

A Gentle Introduction to Conformal Prediction and Distribution-Free Uncertainty Quantification

Anastasios N. Angelopoulos and Stephen Bates

July 4, 2021

Abstract

Black-box machine learning methods are now routinely used in high-risk settings, like medical diagnostics, which demand uncertainty quantification to avoid consequential model failures. Distribution-free uncertainty quantification (distribution-free UQ) is a user-friendly paradigm for creating statistically rigorous confidence intervals/sets for such predictions. Critically, the intervals/sets are valid without distributional assumptions or model assumptions, with explicit guarantees with finitely many datapoints. Moreover, they adapt to the difficulty of the input; when the input example is difficult, the uncertainty intervals/sets are large, signaling that the model might be wrong. Without much work, one can use distribution-free methods on any underlying algorithm, such as a neural network, to produce confidence sets guaranteed to contain the ground truth with a user-specified probability, such as 90%. Indeed, the methods are easy-to-understand and general, applying to many modern prediction problems arising in the fields of computer vision, natural language processing, deep reinforcement learning, and so on. This hands-on introduction is aimed at a reader interested in the practical implementation of distribution-free UQ, including conformal prediction and related methods, who is not necessarily a statistician. We will include many explanatory illustrations, examples, and code samples in Python, with PyTorch syntax. The goal is to provide the reader a working understanding of distribution-free UQ, allowing them to put confidence intervals on their algorithms, with one self-contained document.

Contents

1 Conformal Prediction	3
1.1 Instructions for Conformal Prediction	4
2 Examples of Conformal Procedures	5
2.1 Classification with Adaptive Prediction Sets	5
2.2 Conformalized Quantile Regression	6
2.3 Conformalizing Scalar Uncertainty Estimates	8
2.3.1 The Estimated Standard Deviation	8
2.3.2 Other 1-D Uncertainty Estimates	8
2.4 Conformalizing Bayes	9
3 Risk-Controlling Prediction Sets	11
3.1 Motivating Example: Image Segmentation	11
3.2 Instructions for Risk-Controlling Prediction Sets	12
3.3 Remarks	14
4 Evaluating Prediction Sets	14
4.1 Basic Checks	14
4.2 Evaluating Adaptiveness	15
A Theorem and Proof: Coverage Property of Conformal Prediction	18

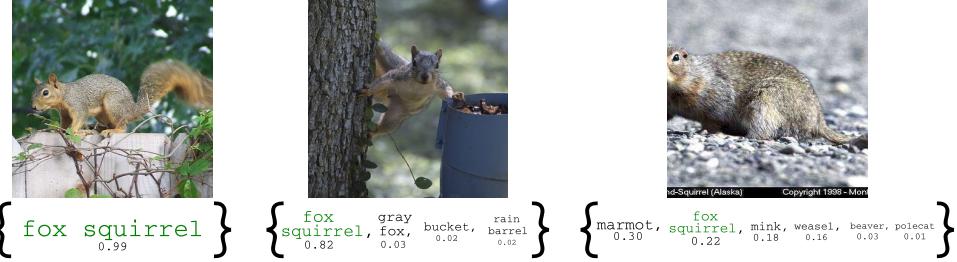


Figure 1: **Prediction set examples on Imagenet.** We show three progressively more difficult examples of the class `fox squirrel` and the prediction sets (i.e., $\mathcal{T}(X)$) generated by conformal prediction.

1 Conformal Prediction

As a first, important example of distribution-free UQ, conformal prediction [1–3] is a straightforward way to generate confidence sets for any model. We will introduce it with a short, pragmatic image classification example, and follow up in later paragraphs with a general explanation. The high-level outline of conformal prediction is as follows. First, we begin with a fitted predicted model (such as a neural network classifier) which we will call \hat{f} . Then, we will create confidence sets (a set of possible labels) for this classifier using a small amount of additional *calibration data*—we will sometimes call this this *calibration step*.

Formally, suppose we have an image X and class $Y \in \{1, \dots, K\}$. We begin with a classifier that outputs softmax scores for each class: $\hat{f}(x) \in [0, 1]^K$. Then, we take a moderate number (e.g., 500) fresh data points $(X_1, Y_1), \dots, (X_n, Y_n)$ that were not used for training for use as calibration data. Using \hat{f} and the calibration data, we seek to construct a *confidence set* of possible labels $\mathcal{T}(X) \subset \{1, \dots, K\}$ that is valid in the following sense:

$$1 - \alpha \leq \mathbb{P}(Y_{n+1} \in \mathcal{T}(X_{n+1})) \leq 1 - \alpha + \frac{1}{n+1}, \quad (1)$$

where (X_{n+1}, Y_{n+1}) is a fresh test point from the same distribution. In words, the probability that the confidence set contains the correct label is almost exactly $1 - \alpha$; we call this property *marginal coverage*. See Figure 1 for examples of $\mathcal{T}(X)$ on the Imagenet dataset.

To construct \mathcal{T} from \hat{f} and the calibration data, we will perform a simple calibration step that requires only 3 lines of python code; see the right panel of Figure 2. We now describe the calibration step in more detail, introducing some terminology that will be helpful later on. First, we set the *conformal score* $s_i = 1 - \hat{f}(X_i)_{Y_i}$ to be one minus the softmax output of the true class (if the model is badly wrong, s_i will be large). Next comes the critical step: define \hat{q} to be the $\lceil (n+1)(1-\alpha) \rceil / n$ empirical quantile of s_1, \dots, s_n (this is essentially the $1 - \alpha$ quantile, but with a small correction). Finally, for a new test data point (where X_{n+1} is known but Y_{n+1} is not), create a prediction set $\mathcal{T}(X_{n+1}) = \{y : \hat{f}(X_{n+1})_y > 1 - \hat{q}\}$ that includes all classes with a high enough softmax output (see Figure 2). Remarkably, this algorithm gives prediction sets that are guaranteed to provide coverage, no matter what (possibly incorrect) model is used or what the (unknown) distribution of the data is.



Figure 2: Illustration of conformal prediction with matching PyTorch code.

Remarks

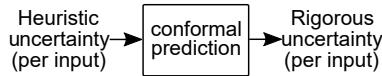
Let us think about the interpretation of \mathcal{T} . The function \mathcal{T} is *set-valued*—it takes in an image, and it outputs a set of classes as in Figure 1. The model’s softmax outputs help to generate the output set. This method constructs a different output set *adaptively to each particular input*. The sets become larger when the model is uncertain or the image is intrinsically hard. This is a property we want, because the size of the set gives you an indicator of the model’s certainty. Furthermore, $\mathcal{T}(X)$ can be interpreted as a set of plausible classes that the image could be assigned to. Finally, $\mathcal{T}(X)$ is a *valid* prediction set, meaning it satisfies Eq. 1.¹ These properties of \mathcal{T} translate naturally to other machine learning problems, like regression, as we will soon see.

With an eye towards generalization, let us review in detail what happened in our classification problem. To begin, we were handed a model that had an inbuilt, but heuristic, notion of uncertainty: softmax outputs. The softmax outputs attempted to measure the conditional probability of each class; in other words, the j th entry of the softmax vector estimated $\mathbb{P}(Y = j | X = x)$, conditionally on an input image x . However, we had no guarantee that the softmax outputs were any good; they may have been arbitrarily overfit or otherwise untrustworthy. Therefore, instead of taking the softmax outputs at face value, we used the holdout set to adjust for their deficiencies.

