Summary of FK7068 Machine Learning for Physicists and Astronomers

Yashar Honarmandi
yasharh@kth.se

20 december 2021

Sammanfattning

Innehåll

1 Basic Concepts

Machine Learning Machine learning is the field of study that gives computers the ability to learn without explicit programming. A computer is said to learn from some experience E with respekt to a task T and a performance measure P if its performance at T as measured by P increases with E.

Supervised Learning Supervised learning deals with labelled data, and the task is to correctly label unfamiliar data. The two main types of supervised learning problems are classification and regression.

Unsupervised Learning Unsupervised learning, bu contrast, deals with data that does not have labels. The task is generally to identify some patterns within the data. Examples of such problems include clustering, anomaly detection, pattern recognition and association mining.

Reinforcement Learning Reinforcement learning is, loosely speaking, based on performing random explorations of prediction processes and learning from the output.

A Note on Hardware Machine learning benefits strongly from parallelization. This is one of the reasons why, despite CPUs having high clock speeds, GPUs are the preferred hardware for training algorithms.

Using Machine Learning Machine learning is notoriously full of pitfalls. Great care must be taken in the preparatory work and the process of creating and training the model. The general steps in the procedure are

1. **Define the problem and plan.** Formulate good questions and assess whether a machine learning approach is adequate and necessary.
2. **Estimate the required computational resources.** Combined with economic aspects, this will determine the limits of the complexity of your model.
3. **Prepare data.** This step is crucial and will include identifying sources, data collection and preprocessing. Relevant questions to ask include whether you have enough data (a quintessential point) and whether the data is sufficiently diverse.
4. **Construct an appropriate model.**
5. **Test the model.**
6. **Deploy.**

See also this article for a well-known and detailed example of how one should think.

Statistical Learning Statistical learning is a framework in which the tools of statistics are used to describe and evaluate a machine learning algorithm.

A useful starting point is the limit of large numbers, for which we can us the Markov density approximation

$$P(X = x) \approx \frac{1}{N} \sum_l \delta(x - x_l),$$

which implies

$$\langle f(X) \rangle \approx \frac{1}{N} \sum_l f(x_l).$$

This holds for identically distributed, independent realisation x_l of $P(X = x)$.

Given this, the input to the learner is the domain set X, points in which are numerical representations of the data, the label space Y, which contains the possible labels that can be ascribed to a point in X, and a set S of pairs $(x, y) \in X \times Y$, which is a realization of the joint probability distribution $P(X = x, Y = y) = f(x, y)$. The output is a prediction rule $h : X \to Y$.

We will need some way to assess the quality of h. This is done by introducing a loss function $L(h(x), y)$, which measures the ability of h to predict y given x. Next, we introduce the risk functional

$$R(h) = \int \mathrm{d}x\,\mathrm{d}y\, f(x,y) L(h(x),y) = \int \mathrm{d}x\, P(X=x) \int \mathrm{d}y\, P(Y=y \mid X=x) L(h(x),y),$$

which measures the expected loss. Assuming a good choice of the loss function, the best choice of h is taken to be the one that minimizes the risk. Such a predictor is called Bayes optimal.

We take f to be unknown, but assuming the training data to be sufficiently large we can construct a Markov density approximation for f. We then find

$$R(h) \approx R_S(h) = \frac{1}{|S|}\sum_i \int \mathrm{d}x\,\mathrm{d}y\,\delta(x-x_i)\delta(y-y_i) L(h(x),y) = \frac{1}{|S|}\sum_i L(h(x_i),y_i),$$

which is the so-called empirical risk. We can now approximately identify an optimal choice of h by minimizing this quantity. This process is called empirical risk minimization. The assumption that the class of hypotheses has a member that minimizes R_S is the realizability assumption.

Bayes Optimal Predictors The Bayes optimal predictor is a predictor which, given the probability distribution $P(y = y \mid X = x)$, labels according to

$$f(x) = \begin{cases} 1, & P(y=y \mid X=x) \geq \frac{1}{2}, \\ 0, & \text{otherwise.} \end{cases}$$

This predictor turns out to minimize the true loss.

