Computational Intelligence II - Project 1

Pedro Henrique Pamplona Savarese - phpsavarese@gmail.com

Code: https://github.com/lolemacs/overfitting-experiment

Introduction

What differs Machine Learning from traditional parameter optimization? In both, we usually have a problem defined by a set of parameters Θ which we want to learn (perform optimization on) in order to minimize a cost function.

One of the main differences (if not the most crucial one) is that, in parameter optimization, we are interested in the cost given by the data used for the optimization itself - in Machine Learning terminology, we're only interested on the training set performance:

$$E_{in}(X, Y, h, \Theta) = \frac{1}{N} \sum_{i=1}^{N} Cost(h(X_i, \Theta), Y_i)$$

Where X and Y denote the dataset (data points and their correspondent mappings), h is the hypothesis (a second-degree polynomial, a decision tree, a perceptron, etc), Θ is the set of h's parameters (the parameters we want to perform search on), and $Cost$ is a function that defines the cost given a prediction and the ground-truth (squared difference, negative log-likelihood, categorical cross-entropy, etc).

Note that in theoretical ML terminology, Θ is implicit in h: we say that h is a hypothesis defined by a set of parameters Θ applied to the hypothesis class H. We will stick with this terminology during this report.

In Machine Learning our aim is to design models that can deal with novel data - just like humans do. For that, minimizing the error on a training set isn't enough: we also need a model that has generalization power.

The aim of this project is to perform an empirical study on the overfitting phenomena - that is, when a model presents a satisfying performance on the training set, but performs poorly on novel data.

The project is divided in 2 experimental setups. On the first one, our goal is to analyze the relation between the size of our dataset, its noise and generalization. On the second, we analyze the complexity of the target concept instead of data noise. Yaser calls the first type of noise stochastic, and the latter deterministic - which is strongly associated with the probability of choosing bad hypotheses during the learning procedure. Deterministic noise can be seen as a way to interpret the relation

between Valiant's PAC Learning model and the hypotheses' set Vapnik-Chervonenkis Dimension.

Methodology

For each experiment, we start by defining the target concept and the hypotheses classes. In both we use a concept class consisting of polynomials for both the target and the hypothesis.

We further restrict the class C of target concepts to Legendre polynomials following a specific normalization. We define such concept class, when restricted to polynomials of degree Q_f, as C_{Q_f}.

We use two hypotheses classes for each experiment: the class of 2nd-degree and 10th-degree polynomials, denoted as H_2 and H_{10}.

Both experiments consist of several runs using different setups (σ^2 and N for the first, Q_f and N for the second), where for each run we generate a new target concept $c \in C$ and N data points $X = (x_1, \ldots, x_N), Y = c(X) + \sigma\epsilon$.

After generating a dataset, we find two hypotheses $h_2 \in H_2, h_{10} \in H_{10}$ using linear least squares. We use $E_{out}(h_{10}) - E_{out}(h_2)$ as a surrogate metric to analyze generalization: when this value is negative, H_{10} is better suited as a hypothesis class to learn C_{Q_f} than H_2 - that is, the model's increased complexity overcomes the negative effects of a possible overfitting.

We proceed to give detailed descriptions on the experiments:

$c \in C$ Generation

The first step for both experiments is to generate a target concept c. The concept class C is defined by a Legendre polynomial with $E_x[c^2] = 1$ (the expected value of the squared concept equals 1). This definition depends on how X is generated (due to the the dependency on the probability density of X): we will sample $x \in X$ uniformly in the $[-1, 1]$ domain: $X \sim U(-1, 1), P(x) = \frac{1}{2}$

To generate c, we first sample a set of coefficients a ($Q_f + 1$ coefficients for a Q_f-th degree polynomial) from a standard normal distribution: $a \sim \mathcal{N}(0, 1)$. Next, we need to perform a normalization such that $E_x[c^2] = 1$.

For this step, it suffices to first calculate the squared norm λ^2:

$$\lambda^2 = \int_{-1}^{1} P(x)c^2(x)dx = \frac{1}{2}\int_{-1}^{1} c^2(x)dx$$

And dividing c's coefficients by such norm:

$$a_{norm} = \frac{a}{\lambda^2}$$

We guarantee that $E_x[c^2] = 1$, as desired.