The holdout set contained $n \approx 500$ fresh data points that the model never saw during training, which allowed us to get an honest appraisal of its performance. The adjustment involved computing conformal scores, which are designed to grow when the model is uncertain, but are not confidence intervals on their own. In our case, the conformal score was one minus the softmax output of the true class, but in general, the score can be any function of X and Y . We then took \hat{q} to be roughly the $1 - \alpha$ quantile of the scores. In this case, the quantile has a simple interpretation—at least 90% of ground truth softmax outputs are guaranteed to be above the level $1 - \hat{q}$ (we prove this rigorously in Appendix A). Taking advantage of this fact, at test-time, we got the softmax outputs of a new image X and collected all classes with outputs above $1 - \hat{q}$ into a prediction set $\mathcal{T}(X)$. Since the softmax output of the new true class Y is guaranteed to be above $1 - \hat{q}$ with probability at least 90%, we finally got the guarantee in Eq. (1).

1.1 Instructions for Conformal Prediction

As we said during the summary, conformal prediction is not specific to softmax outputs or classification problems. In fact, conformal prediction can be seen as a method for taking **any heuristic notion of uncertainty** from **any model** and converting it to a rigorous one (see the diagram below). Conformal prediction does not care if the underlying prediction problem is discrete/continuous or classification/regression.



We next outline conformal prediction for a general input X and output Y (not necessarily discrete).

1. Identify a heuristic notion of uncertainty using the pre-trained model.
2. Define the score function $s(X, Y) \in \mathbb{R}$. (Larger scores encode worse agreement between X and Y .)
3. Compute \hat{q} as the $\frac{\lceil(n+1)(1-\alpha)\rceil}{n}$ quantile of the calibration scores $s_1 = s(X_1, Y_1), \dots, s_n = s(X_n, Y_n)$.
4. Use this quantile to form the prediction sets for new examples, $\mathcal{T}(X) = \{y : s(X, y) \leq \hat{q}\}$.

As before, these sets satisfy the validity property in (1), for any (possibly uninformative) score function and (possibly unknown) distribution of the data. We formally state the coverage guarantee next.

¹Due to the discreteness of Y , a small modification involving tie-breaking is needed to additionally satisfy the upper bound (see [4] for details; this randomization is usually ignored in practice). We will henceforth ignore such tie-breaking.

Theorem 1 (Conformal calibration coverage guarantee). Suppose $(X_i, Y_i)_{i=1,\dots,n+1}$ are i.i.d. and define \hat{q} as in step 3 above and $\mathcal{T}(X)$ as in step 4 above. Then the following holds:

$$P\left(Y_{n+1} \in \mathcal{T}(X_{n+1})\right) \geq 1 - \alpha.$$

See Appendix A for a proof and a statement that includes the upper bound in (1).

Choice of score function

Upon first glance, this seems too good to be true, and a skeptical reader might ask the following question:

How is it possible to construct a statistically valid prediction set even if the heuristic notion of uncertainty of the underlying model is arbitrarily bad?

Let's give some intuition to supplement the mathematical understanding from the proof in Appendix A. If the scores s_i encode information about the model's performance, then the sets will be smaller for easy inputs and bigger for hard ones. If they do not, then the sets may not be useful. For example, they could have a fixed size that does not depend on the input, but still provide marginal coverage. This illustrates an important underlying fact about conformal prediction: although the guarantee always holds, **the usefulness of the prediction sets is primarily determined by the score function**. This should be no surprise—the score function incorporates almost all the information we know about our problem and data, including the underlying model itself. For example, the main difference between applying conformal prediction on classification problems versus regression problems is the choice of score. There are also many possible score functions for a single underlying model, which have different properties. Therefore, constructing the right score function is an important engineering choice. We will next show a few examples of good score functions.

2 Examples of Conformal Procedures

In this section we give examples of conformal prediction applied in many settings, with the goal of providing the reader a bank of techniques to practically deploy. Note that we will focus only on one-dimensional Y in this section. High-dimensional Y , such as in image-to-image regression, is better handled by an extension of conformal prediction, Risk-Controlling Prediction Sets, that we will discuss in Section 3.

2.1 Classification with Adaptive Prediction Sets

Let's begin our sequence of examples with an improvement to the classification example in Section 1. The previous method produces prediction sets with the smallest average size [5], but it tends to undercover hard

```


# Get scores. calib_X.shape[0] == calib_Y.shape[0] == n
sorted, pi = model(calib_X).softmax(dim=1).sort(dim=1,descending=True)
scores = sorted.cumsum(dim=1).gather(1,pi.argsort(1))[range(y.shape[0]),y]
# Get the score quantile
qhat = torch.quantile(scores,np.ceil((n+1)*(1-alpha))/n)
# Deploy (output=list of length n, each element is tensor of classes)
test_sorted, test_pi = model(test_X).softmax(dim=1).sort(dim=1,descending=True)
sizes = (test_sorted.cumsum(dim=1) > qhat).int().argmax(dim=1)
prediction_sets = [ pi[i][0:(sizes[i]+1)] for i in range(sizes.shape[0])]
```

Figure 3: PyTorch code for adaptive prediction sets.

subgroups and overcover easy ones. Here we develop a different method, called *adaptive prediction sets* that avoids this problem. We will follow [6] and [4]. PyTorch code to run this family of procedures is at [this link](#).

As motivation for this new procedure, note that if the softmax outputs $\hat{f}(X)$ were a perfect model of $Y|X$, we would greedily include the top-scoring classes until their total mass just exceeded $1 - \alpha$. Formally, we can describe this oracle algorithm as

$$\{\pi_1, \dots, \pi_k\}, \text{ where } k = \inf \left\{ k : \sum_{j=1}^k \hat{f}(X)_{\pi_j} \geq 1 - \alpha \right\},$$

and π is the permutation of $\{1, \dots, K\}$ that sorts $\hat{f}(X)$ from most likely to least likely. In practice, however, this procedure fails to provide coverage, since $\hat{f}(X)$ is not perfect; it only provides us a heuristic notion of uncertainty. Therefore, we will use conformal prediction to turn this into a rigorous notion of uncertainty.

To proceed, we define a score function inspired by the oracle algorithm:

$$s(X, Y) = \sum_{j=1}^k \hat{f}(X)_{\pi_j}, \text{ where } Y = \pi_k.$$

In other words, we greedily include classes in our set until we reach the true label, then we stop. Unlike the score from Section 1, this one utilizes the softmax outputs of all classes, not just the true class.

The next step, as in all conformal procedures, is to set $\hat{q} = \text{quantile}(s_1, \dots, s_n ; \frac{(n+1)(1-\alpha)}{n})$. Having done so, we will form the prediction sets

$$\mathcal{T}(X) = \{\pi_1, \dots, \pi_k\}, \text{ where } k = \inf \left\{ k : \sum_{j=1}^k \hat{f}(X)_{\pi_j} \geq \hat{q} \right\}. \quad (2)$$

Figure 3 shows PyTorch code to implement this method. As usual, these uncertainty sets (with tie-breaking)

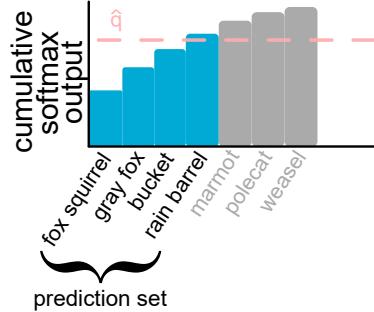


Figure 4: A visualization of the adaptive prediction sets algorithm in Eq. (2). Classes are included from most to least likely until their cumulative softmax output exceeds the quantile.

satisfy (1). See [4] for details.

