

2

Key Concepts of Interpretability

This book covers many model interpretation methods: some produce metrics, other visuals, and some both; some depict your model broadly and others granularly. In this chapter, we will learn about two methods, feature importance and decision regions, as well as the taxonomies used to describe these methods. We will also detail what elements hinder machine learning interpretability as a primer to what lies ahead.

The following are the main topics we are going to cover in this chapter:

- Learning about interpretation method types and scopes
- Appreciating what hinders machine learning interpretability

Technical requirements

Although we began the book with a "toy example," we will be leveraging real datasets throughout this book to be used in specific interpretation use cases. These come from many different sources and are often used only once.

To avoid that, readers spend a lot of time downloading, loading, and preparing datasets for single examples; there's a library called `mldatasets` that takes care of most of this. Instructions on how to install this library are located in the *preface*. In addition to `mldatasets`, this chapter's examples also use the `pandas`, `numpy`, `statsmodel`, `sklearn`, and `matplotlib` libraries. The code for this chapter is located here: <https://github.com/PacktPublishing/Interpretable-Machine-Learning-with-Python/tree/master/Chapter02>.

The mission

Imagine you are an analyst for a national health ministry, and there's a **Cardiovascular Diseases (CVDs)** epidemic. The minister has made it a priority to reverse the growth and reduce the case load to a 20-year low. To this end, a task force has been created to find clues in the data to ascertain the following:

1. What risk factors can be addressed.
2. If future cases can be predicted, interpret predictions on a case-by-case basis.

You are part of this task force!

Details about CVD

Before we dive into the data, we must gather some important details about CVD in order to do the following:

- Understand the problem's context and relevance.
- Extract domain knowledge information that can inform our data analysis and model interpretation.
- Relate an expert-informed background to a dataset's features.

CVDs are a group of disorders, the most common of which is coronary heart disease (also known as *Ischaemic Heart Disease*). According to the World Health Organization, CVD is the leading cause of death globally, killing close to 18 million people annually. Coronary heart disease and strokes (which are, for the most part, a byproduct of CVD) are the most significant contributors to that. It is estimated that 80% of CVD is made up of modifiable risk factors. In other words, some of the preventable factors that cause CVD include the following:

- Poor diet
- Smoking and alcohol consumption habits
- Obesity
- Lack of physical activity
- Poor sleep

Also, many of the risk factors are non-modifiable, and therefore known to be unavoidable, including the following:

- Genetic predisposition
- Old age
- Male (varies with age)

We won't go into more domain-specific details about CVD because it is not required to make sense of the example. However, *it can't be stressed enough how central domain knowledge is to model interpretation*. So, if this example was your job and many lives depended on your analysis, it would be advisable to read the latest scientific research on the subject or consult with domain experts to inform your interpretations.

The approach

Logistic regression is one common way to rank risk factors in medical use cases. Unlike linear regression, it doesn't try to predict a continuous value for each of your observations, but it predicts a probability score that an observation belongs to a particular class. In this case, what we are trying to predict is, given x data for each patient, what is the y probability, from 0 to 1, that they have cardiovascular disease?

Preparations

You will find the code for this example here: <https://github.com/PacktPublishing/Interpretable-Machine-Learning-with-Python/blob/master/Chapter02/CVD.ipynb>.

Loading the libraries

To run this example, you need to install the following libraries:

- `mldatasets` to load the dataset
- `pandas` and `numpy` to manipulate it
- `statsmodels` to fit the logistic regression model
- `sklearn` (scikit-learn) to split the data
- `matplotlib` to visualize the interpretations

You should load all of them first:

```
Import math
import mldatasets
import pandas as pd
import numpy as np
import statsmodels.api as sm
from sklearn.model_selection import train_test_split
import matplotlib.pyplot as plt
```

Understanding and preparing the data

The data to be used in this example should then be loaded into a DataFrame we call `cvd_df`:

```
cvd_df = mldatasets.load("cardiovascular-disease")
```

From this, you should be getting 70,000 records and 12 columns. We can take a peek at what was loaded with `info()`:

```
cvd_df.info()
```

The preceding command will output the names of each column with its type and how many non-null records it contains:

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 70000 entries, 0 to 69999
Data columns (total 12 columns):
age                70000 non-null int64
gender            70000 non-null int64
```

height	70000	non-null	int64
weight	70000	non-null	float64
ap_hi	70000	non-null	int64
ap_lo	70000	non-null	int64
cholesterol	70000	non-null	int64
gluc	70000	non-null	int64
smoke	70000	non-null	int64
alco	70000	non-null	int64
active	70000	non-null	int64
cardio	70000	non-null	int64
dtypes: float64(1), int64(11)			
memory usage: 6.4 MB			

The data dictionary

To understand what was loaded, the following is the data dictionary, as described in the source:

- age: Of the patient in days (Objective Feature)
- height: In centimeters (Objective Feature)
- weight: In kg (Objective Feature)
- gender: A binary where 1: female, 2: male (Objective Feature)
- ap_hi: Systolic blood pressure, which is the arterial pressure exerted when blood is ejected during ventricular contraction. Normal value: < 120 mmHg (Examination Feature)
- ap_lo: Diastolic blood pressure, which is the arterial pressure in between heartbeats. Normal value: < 80 mmHg (Examination Feature)
- cholesterol: An ordinal where 1: normal, 2: above normal, 3: well above normal (Examination Feature)
- gluc: An ordinal where 1: normal, 2: above normal, 3: well above normal (Examination Feature)
- smoke: A binary where 0: non-smoker, 1: smoker (Subjective Feature)
- alco: A binary where 0: non-drinker, 1: drinker (Subjective Feature)
- active: A binary where 0: non-active, 1: active (Subjective Feature)
- cardio: A binary where 0: no CVD, 1: has CVD (Target Feature)

Data preparation

For the sake of interpretability and model performance, there are several data preparation tasks that we can take care of, but the one that stands out right now is age. Age is not something we usually measure in days. In fact, for health-related predictions like this one, we might even want to bucket them into **age groups** since people tend to age differently. For now, we will convert all ages into years:

```
cvd_df['age'] = cvd_df['age'] / 365.24
```

The result is a more understandable column because we expect age values to be between 0 and 120. We took existing data and transformed it. This is an example of **feature engineering**, which is when you use domain knowledge of your data to create features that better represent your problem, thereby improving your models. We will discuss this further in *Chapter 10, Feature Selection and Engineering for Interpretability*, and *Chapter 12, Monotonic Constraints and Model Tuning for Interpretability*. There's value in performing feature engineering simply to make model outcomes more *interpretable* as long as this doesn't hurt model performance. As regards the age column, it can't hurt it because we haven't degraded the data. This is because you still have the decimal points for the years that represent the days.

Now we are going to take a peak at what the summary statistics are for each one of our features using the `describe()` method:

```
cvd_df.describe().transpose()
```

Figure 2.1 shows the summary statistics outputted by the preceding code. In *Figure 2.1*, age is looking good because it ranges between 29 and 65 years, which is not out of the ordinary, but there are some anomalous outliers for `ap_hi` and `ap_lo`. Blood pressure can't be negative, and the highest ever recorded was 370. These records will have to be dropped because they could lead to poor model performance and interpretability:

	count	mean	std	min	25%	50%	75%	max
age	70000.0	53.304309	6.755152	29.564122	48.36272	53.945351	58.391742	64.924433
gender	70000.0	1.349571	0.476838	1.000000	1.00000	1.000000	2.000000	2.000000
height	70000.0	164.359229	8.210126	55.000000	159.00000	165.000000	170.000000	250.000000
weight	70000.0	74.205690	14.395757	10.000000	65.00000	72.000000	82.000000	200.000000
ap_hi	70000.0	128.817286	154.011419	-150.000000	120.00000	120.000000	140.000000	16020.000000
ap_lo	70000.0	96.630414	188.472530	-70.000000	80.00000	80.000000	90.000000	11000.000000
cholesterol	70000.0	1.366871	0.680250	1.000000	1.00000	1.000000	2.000000	3.000000
gluc	70000.0	1.226457	0.572270	1.000000	1.00000	1.000000	1.000000	3.000000
smoke	70000.0	0.088129	0.283484	0.000000	0.00000	0.000000	0.000000	1.000000
alco	70000.0	0.053771	0.225568	0.000000	0.00000	0.000000	0.000000	1.000000
active	70000.0	0.803729	0.397179	0.000000	1.00000	1.000000	1.000000	1.000000
cardio	70000.0	0.499700	0.500003	0.000000	0.00000	0.000000	1.000000	1.000000

Figure 2.1 – Summary statistics for the dataset

For good measure, we ought to make sure that `ap_hi` is always higher than `ap_lo`, so any record with that discrepancy should also be dropped:

```

cvd_df = cvd_df[(cvd_df['ap_lo'] <= 370) &\
                 (cvd_df['ap_lo'] > 0)].reset_
index(drop=True)
cvd_df = cvd_df[(cvd_df['ap_hi'] <= 370) &\
                 (cvd_df['ap_hi'] > 0)].reset_
index(drop=True)
cvd_df = cvd_df[cvd_df['ap_hi'] >=\
                cvd_df['ap_lo']].reset_
index(drop=True)

```

Now, in order to fit a logistic regression model, we must put all objective, examination, and subjective features together as X and the target feature alone as y . After this, you split the X and y into training and test datasets, but make sure to include `random_state` for reproducibility:

```
y = cvd_df['cardio']
X = cvd_df.drop(['cardio'], axis=1).copy()
X_train, X_test, y_train, y_test = \
    train_test_split(X, y, test_size=0.15, random_
    state=9)
```

Learning about interpretation method types and scopes

Now that we have prepared our data and split it into training/test datasets, we can fit the model using the training data and print a summary of the results:

```
log_model = sm.Logit(y_train, sm.add_constant(X_train))
log_result = log_model.fit()
print(log_result.summary2())
```

Printing `summary2` on the fitted model produces the following output:

```
Optimization terminated successfully.
      Current function value: 0.561557
      Iterations 6

Results: Logit
=====
===
Model:                Logit                Pseudo R-squared: 0.190
Dependent Variable:   cardio                AIC:
65618.3485
Date:                2020-06-10 09:10       BIC:
65726.0502
No. Observations:    58404                Log-Likelihood:   -32797.
Df Model:            11                  LL-Null:          -40481.
Df Residuals:        58392                LLR p-value:       0.0000
Converged:           1.0000                Scale:           1.0000
No. Iterations:      6.0000
```


	Coef.	Std.Err.	z	P> z	[0.025
					0.975]
const	-11.1730	0.2504	-44.6182	0.0000	-11.6638
-10.6822					
age	0.0510	0.0015	34.7971	0.0000	0.0482
0.0539					
gender	-0.0227	0.0238	-0.9568	0.3387	-0.0693
0.0238					
height	-0.0036	0.0014	-2.6028	0.0092	-0.0063
-0.0009					
weight	0.0111	0.0007	14.8567	0.0000	0.0096
0.0125					
ap_hi	0.0561	0.0010	56.2824	0.0000	0.0541
0.0580					
ap_lo	0.0105	0.0016	6.7670	0.0000	0.0075
0.0136					
cholesterol	0.4931	0.0169	29.1612	0.0000	0.4600
0.5262					
gluc	-0.1155	0.0192	-6.0138	0.0000	-0.1532
-0.0779					
smoke	-0.1306	0.0376	-3.4717	0.0005	-0.2043
-0.0569					
alco	-0.2050	0.0457	-4.4907	0.0000	-0.2945
-0.1155					
active	-0.2151	0.0237	-9.0574	0.0000	-0.2616
-0.1685					
=====					
===					

The preceding summary helps us to understand which X features contributed the most to the y CVD diagnosis using the model coefficients (labeled `Coef.` in the table). Much like with linear regression, they are like a weight applied to every predictor. However, the linear combination exponent is a **logistic function**. This makes the interpretation more difficult. We explain this function further in *Chapter 3, Interpretation Challenges*.

You can only tell by looking at it that the features with the absolute highest values are `cholesterol` and `active`, but it's not very intuitive in terms of what this means. A more interpretable way of looking at these values is revealed once you calculate the exponential of these coefficients:

```
np.exp(log_result.params).sort_values(ascending=False)
```

The preceding code outputs the following:

<code>cholesterol</code>	1.637374
<code>ap_hi</code>	1.057676
<code>age</code>	1.052357
<code>weight</code>	1.011129
<code>ap_lo</code>	1.010573
<code>height</code>	0.996389
<code>gender</code>	0.977519
<code>gluc</code>	0.890913
<code>smoke</code>	0.877576
<code>alco</code>	0.814627
<code>active</code>	0.806471
<code>const</code>	0.000014
<code>dtype:</code>	<code>float64</code>

Why the exponential? The coefficients are the **log odds**, which are the logarithms of the *odds*. Also, *odds* are the probability of a positive case over the probability of a negative case, where the **positive case** is the phenomenon we are trying to predict. It doesn't necessarily indicate what is favored by anyone. For instance, if we are trying to predict the odds of rain today, the positive case would be that it rained, regardless of whether you predicted rain or not. Odds are often expressed as a ratio. The news could say the probability of rain today is 60% or say the odds of rain are 3:2 or $3/2 = 1.5$. In log odds form, this would be 0.176, which is the logarithm of 1.5. They are basically the same thing, but expressed differently. An exponential function is the inverse of a logarithm, so it can take any *log odds* and return the *odds*.

Back to our CVD case. Now that we have the odds, we can interpret what it means. For example, what do the odds mean in the case of `cholesterol`? It means that the odds of CVD increase by a factor of 1.64 for each additional unit of `cholesterol`, provided every other feature stays unchanged. Being able to explain the impact of a feature on the model in such tangible terms is one of the advantages of an *intrinsically interpretable* model such as logistic regression.

Although the *odds* provide us with useful information, they don't tell us what matters the most and, therefore, by themselves, cannot be used to measure feature importance. But how could that be? If something has higher odds, then it must matter more, right? Well, for starters, they all have different scales, so that makes a huge difference. This is because if you are to measure the odds of how much something increases, you have to know by how much it typically increases because that provides context. For example, we could say that the odds of a specific species of butterfly living one day more are 0.66 after their first eggs hatch. This statement is meaningless to you unless you know the lifespan and reproductive cycle of this species.

To provide context to our odds, we can easily calculate the standard deviation of our features using the `np.std` function:

```
np.std(X_train, 0)
```

The following series is what is outputted by the `np.std` function:

age	6.757537
gender	0.476697
height	8.186987
weight	14.335173
ap_hi	16.703572
ap_lo	9.547583
cholesterol	0.678878
gluc	0.571231
smoke	0.283629
alco	0.225483
active	0.397215
dtype:	float64

As you can tell by the output, binary and ordinal features only typically vary by one at most, but continuous features, such as `weight` or `ap_hi`, can vary 10 – 20 times more, as evidenced by the standard deviation of the features.

Another reason why *odds* cannot be used to measure feature importance is because despite favorable odds, sometimes features are not statistically significant. They are entangled with other features in such a way they might appear to be significant, but we can prove that they aren't. This can be seen in the summary table for the model, under the $P > |z|$ column. This value is called the **p-value**, and when it's less than 0.05, hypothesis testing determines that there's strong evidence that it is significant. However, when it's above this number, especially by a large margin, there's no statistical evidence that it affects the predicted score. Such is the case with `gender`, at least in this dataset.

If we are trying to obtain what features matters most, one way to approximate this is to multiply the coefficients by the standard deviations of the features. Incorporating the standard deviations accounts for differences in variances between features. Hence, it is better if we get `gender` out of the way too while we are at it:

```
coefs = log_result.params.drop(labels=['const', 'gender'])
stdv = np.std(X_train, 0).drop(labels='gender')abs(coefs *
stdv).sort_values(ascending=False)
```

The preceding code produced this output:

<code>ap_hi</code>	0.936632
<code>age</code>	0.344855
<code>cholesterol</code>	0.334750
<code>weight</code>	0.158651
<code>ap_lo</code>	0.100419
<code>active</code>	0.085436
<code>gluc</code>	0.065982
<code>alco</code>	0.046230
<code>smoke</code>	0.037040
<code>height</code>	0.029620
<code>dtype:</code>	<code>float64</code>

The preceding table can be interpreted as an **approximation of risk factors** from high to low according to the model. It is also a **model-specific** feature importance method, in other words, a **global model (modular) interpretation method**. There's a lot of new concepts to unpack here so let's break them down.

Model interpretability method types

There are two model interpretability method types:

- **Model-specific:** When the method can only be used for a specific model class, then it's model-specific. The method detailed in the previous example can only work with logistic regression because it uses its coefficients.
- **Model-agnostic:** These are methods that can work with any model class. We cover these in *Chapter 4, Fundamentals of Feature Importance and Impact*, onward.

Model interpretability scopes

There are several model interpretability scopes:

- **Global holistic interpretation:** You can explain how a model makes predictions simply because you can comprehend the entire model at once with a complete understanding of the data, and it's a trained model. For instance, the simple linear regression example in *Chapter 1, Interpretation, Interpretability, and Explainability; and Why Does It All Matter?*, can be visualized in a two-dimensional graph. You can conceptualize this in memory, but this is only possible because the simplicity of the model allows you to do so, and it's not very common nor expected.
- **Global modular interpretation:** In the same way that you can explain the role of *parts* of an internal combustion engine in the *whole* process of turning fuel into movement, you can also do so with a model. For instance, in the CVD risk factor example, our feature importance method tells us that `ap_hi` (systolic blood pressure), `age`, `cholesterol`, and `weight` are the *parts* that impact the *whole* the most. Feature importance is only one of many global modular interpretation methods but arguably the most important one. *Chapter 4, Fundamentals of Feature Importance and Impact*, goes into more detail on feature importance.
- **Local single-prediction interpretation:** You can explain why a single prediction was made. The next example will illustrate this concept.
- **Local group-prediction interpretation:** The same as single-prediction, except that it applies to groups of predictions.

Congratulations! You've already determined the risk factors with a **global model interpretation method**, but the health minister also wants to know whether the model can be used to interpret individual cases. So, let's look into that.

Interpreting individual predictions with logistic regression

What if you used the model to predict CVD for the entire test dataset? You could do so like this:

```
y_pred = log_result.predict(sm.add_constant(X_test)).to_numpy()
print(y_pred)
```

The resulting array is the probabilities that each test case is positive for CVD:

```
[0.40629892 0.17003609 0.13405939 ... 0.95575283 0.94095239
 0.91455717]
```

Let's take one of the positive cases; test case #2872:

```
print(y_pred[2872])
```

We know that it predicted positive for CVD because the score exceeds 0.5:

```
0.5746680418975686
```

And these are the details for test case #2872:

```
print(X_test.iloc[2872])
```

The following is the output:

age	60.521849
gender	1.000000
height	158.000000
weight	62.000000
ap_hi	130.000000
ap_lo	80.000000
cholesterol	1.000000
gluc	1.000000
smoke	0.000000
alco	0.000000
active	1.000000
Name: 46965, dtype: float64	

So, by the looks of the preceding series, we know that the following applies to this individual:

- A borderline high `ap_hi` (systolic blood pressure).
- Normal `ap_lo` (diastolic blood pressure). Having high systolic blood pressure and normal diastolic blood pressure is what is known as *isolated systolic hypertension*. It could be causing a positive prediction, but `ap_hi` is borderline (130 mmHg being the border), so therefore the condition of *isolated systolic hypertension* is borderline.
- `age` is not too old, but among the oldest in the dataset.
- `cholesterol` is normal.
- `weight` also appears to be in the healthy range.

There are also no other risk factors: glucose is normal, no smoking, no alcohol, and no sedentarism, since the individual is active. It is not clear exactly why it's positive. Is the age and borderline *isolated systolic hypertension* enough to tip the scales? It's tough to understand the reasons for the prediction without putting all the predictions into context, so let's try to do that!

But how do we put everything in context at the same time? We can't possibly visualize how one prediction compares with the other ten thousand for every single feature and their respective predicted CVD diagnosis. Unfortunately, humans can't process that level of dimensionality, even if it were possible to visualize a ten-dimensional hyperplane!

However, we can do it for two features at a time, resulting in a graph that conveys where the decision boundary for the model lies for those features. On top of that, we can overlay what the predictions were for the test dataset based on all the features. This is to visualize the discrepancy between the effect of two features and all eleven features.

This graphical interpretation method is what is termed a **decision boundary**. It draws boundaries for the classes, leaving areas that belong to one class or another. Such areas are called **decision regions**. In this case, we have two classes, so we will see a graph with a single boundary between `cardio=0` and `cardio=1`, only concerning the two features we are comparing.

We have managed to visualize the two decision-based features at a time, with one big assumption that if all the other features are held constant, we can observe only two in isolation. This is also known as the **ceteris paribus** assumption and is critical in a scientific inquiry, allowing us to *control* some variables in order to *observe* others. One way to do this is to fill them with a value that won't affect the outcome. Using the table of odds we produced, we can tell whether a feature increases as it will increase the odds of CVD. So, in aggregate, a lower value is less risky for CVD.

For instance, `age=30` is the least risky value of those present in the dataset for `age`. It can also go in the opposite direction, so `active=1` is known to be less risky than `active=0`. We can come up with optimal values for the remainder of the features:

- `height=165`.
- `weight=57` (optimal for that height).
- `ap_hi=110`.
- `ap_lo=70`.
- `smoke=0`.
- `cholesterol=1` (this means normal).
- `gender` can be coded for male or female, which doesn't matter because the odds for gender (0.977519) are so close to 1.

The following `filler_feature_values` dictionary exemplifies what should be done with the features matching their index to their least risky values:

```
filler_feature_values = {0: 1, 1: 30, 2: 1, 3: 165, 4: 57, 5:
110, 6: 70, 7: 1, 8: 1, 9: 0, 10:0, 11:1 }
```

In the dictionary, the features are numbered and not named because the function we will use to plot the decision regions only takes in NumPy arrays. Also, since, in `statsmodels`, you must explicitly define the **constant** (also known as the **intercept**), the logistic model has an additional 0 feature, which always equals 1.

We also intend to plot the actual predictions for the test dataset. To do this, we must define another dictionary like `filler_feature_values`, but with a range so that, for example, the `filler_feature_value` for `height` is 165. We can then make this range 120, so it includes all cases with heights 165 ± 110 , so this means a range of $[55 - 275]$, which contains all possible heights in the test dataset:

```
filler_feature_ranges = {0: 1, 1: 35, 2: 2, 3: 110, 4: 150, 5:
140, 6: 70, 7: 3, 8: 3, 9: 2, 10:2, 11:2 }
```


The next thing to do is to create a (1,12) shaped NumPy array with test case #2872 so that the plotting function can highlight it. To this end, we first convert it to NumPy and then prepend the *constant* of 1, which must be the first feature, and then reshape it so that it meets the (1,12) dimensions:

```
X_highlight = np.reshape(\
    np.concatenate(([1], X_test.iloc[2872].to_numpy())), (1,
    12))
print(X_highlight)
```

The following is the output:

```
[[ 1.          60.52184865  1.          158.          62.
 130.          80.          1.           1.           0.
 0.           1.          ]]
```

We are good to go now! Let's visualize some decision region plots! We will compare the feature that is thought to be the highest *risk factor*, *ap_hi*, with the following four most important risk factors: age, cholesterol, weight, and *ap_lo*.

The following code will generate the plots in *Figure 2.2*:

```
plt.rcParams.update({'font.size': 14})
fig, axarr = plt.subplots(2, 2, figsize=(12,8), sharex=True,\
    sharey=False)
mldatasets.create_decision_plot(X_test, y_test, log_result, [5,
1], \
    ['ap_hi [mmHg]', 'age [years]'], X_highlight,\
    filler_feature_values, filler_feature_ranges,\
    ax=axarr.flat[0])
mldatasets.create_decision_plot(X_test, y_test, log_result, [5,
7], ['ap_hi [mmHg]', 'cholesterol [1-3]'], X_highlight,\
    filler_feature_values, filler_feature_ranges,\
    ax=axarr.flat[1])
mldatasets.create_decision_plot(X_test, y_test, log_result, [5,
6], ['ap_hi [mmHg]', 'ap_lo [mmHg]'], X_highlight,\
    filler_feature_values, filler_feature_ranges,\
    ax=axarr.flat[2])
```