Probability Terms of Relevance A few terms of relevance are

- accuracy, which is the fraction of predictions that is correct.
- precision, which is the fraction of positive predictions that corresponds to true positives.
- sensitivity, which is the fraction of true positive cases that is identified.
- specificity, which is the fraction of negative predictions that corresponds to true negatives.

Probably Approximately Correct Learning How wrong is the empirical approximation? It is clear that R_S approaches R in the limit of infinite sample sizes. For finite sample sizes, however, we can only make probabilistic statements. We introduce probably approximately correct (PAC) learning as follows: A hypothesis class H is PAC learnable if there exists a number m_H depending on two probabilities such that for every pair of probabilities ϵ, δ and under the realizability assumption, when running the learning algorithm with $m \geq m_H(\epsilon, \delta)$ iid samples, the algorithm returns a hypothesis that satisfies

$$P(|R(h) - R_s(h)| > \epsilon) < \delta.$$

It turns out that in the case of a finite hypothesis class, in the realizable case, we have

$$m_H(\epsilon,\delta) = \frac{1}{\epsilon}\left(\ln(|H|) + \ln\left(\frac{1}{\delta}\right)\right).$$

Agnostic PAC Learning A hypothesis class is agnostically PAC learnable if there exists a number m_H depending on two probabilities and a learning algorithm such that for every pair of probabilities ϵ, δ and every distribution P over $X \times Y$, when running the learning algorithm with $m \geq m_H(\epsilon, \delta)$ iid samples from P, it holds that

$$P(L(h_S) - \min L(h) \leq \varepsilon) \geq 1 - \delta.$$

In this case we instead have

$$m_H(\epsilon,\delta) = \frac{1}{2\epsilon^2}\left(\ln(|H|) + \ln\left(\frac{2}{\delta}\right)\right).$$

The Confusion Matrix The confusion matrix is a 2×2 matrix for which the diagonal elements contain the true positives and negatives, the upper right contains the false positives and the lower left contains the false negatives. There are also extension for more advanced classification problems.

The No Free Lunch Theorem It has been shown that if no assumptions are made about the data, there is no reason to prefer any particular machine learning model. In other words, there is no universal learner (combination of model and learning algorithm) that performs equally well for all problems. This is the no free lunch theorem.

The Bias-Complexity Tradeoff The empirical risk can be divided into two terms. The first is the approximation error, which is equal to the minimal empirical risk within the chosen hypothesis class. The second is the estimation error, which is equal to the difference between the observed empirical risk and the approximation error. The former thus represents the limitations of the hypothesis class itself, while the latter represents the success of the algorithm in producing a hypothesis given the data.

How can you change these? The solution to reduce the approximation error is to change the model. At some point one has to make the model more complicated, however, and it can be hard for a learning algorithm to identify a good hypothesis given the data. This is a tradeoff between the bias in the choice of hypothesis class and the complexity of the model.

In practical contexts, it will be important to validate the performance of your model on a separate dataset. The bias-complexity tradeoff manifests here as a tradeoff between performance on the training and validation sets respectively. We here estimate the approximation error as the error on the training set and the estimation error as the error on the validation set. One typically optimizes with respect to the validation error to try to ensure generalizability.

Linear Predictors A linear predictor is an element of the hypothesis class with elements of the form

$$h(x; w, b) = f(w^T x + b),$$

which can of course be shortened to

$$h(x; w) = f(w^T x)$$

by extending x and w. An often-seen choice is the logistic function

$$\sigma(s) = \frac{1}{e + e^{-s}}.$$

Matrix Regularization Consider the problem

$$Ax = b.$$

In the cases where A is not invertible, we have to do some modification of the problem. This is called regularization.

