Interpretable machine learning

A guide to making black box models interpretable.

Christoph Molnar

2017-11-30

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 models and their decisions interpretable.

Machine learning models are already used to choose the best advertisement for you, it filters out spam from your emails and it even assesses risk in the judicial system which ultimately can have consequences for your freedom. Can everyone trust the learned model? The model might perform well on the training data, but are the learned associations general enough to transfer to new data? Are there some oddities in the training data which the machine learning model dutifully 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 simple, interpretable models and instructions how to do the interpretation. The later chapters focus on general model-agnostics tools that help analysing complex models and making their decisions interpretable. In an ideal future, machines will be able to explain their decisions and the algorithmic age we are moving towards will be as human as possible.

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 am something between statistician and machine learner. I work on making machine learning interpretable. If you are interested in bringing interpretability 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 What to expect from this book

The book will teach you how to make (supervised) machine learning models interpretable. It contains one or the other mathematical 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 are new to machine learning, there are loads of books and other resources for learning the basics. I recommend the book Elements of Statistical Learning from Hastie et al. (2009) and Andrew Ng's "Machine Learning" online course on coursera to get started with machine learning. Both the book and the course are available for free!

This book starts out with exploring the concepts of machine learning interpretability in Chapter 2: It talks about when interpretability is important and different types of explanations. Definitions used throughout the book can be looked up in Chapter 4. All models and methods are explained and demonstrated with real data examples from Chapter 3. One way to make machine learning interpretable is by using interpretable models, like linear models or decision trees. Interpretable models are introduced in Chapter 5. The other option is to use model-agnostic interpretability methods, which are the topic of Chapter 6. This chapter covers methods like partial dependence plots and permutation feature importance. Model-agnostic methods work by changing the input of the machine learning model and measuring changes in the output.

You can either read the book from start to end or directly jump to the methods you are interested in. I hope you will enjoy the read!

1.2 What is machine learning and why is it important?

Machine learning is a method for teaching computers to make and improve predictions or behaviours based on data.

Predicting the value of a house by learning from historical house sales can be done with machine learning. The book focuses on supervised machine learning, which includes all problems where we know the label or the outcome of interest (e.g. the past sale prices of houses) and want to learn to predict. Excluded from supervised learning are, for example,

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

clustering tasks (=unsupervised learning), where we have no label, but want to find clusters of data points. Also excluded are things like reinforcement learning, where an agent learns to optimise some reward by acting in an environment (e.g. a computer playing Tetris). The goal in supervised learning is to learn a predictive model that maps features (e.g. house size, location, type of floor, ...) to an output (e.g. value of the house). If the output is categorical, the task is called classification and if it is numerical, then regression. Machine learning is a set of algorithms that can learn these mappings from training data, which are pairs of input features and a target. The machine learning algorithm learns a model by changing parameters (like linear weights) or learning structures (like trees). The algorithm is guided by a score or loss function that is minimised. In the house value example, the machine minimises some form of difference between the estimated house sales price and the predicted sales price. A fully trained machine learning model can then be used to make predictions for new instances and be integrated into a product or process.

Estimating house values, recommending products, identifying street signs, counting people on the street, assessing a person's credit worthiness and detecting fraud: All these examples have in common that they can and increasingly are realised with machine learning. The tasks are different, but the approach is the same: Step 1 is to collect data. The more, the better. The data needs to have the information you want to predict and additional information from which the prediction should be made. For a street sign detector ("Is there a street sign in the image?") you would collect street images and label them accordingly with street sign yes vs. no. For a loan default predictor you need historical data from actual loans, the information if the customers defaulted on their loans and data that helps you predict, like the customers income, age and so on. For a house value estimator, you would want to collect data from historical house sales and information about the real estate like size, location and so on. Step 2: Feed this information into a machine learning algorithm, which produces a sign detector model, a credit worthiness model or a house value estimator. This model can then be used in Step 3: Integrate the model into the product or process, like a self-driving car, a loan application process or a real estate marketplace website.

Machines exceed humans in a lot of tasks, like playing chess (or, since recently, Go) or predicting the weather. Even if the machine is as good as a human at a task, or slightly worse, there remain big advantages in 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. Replicating a machine learning model on another machine is fast and cheap. Training a second human to do a task can take decades (especially when they are young) and is very costly. A big disadvantage of using machine learning is that insights about the data and the task the machine is solving are hidden within increasingly complex models. You need millions of numbers to describe a deep neural network and there is no way to understand the model in it's entirety. Other models, like the RandomForest, consist of hundreds of decision trees that "vote" to make predictions. Again, to fully understand how the decision was made, you would need to look into the votes and structures of each of the hundreds of trees. That just does not work out, no matter how clever you are or how good your working memory is. The best performing models are blends of multiple models (also called ensembles), which in itself cannot be interpreted, even if each single model would be interpretable. If you only focus on

performance, you automatically will get more and more opaque models. Just have a look at interviews with winners on the kaggle.com machine learning competition platform: The winning models were mostly ensembles of models or very complex models like boosted trees or deep neural networks.

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 The importance of machine learning interpretability

If a machine learning model performs well, **why not just trust** it and ignore why it made a certain decision?

Let's dive deeper into the reasons why interpretability is so important. Machine learning has come to a state where you have to make a trade-off: Do you simply want to know **what** is predicted happen? For example if a client will churn or if a medication will work well for a patient. Or do you want to know **why** something is predicted to 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 predictive performance 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 it can also tell you why a model might fail. Some problems might not need explanations, because they either are low risk, meaning a mistake has no severe consequences, (e.g. a movie recommender system) or the method has already been extensively studied and evaluated (e.g. optical character recognition). The necessity for interpretability comes from an incompleteness in the problem formalisation (Doshi-Velez and Kim, 2017), meaning that for certain problems or tasks it is not enough to get the answer (the **what**), but the model also has to give an explanation how it came to the answer (the **why**), because correctly predicting is not enough to solve the problem. Following reasons drive the demand for interpretability.

- 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 and 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 recognise 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 which discriminate against protected groups. Interpretability is a useful debugging tool to **detect bias** in machine learning models.

Even in low risk environments, like movie recommendation, interpretability in the research and development stage is valuable. Also later when some model is used in a product, things can go wrong. And need for interpretability arises when your model fucks up. 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 misclassifies some huskies as wolfs. If there is an explanation to the classification you can see, that the misclassification 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.

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 discriminating against protected groups (implicit or explicit). An interpretable model can tell you why 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 or Robustness: Small changes in the input don't lead to big changes in the prediction.
- Causality: Only causal relationships are picked up. Meaning a predicted change in a decision due to arbitrary changes in the input values are also happening in reality.
- Trust: It is easier for humans to trust into a system that explains it's decisions compared to a black box

2.2 Scope of interpretability

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

2.2.1 Algorithm transparency

How does the algorithm create the model?

Algorithm transparency is about how the algorithm learns a model from the 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 interpretability and not algorithm transparency. 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 well understood and the inner workings are in the focus of on-going research. It is not clear how they exactly work, so they are less transparent.

2.2.2 Global, holistic model interpretability

How does the trained model make predictions?

You could call a model interpretable 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 interpretability is about understanding how the model makes the decisions, based on a holistic view 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 interpretability helps to understand the distribution of your target variable based on the features. Arguably, global model interpretability 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 short term memory. I'd argue that you cannot really imagine a linear model with 5 features and draw in your head the hyperplane that 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.

2.2.3 Global model interpretability 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 interpretable on a strict parameter level. While global model interpretability 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, the interpretable parts are the weights and the distribution of the features, for trees

it would be splits (used feature plus the cut-off point) and leaf node predictions. Linear models for example look like they would be perfectly interpretable on a modular level, but the interpretation of a single weight is interlocked with all of the other weights. As you will see in Chapter 5.2, the interpretation of a single weight always comes with the footnote that the other input features stay at the same value, which is not the case in many real world applications. A linear model predicting the value of a house, which takes into account both the size of the house 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 less rooms might be worth less than a flat with more rooms when both have the same size. The weights only make sense in the context of the other features used in the model. But arguably the weights in a linear model still have better interpretability than the weights of a deep neural network.

2.2.4 Explain the prediction for a single instance

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

You can zoom in on a single instance and examine what kind of prediction the model makes 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 features rather than having a complex dependence on the features. For example the value of an apartment might not depend linearly on the size, but if you only look at a specific apartment of 100 square meters and check how the prize changes going up plus and minus 10 square meters there is a chance that this subregion in your data space is linear. Local explanations can be more accurate compared to global explanations because of this. This book presents methods that can make single predictions more interpretable in Chapter 6.

2.2.5 Explain the predictions for a group of instances

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

The model predictions for multiple instances can be explained by either using methods for global model interpretability (on a modular level) or single instance explanations. The global methods can be applied by taking the group of instances 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.

2.3 Evaluating interpretability

There is no real consensus what interpretability in machine learning is. Also it is not clear how to measure it. But there is some first research on it and the attempt to formulate some approaches for the evaluation, as described in the following section.

2.3.1 Approaches for evaluating the explanation quality

Doshi-Velez and Kim (2017) propose three major levels of evaluating explainability.

- **Application level evaluation (real task)**: Put the explanation into the product and let the end user test it. For example, on an application level, radiologists would test fracture detection software (which includes a machine learning component to suggest where fractures might be in an x-ray image) directly 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. For example it might be known that the end users understand decision trees. In this case, a proxy for explanation quality might be the depth of the tree. Shorter trees would get a better explainability rating. It would make sense to add the constraint that the predictive performance of the tree remains good and does not drop too much compared to a larger tree.

2.3.1.1 More on function level evaluation

Model size is an easy way to measure explanation quality, but it is too simplistic. For example, a sparse model with features that are themselves not interpretable is still not a good explanation.

There are more dimensions to interpretability:

- Model sparsity: How many features are being used by the explanation?
- Monotonicity: Is there a monotonicity constraint? Monotonicity means that a feature has a monotonic relationship with the target. If the feature increases, the target either always increases or always decreases, but never switches between increasing and decreasing.
- Uncertainty: Is a measurement of uncertainty part of the explanation?
- Interactions: Is the explanation able to include interactions 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

3

Datasets

Throughout the book all the models and techniques will be applied on real datasets, which are freely available online. We will be using different datasets for different tasks: 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., along with weather and seasonal information. The data was kindly open sourced by Capital-Bikeshare and the folks from Fanaee-T and Gama (2013) have added the weather data and the 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.

For the examples, new features were introduced and not all original features used. Here is the list of features that were used:

- season : sprint (1), summer (2), autumn (3), winter (4)
- holiday : Binary feature indicating if the day was a holiday (1) or not (0)
- yr: The year (2011 or 2012)
- days_since_2011: Number of days since the 01.01.2011 (the first day in the dataset). This feature was introduced to account for the trend, in this case that the bike rental service became more popular over time.
- workingday : Binary feature indicating if the day was a workingday (1) or weekend / holiday (0).
- weathersit : The weather situation on that day.
  - Clear, Few clouds, Partly cloudy, Partly cloudy
  - Mist + Cloudy, Mist + Broken clouds, Mist + Few clouds, Mist
  - Light Snow, Light Rain + Thunderstorm + Scattered clouds, Light Rain + Scattered clouds
  - Heavy Rain + Ice Pallets + Thunderstorm + Mist, Snow + Fog
- temp : Temperature in Celsius
- hum: Relative humidity in percent (0 to 100)
- windspeed: Wind speed in km per hour
- cnt: Count of total rental bikes including both casual and registered. The count was used as the target in the regression tasks.

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 an article about spam classification made the data freely available (Alberto et al., 2015).

The comments were 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 "Gangnam Style" from Korean artist Psy. The other artists where Katy Perry, LMFAO, Eminem and Shakira.

You can flip through some 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'.

	CONTENT	CLASS
3	just for test I have to say murdev.com	1
4	me shaking my sexy ass on my channel enjoy ^_^	1
5	watch?v=vtaRGgvGtWQ Check this out .	1
6	Hey, check out my new website!! This site is about kids stuff. kidsmediausa . com	1
7	Subscribe to my channel	1
8	i turned it on mute as soon is i came on i just wanted to check the views...	0
9	You should check my channel for Funny VIDEOS!!	1
10	and u should.d check my channel and tell me what I should do next!	1
11	Hey subscribe to me	1

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 cocktails from tourists on the beach. Also the Google Spam detector probably has changed a lot since 2015.

3.3 Risk factors for cervical cancer (classification)

The cervical cancer dataset contains indicators and risk factors for predicting if a woman will get cervical cancer. The features contain demographics (e.g. age), habits and medical history. The data can be downloaded from the UCI Machine Learning repository is described by Fernandes et al. (2017).

The subset of features, which are used in this book are:

- (int) Age
- (int) Number of sexual partners
- (int) First sexual intercourse (age)

- (int) Num of pregnancies
- (bool) Smokes yes (1) or no (1)
- (int) Smokes (years)
- (bool) Hormonal Contraceptives yes (1) vs no (0)
- (int) Hormonal Contraceptives (years)
- (bool) IUD: Intrauterine device yes (1) vs no (1)
- (int) IUD (years): Number of years with an intrauterine device
- (bool) STDs: Ever had a sexually transmitted disease? Yes (1) vs no (0)
- (int) STDs (number): Number of sexually transmitted diseases.
- (int) STDs: Number of diagnosis
- (int) STDs: Time since first diagnosis
- (int) STDs: Time since last diagnosis
- (bool) Biopsy: Biopsy results "Healthy" or "Cancer". Target outcome.

