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

2024, First Edition

ISBN 9798377509356 (PAPERBACK)

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

commit id: 1a56b53

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 4.1.

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. There's no doubt about it.

(b) These are not actually ghosts. Don't let these dogs bamboozle you.

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.

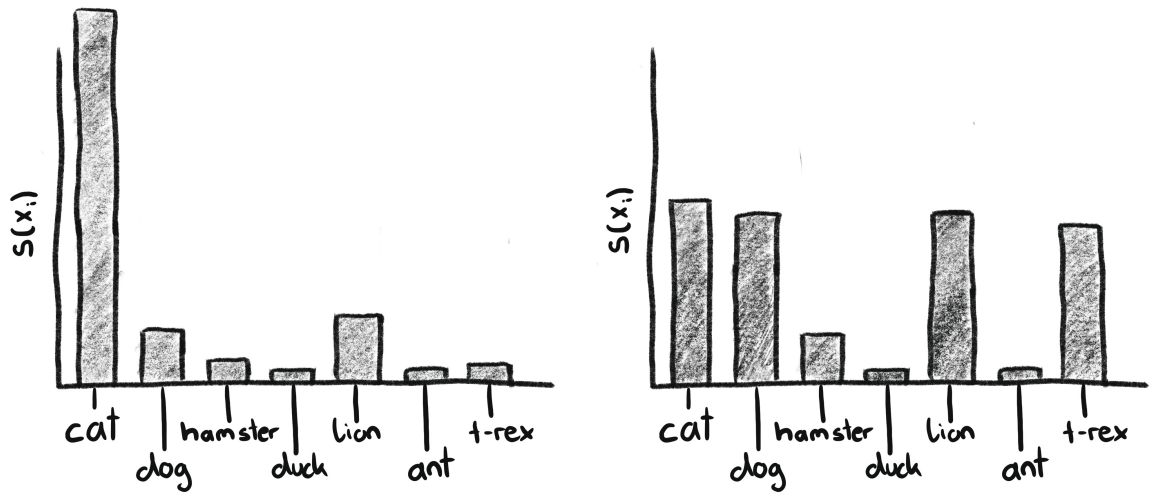
At first glance, aren't we done when the model outputs probabilities and we use them to get an idea of uncertainty? Unfortunately, no. Let's explore why.

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]

[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



(a) Easy (b) Difficult

Figure 4.2: Classification scores by class.

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.

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.

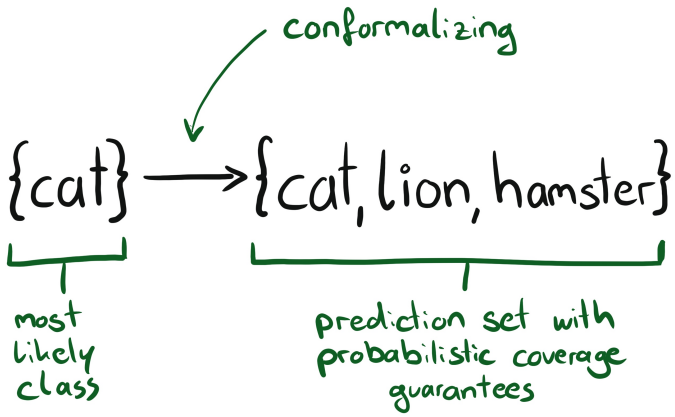
> **Naive Approach**
>
> The naive approach is to take at face value the uncertainty scores that the model spits out - confidence intervals, variance, Bayesian posteriors, multi-class probabilities. The problem: you can't expect these outcomes to be well calibrated.

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.

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:

[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.



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

> ⚠ 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 95%
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.

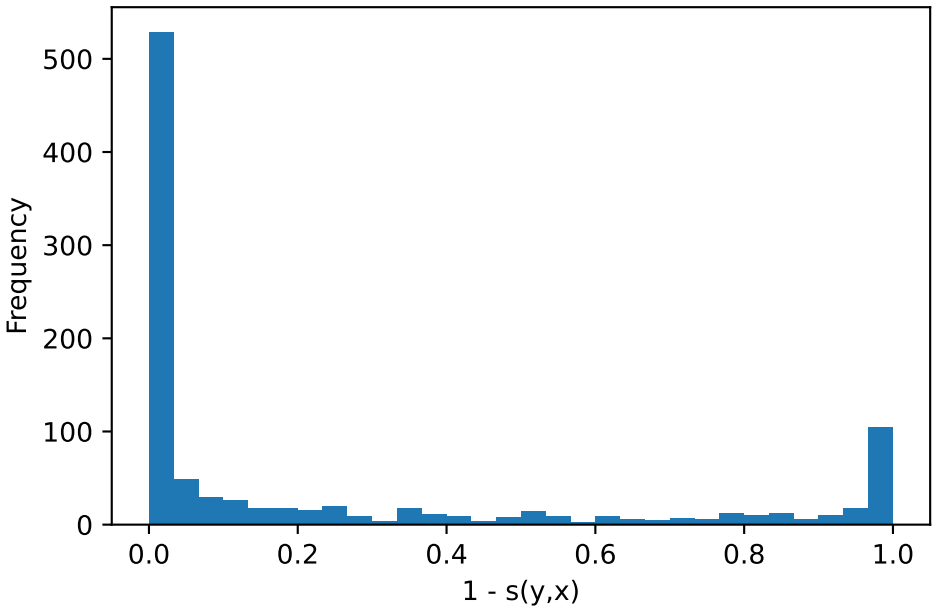
```
# Setting the alpha so that we get 95% prediction sets
alpha = 0.05
# define quantile
q_level = np.ceil((n+1)*(1-alpha))/n
qhat = np.quantile(scores, q_level, method='higher')
```

The quantile level (based on) 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 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.

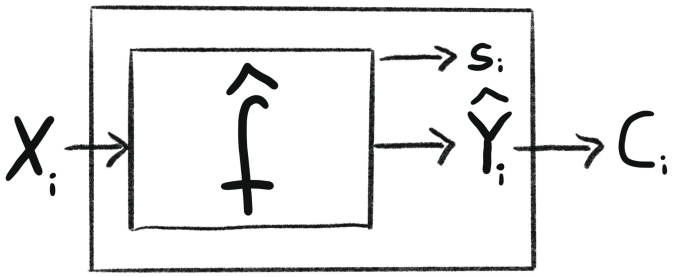


Figure 5.1: Conformal prediction wraps the model

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
Name: count, 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
Name: count, 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.