Interpretable machine learning

INTRODUCTION

Story Time

No

What is machine learning

No

Terminology

Algorithm = set of rules that a machine follows to achieve a particular goal

Machine Learning = set of methods that allow computers to learn from data to make and improve predictions

Machine Learning Algorithm = program used to learn a machine learning model from data

Machine learning model = learned program that maps inputs to predictions

Black box model = system that does not reveal its internal mechanisms. In machine learning, “black box” describes models that cannot be understood by looking at their parameters. Model agnostic methods for interpretability treat machine learning models as black boxes, even if they are not

Interpretable machine learning = methods and models that make the behavior and predictions of machine learning systems understandable to humans

Dataset = table with data from which the machine learns (features + target)

Instance = row within dataset

Features = inputs used for prediction / classification

Target = information to be predicted

Machine learning task = combination of dataset with features and target

Prediction = guess by machine learning model

INTERPRETABILITY

**Importance of interpretability**

* Human curiosity and learning
* Find meaning in the world
* Goal of science = knowledge gain not just problem solving
* Safety measures
* Detecting bias
* Social acceptance
* Explanations are used to manage social interactions
* Debugging and audition
* Can check: fairness, privacy, reliability / robustness, causality, trust
* We don’t need interpretability if the model has no significant impact, if the problem is well studied, or if you know people will manipulate the system with it.

**Taxonomy of interpretability methods**

* Intrinsic or post hoc?
  + Intrinsic: interpretability is achieved by restricting the complexity of the model. Model is interpretable because its structure is simple and understood.
  + Post hoc: interpretability is achieved by applying methods that analyze the model after training. Examples: permutation feature importance.
* Result of the interpretation method
  + Feature summary statistic: feature importance, pairwise feature interaction strengths
  + Feature summary visualization: partial dependence plots (show feature and average predicted outcome)
  + Model internals: e.g. learned weights, can be both model internals and summary statistics for the feature….
  + Data point: methods that return already existent / newly created data points
    - Counterfactual explanations (to explain the prediction of a data instance, the method finds a similar data point by changing some of the features for which the predicted outcome changes in a relevant way)
    - Identification of prototypes of predicted classes
  + Intrinsically interpretable model: approximate a black-box model with an interpretable model
* Model specific or model agnostic?
* Local or global? (Individual prediction or general behavior)

Scope of interpretability

**Algorithm Transparency:**

* How algorithm learns from data
* What kind of relationship it can learn
* E.g.: convolutional neural networks: learns edge detectors and filters on lowest layers.
* Requires knowledge of the algorithm only – not of the data

Global, Holistic Model Interpretability

* How does the trained model make predictions?
  + Holistic view of tis features, weights, parameters, structures
* Which features are important, what kind of interactions between them
* Tough to do really lol

Global Model interpretability on a modular level

* How do parts of the model affect predictions?
* Not all models are interpretable at a parameter level
* Interpretation of a single weight always comes with the footnote that the other input features remain at the same value, which is not the case with many real applications.

Local interpretability for a single prediction

* Why did a model make a certain prediction for an instance?
* Locally the prediction might only depend linearly or monotonically on some features, rather than having a complex dependence on them.

Local interpretability for a group of predictions

* Why did the model make specific predictions for a group of instances? Can aggregate individual explanations

Evaluation of Interpretability

* Application level evaluation (real task): end user test
* Human level evaluation (simple task): carried out with non-experts
* Function level evaluation (proxy task): not necessarily carried out by humans (e.g. for decision trees, a shorter tree = more interpretable)