As the biopsy serves as the gold standard for diagnosing cervical cancer, the classification task 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 a 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!

4

Definitions

To avoid confusion through ambiguity, here are some definitions of terms used in this book.

- An **Algorithm** is a set of rules that a machine follows to achieve a particular goal (alg, 2017)
- A **Machine learning algorithm** is a 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.
- A **(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 information about the architecture.
- **Dataset**: A table containing the data from which the machine learns.
- **Features**: The features/information used for prediction/classification/clustering. A feature is one column in the dataset.
- **Target**: The thing the machine learns to predict.
- **(machine learning) Task**: The combination of a dataset with features and a target. Depending on the type of the target, the task can be classification, regression, survival analysis, clustering or outlier detection.
- **Prediction**: The machine learning models "guess" what the target's value should be based on the given features.
- **Instance**: One row in the dataset.

5

Interpretable models

The most straightforward way to get to interpretable machine learning is to use only a subset of algorithms that create interpretable models.

Very common model types of this group of interpretable models are:

- Linear regression model
- Logistic regression
- Decision trees

In the following chapters we will talk about these models. Not in detail, only the basics, because there are already a ton of books, videos, tutorials, papers and more material. We will focus on how to interpret the models. The chapter covers linear models, logistic regression and decision trees in details and lists some more.

All of the interpretable models types explained in this book are interpretable on a modular level, except for the k-nearest neighbours method. The following table gives an overview over the interpretable model types and their properties. A model is linear if the association between features and target is modelled linearly. A monotonic model ensures that the relationship between a feature and the target outcome is always in the same direction over the whole range of the feature: an increase in the features value will consistently lead to either an increase or a decrease of the target outcome, but never both for this feature. Monotonicity is useful for the interpretation of a model, because it makes it easier to understand a relationship. Some models can automatically include interactions between the features for predicting the outcome. You can always include interactions into any kind of model by manually creating interaction features. This can be important for correctly predicting the outcome, but too many or too complex interactions can hurt interpretability. Some models only handle regression, some only classification and some can manage to do both.

You can use this table to choose a suitable interpretable model for your task.

Algorithm	Linear	Monotonicity	Interactions	Task type
Linear models	Yes	Yes	No	Regression
Logistic regression	No	Yes	No	Classification
Decision trees	No	No	Yes	Class. + Regr.
Naive Bayes	Yes	Yes	No	Classification
k-nearest neighbours	No	No	No	Class. + Regr.
RuleFit	Partially	No	Yes	Class. + Regr.

5.1 Terminology

- Y is the target outcome.
- X are the features (also called variables, covariables, covariates or inputs).
- β are regression weights (also called coefficients).

5.2 Linear models

Linear models have been used since a long time by statisticians, computer scientists and other people with quantitative problems. Linear models learn linear (and therefore monotonic) relationships between the features and the target. The linearity of the learned relationship makes the interpretation easy.

Linear models can be used to model the dependency of a regression target y on p features x. The learned relationships are linear and, for a singular instance i, can be written as:

$$y_i = \beta_0 + \beta_1 \cdot x_{i1} + \ldots + \beta_p x_{ip} + \epsilon_i$$

The i-th instance's outcome is a weighted sum of it's p features. The β_j 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.

Different methods can be used to estimate the optimal weight vector $\hat{\beta}$. The ordinary least squares method is commonly used to find the weights that minimise the squared difference between the actual and the estimated outcome:

$$\hat{\beta} = \arg\min_{\beta_0,\ldots,\beta_p} \sum_{i=1}^{n} \left(y_i - \left(\beta_0 + \sum_{j=1}^{p} \beta_j x_{ij} \right) \right)$$

We won't go into detail about how the optimal weights can be found, but if you are interested you can read Chapter 3.2 of the book "Elements of Statistical Learning" (Hastie et al., 2009) or one of the other zillions of sources about linear regression models.

The biggest advantage of linear regression models is the linearity: It makes the estimation procedure straightforward and most importantly these linear equations have an easy to understand interpretation on a modular level (i.e. the weights). That is one of the main reasons why the linear model and all similar models 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 to quantify the influence of the medication while at the same time accounting for things like sex, age and other features in an interpretable manner.

Linear regression models also come with some assumptions that make them easy to use

and interpret but which are often not satisfied in reality. Assumed is: Linearity, normality, homoscedasticity, independence, fixed features and absence of multicollinearity.

- **Linearity**: Linear regression models force the estimated response to be a linear combination of the features, which is both the greatest strength and biggest limitation. 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 use techniques like regression splines to estimate non-linear effects.
- **Normality**: The target outcome 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 features p-values is not valid.
- **Homoscedasticity** (constant variance): The variance of the error terms ϵ_i is assumed to be constant over 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 house, the error terms around the predicted response have the same variance. This assumption is often violated in reality. In the house example it is plausible that the variance of error terms around the predicted price is higher for bigger houses, since also the prices are higher and there is more room for prices to vary.
- **Independence**: Each instance 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 of your input features. And usually you don't want to do that.
- **Absence of multicollinearity**: Basically you don't want features to be highly correlated, because this messes up the estimation of the weights. In a situation where two features are highly correlated (something like correlation > 0.9) it will become problematic to estimate the weights, since the feature effects are additive and it becomes indeterminable to which of the correlated features to attribute the effects.

5.2.1 Interpretation

The interpretation of a weights in the linear model depends on the type of the corresponding feature:

- Numerical feature: For an increase of the numerical feature x_j by one point, the estimated outcome changes by β_j. An example for a numerical feature is the size of a house.
- Binary feature: A feature, that for each instance takes on one of two possible values. An example is the feature "House comes with a garden". One of the values counts as the reference level (in some programming languages coded with 0), like "No garden". A change

of the feature x_j from the reference level to the other level changes the estimated outcome by β_j

- Categorical feature with multiple levels: A feature with a fixed amount of possible values. An example is the feature "Floor type", with possible levels "carpet", "laminate" and "parquet". One solution to deal with many levels is to one-hot-encode them, meaning each level gets it's own column. From a categorical feature with l levels, you only need $l-1$ columns, otherwise the coding is overparameterised. The interpretation for each level is then according to the binary features. Some language like R allow you to code categorical feature in different ways, see Chapter @ref(cat.code)
- Intercept β_0: The intercept is the feature weight for the constant feature, which is always 1 for all instances. Most software packages automatically add this feature for estimating the intercept. The interpretation is: Given all numerical features are zero and the categorical features are at the reference level, the estimated outcome of y_i is β_0. The interpretation of β_0 is usually not relevant, because instances with all features at zero often don't make any sense, unless the features were standardised (mean of zero, standard deviation of one), where the intercept β_0 reflects the predicted outcome of an instance where all features are at their mean.

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 outcome 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 ($SST = \sum_{i=1}^{n}(y_i - \bar{y})^2$). The SSE tells you how much variance remains after fitting the linear model, which is measured by looking at the squared differences between the predicted and actual target values. SST is the total variance of the target around the mean. So R^2 tells you how much of your variance can be explained by the linear model. R^2 ranges between 0 for models that explain nothing and 1 for models that explain all of the variance in your data.

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 the 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 instances.

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.

5.2.2 Interpretation example

In this example we use the linear model to predict the bike rentals on a day, given weather and calendrical information, see Chapter 3.1. For the interpretation we examine the estimated regression weights. The features are a mix of numerical and categorical features.

	Estimate	Std. Error
(Intercept)	2579.8	251.6
seasonSUMMER	864.8	131.0
seasonFALL	54.3	175.0
seasonWINTER	319.6	119.4
holidayHOLIDAY	-639.8	217.2
workingdayWORKING DAY	67.2	78.2
weathersitMISTY	-394.5	94.3
weathersitRAIN/SNOW/STORM	-1863.5	225.0
temp	109.0	7.6
hum	-18.0	3.3
windspeed	-45.3	7.3
days_since_2011	5.0	0.2

Interpretation of a numerical feature ('Temperature'): An increase of the temperature by 1 degree Celsius increases the expected number of bikes by 109.0 given all other features stay the same.

Interpretation of a categorical feature ('weathersituation')): The estimated number of bikes is -1863.5 lower when it is rainy, snowing or stormy, compared to good weather, given that all features stay the same. Also if the weather was misty, the expected number of bike rentals was -394.5 lower, compared to good weather, given all other features stay the same.

As you can see in the interpretation examples, the interpretations always come with the footnote that 'all other features stay the same'. That's because of the nature of linear models: The target is a linear combination of the weighted features. The estimated linear equation spans a hyperplane in the feature/target space (a simple line in the case of a single feature). The β's specify the slope (gradient) of the hyperplane in each direction. The good side is, that it 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. On the bad side of things, the interpretation ignores the joint distribution with other features. Increasing one feature, but not changing others, might create unrealistic or at least unlikely data points.

5.2.3 Interpretation templates

The interpretation of the features in the linear model can be automated by using following text templates.

Interpretation of a numerical feature

An increase of x_k by one unit increases the expectation for y by β_k units, given all other features stay the same.

Interpretation of a categorical feature

A change from x_k's reference level to the other category increases the expectation for y by β_k, given all other features stay the same.

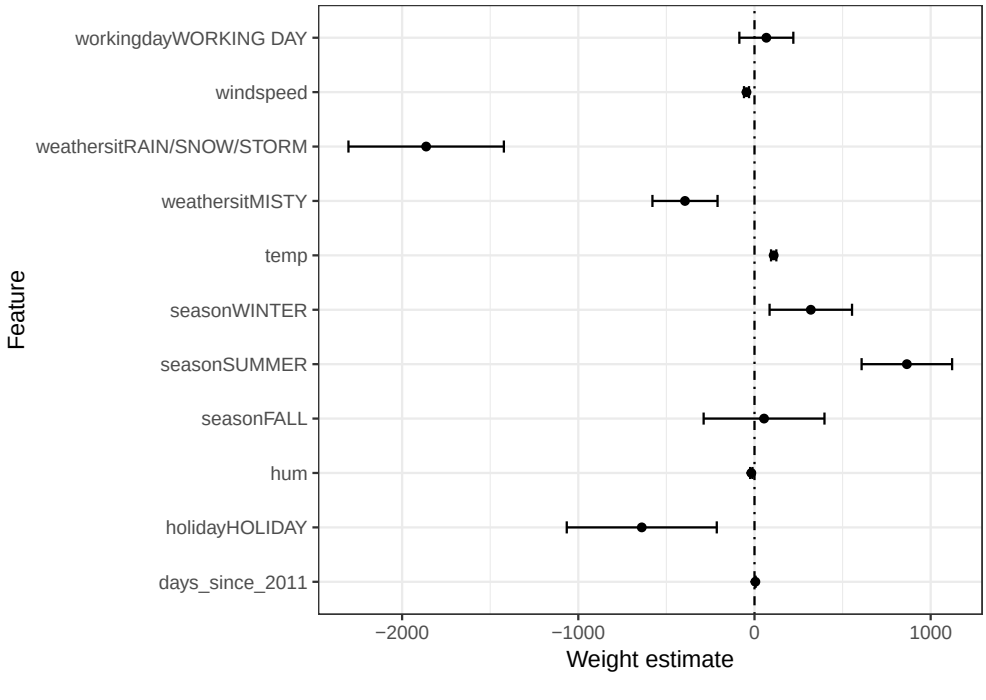
5.2.4 Visual parameter interpretation

Different visualisations make the linear model outcomes easier and quicker to grasp for humans.

5.2.4.1 Weight plot

The information of the coefficient table (weight estimates and variance) can be visualised. Figure 5.2.4.1 shows a weight plot of the linear model fitted before.

\begin{figure}



{

}

\caption{Each row in the plot represents one feature weight. The weights are displayed as points and the 95% confidence intervals around the points with a line. A 95% confidence interval means that if the linear model would be estimated 100 times on similar data, in 95 out of 100 times, the confidence interval would cover the true weight, under the linear model assumptions (linearity, normality, homoscedasticity, independence, fixed features, absence of multicollinearity).} \end{figure} Figure 5.2.4.1 makes clear that rainy/snowy/stormy weather has a strong negative effect on the expected number of bikes. The working day feature's weight is close to zero and the zero is included in the 95% interval, meaning it is not influencing the prediction significantly. Some confidence intervals are very short and the estimates are close to zero, yet the features were important. Temperature is such a candidate.

The problem about the weight plot is that the features are measured on different scales. While for weather situation feature the estimated β signifies the difference between good and rainy/storm/snowy weather, for temperature it signifies only an increase of 1 degree Celsius.

