Fundamentals of Machine Learning*

ME 416 - Prof. Tron

Thursday 4th April, 2019

The most succinct definition of machine learning is the following [2]. A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P, if its performance at tasks in T, as measured by P, improves with experience E. The different instances of machine learning problems differ by the choice of the task T, the performance measure P and the experience E.

1 Task T

At a high level, the basic task of a learning algorithm is to learn a *model*, i.e., a function f that maps an *input vector* x to some predicted output $f(x)$. The specific different tasks mainly differ on the type of output requested by f.

Classification The output of f must belong to a discrete set of *class labels* $\{1, \dots, L\}$. For instance, x might represent an image, and the labels might represent L distinct object categories (e.g., cup, ball, pen, etc.).

Regression The output of f is a real number or vector. It is very similar to the classification problem, except that the output can be a continuous quantity instead of discrete. For instance, x might be a vector containing basic data on a person (e.g., age, gender, income, etc.), and $f(x)$ might represent the predicted probability that that person will buy a specific product. Another example is the one of *denoising* imputation of missing values/, where the input x is some version of some "clean" vector x_{true} with added noise or distorsion/missing values and $f(x)$ represents an estimate of x_{true}.

Structured output The output of f is a vector with some internal structure. /Object detection/ is an example of a structured output task where x represents an image and $f(x)$ is a vector containing the coordinates of a predicted bounding box containing an object of a given class.

Machine translation /Transcription x represents a sequence of symbols in some language/an image or audio, and $f(x)$ represents an equivalent sequence of symbols in some other language/that describes the content of the image.

Note on classification vs detection ★★ In classification, it is assumed that the object occupies more or less the entire image, but its class is unknown; in detection, the class is

*The material in this document closely follows [1].

known, but the position in the image is not. Detection can be implemented with a *sliding-window* approach by running, at every possible location, a classifier that predicts a label in two classes: object present/not present.

In most cases, the learning algorithm uses a *parametric model* $f(x, \Theta)$, where Θ is a set of parameters. The learning algorithm then uses the experience E to adjust the parameters Θ to maximize the performance P. For instance, for a scalar regression problem the model could be a line, where Θ contains the slope and intercept of the line.

TODO: insert a figure for linear regression.

For a binary classification problem, Θ could be a threshold used to decide the class

TODO: insert figure with example on binary classifier

2 Performance measure P

The performance measure quantifies how well the task T is performed, and it is usually specific to that task. Examples of such quantitative measures are:

Accuracy and error rate (classification tasks) : the portion of examples for which the model produces the correct/incorrect output.

Fitting error (regression tasks) distance between the actual output of the model and the desired output.

The choice of P can be intuitive at a high level, but it is often difficult to decide the details based on the desired behavior of the system. For instance, we might have different types of errors that we might want to treat in different ways.

Accuracy curves ★★ In a binary classification task where the classes are "detected"/"absent", there are two types of errors:

False positives (FP, a.k.a. Type I errors) The classifier returns "detected" when in reality the label should be "absent".

False negatives/ (FN, a.k.a. Type II errors) The classifier returns "absent" when in reality the label should be "detected". For some applications, a type of error could be more costly than the other.

TODO: introduce TP and TN,

2.1 Precision-recall curves

For learning algorithms that use parametric models, it is usually the case that by varying one of the parameters (e.g., a threshold) one has a tradeoff between precision and recall (i.e., between FP and FN). This can be captured by the following:

Definition: a *precision-recall* curve is the curve with coordinates $\begin{bmatrix} precision \\ recall \end{bmatrix}$ parametrized by the parameter of interest

TODO: graphical example of curves for linear classifier

3 Experience E

The typical experience E is available to learning algorithms is in the form of a *dataset*, which is a collection of *data points* (input vectors) x.

3.1 Supervised vs unsupervised learning

The learning task T is said to be *supervised* if the dataset contains a "target" result y (e.g., a label in classification, or a value for regression) for each x, and the goal is to have $f(x) \approx y$. Conversely, the learning task T is said to be *unsupervised* if the dataset contains only the input vectors x, and the algorithm needs to figure out a $f(x)$ that "makes sense".

A typical unsupervised task is *clustering*, in which the algorithm needs to divide the data points in homogeneous groups. TODO: graphical example of clustering

3.2 Training, validation and test sets

In the development of a machine learning algorithm, there are two phases: a *training* (development) phase where the function f is learned from a dataset, and a *testing* phase where the algorithm is actually deployed and the function f is used to make prediction on data that has not been seen before. Typically, we are interested in making sure that the learned model f *generalizes* well, in the sense that it performs well (according to the performance P) during testing.

TODO: graphical illustration of generalization: "full set" that includes 2-D unseen data and dataset, two classifiers that have same error on training set but different generalization performance.

In order to evaluate the performance of an algorithm during the development phase (e.g., so that we can compare competing algorithm) in an unbiased way, it is customary to split the dataset available into *training* and *testing* sets (the split is usually around 80% and 20%, respectively).

During training, the algorithm has access only to the training set. Then, after the model f is fixed, its performance is evaluated on the testing set. This is because reporting only the performance on the training dataset would not be indicative of the performance one would obtain in real situations on unseen data. In this light, it is important that the split of the dataset is done in a considerate way; e.g., for a face detection task, we would like to test on faces of new individuals, and not on new images of the same individuals used for training.

Sometimes, the model f has a set of *hyperparameters* Θ' that needs to be hand-tuned (e.g., the number of clusters to use in a clustering task). Manually tuning Θ' to optimize the performance P on the test set is the most tempting option. But again, this would be equivalent to using the testing dataset for learning Θ' (where the learning algorithm is carried out manually), thus producing again results that are not indicative of what performance one would expect in real situation. In order to address this, it is customary to further split the training dataset into a real training set and a *validation* set. The training set is used to learn the model automatically, and the validation set is used to manually tune the hyperparameters to optimize the performance of the model. However, the final choice of Θ' might depend on how the split is performed (just by luck, we might select an "easy" validation set). In order to counteract this, it is customary to perform *cross-validation*, where the performance on the validation set is evaluated by averaging the results on a large number of different random training-validation splits.

References

[1] I. Goodfellow, Y. Bengio, and A. Courville. *Deep Learning*. MIT Press, 2016. `http://www.deeplearningbook.org`.

[2] T. Mitchell. Machine learning, mcgraw-hill higher education. *New York*, 1997.