Properties of Explanations

* An explanation usually relates the feature values of an instance to its model prediction in a humanly understandable way (could also just be a set of data instances – in the case of k-nearest neighbor model)
* Properties of Explanation methods:
  + Expressive power: IF-THEN rules, decision trees, weighted sum, natural language
  + Translucency: how much the explanation method relies on looking into the machine learning model (e.g. its parameters): linear regression model = highly translucent
  + Portability: range of machine learning models with which the explanation method can be used (surrogate models = highly portable, methods specific to 1 model = lowly portable)
  + Algorithmic complexity:
* Properties of individual explanations:
  + Accuracy: how well does an explanation predict unseen data?
  + Fidelity: how well does the explanation approximate the prediction of the black box model? Important for explanation.
  + Consistency: how much does an explanation differ between models that have been trained on the same task and that produce similar predictions?
  + Stability: how similar are the explanations for similar instances?
  + Comprehensibility: how well do humans understand the explanations?
  + Certainty: does the explanation reflect the certainty of the machine learning model?
  + Degree of importance: how well does the explanation reflect the importance of features or parts of the explanation?
  + Novelty: does the explanation reflect whether a data instance to be explained comes from a “new” region far removed from the distribution of training data? High novelty = more likely that model will have low certainty due to lack of data
  + Representativeness: how many instances does an explanation cover?

Human-friendly Explanations

What is an Explanation?

* Why, how?

What is a good explanation?

* Contrastive: What it means for interpretable machine learning: compare differences e.g. to another prediction
* Selected: short, 1 to 3 reasons why even if it’s more complex (LIME method)
* Social: depends on context, target audience, specific application
* Focus on abnormal : abnormal causes are considered to be “good” explanations (e.g. abnormal features)
* Truthful: true in reality and in other situations, i.e. fidelity
* Consistent with prior beliefs of the explainee: confirmation bias (humans tend to ignore information that is inconsistent with their prior beliefs). Tough to deal with this.
* General and probable: cause that explains many events is general and a good explanation (opposite of the abnormal cause…)

DATASETS

Bike rentals (regression)

Youtube spam comments (text classification)

Risk factors for cervical cancer (classification)

INTERPRETABLE MODELS

* Linear regression, logistic regression, decision tree, ruleFit, Naïve Bayes, knn
* Monotonicity is useful for interpretation because it makes it easier to understand a relationship

**LINEAR REGRESSION**

Y=b0 + b1x1 + … + e

* Error is assumed to follow gaussian distribution
* Weight optimization
* Linearity: estimation procedure is simple and understandable
* Estimated weights come with **confidence intervals:** a range for the weight estimate that covers the “true” weight with a certain confidence. A 95% CI = if we repeat the estimation 100 times with newly sampled data, the CI would include the true value in 95 out of 100 cases given that linear regression model is the correct model for the data
* Assumptions necessary for use of linear regression:
  + **Linearity**: because the model forces the linear relationship.
    - If nonlinear association of feature with target: add interaction terms, or use regression splines
  + **Normality**
    - Target outcome given features follows normal distribution (else the estimated Cis are invalid!)
  + **Homoscedasticity**: constant variance of error terms across the entire feature space, although this is usually violated
  + **Independence**: assume that each instance is independent of any other
  + **Fixed features**: features are free of measurement errors (unrealistic assumption) and are treated as given constants
  + **Absence of multicollinearity**: feature effects are additive 🡪 can’t know which of the correlated features is responsible for an effect!

Interpretation:

* Weight interpretation depends on feature type
  + Numerical: increasing the numerical feature by 1 unit changes the estimated outcome by its weight
  + Binary: going from 0 to 1 changes outcome by feature weight
  + Categorical: use one hot encoding to change to binary
  + Intercept beta 0: the feature weight for the constant feature (always 1 for all instances): for an instance with all numerical feature values at zero and categorical values at reference categories, the model prediction is the intercept weight.
  + R-squared measurement: 1-SSE/SST (squared sum of error terms, squared sum of data variance, respectively) – the ratio = how much of the data is explained by the variance\
    - Increases with number of features in the model, even if they do not contain any information about the target value 🡪 Adjusted R-squared
  + Feature importance: absolute value of its t-statistic = estimated weight scaled with its standard error (beta / standard error (beta)): the importance of a feature increases with increasing weight; and the more variance the weight has, the less important the feature is

Example:

Visual interpretation:

* Weight Plot
* Effect plot: weight per feature times the feature value of an instance (use with boxplots)

Explain individual predictions

Encoding of categorical features

* Treatment coding: the weight per category is the estimated difference in the prediction between the corresponding category and the reference category
* Effect coding: the weight per category is the estimated y-difference from the corresponding category to the overall mean (given all other features are zero or the reference category)
* Dummy coding: the weight per category is the estimated mean value of y for each category (given all other feature values are zero or the reference category). Intercept is omitted so that a unique solution can be found using linear model weights.