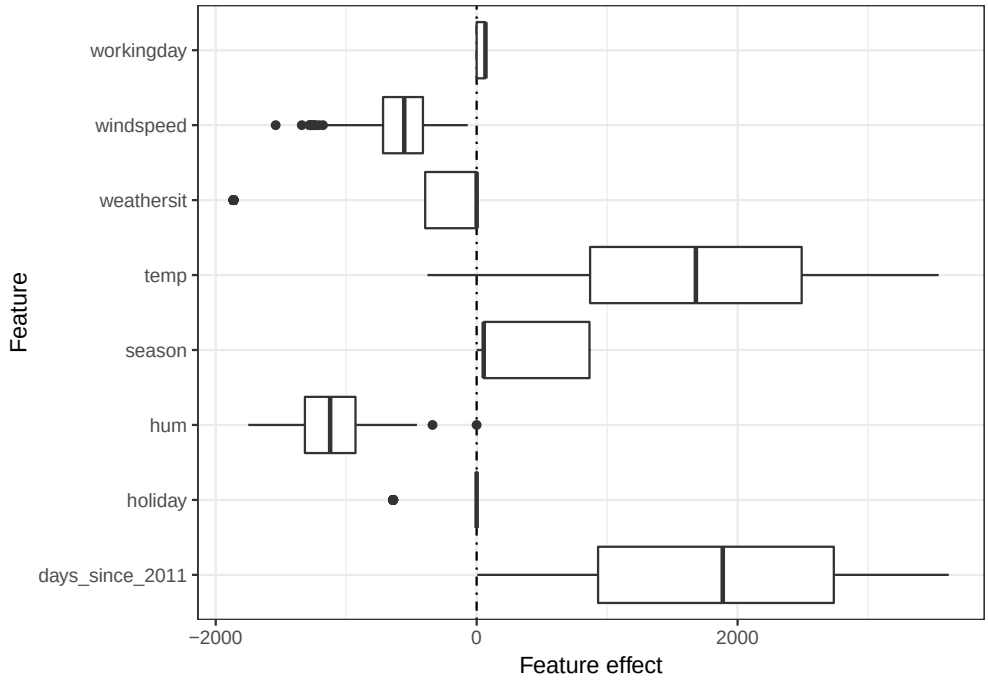


FIGURE 5.1
The feature effect plot shows the distribution of the effects (= feature value times feature weight) over the dataset for each feature.

You can improve the comparison by scaling the features to mean zero and standard deviation of one before fitting the linear model, to make the estimated weights comparable.

5.2.4.2 Effect plot

The weights of the linear model can be analysed more meaningfully, when multiplied with the actual feature values. The weights depend on the scale of the features and will be different if you have a feature measuring some height and you switch from meters to centimetres. 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: $\text{effect}_{i,j} = w_j \cdot x_{i,j}$. The resulting effects are visualised with boxplots: A box in a boxplot 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 lines reach up to $+/-1.58 \text{ IQR}/\sqrt{n}$, with IQR being the inter quartile range ($q_{0.75} - q_{0.25}$). The points are outliers. The categorial feature effects can be aggregated into one boxplot, compared to the weight plot, where each weight gets a row.

The largest contributions to the expected number of bike rentals come from temperature and

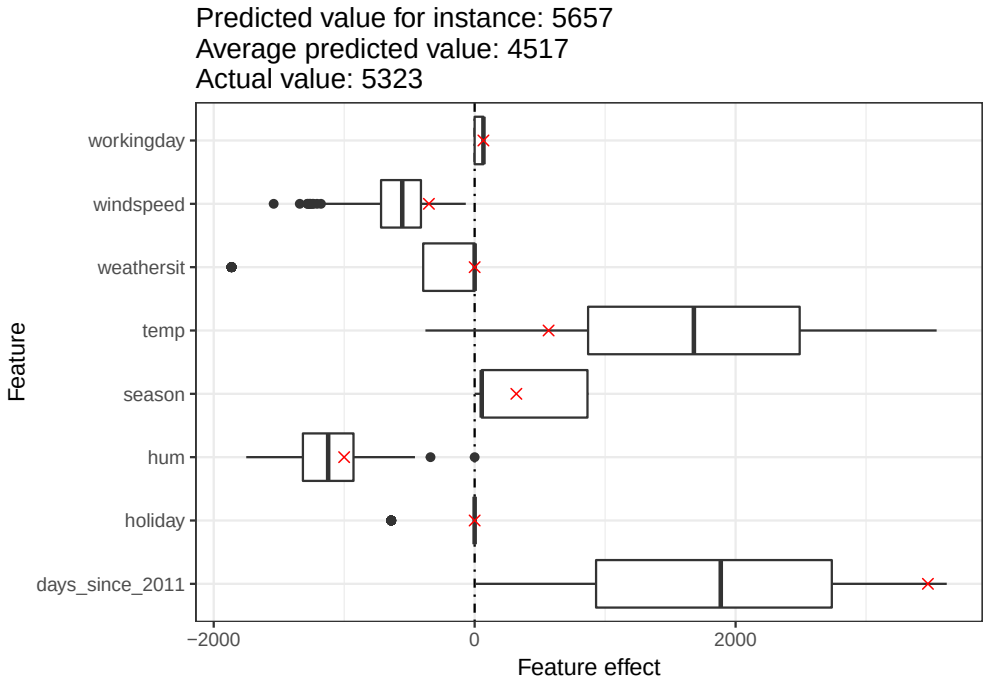


FIGURE 5.2
The effect for one instance shows the effect distribution while highlighting the effects of the instance of interest.

the days feature, which captures the trend that the bike rental service became more popular over time. The temperature has a broad contribution distribution. The day trend feature goes from zero to large positive contribution, because the first day in the dataset (01.01.2011) get's a very low day effect, 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 weight, the instances with a positive effect are the ones that have a negative feature value, so days with a high negative effect of windspeed on the bike rental count have the highest windspeeds.

5.2.5 Explaining single predictions

How much did each feature of an instance contribute towards the prediction? This can, again, be answered by bringing together the weights and feature values of this instance and computing the effects. An interpretation of instance specific effects is only meaningful in comparison with the distribution of each feature's effects.

Let's have a look at the effect realisation for the rental bike count of one instance (= one day). Some features contribute unusually little or much to the predicted bike count, compared to the overall dataset: Temperature (5 degrees) contributes less towards the predicted value compared to the average and the trend feature "days_since_2011" unusually much, because this instance is from late 2011 (698 days).

5.2.6 Coding categorical features

There are several ways to encode a categorical feature and the choice influences the interpretation of the β-weights.

Described in Chapter 5.2 is the treatment coding, which is sufficient in most cases. Using different codings boils down to creating different matrices (=design matrix) from your one column with the categorical feature. This section presents three different codings, but there are many more. The example used has six instances and one categorical feature with 3 levels. For the first two instances, the feature takes on category A, for instances three and four category B and for the last two instances category C.

- **Treatment coding**: The β per level is the estimated difference in y compared to the reference level. The intercept of the linear model is the mean of the reference group (given all other features stay the same). The first column of the design matrix is the intercept, which is always 1. Column two is an indicator whether instance i is in category B, column three is an indicator for category C. There is no need for a column for category A, because then the linear equation would be overspecified and no unique solution (= unique β's) can be found. 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**: The β per level is the estimated y-difference from the level to the overall mean (again, given all other features are zero or the reference level). The first column is again used to estimate the intercept. The weight β_0 which is associated with 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 $\beta_0 + \beta_1$. The interpretation for category C is equivalent. For the reference category A, $-(\beta_1 + \beta_2)$ is the difference to the overall mean and $\beta_0 - (\beta_1 + \beta_2)$ the overall effect.

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

- **Dummy coding**: The β per level is the estimated mean of y for each level (given all feature are at value zero or reference level). Note that the intercept was dropped here, so

that a unique solution for the linear model weights can be found.

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

If you want to dive a bit deeper into different encodings of categorical features, checkout this webpage and this blog post.

5.2.7 The disadvantages of linear models

Linear models can only represent linear relationships. Each **non-linearity or interaction has to be hand-crafted** and explicitly given to the model as an input feature.

Linear models are also often **not that good regarding predictive performance**, because the relationships that can be learned are so restricted and usually oversimplifies how complex reality is.

The interpretation of a weight **can be unintuitive** because it depends on all other features. A feature with high positive correlation with the outcome y and another feature might get a negative weight in the linear model, because, given the other correlated feature, it is negatively correlated with y in the high-dimensional space. Completely correlated features make it even impossible to find a unique solution for the linear equation. An example: You have a model to predict the value of a house and have features like number of rooms and area of the house. House area and number of rooms are highly correlated: the bigger a house is, the more rooms it has. If you now take both features into a linear model, it might happen, that the area of the house is the better predictor and get's a large positive weight. The number of rooms might end up getting a negative weight, because either, given that a house has the same size, increasing the number of rooms could make it less valuable or the linear equation becomes less stable, when the correlation is too strong.

5.2.8 Extending linear models

Linear models have been extensively studied and extended to fix some of the shortcomings.

- Lasso, presented in Chapter 5.2.9.1 is a method to "pressure" weights of irrelevant features to get an estimate of zero. Having unimportant features weighted by zero is useful, because having less terms to interpret makes the model more interpretable.
- Generalised linear models (GLM) allow the target outcome to have different distributions. The target outcome is not any longer required to be normally distributed given the features, but GLMs allow you to model for example Poisson distributed count variables. Logistic regression, presented in Chapter 5.3, is a GLM for categorical outcomes.
- Generalised additive models (GAMs) are GLMs with the additional ability to allow non-

linear relationships with features, while maintaining the linear equation structure (sounds paradox, I know, but it works).

You can also apply all sorts of tricks to go around some of the problems:

- Adding interactions: You can define interactions between features and add them as new features before estimating the linear model. The RuleFit algorithm, introduced in Chapter 5.5.3, can add interactions automatically.
- Adding non-linear terms like polynomials to allow non-linear relationships with features.
- Stratifying data by feature and fitting linear models on subsets.

5.2.9 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. You might even get into a situation with more features than instances and you can't fit a standard linear model at all. The good news is that there are ways to introduce sparsity (= only keeping a few features) into linear models.

5.2.9.1 Lasso

The most automatic and convenient way to introduce sparsity is to use the Lasso method. Lasso stands for "least absolute shrinkage and selection operator" and when added to a linear model, it performs feature selection and regularisation of the selected feature weights.

Let's review the minimization problem, the βs optimise:

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

Lasso adds a term to this optimisation problem:

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

The term $||\beta||_1$, the L1-norm of the feature vector, leads to a penalisation of large β-values. Since the L1-norm is used, many of the weights for the features will get an estimate of 0 and the others are shrunk. The parameter λ controls the strength of the regularising effect and is usually tuned by doing cross-validation. Especially when λ is large, many weights become 0.

5.2.9.2 Other methods for sparsity in linear models

A big spectrum of methods can be used to reduce the number of features in a linear 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 measures 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.

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 the highest R squared). Now again, for the remaining features, fit different versions of your model by adding each feature to your chosen model. Pick the one that performs best. 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 that includes all features and try out which feature you have to remove to get the highest performance increase. Repeat until some stopping criterium is reached.

I recommend using Lasso, because it can be automated, looks at all features at the same time and can be controlled via λ. It also works for the logistic regression model for classification, which is the topic of Chapter 5.3.

5.3 Logistic regression: a linear model for classification

Logistic regression is the linear regression model made fit for classification problems.

5.3.1 What's wrong with linear regression models for classification?

The linear regression model works well in regression setups, 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 1 and use a linear model on it and it would estimate the weights for you. There are just a few problems with that approach:

- A linear model does not output 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 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, which I reproduced in Figure 5.3
- Linear models don't extend to classification problems with multiple classes. You would have to start labeling the next class with a 2, then 3 and so on. The classes might

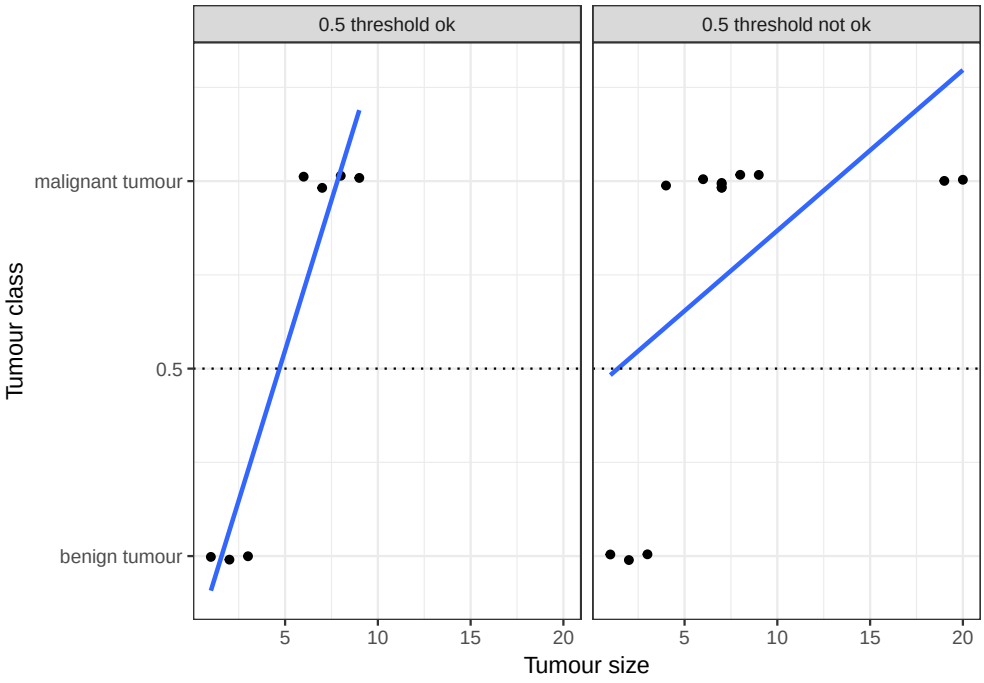


FIGURE 5.3
An illustration why linear regression does not work well in a binary classification setting. A linear model is fitted on artificial data for classifying a tumour as malignant (1) or benign (0), dependant on the size of the tumour. Each point is a tumour, the x-axis shows the size of the tumour, the y-axis the malignancy, points are slightly jittered to reduce over-plotting. The lines display the fitted curve from a 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 separating benign from malignant tumours. After introducing a few more malignant tumour cases, especially with larger tumour sizes, the regression line shifts and a threshold of 0.5 does not separate the classes any longer.

not have any meaningful order, but the linear model would force a weird structure on the relationship between the features and your class predictions. So for a feature with a positive weight, the higher the value of that feature the more it contributes to the prediction of a class with a higher number, even if classes that happened to get a similiar number are not related at all.

