AdaBoost.M1 algorithm presented in *ESL* aside from two notable differences: this algorithm is allowed to return 0 in rare cases and the classifiers that build the ensemble must be derivative forms of GEMs[10]

[7]The similarity in the fitting mostly refers to the way in which the two methods attempt to first find the most impactful split or region before subsequently moving on to smaller and more difficult to classify regions

[8]My personal implementation of the algorithm appears to have a few bugs. Do to such problems I have not included the results of GEM boosting in this paper. With luck I will release those results sometime in the future



(a) Data generated using the same process as figure 2

(b) Resulting weak GEM classifier

Figure 7

6 Conclusion

All things considered the scenario approach is strong tool with a wide variety of applications. From portfolio optimization to statistical learning the scenario approach is capable of it all. In fact, the approach is more than capable in a number of settings where it may even out perform existing methods due to its computational advantages and easily determinable bounds. Given the upside of these benefits, with further research and streamlining the scenario approach shows the potential to shine in a number of areas.

That being said, the scenario approach is by no means a universal multi-tool with the ability to solve all problems. The necessary setup of a sampling method and reliance on the assumption of convexity are limitations that must be considered when deciding to use the scenario approach.

These limitations should not, however, deter those hoping to use the scenario approach in their work. When scenarios can be created with relative ease (as is the case with GEMs) the approach is both efficient and flexible, while still being imbued with the added benefit that strength of the optimization is guaranteed. It should also be noted that when interpretability is desired the scenario approach may prove useful.[9] When used in combination with non "black box" methods the scenario approach may be strikingly interpretable.

Future work investigation how to best use the scenario approach for parameter selection is needed to flush out the application of the scenario approach to existing statistical methodologies like the the statistical learning methods used for classification and predictive modeling. With such advancements the approach could be more easily applied to problems like that shown in 4.1 without the used of methods like GEMs that are built with the scenario approach based optimization in mind.

7 Acknowledgments

I would like to thank all those who have supported me throughout the comps process. From family to friends and of course professors you all played an integral part in my completion of this project.

[9]Recent studies of interpretability within machine learning have shown that interpretable methods are not necessarily less effective than more complex "black boxes"[12]

I would also like to give special thanks to the authors of *Introduction to the Scenario Approach* for both putting out a great deal of well written work on the topic, and providing a some guidance into my own investigation of the GEM algorithm.

References

[1] Christopher M. Bishop. *Pattern Recognition and Machine Learning*. Springer, 2006.

[2] Marco C. Campi. *Classification with guaranteed probability of error*. 2010.

[3] "Chapter 1: Introduction: Uncertainty in optimization and the scenario approach." In: *Introduction to the Scenario Approach*, pp. 1–20. DOI: 10.1137/1.9781611975444.ch1. eprint: https://epubs.siam.org/doi/pdf/10.1137/1.9781611975444.ch1. URL: https://epubs.siam.org/doi/abs/10.1137/1.9781611975444.ch1.

[4] "Chapter 3: Theoretical results and their interpretation." In: *Introduction to the Scenario Approach*, pp. 33–48. DOI: 10.1137/1.9781611975444.ch3. eprint: https://epubs.siam.org/doi/pdf/10.1137/1.9781611975444.ch3. URL: https://epubs.siam.org/doi/abs/10.1137/1.9781611975444.ch3.

[5] "Chapter 5: Proofs." In: *Introduction to the Scenario Approach*, pp. 55–66. DOI: 10.1137/1.9781611975444.ch5. eprint: https://epubs.siam.org/doi/pdf/10.1137/1.9781611975444.ch5. URL: https://epubs.siam.org/doi/abs/10.1137/1.9781611975444.ch5.

[6] "Chapter 6: Region estimation models." In: *Introduction to the Scenario Approach*, pp. 67–78. DOI: 10.1137/1.9781611975444.ch6. eprint: https://epubs.siam.org/doi/pdf/10.1137/1.9781611975444.ch6. URL: https://epubs.siam.org/doi/abs/10.1137/1.9781611975444.ch6.

[7] "Chapter 7: Application to statistical learning." In: *Introduction to the Scenario Approach*, pp. 79–87. DOI: 10.1137/1.9781611975444.ch7. eprint: https://epubs.siam.org/doi/pdf/10.1137/1.9781611975444.ch7. URL: https://epubs.siam.org/doi/abs/10.1137/1.9781611975444.ch7.

[8] Dheeru Dua and Casey Graff. *UCI Machine Learning Repository*. 2017. URL: http://archive.ics.uci.edu/ml.

[9] Laurent El Ghaoni. *Convex Optimization Lecture Notes for EE 227BT Draft, Fall 2013*. 2013. URL: https://people.eecs.berkeley.edu/~elghaoui/Teaching/EE227BT/LectureNotes_EE227BT.pdf.

[10] Trevor Hastie, Robert Tibshirani, and Jerome Friedman. *The elements of statistical learning: data mining, inference and prediction*. 2nd ed. Springer, 2009. URL: http://www-stat.stanford.edu/~tibs/ElemStatLearn/.

[11] Gareth James et al. *An Introduction to Statistical Learning: with Applications in R*. Springer, 2013. URL: https://faculty.marshall.usc.edu/gareth-james/ISL/.

[12] Erico Tjoa and Cuntai Guan. "A Survey on Explainable Artificial Intelligence (XAI): Towards Medical XAI." In: *CoRR* abs/1907.07374 (2019). arXiv: 1907.07374. URL: http://arxiv.org/abs/1907.07374.