Do linear models create good explanations?

* In short no
* Are contrastive, but the reference point usually is artificial and meaningless. Except if all numerical features are mean-centered and all categorical features are effect coded
* Selectivity: use less features

Sparse linear models:

What if you have a lot of features? Tough to have interpretability… Methods to have sparse model:

* Lasso : least absolute shrinkage and selection operator; when applied in a linear regression model, performs feature selection and regularization of the selected feature weights.
  + Adds L1 norm penalty to the weights optimization problem. Since L1 is used, many of the weights receive an estimate of zero, and the others are shrunk.
  + Additional parameter lambda is the weight of the regularization and is fined tuned through cross-validation usually. For large lambdas, most weights will turn to zero.
  + Need to have features standardized of course!
* Manually-selecting features (reduce number of features)
* Univariate selection: e.g. with correlation: only consider features that exceed a certain threshold of correlation between the feature and target.
* Step-wise methods:
  + Forward selection: fit model with 1 feature, select best model (according to r-squared), add feature, select model which works best, etc
  + Backward selection: same but starting with all features and working your way back

Advantages: accepted for inference and predictive models, and you are guaranteed to find optimal weights.

Disadvantages: all nonlinear problems… and also not great for predictive performance (too simple for real life situations which are often complex). Also 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.

**LOGISTIC REGRESSION**

Extension of the linear regression model for classification problems.

* Issues with linear regression: will not spit out probabilities, it treats the classes as numbers (0, 1) and fits the best hyperplane (a line, for a single feature) that minimizes the distances between the points and the hyperplane. It is simply interpolating. Therefore you can get values below zero and above one. There is also no meaningful threshold at which you can distinguish one class from the other. And what about problems with multiple classes? You can label them class 2, 3 but this number gives a meaningful order to the classes.
* Logistic regression uses the logistic function to squeeze the output of a linear equation between 0 and 1. Logistic function is 1/(1+exp(-n)). Just use logistic function on linear regression to output probabilities. Use 0.5 as a threshold.

Interpretation:

* The weights do not influence the probability linearly. If we look at P(y=1)/P(y=0) for a binary classification problem (hence p(y=0) = 1-p(y=1)), and take log of this ratio, it is called **log odds** (probability of event divided by probability of no event, wrapped in log function). Log odds equals the linear formula. Therefore odds = exponential of linear formula.
* Then we can compare what happens when we increase one of the feature values by 1, look at the ratio of the two odds.
* If odds ratio is 2, the probability of event is twice that of no event. If the weight (log-odds ratio) is 0.7then increasing the respective feature by 1 multiplies the odds by exp(0.7), which is 2, and the odds change to 4.
* Categorical features: one-hot encoding
* Intercept beta 0: when all numerical features are zero and the categorical features are at the reference category, the estimated odds are exp(beta 0). It is usually irrelevant to interpret this weight.

Example

Advantages:

Disadvantages: interactions must be added linearly (as in linear regression). Interpretation of the weights is more difficult as they are multiplicative (not additive as in linear regression). Can suffer from complete separation

**GLM, GAM AND MORE**

* feature interaction 🡪 add interactions manually
* target outcome y given the features does not follow a Gaussian distribution 🡪 GLMs
* true relationship between the features and y is not linear🡪 GAMs or feature transformation, or categorization (binning)

**GLMs:**

* keep the weighted sum of features
* allow non Gaussian outcome distributions
* connect the expected mean of this distribution and the weighted sum through a possibly nonlinear function
  + logistic regression: model assumes a Bernoulli distribution for the outcome and the link function is the logistic function
* the probability distribution needs to be from the exponential family
* interpretation: since all weights are in the exponential function, the effect is not additive but multiplicative
* any link function that is not the identity makes interpretation tough

**GAMs:**

* non linearity added to GLM
* the outcome can be modeled by a sum of arbitrary functions of each feature
* how to learn nonlinear functions: splines

