NICE
ACTIMIZE

Online Machine Learning: Incremental Online Learning

Part 2

Insights Article

Danny Butvinik
Chief Data Scientist | NICE Actimize

Abstract

Incremental and online machine learning recently have received more and more attention, especially in the context of learning from real-time data streams as opposed to a traditional assumption of complete data availability. Although a variety of different methods are available, it often remains unclear which are most suitable for specific tasks, and how they perform in comparison to each other. This article reviews the eight popular incremental methods representing distinct algorithm classes.

Introduction

My previous article served as an introduction to online machine learning. This article will provide an overview of a few online incremental learning algorithms (or instance-based incremental learning), that is, the model is learning each example as it arrives. Classical batch machine learning approaches, in which all data are simultaneously accessed, do not meet the requirements to handle the sheer volume in the given time. This leads to more and more accumulated data that are unprocessed. Furthermore, these approaches do not continuously integrate new information into already constructed models, but instead regularly reconstruct new models from scratch. This is not only very time consuming but also leads to potentially outdated models.

Overcoming this situation requires a paradigm shift to sequential data processing in a streaming scheme. This not only allows us to use information as soon as it is available, leading to models that are always up to date, but also reduces the costs for data storage and maintenance.

Incremental and online algorithms fit naturally to this scheme, since they continuously incorporate information into their model, and traditionally aim for minimal processing time and space. Because they can provide continuous largescale and real-time processing, they recently gained more attention particularly in the context of data streams.

Incremental algorithms are also very suitable for learning beyond the production phase, which enables devices to adapt to individual customer habits and environments. This is particularly interesting for financial institutions where fraud detection is one of the centric methods to fight financial crime. **NICE Actimize** — the world leader in crime monitoring, management, and prevention – is focused on detecting fraud in real-time. This is a time-series environment with streams of data. It is very difficult to tackle such tasks, especially when you have highly imbalanced data and issues with ground truth, and only a tiny percentage of the data are fraudulent. While the remaining data are assumed to be legitimate, there is no assurance that among legitimate data there is no fraud. That's why trying to solve the problem using the online incremental approach is game-changing for that domain.

Here the main challenge is not large-scale processing, but rather continuous and efficient learning from few data. Even though incremental learning could be replaced in this case by repetitive batch learning in the cloud, that solution has crucial drawbacks. A permanent connection to the cloud is required to provide anytime models, which may not always be feasible. Furthermore, customers may not be willing to provide data about their daily life due to privacy reasons. Hence, learning directly on the device in an efficient way is still very desirable. A lot of ambiguity is involved regarding the definition of incremental and online learning in the literature. Some authors use them interchangeably, while others distinguish them in different ways. Additional terms such as lifelong or evolutionary learning are also used interchangeably. We define an incremental learning algorithm as one that generates on a given stream of training data

$$s_1, s_2, \ldots, s_t$$

a sequence of models

$$h_1, h_2, \ldots, h_t$$

In our case,

$$s_i = (x_i, y_i) \in R^n \times \{1, \ldots, C\}$$

and

$$h_i : R^n \times \{1, \ldots, C\}$$

is a model function solely depending on

$$h_{i-1}$$

and the recent ***p*** examples

$$s_i, \ldots, s_{i-p}$$

with ***p*** being strictly limited. We specify online learning algorithms as incremental learning algorithms, which are additionally bounded in model complexity and run-time, capable of endless/lifelong learning on a device with restricted resources.

Incremental learning algorithms face the following challenges:

- The model has to adapt gradually – i.e., is constructed without complete retraining.
- Preservation of previously acquired knowledge and without the effect of **catastrophic forgetting**.
- Only a limited number of p training examples are allowed to be maintained.

We explicitly assume the data to be labeled and do not focus on the crucial scenario of learning from unlabeled or partially labeled data streams. The setting of supervised incremental learning can be applied in most prediction scenarios. In these, after a system has made a prediction the true label can often be inferred with some delay.

