THE UNIVERSITY *of* EDINBURGH

Week 11: Overview, Explainable AI, and Revision

Dr Sara Wade
sara.wade@ed.ac.uk

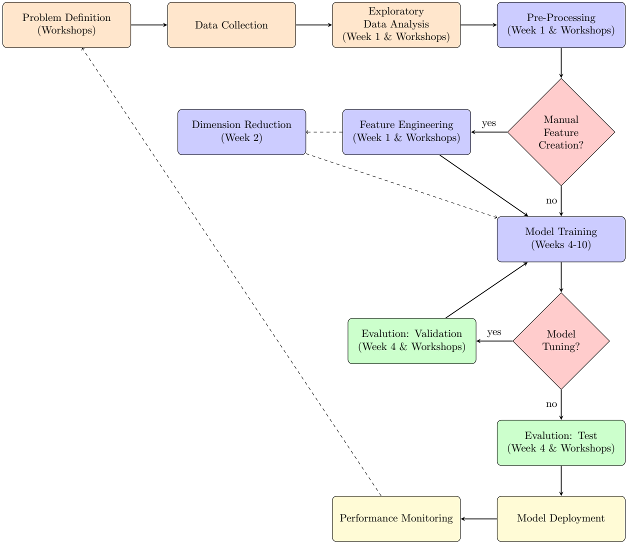
April 1, 2024

Outline

Outline

Course outline

ML Pipeline


Problem Definition (Workshops)
Data Collection
Exploratory Data Analysis (Week 1 & Workshops)
Pre-Processing (Week 1 & Workshops)
Manual Feature Creation?
yes
Feature Engineering (Week 1 & Workshops)
Dimension Reduction (Week 2)
no
Model Training (Weeks 4-10)
Model Tuning?
yes
Evalution: Validation (Week 4 & Workshops)
no
Evalution: Test (Week 4 & Workshops)
Model Deployment
Performance Monitoring

Outline

Avoiding Pitfalls

- It's easy to make mistakes when applying ML, which can lead to unexpected results on data not seen during training and testing.
- For guidelines on how to avoid common mistakes: *How to avoid machine learning pitfalls: a guide for academic researchers*, Lones (2024)
- Recommended to also consult subject-specific guidance when available.

Before you start to build models

- Do take the time to understand your data
  - Make sure that data has reliable source, has been collected using a reliable methodology, and is of good quality
  - Do not assume that, because a data set has been used by a number of papers, it is of good quality
  - *Garbage in, garbage out:* if you train your model using bad data, then you will most likely generate a bad model
- Do exploratory data analysis (e.g. look for missingness, inconsistent records, possible patterns, biases).
- Don't look at all your data; avoid looking closely at any test data in EDA, which may lead to **data leakage**: leakage of information from the test set into the training process.

Before you start to build models

- Do make sure you have enough data
  - If you don't have enough data, then it may not be possible to train a model that generalises
  - However, this may not be evident: it all depends on the signal to noise ratio in the data.
  - If you have limited data, then it's likely that you will have to limit the complexity of the ML models you use, since models with many parameters, like deep neural networks, can easily overfit small data sets.
- Do talk to domain experts
  - Failing to consider the opinion of domain experts can lead to projects which don't solve useful problems, or which solve useful problems in inappropriate ways

Before you start to build models

- Do survey the literature
    - It is important to understand what has and hasn't been done previously *before* you start work.
    - To ignore previous studies is to potentially miss out on valuable information
- Do think about how your model will be deployed

How to reliably build models

- Don't allow test data to leak into the training process
    - A common problem is allowing information about this data to leak into the configuration, training or selection of models
    - When this happens, the data no longer provides a reliable measure of generality, causing failure to generalise to real world data.
    - There are a number of ways that information can leak from a test set; e.g. using the all data for scaling or selecting features before splitting the data.
    - To prevent these issues: partition off a subset of your data at the start of your project, and only use this independent test set once to measure the generality at the end of the project.
    - Incorporating all feature engineering steps in the pipeline, helps to avoid data leakage.