2.2 Conformalized Quantile Regression

So far, we have only discussed classification, where Y is discrete. We will next show how to incorporate uncertainty into regression problems with a continuous output Y , following the algorithm in [7]. To begin, we use quantile regression as a heuristic notion of uncertainty, since it is often the most naturally implemented form of uncertainty available for continuous-valued problems. As a reminder, a quantile regression algorithm attempts to learn the γ quantile of $Y|X$ for each value of X . We will call the true quantile $t_\gamma(x)$ and the fitted model $\hat{t}_\gamma(x)$. Since by definition $Y|X$ lands below $t_{0.05}(x)$ with 5% probability and above $t_{0.95}(x)$ with

```

# model(X)[:,0]=lower quantile, and model(X)[:,1]=upper quantile
scores = max(model(calib_X)[:,0]-calib_Y,model(calib_X)[:,1]-calib_Y)
# Get the score quantile
qhat = torch.quantile(scores,np.ceil((n+1)*(1-alpha))/n)
# Deploy (represent sets as tuple of lower and upper endpoint)
prediction_sets = (model(test_x)[:,0]-qhat, model(test_x)[:,1]+qhat)

```

Figure 5: PyTorch code for conformalized quantile regression.

5% probability, we would expect the interval $[\hat{t}_{0.05}, \hat{t}_{0.95}]$ to have approximately 90% coverage. However, we cannot be sure, because we do not know how accurate our quantile regression procedure was. Therefore, we use the estimated quantiles as our heuristic notion of uncertainty, and then use conformal prediction to rigorously calibrate it. We include a PyTorch pseudocode version of this algorithm in Figure 5.

After training an algorithm to output two such quantiles (this can be done with a standard loss function, see below), $\hat{t}_{\alpha/2}$ and $\hat{t}_{1-\alpha/2}$, we can define the score function to be the projective distance from Y onto $[\hat{t}_{\alpha/2}, \hat{t}_{1-\alpha/2}]$,

$$s(X, Y) = \max \{ \hat{t}_{\alpha/2}(X) - Y, Y - \hat{t}_{1-\alpha/2}(X) \}.$$

After computing the scores on our calibration set and setting $\hat{q} = \text{quantile}(s_1, \dots, s_n ; \frac{\lceil (n+1)(1-\alpha) \rceil}{n})$, we can form valid prediction sets by taking

$$\mathcal{T}(X) = [\hat{t}_{\alpha/2}(X) - \hat{q}, \hat{t}_{1-\alpha/2}(X) + \hat{q}] . \quad (3)$$

Intuitively, the set $\mathcal{T}(X)$ just grows or shrinks the distance between the quantiles by \hat{q} to achieve coverage.

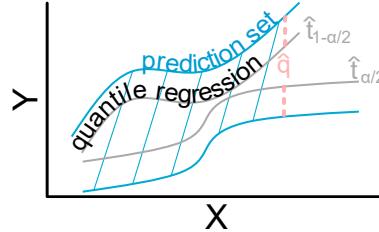
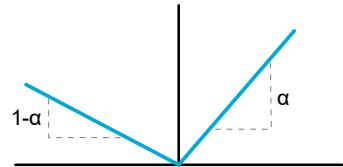


Figure 6: A visualization of the conformalized quantile regression algorithm in Eq. (3). We adjust the upper and lower quantiles produced by quantile regression by the constant \hat{q} , picked during the calibration step.

As before, \mathcal{T} satisfies the coverage property in Eq. (1). However, unlike our previous example in Section 1, \mathcal{T} is no longer a set of classes, but instead a *continuous interval* in \mathbb{R} . Quantile regression is not the only way to get such continuous-valued intervals. However, it is often the best way, especially if α is known in advance. The reason is because the intervals generated via quantile regression even without conformal prediction, i.e. $[\hat{t}_{\alpha/2}, \hat{t}_{1-\alpha/2}]$, have good coverage to begin with. Furthermore, they have asymptotically valid conditional coverage (a concept we will explain later in Section 4). These properties propagate through the conformal procedure and lead to prediction sets with good performance.

One attractive feature of quantile regression is that it can easily be added on top of any base model simply by changing the loss function to a *quantile loss* (informally referred to as a *pinball loss*),

$$L_\gamma(\hat{t}_\gamma, Y) = (Y - \hat{t}_\gamma)\gamma \mathbb{1}\{Y > \hat{t}_\gamma\} + (\hat{t}_\gamma - Y)(1 - \gamma) \mathbb{1}\{Y \leq \hat{t}_\gamma\}.$$



The reader can think of quantile regression as a generalization of L1-norm regression: when $\gamma = 0.5$, the loss function reduces to $L_{0.5} = |\hat{t}_\gamma(X) - Y|$, which encourages $\hat{t}_{0.5}(X)$ to converge to the conditional median. Changing γ just modifies the L1 norm as in the illustration above to target other quantiles. In practice, one can just use a quantile loss instead of MSE at the end of any algorithm, like a neural network, in order to regress to a quantile.

2.3 Conformalizing Scalar Uncertainty Estimates

2.3.1 The Estimated Standard Deviation

As an alternative to quantile regression, our next example is a different way of constructing prediction sets for continuous Y with a less rich but more common notion of heuristic uncertainty: an estimate of the standard deviation $\hat{\sigma}(x)$. For example, one can produce uncertainty scalars by assuming $Y | X$ follows some parametric distribution—like a Gaussian distribution—and training a model to output the mean and variance of that distribution. To be precise, in this setting we assume $Y|X \sim \mathcal{N}(\mu, \sigma)$, and we have models $\hat{f}(X)$ and $\hat{\sigma}(X)$ (usually the same neural network) trained to maximize the likelihood of the data with respect to $\mathbb{E}[Y | X]$ and $\sqrt{\text{Var}[Y|X]}$ respectively. Then, $\hat{f}(X)$ gets used as the point prediction and $\hat{\sigma}(X)$ gets used as the uncertainty. This strategy is so common that it is commoditized: there are inbuilt PyTorch losses, such as `GaussianNLLLoss`, that enable training a neural network this way. However, we usually know Y isn't conditionally Gaussian, so even if we had infinite data, $\hat{\sigma}(X)$ would not necessarily be reliable. We can use conformal prediction to turn this heuristic uncertainty notion into rigorous prediction intervals, which we will describe in detail next.

2.3.2 Other 1-D Uncertainty Estimates

More generally, we assume there is a function $u(x)$ such that larger values encode more uncertainty. This single number can have many interpretations beyond the standard deviation. For example, one instance of an uncertainty scalar simply involves the user creating a model for the magnitude of the residual. In that setting, the user would first fit a model \hat{f} that predicts Y from X . Then, they would fit a second model \hat{r} (possibly the same neural network), that predicts $|Y - \hat{f}(X)|$. If \hat{r} were perfect, we would expect the set $[\hat{f}(X) - \hat{r}(X), \hat{f}(X) + \hat{r}(X)]$ to have good coverage. However, our learned model of the error \hat{r} is often quite poor in practice.

There are many more such uncertainty scalars than we can discuss in this document in detail, including

1. measuring the variance of $\hat{f}(X)$ across an ensemble of models,
2. measuring the variance of $\hat{f}(X)$ when randomly dropping out a fraction of nodes in a neural net,
3. measuring the variance of $\hat{f}(X)$ to small, random input perturbations,
4. measuring the variance of $\hat{f}(X)$ over different noise samples input to a generative model,
5. measuring the magnitude of change in $\hat{f}(X)$ when applying an adversarial perturbation, etc.

These cases will all be treated the same way. There will be some point prediction $\hat{f}(X)$, and some uncertainty scalar $u(X)$ that is large when the model is uncertain and small otherwise (in the residual setting, $u(X) := \hat{r}(X)$, and in the Gaussian setting, $u(X) := \hat{\sigma}(X)$). We will proceed with this notation for the sake of generality, but the reader should understand that u can be replaced with any function.