An algorithm has to be chosen according to the preconditions of a given task since there cannot exist a method that optimally performs in every scenario. To date, various interesting incremental learning algorithms have been published with various strengths and weaknesses. However, there are only a few sources providing information about them. That's because no comparative in-depth study, experimentally comparing the most popular methods according to the most relevant criteria, is available. Extensive literature research usually leads to the original publications of considered algorithms.

This article will analyze the core features of some of the online incremental methods. My focus lies in the classification under supervised learning for online incremental algorithms. I primarily perform an evaluation on stationary datasets (i.e. the assumption is that the stream

$$s_1, s_2, \ldots,$$

is independent identically distributed). However, I briefly evaluate and discuss the methods in the context of concept drift.

Algorithms

Incremental Support Vector Machine (ISVM) is the most popular exact incremental version of the SVM and was introduced in my article about ISVM (click here). Additionally, a limited number of examples – so-called "candidate vectors" – is maintained. These are examples which could be promoted to support vectors depending on the future examples. The smaller the set of candidate vectors, the higher the probability of missing potential support vectors. The ISVM is a lossless algorithm — it generates the same model as the corresponding batch algorithm — if the set of candidate vectors contains all previously seen data.

LASVM is an online approximate SVM solver. In an alternative manner, it checks whether the currently processed example is a support vector and then removes obsolete support vectors. For both steps, it heavily utilizes sequential direction searches as it is also done in the Sequential Minimal Optimization (SMO) algorithm. In contrast to the ISVM, it does not maintain a set of candidate vectors; instead, it only considers the current example as possible support-vector. This leads to an approximate solution but significantly reduces the training time.

Online Random Forest (ORF) is an incremental version of the Random Forest algorithm. A predefined number of trees grows continuously by adding splits whenever enough samples are gathered within one leaf. Instead of computing locally optimal splits, a predefined number of random values are tested according to the scheme of Extreme Random Trees. The split value, which most optimizes the Gini index, is selected. Tree ensembles are very popular, due to their high accuracy, simplicity and parallelization capability. Furthermore, they are insensitive to feature scaling and can be easily applied in practice.

Incremental Learning Vector Quantization (ILVQ) is an adaptation of the static Generalized Learning Vector Quantization (GLVQ) to a dynamically growing model, which inserts new prototypes when necessary. The insertion rate is guided by the number of misclassified samples. We use the version that introduces a prototype placement strategy, minimizing the loss on a sliding window of recent samples. Metric learning, as described in [38, 39], can also be applied to further extend the classification abilities.

Learn++ (LPPCART) processes incoming samples in chunks with a predefined size. For each chunk, an ensemble of base classifiers is trained and combined through weighted majority voting to an “ensemble of ensembles.” Like the AdaBoost [41] algorithm, each classifier is trained with a subset of chunk examples drawn according to a distribution, ensuring a higher sample probability for misclassified inputs. LPP is a model-independent algorithm and several different base classifiers such as SVM, Classification and Regression Trees (CART) and Multilayer Perceptron have been successfully applied by the authors. As the original author, we employ the popular CART as base classifiers. Chunk-wise trained models inherently incorporate an adaption delay depending on the chunk size. This algorithm was recently utilized.

Incremental Extreme Learning Machine (IELM) reformulates the batch ELM least-squares solution into a sequential scheme. As the batch version, it drastically reduces the training complexity by randomizing the input weights. The network is static, and the number of hidden neurons must be predefined. This method can process the data one-by-one or in chunks, which significantly reduces the overall processing time. However, a valid initialization of the output weights requires at least as many examples as the number of used hidden neurons.

Naive Bayes (NBGauss) fits one axis-parallel Gaussian distribution per class and uses them as a likelihood estimation in the Naive Bayes algorithm. The sparse model allows very efficient learning in terms of processing time and memory requirements. This algorithm learns efficiently from few training examples and has been successfully applied in real-world situations like spam filtering and document classification. The major drawbacks of this lossless algorithm are the independence assumption of the features, as well as its inability to handle multimodal distributions.