Y generation

In both experiments we add a noise factor $\sigma\epsilon$ to $c(X)$ in order to generate Y. For that, we sample N observations from a normal distribution with 0 mean and σ^2 variance and add them to the N mapped values $c(X)$, having then $Y = c(X) + \sigma\epsilon$.

$h \in H$ Search

We need to find good hypotheses $h_2 \in H_2$ and $h_{10} \in H_{10}$ for each run. First, remember that we define $h_m \in H_m$ as such:

$$h_m = \sum_{i=0}^{m} a_m x^m$$

That is, a classic m-degree polynomial with coefficients $a = (a_0, \ldots, a_m)$. The set of parameters a can also be used to define h_m, like noted previously - meaning that finding an hypothesis $h_m \in H_m$ is equivalent to finding a good set of parameters a_m.

For this step we first define our cost function as the classic squared error:

$$Cost(h_m(x), y) = (h_m(x) - y)^2$$

Using the total cost as the sum/mean over the training set grants us the ability to perform closed-form least squares minimization:

$$a_m = (X^T X)^{-1} X^T Y$$

We thus use h_m defined as the polynomial from the class H_m with coefficients a_m.

Calculating $E_{out}(h_{10}) - E_{out}(h_2)$

Using the previously given definitions of E_{in} and $Cost$, we define $E_{out}(h)$ as:

$$E_{out}(h) = \frac{1}{N} \sum_{i=1}^{N} (h(X_i) - Y_i)^2$$

Where here X and Y are composed of data points that were **not** using for training (that is, they were not used to find h).

For both experiments we use $N = 200$ as the number of data points to calculate out-of-sample error. For both experiments the maximum number of data points in the training set is 200, thus this number was chosen since it guarantees that the out-of-sample error uses a population at least as big as the one used for training.

General Procedure

Each experiment run can be thus described as a sequence of the well-defined steps above:

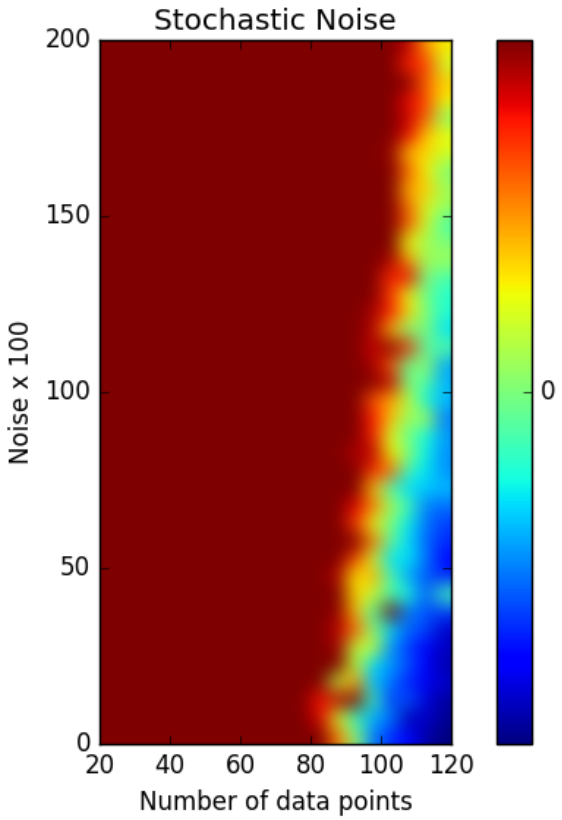
- Given N, σ^2 and Q_f
- Sample $N + 200$ values from $U(0, 1)$ to generate X.
- Sample $Q_f + 1$ values from $\mathcal{N}(0, 1)$ to generate an unormalized Legendre polynomio c_{Q_f}.
- Normalize the generated polynomio such that $E_x[c^2_{Q_f}] = 1$.
- Sample $N + 200$ values from $\mathcal{N}(0, \sigma^2)$, add to $c_{Q_f}(X)$ to generate Y.
- Find $h_2 \in H_2$ and $h_{10} \in H_{10}$ using the first N points from X and Y.
- Calculate $E_{out}(h_{10}) - E_{out}(h_2)$ using the last 200 points from X and Y.

