Interpretable machine learning

A guide to making black box models interpretable.

Christoph Molnar

2017-11-09

Contents

Preface

Machine learning has a huge potential to improve products, processes and research. But machines usually don't give an explanation for their predictions, which hurts trust and creates a barrier for the adoption of machine learning. This book is about making machine learning interpretable. As the programmer of an algorithm you want to know whether you can trust the learned model. Did it learn generalizable features? Or are there some odd artifacts in the training data which the algorithm picked up? This book will give you an overview over techniques that you can use to make black boxes as transparent as possible and make their predictions interpretable. The first part of the book introduces algorithms that produce simple, interpretable models and instructions how to interpret the output. The later chapters focus on general tools that help analyzing complex models and making their decisions interpretable. In an ideal future, machines will be able to explain their decisions and make a transition into an algorithmic age more human.

This books is recommended for machine learning practitioners, data scientists, statisticians and anyone else interested in making machine decisions more human.

About me: My name is Christoph Molnar, I consider myself both a statistician and machine learner. I work as a data scientist and also offer courses in interpretable machine learning. If you are interested in bringing intepretability to your machine learning models, feel free to contact me!

Mail: christoph.molnar.ai@gmail.com

Website: https://christophm.github.io/

1

Introduction

1.1 Who should read this book

This book is for everyone who wants to learn how to make machine learning models more explainable.

It is a recommended reading for machine learning practitioners, statisticians, data scientists, scientists and everyone who has contact with machine learning applications. It contains one or the other formula, but it's kept at a manageable level of math. This book is not for people who are trying to learn machine learning from scratch. If you want to learn machine learning, there are loads of books and other resources for learning the basics.

1.2 Outline

This book starts out with framing the problem: What aspects does explainability have? It goes on and lays out the 'simple' machine learning models (linear models, decision trees, decision rules) that are interpretable. The following chapter explains different methods for explaining and understanding the data better, followed by the core chapter about methods for explaining the output of any black box model. Later methods that are specific for certain model classes are shown. The book concludes with an outlook on the future.

1.3 What is machine learning and why is it important?

So what is machine learning? In general we speak of predictive models, that take some input vector (features) and map it to an output. If the output is a categorical variable people often name it classification and if it is a numerical variable, then regression. Machine learning is a set of algorithms that can learn these mappings from training data, which are pairs of input features and output variable. The machine learning algorithm learns a model by changing parameters (like linear weights) or learning structures (like trees). The algorithm is guided

FIGURE 1.1
Guy on a pile of data explaining how math and data have to be stirred in a machine learning system until the right answers show up. Credits:xkcd.com

by a score or loss function that is minimized. A fully trained machine learning model can then be used to make predictions for new instances.

Recommendation of products, identifying street signs, counting people on the street, assessing a person's credit worthiness, detecting fraud: All these examples have in common that they can, and are increasingly, realized with machine learning models. The tasks are different, but the approach is the same: Step 1 is to collect data. This can be images with and without street signs plus the information which sign is visible or the personal data from loan applicants together with the information if they repaid their loan or not. Step 2: Feed this information into a machine learning algorithm, which produces a sign detector model or a credit worthiness model. This model can then be used in Step 3: Integrate the model into the product or process, like an self-driving car or a loan application process.

There are a lot of tasks in which machines exceed humans. Even if the machine is as good as a human at a task, or slightly worse, there remains big advantages, and that is speed, reproducibility and scale. A machine learning model that has been implemented once, can do a task much faster than humans, will reliably produce the same results from the same input and can be copied endlessly.

2

Interpretability

So far, I haven't found a good scientific definition of "Machine learning model interpretability" or how to measure the goodness of an explanation. Throughout the book, I will use this rather simple, yet elegant definition from Miller (2017): **Interpretability is the degree to which a human can understand the cause of a decision.** The higher the interpretability of a model, the easier it is for someone to comprehend why certain decisions (read: predictions) were made. A model has better interpretability than another model, if it's decisions are easier to comprehend for a human than decisions from the second model. I will be using both the terms interpretable and explainable equally.

2.1 When is interpretability important?

Let' dive deeper into reasons why interpretability is important. Interpretability in machine learning is the ability of a machine learning system to explain or to present a decision in an understandable way for humans. Machine learning has come to a state where you have to make a trade-off: Do you simply want to know **what** will happen? For example if a client will churn or if medication will work well for a patient. Or do you want to know **why** something will happen and paying for the interpretability with accuracy? In some cases you don't care why a decision was made, only the assurance that the accuracy was good on a test dataset is enough. But in other cases knowing the 'why' can help you understand more about the problem, the data and also know why a model might fail. Two sorts of problems might not need explanations, because they either are low risk (e.g. movie recommender system) or the method is already extensively studied and evaluated (e.g. optical character recognition). The necessity for interpretability comes from an incompleteness in the problem formalization (Doshi-Velez and Kim, 2017), meaning that for certain problems/tasks it is not enough to get the answer (the "what"), but the model also has to give an explanation (the *why**)

- There is a shift in many scientific disciplines from qualitative to quantitative methods (e.g. sociology, psychology), and also towards machine learning (biology, genomics). The **goal of science** is to gain knowledge, but many problems can only be solved with big datasets and black box machine learning models. Interpretability allows to extract additional knowledge.
- It is **human nature** wanting to understand things, to have some form of control.
- Machine learning models are taking over real world tasks, that demand **safety measurements** and testing. A self-driving car automatically detects cyclists, which is as desired.

You want to bet 100% sure that the abstraction the system learned will be fail-safe, because running over cyclists is quite bad. An explanation might reveal that the most important feature learned is to recognize the two wheels of a bike and this explanation helps to think about edge cases like bikes with side bags, that partially cover the wheels.

- By default most machine learning models pick up biases from the training data. This can turn your machine learning models into racists (see Microsoft failed experiment: Tay) or discriminate in against other demographic, protected groups. Intepretability is a useful debugging tool for black box algorithm. So even in low risk environments (e.g. movie recommenders) explainability in the research and development stage is valuable. Also later when some model is used in a product, things can go wrong. And needed for explainability arises when something goes wrong. Because having an explanation for a faulty classification helps to understand the cause of the fault. It delivers a direction for how to fix the system. Consider an example of a husky versus wolf classifier, that missclassifies some huskies as wolfs. If there is an explanation to the classification you can see, that the missclassification happened due to the snow on the image. The classifier learned to use snow as a feature for classifying images as wolfs, which might make sense in terms of separating features in the training data set, but not in the real world use.

2.2 The bigger picture

Let's take a look from further away. What do we want to explain, and what kind of 'layers' are inbetween? The infographic displays the concepts, see Figure 2.1. The bottom layer is the 'World'. This could literally be nature itself, like the biology of the human body and how it reacts to medication, but also human behaviour like if people payed back their loans. The 'World'-layer contains everything that can be observed and is of interest. Ultimately we want to learn something about the 'World' and interact with it.

The second layer is the 'Data'-layer. We have to digitalise the 'World' in to make it processable for computers and also to store information. The 'Data'-layer contains anything from images, texts, tabular data and so on.

With machine learning on top of the 'Data'-layer we get to the 'Black Box Model'-layer. Machine learning algorithms learns with data from the real world to make predictions / classifications or finds structures.

Now with the 'Interpretable models'-layer we come the part that this book is concerned with. On top of the 'Black-Box-Layer' we want to have something that helps us deal with the opaqueness of machine learning models. What were the important attributes for a particular diagnosis? Why was a financial transaction classified as fraud?

On top of that, there is the 'Explanations'-layer. I put it as a layer separate from 'Interpretable models', since the simple models deal with capturing associations and it is useful to think of the explanation as independent. There are different ways to present the results of a linear

regression model for example, it could be a coefficient table, a coefficient plot with confidence intervals, a colored bar chart, a few sentences, ... It depends on the target audience what representation which explanation to choose.

The last layer is 'Human'. Look this one is waiving at you because you are reading this book and you are helping to provide better explanations for black box models! Humans are the consumers of the explanations ultimately.

This layered abstraction also helps in understanding what the difference between statisticians and machine learning practicioners is. Statistician are concerned with the 'Data' layer, like planning clinical trials or designing surveys. The they skip the 'Black Box Model'-layer and go right to the 'Interpretable Models' and from there to the explanations for our human. Machine learning specialists are also concerned with the 'Data'-layer, like collecting labeled samples of skin cancer images or crawling Wikipedia. Then comes the machine learning model. 'Interpretable models' and 'Explanations' are skipped and the human deals directly with the 'Black Box Model'. It's a nice thing, that in explainable machine learning, the work of a statistican and a machine learner fuses and becomes something better.

Of course this graphic does not capture everything: Data could come from simulations. Black box models also output predictions that might not even reach humans, but only feed other machines and so on. But overall it is a useful abstraction for understanding what explainable machine learning is.

2.3 Scope of intepretability

An algorithm trains a model, which produces the predictions. Each step can be evaluated in terms of transparency or explainability.

2.3.1 Algorithm transparency

How does the algorithm create the model?

Algorithm transparency is about how the algorithm learns a model from data and what kind of relationships it is capable of picking up. If you are using convolutional neural networks for classifying images, you can explain that the algorithm learns edge detectors and filters on the lowest layers. This is an understanding of how the algorithm works, but not of the specific model that is learned in the end and not about how single predictions are made. For this level of transparency only knowledge about the algorithm and not about the data or concrete learned models are required. This book focuses on model explainability. Algorithms like the least squares method for linear models are well studied and understood. They score high in transparency. Deep learning approaches (pushing a gradient through a network with millions of weights) are less understood and the inner workings are in the focus on-going research. It is not clear how they exactly work, so they are less transparent.

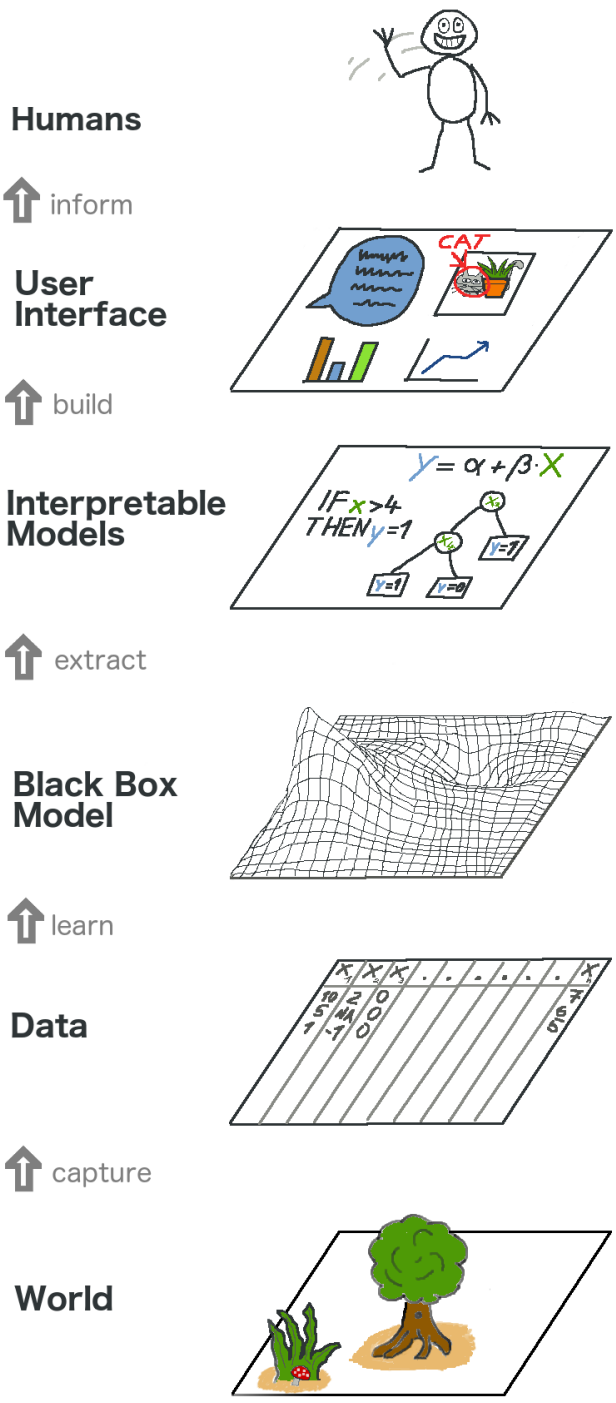


FIGURE 2.1
The big picture of explainable machine learning. The real world goes through many layers before it reaches the human in forms of explanations.

2.3.2 Global, holistic model explainability

How does the trained model make predictions?

You could call a model explainable if you can comprehend the whole model at once (Lipton, 2016). To explain the global model output, you need the trained model, knowledge about the algorithm and the data. This level of explainability is about understanding how the model makes the decisions, based on a holistic few on it's features and each learned components like weights, parameters and structures. Which features are the important ones and what kind of interactions are happening? Global model explainability helps to understand the distribution of your target variable based on the features. Arguably, global model explainability is very hard to achieve in practice. Any model that exceeds a handful of parameters or weights, probably won't fit an average human's brain capacity. I'd argue that you cannot really imagine a linear model with 5 features and draw in your head the hyperplane that the was estimated in the 5-dimensional feature space. Each feature space with more than 3 dimensions is just not imaginable for humans. Usually when people try to comprehend a model, they look at parts of it, like the weights in linear models.

2.3.3 Global model explainability on a modular level

How do parts of the model influence predictions?

You might not be able to comprehend a naive bayes model with many hundred features, because there is no way you could hold all the feature weights in your brain's working memory. But you can understand a single weight easily. Not many models are explainable on a strict parameter level.While global model explainability is usually out of reach, there is a better chance to understand at least some models on a modular level. In the case of linear models parts to understand are the weights and the distribution of the features, for trees it would be splits (used feature and cut-off point) and leaf node predictions. Linear models for example look like they would be, but the interpretation of a single weight is interlocked with all of the other weights. As you will see in Chapter [#limo], the interpretation of a single weight always comes with the footnote that the other input features stay at the same way value, which is not the case in many real world applications. A linear model predicting the rent of a flat, which takes into account both the size of the flat and the number of rooms might have a negative weight for the rooms feature, which is counter intuitive. But it can happen, because there is already the highly correlated flat size feature and in a market where people prefer bigger rooms, a flat with more rooms might be worth less than a flat with more rooms and same square meters. The weights only make sense in the light of the other features used in the model. But arguably a linear models weights still have better explainability than the weights of a deep neural network.

2.3.4 Explain the decision for a single instance

Why did the model make a specific decision for an instance?

You can go all the way down to a single observation and examine what kind of classification or decision the model gives for this input, and why it made this decision. When you look at one example, the local distribution of the target variable might behave more nicely. Locally it might depend only linearly or monotonic on some variables rather than having a complex dependency on the features. For example the rent of an apartment might not depend linearly on the size, but if you only look at a specific apartment of 100 square meter and check how the prize changes going up plus and minus 10 square meters there is a chance that this sub region in your data space is linear. Local explanations can be more accurate compared to global explanations because of this.

2.3.5 Explain the decisions for a group of instances

Why did the model make specific decisions for a group of instances?

The model output for multiple instances can be explained by using methods for global model explainability and single instance explanations. The global methods can be applied by taking the group of observations pretending it's the complete dataset and using the global methods on this subset. The single explanation methods can be used on each instance and listed or aggregated afterwards for the whole group.

2.4 Evaluating interpretability

There is no real consensus what explainability in machine learning is. Also it is not clear how to measure it.

2.4.1 Approaches for evaluation of the explanation quality

(Doshi-Velez and Kim, 2017) proposes 3 major levels of evaluating explainability. - Application level evaluation (real task): Put the explanation into the product and let the end user test it. On an application level the radiologists would test the fracture detection software in order to evaluate the model. This requires a good experimental setup and an idea of how to assess the quality. A good baseline for this is always how good a human would be at explaining the same decision. - Human level evaluation (simple task) is a simplified application level evaluation. The difference is that these experiments are not conducted with the domain experts, but with lay humans. This makes experiments less expensive (especially when the domain experts are radiologists) and it is easier to find more humans. An example would be to show a user different explanations and the human would choose the best . - Function level evaluation (proxy task) does not require any humans. This works best when the class of models used is already evaluated by someone else in a human level evaluation.

2.4.1.1 Function level evaluation

Model size is an easy way to measure, but might be too simplistic.

Dimensions of interpretability:

- Model sparsity: How many features are being used by the explanation?
- Monotonicity: Is there a Monotonicity constraint?
- Uncertainty: Is a measurement of uncertainty part of the explanation?
- Interactions: Is the explanation able to include interaction of features?
- Cognitive processing time: How long does it take to understand the explanation.
- Feature complexity: What features were used for the explanation? PCA components are harder to understand than word occurrences for example.
- Description length of explanation

If you can ensure that the machine learning model can explain decisions, following traits can also be checked more easily (Doshi-Velez and Kim, 2017).

- Fairness: Unbiased, not discrimating against protected groups (implicit or explicit). An interpretable model can tell you why it decided it decided a certain person is not worthy of a credit and for a human it becomes easy to decide if the decision was based on a learned demographic (e.g. racial) bias.
- Privacy: sensitive information in the data is protected.
- Reliability/Robustness: Small changes in the input don't lead to big changes in the ouput/decision.
- Causality: Only causal relationships are picked up. So a predicted change in a decision due to arbitrary changes in the input values, are also happening in reality.
- Usability:
- Trust: It is easier for humans to trust into a system that explains it's decisions compared to a black box

3

Datasets

Throughout the book all the models and techniques will be used on real datasets, which are freely available online. We will be using different datasets for different types of problems: classification, regression and text classification.

3.1 Bike sharing counts (regression)

This dataset contains daily counts of bike rentals from bike sharing company Capital-Bikeshare in Washington D.C., including weather and seasonal information. The data is kindly open source by Capital-Bikeshare and the folks from (Fanaee-T and Gama, 2013) have added weather data and seasonal information. The goal is to predict how many rental bike will be out on the street given weather and day. The data can be downloaded from the UCI Machine Learning Repository.