5.3.2 Logistic regression

A solution for classification is logistic regression. Instead of fitting a straight line or hyperplane, the logistic regression model uses a non-linear function, the logistic function to squeeze the output of a linear equation between 0 and 1. The logistic function is defined as:

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

And it looks like shown in Figure 5.4.

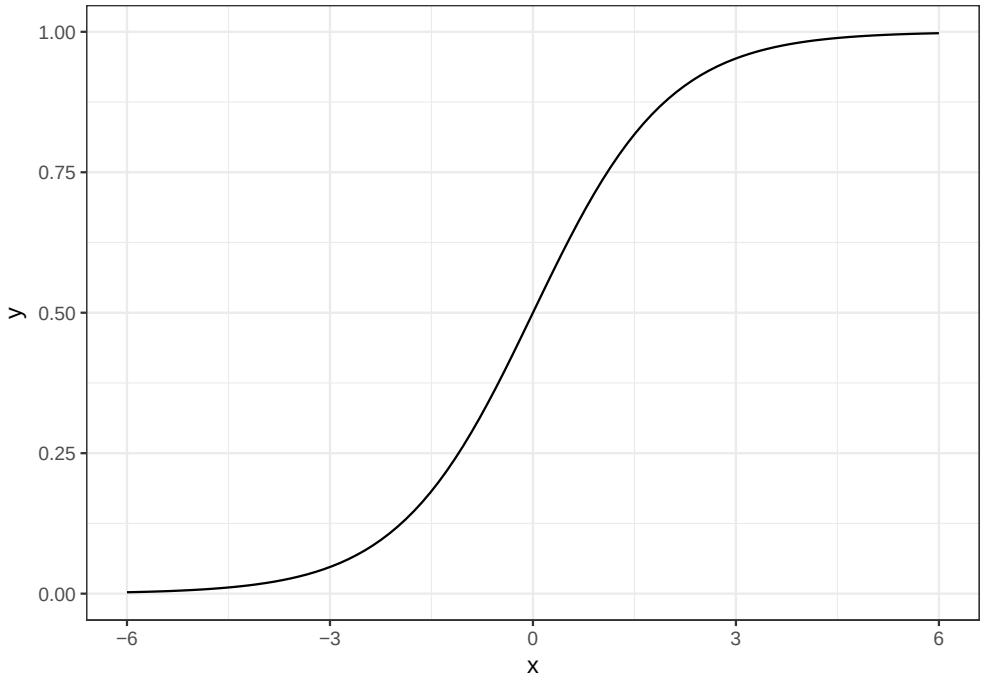


FIGURE 5.4
The logistic function. It only outputs numbers between 0 and 1. At input 0 it outputs 0.5.

The step from linear regression models to logistic regression is kind of straightforward. In the linear regression model we modelled the relationship between the outcome and the features with a linear equation:

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

For the classification we prefer probabilities, which are between 0 and 1, so we wrap the right side of the equation into the logistic regression function and like that force the output to only take on values between 0 and 1.

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

Let's revisit the tumour size example from Figure 5.3 again. But now instead of the linear regression model, we use the logistic regression model:

It works better with logistic regression and we can use 0.5 as a threshold in both cases. Including the additional points does not affect the estimated curve much.

5.3.3 Interpretation

The interpretation of the logistic regression weights differs from the linear regression case, because in logistic regression the outcome is a probability between 0 and 1, and the weights don't affect the probability linearly, but are squeezed through the logistic function. That's why we need to reformulate the equation for the interpretation, so that there is only the linear term left on the right side of the formula.

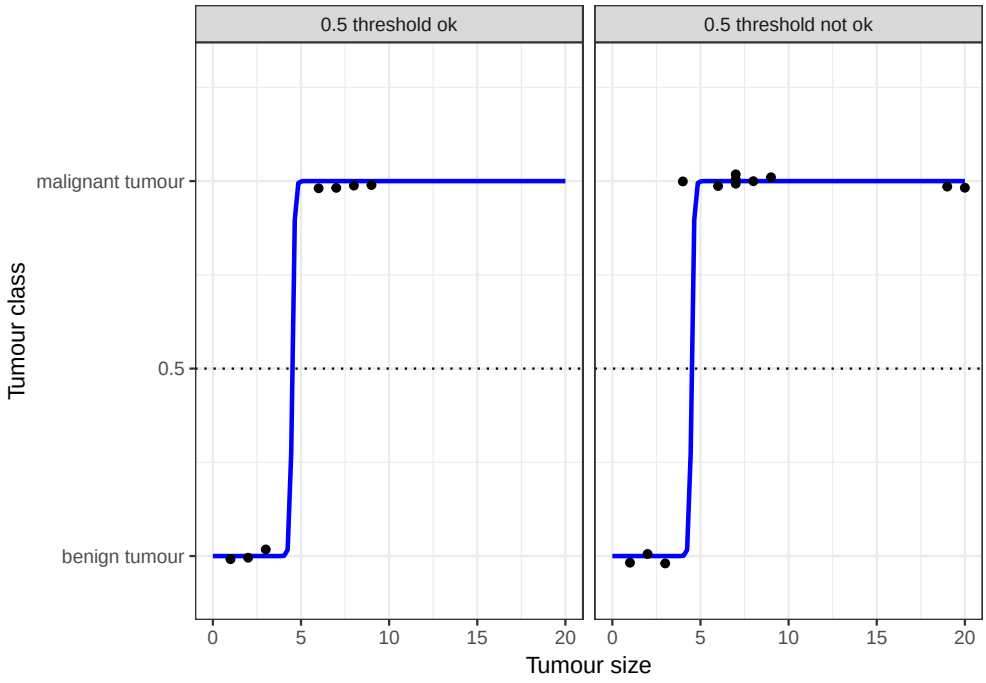


FIGURE 5.5
The logistic regression model successfully finds the correct decision boundary to distinguish between malignant and benign tumours dependent on the size of the tumour in this example. The blue line is the logistic function shifted and squeezed so that it fits the data.

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

$\frac{P(y_i=1)}{1-P(y_i=1)}$ is also called odds (probability of event divided by probability of no event) and $log\left(\frac{P(y_i=1)}{1-P(y_i=1)}\right)$ is called 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_j is changed by 1 point. For this we can first apply the $exp()$ function on both sides of the equation:

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

Then we compare what happens when we increase one of the $x_{i,j}$ by 1. But instead of looking at the difference, we look at the ratio of the two predictions:

$$\frac{odds_{i,x_j+1}}{odds_i} = \frac{exp\left(\beta_0 + \beta_1 x_{i,1} + \ldots + \beta_j(x_{i,j}+1) + \ldots + \beta_p x_{i,p}\right)}{exp\left(\beta_0 + \beta_1 x_{i,1} + \ldots + \beta_j x_{i,j} + \ldots + \beta_p x_{i,p}\right)}$$

Using the rule that $\frac{exp(a)}{exp(b)} = exp(a-b)$ gives us:

$$\begin{aligned}\frac{odds_{i,x_j+1}}{odds_i} = \quad & exp\left(\beta_0 + \beta_1 \cdot x_{i,1} + \ldots + \beta_j \cdot (x_{i,j}+1) + \ldots + \beta_{i,p} x_{i,p}\right) \\ & -exp\left(\beta_0 + \beta_1 \cdot x_{i,1} + \ldots + \beta_j \cdot x_{i,j} + \ldots + \beta_p x_{i,p}\right)\end{aligned}$$

And then we can remove a lot of terms from the equation, which is convenient:

$$\frac{odds_{i,x_j+1}}{odds_i} = exp\left(\beta_j(x_{i,j}+1) - \beta_j x_{i,j}\right) = exp\left(\beta_j\right)$$

And we end up with something simple like $\exp(\beta_j)$. So a change of x_j by one unit changes the odds ratio (multiplicatively) by a factor of $\exp(\beta_j)$. We could also interpret it this way: A change in x_j by one unit changes the log odds ratio by β_j units, but most people do the former because thinking about the $log()$ of something is known to be hard on the brain. Interpreting the odds ratio already needs a bit of getting used to. For example 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_j (=odds ratio) of 0.7, then an increase in the respective x_j by one unit multiplies the odds by $\exp(0.7) \approx 2$ and the odds change to 4. But usually you don't deal with the odds and only interpret the β's as the odds ratios. Because for actually calculating the odds you would need to set a value for each feature x_j, 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 feature types:

- Numerical feature: For an increase of one unit of the feature x_j, the estimated odds change (multiplicatively) by a factor of $\exp(\beta_j)$

TABLE 5.2
The results from fitting a logistic regression model on the cervical cancer dataset. Shown are the features used in the model, their estimated weights and according odds ratios and the standard errors of the estimated weights.

	Weight	Odds ratio	Std. Error
Intercept	2.91	18.36	0.32
Hormonal contraceptives y/n	0.12	1.12	0.30
Smokes y/n	-0.26	0.77	0.37
Num. of pregnancies	-0.04	0.96	0.10
Num. of diagnosed STDs	-0.82	0.44	0.33
Intrauterine device y/n	-0.62	0.54	0.40

- Binary categorical feature: One of the two values of the feature is the reference level (in some languages the one that was coded in 0). A change of the feature x_j from the reference level to the other level changes the estimated odds (multiplicatively) by a factor of $\exp(\beta_j)$
- Categorical feature with many levels: One solution to deal with many possible feature values is to one-hot-encode them, meaning each level gets it's own column. From a categorical feature with L levels, you only need L-1 columns, otherwise it is over-parameterised. The interpretation for each level is then according to the binary features.
- Intercept β_0: Given all numerical features are zero and the categorical features are at the reference level, the estimated odds are $\exp(\beta_0)$. The interpretation of β_0 is usually not relevant.

5.3.4 Example

We use the logistic regression model to predict cervical cancer given some risk factors. The data are described in Chapter 3.3

Interpretation of a numerical feature ('Num. of diagnosed STDs'): An increase of the number of diagnosed STDs (sexually transmitted diseases) 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 feature ('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 to 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'.

5.4 Decision trees

Linear regression models and logistic regression fail in situations where the relationship between features and outcome is non-linear or where the features are interacting with each other. Time to shine for the decision trees! Before you read about decision trees, have a look at Figure 5.6 for an illustration. Tree-based models split the data according to certain cutoff values in the features multiple times. Splitting means that different subsets of the dataset are created, where each instance belongs to one subset. The final subsets are called terminal or leaf nodes and the intermediate subsets are called internal nodes or split nodes. For predicting the outcome in each leaf node, a simple model is fitted with the instances in this subset (for example the subsets average target outcome). Trees can be used for classification and regression.

There are a lot of tree algorithms with different approaches for how to grow a tree. They differ in the possible structure of the tree (e.g. 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 induction. We will focus on CART, but the interpretation is similar for most of the tree types. I recommend the book 'The elements of statistical learning' (Hastie et al., 2009) for a more detailed introduction.

The following formula describes the relationship between outcome y and the features x.

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

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

But where do the subsets come from? This is quite simple: The algorithm takes a feature and tries which cut-off point minimises the sum of squares of y for a regression tasks or the Gini index of the class distribution of y for classification tasks. The best cut-off point makes the two resulting subsets as different as possible in terms of the target outcome. For categorical features the algorithm tries to build subsets by trying different groupings of categories. After this was done for each feature, the algorithm looks for the feature with the best cut-off and chooses it to split the node into two new nodes. The algorithm continues doing this recursively in both of the new nodes until a stopping criterium is reached. Possible criteria are: A minimum number of instances that have to be in a node before the split or the minimum number of instances that have to be in a terminal node.