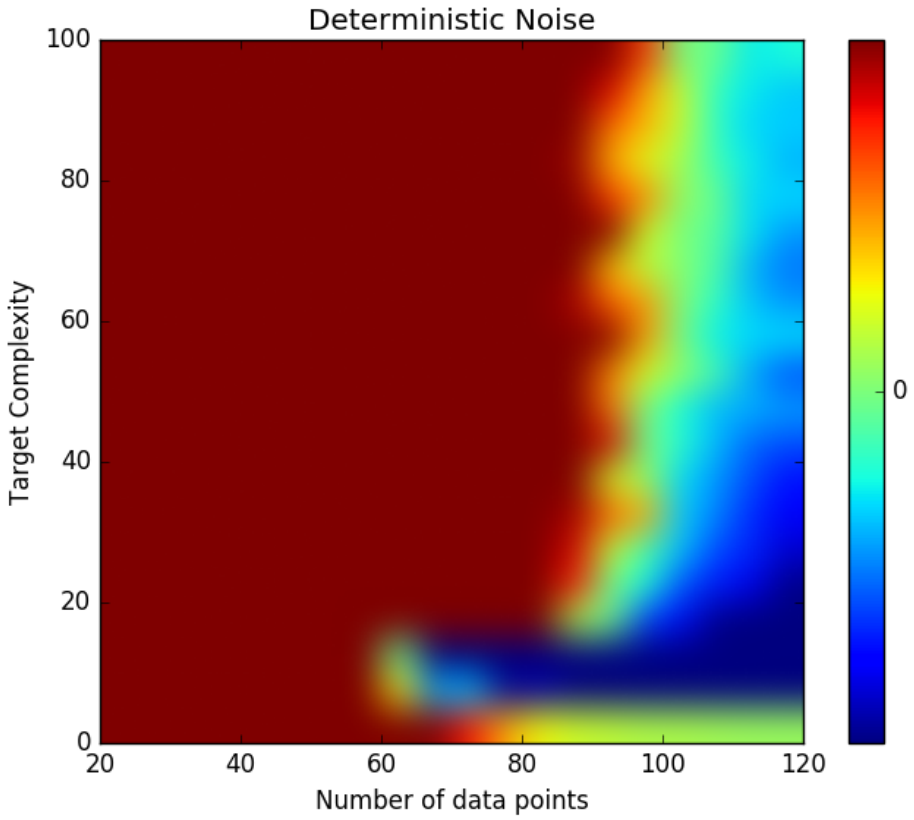
For the first experiment, we use $\sigma^2 = (0, 0.05, 0.1, \ldots, 2.0)$, $N = (20, 25, 30, \ldots, 200)$ and $Q_f = 20$. For the second, we use $Q_f = (0, 5, 10, \ldots, 100)$, $N = (20, 25, 30, \ldots, 200)$ and $\sigma^2 = 0.1$.

Several runs are performed, and the mean is taken to plot a graph.

Results

After running each experiment set for around 30.000 runs, the following graphs were generated (the color scale goes from -0.2 to 0.2):





Discussion

From the first experiment, we can take a few interesting observations. First, even though h_{10} has half the number of degrees of freedom than c_{20}, it performs worse than h_2 if less than around 90 data points are provided for training - even when no noise is present. This agrees with the famous rule of thumb used in practice by data scientists: if a hypothesis has m parameters, then around m^2 data points are a rough minimum amount to prevent severe overfitting. We can also see that the number of needed data points to avoid overfitting grows linearly with the noise level.

From the second experiment, the most interesting phenomena is the transition phase that happens at $Q_f = 15$ and the exponential grow on Q_f. That is, for a number of data points between 60 and 90, h_{10} doesn't overfit if $5 < Q_f < 15$. For $N > 90$, h_{10} is better unless Q_f is really small (roughly less than 4).