Towards Efficient Machine Unlearning via Incremental View Maintenance

Sebastian Schelter [1]

Abstract

Recent laws such as the GDPR require machine learning applications to *unlearn* parts of their training data if a user withdraws consent for their data. Current unlearning approaches accelerate the retraining of models, but come with hidden costs due to the need to reaccess training data and redeploy the resulting models.

We propose to look at machine unlearning as an *incremental view maintenance* problem, leveraging existing research from the data management community on efficiently maintaining the results of a query in response to changes in its inputs. Our core idea is to consider ML models as views over their training data, and express the training procedure as a differential dataflow computation, whose outputs can be automatically updated. As a consequence, the resulting models can be continuously trained over streams of updates and deletions. We discuss important limitations of this approach, and provide preliminary experimental results for maintaining a state-of-the-art sequential recommendation model.

1. Introduction

Enacting the right to be forgotten. Software systems that learn from user data with machine learning (ML) have become ubiquitous over the last years, and participate in many critical decision-making processes, e.g., about loans, job applications and medical treatments (Stoyanovich et al., 2020). Recent laws such as the "right to be forgotten" (GDPR.eu, a) (Article 17 of the General Data Protection Regulation (GDPR)) require companies and institutions that process personal data to delete user data upon request: *"The data subject shall have the right to [...] the erasure of personal data concerning him or her without undue delay [...] where the data subject withdraws consent"*.

[1]University of Amsterdam. Correspondence to: Sebastian Schelter <s.schelter@uva.nl>.

Workshop on Challenges in Deploying and Monitoring ML Systems at the International Conference on Machine Learning 2021.

Recent research (Schelter, 2020; Cao & Yang, 2015; Ginart et al., 2019; Wu et al., 2020) argues that it is not sufficient to merely delete personal user data from primary data stores such as databases, and that machine learning models that have been trained on the stored data also fall under the regulation.

The hidden costs of machine unlearning. There exists no automated deletion mechanism for ML models derived from the data. The machine learning community has been working on this issue under the umbrella of *machine unlearning* (Cauwenberghs & Poggio, 2001; Karasuyama & Takeuchi, 2009; Cao & Yang, 2015; Ginart et al., 2019; Neel et al., 2020). Given a model, its training data and a set of user data to unlearn, they propose efficient ways to accelerate the retraining of the model.

However, these approaches ignore the constraints imposed by the complexity of deployment pipelines in real-world ML applications (Olston et al., 2017; Sculley et al., 2015; Schelter et al., 2018). ML models are deployed in serving systems that can efficiently answer online prediction requests with low latency, but in order to update a model, heavy-weight pipelines that have to spin up infrastructure, retrain the model and run expensive evaluation workloads have to be executed (Andrews et al., 2016; Olston et al., 2017; Böse et al., 2017). Data removal in response to GDPR requests is typically integrated into these pipelines, and will only affect the model once it is being redeployed via the heavyweight pipeline. Going through this whole process only to unlearn a single training example does not make sense economically and operationally.

Low-latency machine unlearning. The GDPR law does not specify how soon data must be erased after a deletion request (Shastri et al., 2020), yet it states the *"obligation to erase personal data without undue delay"* (GDPR.eu, a) using *"appropriate and effective measures"* (GDPR.eu, b). Currently, data erasure seems to be a rather tedious and lengthy process in practice; data erasure from active systems in the Google cloud, for example, can take up to two months (Google, 2021). We argue that it is an open and important academic question to determine how fast data can be erased from deployed ML models, and that we should design systems that empower users to exercise their right to be forgotten as timely as possible.

In this paper, we propose to look at machine unlearning as an *incremental view maintenance* (IVM) problem, repurposing techniques from data management research (Blakeley et al., 1986). In IVM, we efficiently maintain the materialised result of a query in response to changes in the query inputs. Our core idea is to consider ML models as views over their training data, and express the training procedure as a differential dataflow computation, which can be automatically updated (Section 2). *As a consequence, the resulting models can be continuously trained over streams of updates and deletions.* In summary, we provide the following contributions:

- We propose to consider ML models as materialised views over their training data, which support efficient unlearning via view maintenance techniques (Section 2).
- We discuss important limitations of our approach with respect to gradient-based learning (Section 3).
- We show how to efficiently maintain a state-of-the-art sequential recommendation model over a dataset of clicks from an ecommerce platform (Section 4).