**Feature transformation:**

* usually use log, square root, square, exponential

**Feature categorization:**

* needs more data
* more likely to overfit
* unclear how to discretize the feature meaningfully

**Solving other issues**

* data violates the assumption of being independent and identically distributed 🡪 mixed models or generalized estimating equations
* model has heteroscedastic errors 🡪 robust regression
* outliers strongly influence model 🡪 robust regression
* I want to predict the time until an event occurs (some time there is not enough time to observe the event, so there is censored data) 🡪 parametric survival models, cox regression, survival analysis
* Outcome to predict is a category (more than 2 categories) 🡪 multinomial regression (is a GLM)
* Predict ordered categories 🡪 proportional odds model
* Outcome is a count 🡪 Poisson regression (is a GLM)
* Not sure which features need to be included 🡪 causal inference, mediation analysis
* Missing data 🡪 multiple imputation
* Want to integrate prior knowledge into models 🡪 Bayesian inference

**Decision Tree**

* Split data according to cutoff values. Through splitting, multiple subsets are created (leaf nodes), with intermediate subsets (split nodes / internal).
* Regression and classification
* Subsets: take a feature and determine which cut-off point minimizes the variance of y for a regression task or the Gini index of the class distribution of y for classification task
  + Variance: tells us how much the y values in a node are spread around their mean value
  + Gini index: tells us how impure a node is: if all classes have the same frequency, the node is impure, if only one class is present then the node is maximally pure
  + Best cut off point makes the two resulting subsets as different as possible with respect to the target outcome
* Interpretation: easy
* Feature importance: one way: go through all the splits for which the feature was used and measure how much It has reduced the variance / Gini index compared to the parent node. Tells you how much a feature helped to improve the purity of all nodes.
* Advantages:
  + Capturing interactions between features and data
  + Natural visualization
  + Interpretation is simple / good explanations are possible
* Disadvantages:
  + Fail to deal with linear relationships
  + Lac of smoothness: small change in input can have big change on output
  + Unstable: if a parent split changes every child changes 🡪 not great for confidence
  + Number of terminal nodes increases quickly with depth…

**Decision rules:**

* If then statement
* Support / coverage = percentage of instances to which the condition of a rule applies
* Accuracy / confidence = measure of how accurate the rule is in predicting the correct class for the instances to which the condition of the rule applies
* Rules can overlap
* Combining rules:
  + Decision list: ordered rules (if condition of first rule is true, use first rule prediction, if not go to second)
  + Decision set: rules are either mutually exclusive or there is a strategy for resolving conflicts (e.g. majority voting)
* Easy to interpret
* Fast
* Only select relevant features
* Only classification
* Features have to be categorical
* Bad in describing linear relationships

RuleFit:

* Learns a sparse linear model with original features and also a number of new features that are decision rules (new features to capture interactions between original features)
* 1) create rules from decision trees
* 2) fit linear model with original features and new rules as input

**Naïve Bayes Classifier**

* For each feature, it calculates the probability for a class depending on the value of the feature.
* Conditional probability model
* Assumes independence (strong / naïve assumption)

**K-Nearest Neighbors**:

* Assigns the most common class of the nearest neighbors of an instance.
* Which distance measurement, which k?
* Only interpretable if there are very few features

MODEL-AGNOSTIC METHODS

Requirements for a model-agnostic explanation:

* Model flexibility: works for random forests or for deep neural networks, etc…
* Explanation flexibility: not being limited to 1 form of explanation
* Representation flexibility: not being limited to 1 type of feature representation

