COMP 333 — Week 11 Machine Learning Process

Machine Learning Process

EDA builds *models* to capture the insight
to predict outcomes in new situations
as aids to decision-making.

The criteria for our choice of model are:

- Whether the models meets the *business goal*
- How much *pre-processing* the model needs
  including the time required to build (train) the model
- How *accurate* is the model
  in general, how well does it perform during evaluation
- How *explainable* is the model
- How *fast* is the model in making predictions
- How *scalable* is the model
  when deployed
  for both building the model and making predictions

From the video
Choosing the right ML algorithm,
by MS Azure Team, DevDays Asia 2017
`https://channel9.msdn.com/Events/OpenSourceTW/DevDays-Asia-2017/AI12`

ML Process in Overview

Once you have clean data,
and have completed feature engineering,

there are several major steps for machine learning:

Training a model

Evaluating a model

Deploying a model

Using the model to make predictions

Validating the model in real-world use

Explaining predictions of the model

Scaling up

Most discussions of machine learning
will focus in the first two steps

training a model, and

evaluating a model.

Building an ML Model

We will assume we are doing supervised learning.

Input The key input to the ML process
is a clean dataset
with a target variable that is labelled
and a set of features that form the basis of the prediction.

Input Concerns In general, there are several concerns

- How many observations do I have?
- Do I have enough observations for each label value?
- Is the dataset balanced?
  That is, for each label value, do I have roughly the same number of observations?

Separation of Training and Evaluation

For the evaluation of a model,
it is important that the test data
is independent of
the training data.

In particular,
if an observation is used during the training of a model
then that observation cannot be in the test dataset.

Splitting the dataset The (independent) **test set** is taken as
a random subset of the dataset
with the remainder of the dataset being used as the **training set**.

This is called *splitting* the dataset into training set + test set.

It is also called *hold-out* of a test set
when you split up your dataset into training set + test set.

A common split is 20% as the test set and 80% as the training set.

Training the model The model is constructed by the ML algorithm
using the training data.

This is called *training* the model,

or *fitting* the model to the data.

Evaluating the model The model is evaluated by running the model on unseen data,
which is the test set.

For each observation in the test set,
you compare the model's prediction
with the target value in the observation.

From the test run, you calculate **performance metrics**.

To summarize:

- Split the dataset into two pieces: a training set and a testing set.
- Train the model on the training set.
- Test the model on the testing set, and evaluate how well our model did.

Advantages of train/test split:

- Model can be trained and tested on different data than the one used for training.
- Response values are known for the test dataset, hence predictions can be evaluated
- Testing accuracy is a better estimate than training accuracy of out-of-sample performance.

From

Learning Model Building in Scikit-learn : A Python Machine Learning Library

`https://www.geeksforgeeks.org/learning-model-building-scikit-learn-python-machine-learni`
`?ref=rp`

Performance Metrics

There are number of common performance metrics in use.

Here we present them in the context of a binary classifer.

For a binary classifier, we have one class and data

labelled as *Positive* (P) examples, or *Negative* (N) examples.

Our classifier makes predictions, either P or N.

To define our metrics, we count

Total = the number of test cases

TP = the number of *true positives*
that is, a P test case that is predicted as P

TN = the number of *true negatives*
that is, a N test case that is predicted as N

FP = the number of *false positives*
that is, a N test case that is predicted as P

FN = the number of *false negatives*
that is, a P test case that is predicted as N

Note that

Total = TP + TN + FP + FN

Number of Positive actual test cases = TP + FN

Number of Negative actual test cases = TN + FP

Number of Positive predictions = TP + FP

Number of Negative predictions = TN + FN

Precision is a measure of how many positive predictions were really positive:

Precision = TP / (TP + FP)

Recall is a measure of how many positive test cases were predicted as positive:

Recall = TP / (TP + FN)

Accuracy is a measure ofthe number of correct predictions made by the model overall:

Accuracy = (TP + TN) / Total

Specificity is a measure of how many negative test cases were predicted negative:

Specificity = TN / (TN + FP)

Specificity is the opposite of recall.

Sensitivity is the same as recall:

Sensitivity = TP / (TP + FN)

F-measure also called *F1-measure* or *F1-score* or *F-score*,

is the harmonic mean of precision and recall:

F-measure = 2 x Precision x Recall / (Precision + Recall)

F-measure = 2 x TP / (2xTP + FP + FN)

MCC (Matthews Correlation Coefficient) a correlation coefficient between the observed (actual) and predicted binary classifications.

It returns a value between +1 and +1.

A coefficient of +1 represents a perfect prediction,

0 no better than random prediction

and -1 indicates total disagreement between prediction and observation.

MCC = [(TP x TN) - (FP x FN)] / sqrt((TP+FP)x(TP+FN)x(TN+FP)x(TN+FN))

Confusion matrix The *confusion matrix* shows the ways in which your classification model is confused,

that is, when the model is making wrong predictions.

It is a useful aid to understanding your model.

What is a Confusion Matrix in Machine Learning

by Jason Brownlee

`https://machinelearningmastery.com/confusion-matrix-machine-learning/`

Which single metric to use? The best metric to use as

a single performance metric is **MCC**

as it considers both correct and incorrect predictions,

and it works well for imbalanced data.

Cross-Validation

Cross-validation is an evaluation method.

Let us look at k-fold cross-validation, where k = 5.

We divide our dataset D into 5 *"folds"*,
that is, disjoint subsets D1, D2, D3, D4, D5 of D
of the same size and chosen randomly.

The evaluation does 5 iterations: i = 1 .. 5

> use Di as the test set
> train your model on $D \setminus Di$
> use this model to predict each test case in Di

At the end, you have a prediction for each case in the dataset D
and you can calculate performance metrics
comparing actual classification to the predicted classification.

It is common to use 5-fold and 10-fold cross-validation.

Leave-One-Out Cross-Validation (LOOCV) is the extreme cross-validation
where k is the size of D.
That is, you use each d in D as a test case
and the remainder $D \setminus \{d\}$ as the training set.

Evaluation versus Validation

I like to reserve the term *validation*
for confirmation that a model in real-world deployment and use
is performing as well as, or better than, (human) experts
in its predictions.

Or that the value of the model in the decision-making process
has been tested in the real-world context
and been accepted by the decision makers.

So *validation* is an evaluation,
but in a real-world context
and measured in its context of application.

Other people make the distinction between

internal validation, and

external validation

where internal validation is evaluation in a test set-up
and external validation is evaluation in the real-world context.

So their *external validation* is what I call *validation*
and I would call their *internal validation* as evaluation.

Beware The use of the two terms *evaluation* and *validation*
is very confusing in practice.

For example, the term *cross-validation*
is the standard term for the method of evaluation in ML.