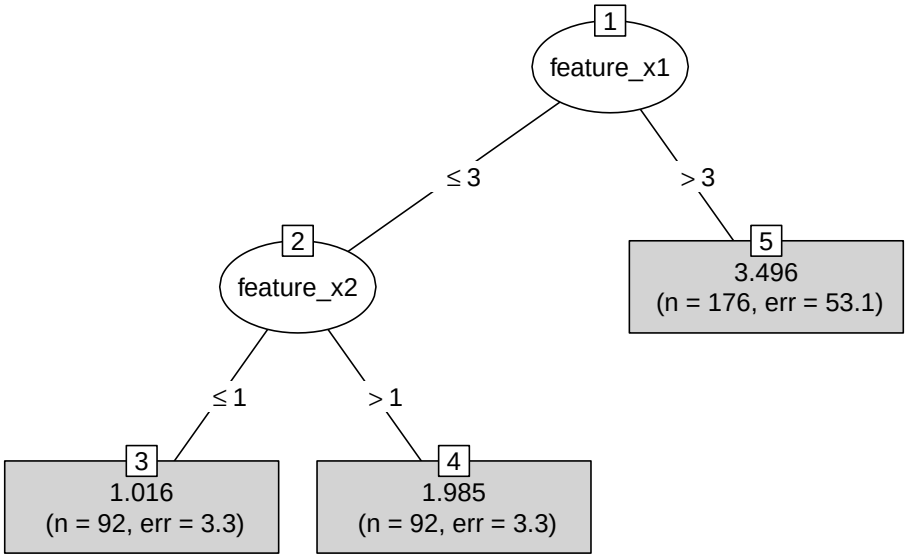


FIGURE 5.6
Decision tree with artificial data. Instances with a value bigger than 3 for feature x1 end up in node 5. All other instances are assigned to node 3 or node 4, depending whether feature x2 values exceed 1

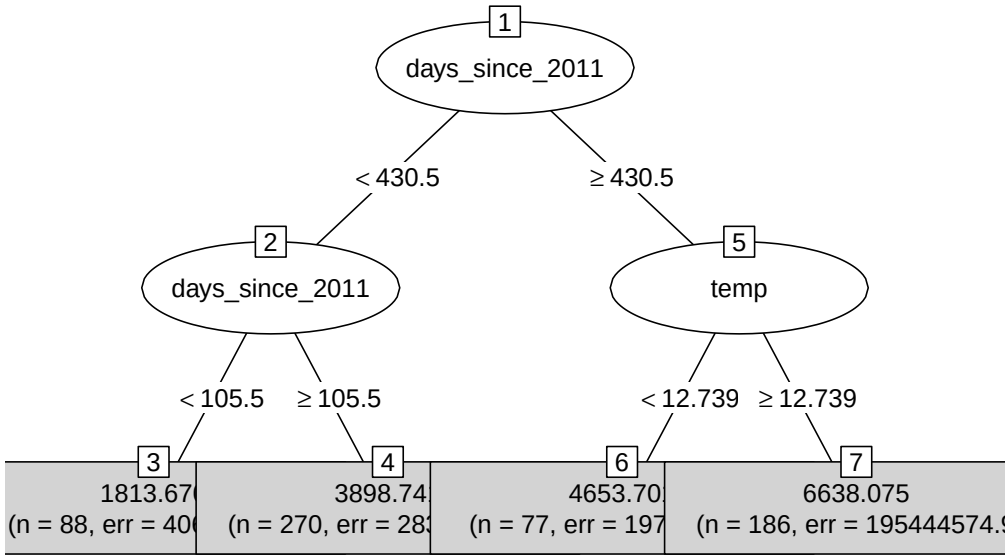


FIGURE 5.7
Regression tree fitted on the bike rental data. The maximally allowed depth for the tree was set to 2. The features picked for the tree splits were the trend feature (days since 2011) and the temperature (temp)

5.4.1 Interpretation

The interpretation is simple: 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 outcome is $\hat{y}_{\text{leafnode}}$.

5.4.2 Interpretation example

Let's have a look again at the bike rental data from Chapter 3.1. We want to predict the number of bike rentals on a given day. The learned tree is visualised in Figure 5.7.

The first split and one of the second splits was done in the trend feature, which tells how many days passed since beginning of the data collection and covers the trend that the bike rental service became more popular over time. For days that came before the 105th day the predicted number of bike rentals is ca. 1800, between the 106th and 430th day it is around 3900. For days after the 430th day, depending on the temperature, the prediction is either 4600 (if below 12 degrees) or 6600 (if above 12 degrees).

5.4.3 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 often easier to grasp than points on a hyperplane like in linear regression. The interpretation is arguably pretty straightforward. The tree structure also has a **natural visualization**, with it's nodes and edges.

5.4.4 Disadvantages

Handling of linear relationships, that's what trees suck at. Any 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 value of a house and the tree splits in the square meters multiple times. One of the splits is at 100.5 square meters. Imagine a user of a house prize estimator, that uses your decision tree model: She measures her house, concludes that the house has 99 square meters, types it into some nice web interface and get's a prediction of 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 house 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, because there was no change from 99 square meters to 100, but from 100 to 101.

Trees are also quite **unstable**, so a few changes in the training dataset might create a completely different tree. That's because each split depends on the parent split. And if a different feature gets selected as the first split feature, the whole tree structure will change. It does not generate confidence in the model if the structure flips so easily.

5.5 Other Interpretable Models

The list of interpretable models is evergrowing and of unkown size. It contains simple models like linear models, decision trees and naive Bayes, but also more complex ones that combine or modify non-interpretable machine learning models to make them interpretable. Especially publications about the latter type of models are currently created with a high frequency and it is hard to keep up with the developments. We only tease a few additional ones in this chapter, especially the simpler and more established candidates.

5.5.1 Naive Bayes classifier

The naive Bayes classifier makes use of the Bayes' theorem of conditional probabilities. For each feature it computes the probability for a class given the value of the feature. The naive Bayes classifier 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 C_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. The probability of a class, given the features is the class probability times the probability of each feature given the class, normalized by Z. This formula can be derived by using the Bayes' theorem.

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

5.5.2 k-Nearest Neighbours

The k-nearest neighbour method can be used for regression and classification and uses the closest neighbours of a data point for prediction. For classification it assigns the most common class among the closest k neighbours of an instance and for regression it takes the average of the outcome of the neighbours. The tricky parts are finding the right k and deciding how to measure the distance between instances, which ultimately defines the neighbourhood.

This algorithm is different from the other interpretable models presented in this book, since it is an instance-based learning algorithm. How is k-nearest neighbours interpretable? For starters, it has no parameters to learn, so there is no interpretability on a modular level, like in linear models. Also, it lacks 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 for the prediction. If the method is interpretable solely depends on the question if you can 'interpret' a 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.

5.5.3 RuleFit

The RuleFit algorithm (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

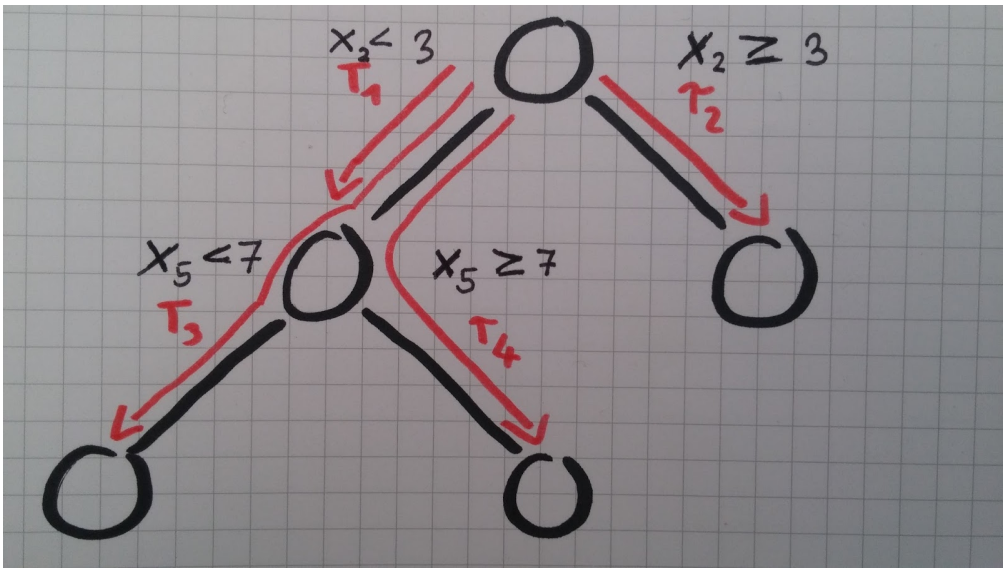


FIGURE 5.8
4 rules can be generated from a tree with 3 terminal nodes.

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.

Step 1: Rule generation

The rules that the algorithm generates have a simple form: For example: "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 in an example: The goal is to predict the median house value in the Boston neighbourhood. One of the rules (read: features) 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. Figure 5.8 illustrates the rule generation. You simply chain the binary decisions that lead to a certain node, and 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 or classifying y with your original features X). Each resulting tree 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 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 regression is a good choice in this scenario, see also Chapter 5.2.9.1. Next to the rules, also all numerical features from your original dataset will be used in the Lasso linear model. Every rule and numerical feature gets a weight (β). And thanks to the regularisation, a lot of these weights will be estimated to zero. The numerical features 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 numerical features 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.

5.5.4 And so many more ...

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

6

Model-agnostic methods

Separating the explanations from the machine learning model (= model-agnostic interpretability methods) has some benefits (Ribeiro et al., 2016). The big advantage of model-agnostic interpretability methods over model-specific ones is their flexibility. The machine learning developer is free to use any machine learning model she likes, when the interpretability methods can be applied to any model. Anything that is build on top of an interpretation of a machine learning model, like a graphic or some user interface, also becomes independent of the underlying machine learning model. Usually not one but many types of machine learning models are evaluated for solving a task and if you compare the models in terms of interpretability, it is easier to do with model-agnostic explanations, because the same method can be used for any type of model.

An alternative to model-agnostic interpretability methods is using only interpretable models as introduced in Chapter 5, which often has the big disadvantage to usually loose accuracy compared to other machine learning models and locking you into one type of model and interpretability method. The other alternative is to use model-specific interpretability methods. The drawback here is also 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 are (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 deep 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 a decision tree or a graphic with feature importances.
- **Representation flexibility:** The explanation system should not have to use the same feature representation as the model that is being explained. For a text classifier that uses abstract word embedding vectors it might be preferable to use the presence of single words for the explanation.

The bigger picture

Let's take a high level view on model-agnostic interpretability. Figure 6.1 shows how we first abstract the world by capturing it by collecting data and abstract it further by learning the essence of the data (for the task) with a machine learning model. Interpretability is just another layer on top, that helps humans understand.

- 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 more abstract things like the

real estate market. 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 order to make it processable for computers and also to store information. The 'Data'-layer contains anything from images, texts, tabular data and so on.
- By fitting machine learning models on top of the 'Data'-layer we get the 'Black Box Model'-layer. Machine learning algorithms learn with data from the real world to make predictions or find structures.
- On top of the 'Black Box Model'-layer is the 'Interpretability Methods'-layer that helps us deal with the opaqueness of machine learning models. What were the important features for a particular diagnosis? Why was a financial transaction classified as fraud?
- The last layer is occupied by a 'Human'. Look! This one is waving 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 differences in approaches between statisticians and machine learning practitioners are. Statistician are concerned with the 'Data' layer, like planning clinical trials or designing surveys. They skip the 'Black Box Model'-layer and go right to the 'Interpretability Methods'-layer. 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. 'Interpretability Methods' is skipped and the human deals directly with the black box models prediction. It's a nice thing, that in interpretable machine learning, the work of a statistician 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 how (model-agnostic) interpretability becomes this new layer on top of machine learning models.

6.1 Partial dependence plot

The partial dependence plot shows the marginal effect of a feature on the predicted outcome (Friedman, 2001). Have a glance at Figure 6.3 to get some intuition before diving into the theory of partial dependencies. A partial dependence plot can show if the relationship between the target and a feature is linear, monotonic or something else. Applied to a linear regression model, partial dependence plots will always show a linear relationship, for example.

The partial dependence function for regression is defined as:

$$\hat{f}_{x_S}(x_S) = E_{x_C}\left[\hat{f}(x_S, x_C)\right] = \int \hat{f}(x_S, x_C) dP(x_C)$$

The term x_S is the set of features for which the partial dependence function should be plotted and x_C are the other features that were used in the machine learning model $\hat{f}$.

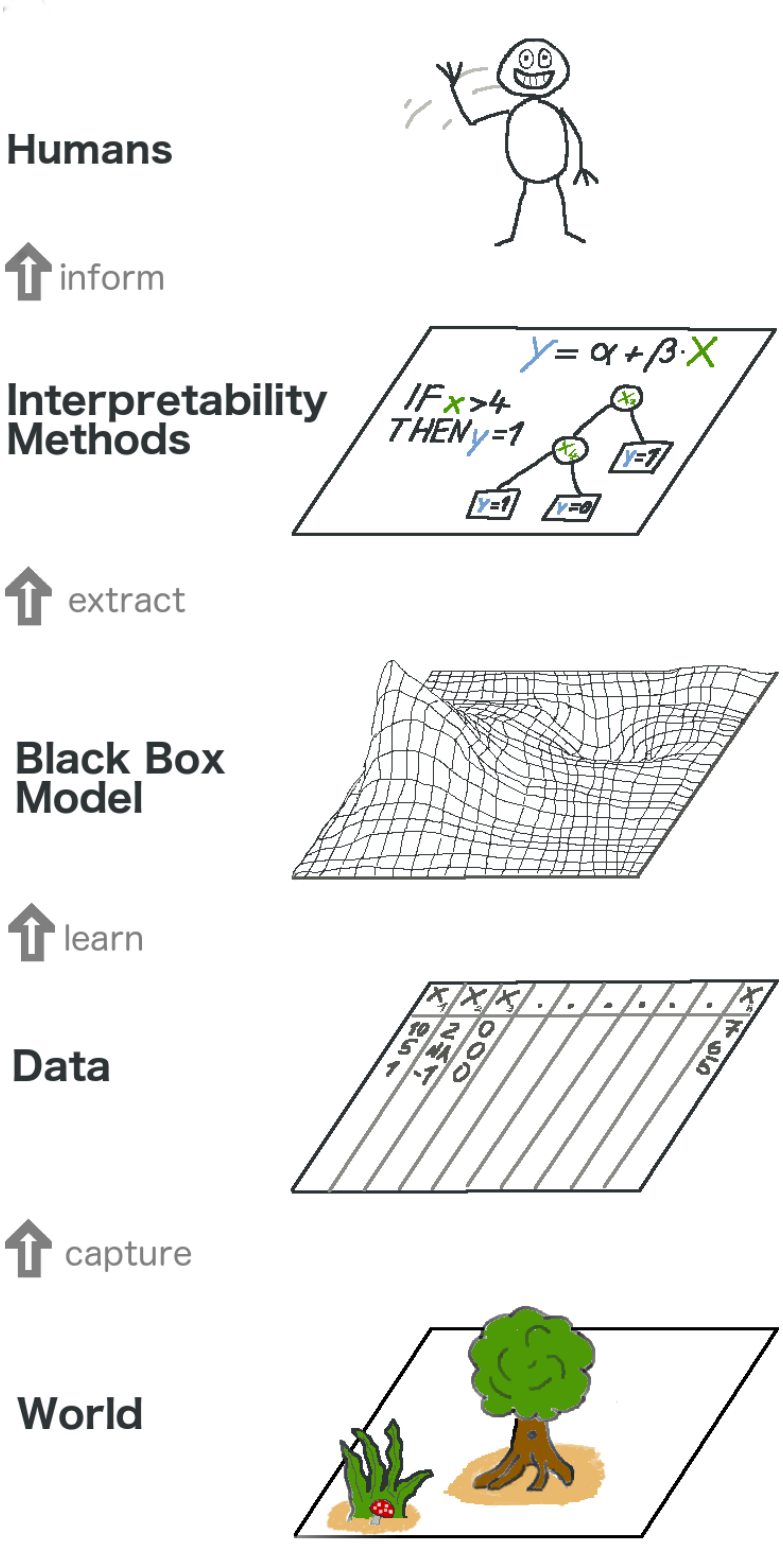


FIGURE 6.1
The big picture of explainable machine learning. The real world goes through many layers before it reaches the human in forms of explanations.

