Classification in Big Data Environments

Clemens Damke

Intelligent Systems and Machine Learning Group (ISG)
Heinz Nixdorf Institute
Paderborn University
Warburger Straße 100
33098 Paderborn

Abstract. *TODO*

Keywords: Big Data · Automated Machine Learning

1 Introduction

Over the last few years Big Data processing has become increasingly important in many domains. This increase in the data volume also poses new challenges for machine learning applications. The training time of learners is usually polynomially dependent on the size of the training dataset $\mathcal{D}_{train}$, i. e. $\mathcal{O}(|\mathcal{D}_{train}|^{\alpha}), \alpha \geq 1$. Since training has to be repeated for every iteration of cross-validation and hyperparameter search, always using the entire dataset quickly becomes infeasible. This paper gives an overview of approaches to tackle this problem with a focus on the domain of classification problems.

The process of finding a hypothesis can in general be split into three phases:

1. **Model selection:** At first a model has to be selected that determines the class of hypothesis spaces out of which the final hypothesis will be selected. The choice of model is often implicitly encoded in a learner L. Automating this step is non-trivial. In practice the model is typically selected by experts with domain specific knowledge about the problem at hand.

2. **Hyperparameter search:** Optimizing a vector λ in the hyperparameter space Λ_L of the learner L representing a hypothesis space $\mathcal{H}_\lambda$. A naïve approach to do this is to systematically try configurations using a grid search or a random search over Λ_L. To evaluate the quality of a given λ, L is usually trained on a training dataset $\mathcal{D}_{train}$ using λ. This yields a hypothesis $\hat{h}_\lambda \in \mathcal{H}_\lambda$ that is evaluated using a validation dataset $\mathcal{D}_{valid}$. The goal of hyperparameter optimization is to minimize the loss $l(\lambda)$ of $\hat{h}_\lambda$ on $\mathcal{D}_{valid}$, i. e. to find an approximation $\hat{\lambda}$ of $\lambda^* := \arg\min_\lambda l(\lambda)$.

3. **Training or parameter search:** Let w be a vector in the parameter space $W_{\mathcal{H}_\lambda}$, describing a hypothesis $h_{\lambda,w} \in \mathcal{H}_\lambda$ given a hyperparameter configuration λ. The goal of parameter search is to find an approximation $\hat{h}_\lambda$ of the hypothesis $h^*_\lambda := \arg\min_{h_{\lambda,w}} \ell(\mathcal{D}_{train}|h_{\lambda,w})$, with $\ell(\mathcal{D}_{train}|h_{\lambda,w})$ being the empirical loss of $h_{\lambda,w}$ on a given training dataset $\mathcal{D}_{train}$ according to some loss function ℓ. Depending on the learner L, various kinds of optimization methods are used to find this minimum, e. g. Bayesian optimization, quadratic programming or, if $\nabla_w e(\mathcal{D}_{train}|h_{\lambda,w})$ is

computable, gradient descent. The quality l of $\hat{h}_\lambda$ is measured by the loss on a validation or test dataset, i. e. $l(\lambda) := \ell(\mathcal{D}_{valid}|\hat{h}_\lambda)$.

This paper is structured according to the last two phases, i. e. we will assume that the learner L is given. Section 2 describes ways to speed up the hyperparameter search. Section 3 then describes how to improve the training methods of existing learners. Most of the techniques described in this paper improve upon independent components of the hypothesis finding process allowing them to be combined.

2 Hyperparameter optimization

As described in the introduction, the goal of hyperparameter optimization is to find a global minimum of l. Since l is generally unknown, analytical methods or gradient descent cannot usually be applied. The only way the get information about l is to evaluate it, which is costly. There are multiple ways to reduce the total cost of those evaluations:

1. **Number T of evaluations of l:** During optimization multiple hyperparameter configurations $\lambda_1, \ldots, \lambda_T$ will be evaluated using l. T is usually fixed when using a grid search or a random search. After evaluating T configurations, the best one is chosen. Those naïve approaches assume that $l(\lambda)$ is independent of $l(\lambda')$ for all pairs $\lambda \neq \lambda'$. We will see that this strong assumption of independence is not necessarily true which in turn allows reducing T.
2. **Training dataset size S:** The performance of a given configuration $l(\lambda)$ is computed by training the learner on $\mathcal{D}_{train}$ which is expensive for big datasets. By training on S instead of $|\mathcal{D}_{train}|$ datapoints the evaluation can be sped up.
3. **Number of training iterations E:** Depending on the learner, training often is an iterative process, e. g. gradient descent. To speed up hyperparameter optimization training could be terminated before convergence.