Stochastic Gradient Descent (SGDLin) is an efficient optimization method for learning a discriminative model by minimizing a loss function such as the Hinge — or Logistic loss. We use SGD to learn a linear model by minimizing the Hinge loss function. SGD has been recently revived in the context of large-scale learning. When coupled with linear models, it performs especially well for sparse, high-dimensional data as often encountered in the domain of text classification or natural language processing. However, linear models are a misfit whenever non-linear class boundaries are required, which is particularly often the case for low dimensional data.

Even though new versions of the algorithms are continuously proposed, we argue that the chosen methods reflect the general properties of the respective family. Therefore, the conclusions in this paper are commonly applicable for current and upcoming variations of the corresponding algorithm. This is particularly highlighted by both SVMs, which perform very similar to the difference that LASVM can process slightly larger datasets due to its approximate nature. However, both share the same drawbacks regarding large or noisy datasets. These drawbacks are also shared by a recently proposed LASVM version, albeit to a lesser degree, since a mechanism to reduce the number of support vectors is introduced. Various extensions for the LPP and the IELM algorithm have been proposed. Most of them are tackling non-stationary environments by introducing forgetting mechanisms. However, the major focus of this article is incremental learning in stationary environments, rather than forgetting, which is rather harmful and deteriorates the performance.

Furthermore, the basic principle of the algorithms and the corresponding advantages and disadvantages remain. In the case of LPP, it is the flexibility of arbitrary base classifiers on the one hand, and the limited knowledge integration across chunks on the other. Methods for speeding up the convergence of SGD were presented. However, the results obtained by the SGD algorithm in our experiments are not due to slow convergence of the SGD algorithm, but rather highlight the general benefits and limitations of linear models, such as a low model complexity and linear class boundaries.

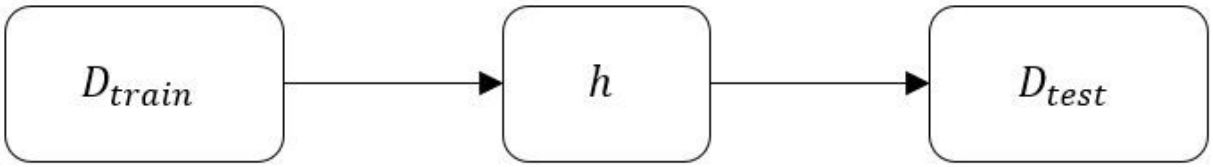

Fig 1: Classical scheme of evaluating a batch algorithm on offline mode

The learning objective in supervised classification is to predict a target variable y ∈ {1, . . . , c} given a set of features. We consider two different evaluation settings which allow the inference of different aspects regarding the algorithmic performance and provide together even deeper insight.

Offline Setting

In the offline setting, a batch algorithm generates a model ***h*** based on a training set

$$D_{train} = \{(x_i, y_i) \mid i \in \{1, \ldots, j\}\}$$

In the subsequent test phase, the model is applied to another set

$$D_{test} = \{(x_i, y_i) \mid i \in \{1, \ldots, k\}\}$$

whose labels are kept hidden. Figure 1 depicts the process. The model predicts a label

$$\hat{y}_i = h(x_i)$$

$$\hat{y}_i = h(x_i)$$

for every point in the testing set and the 0–1 loss for every point

$$x_i \in D_{test} \text{ and the 0-1 loss } \mathcal{L}(\hat{y}_i, y_i) = 1\ (\hat{y}_i \neq y_i)$$

is calculated. The average accuracy on the test set enables analysis in terms of the generalization ability to unseen examples.