**Partial dependence plot:**

* Shows marginal effect one or two features have on the predicted outcome of a ML model.
* Xs are the features for which the partial dependence function should be plotted
  + Usually just 1 or 2
  + Those for which we want to know the effect on the prediction
* Xc are the other features used in the machine learning model f
  + Combined xs and xc make up the total feature space
* Marginalize the ML model output over the distribution of the features xc, so that the function shows the relationship between the features in set S (those we are interested in) and the predicted outcome.
  + **Marginal probability mass function:** Given a known joint distribution of two discrete random variables, say, X and Y, the marginal distribution of either variable--X for example--is the probability distribution of X when the values of Y are not taken into consideration. This can be calculated by summing the joint probability distribution over all values of Y.
* Estimate a partial function by calculating averages in the training data. Sum over all instances of the dataset for each partial function.
* Advantages:
  + Intuitive
  + Interpretation is clear
  + Easy to implement
  + Causal interpretation
* Disadvantages:
  + Max number of features in PDP, realistically, is 2.
  + What about feature distribution? Use a rug.
  + Assumption of independence of features for which the partial dependence is computed are not correlated with other features: e.g. predict speed with weight and height features, for the partial dependence of height we need to assume that other features (weight) are not correlated to height.
    - Use accumulated local effect plots (work with conditional instead of marginal distributions)
  + Heterogeneous effects might be hidden: PDP shows average marginal effects. PD curve of a feature that is positively associated with half of the data points and negatively associated with the other half would be a straight line… Does not specific on specific instances but averages
    - Individual conditional expectation curves

**Individual Conditional Expectation (ICE):**

* Display one line per instance that shows how the instance’s prediction changes when a feature changes.
* Basically a PDP is the average of the lines of an ICE plot.

**Accumulated Local Effects**

* Describe how features influence the prediction of a ML model on average.
* Why PDP can be wrong: ML model that predicts value of house depending on number of rooms and size of living area. The PDP algorithm for the effect of living area on predicted value:
  + Select feature (living area)
  + Define grid
  + Per-grid value: replace feature with grid value, and average predictions
  + Draw curve
  + To compute the feature effect of feature 1 (30m2), the PDP replaces feature 1 of all instances with this value and computes the mean… for the first grid value of the PDP (say the lowest living area, 30 m2) we replace all instances by 30m2, even the houses with the highest number of rooms (e.g. 10 rooms). PDP includes these weird new datapoints in the feature effect estimation.
* How to get a feature effect estimate that respects the correlation of features? We can average over the conditional distribution of the feature (at value x1, we average predictions of instances with a similar x1 value) = marginal plots (but really using conditional distribution, not marginal).
  + Does not solve issue: if we average the predictions of all houses of about 30m2, we estimate the combined effect of living area and of number of rooms, because they are correlated. Suppose that only the number of house had an effect on house value. The m-plot would still show that the size of the living area increases the predicted house value, since the number of rooms is positively correlated with the living area…
* ALE plots: compute the differences in predictions, instead of the averages.
  + For effect of living area at 30m2: use all houses with about 30m2, get model predictions pretending these houses were 31m2 minus predictions pretending they were 29m2.
  + The use of difference blocks the effect of other features.
* Recap:
  + PDP: let me show you what the model predicts on average when each data instance has the value v for that feature. I ignore whether the value v makes sense for all data instances.
  + M-plot: let me show you what the model predicts on average for data instances that have values close to v for that feature. The effect could be due to that feature, but also due to correlated features.
  + ALE plots: let me show you how the model predictions change in a small window of the feature around v for data instances in that window.
* Effect: is an ALE estimate of -2 is at x\_j=3, it means that when the j\_th feature has value 3, the prediction is lower by 2 compared to the average prediction.
* Advantages: fast, interpretable, centered at zero, can be 2D.
* Disadvantages: not accompanied by ICE curves (so no heterogeneity investigation), can be shaky with too many intervals, less accurate with too few, interpretation remains difficult when features are strongly correlated.
* If two features are strongly correlated, it makes sense to only analyze the effect of changing both features together, not in isolation…

**Feature Interaction**

* Estimating the interaction strength: measure how much of the variation of the prediction depends on the interaction of features (H-statistic)
  + Basically: calculate the variance of the output of the partial dependence (to measure the interaction between two features) or of the entire function (to measure the interaction between a feature and all other features). The amount of variance explained by the interaction (difference between observed and no-interaction partial dependence) is used as interaction strength statistic.
  + Assume that 2 features don’t interact 🡪 partial dependence function is the sum of the partial dependence functions of single features. This is the “no interaction” PD. Compare it to the actual PD.
* Advantages: meaningful, dimensionless, detects all kind of interactions
* Disadvantages: computationally expensive, results can be unstable