Variables in the dataset:

- instant: record index
- dteday : date
- season : season (1:springer, 2:summer, 3:fall, 4:winter)
- yr : year (0: 2011, 1:2012)
- mnth : month (1 to 12)
- hr : hour (0 to 23)
- holiday : weather day is holiday or not (extracted from [Web Link])
- weekday : day of the week
- workingday : if day is neither weekend nor holiday is 1, otherwise is 0.
- weathersit :
- 1: Clear, Few clouds, Partly cloudy, Partly cloudy
- 2: Mist + Cloudy, Mist + Broken clouds, Mist + Few clouds, Mist
- 3: Light Snow, Light Rain + Thunderstorm + Scattered clouds, Light Rain + Scattered clouds
- 4: Heavy Rain + Ice Pallets + Thunderstorm + Mist, Snow + Fog
- temp : Temperature in Celsius
- atemp: Feeling temperature in Celsius
- hum: Humidity
- windspeed: Wind speed in km per hour
- casual: count of casual users

- registered: count of registered users
- cnt: count of total rental bikes including both casual and registered

You can look at a sample of 50 dates here:

3.2 Youtube spam comments (text classification)

As an example for text classification we will be using 1956 comments from 5 different YouTube videos . Thankfully the authors that used this dataset in a paper about spam classification made the data freely available (Alberto et al., 2015).

Data was collected through the YouTube API from five of the ten most viewed videos on YouTube in the first half of 2015. All of the 5 videos are music videos. One of them is the wildly popular "Gangnam Style" from korean artist Psy. The other artists where Katy Perry, LMFAO, Eminem and Shakira.

You can flip through the comments. The comments had been hand labeled as spam or legitimate. Spam has been coded with a '1' and legitimate comments with a '0'.

You could also go over to YouTube and have a look at the comment section. But please don't get trapped in the YouTube hell, ending up watching videos about monkeys stealing and drinking coktails from tourists on the beach. Also the Google Spam detector probably has changed a lot since 2015. Watch the view-record breaking video below "Gangnam Style" below:

3.3 Risk factors for cervical cancer (classification)

The cervical cancer dataset contains indicators and risk factors for predicting if a woman gets cervical cancer. The features contain demograpics (e.g. age), habits and medical history. The data can be downloaded from the UCI Machine Learning repository.

The variable contained in the dataset are:

- (int) Age
- (int) Number of sexual partners
- (int) First sexual intercourse (age)
- (int) Num of pregnancies
- (bool) Smokes
- (bool) Smokes (years)
- (bool) Smokes (packs/year)
- (bool) Hormonal Contraceptives

- (int) Hormonal Contraceptives (years)
- (bool) IUD
- (int) IUD (years)
- (bool) STDs
- (int) STDs (number)
- (bool) STDs:condylomatosis
- (bool) STDs:cervical condylomatosis
- (bool) STDs:vaginal condylomatosis
- (bool) STDs:vulvo-perineal condylomatosis
- (bool) STDs:syphilis
- (bool) STDs:pelvic inflammatory disease
- (bool) STDs:genital herpes
- (bool) STDs:molluscum contagiosum
- (bool) STDs:AIDS
- (bool) STDs:HIV
- (bool) STDs:Hepatitis B
- (bool) STDs:HPV
- (int) STDs: Number of diagnosis
- (int) STDs: Time since first diagnosis
- (int) STDs: Time since last diagnosis
- (bool) Dx:Cancer
- (bool) Dx:CIN
- (bool) Dx:HPV
- (bool) Dx
- (bool) Hinselmann: target variable
- (bool) Schiller: target variable
- (bool) Cytology: target variable
- (bool) Biopsy: target variable

As the biopsy serves as the gold standard for diagnosing cervical cancer, the classification tasks in this book used the biopsy outcome as the target. Missing values for each column were imputed by the mode (most frequent value), which is probably a bad solution, because the value of the answer might be correlated with the probability for missingness. There is probably this bias, because the question are of a very private nature. But this is not a book about missing data imputation, so the mode imputation will suffice!

(Fernandes et al., 2017)

4

Definitions

- An **Algorithm** is a set of rules that a machine follows to achieve a particular goal (alg, 2017)
- **Machine learning algorithm** is an set of rules that a machine follows to learn how to a achieve a particular goal. The output of a machine learning algorithm is a machine learning model.
- **(Machine learning) Model** is the outcome of a machine learning algorithm. This can be a set of weights for a linear model or neural network plus the architecture.
- **Features** are the variables/information used for prediction/classification/clustering.
- **(machine learning) Task** can be classification, regression, survival analysis, clustering, outlier detection
- **Instance** One row in the dataset.

5

Interpretable models

The most straight forward way to achieve explainable machine learning algorithms is to use only a subset algorithms that yield an understandable model structure.

These are:

- Linear models (sparse)
- Decision trees
- Decision rules

In the following chapters we will talk about the algorithm with it's variants. Not in detail, only the basics, because there are already a ton of books, videos, tutorials, papers and so on about them. We will focus on how to interpret the models and why they are explainable. The chapter covers linear models, decision trees, decision rules, neighbour methods and graphical models.

5.1 Terminology

- Y is the target variable in supervised settings.
- X are the features or covariates.
- w are the weights.
- β are regression weights.

5.2 Overview

Algorithm	Linear	Monotonicity	Interaction built-in
Linear models	Yes	Yes	No
Decision trees	No	Not by default	Yes
Decision rules	No	Not by default	Yes
Naive bayes	Yes	Yes	No
Nearest neighbours	No	No	No

5.3 Linear models

Linear models have been and are still used by statistician, computer scientists and other people with quantitative problems. They learn straightforward linear (and monotonic) relationships between the target and the features. The target changes by a learned weight depending on the feature. Monotonicity makes the interpretation easy.

Linear models can be used to model the dependency of a regression variable (here Y) on K covariates. As the name says, the learned relationships are linear in the form of

$$y_i = \beta_0 + \beta_1 \cdot x_{i,1} + \ldots + \beta_K x_{i,K} + \epsilon_i$$

The i-th observation's outcome is a weighted sum of it's K features. The β_k represent the learned feature weights or coefficients. The ϵ_i is the error we are still making, the difference between the predicted and actual outcome.

The biggest advantage is the linearity: It makes the estimation procedure straight forward and most importantly these linear equations have an easy to understand interpretation. That is one of the main reasons why the linear model and all it's descendants are so widespread in academic fields like medicine, sociology, psychology and many more quantitative research fields. In this areas it is important to not only predict e.g. the clinical outcome of a patient, but also quantify the influence of the medication while at the same time accounting for things like sex, age and other variables.

Linear regression models also come with some assumptions that make them easy to use and interpret but are often not given in reality. The assumptions are: linearity, normality, homoscedasticity, independence, fixed features, abscence of multicollinearity. **Linearity**: Linear regression models allow the mean of the response to be only a linear combination of the features, which is both the greatest strength and biggest limitation. Linearity makes the estimation procedure easy. Also linearity leads to interpretable models: linear effects are simple to quantify and describe (see also next chapter) and are additive, so it is easy to separate the effects. If you suspect interactions of features or a non-linear association of a feature with the target value, then you can add interaction terms and things like regression splines to estimate non-linear effects. **Normality**: The target value given the features are assumed to follow a normal distribution. If this assumption is violated, then the estimated confidence intervals of the feature weights are not valid. Any interpretation of the p-values (p-value = Probability that the confidence interval of the feature weight covers the 0) is not valid. **Homoscedasticity** (constant variance): The variance of the error terms ϵ_i is assumed to be constant along the whole feature space. Let's say you want to predict the value of a house given the living area in square meters. You estimate a linear model, which assumes that no matter how big the flat, the error terms around the predicted response have the same variance. This assumption is in reality often violated. In the house example it is plausible that the variance of error terms around the predicted price is higher in bigger

houses, since also the prices are higher and there is more wriggle room for prices to vary. **Indepence**: Each observation is assumed to be independent from the next one. If you have repeated measurements, like multiple records per patient, the data points are not independent from each other and there are special linear model classes to deal with these cases, like mixed effect models or GEEs. **Fixed features**: The input features are seen as 'fixed', carrying no errors or variation, which, of course, is very unrealistic and only makes sense in controlled experimental settings. But not assuming fixed features would mean that you have to fit very complex measurement error models that account for the measurement errors. And usually you don't want to do that. **Abscence of multicollinearity**: Basically you don't want features to be highly correlated, because this messes up the estimation of your models. In a situation where two variables are highly correlated (something >0.9) the linear model will have problems estimating the weights, since the models are additive and the model does not know to which feature to attribute the effects.

5.3.1 Interpretation

The interpretation of the coefficients:

- Continuous regression variable: For an increase of one point of the variable x_j the estimated outcome changes by β_j
- Binary categorical variables: One of the variables is the reference level (in some languages the one that was coded in 0). A change of the variable x_i the reference level to the other category changes the estimated outcome by β_i
- categorical variables with many levels: One solution to deal with many variables is to one-hot-encode them, meaning each level gets it's own column. From a categorical variable with L levels, you only need L-1 columsn, otherwise it is over parameterized. The interpretation for each level is then according to the binary variables. Some language like R allow to
- Intercept β_0: The interpretation is: Given all continuous variables are zero and the categorical variables are on the reference level, the estimated outcome of y_i is β_0. The interpretation of β_0 is usually not relevant.

Another important measurement for interpreting linear models is the R^2 measurement. R^2 tells you how much of the total variance of your target variable is explained by the model. The higher R^2 the better your model explains the data. The formula to calculate R^2 is: $R^2 = 1 - SSE/SST$, where SSE is the squared sum of the error terms ($SSE = \sum_{i=1}^{n}(y_i - \hat{y}_i)^2$) and SST is the squared sum of the data variance ($SST = \sum_{i=1}^{n}(y_i - \bar{y})^2$). R^2 ranges between 0 for models that explain nothing and 1 for models that explain all of the datas variance.

There is a catch, because R^2 increases with the number of features in the model, even if they carry no information about the target value at all. So it is better to use the adjusted R-squared $\bar{R}^2$, which accounts for number of features used in the model. It's calculation is $\bar{R}^2 = R^2 - (1 - R^2)\frac{p}{n-p-1}$, where p is the number of features and n the number of observations.

It isn't helpful to do interpretation on a model with very low R^2 or $\bar{R}^2$, because basically the

model is not explaining much of the variance, so any interpretation of the weights are not meaningful.

5.3.2 Interpretation example

We now use the linear model to predict the bike rentals on a day, given weather and calendrical information.

	Estimate	Std. Error
(Intercept)	2579.832417	251.5665243
seasonSUMMER	864.839727	131.0342448
seasonFALL	54.305585	174.9524606
seasonWINTER	319.626870	119.3930717
holidayHOLIDAY	-639.783043	217.2195639
workingdayWORKING DAY	67.239296	78.1596156
weathersitMISTY	-394.473877	94.2803595
weathersitRAIN/SNOW/STORM	-1863.482641	225.0341002
temp	108.961908	7.6228315
hum	-18.001893	3.3250623
windspeed	-45.306120	7.2812447
days_since_2010	4.977309	0.1870971

Interpretation of a numerical variable ('Temperature'): An increase of the temperature of 1 degree Celsius increases the number of bikes by 108.96 given all other features stay the same.

Interpretation of a categorical variable ('weathersituation')): The number of bikes is -1863.48 lower when it is rainy, snowing or stormy, compared to good weather, given that all features stay the same. Also if the weather was only misty, the number of bike rentals was -394.47 lower, compared to good weather, given all features stay the same.

As you can see in the interpretation examples, the interpretations are always coming with the clause that 'all other features stay the same'. That's because of the nature of linear models: All features are input linearly into the function with no interactions (unless explicitly specified). The good side is, that is isolates the interpretation. If you think of the features as turn-switches that you can turn up or down, it is nice to see what happens when you would just turn the switch for one feature.

5.3.3 Interpretation templates

Interpretation of a numerical feature:

An increase of x_k by one unit increases the expectation for y by $\beta_x k$ units if all other features X stay the same.

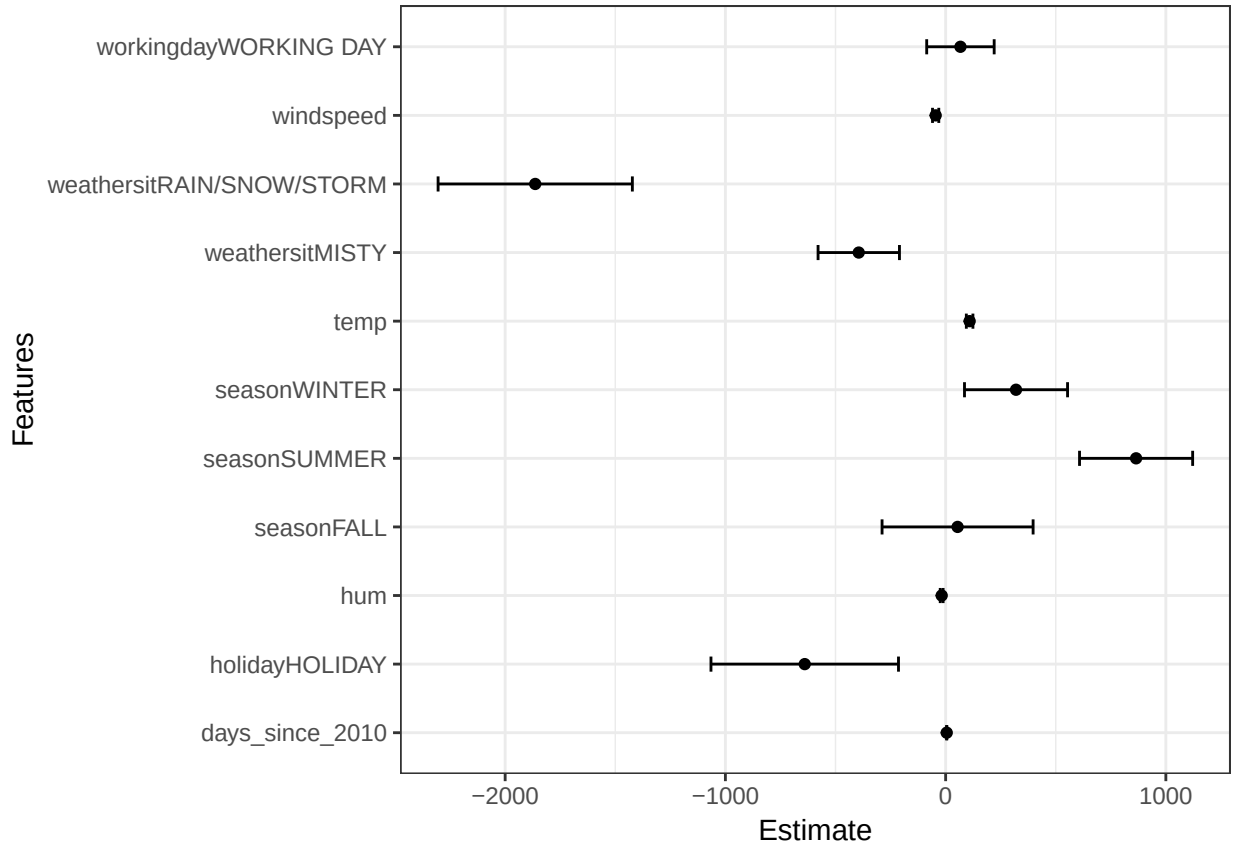
Interpretation of a categorical feature:

The category coded with 1 of x_k increases the expectation for y by β_k compared to the reference category (coded with 0).

5.3.4 Visual parameter interpretation

5.3.4.1 Weight plot

The information of the coefficient table can also be put into a visualization, which makes the weights and the uncertainty about them can be made understandable on one glance. The weight is displayed as a point and the 95% confidence interval around the point with a line. The 95% confidence interval means that if the linear model was repeated 100 times on

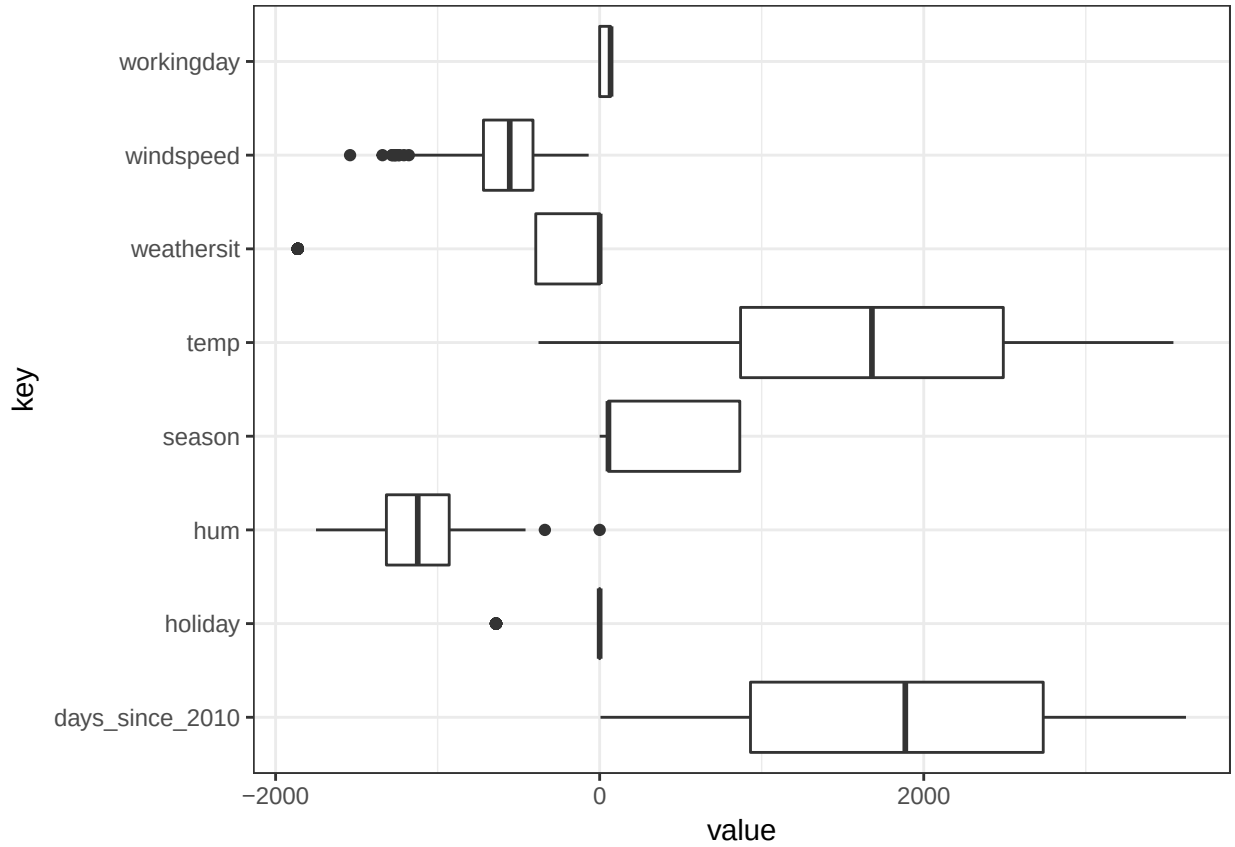


TODO: Add interpetation

5.3.4.2 Effect plot

The weights of the linear model only have meaning, when combined with the actual features. The weights depend on the scale of the features and will be different if you have a features measuring some height and you switch from inches to centemeters. The weight will change, but the actual relationships in your data will not. Also it is important to know the distribution of your feature in the data, because if you have a very low variance, it means that almost all instances will get a similar contribution from this feature. The effect plot can help to understand how much the combination of a weight and a feature contributes to the predictions

in your data. Start with the computation of the effects, which is the weight per feature times the feature of an instance: $eff_{i,k} = w_k \cdot x_{i,k}$. The resulting effects are visualized with boxplots: The box contains the effect range for half of your data (25% to 75% effect quantiles). The line in the box is the median effect, so 50% of the instances have a lower and the other half a higher effect on the prediction than the median value. The whiskers are $+/i1.58IQR/\sqrt{n}$, with IQR being the inter quartile range ($q_{0.75} - q_{0.25}$). The points are outlier to the whiskers.