2.1 FABOLAS

The first approach we will discuss is called Fabolas (Fast Bayesian Optimization of Machine Learning Hyperparameters on Large Datasets) [4]. It can be applied to any learner L and is based upon two main ideas:

1. The validation loss l is modeled as a *Gaussian process* (GP) f based on the assumption that two configurations λ and λ' will perform similar if they are similar according to some kernel $k(\lambda, \lambda')$. The Gaussian process f is used as a surrogate to estimate the expected value and variance of l given λ. Using *Bayesian optimization* l will be probed at promising positions to iteratively improve f. Hyperparameter configurations that are expected to perform worse than the current optimum will not be probed. This effectively reduces T.
2. The training dataset size S is modeled as an additional hyperparameter of f giving the optimizer an additional degree of freedom. This allows extrapolating the value of l when trained on the complete dataset while only probing smaller subsets which effectively reduces S.

We will now describe how those two ideas can be applied.

Gaussian processes A Gaussian process is a family of *random variables* (RVs) $(X_\theta)_{\theta\in\Theta}$, s. t. every finite subset of them follows a multivariate normal distribution. More intuitively it can be understood as a probability distribution over functions $f : \Theta \to \mathbb{R}$ where $X_\theta \mathrel{\widehat{=}} f(\theta)$. Prior knowledge about the likelihood of each f is described by a prior mean function $\mu_0(\theta) = \mathbb{E}[f(\theta)]$ and a positive-definite kernel $k(\theta_1, \theta_2) = \mathrm{Cov}(f(\theta_1), f(\theta_2))$. Let $\mathcal{D}_n = \{(\boldsymbol{\theta}_i, \boldsymbol{y}_i)\}_{i=1}^n$ denote a set of observations. Those observations can be used to update the means and variances of the RVs via GP regression. This collapses the space of possible functions f to those functions that align with $\mathcal{D}_n$:

$$\begin{aligned}
\boldsymbol{m} &:= (\mu_0(\boldsymbol{\theta}_1), \ldots, \mu_0(\boldsymbol{\theta}_n))^T \\
\boldsymbol{k}(\theta) &:= (k(\boldsymbol{\theta}_1, \theta), \ldots, k(\boldsymbol{\theta}_n, \theta))^T \\
\boldsymbol{K} &\in \mathbb{R}^{n\times n}, \boldsymbol{K}_{ij} := k(\boldsymbol{\theta}_i, \boldsymbol{\theta}_j) \\
\mathbb{E}[f(\theta) \,|\, \mathcal{D}_n] &:= \mu_n(\theta) = m_0(\theta) + \boldsymbol{k}(\theta)^T \boldsymbol{K}^{-1}(\boldsymbol{y} - \boldsymbol{m}) \qquad (1) \\
\mathrm{Cov}(f(\theta), f(\theta') \,|\, \mathcal{D}_n) &:= k(\theta, \theta') - \boldsymbol{k}(\theta)^T \boldsymbol{K}^{-1} \boldsymbol{k}(\theta') \qquad (2)
\end{aligned}$$

Fabolas works by modeling the loss function l as a Gaussian process $f \sim \mathcal{GP}(m, k)$ with parameter set $\Theta := \Lambda \times [0, 1]$ where $\mu_0(\lambda, s) = \mathbb{E}[f(\lambda, s)] = \mathbb{E}[l(\lambda) \,|\, \text{training size } s]$. To model the covariances between different combinations of hyperparameters and training set sizes, the following product kernel is used:

$$k((\lambda, s), (\lambda', s')) := k_{5/2}(d(\lambda, \lambda')) \cdot k_{\mathrm{lin}}(s, s') \qquad (3)$$

Here $k_{5/2}$ denotes the stationary Matérn kernel with d being the Mahalanobis distance between the two compared hyperparameter configurations. k_{lin} essentially is a simple linear kernel modeling the assumption that l monotonically decreases when s is increased. Figure 1 gives an intuition for this choice of kernel. We refer to Klein et al. [4] for the details.