**Permutation feature importance**

* Measure the increase in prediction error of the model after we permuted the feature’s values (this breaks the relationship between feature and true outcome).
* A feature is important if shuffling its values increases the model error (in this case the model relied on the feature for prediction)
* Some extreme examples: what if no feature are important to the prediction? Then the permutation would yield “good” results…
* How much the model relies on each feature for making predictions 🡪 use training data
* How much the feature contributes to the performance of the model on unseen data 🡪 test data
* Advantages: takes into account all interactions, global insight, comparable across different problems, does not require retraining the model.
* Disadvantages: use test or train data to compute feature importance? Need access to true outcome, results might vary, adding a correlated feature can decrease the importance of the associated feature (splits the importance between both)

**Global surrogate:**

* Interpretable model trained to approximate the predictions of a black-box model.
* Model agnostic
* Algorithm:
  + Select dataset
  + Get predictions of black box models for this dataset
  + Select interpretable model
  + Train interpretable model on dataset and its predictions
  + Measure how well the surrogate model replicates the predictions of the black box model
* Advantages: flexible, intuitive
* Disadvantages: draw conclusions about the model, not about the data (surrogate model never sees the real outcome), etc…

**Local Surrogate (LIME):**

* Focuses on training local surrogate models to explain individual predictions
* Surrogate has to be a good approximation locally, but not globally
* Algorithm:
  + Select instance of interest for which you want an explanation of its black box prediction
  + Perturb dataset and get black box predictions for new points
  + Weight new samples according to their proximity to the instance of interest
  + Train weighted interpretable model on dataset with variations
  + Explain prediction by interpreting local model
* Advantages: fidelity measures is good
* Disadvantages: instability

**Scoped rules (anchors):**

* Explains individual predictions of any black=box classification model by finding a decision rule that “anchors” the prediction sufficiently (a rule anchors a prediction if changes in other feature values do not affect the prediction).
* Rules are if-then; e.g. if sex=female, and class=first, then predict survived = true with precision 97% and coverage 15%.
* Coverage = to which other, possibly unseen instances they apply
* Efficient, easy to interpret
* Requires discretization usually, coverage can be undefined in certain domains

**Shapley values**

* A prediction can be explained by assuming that each feature value of the instance is a “player” in a game where the prediction is the payout
* Example: predict house price, for size, floor, park nearby and cats banned.
  + 300k for size 50m2 2nd floor, park nearby and cats banned
  + Average prediction for all apartments is 310k
    - How much has each feature value contributed to the prediction compared to the average prediction?
  + Game = prediction task for a single instance of the dataset
  + Gain = actual prediction for this instance – average prediction of all instances
  + Players = feature values of the instance that collaborate to receive the gain (predict a certain value)
  + Here: feature values = park-nearby, cat-banned, area-50, floor-2, gain=300k, we want to explain the -10k in difference
  + Evaluate cat-banned contribution when it is added to a coalition of park-nearby and size-50. Simulate that only park-nearby, cat-banned, size-50 are in a coalition by randomly drawing another apartment from the data and using its value for the floor feature (e.g. floor-1). Then we predict the price of the house with this combination (e.g. 310k). In a second step, we remove cat-banned from the coalition and replace it with a random value of the cat allowed/banned feature from the randomly drawn apartment (e.g. cat-allowed). We predict the house price for the coalition of park-nearby and size-50 (e.g. 320k). The contribution of cat-banned was therefore 310k-320k=-10k.
  + Repeat sampling step and average contributions
  + Repeat this for all possible coalitions
* Shapley value = average marginal contribution of a feature value across all possible coalitions
* The Shapley value of a feature value is its contribution to the payout, weighted and summed over all possible feature value combinations.
* Advantages: allows contrastive explanations, and the difference between the prediction and the average prediction is fairly distributed among the feature values of the instance.
* Disadvantages: computing time, can be misinterpreted, always uses all features (humans prefer only a subset usually). Can include unrealistic data instances.