Now that we have our heuristic notion of uncertainty in hand, we can define a score function,

$$s(X, Y) = \frac{|Y - \hat{f}(X)|}{u(X)}.$$

```

# model(X)[:,0]~E(Y|X), and model(X)[:,1]~stddev(Y|X)
scores = abs(model(calib_X)[:,0]-calib_Y)/model(calib_X)[:,1]
# Get the score quantile
qhat = torch.quantile(scores,np.ceil((n+1)*(1-alpha))/n)
# Deploy (represent sets as tuple of lower and upper endpoint)
muhat, stdhat = (model(test_x)[:,0],model(test_x)[:,1])
prediction_sets = (muhat-stdhat*qhat, muhat+stdhat*qhat)

```

Figure 7: PyTorch code for conformalized uncertainty scalars.

This score function has a natural interpretation: it is a multiplicative correction factor of the uncertainty scalar (i.e., $s(X, Y)u(X) = |Y - \hat{f}(X)|$). As before, taking \hat{q} to be the $\frac{(1-\alpha)(n+1)}{n}$ quantile of the calibration scores guarantees us that for a new example,

$$\mathbb{P}[s(X, Y) \leq \hat{q}] \geq 1 - \alpha \implies \mathbb{P}\left[|Y - \hat{f}(X)| \leq u(X)\hat{q}\right] \geq 1 - \alpha.$$

Naturally, we can then form prediction sets using the rule

$$\mathcal{T}(X) = \left[\hat{f}(X) - u(X)\hat{q}, \hat{f}(X) + u(X)\hat{q} \right]. \quad (4)$$

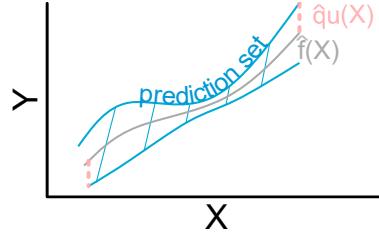


Figure 8: A visualization of the uncertainty scalars algorithm in Eq. (4). We adjust the upper and lower quantiles produced by quantile regression by the constant \hat{q} , picked during the calibration step.

Let's reflect a bit on the nature of these prediction sets. The prediction sets are valid, as we desired. Due to our construction, they are also symmetric about the prediction, $\hat{f}(X)$, although symmetry is not strictly required. Nonetheless, the relationship between the upper endpoint and lower endpoint of the prediction set must be fixed a-priori by the developer. This is not true of quantile regression, which allows the relationship between the upper and lower quantile to adapt to the data. As a consequence, the prediction sets generated by uncertainty scalars are often larger than those from quantile regression, since the intervals extend both above and below \hat{f} by roughly the magnitude of the residual. Finally, uncertainty scalars do not necessarily scale properly with α . For example, the uncertainty scalar $\hat{\sigma}$ estimates the standard deviation of a Laplace approximation to $Y | X$; there is no reason to believe that this quantity would be a multiplicative constant away from the 90% quantile of the residual's magnitude. Nonetheless, uncertainty scalars remain in use because they are easy to deploy and have been commoditized in popular machine learning libraries. In conclusion, quantiles should be used when possible, and uncertainty scalars are primarily for convenience (e.g. with a pre-trained model). See Figure 7 for a PyTorch implementation of this method.

2.4 Conformalizing Bayes

Our final example of conformal prediction will use a Bayesian model. Bayesian predictors, like Bayesian neural networks, are commonly studied in the field of uncertainty quantification, but rely on many unverifiable and/or incorrect assumptions to provide coverage. Nonetheless, we should incorporate any prior information

we have into our prediction sets. We will now show how to create valid prediction sets that are also Bayes optimal among all prediction sets that achieve $1 - \alpha$ coverage. These prediction sets use the posterior predictive density as a conformal score. The Bayes optimality of this procedure was first proven in [8]. Because our algorithm reduces to picking the labels with high posterior predictive density, the PyTorch code will look exactly the same as in Figure 2. The only difference is interpretation, since the softmax now represents an approximation of a continuous distribution rather than a categorical one.

Let us first describe what a Bayesian would do, given a Bayesian model $\hat{f}(X)_Y$, which estimates the value of the posterior distribution at label Y . If one believed all of the necessary assumptions—mainly, a correctly specified model and asymptotically large n —the following would be the optimal prediction set:

$$S(X) = \left\{ y : \hat{f}(X)_y > t \right\}, \text{ where } t \text{ is chosen so } \int_{y \in S(X)} \hat{f}(X)_y dy = 1 - \alpha.$$

However, because we cannot make assumptions on the model and data, we can only consider $\hat{f}(X)$ to be a heuristic notion of uncertainty.

Following our now-familiar checklist, we can define a conformal score,

$$s(X, Y) = -\hat{f}(X)_Y,$$

which is high when the model is uncertain and otherwise low. After computing \hat{q} over the calibration data, we can then construct prediction sets:

$$\mathcal{T}(X) = \left\{ y : \hat{f}(X)_y > -\hat{q} \right\}. \quad (5)$$

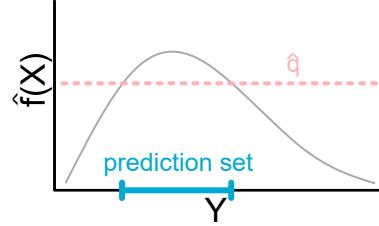


Figure 9: A visualization of the conformalized Bayes algorithm in Eq. (5). The prediction set is a superlevel set of the posterior predictive density.

This set is valid because we chose the threshold \hat{q} via conformal prediction. Furthermore, when certain technical assumptions are satisfied, it has the best Bayes risk among all prediction sets with $1 - \alpha$ coverage. To be more precise, under the assumptions in [8], $\mathcal{T}(X)$ has the smallest average size of any conformal procedure with $1 - \alpha$ coverage, where the average is taken over the data *and* the parameters. This result should not be a surprise to those familiar with decision theory, as the argument we are making feels similar to that of the Neyman-Pearson lemma. This concludes the final example.

Remarks

As our examples have shown, conformal prediction is a simple and pragmatic technique with many use cases. It is also easy to implement and computationally trivial. Additionally, the above four examples serve as roadmaps to the user for designing score functions with various notions of optimality, including average size, adaptiveness, and Bayes risk. Lastly, we note that there is a generalization of what we have presented here that doesn't require data splitting, but is instead more computationally intensive, sometimes called full conformal prediction [1]. In summary, for problems with a simple structure, where the notion of coverage makes sense, conformal prediction is very effective and flexible.