```

mldatasets.create_decision_plot(X_test, y_test, log_result, [5,
4], ['ap_hi [mmHg]', 'weight [kg]'], X_highlight,\
    filler_feature_values, filler_feature_ranges,\
    ax=axarr.flat[3])
plt.subplots_adjust(top = 1, bottom=0, hspace=0.2, wspace=0.2)
plt.show()

```

In the plot in *Figure 2.2*, the circle represents test case #2872. In all the plots bar one, this test case is on the negative (left-side) decision region, representing `cardio=0` classification. The borderline high `ap_hi` (systolic blood pressure) and the relatively high age is barely enough for a positive prediction in the top-left chart. Still, in any case, for test case #2872, we have predicted a 57% score for CVD, so this could very well explain most of it.

Not surprisingly, by themselves, `ap_hi` and a healthy `cholesterol` are not enough to tip the scales in favor of a definitive CVD diagnosis according to the model because it's decidedly in the negative decision region, and neither is a normal `ap_lo` (diastolic blood pressure). You can tell from these three charts that although there's some overlap in the distribution of squares and triangles, there is a tendency for more triangles to gravitate toward the positive side as the y -axis increases, while fewer squares populate this region:

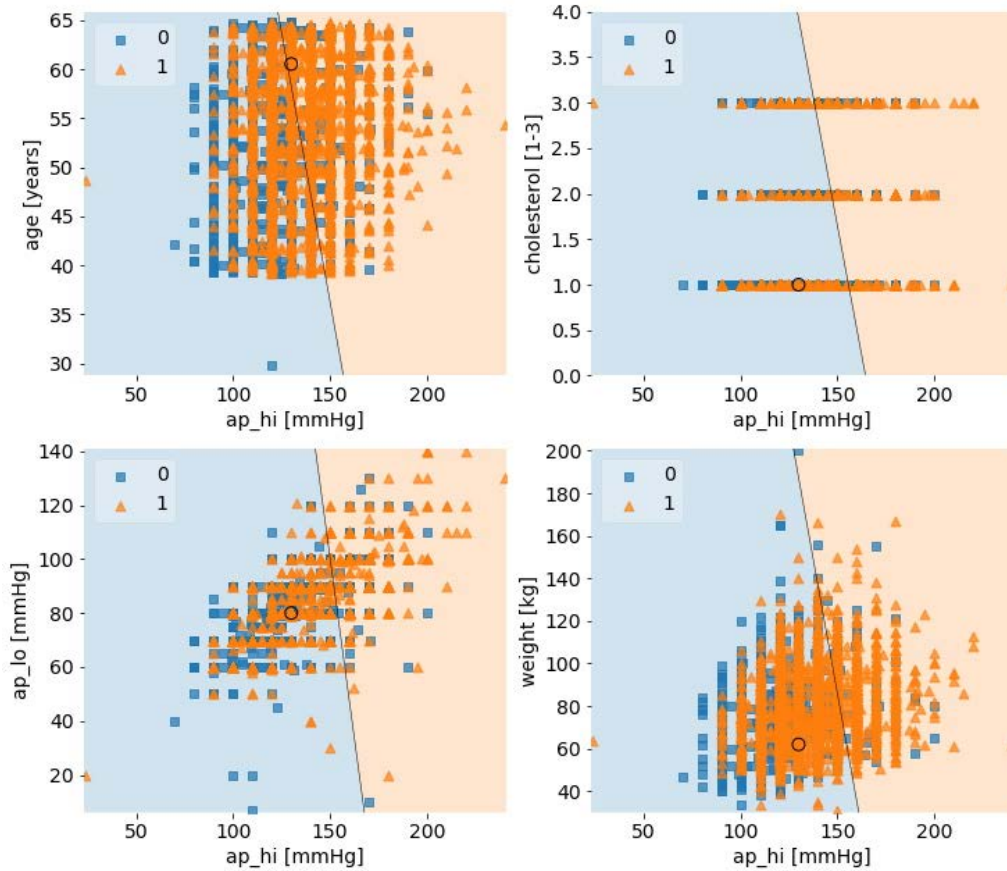


Figure 2.2 – The decision regions for `ap_hi` and other top risk factors, with test case #2872

The overlap across the decision boundary is expected because, after all, these squares and triangles are based on the effects of **all** features. Still, you expect to find a somewhat consistent pattern. The chart with `ap_hi` versus `weight` doesn't have this pattern vertically as `weight` increases, which suggests something is missing in this story... Hold that thought because we are going to investigate that in the next section!

Congratulations! You have completed the second part of the minister's request.

Decision region plotting, a **local model interpretation method**, provided the health ministry with a tool to interpret individual case predictions. You could now extend this to explain several cases at a time, or plot all-important feature combinations to find the ones where the circle is decidedly in the positive decision region. You can also change some of the filler variables one at a time to see how they make a difference. For instance, what if you increase the filler age to the median age of 54 or even to the age of test case #2872. Would a borderline high `ap_hi` and healthy `cholesterol` now be enough to tip the scales? We will answer this question later, but first let's understand what can make machine learning interpretation so difficult.

Appreciating what hinders machine learning interpretability

In the last section, we were wondering why the chart with `ap_hi` versus `weight` didn't have a conclusive pattern. It could very well be that although `weight` is a risk factor, there are other critical *mediating variables* that could explain the increased risk of CVD. A **mediating variable** is one that influences the strength between the independent and target (*dependent*) variable. We probably don't have to think too hard to find what is missing. In *Chapter 1, Interpretation, Interpretability, and Explainability; and Why Does It All Matter?*, we performed linear regression on `weight` and `height` because there's a linear relationship between these variables. In the context of human health, `weight` is not nearly as *meaningful* without `height`, so you need to look at both.

Perhaps if we plot the decision regions for these two variables, we will get some clues. We can plot them with the following code:

```
fig, ax = plt.subplots(1,1, figsize=(12,8))
mldatasets.create_decision_plot(X_test, y_test, log_result, [3,
4], ['height [cm]', 'weight [kg]'], X_highlight,\
    filler_feature_values, filler_feature_ranges, ax=ax)
plt.show()
```

The preceding snippet will generate the plot in *Figure 2.3*:

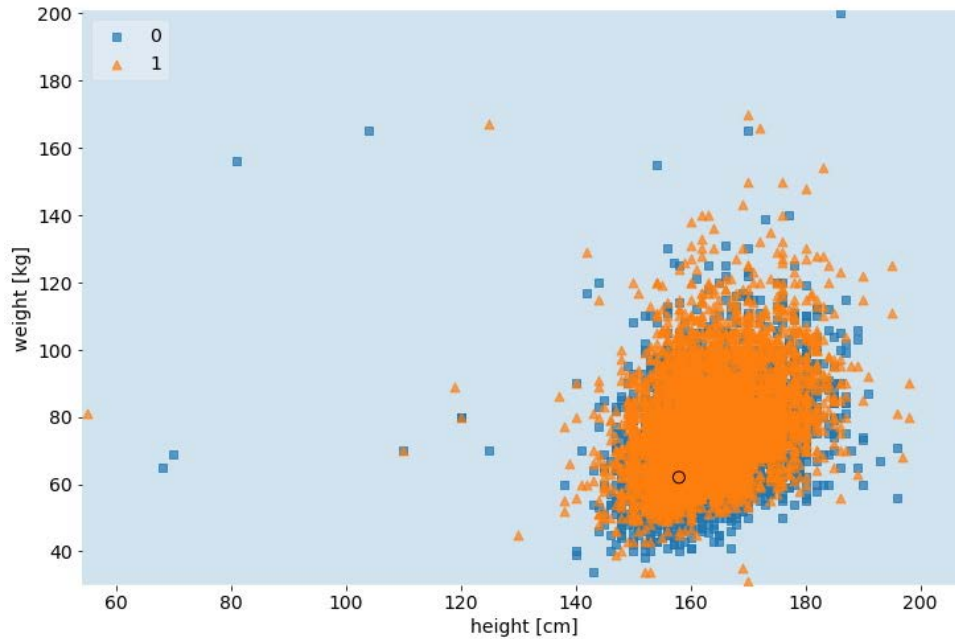


Figure 2.3 – The decision regions for weight and height, with test case #2872

No decision boundary was ascertained in *Figure 2.3* because if all other variables are held constant (at a less risky value), no height and weight combination is enough to predict CVD. However, we can tell that there is a pattern for the orange triangles, mostly located in one ovular area. This provides exciting insight that even though we expect weight to increase when height increases, the concept of an inherently unhealthy weight is not one that increases linearly with height.

In fact, for almost two centuries, this relationship has been mathematically understood by the name **body mass index (BMI)**:

$$\text{BMI} = \frac{\text{weight}_{kg}}{\text{height}_m^2}$$

Before we discuss BMI further, you must consider complexity. Dimensionality aside, there are chiefly three things that introduce complexity that makes interpretation difficult:

1. Non-linearity
2. Interactivity
3. Non-monotonicity

Non-linearity

Linear equations such as $y = a + bx$ are easy to understand. They are additive, so it is easy to separate and quantify the effects of each of its terms (a and bx) from the outcome of the model (y). Many model classes have linear equations incorporated in the math. These equations can both be used to fit the data to the model and describe the model.

However, there are model classes that are inherently non-linear because they introduce non-linearity in their training. Such is the case for *deep learning* models because they have non-linear activation functions such as *sigmoid*. However, logistic regression is considered a **generalized linear model (GLM)** because it's additive. In other words, the outcome is a sum of weighted inputs and parameters. We will discuss GLMs further in *Chapter 3, Challenges of Interpretability*.

However, even if your model is linear, the relationships between the variables may not be linear, which can lead to poor performance and interpretability. What you can do in these cases is adopt either of the following approaches:

- *Use a non-linear model class*, which will fit these non-linear feature relationships much better, possibly improving model performance. Nevertheless, as we will explore in more detail in the next chapter, this can make it less interpretable.
- *Use domain knowledge to engineer a feature that can help "linearize" it*. For instance, if you had a feature that increased exponentially against another, you can engineer a new variable with the logarithm of that feature. In the case of our CVD prediction, we know BMI is a better way to understand weight in the company of height. Best of all, it's not an *arbitrary* made-up feature, so it's easier to interpret. We can prove this point by making a copy of the dataset, engineering the BMI feature in it, training the model with this extra feature, and performing local model interpretation. The following code snippet does just that:

```
X2 = cvd_df.drop(['cardio'], axis=1).copy()
X2["bmi"] = X2["weight"] / (X2["height"]/100)**2
X2_train, X2_test, __, _ = train_test_split(X2, y, \
                                           test_size=0.15, random_state=9)
```

To illustrate this new feature, let's plot BMI against both weight and height using the following code:

```
fig, axs = plt.subplots(1,3, figsize=(15,4))
axs[0].scatter(X2["weight"], X2["bmi"], color='black',
s=2) axs[0].set_xlabel('weight [kg]')
axs[0].set_ylabel('bmi')
axs[1].scatter(X2["height"], X2["weight"], color='black',
s=2)
axs[1].set_xlabel('height [cm]')
axs[1].set_ylabel('weight [kg]')
axs[2].scatter(X2["bmi"], X2["height"], color='black',
s=2) axs[2].set_xlabel('bmi')
axs[2].set_ylabel('height [cm]')
plt.subplots_adjust(top = 1, bottom=0, hspace=0.2,
wspace=0.3) plt.show()
```

Figure 2.4 is produced with the preceding code:

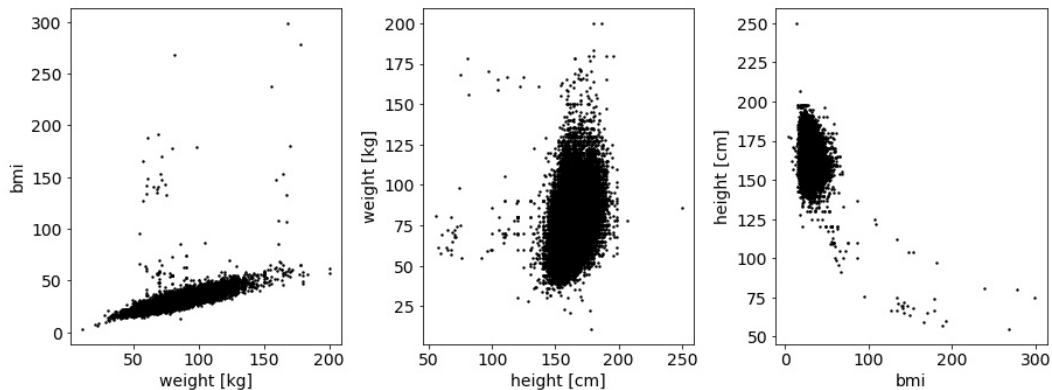


Figure 2.4 – Bivariate comparison between weight, height, and bmi

As you can appreciate by the plots in *Figure 2.4*, there is a more definite linear relationship between bmi and weight than between height and weight and, even, between bmi and height.

Let's fit the new model with the extra feature using the following code snippet:

```
log_model2 = sm.Logit(y_train, sm.add_constant(X2_train))
log_result2 = log_model2.fit()
```

Now, let's see whether test case #2872 is on the positive decision region when comparing ap_hi to bmi:

```
filler_feature_values2 = {0: 1, 1: 60, 2: 1, 3: 165, 4:
57, 5: 110, 6: 70, 7: 1, 8: 1, 9: 0, 10:0, 11:1, 12:20
}
filler_feature_ranges2 = {0: 1, 1: 35, 2: 2, 3: 120, 4:
150, 5: 140, 6: 70, 7: 3, 8: 3, 9: 2, 10:2, 11:2, 12:250
}
X2_highlight = np.reshape(\
    np.concatenate(([1],X2_test.iloc[2872].to_numpy())), (1,
13))
fig, ax = plt.subplots(1,1, figsize=(12,8))
mldatasets.create_decision_plot(X2_test, y_test, log_
result2, [5, 12], ['ap_hi [mmHg]', 'bmi'],\ X2_highlight,
filler_feature_values2,\
                                filler_feature_ranges2, ax=ax)
plt.show()
```

The preceding code plots decision regions in the following *Figure 2.5*:

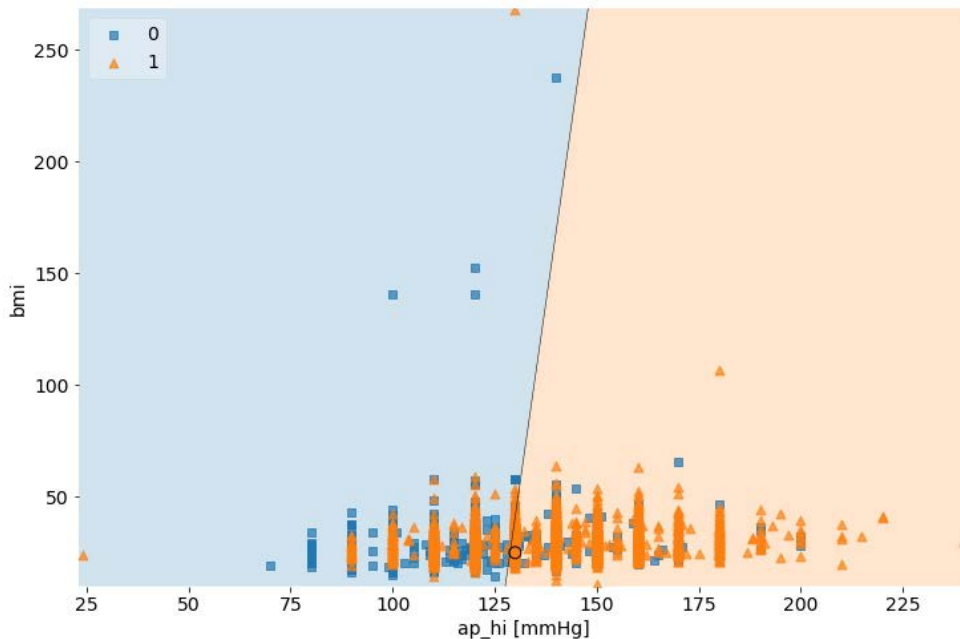


Figure 2.5 – The decision regions for ap_hi and bmi, with test case #2872

Figure 2.5 shows that `ap_hi` and `bmi` can help explain the positive prediction for CVD because the circle is in the positive decision region. Please note that there are some likely anomalous `bmi` outliers (the highest BMI ever recorded was 204), so there are probably some incorrect weights or heights in the dataset.

What's the problem with outliers?

Outliers can be **influential** or **high leverage** and therefore affect the model when trained with these. Even if they don't, they can make interpretation more difficult. If they are **anomalous**, then you should remove them, as we did with blood pressure at the beginning of this chapter. And sometimes, they can hide in plain sight because they are only perceived as *anomalous* in the context of other features. In any case, there are practical reasons why outliers are problematic, such as making plots like the preceding one "zoom out" to be able to fit them while not letting you appreciate the decision boundary where it matters. And there are also more profound reasons, such as losing trust in the data, thereby tainting trust in the models that were trained on that data. This sort of problem is to be expected with real-world data. Even though we haven't done it in this chapter for the sake of expediency, it's essential to begin every project by thoroughly exploring the data, treating missing values and outliers, and other data housekeeping tasks.

Interactivity

When we created `bmi`, we didn't only linearize a non-linear relationship, but we also created interactions between two features. `bmi` is, therefore, an **interaction feature**, but this was informed by domain knowledge. However, many model classes do this automatically by permutating all kinds of operations between features. After all, features have *latent* relationships between one another, much like `height` and `width`, and `ap_hi` and `ap_lo`. Therefore, automating the process of looking for them is not always a bad thing. In fact, it can even be absolutely necessary. This is the case for many deep learning problems where the data is unstructured and, therefore, part of the task of training the model is looking for the latent relationships to make sense of it.

However, for structured data, even though interactions can be significant for model performance, they can hurt interpretability by adding potentially unnecessary complexity to the model and also finding latent relationships that **don't mean anything** (which is called a **spurious relationship or correlation**).

Non-monotonicity

Often, a variable has a meaningful and consistent relationship between a feature and the target variable. So, we know that as age increases, the risk of CVD (*cardio*) must increase. There is no point at which you reach a certain age and this risk drops. Maybe the risk slows down, but it does not drop. We call this **monotonicity**, and functions that are *monotonic* are either always increasing or decreasing throughout their entire domain.

Please note that **all** linear relationships are monotonic, but not all monotonic relationships are necessarily linear. This is because they don't have to be a straight line. A common problem in machine learning is that a model doesn't know about a monotonic relationship that we expect because of our domain expertise. Then, because of noise and omissions in the data, the model is trained in such a way in which there are ups and downs where you don't expect them.

Let's propose a hypothetical example. Let's imagine that due to a lack of availability of data for 57-60-year-olds, and because the few cases we did have for this range were negative for CVD, the model could learn that this is where you would expect a drop in CVD risk. Some model classes are inherently monotonic, such as logistic regression, so they can't have this problem, but many others do. We will examine this in more detail in *Chapter 12, Monotonic Constraints and Model Tuning for Interpretability*:

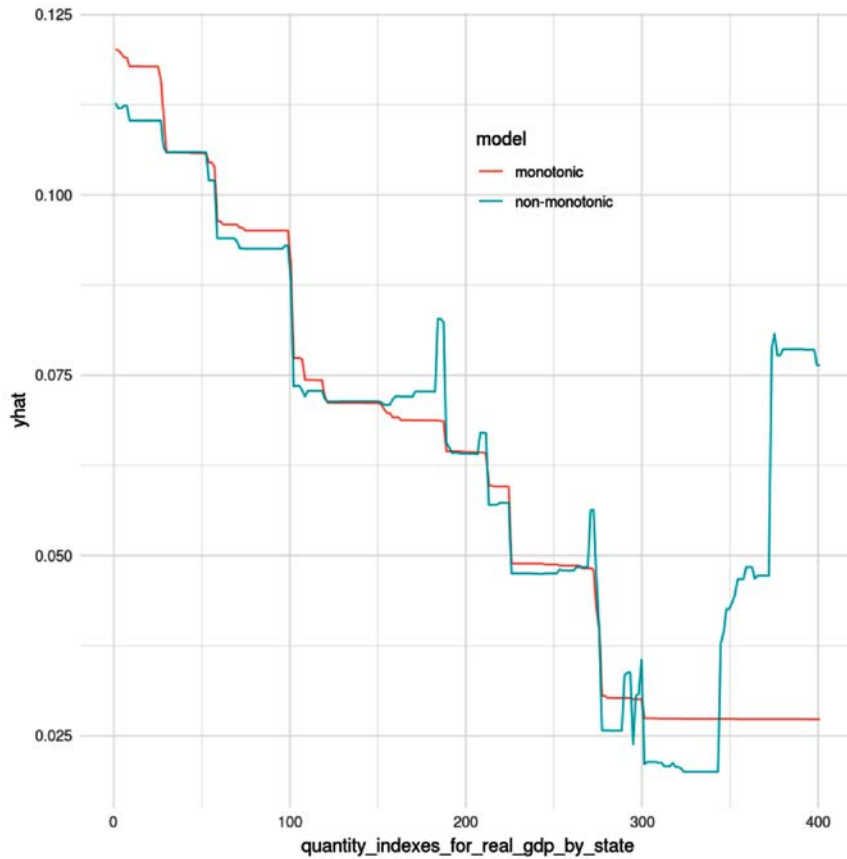


Figure 2.6 – A partial dependence plot between a target variable (yhat) and a predictor with monotonic and non-monotonic models

Figure 2.6 is what is called a **Partial Dependence Plot (PDP)**, from an unrelated example. PDPs are a concept we will study in further detail in *Chapter 4, Fundamentals of Feature Importance and Impact*, but what is important to grasp from it is that the prediction yhat is supposed to decrease as the feature quantity_indexes_for_real_gdp_by_state increases. As you can tell by the lines, in the monotonic model, it consistently decreases, but in the non-monotonic one, it has jagged peaks as it decreases, and then increases at the very end.

Mission accomplished

The first part of the mission was to understand risk factors for cardiovascular disease, and you've determined that the top four risk factors are systolic blood pressure (`ap_hi`), age, cholesterol, and weight according to the logistic regression model, of which only age is non-modifiable. However, you also realized that systolic blood pressure (`ap_hi`) is not as meaningful on its own since it relies on diastolic blood pressure (`ap_lo`) for interpretation. The same goes for weight and height. We learned that the interaction of features plays a crucial role in interpretation, and so does their relationship with each other and the target variable, whether linear or monotonic. Furthermore, the data is only a representation of the truth, which can be wrong. After all, we found *anomalies* that, left unchecked, can bias our model.

Another source of bias is how the data was collected. After all, you can wonder why the model's top features were all objective and examination features. Why isn't smoking nor drinking a larger factor? To verify whether there was *sample bias* involved, you would have to compare with other more trustworthy datasets to check whether your dataset is under-representing drinkers and smokers. Or maybe the bias was introduced by the question that asked whether they smoked now, and not whether they had ever smoked for an extended period.

Another type of bias that we could address is *exclusion bias* — our data might be missing information that explains the truth that the model is trying to depict. For instance, we know through medical research that blood pressure issues such as isolated systolic hypertension, which increases CVD risk, are caused by underlying conditions such as diabetes, hyperthyroidism, arterial stiffness, and obesity, to name a few. The only one of these conditions that we can derive from the data is obesity, and not the other ones. If we want to be able to interpret a model's predictions well, we need to have all relevant features. Otherwise, there will be gaps we cannot explain. Maybe once we add them, they won't make much of a difference, but that's what the methods we will learn in *Chapter 10, Feature Selection and Engineering for Interpretability*, are for.

The second part of the mission was to be able to interpret individual model predictions. We can do this well enough by plotting decision regions. It's a simple method, but it has many limitations, especially in situations where there are more than a handful of features, and they tend to interact a lot with each other. *Chapter 6, Local Model-Agnostic Interpretation Methods*, and *Chapter 7, Anchor and Counterfactual Explanations*, will cover better local interpretation methods. However, the decision region plot method helps illustrate many of the concepts surrounding decision boundaries we will discuss in those chapters.

Summary

After reading this chapter, you should know about two model interpretation methods: feature importance and decision boundaries. You also learned about model interpretation method types and scopes and the three elements that impact interpretability in machine learning. We will keep mentioning these fundamental concepts in subsequent chapters. For a machine learning practitioner, it is paramount to be able to spot them so you can know what tools to leverage to overcome interpretation challenges. In the next chapter, we will dive deeper into this topic.

Further reading

- Molnar, Christoph. *Interpretable Machine Learning. A Guide for Making Black Box Models Explainable*, 2019: <https://christophm.github.io/interpretable-ml-book/>.
- *Mlxtend Documentation. Plotting Decision Regions*. http://rasbt.github.io/mlxtend/user_guide/plotting/plot_decision_regions/.