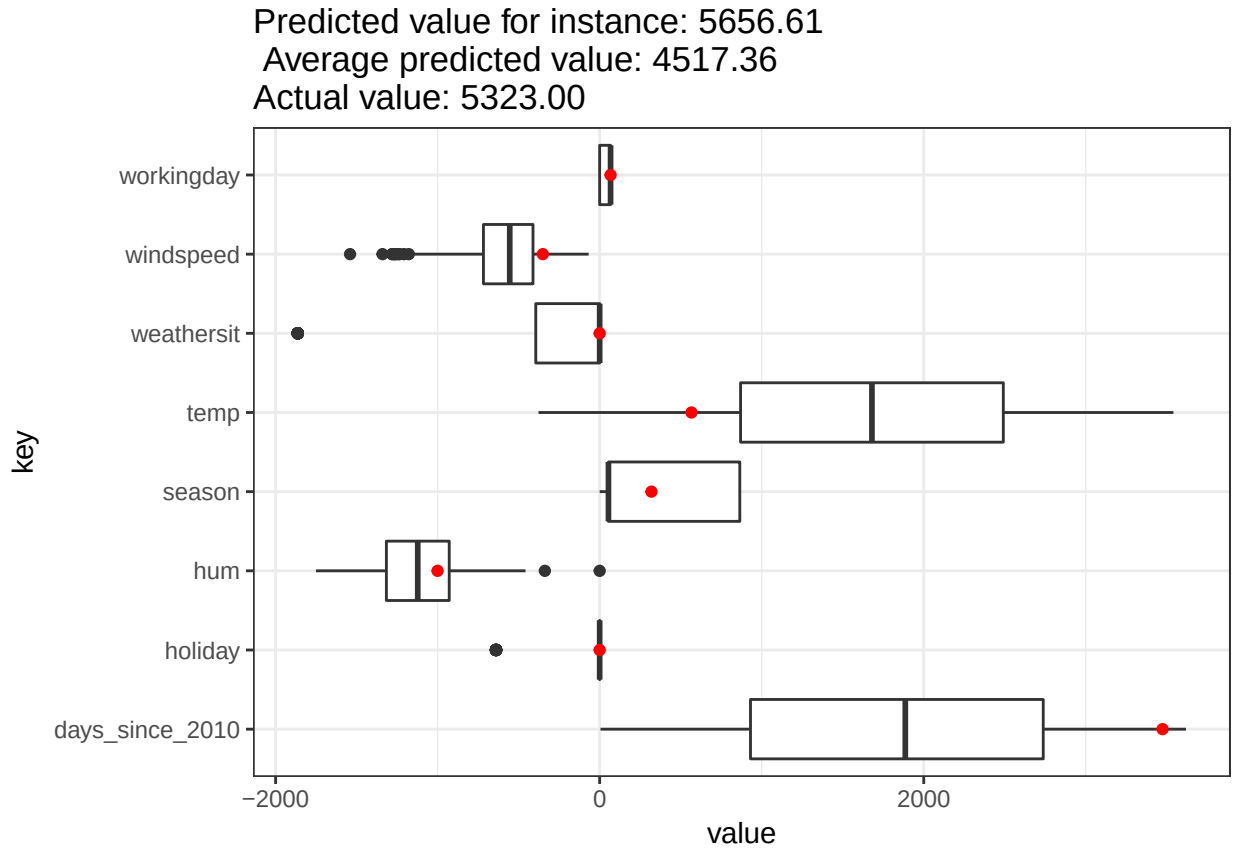


The largest contributions are from temperature and the days variable, which capture the trend that the bike rental became more popular over time. The temperature has a high contribution distribution. The day trend variable has goes from zero to large positive contribution, because the first day in the dataset (1.1.2011) get's a very low contribution, and the estimated weight with this feature is positive (4.98), so the effect gets higher with every day and is highest for the latest day in the dataset (31.12.2012). Note that for effects from a feature with a negative effect, the instances with a positive effect (or the least negatives) are the ones that have a negative feature value (negative times negative is positive), so days with a high positive effect of windspeed on the bike rental count have the lowest windspeeds.

5.3.5 Explaining single predictions

Why did a certain instance get the prediction it got from the linear model? This can again be answered by bringing together the weights and features and computing the effect. Now

the effect will tell you how much each feature contributed towards the sum of the prediction. This is only meaningful if you compare the instance specific effects with the mean effects.



Let's have a look at the effect realization for the rental bike count of one observation (= one day). Some features contribute unusually much to the predicted bike count: Temperature (5.20089) and the trend variable "days_since_2010", because this instance is from late 2011 (value =).

5.3.6 Coding categorical variables:

There are several ways to represent a categorical variable, which has an influence on the interpretation: http://stats.idre.ucla.edu/r/library/r-library-contrast-coding-systems-for-categorical-variables/ and http://heidiseibold.github.io/page7/

Described above is the treatment coding, which is usually sufficient. Using different codings boils down to creating different matrices from your one column with the categorical feature. I present three different codings, but there are many more. The example has six instances and one categorical feature with 3 levels. The first two instances are in category A, instances three and four are in category B and the last two instances are in category C.

- **Treatment coding** compares each level to the reference level. The intercept is the mean of the reference group. The first column is the intercept, which is always 1. Column two is an indicator whether instance i is in category B, columns three is an indicator for category

C. There is no need for a column for category A, because than the system would be over specified. Knowing that an instance is neither in category B or C is enough.

$$\begin{pmatrix} 1 & 0 & 0 \\ 1 & 0 & 0 \\ 1 & 1 & 0 \\ 1 & 1 & 0 \\ 1 & 0 & 1 \\ 1 & 0 & 1 \end{pmatrix}$$

- **Effect coding** compares each level to the overall mean of y. The first column is again the intercept. The weight β_0 which is associated to the intercept represents the overall mean and β_1, the weight for column two is the difference between the overall mean and category B. The overall effect of category B is $\beta_0 + \beta_1$. Interpretation for category C is equivalent. For the reference category A, $-(\beta_1 + \beta_2)$ is the difference of the category C to the overall mean and $\beta_0 - (\beta_1 + \beta_2)$ the overall effect of category C.

$$\begin{pmatrix} 1 & -1 & -1 \\ 1 & -1 & -1 \\ 1 & 1 & 0 \\ 1 & 1 & 0 \\ 1 & 0 & 1 \\ 1 & 0 & 1 \end{pmatrix}$$

- **Dummy coding** compares each level to the level mean of y. If all level are have the same frequency the resulting coefficients will be the same as in effect coding. Note that the intercept was dropped here.

$$\begin{pmatrix} 1 & 0 & 0 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 1 \end{pmatrix}$$

5.3.7 The disadvantages of linear models

They can only represent linear relationships as the name suggests. Each non-linearity or interaction has to be hand-crafted and explicitly given to the model as an input feature. Because of possible high correlation between features, it is possible that a feature that is positively correlated with the outcome might get a negative weight in a linear model, because in the high dimensional space it is negatively correlated. An example: You have a model to predict the rent price and have features like number of rooms and size of the flat. Of course flat size and room number are highly correlated, the bigger a flat the more rooms it has. If you now take both variables into a linear model it might happen, that the flat size is the better predictor and get's a large positive weight. The room number might end up getting a negative weight, because given that a flat has the same size, increasing the number of rooms could make it less valuable.

5.3.8 Towards complexer relationships within linear model class

- Adding interactions
- Adding non-linear terms like polynomials
- Stratifying data by variable and fitting linear models on subsets

---

5.4 Sparse linear models

The examples for the linear models that I chose look all nice and tidy, right? But in reality you might not have just a handful of features, but hundreds or thousands. And your normal linear models? Interpretability goes downriver. But there are ways to introduce sparsity (= only keeping a few features) into the linear models. The most automatic and convient to use is the LASSO method. LASSO stands for "least absolute shrinkage and selection operator" and when added to a linear model, it performs variable selection and regularization of the selected variables. We did not dive into the optimization problem of finding the best coefficients of the linear model. But basically it involves solving the least-squares equation:

$$min_{\beta_0,\beta}\left(\frac{1}{n}\sum_{i=1}^{n}(y_i-\beta_0-x_i^T\beta)^2\right)$$

LASSO adds a term to this optimization problem:

$$min_{\beta_0,\beta}\left(\frac{1}{n}\sum_{i=1}^{n}(y_i-\beta_0-x_i^T\beta)^2+\lambda||\beta||_1\right)$$

The term $||\beta||_1$ is the L1-norm of the feature vector, that leads to a penalization of large values in β. Since the L1-norm is used, many of the coefficients for the features will get an estimate of 0 and the others are shrinked. The weight λ says how strong the regularizing effect should be and is usually tuned by doing cross-validation. Especially when λ is large, many coefficients are driven to 0.

There are lots of other methods for reducing the number of features in your linear regression model:

Methods that include a pre-processing step: - Hand selected features: You can always use expert knowledge to choose and discard some features. The big drawback is, that it can't be automated and you might not be an expert. - Use some measure to pre-select features: An example is the correlation coefficient. You only take features into account that exceed some chosen threshold of correlation between the feature and the target. Disadvantage is that it only looks at the features one at a time. Some features might only show correlation after the linear model has accounted for some other features. Those you will miss with this approach.

Then there are also step-wise procedures: - Forward selection: Fit the linear model with one feature. Do that with each feature. Choose the model that works best (for example decided by R squared measurement). Now again, for the remaining features, fit different versions of your model by adding each feature. Pick the one that performs best again. Continue until

some criterium is reached, like the maximum number of features in the model. - Backward selection: Same as forward selection, but instead of adding features, start with the model with all features and try which feature removal brings the best performance increase until some stopping criterium is reached.

I recommend using LASSO. It also works for the logistic regression model for classification models, which is the topic of the following chapter.

5.5 Logistic regression: a linear model for classification

Logistic regression is the linear regression models counterpart for classification problems.

5.5.1 What's wrong with linear regression for classification?

The gaussian linear model works well in most regression setup, but fails in the classification case. Why is that? In case of two classes, you could label one of the classes with 0 and the other with a 1 and use a linear model on it and it would work. There are a few problems with that approach:

- A linear model does try to give you probabilities, but it treats the classes as numbers (0 and 1) and fits the best hyperplane (if you have one feature, it's a line) that minimizes the distances between the points and the hyperplane. So it simply interpolates between the points, but there is no meaning in it and you cannot interpret it as probabilities.
- Also a linear model will extrapolate the features and give you values below zero and above one, which are not meaningful and should tell you that there might be a more clever approach to doing classification.
- Since the predicted outcome is not a probability but some linear interpolation between points there is no meaningful threshold at which you can distinguish one class from the other. A good illustration of this issue was given on (Stackoverflow)[https://stats.stackexchange.com/questions/22381/why-not-approach-classification-through-regression], which I reproduced in Figure @ref{fig:linear-class-threshold}
- Linear models don't extend to classification problems with multiple classes. You would have to start giving labeling the next class with a 2, then 3 and so on. The classes might not have any order to them, but the linear model would force a weird structure on the relationship between the features and your class predictions. So for all features with a positive weight, the higher the features value the more they contribute to the prediction of a class with a higher number, even if the classes with similar numbers are not really related.

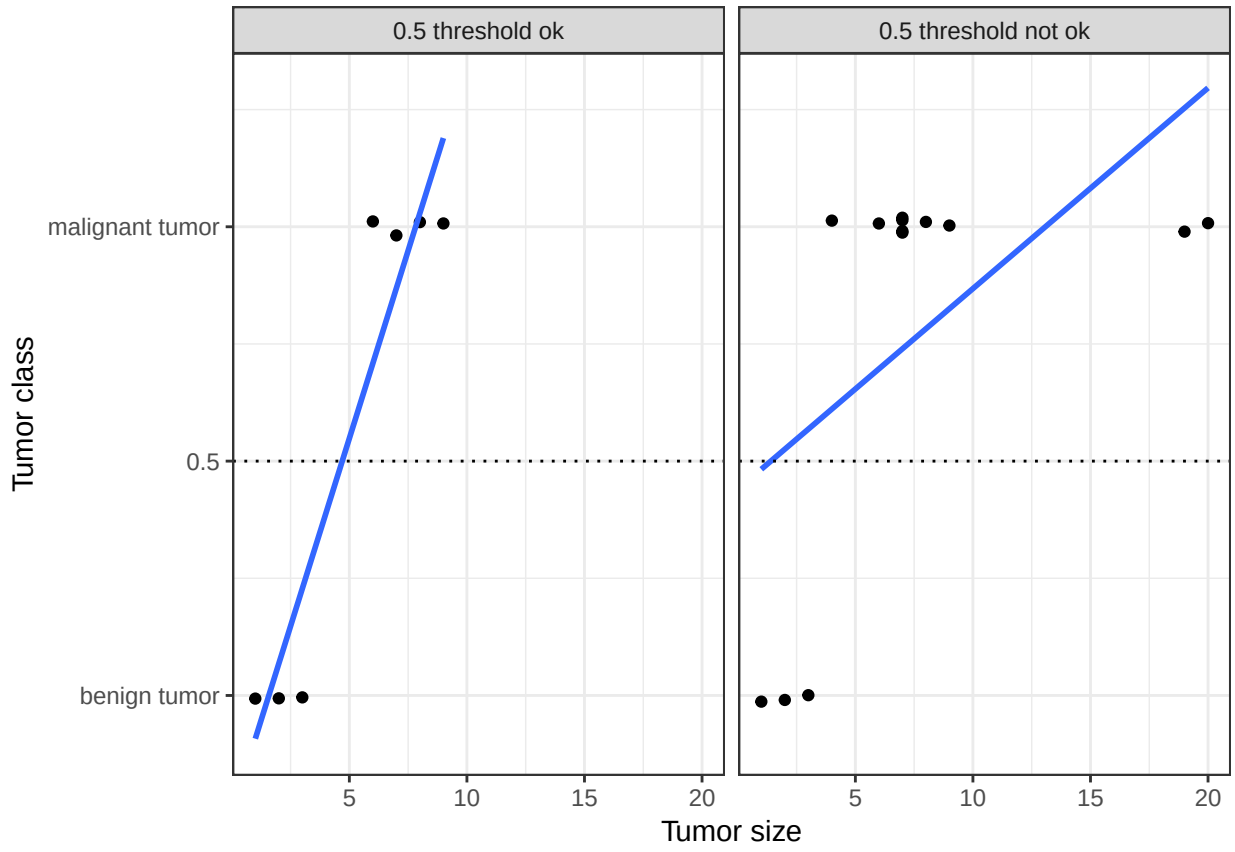


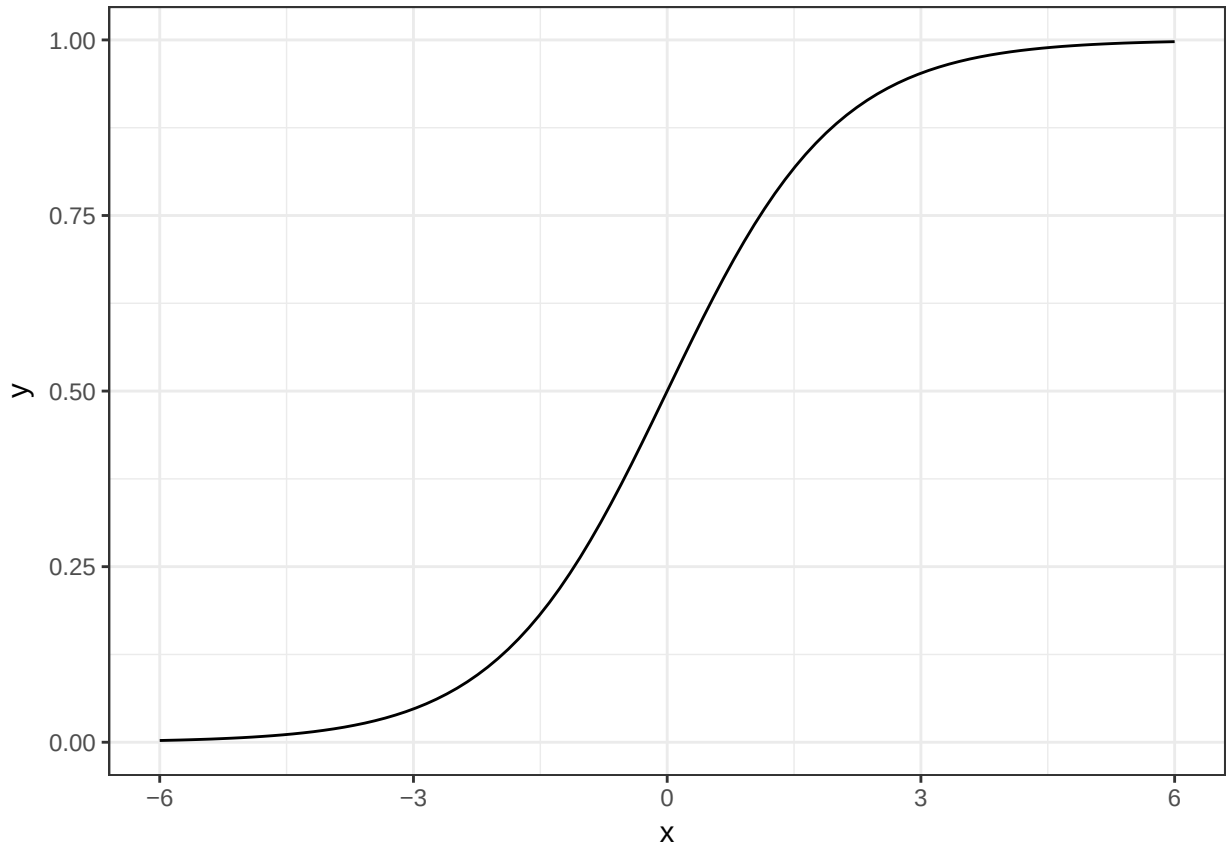
FIGURE 5.1
An illustration why linear regression does not work well in a binary classification setting. A linear model is fitted on the artificial task of classifying a tumor as malignant (1) or benign (0) depending on the tumor size. Each point is a tumor, the x-axis shows the size of the tumor, the y-axis the malignancy, points are slightly jittered to avoid overplotting of points. The lines display the fitted curve from the linear model. In the data setting on the left, we can use 0.5 as a threshold for the predicted outcome of the linear model for seperating benign from malignant tumors. After introducing a few more malignant tumor cases, especially one with a large tumor size the regression line shifts and a threshold of 0.5 would not separate the classes any longer. That's a reason why logistic regression is better suited for classification problems.