How to reliably build models

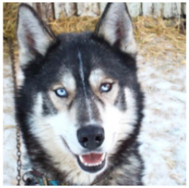
- Do try out a range of different models
  - There's no such thing as a single best ML model (No Free Lunch Theorem); your job is to find the ML model that works well for your particular problem.
- Do include simple baselines as well as state-of-the-art methods.
- Don't use inappropriate models
  - Modern ML libraries also make it easy to apply inappropriate models to your data; e.g. unnecessarily complex models, ignoring temporal dependence, etc.
  - if you're planning to use your model in practice, don't use models that aren't appropriate for your use case

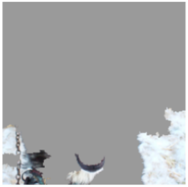
How to reliably build models

- Do be careful where and how you do feature selection.
- Do optimise your model's hyperparameters
    - Many models have hyperparameters, which may significantly effect the performance of the model
    - You should use some kind of hyperparameter optimisation strategy, e.g. grid or random search.
    - However, you should understand limitations of such optimization strategies (poorly identified optimum, variability across folds and randomization), considering not only the optimum but also the complexity/interpretability tradeoff.
    - Avoid data leakage in hyperparameter tuning through pipelines.

How to reliably build models

- Do avoid learning spurious correlations
    - Spurious correlations are features within data which are correlated with the target variable, but which have no semantic meaning.
    - More complex data tends to contain more of these spurious correlations, and more complex models have more capacity to overfit spurious correlations
    - It is always worth looking at your trained model to see whether it's responding to appropriate features within your data


(a) Husky classified as wolf


(b) Explanation

How to robustly evaluate models

- Do use an appropriate test set
  - Always use a test set to measure and approximate the generalization error
  - It's important to make sure the test data set is appropriate. It should not overlap with the training set; it should be representative of the wider population; and the partition should reflect how the dataset will be deployed.
- Don't do data augmentation before splitting your data
  - Data augmentation can be a useful technique, e.g. for balancing datasets
  - However, it's important to do data augmentation only on the training set, and not on the testing data that's going to be used for testing, otherwise this can lead to a number of problems

How to robustly evaluate models

- Do avoid sequential overfitting
  - this is a form of data leakage, where you train models in succession and use each model's test performance to guide the configuration of the next model.
  - Rather, a separate validation set should be used to measure performance as well as analysing the model fit (e.g. residuals) to suggest model tweaks.
- Do consider uncertainty and robustness in the test error
  - Don't over-interpret very small improvements in test performance.
  - The test error is an approximation of the generalization error and it is important to try to understand the uncertainty, either with multiple test sets or resorting to the CLT to compute confidence intervals.
  - Many ML models can be unstable; if you train multiple times or change the data slightly, the performance can vary significantly. You can assess the robustness either with cross-validation (in this case, be careful to use nested cross-validation for hyperparameter tuning) or bootstrap.

How to robustly evaluate models

- Do choose metrics carefully
    - For example, for classification tasks with imbalanced data, accuracy can be a very misleading metric
- Don't ignore temporal dependencies in time series data

How to compare models fairly

- Don't assume a bigger number means a better model
  - If the models are trained or evaluated on different partitions of the same data set, then this may lead to small differences in performance. If they used different data sets entirely, then this may account for even large differences.
  - Another reason for unfair comparisons is the failure to carry out the same amount of hyperparameter optimisation when comparing models
  - For fair comparison, you should freshly implement all the models you're comparing, optimise each one to the same degree, and consider also uncertainty in the test performance
- Don't always believe results from community benchmarks
  - If access to the test data is unrestricted, then you can't assume that people haven't used during training
- Do consider combining models (carefully)

How to report your results

- Do be transparent
    - Be transparent about what you've done and discovered, making it easier for others to build upon your work.
    - It's good practice to share your models in an accessible way, which also encourages you to be more careful, document your experiments, and write clean code
    - Reproducibility is gaining prominence, i.e. your workflow must be adequately documented and shared for publication
- Do report performance in multiple ways
    - For example, consider multiple datasets, multiple metrics, and be clear about what metrics are being used.