However, modern prediction tasks eschew the simple structure of the settings we described above. Machine learning now often focuses on highly complex and structured outputs, like image segmentations, protein

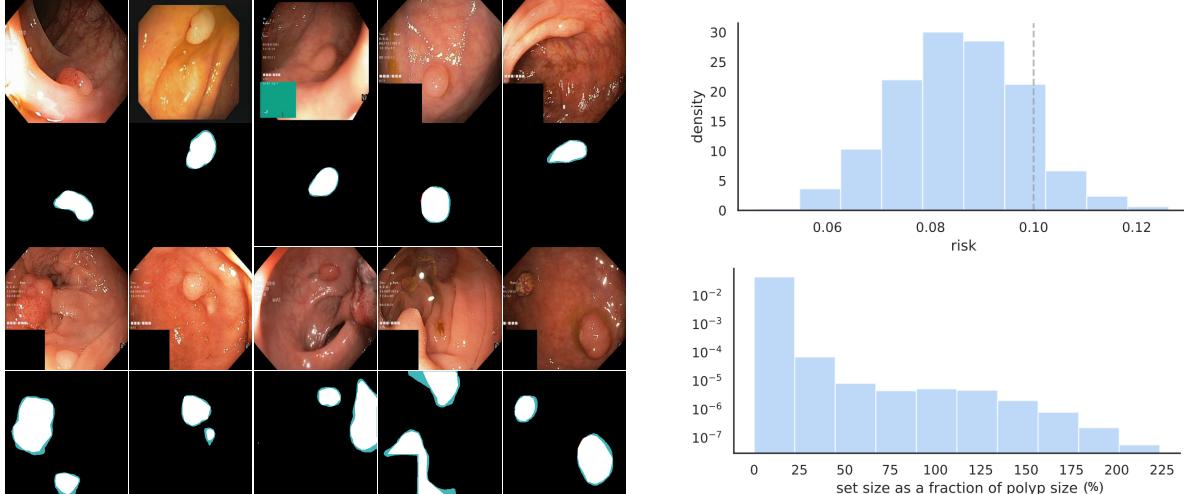


Figure 10: The result of running RCPS on a human gut polyp segmentation task. We set the risk level to $\alpha = 0.1$, the failure rate to $\delta = 0.1$, and used $n = 1000$ calibration points. Input images are shown above RCPS segmentation maps which are guaranteed to contain 90% of each polyp with probability at least $1 - \delta$. White pixels are true positives, blue pixels are false positives, red pixels are false negatives, and black pixels are true negatives. The plots on the right-hand side show the risk and set size over 100 different random calibration-validation splits.

folding, and multilabel classification. For these problems, the notion of coverage does not make sense, and we must generalize conformal prediction in order to provide distribution-free uncertainty quantification.

3 Risk-Controlling Prediction Sets

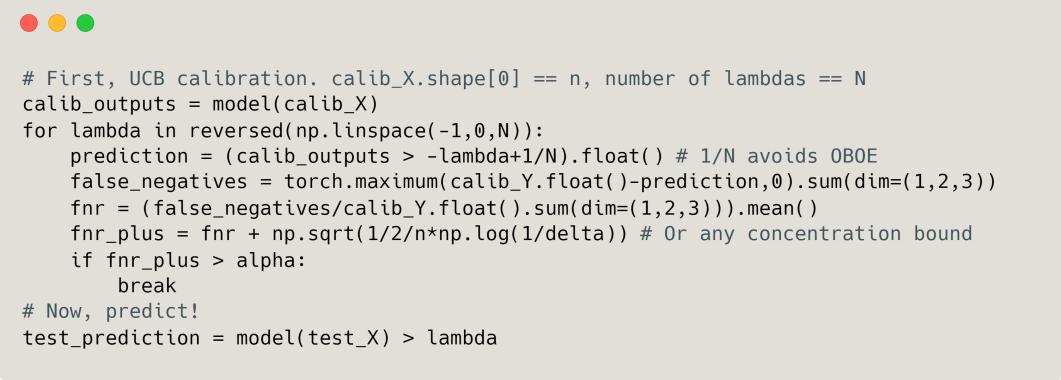
Conformal prediction is an attractive general purpose tool, but it does not apply to prediction tasks where the output is complicated (see Section 3.3 for a deeper analysis). As an example of a prediction task where conformal prediction doesn't apply, take binary image segmentation. In that setting, we are given an image and seek to return a subset of pixels that contain an object of interest; see Figure 10. This section presents a generalization of conformal prediction, *risk-controlling prediction sets* (RCPS), that naturally addresses these problems, leading to a more flexible form of distribution-free uncertainty quantification that can be used in complicated settings. Like conformal prediction, RCPS uses prediction sets to capture uncertainty, but unlike conformal prediction, this new technique can use other notions of statistical error besides the coverage/miscoverage error in (1) that we previously considered. We proceed by first outlining RCPS for an image segmentation task in Section 3.1 and then give the general technique in Section 3.2.

3.1 Motivating Example: Image Segmentation

We begin with an image segmentation task in medical imaging where we seek to detect tumor regions from a colonoscopy image; see the left panel of Figure 10 for an example. Intuitively, our goal is to predict a binary mask with the same dimension as the image that is guaranteed to take on the value '1' for 90% of the tumor pixels, regardless of the quality of the underlying learned segmentation model. Formally, the features are an image $X \in \mathcal{X} = \mathcal{R}^{d_1 \times d_2 \times 3}$ and the response is a subset of pixels $Y \in \mathcal{Y} = \{0, 1\}^{d_1 \times d_2}$.

For this task, we will seek to learn a prediction set $\mathcal{T} : \mathcal{X} \rightarrow \mathcal{Y}$ that certifiably contains most of the tumor pixels. In particular, we will find a prediction set such that the false negative rate is controlling at level 10%:

$$\mathbb{E} \left[1 - \frac{|\mathcal{T}(X) \cap Y|}{|Y|} \right] \leq 10\%, \quad (6)$$



```

# First, UCB calibration. calib_X.shape[0] == n, number of lambdas == N
calib_outputs = model(calib_X)
for lambda in reversed(np.linspace(-1,0,N)):
    prediction = (calib_outputs > -lambda+1/N).float() # 1/N avoids OBOE
    false_negatives = torch.maximum(calib_Y.float()-prediction,0).sum(dim=(1,2,3))
    fnr = (false_negatives/calib_Y.float().sum(dim=(1,2,3))).mean()
    fnr_plus = fnr + np.sqrt(1/2/n*np.log(1/delta)) # Or any concentration bound
    if fnr_plus > alpha:
        break
# Now, predict!
test_prediction = model(test_X) > lambda

```

Figure 11: PyTorch code for image segmentations that control the false negative rate via RCPS.

where $|Y|$ denotes the number of the (binary) entries of Y that are 1. In words, the above is the expected fraction of missed tumor pixels, which we seek to control at level 10%.

Risk-controlling prediction sets are a technique to learn a rule \mathcal{T} that satisfies monotonic error control properties like the one in (6). As before with conformal prediction, we begin with a pre-trained prediction model \hat{f} . In this case, we use a deep network that returns a score for each pixel, $\hat{f} : \mathcal{X} \rightarrow [0, 1]^{d_1 \times d_2}$. Based on this model, we can construct \mathcal{T} as follows:

$$\mathcal{T}(X) = \{(i, j) : \hat{f}_{(i,j)}(X) \geq \lambda\},$$

for a carefully-chosen threshold λ . That is, we take our raw network \hat{f} and select all pixels that exceed a threshold λ . This general approach to forming a prediction is natural, but choosing λ in the right way is essential to guarantee that (6) holds; i.e., to rigorously guarantee that we don't miss too many tumor pixels. This is the heart of RCPS, and in the next section we will give the algorithm for learning λ such that the statistical guarantees hold. Implementing the procedure is straightforward; see the Pytorch code for this example in Figure 11. Code for reproducing these results can be found in [the RCPS GitHub repository](#).

3.2 Instructions for Risk-Controlling Prediction Sets

Next, we describe the general Risk-controlling prediction sets method. We first introduce notation and the RCPS property in Definition 1. Then, we describe the calibration algorithm.