Classification and Cross-Entropy Consider a classification problem with N classes, and restrict your hypothesis class to some set of functions

$$P(y^n \mid x, w) = f_w^n(x)$$

parametrized by the w. The likelihood of observing some given data set can be written

$$L(\{y_i\}) = \prod_i f_w^{n_i}(x_i) \prod_{n \neq n_i} (1 - f_w^n(x_i)),\ y^{n_i} = 1,$$

or more neatly,

$$L(\{y_i\}) = \prod_i \prod_n (f_w^n(x_i))^{y_i^n} (1 - f_w^n(x_i))^{1-y_i^n}.$$

The log likelihood is

$$\ln(L) = \sum_i \sum_n y_i^n \ln(f_w^n(x_i)) + (1 - y_i^n) \ln(1 - f_w^n(x_i)),$$

and its maximization is equivalent to minimizing the cross-entropy

$$S = -\sum_i \sum_n y_i^n \ln(f_w^n(x_i)) + (1 - y_i^n) \ln(1 - f_w^n(x_i)).$$

Stochastic Gradient Descent Gradient descent is a suboptimal way of minimizing a function, so we will instead opt for a different method.

Suppose we divide our data into $\frac{M}{I}$ subsets indexed by a k. We denote each subset B_k. The empirical risk is then is then

$$R_S = \sum_{i \in B_k} l(x_i) + \sum_{i \notin B_k} l(x_i).$$

The former is the in-batch loss and the latter the error. Discarding the former we have

$$\vec{\boldsymbol{\nabla}} R_S \approx \sum_{i \in B_k} \vec{\boldsymbol{\nabla}} l(x_i).$$

For iid batches this is an unbiased estimator of $\vec{\boldsymbol{\nabla}} R_S$. The stochastic gradient descent algorithm thus involves doing one gradient descent per batch. Such a cycle is called an epoch. This is advantageous because it is faster and lessens the risk of getting stuck in local minima.

Neural Networks Neural networks are elements in a very particular hypothesis class. They are conceptually inspired by neurons.

The structure is as follows: The network consists of layers. Each layer consists of a set of neurons. The neurons are generally coupled to all neurons in the previous layers. Computationally, each neuron takes as input the set of values stored in each neuron. It then adds them using some weights and outputs the result when this weighted sum is input into some so-called activation function. We may thus illustrate the network as a directed graph, where each neuron is a node and there are edges connecting all nodes in adjacent layers. These edges are directed, such that there is only coupling one way.

As an example, consider the simplest possible neural network, with one input layer and one output layer, the latter having only a single neuron. The input layer will then consist of all the elements in x (as well as a 1), and the neuron will first compute $\sum w_i x_i = w^T x$, and output $\sigma(w^T x)$.

The power of neural networks lies in the fact that if the activation function is nonlinear, then the network can be used to perform nonlinear classification or regression tasks. The design of the network, however, is where the art lies.

Neural networks need to be trained, of course. The algorithm for doing that is called back-propagation. To describe the mathematical basis we introduce a bit of notation. We let:

- V_t be the set of neurons in layer t.
- W_t be the weight matrix between V_t and V_{t+1}.
- k_t be the number of neurons in layer t.
- $\mathbf{a}_t$ be the input to layer t.
- $\mathbf{o}_t$ be the output from layer t.
- $\boldsymbol{\sigma}$ be the activation function applied element-wise to the output of one layer.

Clearly we have $\mathbf{a}_t = W_{t-1}\mathbf{o}_{t-1}$ and $\mathbf{o}_t = \boldsymbol{\sigma}(\mathbf{a}_t)$. The weights in any one layer appear exactly once in the output, so it would be beneficial to switch from a matrix notation in W to a vector notation. To that end, introduce the matrix $O_{t-1} = \mathrm{diag}(\mathbf{o}_{t-1}^T)$ and $\mathbf{w}_{t-1}$ the vector obtained when concatenating the rows of W_{t-1}.

It then follows that $O_{t-1}\mathbf{w}_{t-1} = W_{t-1}\mathbf{o}_{t-1}$. Now, we note that the loss function has a recursive structure - that is,

$$\begin{aligned} L(\{W_t\}) =& L(\mathbf{o}_T) \\ =& L(\boldsymbol{\sigma}(\mathbf{a}_T)) \\ =& L(\boldsymbol{\sigma}(W_{T-1}\mathbf{o}_{T-1})) \\ =& \dots \end{aligned}$$

We can write this as $L = \ell_t \circ \boldsymbol{\sigma}$ for some choice of t. ℓ_t thus acts as a subsystem loss function, handling the contributions to the loss from layers starting at t. We then have

$$\vec{\boldsymbol{\nabla}}_{\mathbf{w}_{t-1}} L = \vec{\boldsymbol{\nabla}}_{\boldsymbol{\sigma}} \ell_t \frac{\partial \boldsymbol{\sigma}}{\partial \mathbf{w}_{t-1}}.$$