How to report your results

- Don't generalise beyond the data (don't overplay your findings, and be aware of their limitations)
  - Don't make general statements that are not supported by the data used to train and evaluate models
  - For instance, if your model does really well on one data set, this does not mean that it will do well on other data sets
  - Multiple data sets may provide more robust insights, but there will always be a limit to what you can infer from any experimental study.
  - One issue is bias, or sampling error: that the data is not sufficiently representative of the real world.
  - Another issue is quality, e.g in deep learning, the need for quantity of data limits the amount of quality checking that can be done
- Do look at your models
  - Do not just report performance metrics; give insight into what the model actually learnt and how it makes decisions.

Outline

Interpretable Machine Learning

Many ML models are considered **black-box**: cannot be understood by looking at their parameters (e.g. a neural network).

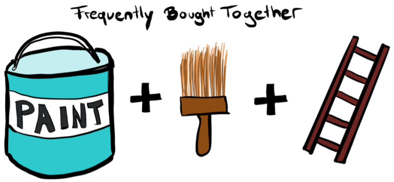
Interpretable Machine Learning refers to methods that make the behavior and predictions of machine learning systems understandable to humans.

Model-agnostic methods for interpretability are general, not tied to specific ML models.

Why do we care about Interpretability?

Why not just trust the model and ignore why it made a certain decision?

- A single performance metric, such as classification accuracy, is an incomplete description of real-word tasks.
- Knowing *why* can help to learn more about the problem, the data and why a model might fail.
- In high-risk environments, a mistake can have serious consequences (e.g. self-driving cars), but even in low-risk settings, humans like explanations.

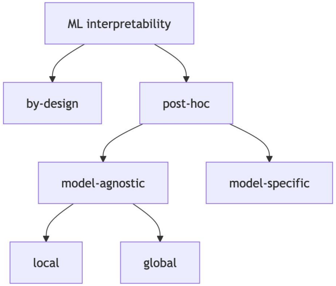


Why do we care about Interpretability?

- To help detect **bias**, e.g. imagine building a model for automatic approval or rejection of credit applications, using data in which a minority demographic has been historically disenfranchised.
- **Fairness**: ensuring unbiased predictions that do not implicitly or explicitly discriminate against underrepresented groups (e.g. Dutch child benefit scandal).
- **Social acceptance** and **trust**, **reliability** or **robustness**, **auditing** and **debugging**, **causality**.

Taxonomy of Interpretable ML

- **Model-specific** or **model-agnostic**?
  - $\rightarrow$ Model-specific interpretation tools are limited to specific models.
  - $\rightarrow$ Model-agnostic tools can be used on any model and are applied after the model has been trained (post hoc).
- **Local or global?** Does the interpretation tool explain an *individual prediction* or the *average* model behavior?



Global model-agnostic post-hoc methods

Global model-agnostic post-hoc methods include:

- **Partial dependence plot**
- Accumulated local effect plots
- Feature interaction (H-statistic)
- Functional decomposition
- **Permutation feature importance**
- Leave one feature out (LOFO)
- Surrogate models
- Prototypes and criticisms

Local model-agnostic post-hoc methods

Local model-agnostic post hoc methods include:

- Ceteris paribus plots
- **Individual conditional expectation**
- Local surrogate models (LIME)
- Scoped rules (anchors)
- Counterfactual explanations
- **Shapley values** and **SHAP** is a computation method for Shapley values (also suggests global interpretation)

Partial Dependence Plots (PDPs)

PDPs show the marginal effect one or two features have on the predicted outcome of an ML model.