Formalization of error control

Let $(X_i, Y_i)_{i=1, \dots, n}$ be an independent and identically distributed (i.i.d.) set of variables, where the features vectors X_i take values in \mathcal{X} and the response Y_i take values in \mathcal{Y} . The researcher starts with a pre-trained predictive model \hat{f} . The remainder of this paper shows how to subsequently create set-valued predictors from \hat{f} that control a certain statistical error notion, regardless of the quality of the initial model fit or the distribution of the data. Next, let $\mathcal{T} : \mathcal{X} \rightarrow \mathcal{Y}'$ be a set-valued function (a *tolerance region*) that maps a feature vector to a set-valued prediction. This function would typically be constructed from the predictive model, \hat{f} . We further suppose we have a collection of such set-valued predictors indexed by a one-dimensional parameter λ taking values in a closed set $\Lambda \subset \mathbb{R} \cup \{\pm\infty\}$ that are *nested*, meaning that larger values of λ lead to larger sets:

$$\lambda_1 < \lambda_2 \implies \mathcal{T}_{\lambda_1}(x) \subset \mathcal{T}_{\lambda_2}(x). \quad (7)$$

To encode a notion of error, let $L(y, \mathcal{S}) : \mathcal{Y} \times \mathcal{Y}' \rightarrow [0, 1]$ be a *loss function* on prediction sets. For example, we could take $L(y, \mathcal{S}) = \mathbb{1}\{y \notin \mathcal{S}\}$, which is the loss function corresponding to classical tolerance

regions. Another example is the false negative rate used in the image segmentation example (6). We require that the loss function respects the following nesting property:

$$\mathcal{S} \subset \mathcal{S}' \implies L(y, \mathcal{S}) \geq L(y, \mathcal{S}'). \quad (8)$$

That is, larger sets lead to smaller loss. We then define the *risk* of a set-valued predictor \mathcal{T} to be

$$R(\mathcal{T}) = \mathbb{E}[L(Y_{n+1}, \mathcal{T}(X_{n+1}))].$$

Since we will primarily be considering the risk of the tolerance functions from the family $\{\mathcal{T}_\lambda\}_{\lambda \in \Lambda}$, we will use the notational shorthand $R(\lambda)$ to mean $R(\mathcal{T}_\lambda)$. We further assume that there exists an element $\lambda_{\max} \in \Lambda$ such that $R(\lambda_{\max}) = 0$.

With the notation in place, we will seek to control statistical error in the following sense:

Definition 1 (Risk-controlling prediction sets). *Let \mathcal{T} be a random function taking values in the space of functions $\mathcal{X} \rightarrow \mathcal{Y}'$ (e.g., a functional estimator trained on data). We say that \mathcal{T} is a (α, δ) -risk-controlling prediction set if, with probability at least $1 - \delta$, we have $R(\mathcal{T}) \leq \alpha$.*

Above, α and δ are error levels pre-specified by the user. For example, in the image segmentation example we take $\alpha = 0.1$ and $\delta = 0.1$.

The procedure

Recalling Definition 1, our goal is to find a set function whose risk is less than some user-specified threshold α . To do this, we search across the collection of functions $\{\mathcal{T}_\lambda\}_{\lambda \in \mathcal{T}}$ and estimate their risk on the calibration data $(X_1, Y_1), \dots, (X_n, Y_n)$. We then show that by choosing the value of λ in a certain way, we can guarantee that the procedure has risk less than α with high probability.

Our procedure uses a pointwise upper confidence bound (UCB)

$$\hat{R}^+(\lambda) = \frac{1}{n} \sum_{i=1}^n L(Y_i, \mathcal{T}_\lambda(X_i)) + \sqrt{\log(1/\delta)/2n}$$

for the risk function for each λ , which satisfies

$$P\left(R(\lambda) \leq \hat{R}^+(\lambda)\right) \geq 1 - \delta, \quad (9)$$

by Hoeffding's inequality. (Improvements on this simple choice of $\hat{R}^+(\lambda)$ are possible; see [9].)

From here, our procedure is simple: we then choose $\hat{\lambda}$ as the smallest value of λ such that the entire confidence region to the right of λ falls below the target risk level α :

$$\hat{\lambda} \triangleq \inf \left\{ \lambda \in \Lambda : \hat{R}^+(\lambda') < \alpha, \forall \lambda' \geq \lambda \right\}. \quad (10)$$

See Figure 12 for a visualization. This procedure is simple to carry out in software—we give the code implementing this in Python for the image segmentation example in Figure 11.

This choice of λ results in a set-valued predictor that controls the risk with high probability:

Theorem 2 (Validity of UCB calibration). *Let $(X_i, Y_i)_{i=1, \dots, n}$ be an i.i.d. sample, let $L(\cdot, \cdot)$ be a loss satisfying the monotonicity condition in (8), and let $\{\mathcal{T}_\lambda\}_{\lambda \in \Lambda}$ be a collection of set predictors satisfying the nesting property in (7). Suppose (9) holds pointwise for each λ , and that $R(\lambda)$ is continuous. Then for $\hat{\lambda}$ chosen as in (10),*

$$P(R(\mathcal{T}_{\hat{\lambda}}) \leq \alpha) \geq 1 - \delta.$$

That is, $\mathcal{T}_{\hat{\lambda}}$ is an (α, δ) -RCPS.

See [9] for a proof and extensions to multi-label classification, protein folding, and classification with different losses per-class.

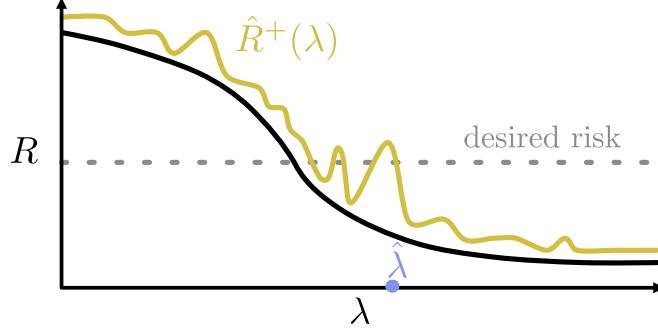


Figure 12: This is a visual explanation of the UCB calibration procedure used in RCPS to choose $\hat{\lambda}$. When λ is large, sets are large; we choose the largest λ that crosses the desired risk.

3.3 Remarks

We next give some more advanced remarks about RCPS. The inability of conformal prediction to handle tasks like image segmentation arises because it controls an inflexible and conservative notion of error. Expanding out the definition of miscoverage from (1) gives

$$\mathbb{P}[Y_{n+1} \notin \mathcal{T}(X_{n+1})] \leq \alpha \iff \mathbb{E}\left[\underbrace{\mathbb{1}_{\{Y_{n+1} \notin \mathcal{T}(X_{n+1})\}}}_{\text{binary loss function}}\right] \leq \alpha,$$

which can be interpreted as the expected value of a binary loss—i.e., a binary *risk*. That is, conformal prediction is essentially RCPS with a binary loss function. This has several issues. First, not all problems have a naturally binary loss. Different mistakes may cost different amounts. For example, missing the class ‘cancer’ may be worse than missing the class ‘concussion’, but conformal would assign both mistakes the same loss. Second, the binary loss can be too conservative in high dimensions. If Y has one million dimensions (such as in our image example) ensuring $Y \in \mathcal{T}(X)$ often requires enlarging $\mathcal{T}(X)$ to the point of uselessness. This is because coverage needs to hold simultaneously over all 1M dimensions, and one small error along one dimension spoils the entire set; thus the set must be very large to avoid most such cases.

As a final note, the reader should notice that unlike the standard conformal setting, an RCPS can be in any set-valued space. It does not need to be in the power set of \mathcal{Y} . In the segmentation problem for example, the predictions themselves are set-valued, and therefore indistinguishable from prediction sets—that is, the RCPS live in the same space as the predictions. This general setting is often desirable for modern prediction problems, where the high-dimensional nature of Y makes $2^{\mathcal{Y}}$ difficult to reason about or even represent. For example, designing and communicating a probabilistic guarantee about a single segmentation mask is straightforward, while one about a set of many possible segmentation masks is confusing and generally not desired.