Because the activation function acts element-wise we have

$$\frac{\partial \boldsymbol{\sigma}}{\partial \mathbf{w}_{t-1}} = \text{diag}(\vec{\boldsymbol{\nabla}}\sigma)\frac{\partial}{\partial \mathbf{w}_{t-1}}(O_{t-1}\mathbf{w}_{t-1}) = \text{diag}(\vec{\boldsymbol{\nabla}}\sigma)O_{t-1}.$$

The first factor is to be evaluated at $\boldsymbol{\sigma}(O_{t-1}\mathbf{w}_{t-1}) = \mathbf{o}_t$ and the second at $O_{t-1}\mathbf{w}_{t-1} = \mathbf{a}_t$. Thus we find

$$\vec{\boldsymbol{\nabla}}_{\mathbf{w}_{t-1}} L = \vec{\boldsymbol{\nabla}}_{\boldsymbol{\sigma}} \ell_t(\mathbf{o}_t)\text{diag}(\vec{\boldsymbol{\nabla}}\sigma(\mathbf{a}_t))O_{t-1}.$$

This is a vector comprised of the successive components of $\boldsymbol{\delta}$, the derivatives of σ and $\mathbf{o}_{t-1}$. Thus, by computing δ_t the derivative is known. We can do this in a recursive manner by noting that it is trivial for the outer layer. Next, as

$$\ell_t(\mathbf{o}_t) = \ell_{t+1}(\boldsymbol{\sigma}(W_t\mathbf{o}_t)),$$

hence

$$\vec{\boldsymbol{\nabla}}\ell_t(\mathbf{o}_t) = \vec{\boldsymbol{\nabla}}\ell_{t+1}(\mathbf{o}_{t+1})\text{diag}(\vec{\boldsymbol{\nabla}}\sigma(\mathbf{a}_{t+1}))W_t,$$

implying

$$\boldsymbol{\delta}_t = \boldsymbol{\delta}_{t+1}\text{diag}(\vec{\boldsymbol{\nabla}}\sigma(\mathbf{a}_{t+1}))W_t.$$

In this sense we propagate information back through the network. The algorithm is thus

1. Start with $\mathbf{o}_0 = \mathbf{x}$.
2. Compute $\mathbf{a}_t = W_{t-1}\mathbf{o}_{t-1}$ and $\mathbf{o}_t = \boldsymbol{\sigma}(\mathbf{a}_t)$ for each layer.
3. Set $\boldsymbol{\delta}_T = \vec{\boldsymbol{\nabla}}L(\mathbf{o}_T)$.
4. Compute $\boldsymbol{\delta}_t = \boldsymbol{\delta}_{t+1}\text{diag}(\vec{\boldsymbol{\nabla}}\sigma(\mathbf{a}_{t+1}))W_t$ for each layer.
5. Set the partial derivative with respect to each set of weights to be $\boldsymbol{\delta}_t\text{diag}(\vec{\boldsymbol{\nabla}}\sigma(\mathbf{a}_t))O_{t-1}$.

Problems With Training Neural Networks A first problem with training neural networks is vanishing or exploding gradients. To illustrate it, consider a neural network with one neuron per layer. We then have

$$\boldsymbol{\delta}_0 = \boldsymbol{\delta}_L \prod_{l=1}^{L} \text{diag}(\vec{\boldsymbol{\nabla}}\sigma(\mathbf{a}_l))W_l.$$

For a network with many layers the product can thus very quickly either diverge or drop to zero. Only if each factor is close to the identity will the gradients stay behaved. Beyond applying stochastic gradient descent, which is an implicit regularizer, there are other treatments of the gradient that can be performed to ensure that this holds.

Another problem is that different weights can change such that they compensate for mistakes in other places in the neural networks. This is called co-adaption, and it is bad because such fixes are generally overfitted to the training data. This can be solved by using dropout.

Next there is the problem that layer inputs change during training, requiring low learning rates. This can be somewhat circumvented by standardizing inputs.

