ml-universal-workflow

May 25, 2021

1 The universal workflow of machine learning

1.1 Defining the problem and assembling a dataset

1. What will your input data be?
2. What are you trying to predict?
3. What type of problem are you facing?
    - Binary classification
    - Multiclass classification
    - Scalar regression

    - Vector regression
    - Multiclass, multilabel classification
    - Clustering
    - Generation
    - Reinforcement learning
    - etc.

1.2 Note: Be aware of hypotheses you make at this stage

1. You hypothesize that your outputs can be predicted given your inputs.
2. You hypothesize that your available data is sufficiently informative to learn the relationship between inputs and outputs

Not all problems can be solved; just because you've assembled examples of inputs X and targets Y doesn't mean X contains enough information to predict Y.

Keep in mind that machine learning can only be used to memorize patterns that are present in your training data. You can only recognize what you've seen before. Using machine learning trained on past data to predict the future is making the assumption that the future will behave like the past. That often isn't the case.

1.3 Choosing a measure of success

- balanced-classification problems
    - accuracy, ROC AUC, etc.
- class-imbalanced problems
    - precision, recall, etc.
- ranking problems, multilabel classification
    - average precision

- etc.

1.4 Deciding on an evaluation protocol

- Hold-out validation
- K-fold cross-validation
- Iterated K-fold validation

1.5 Preparing your data

How to format your data? (assuming a deep neural network) - data formatted as tensors - values taken by tensors scaled to small values, ie. [-1, 1] or [0, 1] - data normalized if values in different ranges - feature engineering

1.6 Developing a model that does better than a baseline

Your goal is to achieve *statistical power* : to develop a model capable of beating a dumb baseline.

Three key choices to make: 1. Last-layer activation - linear - sigmoid - softmax - etc. 2. Loss function - binary_crossentropy - mse - etc. 3. Optimization configuration - rmsprop - etc.

Note that there isn't always a direct way to turn a metric into a loss function.

Loss functions need to be: - computable given mini-batch of data (as little as a single data point) - differentiable (can't use backpropagation otherwise)

1.7 Note: Choosing the right last-layer activation and loss function

Problem type

Last-layer activation

Loss function

Binary classification

sigmoid

binary_crossentropy

Multiclass, single-label classification

softmax

categorical_crossentropy

Multiclass, multi-label classification

sigmoid

binary_crossentropy

Regression to arbitrary values

None

mse

Regression to values between 0 and 1

sigmoid

mse or binary_crossentropy

1.8 Scaling up: developing a model that overfits

To figure out how big a model you need, you must develop a model that overfits.

1. Add layers.
2. Make the layers bigger.
3. Train for more epochs.

Always monitor the training and validation loss, as well as any metrics you care about to know if overfitting is achieved.

1.9 Regularizing your model and tuning your hyperparameters

- Add dropout
- Add L1 and/or L2 regularization
- Try different architectures: add / remove layers
- Try different hyperparameters
  - units per layer
  - learning rate
  - etc.
- Iterate on feature engineering

Note: Using feedback from your validation process to tune your model over many iterations may cause the model to overfit to the validation process.

Note: If the performance on the test set is significantly worse than the one on validation data, this may mean either that your **validation process wasn't reliable** or your **model had overfitted to the validation process**. (Switch to a more reliable evaluation protocol, eg. k-fold validation)