5.5.2 Logistic regression

The solution is logistic regression. Instead of fitting a straight line/hyperplane and uses a non-linear function, the logistic function to squeeze the output between 0 and 1. The logistic function is defined as

$$logistic(\eta) = \frac{1}{1 + exp(-\eta)}$$

And it looks like this:



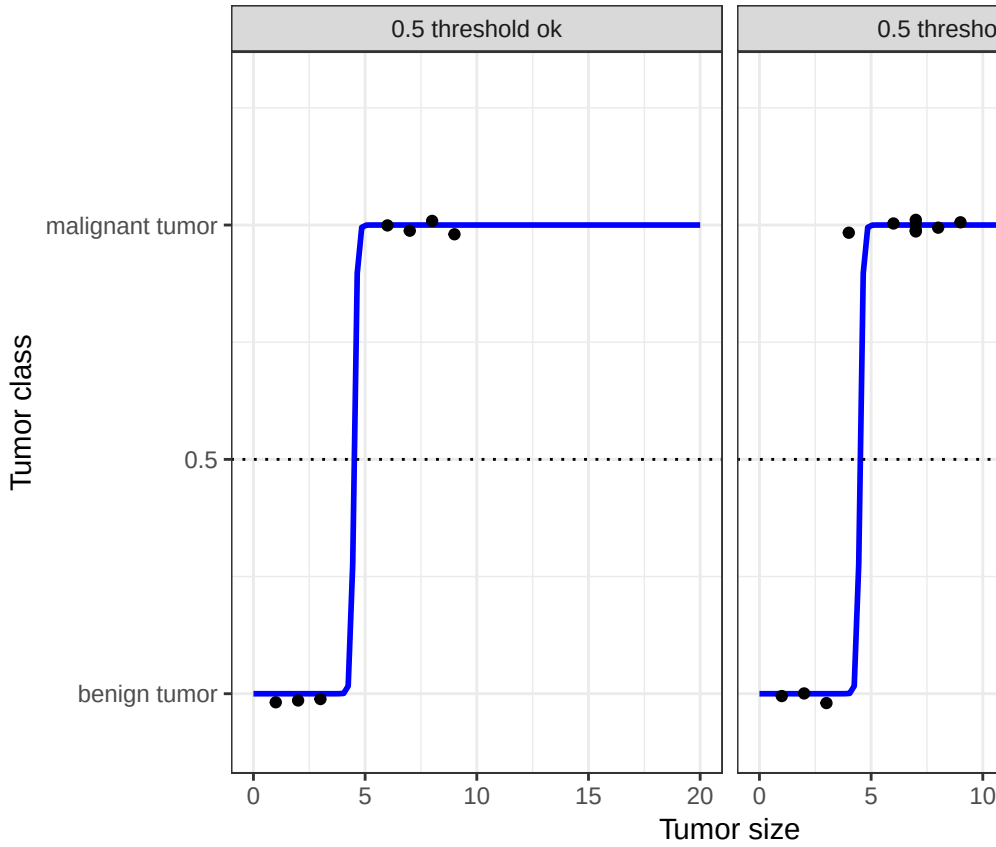
The step from linear regression models to logistic regression is kind of straight forward. Before we modeled the relationship like this:

$$\hat{y}_i = \beta_0 + \beta_1 \cdot x_{i,1} + \ldots + \beta_K x_{i,K}$$

Now we want probabilities, which are between 0 and 1, so we wrap the right side of the equation into the logistic regression function and simply force the output to be between 0 and 1:

$$P(y_i = 1) = \frac{1}{1 + exp(-(\beta_0 + \beta_1 \cdot x_{i,1} + \ldots + \beta_K x_{i,K}))}$$

Let's check the tumor size example again. But now instead of the linear regression model, we



use the logistic regression model:
It works better than with logistic regression and we can use 0.5 as a threshold. The line does not shift much, when including the additional datapoints.

5.5.3 Interpretation

The interpretation of the coefficients differs from linear regression models. Because now our target value is not some arbitrary number, but a probability between 0 and 1. Also through the logistic function, the influence of the features on the target probability has become non-linear. That's why we need to reformulate the equation for the interpretation>

$$log\left(\frac{P(y_i=1)}{(1-P(y_i=1))}\right) = log\left(\frac{P(y_i=1)}{P(y_i=0)}\right) = \beta_0 + \beta_1 \cdot x_{i,1} + \ldots + \beta_K x_{i,K}$$

$\frac{P(y_i=1)}{(1-P(y_i=1))}$ is also called odds (probability of event vs. probability of no event) and $log\left(\frac{P(y_i=1)}{(1-P(y_i=1))}\right)$ are the log odds. So with a logistic regression model we have a linear model for the log odds. Great! Doesn't sound helpful! Well, with a bit of shuffling again, you can find out how the prediction changes, when one of the features $x\cdot, k$ is changed by 1 point. For this we can first apply the $exp()$ function on both sides of the equation:

$$\frac{P(y_i=1)}{(1-P(y_i=1))} = odds_i = exp\left(\beta_0 + \beta_1 \cdot x_{i,1} + \ldots + \beta_K x_{i,K}\right)$$

Then we compare what happens when we increase one of the $x'_{i,j}s$ by 1. But instead of looking at the difference, we look at the ratio of the two predictions, you will see why:

$$\frac{odds_{i,x_i+1}}{odds_i} = \frac{exp\left(\beta_0 + \beta_1 \cdot x_{i,1} + \ldots + \beta_k \cdot (x_{i,k}+1) + \ldots + \beta_K x_{i,K}\right)}{exp\left(\beta_0 + \beta_1 \cdot x_{i,1} + \ldots + \beta_k \cdot x_{i,k} + \ldots + \beta_K x_{i,K}\right)}$$

Using the rule that $\frac{exp(a)}{exp(b)} = exp(a-b)$ gives us:
$$\frac{odds_{i,x_i+1}}{odds_i} = exp\left((\beta_0 + \beta_1 \cdot x_{i,1} + \ldots + \beta_k \cdot (x_{i,k}+1) + \ldots + \beta_K x_{i,K}) - (\beta_0 + \beta_1 \cdot x_{i,1} + \ldots + \beta_k \cdot x_{i,k} + \ldots +\right.$$

And then we can remove a lot of terms from the equation, which is convenient:
$$\frac{odds_{i,x_i+1}}{odds_i} = exp\left(\beta_k \cdot (x_{i,k}+1) - \beta_k \cdot x_{i,k}\right) = exp\left(\beta_k\right)$$

And we end up with something simple like $\exp(\beta_k)$. So a change in x_k by one unit changes the odds ratio (multiplicatively) by a factor of $\exp(\beta_k)$. We could also interpret it this way: A change in x_k by one unit change the log odds ratio by β_k units, but most people do the former because thinking in logs is known to be hard on the brain. Interpreting the odds ratio already needs a bit of getting used to. If you have odds of 2, it means that the probability for $y_i = 1$ is twice as big as $y_i = 0$. If you have a β (=odds ratio) of 0.7, then an increase in the respective x by one unit multiplies the odds by $\exp(0.7) \approx 2$ and your odds would be 4. But usually you don't deal with the odds and only interpret the β as the odds ratios. Because for actually calculating the odds you would need to set a value for each $x_{i,k}$ for all k, which only makes sense if you want to look at one specific instance of your dataset.

Here are the interpretations for the logistic regression model with different variable types:

- Continuous variable: For an increase of one unit of the variable x_j the estimated odds change (multiplicatively) by a factor of $\exp \beta_j$
- Binary categorical variables: One of the variables is the reference level (in some languages the one that was coded in 0). A change of the variable x_i the reference level to the other category changes the estimated odds change (multiplicatively) by a factor of $\exp \beta_j$
- Categorical variables with many levels: One solution to deal with many variables is to one-hot-encode them, meaning each level gets it's own column. From a categorical variable with L levels, you only need L-1 columsn, otherwise it is over parameterized. The interpretation for each level is then according to the binary variables. Some language like R allow to
- Intercept β_0: The interpretation is: Given all continuous variables are zero and the categorical variables are on the reference level, the estimated odds are is $\exp \beta_0$. The interpretation of β_0 is usually not relevant.

5.5.4 Example

With the logistic regression model we can predict cervical cancer given risk factors.

	Estimate	Odds ratio	Std. Error
Intercept	2.9101469	18.3594963	0.3225918
Hormonal contraceptives y/n	0.1166594	1.1237366	0.2989597
Smokes y/n	-0.2557759	0.7743154	0.3719329
Num. of pregnancies	-0.0368039	0.9638651	0.0965331
Num. of diagnosed STDs	-0.8154926	0.4424213	0.3260103
Intrauterine device y/n	-0.6163016	0.5399376	0.3995933

Interpretation of a numerical variable ('Num. of diagnosed STDs'): An increase of the number of diagnosed STDs changes (decreases) the odds for cancer vs. no cancer multiplicatively by 0.44, given all other features stay the same. Keep in mind that correlation does not imply causation. No recommendation here to get STDs.

Interpretation of a categorical variable ('Hormonal contraceptives y/n'): For women with hormonal contraceptives, the odds for cancer vs no cancer are by a factor of 1.12 higher, compared women without hormonal contraceptives, given all other features stay the same.

Again as in the linear models, the interpretations are always coming with the clause that 'all other features stay the same'.

5.6 Decision trees

Linear models fail in situation where the relationship is non-linear and/or where the features are interacting with each other. Time to shine for the decision trees! Tree-based models partition the data along the features into rectangles. For predicting the outcome in each rectangle it fits a simple model (for example the average of the outcome of the instances that fall into this rectangle). Trees have an intuitive structure starting from a root and splitting into nodes, according to cutoff values of the features. After each split, the instances fall into one of the new nodes. At the end of the training all the instances from the training data set are assigned into one of the leaf nodes. See Figure @ref{fig:tree-artificial} for illustration.

There are a lot of different tree algorithms. They differ in structure (number of splits per node), criteria for how to find the splits, when to stop splitting and how to estimate the simple models within the leaf nodes. Classification and regression trees (CART) is one of the more popular algorithms for tree building. This book will only talk about CART, because in the interpretation they are all the same. If you know of some tree algorithm with a different interpretation, I would welcome your feedback.

Each of these rectangles is associated with a simple model of the outcome of the interest. This is usually estimated by taking the mean of outcomes from all training instances that fall into a rectangle. I recommend the book 'The elements of statistical learning' (Hastie et al., 2009) for a more detailed introduction.

The following formula describes relationship of y and x (in which rectangle does x fall?)

$$\hat{y}_i = \hat{f}(x_i) = \sum_{m=1}^{M} c_m I\{x_i \in R_m\}$$

Each instance x_i falls into exactly one leaf node (=rectangle), so $I_{\{x_i \in R_m\}}$ is only 1 for the this single leaf node (I is the identity function which is 1 if $x_i \in R_m$ and else 0). If x_i falls into leaf node R_l, the predicted outcome $\hat{y} = c_l$, where c_l is the mean of all the training instances in leaf node R_l.

But where do the 'rectangles' come from? This is quite simple: The algorithm takes a feature

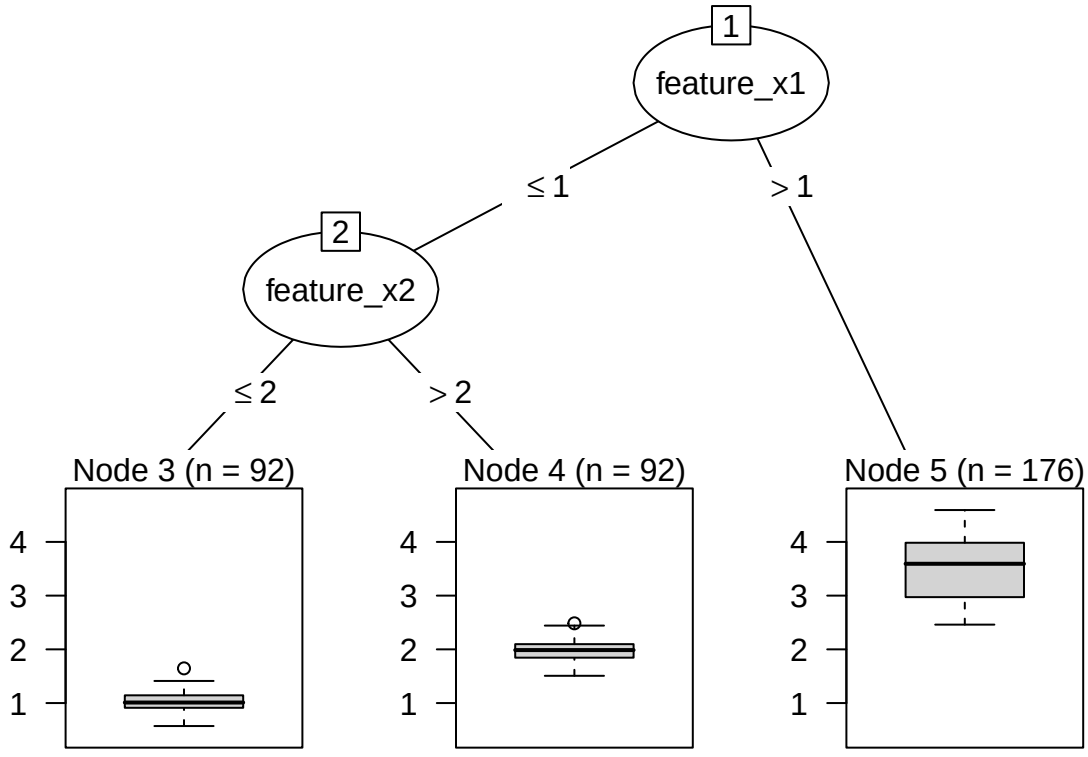


FIGURE 5.2
Exemplary decision tree with artificial data

and tries which cut-off point minimizes the sum of squares if it is a regression task or the Gini index in classification tasks. It's the cut-off point that makes the two resulting subsets as different as possible in terms of the outcome variable of interest. For categorial features the algorithm tries different groupings by category into to nodes. After this was done for each feature, the algorithm looks for the feature with the best cut-off and chooses this to split the node into two new nodes. The algorithm continues doing this in both new nodes until the stopping criteria is reached. Possible criteria are: A minimum number of observations that have to be in a node before the split, the minimum number of instances that have to be in a terminal node.

A common strategy is to grow a tree fully and then cut it back to optimize it's complexity measure *cp*.