The Universal Approximation Theorem The universal approximation theorem is a statement about how closely we can approximate functions with neural networks. To prove it, consider a so-called feed-forward neural network with L neurons in a single hidden layer. Its output is

$$y(\mathbf{x}) = \sum_{l=0}^{L} \alpha_l \sigma(\mathbf{w}_l^T \mathbf{x} + b_l),$$

where we have used some sigmoid activation function (this just means that σ goes to 0 and 1 respectively at the two infinities). We also introduce the following definition: σ is discriminatory for a measure $\mu \in M(l_d)$ (this can for instance be a probability measure $\mathrm{d}\mu = \mathrm{d}\mathbf{x}\, P(\mathbf{x})$) if the fact that

$$\int \mathrm{d}\mu\, \sigma(\mathbf{w}_l^T \mathbf{x} + b_l) = 0$$

for all $\mathbf{w}_l, b_l$ implies that $\mu = 0$. This definition is useful because it tells us that discriminatory functions are non-destructive with respect to the metric - that is, they are indeed able to capture some of the information in the network.

The theorem states the following: Let l_d be the hypercube $[0,1]^d$ and the space of real-valued continuous functions on l_d be $C(l_d)$. Let also σ be a discriminatory function on l_d. Then, for every $\varepsilon > 0$ and function $f \in C(l_d)$, there exists an integer N and real $\alpha_l, b_l, \mathbf{w}_l$ for $l = 1, \dots, N$ such that the function

$$G(\mathbf{x}) = \sum_{l=0}^{L} \alpha_l \sigma(\mathbf{w}_l^T \mathbf{x} + b_l)$$

satisfies

$$|G(\mathbf{x}) - f(\mathbf{x})| < \varepsilon \;\forall\; \mathbf{x} \in l_d.$$

To prove it, let S be the set of functions of the form G. Clearly this is a linear subspace of $C(l_d)$. To prove it, we will show that the closure of S (the smallest closed set containing S) is all of $C(l_d)$. Assuming the contrary, the closure of S is instead some different subset R of $C(l_d)$. We have by the Hahn-Banach theorem that there is a bounded non-trivial linear functional L of $C(l_d)$ such that $L(R) = L(S) = 0$. This can be represented as

$$L(h) = \int_{l_d} \mathrm{d}\mu\, h(\mathbf{x})$$

for some measure μ and all h. By our assumptions there must then exist a measure such that

$$L(\sigma) = \int_{l_d} \mathrm{d}\mu\, \sigma(w_l^T \mathbf{x} + b_l) = 0,$$

but if σ is discriminatory, then this is a contradiction. This implies that R must indeed be $C(l_d)$, completing the proof.

Convolutional Neural Networks From the properties of eyesight, it appears that the brain processes visual data in small parcels. The processing is built up of simple tools, such as edge detectors and depth detectors. Notably only a small number of shapes form primitive visual objects. These properties should enter into the design of neural networks for image processing.

Convolutional neural networks are based on these concepts. It uses a set of filters that are convolved with the image (swept over the image, with the local dot product being summed up) in so-called convolution layers. Beyond this, in order to get output you typically use some kind of pooling operation, which coarse-grains.

The Curse of Dimensionality As the number of features of the data increases, the amount of training data required to obtain generalizability grows exponentially.

Dimensionality Reduction Dimensionality reduction is a set of techniques for mapping features in data to a space of lower dimension. Good dimensionality reduction techniques preserve data variance.

Fisher's Linear Discriminant Fisher's linear discriminant is a dimensionality reduction technique based on the goal of finding a projection operation on the data such that class separation is maximized, the variance between classes is maximized and the class overlap is minimized through the reduction of in-class variance.

For two classes in two dimensions we may take the projected means to be

$$m_i = w^T \mu_i,$$

where the μ_i are in-class means. The projected in-class variances are then

$$\sigma_i^2 = \frac{1}{N_i} \sum_n (w^T x_n^i - m_i)^2.$$

The function we want to optimize is

$$L(w) = \frac{(m_2 - m_1)^2}{\sigma_1^2 + \sigma_2^2}.$$

Principal Component Analysis Principle component analycis (PCA) is a dimensionality reduction based on finding high-variance directions in feature space and removing others. The underlying assumption is that valuable information is contained in directions with the largest variance.