4 Evaluating Prediction Sets

We have spent the last three sections learning how to form valid prediction sets satisfying rigorous statistical guarantees. Now we will briefly discuss how to evaluate them. For simplicity, we will focus on conformal prediction, but the below discussion translates precisely to RCPS, usually just by replacing the word ‘coverage’ with ‘risk’.

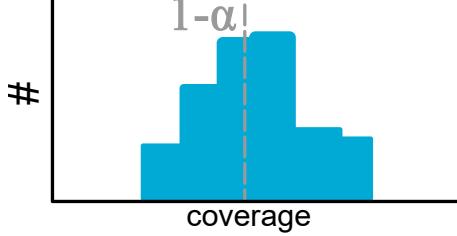
4.1 Basic Checks

Coverage. As an obvious first step, the user will want to assess whether the conformal procedure has the correct coverage. This can be accomplished by running the procedure R times, randomly splitting the data

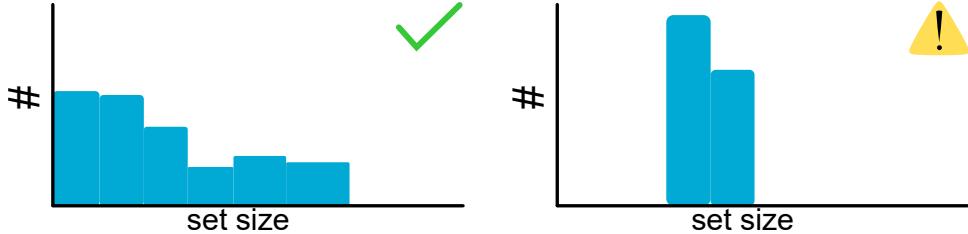
into a calibration and validation datasets, and then calculating the empirical average coverage,

$$C_j = \frac{1}{n'} \sum_{i=1}^{n'} \mathbb{1} \left\{ Y_i^{(\text{val})} \in \mathcal{T} \left(X_i^{(\text{val})} \right) \right\}, \text{ for } j = 1, \dots, R,$$

where n' is the size of the validation set. A histogram of these R coverage values should look be centered at $1 - \alpha$; see below. If properly implemented, conformal prediction is guaranteed to satisfy this property.



Set size. Having confirmed that the coverage guarantee holds empirically, one can plot histograms of set sizes. This histogram helps us in two ways. Firstly, a large average set size indicates the conformal procedure is not very precise, indicating a possible problem with the score or underlying model. Secondly, the spread of the set sizes shows whether the prediction sets properly adapt to the difficulty of examples. A wider spread is generally desirable, since it means that the procedure is effectively distinguishing between easy and hard inputs.



It can be tempting to stop evaluations after plotting the coverage and set size, but certain important questions remain unanswered. A good spread of set sizes does not necessarily indicate that the sets adapt properly to the difficulty of X ; it can be worth plotting the score against the residual $|Y - \hat{f}(X)|$ to verify that hard examples have large scores. But even if the score correlates well with difficulty, prediction sets can still fail for certain types of inputs. We discuss how to evaluate this next.

4.2 Evaluating Adaptiveness

A key design consideration for conformal prediction is *adaptiveness*: we want the procedure to return larger sets for harder inputs and smaller sets for easier inputs. While most reasonable conformal procedures will satisfy this to some extent, we now discuss precise metrics for adaptiveness that allow the user to check a conformal procedure and to compare multiple alternative conformal procedures. Adaptiveness is typically formalized by asking for the *conditional coverage* [10] property:

$$\mathbb{P}[Y \in \mathcal{T}(X) | X] \geq 1 - \alpha. \quad (11)$$

That is, for every value of the input X , we seek to return prediction sets with $1 - \alpha$ coverage. This is a stronger property than the *marginal coverage* property in (1) that conformal prediction is guaranteed to achieve. In other words, conformal procedures are not guaranteed to satisfy (11), so we must check how close our procedure comes to approximating it.

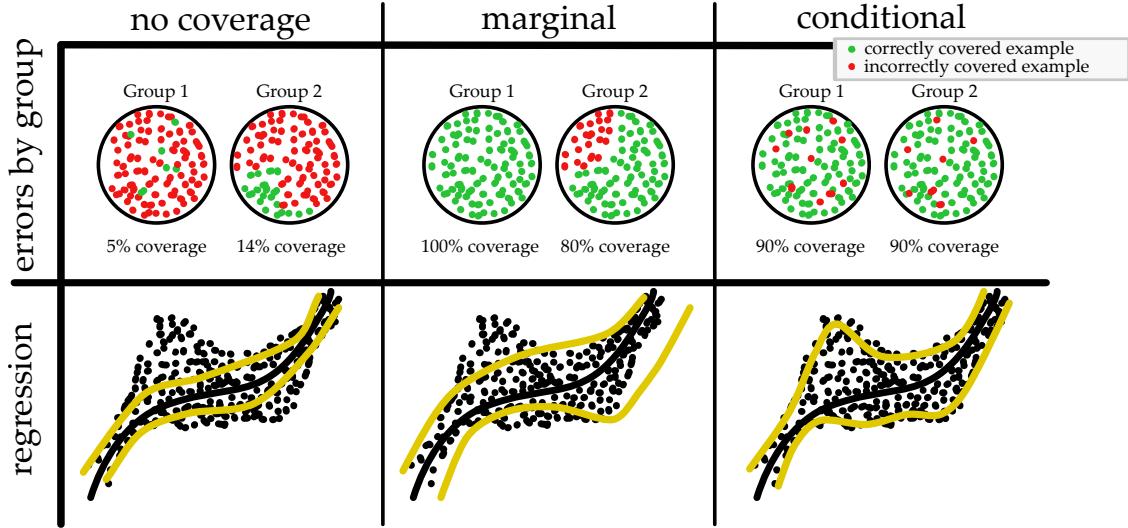


Figure 13: This illustration shows prediction sets that have either no coverage, marginal coverage, or conditional coverage (at a level of 90%). In the marginal case, all of the errors happen in the same groups and regions in X -space. Conditional coverage disallows this behavior, and errors are evenly distributed.

The difference between marginal and conditional coverage is subtle but of great practical importance, so we will spend some time think about the differences here. Imagine there are two groups of people, group A and group B, with frequencies 90% and 10%. The prediction sets always cover Y among people in group A and never cover Y when the person comes from group B. Then the prediction sets have 90% coverage, but not conditional coverage. Conditional coverage would imply that the prediction sets cover Y at least 90% of the time in both groups. This is necessary, but not sufficient; conditional coverage is a very strong property that states the probability of the prediction set needs to be $\geq 90\%$ for a particular person. In other words, for any subset of the population, the coverage should be $\geq 90\%$. See also Figure 13 for a visualization of the difference between conditional and marginal coverage.

Feature-stratified coverage metric. As a first metric for conditional coverage, we will formalize the example we gave earlier, where coverage is unequal over some groups. The reader can think of these groups as discrete categories, like race, or as a discretization of continuous features, like age ranges. Formally, suppose we have features $X_{i,1}$ that take values in $\{1, \dots, G\}$ for some G . (Here, $i = 1, \dots, n$ indexes the example and 1 indexes the feature.) Let $\mathcal{I}_g \subset \{1, \dots, n\}$ be the set of observations such that $X_{i,1} = g$ for $g = 1, \dots, G$. Since conditional coverage implies that the procedure has the same coverage for all values of X_1 , we use the following measure:

$$\text{FSC metric : } \min_{g \in \{1, \dots, G\}} \frac{1}{|\mathcal{I}_g|} \sum_{i \in \mathcal{I}_g} \mathbb{I}_{\{Y_i \in \mathcal{T}(X_i)\}}$$

In words, this is the observed coverage for all units for which the discrete feature takes value g . If conditional coverage were achieved, this would be $1 - \alpha$, and values farther below $1 - \alpha$ indicate a greater violation of conditional coverage. Note that this metric can also be used with a continuous feature by binning the features into a finite number of categories.