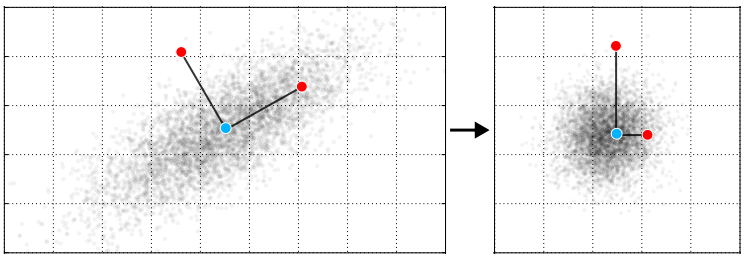

Fig. 1: Intuition for the Mahalanobis distance. Using the Euclidean distance the red points would be equally far away from the blue one. The Mahalanobis distance fixes this by first normalizing hyperparameters.

Bayesian optimization To find $\arg\min_\lambda l(\lambda)$ the bias and variance of f has to be reduced by probing l at promising positions. This is called Bayesian optimization.

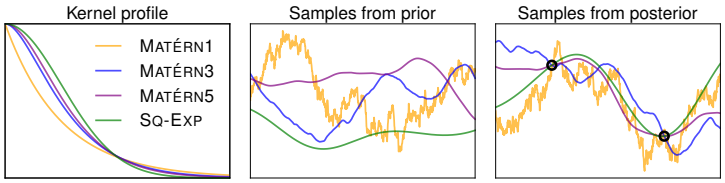

Fig. 2:

The estimated minimum after n probes is described by $\arg\min_\lambda \mu_n(\lambda, s = 1)$, i. e. the configuration with the smallest predicted error on the full test dataset. To reduce the number of probes required until this minimum converges, an *acquisition function* is used. Its role is to trade-off exploration vs. exploitation of l by describing the expected utility of probing (λ_{n+1}, s_{n+1}) given a set of previous probes $\mathcal{D}_n$. Fabolas uses an aquisition function that rates configurations by their *information gain* per computation time. It measures the expected amount of available information, i. e. entropy H, about the optimal configuration if a given configuration were probed. computation time c of a given configuration via a separate Gaussian process

$$a_F(\lambda, s) := \frac{1}{c(\lambda, s)} \mathbb{E}_y \left[p(y \mid \lambda, s, \mathcal{D}_n) \cdot \mathrm{H}_{\hat{\lambda}}(p_{\min}(\hat{\lambda} \mid \mathcal{D}_n \cup \{(\lambda, s, y)\})) \right] \tag{4}$$

$$p_{\min}(\lambda \mid \mathcal{D}) := p(\lambda \in \arg\min_{\lambda'} f(\lambda', s = 1) \mid \mathcal{D})$$

Since it is infeasible to compute a_F numerically its maximum is estimated using *Markov-Chain Monte Carlo* (MCMC). The estimated most promising configuration will be probed. The resulting loss value and runtime are then used to update the loss model f and cost model c via GP regression.

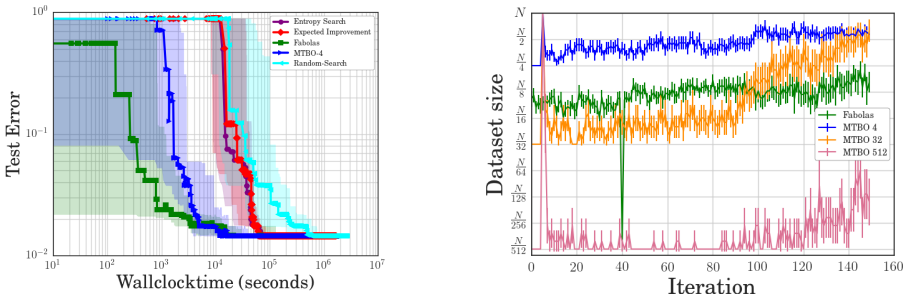

Fig. 3: *TODO*

Evaluation

2.2 Learning Curve Extrapolation

The second approach for speeding up hyperparameter optimization focuses on reducing the number of training iterations E. It can in principle be applied to any iterative learner and can be integrated into any hyperparameter optimizer. The idea

is to monitor the learning curve of a learner during training with a hyperparameter configuration λ. If it is unlikely that a good accuracy will be reached with λ, training will be terminated before convergence.

The method was first described by Domhan et al. [3] in the context of hyperparameter optimization for *deep neural networks* (DNNs) that are trained using stochastic gradient descent.