**Shapley Additive Explanations (SHAP):**

* Connect LIME and Shapley values.
* A player can be a single or a group of features…
* Goal: explain prediction of an instance x by computing the contribution of each feature to the prediction
  + Tells us how to fairly distribute the payout / prediction among the features. A player can be an individual feature or a group of features (e.g. super pixel instead of a simple pixel)
  + SHAP: represented as an additive feature attribution method:   
     g(z’) = phi\_0 + SUM(phi\_j\*z’\_j)
    - G explanation model
    - Z’ is 0 or 1 is the coalition vector (entry of 1 means the feature is present, 0 means it is absent)
    - Phi is the feature attribution for feature j = shapley values
* Difference to LIME:
  + Weighting of the instances in the regression model:
    - LIME weighs the instances according to how close they are to the original instance. The more 0s in the coalition vector, the smaller the weight.
    - SHAP weighs the sampled instances according to the weight the coalition would get in the Shapley value estimation. Small coalitions (few 1s) and big ones (lots of 1s) get the largest weights. You want to avoid coalitions with half the features: you don’t learn as much as with just a few or all but a few…
      * Use the SHAP kernel to weigh accordingly.
* TreeSHAP (vs KernelSHAP): for tree-based ML models. Can produce unintuitive feature attributions, but way faster.

Examples:

* Visualize feature attributions such as Shapley values as forces: each value increase or decreases the prediction.
* **SHAP feature importance**: features with large absolute Shapley values are important.
* **SHAP dependence plot**: pick a feature, and for each data instance plot a point with the feature value on the x-axis and the corresponding Shapley value on the y-axis
* **SHAP interaction values**: interaction effect is the additional combined feature effect after accounting for the individual feature effects.
  + subtract the main effect of the features so that we get the pure interaction effect after accounting for the individual effects. Average over all possible feature coalitions.
  + For 1 instance: matrix M\*M where M is the number of features.
  + To visualize: basically a dependence plot with colors showing the strength of interaction with another variable
* **SHAP clustering**: cluster on shapley values of each instance: cluster by explanation similarity.

EXAMPLE BASED EXPLANATIONS

Select particular instances of dataset to explain model behavior, not by creating summaries of features.

* only makes sense if data can be represented in a human-understandable way.

**Counterfactual Explanations**:

* if X had not occurred, Y would not have occurred.
* A counterfactual explanation of a prediction describes the smallest change to the feature values that changes the prediction to a predefined output.
  + Counterfactuals do not have to be actual instances from the training data, but can be a new combination of feature values.
  + Human friendly explanations: are contrastive to the current instance and because they are selective, meaning they usually focus on a small number of feature changes.
    - Suffer from Rashomon effect: there can be different explanations for one effect, and these explanations can contradict each other (“multiple truths”)
* Example: Airbnb loan, you expect 1k but your model says your apartment can be rented for 900. You look at features e.g. pets allowed, and tell yourself ok what is the smallest change to the features that I can apply that will change the model prediction? And if you change window insulation or pets allowed maybe you can go over 1k.
* Requirements:
  + Counterfactual instance produces the predefined prediction as closely as possible
    - You want to look for a counterfactual where the predicted probability of the rare class is increased to 10% instead of the current 2%; your minimum changes to the features need to make it really close to 10%, not much less or much more…
  + Counterfactual should be as similar as possible to the instance regarding feature values
    - Modify as little as possible so you know what you are acting on
  + Counterfactual instance should have feature values that are likely
    - E.g. for an apartment, don’t use negative number of rooms, or with 150 rooms
* Algorithm:
  + Select an instance x to be explained, the desired outcome y’, a tolerance epsilon, a low initial value of lambda
  + Sample a random instance as initial counterfactual
  + Optimize the loss with the initially sampled counterfactual as starting point. The loss is lambda \* quadratic distance between model prediction for the counterfactual x’ and the desired outcome y’, plus the distance between the instance x to be explained and the counterfactual x’.
  + While the quadratic distance between the model prediction for the counterfactual x’ and the desired outcome y’ is bigger than epsilon, increase its weight lambda, continue the loss optimization with the current counterfactual as the starting point.
* Advantages: does not require access to the data or to the model, easy, clear
* Disadvantages: multiple counterfactual explanations per instance, no guarantee that you find a counterfactual instance for a given tolerance, does not handle categorical features