To study it, suppose the data is centered introduce the matrix X^T, the columns of which are the data points. The covariance matrix for the features is then

$$\Sigma = \frac{1}{N-1} X^T X.$$

Because Σ is symmetric, we can perform an eigenvalue decomposition. Denoting the eigenvector matrix as V and the eigenvalue matrix as Λ, we say that the columns of V define the principal directions. Next we discard directions with small variance, reducing the dimensionality from p tp $\tilde{p}$, by introducing a projection matrix $\tilde{V_p}$ with dimension $p \times \tilde{p}$ and a new data matrix $\tilde{Y} = X\tilde{V_p}$. The columns of $\tilde{V_p}$ are the $\tilde{p}$ eigenvectors with largest eigenvalues. The percentage

$$\eta_i = \frac{\lambda_i}{\sum\limits_i \lambda_i}.$$

Stochastic Neighborhood Embedding t-stochastic neighborhood embedding (t-SNE) is a technique meant for capturing different kinds of structure in data. It is non-parametric and, most notably, non-linear.

We will need to introduce some ideas first. We can assign a probability distribution to the neighborhood of each data point. The probability that x_i is the neighborhood of x_j is given by

$$P(x_j \mid x_i) = \frac{e^{-\frac{|x_i - x_j|}{2\sigma_i}}}{\sum\limits_{k \neq i} e^{-\frac{|x_i - x_k|}{2\sigma_i}}}.$$

We also define $p_{i \mid j}$ to be the likelihood that x_i is in the neighborhood of x_j and take $p_{i \mid i} = 0$. The σ_i are determined by fixing the entropy

$$H(p_i) = -\sum_{j \neq i} p_{j \mid i} \log_2(p_{j \mid i}),$$

or equivalently the perplexity $\Sigma = 2^{H(p_i)}$. This will entail that points in dense regions have small σ_i. Outliers, however, contribute very weakly to all neighborhoods, making their assignment difficult. One can instead define a symmetrized distribution

$$p(x_j \mid x_i) = \frac{1}{2N}(P(x_j \mid x_i) + P(x_i \mid x_j)),$$

which is not vanishing.

The idea of the method is to embed the high-dimensionality neighborhood into a lower-dimensional space in a way that preserves neighborhood relations. In this lower-dimensionality space, where the data points have coordinates y_i, we assign the probability distributions

$$q(y_j \mid y_i) = \frac{\frac{1}{1+|y_j-y_i|^2}}{\sum\limits_{k\neq i} \frac{1}{1+|y_k-y_i|^2}}.$$

The method then matches the lower- and higher-dimensionality distributions by minimizing the Kullback-Leibler divergence

$$D_{\mathrm{KL}}(p,q) = \sum_{i,j} p(x_j \mid x_i) \ln\left(\frac{p(x_j \mid x_i)}{q(y_j \mid y_i)}\right).$$

While t-SNE is powerful, note that it can rotate data. The results are also stochastic. While the method preserves short-distance information, it also deforms scale. The method is also computationally expensive, its efficiency only recently being obtained at the time of writing.

Clustering Clustering algorithms are algorithms that classify data point. Good clustering outputs categorize similar data points into the same category and different data points into different categories. These requirements can be contradictory, a by-construction limitation of these methods.

KMeans Given a set of observations and a number K, KMeans divides the data into a set of K clusters X_i. It does this by minimizing the cost function

$$C(x,\mu) = \sum_{i=1}^{K} \sum_{j\in X_i} d(x_j,\mu_i),$$

where the μ_i is the in-cluster mean and d is the metric function. This problem is NP-hard, so finding optimal solutions is computationally infeasible. Iterative solutions generally work well, but the output might differ from time to time. The iterative solutions are typically based on the following steps:

1. perform some (typically randomized) initial assigment.
2. compute the in-sample means.
3. reassign data points according to which in-cluster mean they are closest to.
4. repeat from steps 2.

One usually also repeats the procedure different times, choosing the run with the minimal cost function.

DBSCAN

Cluster Agglomeration