TODO: explain the extrapolation model

3 Optimizing Training

Now follows an overview of approaches to speed up the training process. We will discuss four approaches:

1. A general purpose method that combines subsampling with bootstrapping.
2. An iterative method to select the optimal subsample size during gradient descent.
3. Improving the quality of subsampling for logistic regression by weighing the samples.
4. Speeding up the training of SVMs via k-means clustering.

3.1 Bag of Little Bootstraps

The first approach we will discuss is called *Bag of Little Bootstraps* [5]. It combines subsampling with bootstrapping and is particularly well suited for parallelized implementations.

TODO: explain BLB

3.2 Subsample Size Selection for Gradient Descent

Next we will discuss gradient descent based optimization. The sample size S that describes the number of datapoints that are considered in a single gradient descent step heavily influences the optimizer's behavior:

- In the stochastic approximation regime small samples, typically $S = 1$, are used. This causes fast but noisy steps.
- In the batch regime large samples, typically $S = |\mathcal{D}_{train}|$, are used. Steps are expensive to compute but more reliable.

Both extremes are usually not suitable for Big Data applications. Very small samples cannot be parallelized well, making them a bad fit for the compute clusters that are typically available nowadays. The gradients for very big samples however are often too slow to compute. S should ideally lie somewhere in between.

Byrd et al. [2] describe an iterative algorithm that dynamically increases S as long as this promises to significantly reduce the gradient noise.

3.3 Subsampling for Logistic Regression

Subsampling usually increases the mean squared error (MSE) of the resulting hypothesis compared to one that is trained on the full dataset. OSMAC [6] is a method that improves upon naïve subsampling by weighing the samples.

TODO: explain OSMAC

3.4 SVM-KM

To speed up the training of SVMs Almeida et al. [1] proposed a simple method that reduces the dataset size via k-means clustering. It can be described as a three-step procedure:

1. Group the training samples $\mathcal{D}_{train}$ into k clusters $C_1, \ldots, C_k$ with centers $c_1, \ldots, c_k$ where k should be determined via hyperparameter optimization.
2. Check for each cluster C_i whether all associated datapoints belong to the same class, i. e. $\exists\, z \in \{+1, -1\} : \forall (x, y) \in C_i : y = z$. If yes, all datapoints in C_i are removed from $\mathcal{D}_{train}$ and replaced by c_i. If not, they are kept in the dataset. The intuition behind this is that clusters with points from multiple classes might be near the decision boundary so they are kept to serve as potential support vectors.
3. Finally standard SVM training is performed on the reduced training dataset.

Evaluation *TODO*

4 Conclusion

TODO

References

[1] M. Barros de Almeida, A. de Padua Braga, and J. P. Braga. "SVM-KM: speeding SVMs learning with a priori cluster selection and k-means". In: *Proc. Vol.1. Sixth Brazilian Symp. Neural Networks*. Nov. 2000, pp. 162–167 (cit. on p. 6).

[2] Richard H. Byrd, Gillian M. Chin, Jorge Nocedal, and Yuchen Wu. "Sample size selection in optimization methods for machine learning". In: *Mathematical Programming* 134.1 (2012), pp. 127–155 (cit. on p. 5).

[3] Tobias Domhan, Jost Tobias Springenberg, and Frank Hutter. "Speeding Up Automatic Hyperparameter Optimization of Deep Neural Networks by Extrapolation of Learning Curves". In: *Proceedings of the 24th International Conference on Artificial Intelligence*. IJCAI'15. Buenos Aires, Argentina: AAAI Press, 2015, pp. 3460–3468 (cit. on p. 5).

[4] Aaron Klein, Stefan Falkner, Simon Bartels, Philipp Hennig, and Frank Hutter. "Fast Bayesian Optimization of Machine Learning Hyperparameters on Large Datasets". In: (May 23, 2016). arXiv: `1605.07079v2 [cs.LG]` (cit. on pp. 2, 3).

[5]Ariel Kleiner, Ameet Talwalkar, Purnamrita Sarkar, and Michael I. Jordan. “A Scalable Bootstrap for Massive Data”. In: (Dec. 21, 2011). arXiv: 1112.5016v2 [stat.ME] (cit. on p. 5).

[6]HaiYing Wang, Rong Zhu, and Ping Ma. “Optimal Subsampling for Large Sample Logistic Regression”. In: (Feb. 3, 2017). arXiv: 1702.01166v2 [stat.CO] (cit. on p. 6).