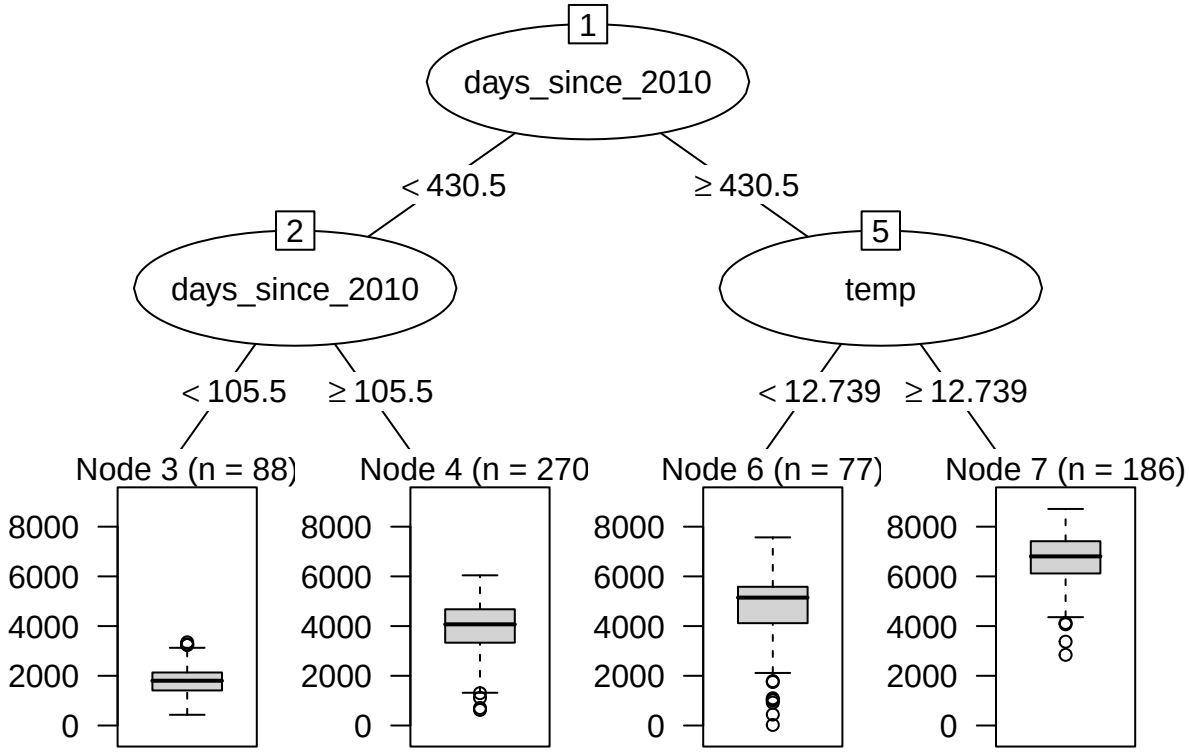
5.6.1 Interpretation

It's easy: Starting from the root node you go to the next nodes and the edges tell you which subsets you are looking at. Once you reach the leaf node, the node tells you the predicted outcome. All the edges are connected by 'AND'.

Template: If feature x is [smaller/bigger] than threshold c AND ..., then the predicted value is $\hat{y}$.

5.6.2 Interpretation example

Let's have a look again at the speed dating example. Again we want to predict the rating from the participants, how much they will like the rating partners.



The first split was done in the workingday variable, which tells if a day is a working day or a saturday/sunday/holiday. On days without work, the number of rental bikes was higher on average. In both child nodes the the next feature that was chosen was temperature.

In waves of size 20 or smaller, participants who gave 5/10 or more to importance of same religion of partner, they also rated lower on average (median around 6). If religion was less important (4 or lower), than they gave higher ratings.

5.6.3 Advantages

The tree structure is perfectly suited to **cover interactions** between features in the data. The data also ends up in **distinct groups**, which are easier to grasp than points on a hyperplane like in linear regression. The interpretation is arguably pretty straight forward. The tree structure also has a **natural visualization**, with it's nodes and edges.

5.6.4 Disadvantages

Handling of real linear relationships, that's what trees suck at. Any real linear relationship between an input feature and the outcome has to be approximated by hard splits, which produces a step function. This is not efficient. This goes hand in hand with **lack of smoothness**. Slight changes in the input feature can have a big impact on the predicted

outcome, which might not be desirable. Imagine a tree that predicts the worth of a house and the tree splits in the square meters multiple times. One of the splits is at 100.5 square meters. When a user measure his house and arrives at 99 square meters, types it into some nice web interface and get's 200 000 Euro. The user notices that she forgot to measure a small storeroom with 2 square meters. The storeroom has a skewed wall, so she is not sure if she can count it fully towards the whole flat area or only half of the space. So she decides to try both 100.0 and 101.0 square meters. The results: 200 000 Euro and 205 000 Euro, which is quite unintuitive.

Trees are also quite **unstable**, so a few changes in the training data set might create a completely different tree. That's because each splits depends on the parent split. It does not generate trust if the structure flips so easily.

5.7 Other simple, interpretable models

5.7.1 Naive bayes classifier

The naive Bayes classifier make use of the Bayes'theorem. For each feature it computes the probability for a class given the features value. The clue is that naive Bayes does so for each feature independently, which is the same as having a strong (=naive) assumption of independence of the features. Naive Bayes is a conditional probability model and models the probability of a class k in the following way:

$$P(C_k|x) = \frac{1}{Z}P(C_k)\prod_{i=1}^{n}P(x_i|C_k)$$

The term Z is a scaling parameter that ensures that the probabilities for all classes sum up to 1.

Naive Bayes is an interpretable model, because of the independence assumption. For each classification it is very clear for each feature how much it contributes towards a certain class prediction.

5.7.2 k-nearest neighbours

k-nearest neighbour can be used for regression and classification and uses the closest neighbours for a data point for prediction. For classification it assigns the class most common the closest k neighbours of an instance and for regression it takes the average of the outcome. The tricky parts are finding the right k and defining the neighbourhood. This algorithm is different from the other intepretable models presented in this book, since it is an instance-based learning algorithm. How is k-nearest neighbour interpretable? For starters, there is no global model interpretability, since the model is inherently local and there are no global weights or structures that are learned explicitly by the k-nearest neighbour method. Maybe

it is interpretable on a local level? To explain a prediction, you can always retrieve the k-neighbours that were used. If this is interpretable solely depends if you can 'interpret' single instances in the dataset. If the dataset consists of hundreds or thousands of features, then it is not interpretable I'd argue. But if you have few features or a way to reduce your instance to the most important features, presenting the k-nearest neighbours can give you good explanations.

5.7.3 RuleFit

The RuleFit algorithm from Friedman and Propescu (Friedman and Popescu, 2008) is a regression and classification approach that uses decision rules in a linear model. RuleFit consists of two components: The first component produces "rules" and the second component fits a linear model with these rules as input (hence the name "RuleFit"). It enables automatic integration of interactions between features into a linear model, while having the interpretability of a sparse linear model.

There are two steps involved:

Step 1: Rule generation:

The rules that the algorithm generates have a simple form:

if $x2 < 3$ and $x5 < 7$ then 1 else 0

The rules are generated from the covariates matrix X. You can also see the rules simply as new features based on your original features.

The RuleFit paper uses the Boston housing data as example: The goal is to predict the median house value in the Boston neighborhood. One of the rules that is generated by RuleFit is:

if (number of rooms > 6.64) and (concentration of nitric oxide < 0.67) then 1 else 0

The interesting part is how those rules are generated: They are derived from Decision Trees, by basically disassembling them. Every path in a tree can be turned into a decision rule. You simply chain the binary decisions that lead to a certain node, et voilà, you have a rule. It is desirable to generate a lot of diverse and meaningful rules. Gradient boosting is used to fit an ensemble of decision trees (by regressing/classifying y with your original features X). Each resulting trees is turned into multiple rules.

Another way to see this step is a black box, that generates a new set of features X' out of your original features X. Those features are binary and can represent quite complex interactions of your original X. The rules are chosen to maximise the prediction/classification task at hand.

Step 2: Sparse linear model

You will get A LOT of rules from the first step (and that is what you want). Since the first step is only a feature transformation function on your original data set you are still not done with fitting a model and also you want to reduce the number of rules. Lasso or L1 regularised

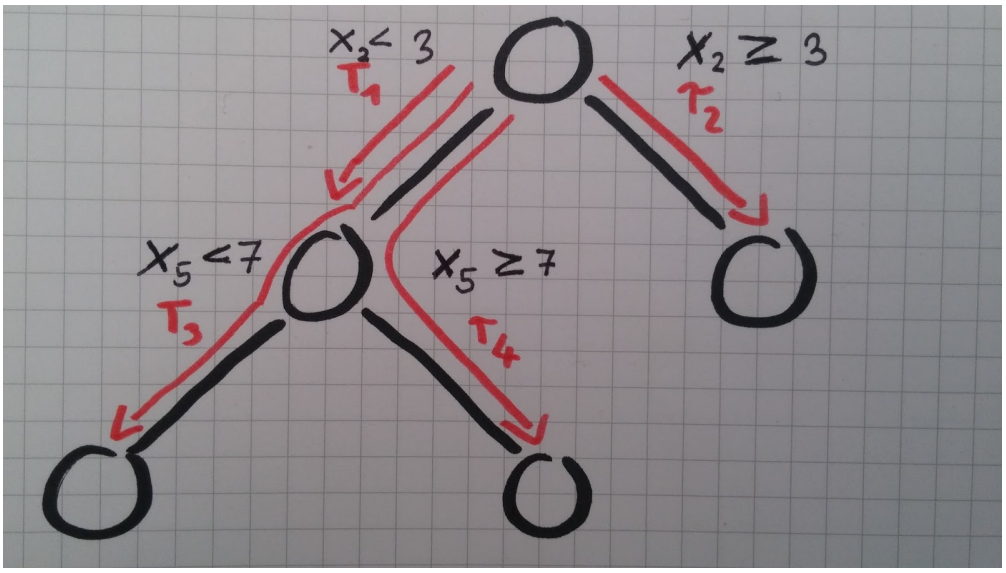


FIGURE 5.3
4 rules can be generated from a tree with 3 terminal nodes.

regression is good in this scenario. Next to the rules also all numeric variables from your original data set will be used in the Lasso linear model. Every rule and numeric variable gets a coefficient (beta). And thanks to the regularisation, a lot of those betas will be estimated to zero. The numeric variables are added because trees suck at representing simple linear relationships between y and x. The outcome is a linear model that has linear effects for all of the numeric variables and also linear terms for the rules.

The interpretation is the same as with linear models, the only difference is that some features are now binary rules.

The paper not only introduces RuleFit and evaluates it, but it also comes with a bunch of useful tools, (comparable to Random Forest): Measurement tools for variable importance, degree of relevance of original input variables and interaction effects between variables.

5.7.4 And so many more ...

There are lots and lots of algorithms that produce interpretable models and not all will be listed here. If you are a researcher or just a big fan and user of a certain interpreable method that is not listed here, get in touch with me and add the method to this book!

6

Model-agnostic tools for interpretability

Separating the explanations from the machine learning model (= model-agnostic explanations) gives some benefits. The big advantage of model-agnostic vs model-specific explanation algorithm is the flexibility. When the explanation system is independently applicable even when the underlying model is switched, it frees the practitioner to use different machine learning models without restrictions. It is also more efficient to build interfaces on top of model-agnostic systems, because this has to be done only once and not for each model-specific explanation system. Usually not one but many types of machine learning models are tested in development time and if you want to compare the models in terms of interpretability this is easier with model-agnostic explanations because the system is the same for both models that are being compared (Ribeiro et al., 2016).

The alternatives are either using only interpretable models as introduced in Chapter 2, which has the big disadvantage to usually loose accuracy compared to other approaches. The other alternative is to use more flexible model classes that come with built in explanations. The drawback here is that it ties you to this one algorithm and it will be hard to switch to something else.

Desirable aspects of a model-agnostic explanation system (Ribeiro et al., 2016): - Model flexibility: Not being tied to an underlying particular machine learning model. The method should work for random forests as well as convolutional neural networks - Explanation flexibility: Not being tied to a certain form of explanation. In some cases it might be useful to have a linear formula in other cases some decision rules - Representation flexibility: The explanation system should not have to use the same feature representation as the model that is being explained. So when a text classifier uses abstract word embedding vectors, it might be preferable to use the presence of single words for the explanation.

6.1 Partial dependence plot

The partial dependence plot shows the marginal effect of a variable on the target (regression / classification) (Friedman, 2001). A partial dependence plot can show if the relationship between target and feature is linear, monotonic or something else. In linear regression, those plots will always show a linear relationship.

The partial dependence function for regression is defined as:

$$f_S = E_{x_C}[f(x_S, x_C)] = \int f(x_S, x_C) dP(x_C)$$

The x_S is the set of variables for which the partial dependence should be depicted and x_C are the other variables that were used in the machine learning model. Partial dependence works by averaging out the other variables, so that the remaining function shows the relationsship between the x_S, in which we are interested, and the target. x_S is fixed and x_C is varying.

The integral is estimated by calculating averages in the training data, which looks like this for regression:

$$\hat{f}(x_S) = \frac{1}{n}\sum_{i=1}^{n} f(x_S, x_{Ci})$$

In this formula, x is the variable for which to calculate the partial dependence, x_{iC} is the other variables and n the number of instances in the data set.

For classification it is the logits:

$$f(x) = \log p_k(x) - \frac{1}{K}\sum_{j=1}^{K} \log p_j(x)$$

Partial dependence plots are only partially global: They are global because they take into account all instances, but it is local in the feature, because partial dependence plots only examine one variable, as the name suggests.

6.1.0.1 Examples

In practice x_S usually only contains one variable or a maximum of two, because one variable produces 2D plots and two variables produce 3D plots. Everything beyond that is quite tricky. Even 3D on a 2D paper or monitor is already challenging. This example here shows an artificial dataset with two x variables on which a Random Forest was trained.

Let's turn to the regression example with the bike counts again and have a look at how the weather effects look like. @ref{fig:dpd-bike} shows the average influence of the weather features on the predicted bike counts. Warm, but not too hot weather makes the model predict a high number of bikes rentals. The potential bikers are increasingly inhibited in engaging in cycling when humidity reaches above 60%. Also the more wind the less people like to bike, which personally I can understand. Interestingly the predicted bike counts don't drop between 25 and 35 km/h, but maybe there is just not enough training data. At least intuitively I would expect the bike rentals to drop with each increase in windspeed, especially when the windspeed is very high.

6.2 Individual Conditional Expectation (ICE) plot

The partial dependence plot is for visualizing the averaged effect of a feature is a global method, because it does not focus on the partial dependence of a specific instance, but on an average over all. The equivalent to a PDP for local expectations is called individiual conditional expectation (ICE) plot (Goldstein et al., 2015). An ICE plot visualizes the

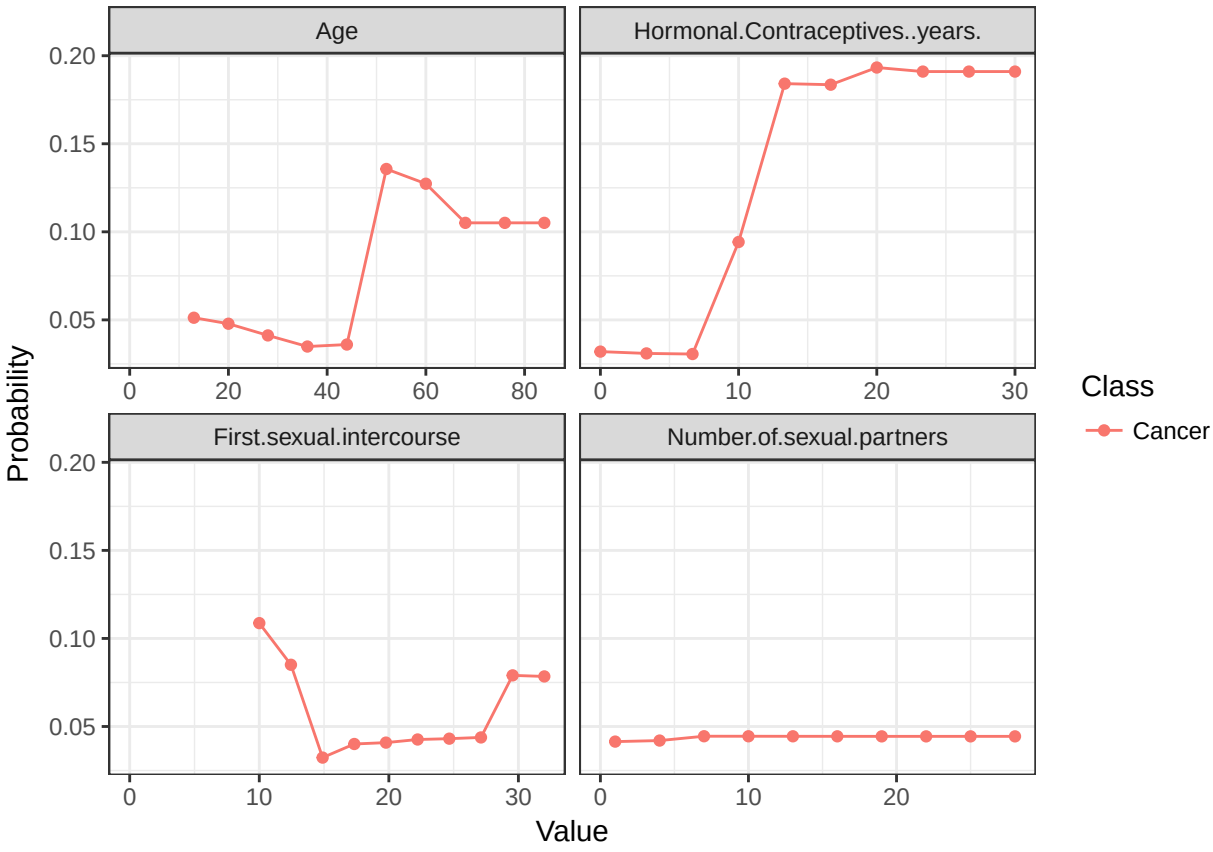


FIGURE 6.1
Partial dependence plot of cancer probability and different factors. For the age feature, the models partial dependence shows that on average, the cancer probability is low before 45, spikes between age 45 and 55 and plateaus after that.

