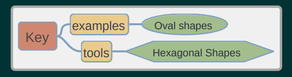
DMLS Chapter 5: Feature Engineering



Data Leakage

refers to a phenomenon where labels **leak** into the set of features in a manner that does not happen during inference

e.g. covid infested hospital had patient lie down during tests. this made lying down a feature that's highly correlated with the "covid positive" label.

common causes

- data splits: splitting time-correlated data randomly instead of training on past and testing on future
- using test data for scaling
- filling in missing data with stats for test set
- duplicates left and split between train and test
- Group leakage: a group has highly correlated labels, but is split between train and test (2 CT scan of the same patient)
- Data generation process; *e.g.* hardware used to generate data

detecting data leakage

- measure the predictive power of each feature
- data leakage can be in a tuple of features instead of just a single feature
- Only measure performance on the test set in the end. do not use it to make design choices.

Engineering Good Features

generally, adding features performs better

However too many features could

- overfit
- cause data leakage
- increase memory requirements
- increase feature calculation overhead
- useless features become technical debt

L1 regularization can help detect them

Feature importance

- can be measured with built-in XGBoost functions
- SHAP: measure feature importance for the entire model, but also for each prediction

Feature generalization

feature coverage: percentage of samples that have this feature

rule of thumb: if it's not in a lot of samples, the feature's not going to generalize well

distribution of feature values

Does this feature have the same distribution in seen and unseen data ? If not, it might not generalize well

this can be mitigated through discretization of continuous values to diminish distribution discrepancy

Learned Features vs Engineered Features

Deep Learning is also called feature learning

Feature Engineering can be very iterative

- what feature
- how many
- requires domain expertise

Common Feature Engineering Operations

Handling missing values

types

- Missing not at random
  - the values missing are missing because of the value itself; *e.g.* undisclosed high salary
  - The fact that values are missing **is** information
- Missing at random
  - The value is missing because of another factor; *e.g.* people of gender A don't like to disclose their age
- Missing completely at random
  - There's simply no pattern to be found among sample with missing values
  - This is pretty rare, so this should be **investigated when encountered**

handling

- deletion
  - easy
  - columns
    - if the number is high
      - ⚠ may remove important information
  - rows
    - if the number is low
      - ⚠ may remove important corner cases
  - ⚠ can create biases
- imputation
  - common choices
    - empty string
    - mean, median, mode
  - ⚠ risks adding noise, biases, or data leakage with aggregations

Scaling

- bring all your features to similar ranges, *e.g.* [0, 1]
- boost ML algos performance
- if your distribution is skewed, you can apply a **log-transformation** to try and make it normally distributed

discretization

setting thresholds to define intervals that turn continuous data into discrete data

- can be used to bucket discrete values too
- introduces discontinuities, *e.g.* 34.9!=35
- subject matter expertise is required

Encoding categorical features

considered categories may be static or dynamic

- brands on Amazon

handling dynamic

- add **unknown** category
  - yields very bad estimates for this category
- The hashing trick
  - randomly hash categories into a predifined range of indices
    - According to Booking.com, for 50% colliding categories, performance loss is only 0.5%.
  - Vowpal Wabbit
  - sklearn
  - Tensorflow
  - pensim

feature crossing

- for features with non-linear relations
- make a cartesian product of features
- good when using linear models
- occasionally helps NNs learn non-linear relationships faster
- DeepFM and xDeepFM
- ⚠ Yields very large feature spaces
  - can cause overfit

discrete and continuous positional embeddings

In sequences, create position information to each sequence element as an embedded feature

- use multidimensional sinusoidal functions of the position as is done by Vaswani et al., 2017
- one embedding per position, as in recent HuggingFace Transformer implementations

Add this information through addition with the original feature vector

generalizes to multidimensional sequential inputs(images, 3D structures ...)