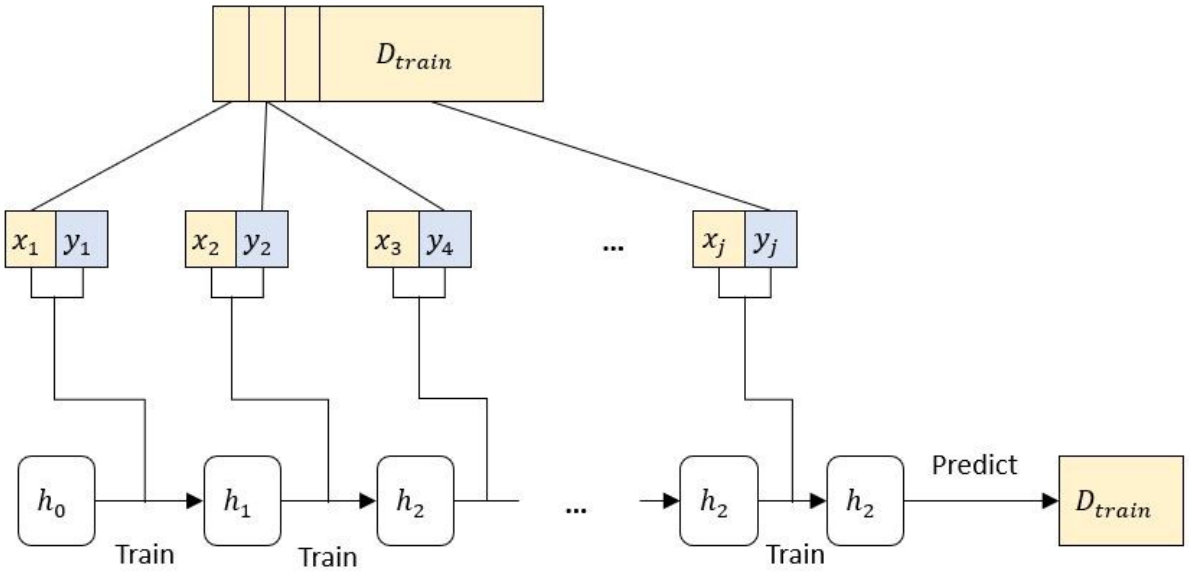

Figure 2: The process of testing an incremental algorithm in the off-line setting. Noticeably, only the last constructed model is used for prediction. All data used during the training set is obtained from the D training set.

The evaluation of an incremental algorithm in this setting is different as shown in Figure 2. Instead of accessing all training data at once, it is sequentially processed in predefined order. The algorithm generates the sequence of tuples

$$(x_1, y_1)\ , (x_2, y_2)\ , \ldots (x_j, y_j)$$

a corresponding sequence of models

$$h_1, h_2, \ldots, h_j$$

Thereby, a model

$$h_i$$

is solely based on the previously constructed model and a limited amount of ***p*** recent tuples

$$h_i = train\left(h_{i-1}, (x_i, y_i), \ldots, (x_{i-p+1}, y_{i-p+1})\right)$$

Only a last model ***hj*** is applied to the test set to determine the offline accuracy ξ

$$\xi(D_t) = \frac{1}{k}\sum_{i=1}^{k} 1 - \mathcal{L}(\hat{y}_i, y_i) = \frac{1}{k}\sum_{i=1}^{k} 1 - \mathcal{L}\big(h_j(x_i), y_i\big)$$