**Adversarial Examples:**

An instance with small, intentional feature perturbations that cause a machine learning model to make a false prediction = counterfactual examples aiming to deceive the model (not interpret it)

* Mostly used in NN (GANs for example)
* Fast gradient sign method: to generate adversarial images
  + Original image x is manipulated by adding or subtracting a small error to each pixel. Whether we add or subtract the error depends on the sign of the gradient for a pixel. Adding errors in the direction of the gradient means that the image is intentionally altered so that the model classification fails.
  + The sign of the gradient is positive if an increase in pixel intensity increases the loss; negative if a decrease in pixel intensity increases the loss.
    - This occurs when NN treats relationship between input pixel intensity and class score linearly. Happens in NN architectures which favor linearity: LSTMs, maxout networks, NN using ReLU activation units.
* 1 pixel attack: just look for which pixel would change the outcome: look for a modified example x’ which comes close to the original image x but changes the outcome.
  + Using differential evolution to find out which pixel is to be changed and how: population of individuals called candidate solutions recombines generation by generation until a solution is found. Each candidate solution encodes a pixel modification. The search starts with N candidates solutions and creates a new generation of candidate solutions from the parent generation using a certain formula.
* Adversarial patch: completely replace a part of the image with a patch that can take on any shape. Optimize patch so it works in many different solutions.
* Robust adversarial examples: create adversarial example that is adversarial over transformations! Other methods don’t work if you rotate the image for example, this one does.
* Black box attack:
  + Start with a few images that come from the same domain as the training data (e.g. digits for a digit classifier)
  + Get predictions of the current set of images from the black box
  + Train a surrogate model on the current set of images
  + Create a new set of synthetic images using a heuristic that examines for the current set of images in which direction to manipulate the pixels to make the model output have more variance. Repeat

**Prototypes and criticisms:**

* Prototype = data instance that is representative of all the data
* Criticism = data instance that is not well represented by the set of prototypes
* Prototypes are selected manually to cover the centers of the data distribution, the criticisms are points in a cluster without a prototype.
* Usually use clustering algorithms to find prototypes.
* To find both prototypes and critics: use **MMD-critic** method. Basically: select number of prototypes and criticisms, find prototypes so that the distribution of the prototypes is close to the data distribution; find criticisms so that they are where the data distribution is different from the prototype distribution
  + **MMD**: maximum mean discrepancy: measures discrepancy between two distributions
* Use with ML:
  + Find prototypes and criticisms using MMD-critic.
  + Train ML model
  + Predict outcomes for the prototypes and criticisms with trained model
  + Analyze predictions! Where was algorithm wrong, etc…

Influential instances

* A training instance is influential when its deletion from the training data considerably changes the parameters or predictions of the model. E.g. if you take 10 people and ask them their salary.
  + **Deletion diagnostics**: remove 1 of the data points and see how much it affects the result (9 people), do it for each data point.
    - DFBETA, Cook’s distance, but need to retrain the model each time.
      * DFBETA: effect of deletion on model parameters
      * Cook’s distance: effect of deletion on model outcome
  + **Influence functions**: derive the above mathematically, how the mean value can be influenced
    - Need a loss function that is twice differentiable w.r.t. its parameters
    - Instead of deleting the instance: upweight the instance by a small step.
      * Approximate the loss around the current model parameters
* Outliers: can be influential if it influences the model
* Advantages: One of the best debugging tools in ML, model agnostic for deletion diagnostics
* Disadvantages: expensive, no clear cutoff for influence measure. Only takes into account deletion of single instances….

NEURAL NETWORK INTERPRETATION

Learned Features

* Deep NN learn high-level features in hidden layers: usually
  + First convolutional layers learn features such as edges and simple textures
  + Later convolutional layers: learn feature such as patterns and complex textures
  + Last convolutional layers: learn objects or parts of objects
  + FC layers: connect to prediction class

**Feature visualization:**

For a unit of NN, find the input that maximizes the activation of that unit. Unit = individual neurons, channels / feature maps, entire layers, final class probability, i.e. it depends.