Usually, there are only one or two features in x_S. Together, x_S and x_C make up x. Partial dependence works by averaging the machine learning model output $\hat{f}$ over the distribution of the features x_C, so that the remaining function shows the relationship between the x_S, in which we are interested, and the predicted outcome.

The partial function $\hat{f}_{x_S}$ along x_S is estimated by calculating averages in the training data, which is also known as Monte Carlo method:

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

In this formula x_{iC} are concrete values taken from the data for the features we are not interested and n the number of instances in the dataset. Note that $\hat{f}_{x_S}$ only depends on features x_S as input. For classification, where the machine model outputs probabilities, the partial dependence function displays the probability for a certain class given different values for features x_S. A straightforward way to handle multi-class problems is to plot one line per class.

The partial dependence plot is a global method: The method takes into account all instances and makes a statement about the global relationship of a feature with the predicted outcome.

6.1.1 Examples

In practice, the set of features x_S usually only contains one feature or a maximum of two, because one feature produces 2D plots and two features produce 3D plots. Everything beyond that is quite tricky. Even 3D on a 2D paper or monitor is already challenging.

Let's turn to the regression example with the bike counts from Chapter 3.1 and have a look at how the weather affects the predicted bike rentals. We first fit a machine learning model on the dataset, for which we want to analyse the partial dependencies. In this case, we fitted a RandomForest to predict the bike rentals and make use of the partial dependence method to understand what relationships the model learned. Figure 6.2 shows the influence of the weather features on the predicted bike counts. For warm (but not too hot) weather, the model predicts a high number of bike rentals on average. 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 makes sense. Interestingly, the predicted bike counts don't drop between 25 and 35 km/h windspeed, but maybe there is just not enough training data. At least intuitively I would expect the bike rentals to drop with any increase in windspeed, especially when the windspeed is very high.

Figure 6.3 shows a partial dependence plot example for cervical cancer classification (see Chapter 3.3 for the data). Again, we fit a RandomForest to predict whether a woman has cervical cancer given some risk factors. Any other type of model would have worked as well. The interpretation is analogue to the partial dependence plots for regression.

Figure 6.4 shows an example of a partial dependence plot with two features.

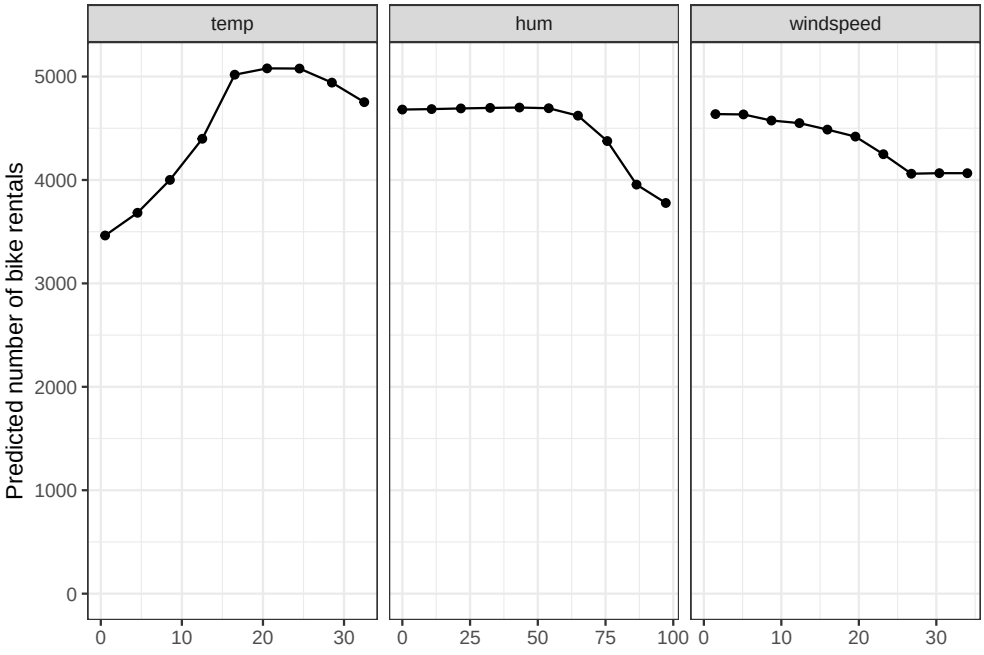


FIGURE 6.2
Partial dependence plot of rental bike count and different weather measurements (Temperature, Humidity, Windspeed). The biggest differences can be seen in different temperatures: With rising temperatures, on average the bike rentals rise, until 20C degrees, where it stays the same also for hotter temperatures and drops a bit again towards 30C degrees.

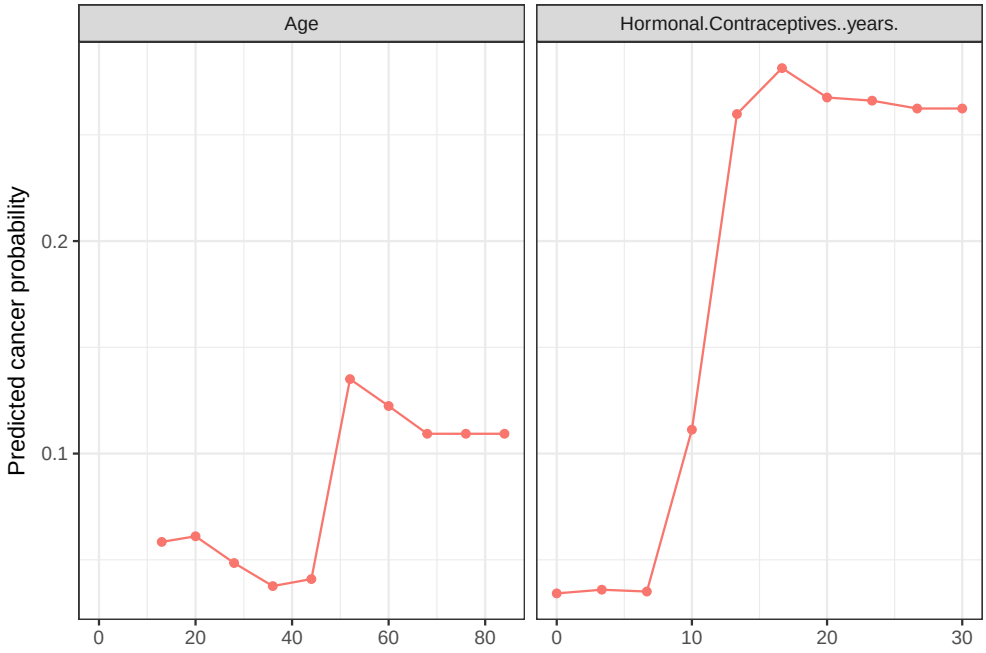


FIGURE 6.3
Partial dependence plot of cancer probability and the risk factors age and number of years with hormonal contraceptives. For the age feature, the models partial dependence shows that on average, the cancer probability is low before 45, spikes between age 45 and 55 and plateaus after that. The number of years on hormonal contraceptives is associated with a higher cancer risk especially after 15 years.

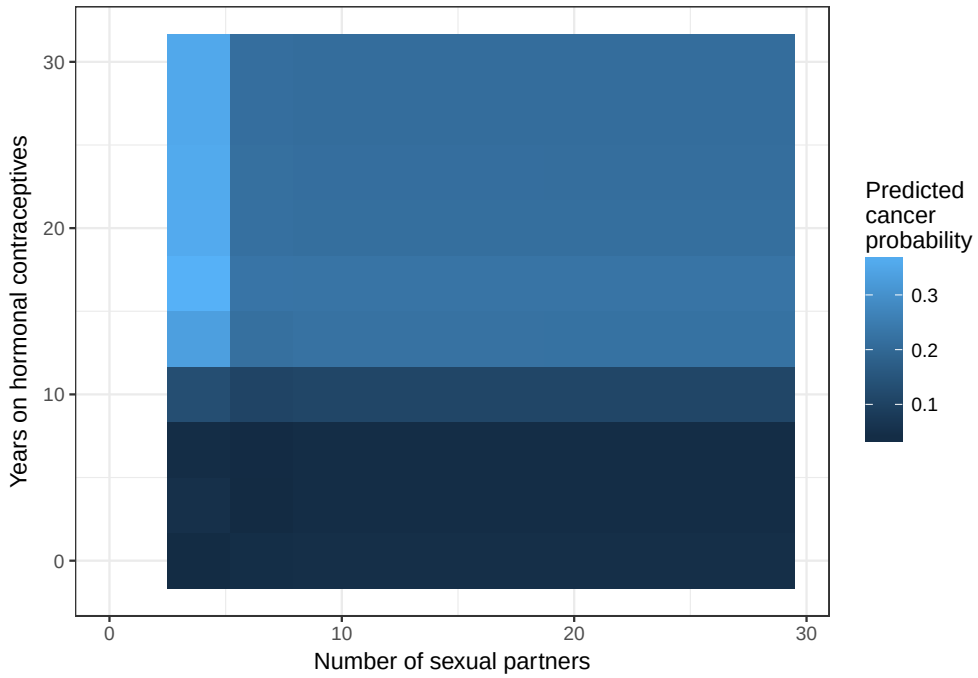


FIGURE 6.4
Partial dependence plot of cancer probability and the interaction of number of years on hormonal contraceptives and number of sexual partners. Interestingly, there is some odd interaction between the two features when the number of sexual partners is 1 and the years of on hormonal contraceptives 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 answer this question.

6.2 Individual Conditional Expectation (ICE) plot

The partial dependence plot for visualising the average effect of a feature is a global method, because it does not focus on specific instances, but on an overall average. The equivalent to a PDP for local expectations is called individual conditional expectation (ICE) plot (Goldstein et al., 2015). An ICE plot visualises the dependence of the predicted response on a feature for EACH instance separately, resulting in multiple lines, one for each instance, compared to one line in partial dependence plots. A PDP is the average of the lines of an ICE plot. The values for a line (and one instance) can be computed by leaving all other features the same, creating variants of this instance by replacing the feature's value with values from a grid and letting the black box make the predictions with these newly created instances. The result is a set of points for an instance with the feature value from the grid and the respective predictions.

So, what do you gain by looking at individual expectations, instead of partial dependencies? Partial dependence plots can obfuscate a heterogeneous relationship that comes from interactions. PDPs 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. In case of interactions, the ICE plot will give a lot more insight.

A more formal definition: In ICE plots, for each instance 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.

6.2.1 Example

Let's go back to the dataset about risk factors for cervical cancer from Chapter 3.3 and see how each instance's prediction is associated with the feature 'Age'. The model we will analyse is a RandomForest that predicts the probability of cancer for a woman given risk factors. In the partial dependence plot from Chapter 6.1 we have seen that the cancer probability increases around the age of 50, but does it hold true for each woman in the dataset? The ICE plot (Figure 6.5) reveals that the most women's predicted probability follows the average pattern of increase at 50, but there are a few exceptions: For the few women that have a high predicted probability at a young age, the predicted cancer probability does not change much with increasing age.

Figure 6.6 shows an ICE plot for the bike rental prediction (the underlying prediction model is a RandomForest). The data is described in Chapter 3.1. All curves seem to follow the same course, so there seem to be no obvious interactions. That means that the PDP is already a good summary of the relationships of the displayed features and the predicted bike rentals.