Set-stratified coverage metric. We next consider a more general-purpose metric for how close a conformal procedure comes to satisfying (11), introduced in [4]. First, we discretize the possible cardinalities of $\mathcal{T}(X)$, into G bins, B_1, \dots, B_G . For example, in classification we might divide the observations into units into three groups, depending on whether $\mathcal{T}(X)$ has one element, two elements, or more than two elements.

Let $\mathcal{I}_g \subset \{1, \dots, n\}$ be the set of observations falling in bin g for $g = 1, \dots, G$. Then we consider the following

$$\text{SSC metric : } \min_{g \in \{1, \dots, G\}} \frac{1}{|\mathcal{I}_g|} \sum_{i \in \mathcal{I}_g} \mathbb{I}_{\{Y_i \in \mathcal{T}(X_i)\}}$$

In words, this is the observed coverage for all units for which the set size $|\mathcal{T}(X)|$ falls into bin g . As before, if conditional coverage were achieved, this would be $1 - \alpha$, and values farther below $1 - \alpha$ indicate a greater violation of conditional coverage. Note that this is the same expression as for the FSC metric, except that the definition of \mathcal{I}_g has changed. Unlike the FSC metric, the user does not have to define an important set of discrete features a-priori—it is a general metric that can apply to any example.

See [11] and [12] for additional metrics of conditional coverage.

References

- [1] V. Vovk, A. Gammerman, and G. Shafer, *Algorithmic Learning in a Random World*. Springer, 2005.
- [2] H. Papadopoulos, K. Proedrou, V. Vovk, and A. Gammerman, “Inductive confidence machines for regression,” in *Machine Learning: European Conference on Machine Learning*, 2002, pp. 345–356.
- [3] J. Lei and L. Wasserman, “Distribution-free prediction bands for non-parametric regression,” *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, vol. 76, no. 1, pp. 71–96, 2014.
- [4] A. N. Angelopoulos, S. Bates, J. Malik, and M. I. Jordan, “Uncertainty sets for image classifiers using conformal prediction,” in *International Conference on Learning Representations*, 2021.
- [5] M. Sadinle, J. Lei, and L. Wasserman, “Least ambiguous set-valued classifiers with bounded error levels,” *Journal of the American Statistical Association*, vol. 114, pp. 223–234, 2019.
- [6] Y. Romano, M. Sesia, and E. J. Candès, “Classification with valid and adaptive coverage,” *arXiv:2006.02544*, 2020.
- [7] Y. Romano, E. Patterson, and E. Candès, “Conformalized quantile regression,” in *Advances in Neural Information Processing Systems*, vol. 32, 2019, pp. 3543–3553.
- [8] P. Hoff, “Bayes-optimal prediction with frequentist coverage control,” *arXiv:2105.14045*, 2021.
- [9] S. Bates, A. Angelopoulos, L. Lei, J. Malik, and M. I. Jordan, “Distribution-free, risk-controlling prediction sets,” *arXiv:2101.02703*, 2021.
- [10] V. Vovk, “Conditional validity of inductive conformal predictors,” in *Proceedings of the Asian Conference on Machine Learning*, vol. 25, 2012, pp. 475–490.
- [11] M. Cauchois, S. Gupta, and J. Duchi, “Knowing what you know: Valid and validated confidence sets in multiclass and multilabel prediction,” *arXiv:2004.10181*, 2020.
- [12] S. Feldman, S. Bates, and Y. Romano, “Improving conditional coverage via orthogonal quantile regression,” *arXiv:2106.00394*, 2021.
- [13] R. J. Tibshirani, R. Foygel Barber, E. Candes, and A. Ramdas, “Conformal prediction under covariate shift,” in *Advances in Neural Information Processing Systems 32*, 2019, pp. 2530–2540.
- [14] J. Lei, M. G’Sell, A. Rinaldo, R. J. Tibshirani, and L. Wasserman, “Distribution-free predictive inference for regression,” *Journal of the American Statistical Association*, vol. 113, no. 523, pp. 1094–1111, 2018.

A Theorem and Proof: Coverage Property of Conformal Prediction

This is a standard proof of validity for split-conformal prediction first appearing in [2], but we reproduce it here for completeness. Let us begin with the lower bound.

Theorem A.1 (Conformal calibration coverage guarantee). *Suppose $(X_i, Y_i)_{i=1,\dots,n+1}$ are i.i.d. Then define \hat{q} as*

$$\hat{q} = \inf \left\{ q : \frac{|\{i : s(X_i, Y_i) \leq q\}|}{n} \geq \frac{\lceil (n+1)(1-\alpha) \rceil}{n} \right\}.$$

and define the resulting prediction sets as

$$\mathcal{T}(X) = \{y : s(X, y) \leq \hat{q}\}.$$

By choosing \hat{q} this way, we can generate a prediction set satisfying the following coverage guarantee:

$$P(Y_{n+1} \in \mathcal{T}(X_{n+1})) \geq 1 - \alpha.$$

This is the same coverage property as (1) in the introduction, but written more formally. As a technical remark, the theorem also holds if the observations to satisfy the weaker condition of exchangeability; see [1]. Below, we prove the lower bound.

Proof of Theorem A.1. Let $s_i = s(X_i, Y_i)$ for $i = 1, \dots, n+1$. To avoid handling ties, we consider the case where the s_i are distinct with probability 1. See [13] for a proof in the general case.

Without loss of generality we assume the calibration scores are sorted so that $s_1 < \dots < s_n$. In this case, we have that $\hat{q} = s_{\lceil (n+1)(1-\alpha) \rceil}$. Note that

$$\{Y_{n+1} \in \mathcal{T}(X_n + 1)\} = \{s_{n+1} \leq \hat{q}\}.$$

Putting this together,

$$\{Y_{n+1} \in \mathcal{T}(X_n + 1)\} = \{s_{n+1} > s_{\lceil (n+1)(1-\alpha) \rceil}\}.$$

Now comes the crucial insight. By symmetry of the variables $(X_1, Y_1), \dots, (X_{n+1}, Y_{n+1})$, we have

$$P(s_{n+1} \leq s_k) = \frac{k}{n+1}$$

for any integer k . In words, s_{n+1} is equally likely to fall in anywhere between the calibration points s_1, \dots, s_n . Note that above, the randomness is over all variables s_1, \dots, s_{n+1} .

From here, we conclude

$$P(s_{n+1} \leq s_{\lceil (n+1)(1-\alpha) \rceil}) = \frac{\lceil (n+1)(1-\alpha) \rceil}{(n+1)} \geq 1 - \alpha,$$

which implies the desired result. □

Now we will discuss the upper bound. Technically, the upper bound only holds when the distribution of the conformal score is continuous, avoiding ties. In practice, however, this condition is not important, because the user can always add a vanishing amount of random noise to the score. We will state the theorem now, and defer its proof.

Theorem A.2 (Conformal calibration upper bound). *Additionally, if the scores s_1, \dots, s_n have a continuous joint distribution, then*

$$P(Y_{n+1} \in \mathcal{T}(X_{n+1}, U_{n+1}, \hat{q})) \leq 1 - \alpha + \frac{1}{n+1}.$$

Proof. See Theorem 2.2 of Lei, G'Sell, Rinaldo, Tibshirani, and Wasserman [14]. □