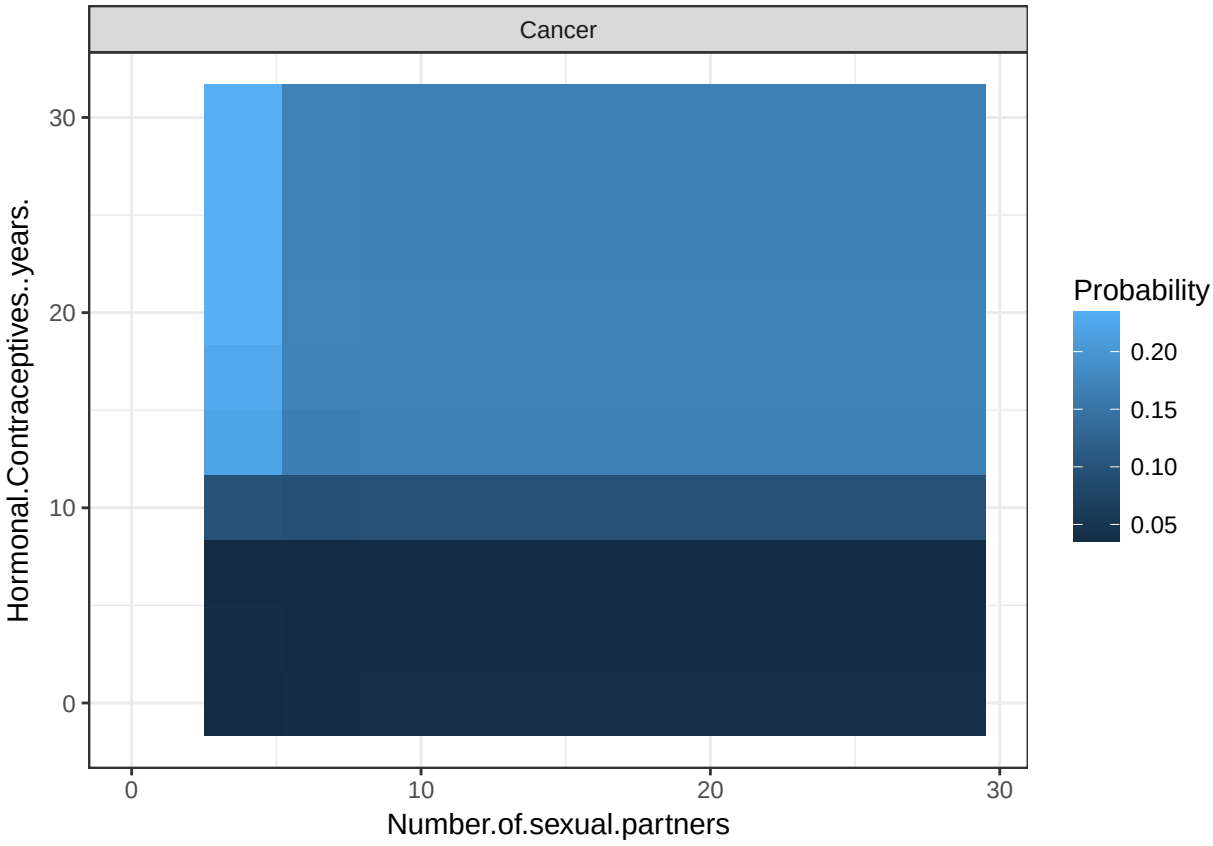


FIGURE 6.2
Partial dependence plot of cancer probability and the interaction of number of years on hormonal contraceptives and number of sexual partners. Interestingly, there is some odd interaction between the two variables when the number of sexual partners is 1 and the years of on hormonal contraceptives larger than 12. There are actually only two women in that group, who both happen to have cancer. So my best guess is that this was random and the model did overfit on those two women, but only more data could solve this question.

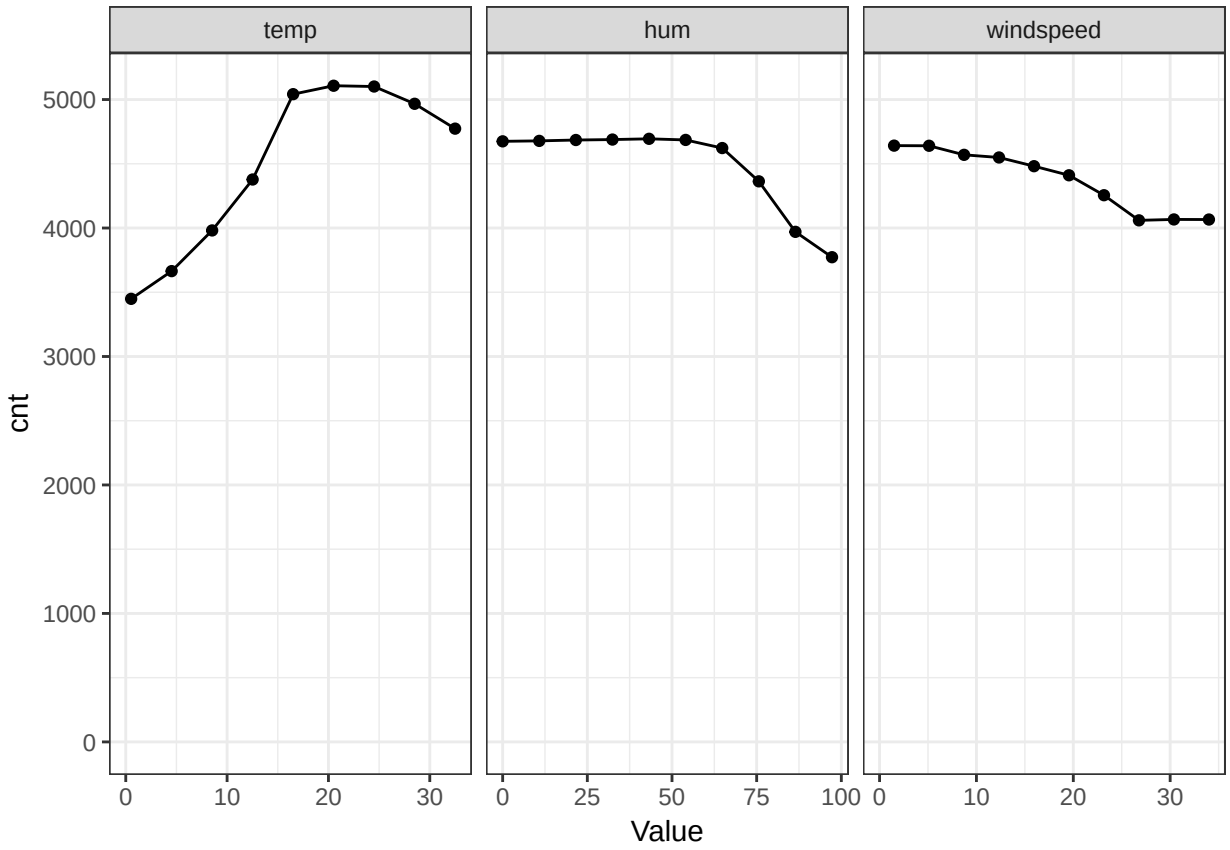


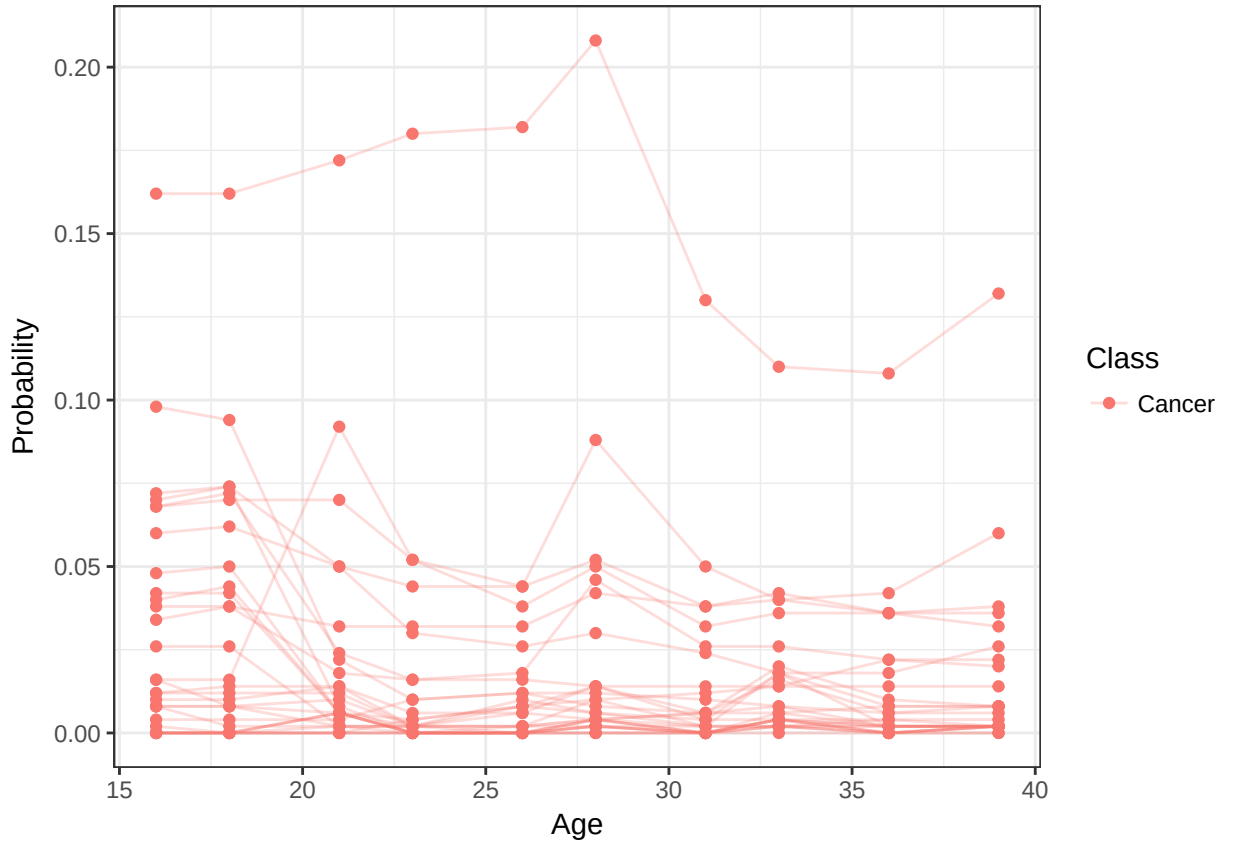
FIGURE 6.3
Partial dependence plot of rental bike count and different weather measurements (Temperature, Humidity, Windspeed). The biggest differences can be seen in different temperatures: With rising temperatures, on average the bike rentals rise, until 20C degrees, where it stays the same also for hotter temperatures and drops a bit again towards 30C degrees.

dependence of each instance's predicted response on a feature. They are event simpler than PDPs, since no averaging is needed. Instead of drawing one line for a feature, each instance in the dataset gets it's own line. The values for a line can be computed easily, by leaving all other features the same, but creating variants of the instance of interest and letting the black box make the predictions or classifications. The result is a set of points for a varying feature, for one specific instance. The lines for the instances can look quite differently (if the black box allows interactions between features), because the course of the line depends on the specific values of each instance. For drawing each line, the x_C are fixed for this one instance, and the x_C is varied on a grid and the $\hat{(}y)$ calculated with $\hat{(}f)$.

So, what do you gain by looking at individual expectations, instead of partial dependencies? This averaged display can obfuscate a heterogeneous relationship that comes from interactions. ICE plots (Goldstein et al., 2015) solve this problem by plotting the relationship between feature and predicted response for individual observations. It can be seen as an extension to the standard PDP. PDP can show you how the average relationship between feature x_S and $\hat{(}y)$ looks like. This works only well in cases where the interactions between x_S and the remaining x_C are weak. If there are interactions, a ICE plot will give a lot more insight.

A more formal definition: In ICE plots, for each observation in $\{(x_{S_i}, x_{C_i})\}_{i=1}^{N}$ the curve $\hat{f}_S^{(i)}$ is plotted against x_{S_i}, while x_{C_i} is kept fixed. #### Example Let's go back to the dataset about risk factors for cervical cancers and see how each instance's prediction is associated with the feature 'Age'. In the partial dependence plot chapter {?} we have seen that the probability increases around the age of 50, but does this hold true for each woman in the dataset? The ICE plot reveals that the most women's predicted probability follows the average pattern of increase at 50, but there are a few exceptions: In a few cases, the prediction of cancer probability does

not change much with the age, and that is for women that have a high predicted probability.



6.2.0.1 Centered ICE plot

There is one issue with the ICE plot: It can be hard to see if the individual conditionl expectations curve differ between individuals when they start at different $\hat{f}^{(i)}$. An easy fix is to center the curves at a certain point in x_S and only display the difference in predited response. The resulting plot is called centered ICE plot (c-ICE). It is a kind of anchoring, and doing this at the lower end of x_S is a good choice. The new curves are defined as:

$$\hat{f}_{cent}^{(i)} = \hat{f}_i - 1\hat{f}(x^*, x_{C_i}),$$

where 1 is a vector of 1's with the appropriate dimensions (usually one- or two-dimensional), and $\hat{f}$ the fitted model.

6.2.0.2 Example

Taking Figure @ref{fig:ice-cervical} and centering the lines at the youngest observed age yields Figure @ref{fig:ice-cervical-centered}. It is easier to see now, how the relative change of the curves from the youngest age is. This can be useful when we are not interested in seing the absolute change of a predicted value, but

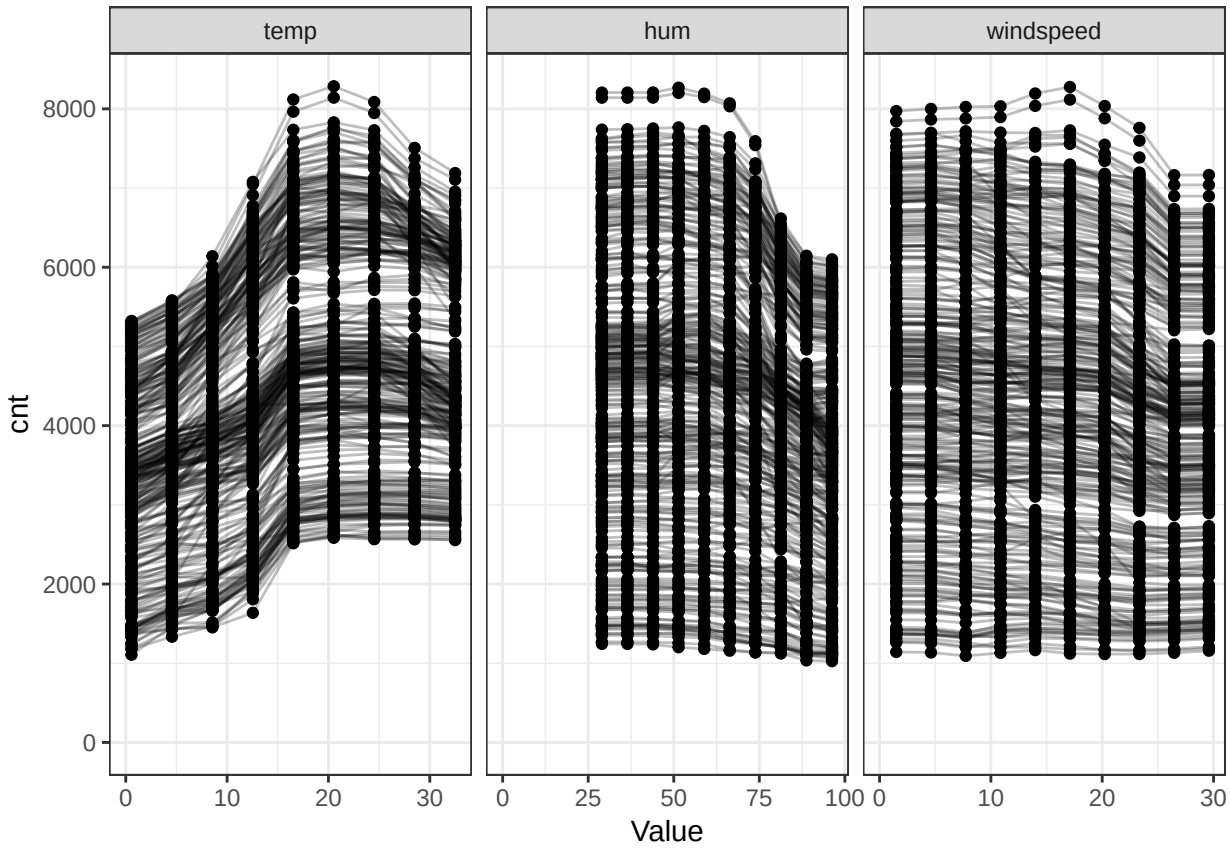
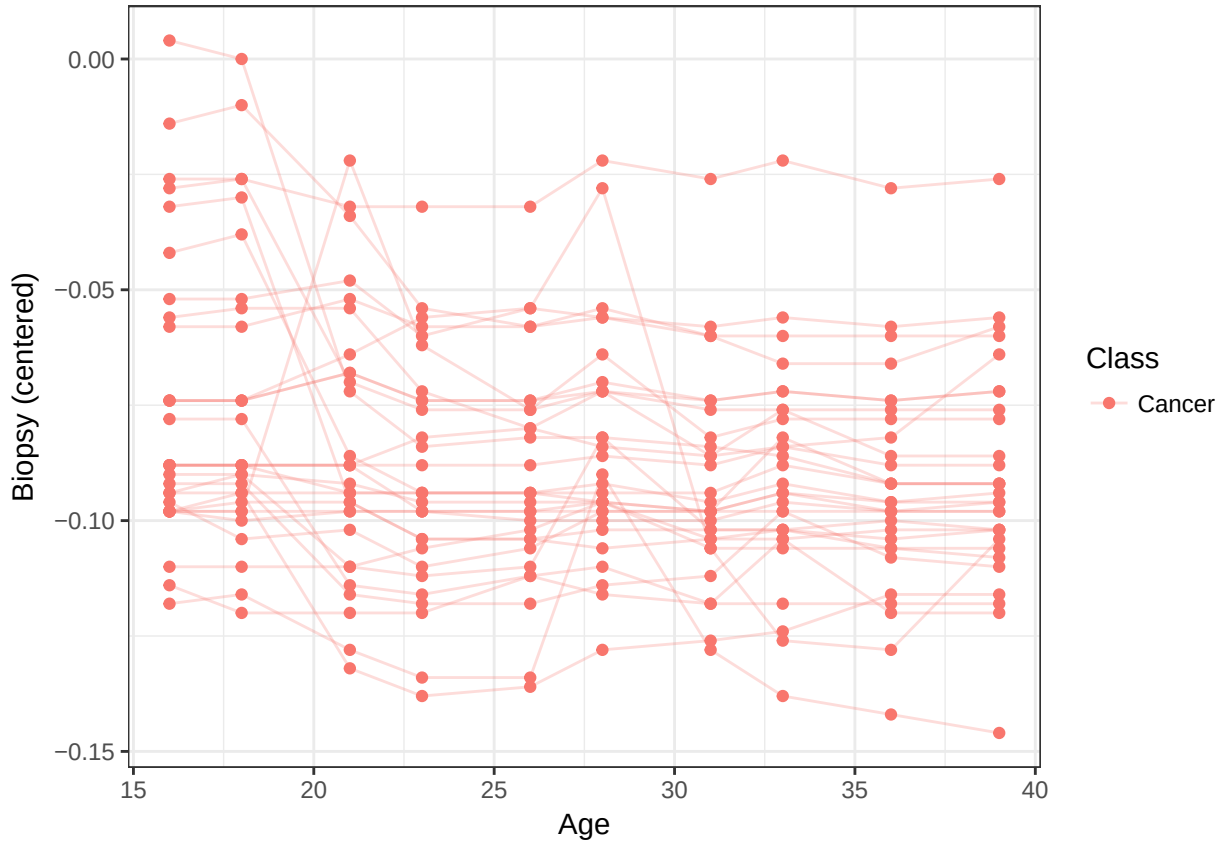
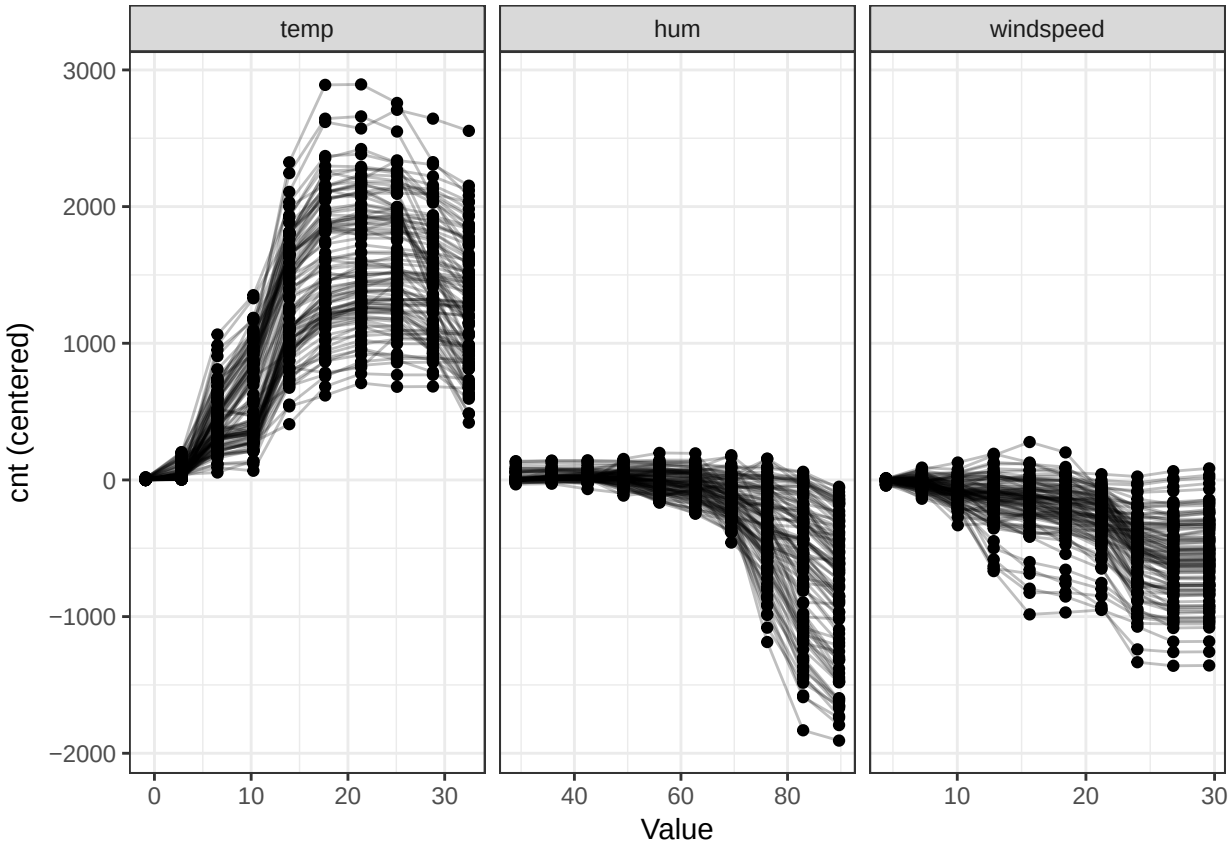