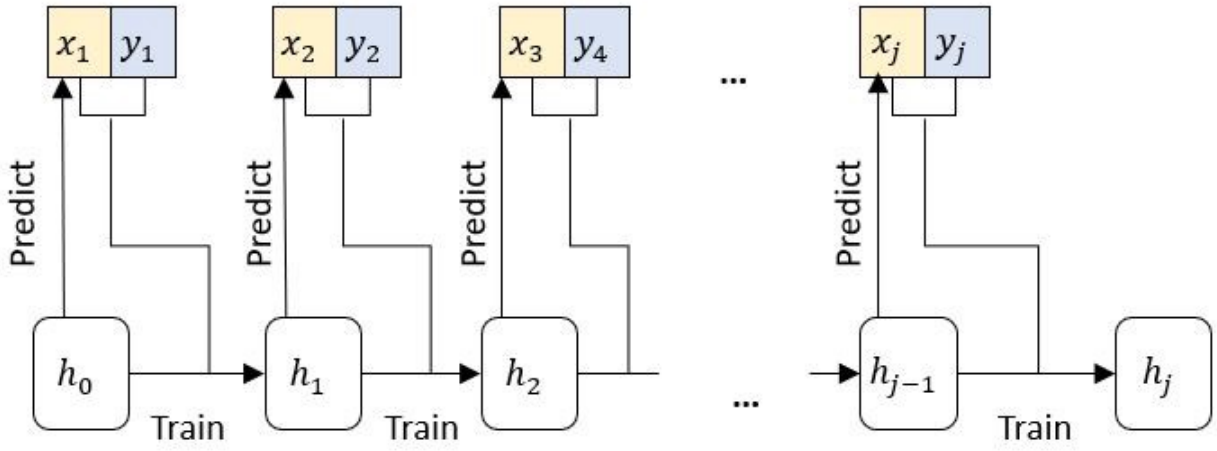

Figure 3: The online learning scheme. Data are not split into training and testing sets. Instead, each model subsequently predicts one example, which is later used for the construction of the next model.

Hence, this setting allows only inference about the generalization ability of the last model and neglects all preceding models. Such an evaluation is useful in data streams scenarios, for example, where a lot of training data are available to continuously construct as accurate a model as possible.

Online Setting

Data stream classification is usually evaluated in the online setting, which is depicted in Figure 3. A potentially infinite sequence

$$S = (s_1, s_2, \dots, s_t, \dots)$$

of tuples

$$s_i = (x_i, y_i)$$

arrives one after another. As ***t*** represents the current timestamp, the learning objective is to predict the corresponding label

y_t

for a given input

x_t

which is supposed to be unknown. The prediction

$$\hat{y}_t = h_{t-1}(x_t)$$

is done according to the previously learned model

h_{t-1}

Afterward, the true label is revealed and the loss

$\mathcal{L}(\hat{y}_t, y_t)$

determined. The online accuracy for a sequence up to the current time ***t*** is given by

$$E(S) = \frac{1}{t}\sum_{i=1}^{t} 1 - \mathcal{L}(h_{i-1}(x_i), y_i)$$

The main difference to the previous setting is that all intermediate models are considered for the performance evaluation, but each of them predicts only the following example. Additionally, the data for training and testing are not strictly disjunct, but instead, each instance is initially used for model testing and then for the adaption.

Regarding non-stationary data, a high online accuracy does not necessarily imply a high generalization ability of the models. For instance, in case of a strong auto-correlation of the labels, an algorithm simply predicting the previous label achieves accurate results without learning any data structure. However, for i.i.d. data the online accuracy of an incremental algorithm is typically correlated with the average generalization ability of all constructed models. The online accuracy is a reasonable evaluation measure for tasks requiring an immediate prediction even after a few training examples.

The combination of both accuracies, both offline and online, enables conclusions about the learning curve: In the case of two different models *A*, ***B*** having the same offline accuracy, but ***A*** having a higher online accuracy implies that ***A*** converges on average faster than ***B*** and vice versa.

This article provided a general overview of the most prevalent method of online incremental learning. Additional articles will have more focus on specifically chosen algorithms.

Learn more about NICE Actimize Financial Crime Analytics research here.

ABOUT NICE ACTIMIZE

NICE Actimize is the largest and broadest provider of financial crime, risk and compliance solutions for regional and global financial institutions, as well as government regulators. Consistently ranked as number one in the space, NICE Actimize experts apply innovative technology to protect institutions and safeguard consumers and investors assets by identifying financial crime, preventing fraud and providing regulatory compliance. The company provides real-time, cross-channel fraud prevention, anti-money laundering detection, and trading surveillance solutions that address such concerns as payment fraud, cybercrime, sanctions monitoring, market abuse, customer due diligence and insider trading.

info@niceactimize.com | www.niceactimize.com/blog | @nice_actimize | /company/actimize | NICEactimize

NICE · ACTIMIZE