Let $\boldsymbol{x}_S$ denote the subset of features for the PDP and $\boldsymbol{x}_C$ denote the other features. The partial dependence function for regression is defined as:

$$f_S(\boldsymbol{x}_S) = \mathrm{E}_{\boldsymbol{x}_C}[f(\boldsymbol{x}_D)] = \int f(\boldsymbol{x}_D) p(\boldsymbol{x}_C \mid \boldsymbol{x}_S) d\boldsymbol{x}_C.$$

Assuming **independence** between $\boldsymbol{x}_S$ and $\boldsymbol{x}_C$ and approximating the distribution of $\boldsymbol{x}_C$ with the empirical, the estimated partial function is:

$$\widehat{f}_S(\boldsymbol{x}_S) = \frac{1}{N} \sum_{n=1}^{N} f(\boldsymbol{x}_S, \boldsymbol{x}_{C,n}).$$

Partial Dependence Plots (PDPs)

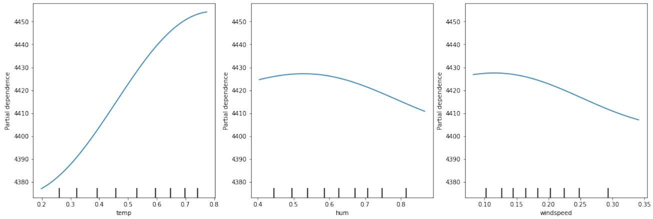


Figure: PDPs for predicting daily bike rentals with SVR.

+ Clear, casual interpretation, easy to compute, rug/histogram can be added to avoid overinterpreting regions with little data.
- Independence assumption, maximum of two features, heterogeneous effects may be hidden.

Individual Conditional Expectation Plots (ICEs)

PDP is an average of the ICE curves, which show how changing one feature changes the prediction for each training point:

$$\widehat{f}_n(x_d) = f(x_{n,1}, \ldots, x_{n,d-1}, x_d, x_{n,d+1}, \ldots, x_D).$$

The ICE plot visualizes these curves for all (or a random subset of) training points.

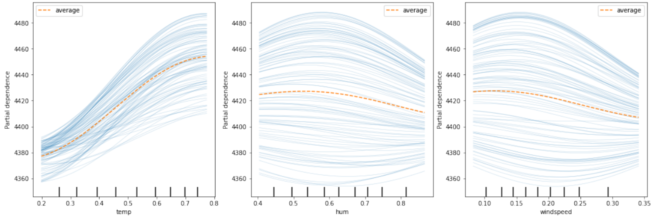


Figure: ICEs for predicting the daily bike rentals.

Individual Conditional Expectation Plots (ICEs)

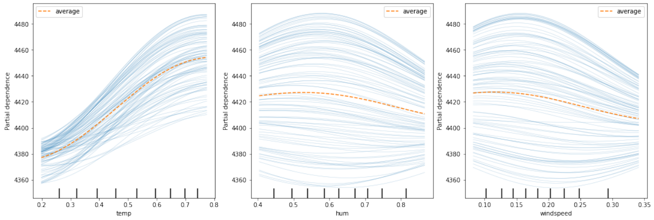


Figure: ICEs for predicting daily bike rentals.

+ Clear interpretation, easy to compute, heterogeneous effects can be uncovered.
- Can display only one feature, may lead to overinterpretation if some features are highly dependent.

Permutation Feature Importance (PFI)

PFI measures the importance of a feature as an increase in loss when the feature is permuted.

Concept: A feature is *important* if shuffling its values increases the model error, while it is *unimportant* if shuffling its values leaves the model error unchanged.

PFIs should be computed on the **test data**, otherwise they can falsely highlight irrelevant features to overfitting on the training data.