FIGURE 6.4
Individual conditional expectation plot of expected bike rentals and weather conditions . Same effects as in the partial dependence plot can be seen.

rather the difference in prediction compared to a fixed point of the feature range.





Derivative ICE plot Another way to make it it visually easier to spot heterogenity is to look at the individual derivatives of $\hat{(}f)$ with respect to x_S instead of the predicted response $\hat{(}f)$. The resulting plot is called derivative ICE plot (d-ICE). The derivatives of a function (or curve) tells you in which direction changes occur and if any occur at all. With the derivative ICE plot it is easy to spot value ranges in a feature where the black box's predicted value changes for (at least some) instances. If there is no interaction between x_S and x_C, then $\hat{f}$ can be expressed as:

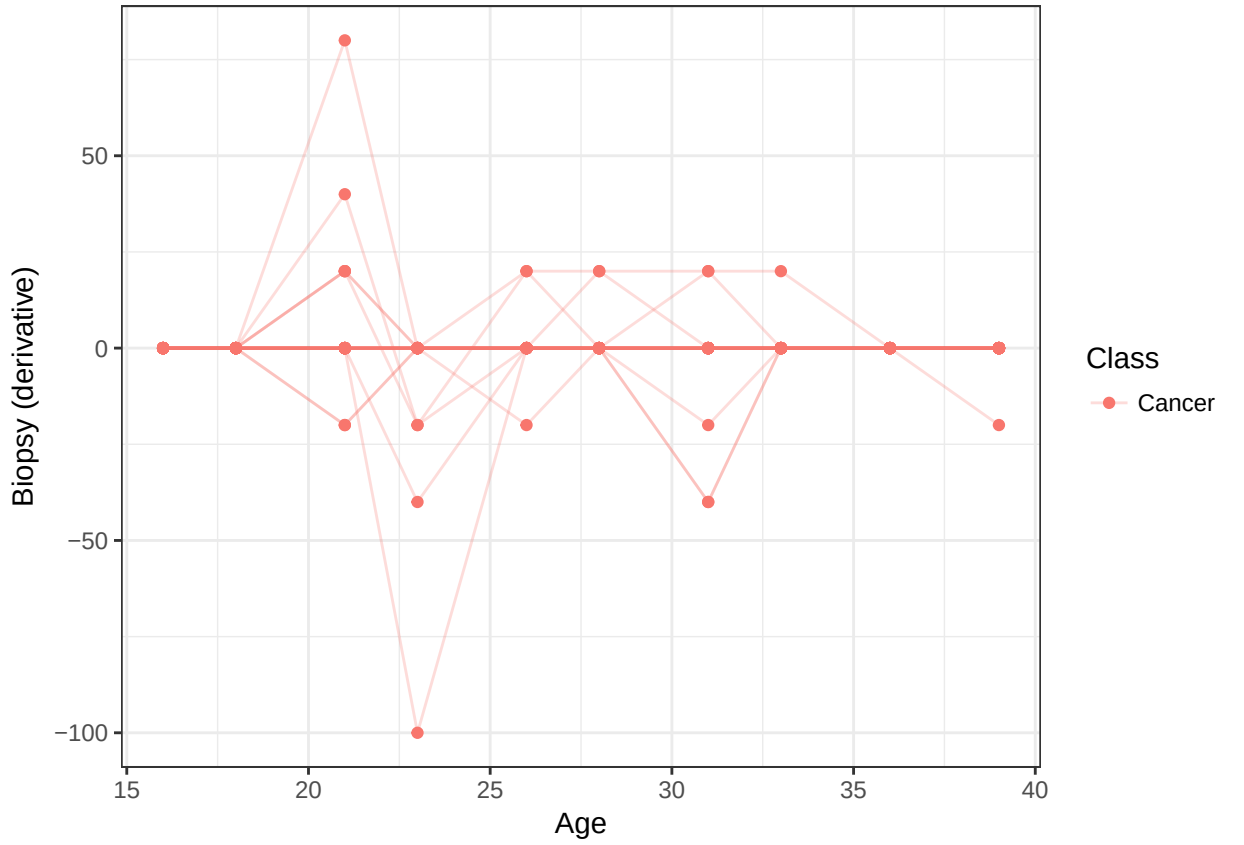
$$\hat{f}(x) = \hat{(}f)(x_S, x_C) = g(x_S) + h(x_C), \text{ so that } \frac{\delta \hat{f}(x)}{\delta x_S} = g'(x_S)$$

Without interactions, the individual partial derivatives should be the same for all observations. If they differ, it's because of interactions and it will become visible in the d-ICE plot. In addition to displaying the individual curves for derivative $\hat{f}$, showing the standard deviation of derivative $\hat{f}$ helps to highlight regions in x_S with heterogeneity in the estimated derivatives.

(Goldstein et al., 2015)

6.2.0.3 Example

As we have seen, the most changes in estimated cancer probability happen around age 45. This is confirmed by the derivative ICE plot in Figure @ref{fig:ice-cervical-derivative}.



6.3 Permutation feature importance

The permutation importance measurement was orginially introduced for RandomForests (Breiman, 2001). It is calculated on the out-of-bag instances and works by estimating the original model performance and checking what happens with the model performance when you permute each feature. A big loss in performance means a big feature importance. The idea of permutation of features is per se model-agnostic, only the OOB-scheme is specific for ensemble methods. In can be used for any model when a hold-out dataset is used, instead of OOB samples. Of course you could also use the training data, but you risk getting variable importance measures that overfit your training data, since the model was already trained on it.

Algorithm (Breiman, 2001):

Input: Trained model $\hat{f}$, hold-out dataset D, number of permutations n_{perm}

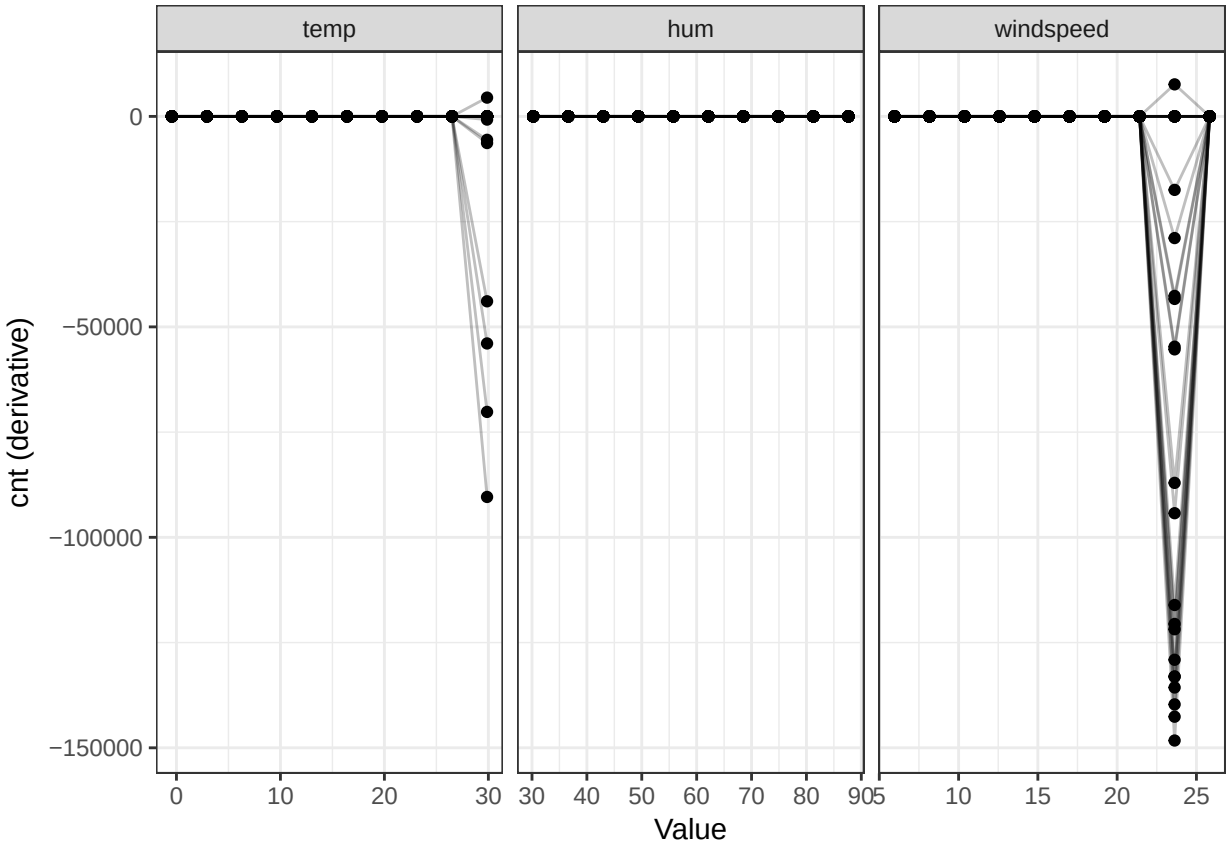


FIGURE 6.5
Derivative individual conditional expectation plot of expected bike rentals and weather conditions.

1. Estimate performance $Perf$ of $\hat{f}$ with D (e.g. MSE for regression or accuracy for classification)
2. For each feature $j \in 1, \ldots, J$ do:

- For $i \in 1, \ldots, n_{perm}$
  - Get $D_{j_{perm}}$ by permuting feature X_j in data D. This breaks the association between X_j and Y.
  - Estimate performance $Perf_{i,j_{perm}}$ of $\hat{f}$ with $D_{j_{perm}}$
  - Calculate permutation variable importance $VI_i(X_j) = Perf_{i,j_{perm}} - Perf$
- Calculate mean variable importance: $VI(X_j) = \frac{1}{n_{perm}} \sum_{i=1}^{n_{perm}} VI_i(X_j)$
- Optional: Calculate p-value $p = \frac{I(Perf_{j_{perm}} > Perf)}{n_{perm}}$

3. Sort variables by descending VI.

The feature with the highest VI measure is the most important globally in your model. With the p-value you can additionally check if a feature importance is significantly different from 0. You might want to adjust your α confidence level for multiple testing.

You can also find the algorithm in more detail in (Strobl et al., 2008). The authors additionally suggest a conditional feature importance measurement, which is not (yet) covered in this book. The standard permutation feature importance only works with marginal feature improvements and cannot distinguish between correlation and spurious correlation. (Strobl et al., 2008) suggest to condition the importance measure also on other features, which makes it possible to account for correlation among the features.

6.3.1 Model dependent feature importance

Some model classes already come with built in feature importance measurements. A few examples: - RandomForest: Permutation based feature importance - CART and boosting: mean decrease in Gini impurity index - Linear Model: (absolute value of) t-test statistic for each feature

6.4 Local surrogate models (LIME)

Local interpretable model-agnostic explanations (LIME) is a method for fitting local, interpretable models that can explain single predictions or classifications of any black-box machine learning model. LIME explanations are local surrogate models. Instead of trying to fit a global surrogate model, LIME focuses on a prediction done by a black-box algorithm and explains it's outcome.

The idea is quite simple, really. First of all, forget about the training data and imagine you only have the black box model where you can input data points and get the models outcome.

You can probe the box as often as you want. Your goal is to understand why the machine learning model gave the outcome it produced. LIME tests out what happens to the model's predictions when you put some variations of your data point of interest into the machine learning model. This basically generates a new dataset consisting of the perturbed samples and the associated model's outcome. On this dataset LIME then trains a simple model weighted by the proximity of the sampled instances to the instance of interest. The simple mode can basically be any from Section [#simple], for example LASSO or a short tree. The learned model should be a good approximation of the machine learning model locally, but it does not have to be so globally. This kind of accuracy is also called local fidelity.

The recipe:

- Choose your instance of interest for which you want to have an explanation of it's black box outcome
- Make some variations of the instances and check what the black box predicts in the neighbourhood of the instance of interest.
- Fit a local, interpretable model on the dataset with the variations
- Explain prediction by interpreting the local simple model.

In the current implementation, only LASSO can be chosen as a simple model. Upfront you have to choose K, the number of features that you want to have in your simple model. The lower the K, the simpler the model is to understand, higher K potentially creates models with higher fidelity. There are different methods for how to fit models with exactly K features. The most natural with LASSO is the lasso path. Starting from a model with a very high regularisation parameter λ yields a model with only the intercept. By refitting the LASSO models with slowly decreasing λ one after each other the features are getting weight estimates different from zero. When K features are in the model, you reached the desired number of features. Other strategies are forward or backward selection of features. This means you either start with the full model (=containing all features) or with a model with only the intercept and then testing which feature would create the biggest improvement when added or removed, until a model with K features are reached. Other simple models like decision trees are currently not implemented.

As always, the devil is in the details. In a high-dimensional space, defining a neighbourhood is not trivial. Distance measures are quite arbitrary and distances in different dimensions (aka features) might not be comparable at all. How big should the neighbourhood be you look into? If it is too small, then there might be no difference in the predictions of the machine learning model at all. The other question is: How do you get the variations of the data? This differs depending on the type of data, which can be either text, an image or tabular data. For text and image the solution is turning off and on single words or superpixels. In the case of tabular data LIME creates new samples by pertubing each feature individually, by drawing from a normal distribution with mean and standard deviation from the feature.