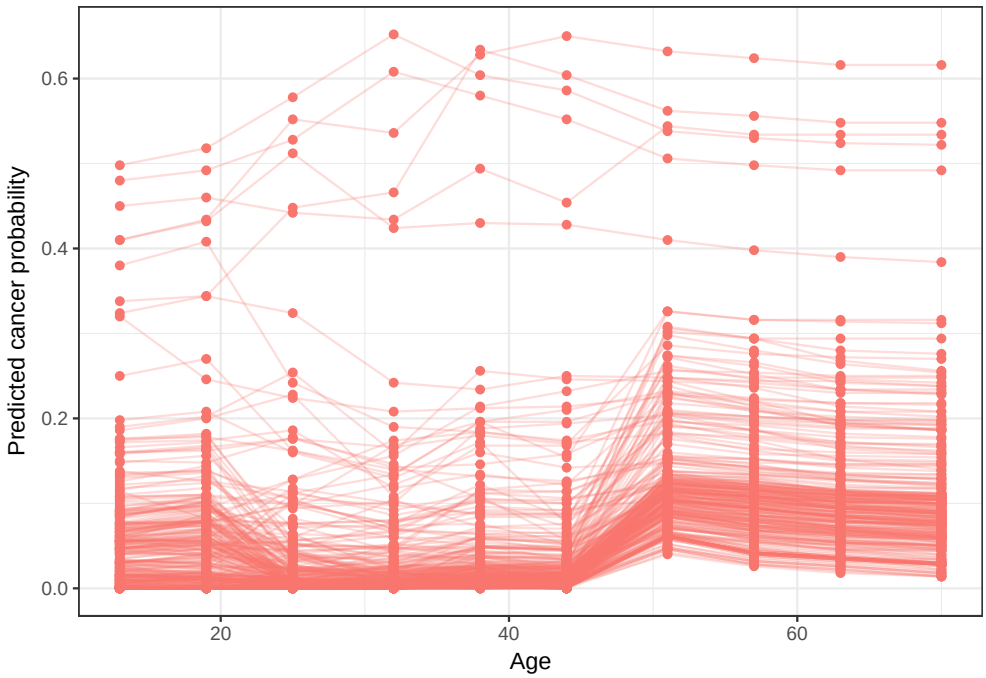


FIGURE 6.5
Individual conditional expectation plot of cervical cancer probability by age. Each line represents the conditional expectation for one woman. Most women with a low cancer probability in younger years see an increase in predicted cancer probability, given all other feature value stay the same. Interestingly for a few women that have a high estimated cancer probability bigger than 0.4, the estimated probability does not change much with higher age.

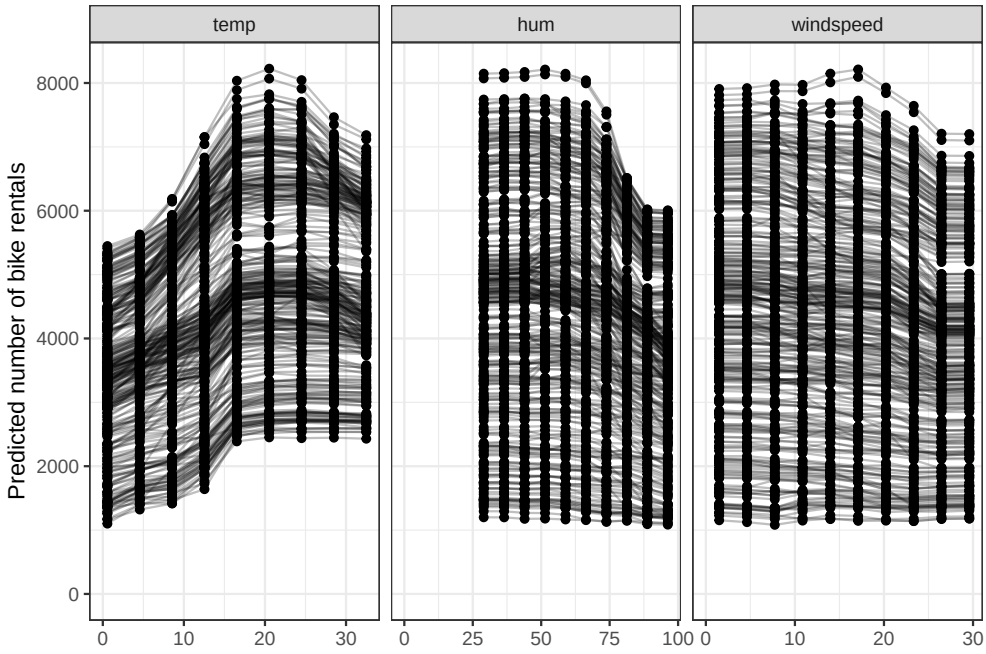


FIGURE 6.6
Individual conditional expectation plot of expected bike rentals and weather conditions. The same effects as in the partial dependence plots can be observed.

6.2.1.1 Centred ICE plot

There is one issue with ICE plots: It can be hard to see if the individual conditional expectation curves differ between individuals, because they start at different $\hat{f}(x)$. An easy fix is to centre the curves at a certain point in x_S and only display the difference in the predicted response. The resulting plot is called centred ICE plot (c-ICE). Anchoring the curves at the lower end of x_S is a good choice. The new curves are defined as:

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

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

6.2.1.2 Example

Taking the plot in Figure 6.5 and centring the lines at the youngest observed age yields Figure 6.7. With the centred ICE plots it is easier to compare the curves of individual instances. This can be useful when we are not interested in seeing the absolute change of a predicted value, but rather the difference in prediction compared to a fixed point of the feature range.

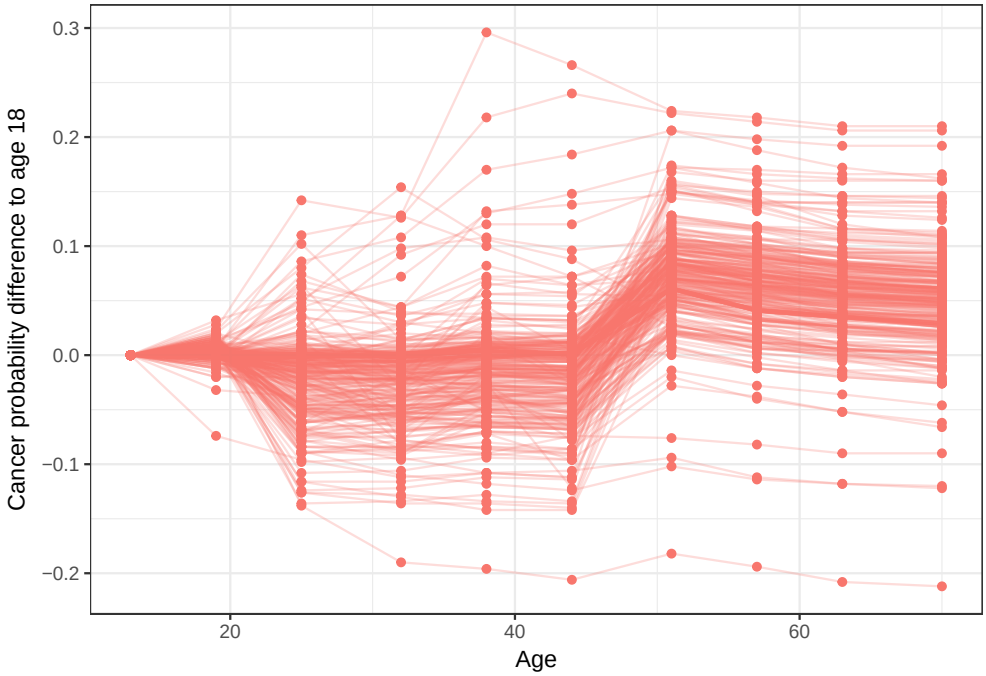


FIGURE 6.7
Centred ICE plot for predicted cervical cancer risk probability by age. The lines are fixed to 0 at age 13 and each point shows the difference to the prediction with age 13. Compared to age 18, the predictions for most instances stay the same and see an increase up to 20 percent. A few cases show the opposite behaviour: The predicted probability decreases with increasing age.

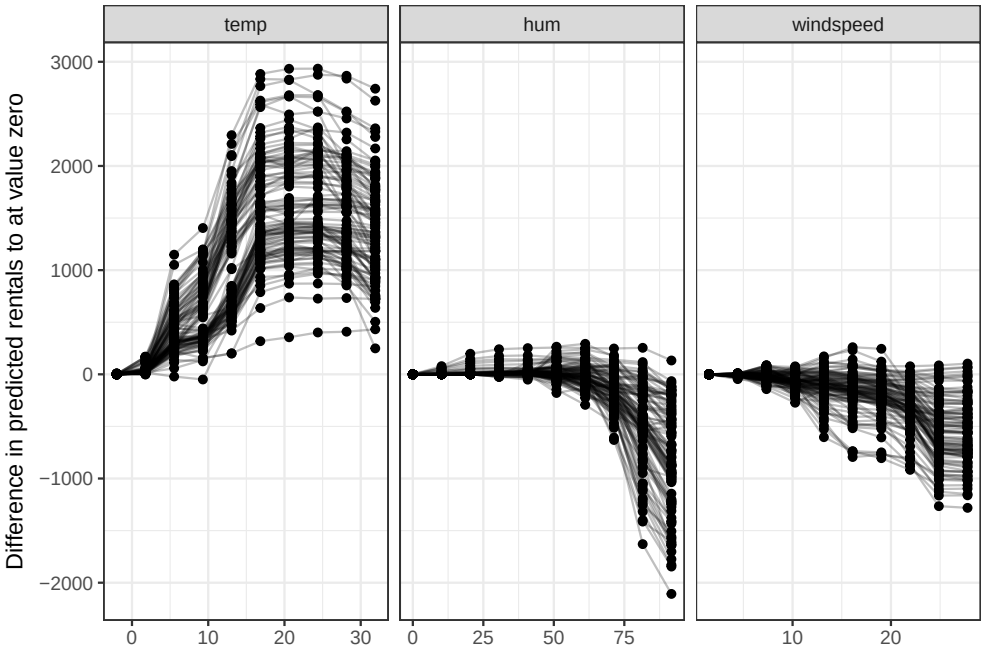


FIGURE 6.8
Centred individual conditional expectation plots of expected bike rentals by weather condition. The lines were fixed at value 0 for each feature and instance. The lines show the difference in prediction compared to the prediction with the respective feature value at 0.

6.2.1.3 Derivative ICE plot

Another way to make it visually easier to spot heterogeneity 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) tell 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 values change 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 instances. 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.

6.2.1.4 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 6.9.

Figure 6.10 shows the derivative ICE plot for the bike rental model.

6.3 Permutation feature importance

The permutation feature importance measurement was introduced for RandomForests by Breiman (2001). It works by measuring what happens to the model performance when you permute each feature. A big loss in performance means a big feature importance. The idea of the permutation of features is per se model-agnostic, only the out-of-bag (=datapoints not used for model fitting) estimation scheme is specific for bootstrapped ensemble methods. Permutation feature importance can be used for any model when a hold-out dataset, instead of out-of-bag samples is used. Of course you could also use the training data, but you risk getting variable importance measures that overfit your data, since the model was already trained on it.

The algorithm (Breiman, 2001), generalised for model-agnostic application, is defined as:

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

1. Estimate performance $Perf$ of $\hat{f}$ on dataset D (e.g. mean squared error)
2. For each feature $j \in 1, \ldots, p$ do
   - For $i \in 1, \ldots, n_{perm}$

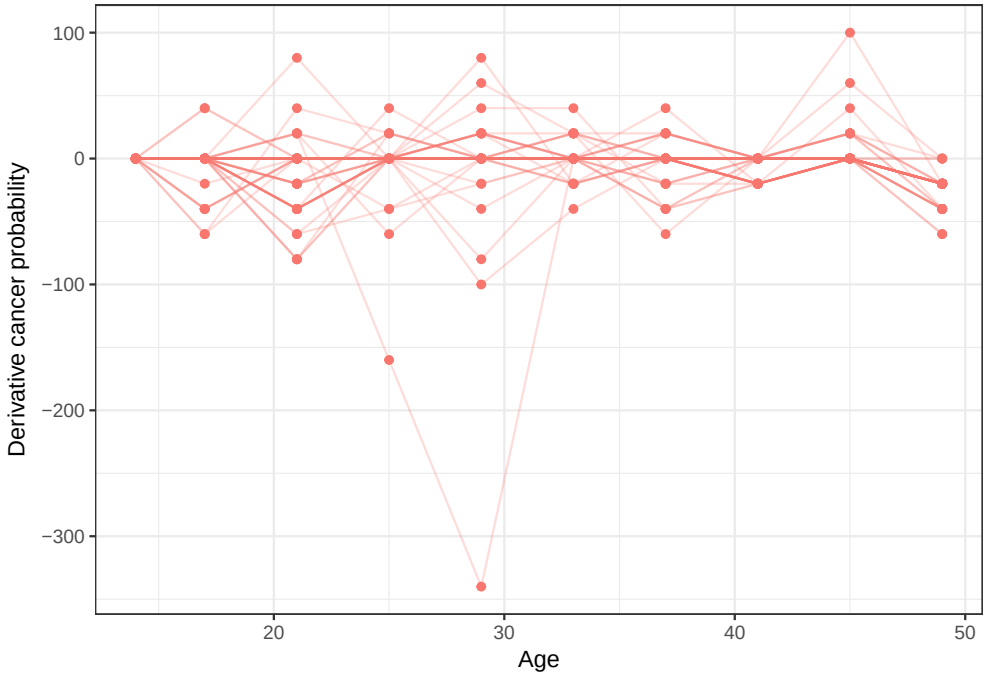


FIGURE 6.9
Derivative ICE plot of predicted cancer probability by age. Between age 14 and the early forties, a few instance see changes in prediction both upwards and downwards, but the majorities derivatives are near zero. Between age 45 and 50, most women's prediction curves have a positive derivative, indicating an increase in predicted cancer probability.