Permutation Feature Importance (PFI)

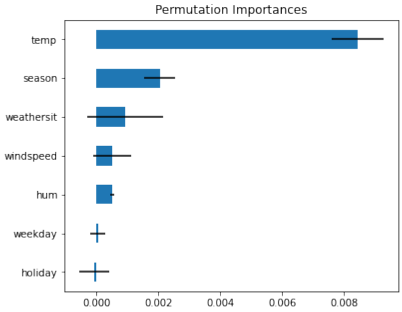


Figure: Predicting daily bike rentals. The dot shows the average importance across multiple permutations, and the line shows the 5% and 95% quantiles from repeated permutation.

+ Clear interpretation, easy to compute, global insights.
- Doesn't tell you *how* the feature influences predictions, can be misleading when features are correlated.

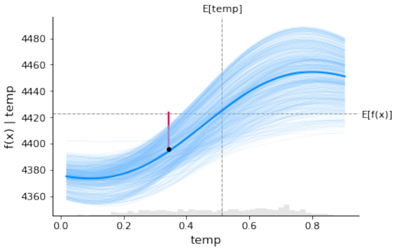
Shapley values and SHAP

Shapley values, a method from coalitional game theory, tell us how to fairly assign the prediction among the features.

$\rightarrow$ for each data point, this allows us to understand how much each feature has contributed to the prediction compared to the average prediction.

However, exact Shapley values are computationally infeasible, and **SHAP** is a computation method to estimate them.

SHAP: local explanations



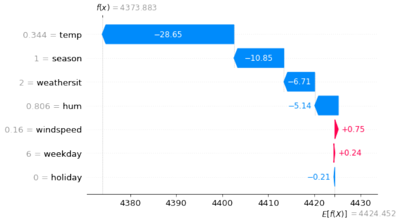


Figure: SHAP for a support vector.

+/- Interpretation: estimated Shapley value is the contribution of a feature value to the difference between the actual prediction and the mean prediction, given the current set of feature values.

+ Provides contrastive explanation with the average prediction, based on solid theory.

SHAP: global interpretations

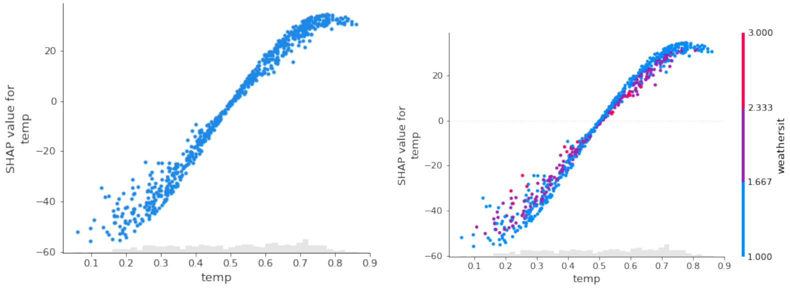
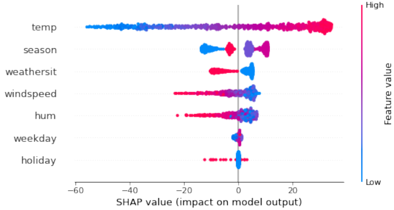
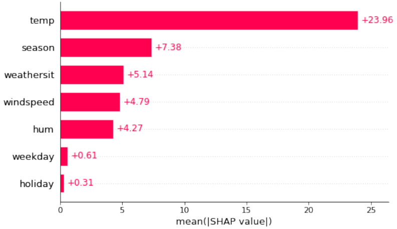


Figure: SHAP dependence plot.

+ SHAP dependence plots visualize how the SHAP values depend on the feature values.
+ Interactions can be visualized by coloring by the SHAP value of another feature.

SHAP: global interpretations





+ Global SHAP feature importance: by averaging the absolute SHAP across the data.
- Can be misleading if features are correlated.

References

- Interpretable Machine Learning
- DALEX: MoDel Agnostic Language for Exploration and eXplanation (available in both Python and R)
- Explanatory Model Analysis
- Explainable AI
- A hitchhiker's guide to responsible machine learning