6.4.0.1 LIME for tabular data

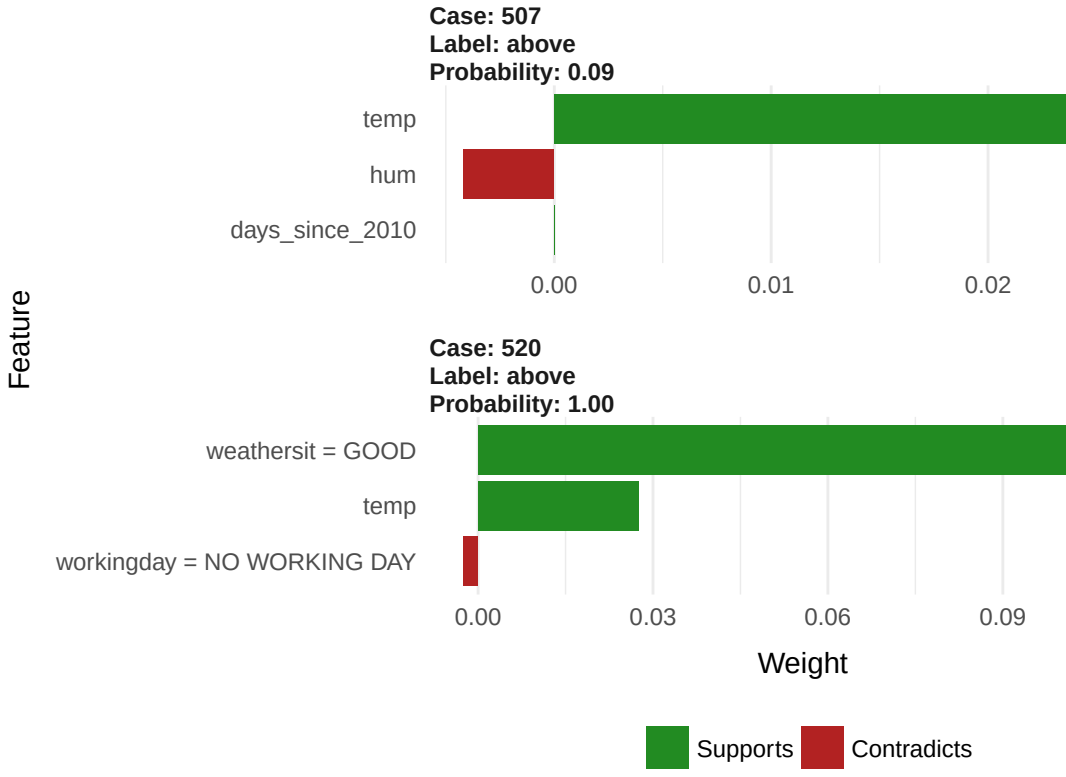
Tabular data means any data that comes in tables, where each row represents an instance and each column a feature. Sampling is not done around the point, but from the training data's mass center. Has it's problems. But it increases the likelihood that the outcome for some of the sampled points predictions differ from the data point of interest and that LIME can learn at least some explanation.

Figure 6.6 explains how the sampling and local model fitting works.

6.4.0.2 Example

Let's look at a concrete example. We go back to the bike rental and turn the prediction problem into a classification: After accounting for the trend that the bike rental get's more popular over time we want to know on a given day if the number of rented bikes will be above or below the trend line. You can also interpret 'above' as being above the mean bike counts, but adjusted for the trend.

First we train a Random Forest on the classification task. Given seasonal and wheather information, on which day will the number of bike rentals be above the trend-free average? The Random Forest has 100 trees.



The continuous features are categorised into bins by quantiles for the explanation models. The explanations are set to contain 3 features. Figure @ref{fig:lime-tabular-example-explain-plot-1} shows the results of the sparse local linear model that was fitted for two instances

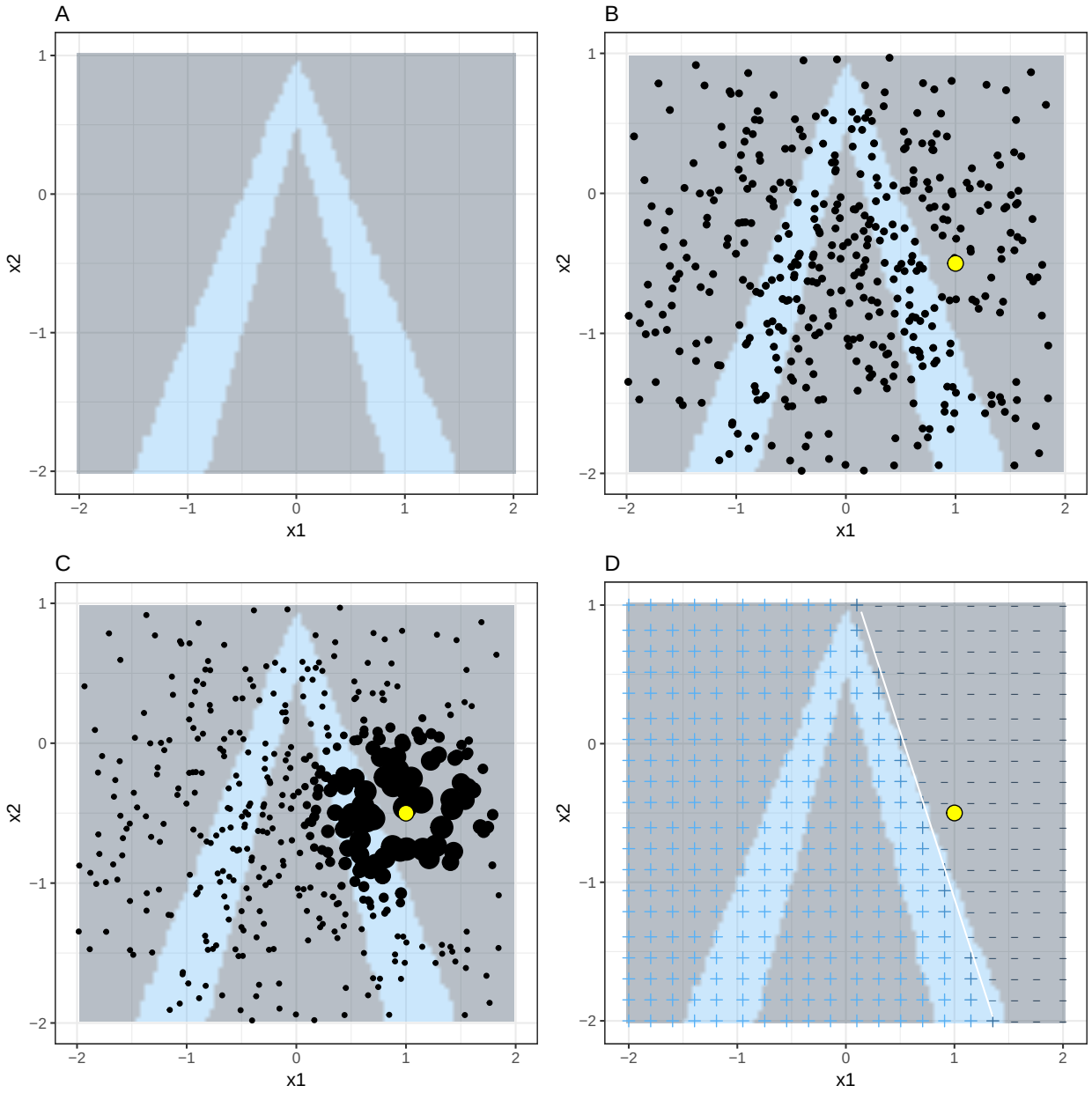


FIGURE 6.6
How LIME sampling works: A) The trainig data has two classes. The most data points have class 0, and the ones with class 1 are grouped in an upside-down V-shape. The plot displays the decision boundaries learned by a machine learning model. In this case it was a Random Forest, but it does not matter, because LIME is model-agnostic and we only care about the decision boundaries. B) The yellow is the instance of interest, for which an explanation is desired. The black dots are data sampled from a normal distribution around the means of the features in the training sample. This has only to be done once and can be reused for other explanations. C) Introducing locality by giving points near the instance of interest a higher weights. D) The colors and signs of the grid display the classifications of the locally learned model form the weighted samples. The white line marks the decision boundary (P(class) = 0.5) at which the classification changes.

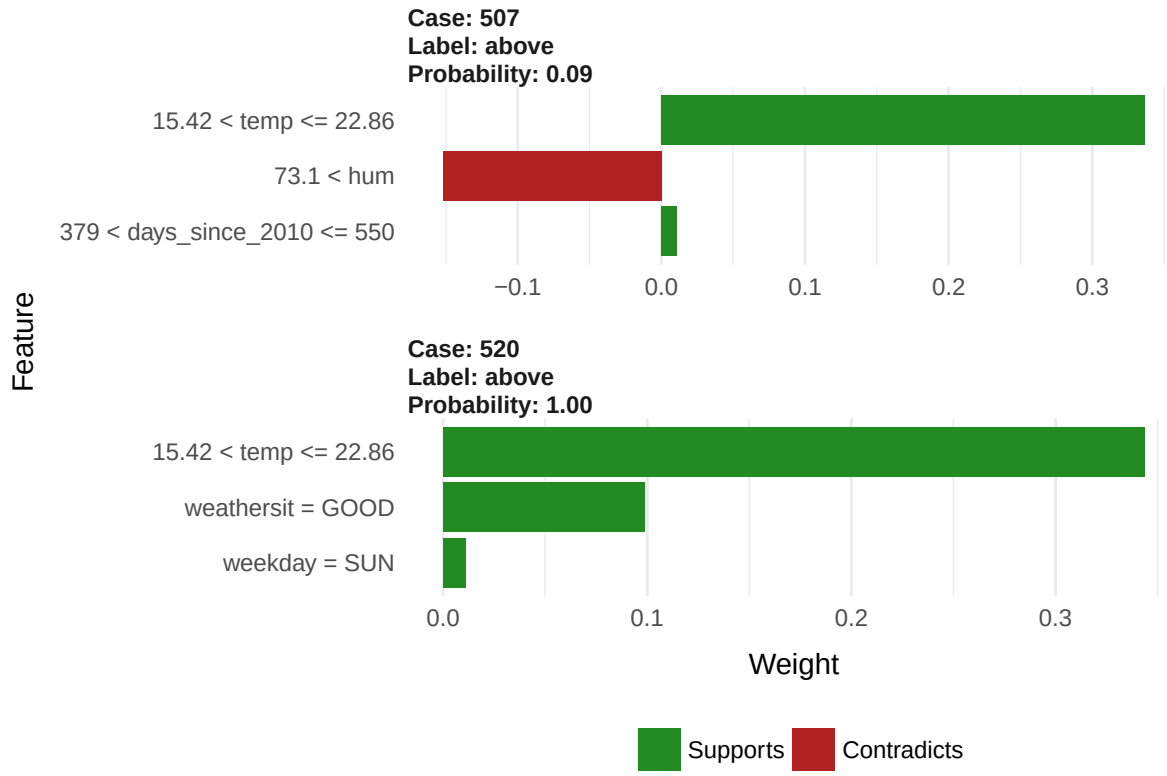


FIGURE 6.7
Explanations for two instances. This time continuous features were turned into categorial features by binning them.

with different predicted classes. It becomes clear from the figure, that it is easier to interpret categorical features than continuous features. Figure @ref{fig:lime-tabular-example-explain-plot-2} shows a variant where the continuous features are turned into categorial features by putting them into bins along the quantiles.

6.4.0.3 LIME for images

For images the sampling procedure works differently. Instead of sampling single pixels, LIME create variations of the image by turning off superpixel.

6.4.0.4 LIME for text

LIME for text works a bit differently than for tabular data. Variation of the point to be explained are created differently: Starting from the original text, new texts are created by randomly removing words from it.

6.4.0.5 Example

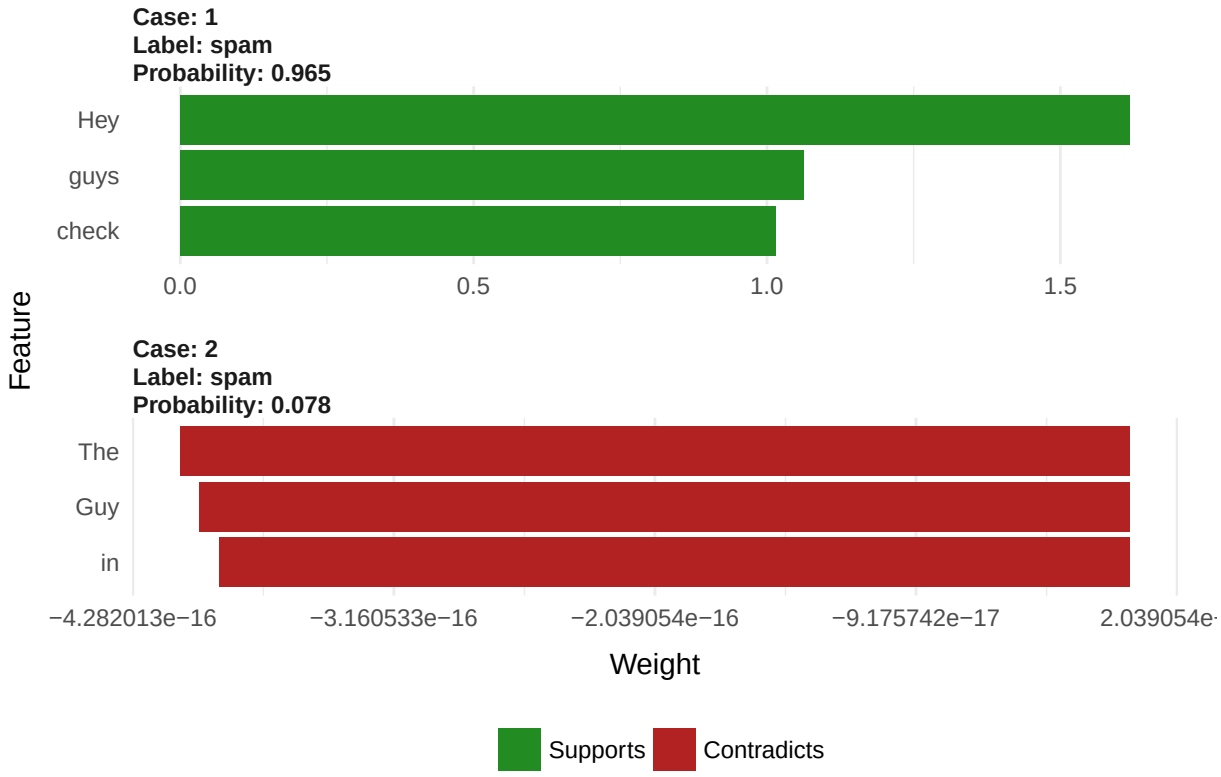
In this example we classify spam vs. ham of YouTube comments. The dataset is described in [#TubeSpam].

The black box model is a decision tree on the document word matrix. Each comment is one document (= one row) and each column is a the number of occurences of a specific word. A decision tree was trained on this data. As discussed in Section [#simple], decision trees are easy to understand, but in this case the tree is very deep. Also in the place of this tree there could have been a recurrent neural network or a support vector machine that was trained on the embeddings from word2vec. The machine learning model was trained on 80% of the approximately 2000 comments. From the remaining comments two were selected for showing the explanations.

Let's look at tow comments of this dataset and the corresponding classes:

In the next step we create some variations of the datasets, which are used in a local model. For example some variations of one of the comments.

Each column corresponds to one word in the sentence. Each row is a variation. 1 indicates that the word is part of this variation. 0 indicates that the word was removed. The corresponding sentence for the first variation is "`Guy in the yellow suit Jae-suk`".



Problems with LIME

- LIME does not work if the classification is very unbalanced (one class is very common) and the black box only predicts one class
- Defining the neighbourhood is tricky.

Bibliography

(2017). Definition of algorithm. https://www.merriam-webster.com/dictionary/algorithm. Accessed: 2017-02-12.

Alberto, T. C., Lochter, J. V., and Almeida, T. A. (2015). Tubespam: Comment spam filtering on youtube. In *Machine Learning and Applications (ICMLA), 2015 IEEE 14th International Conference on*, pages 138–143. IEEE.

Breiman, L. (2001). Random forests. *Machine learning*, 45(1):5–32.

Doshi-Velez, F. and Kim, B. (2017). Towards A Rigorous Science of Interpretable Machine Learning. (Ml):1–13.

Fanaee-T, H. and Gama, J. (2013). Event labeling combining ensemble detectors and background knowledge. *Progress in Artificial Intelligence*, pages 1–15.

Fernandes, K., Cardoso, J. S., and Fernandes, J. (2017). Transfer learning with partial observability applied to cervical cancer screening. In *Iberian Conference on Pattern Recognition and Image Analysis*, pages 243–250. Springer.

Friedman, J. H. (2001). Greedy function approximation: a gradient boosting machine. *Annals of statistics*, pages 1189–1232.

Friedman, J. H. and Popescu, B. E. (2008). Predictive learning via rule ensembles. *The Annals of Applied Statistics*, pages 916–954.

Goldstein, A., Kapelner, A., Bleich, J., and Pitkin, E. (2015). Peeking inside the black box: Visualizing statistical learning with plots of individual conditional expectation. *Journal of Computational and Graphical Statistics*, 24(1):44–65.

Hastie, T., Tibshirani, R., and Friedman, J. (2009). *The elements of statistical learning*.

Lipton, Z. C. (2016). The Mythos of Model Interpretability. *ICML Workshop on Human Interpretability in Machine Learning*, (Whi).

Miller, T. (2017). Explanation in artificial intelligence: Insights from the social sciences. *arXiv preprint arXiv:1706.07269*.

Ribeiro, M. T., Singh, S., and Guestrin, C. (2016). Model-Agnostic Interpretability of Machine Learning. *ICML Workshop on Human Interpretability in Machine Learning*, (Whi).

Strobl, C., Boulesteix, A.-L., Kneib, T., Augustin, T., and Zeileis, A. (2008). Conditional variable importance for random forests. *BMC bioinformatics*, 9:307.