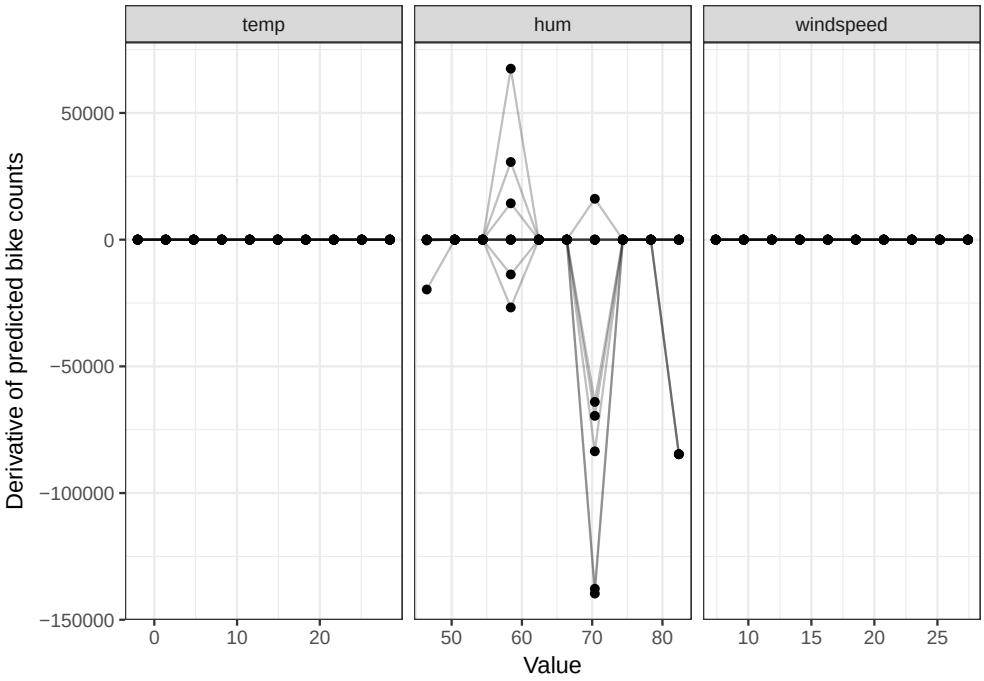


FIGURE 6.10
Derivative individual conditional expectation plot of expected bike rentals and weather conditions.

  - –Generate data $D_{j_{perm}}$ by permuting feature X_j in data D. This breaks the association between X_j and Y.
  - –Estimate performance $Perf_{i,j_{perm}}$ of $\hat{f}$ on dataset $D_{j_{perm}}$
  - –Calculate permutation feature importance $FI_i(X_j) = Perf_{i,j_{perm}} - Perf$
- Calculate mean variable importance for the sample: $FI(X_j) = \frac{1}{n_{perm}} \sum_{i=1}^{n_{perm}} FI_i(X_j)$

3. Sort variables by descending FI.

The feature with the highest importance measure FI makes the most difference in performance globally in your model. I haven't found any paper that generalises permutation feature importance, so that it can be applied model-agnostic. Please drop me a mail if you know about model-agnostic feature importance.

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. his 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 6.11 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 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.

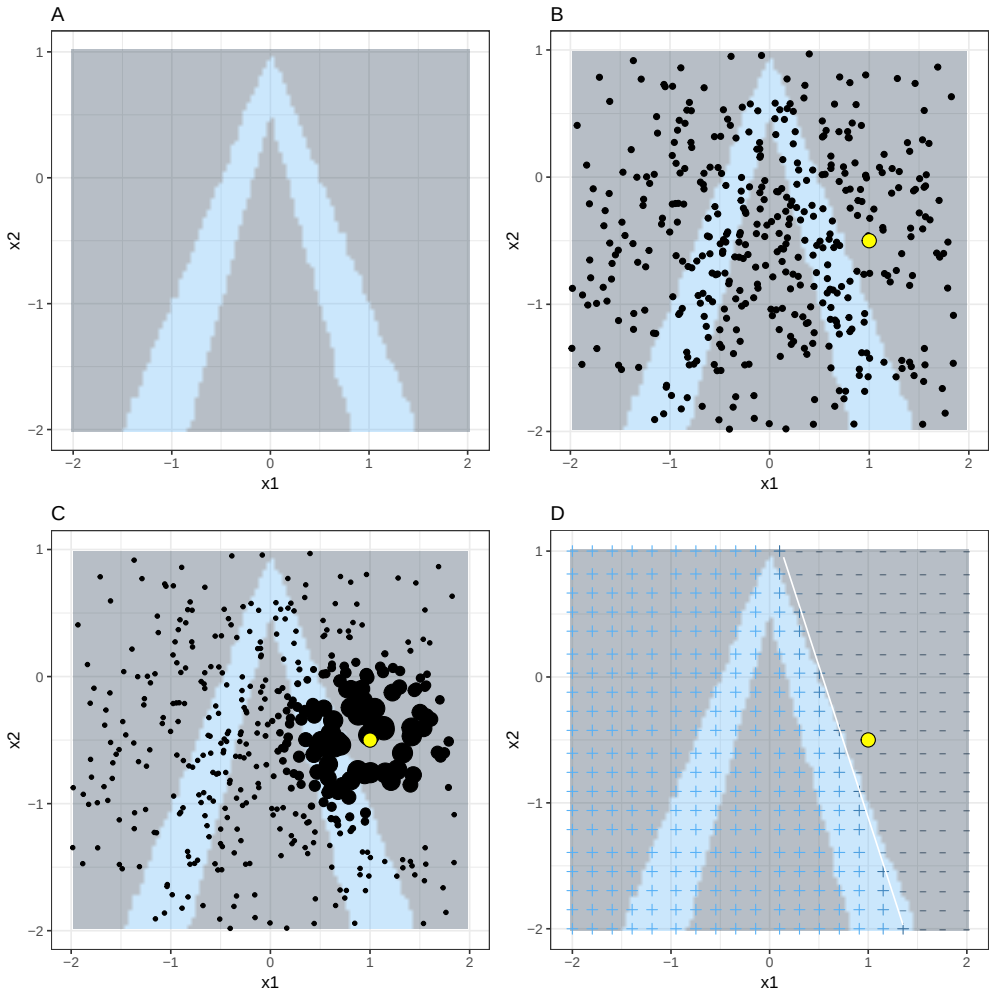


FIGURE 6.11
How LIME sampling works: A) The trainig data has two classes. The most data points have class 0, and the ones with class 1 are grouped in an upside-down V-shape. The plot displays the decision boundaries learned by a machine learning model. In this case it was a Random Forest, but it does 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.

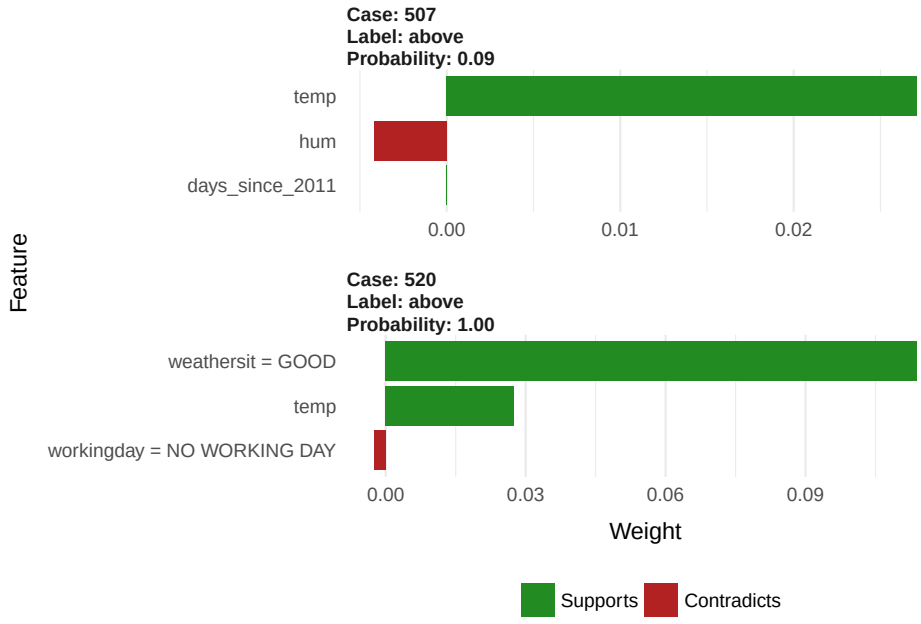


FIGURE 6.12
Explanations for two instances. The first instance got a probability of 0.13 for a match by the Random Forest. The correlation of interests between the participant and the partner is low which reduces the probability of a match for this participant. Also the importance for race of the partner is higher than 6, which lowers the probability of a match. In the second case, the probability for a match is quite high with 0.46. The reasons are the high correlation (>0.43) and that the partner is younger then 25.

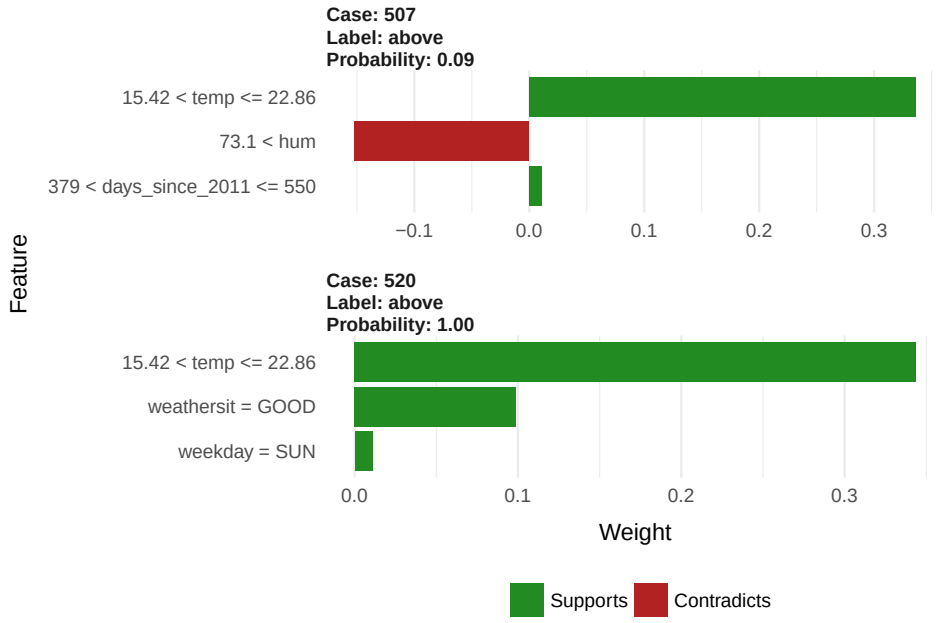


FIGURE 6.13
Explanations for two instances. This time continuous features were turned into categorial features by binning them.

6.4.0.5 Example

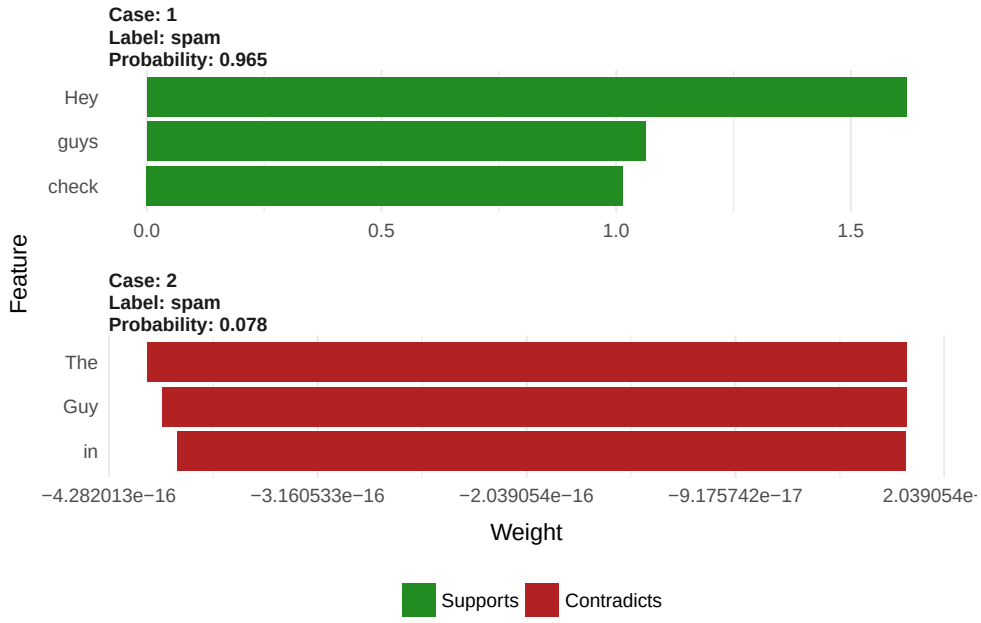
In this example we classify spam vs. ham of YouTube comments. The dataset is described in [#TubeSpam].

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 occurrences 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`".



6.4.0.6 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).