2. Machine Unlearning as View Maintenance Problem

Core idea. We explore how to leverage existing view maintenance techniques from relational query processing to efficiently remove data from certain classes of machine learning models. Incremental View Maintenance (IVM) (Blakeley et al., 1986) allows us to materialize the results of a relational query on a database and efficiently update the query result in response to changes in the underlying database. In order to apply these techniques to ML models and pipelines, we need to map the training procedures to relational (or general dataflow) operations on sets, for which efficient maintenance techniques are known.

Toy example. In the following, we discuss our ideas on the toy example of maintaining a Naive Bayes classifier over an email database. The input data for the classifier comprises of a matrix $\mathbf{X} \in \{0,1\}^{m \times d}$ of m d-dimensional observations (the occurrences of terms in emails), and the binary target variable $\mathbf{y} \in \{0,1\}^m$ which denotes whether a mail is considered spam or not. With a uniform prior estimate, the MNB classifier assigns its class prediction $\hat{y}$ as: $\hat{y} = \text{argmax}_y \sum_j \log x_j \theta_{yj}$, where the probability $P(x_j = 1|y) = \theta_{yj}$ of encountering feature j in class y is estimated using a smoothed version of the maximum likelihood estimate $\hat{\theta}_{yj} = \frac{N_{yj}+1}{N_y+d}$, where N_y denotes the number of samples for class y and the alphas. The main computational challenge in training the NB model is the determination of the counts N_{yj}, which denote the number of times the value for feature j occurs in class y.

Incremental view maintenance for model training with relational operations. Let us assume that our database contains two relations: the relation `email(mid, term)` models the occurrences of terms in a mail (identified by `mid`), while the relation `labeled(mid,spam,verified)` denotes which emails have already been labeled as `spam` or not, and whether the labeling has been `verified`. Training the Naive Bayes model on this database requires us to join the `email` and `labeled` relation (which we filter for verified entries) and to count resulting occurrences of terms per label. The training can be expressed in SQL as follows:

```
SELECT email.term, labeled.spam, COUNT(*)
FROM email JOIN labeled ON mid
WHERE labeled.verified
GROUP BY email.term, labeled.spam
```

Let us now discuss how to maintain the resulting counts in light of additions and deletions to our source relations. We switch to an algebraic view of the query for this. Relational query processors operate on sets of tuples, based on the relational algebra, which defines the semantics of relations queries over sets. Our example query can be expressed as $\gamma_{count(\text{term,is_spam})}(E \bowtie_{\text{mid}} (\sigma_{\text{verified}}(L)))$, denoting a selection $\sigma_{\text{verified}}(L)$ over the labeled entries L whose results are joined with the emails E, grouped by term and label type and counted to estimate the likelihood $P(x_j = 1|y_c)$ for all terms (categorical features) x_j and labels y_c.

The goal of incremental view maintenance is to efficiently update the results of this query in response to changes in the inputs of the source relations E and L. Let δ_E and δ_L denote sets of updates (additions or deletions) to the the source relations. The algebraic nature of the relational operations allows for the derivation of efficient update rules (Blakeley et al., 1986). Selection for example is a linear operation, which means that the update δ_L can be processed independently of the previous operator outputs $\sigma_{\text{verified}}(L \cup \delta_L) = \sigma_{\text{verified}}(L) \cup \sigma_{verified}(\delta_L)$. Selections therefore do not need to maintain state and can be pipelined with other operations. The equi-join operation distributes over the deltas to the source relations $(E \cup \delta_E) \bowtie_{\text{mid}} (L \cup \delta_L) = (E \bowtie_{\text{mid}} L) \cup (E \bowtie_{\text{mid}} \delta_L) \cup (\delta_E \bowtie_{\text{mid}} L) \cup (\delta_E \bowtie_{\text{mid}} \delta_L)$. This means that a query processor can compute the updates to the previous join result $E \bowtie_{\text{mid}} L$ by joining the deltas with the previous versions of the relations ($E \bowtie_{\text{mid}} \delta_L$ and $\delta_E \bowtie_{\text{mid}} L$) as well as with themselves $\delta_E \bowtie_{\text{mid}} \delta_L$. These updates can be efficiently computed by maintaining indexed versions of the input relations. If these indexes allow random access in constant time, the asymptotic complexity of updating the join result is linear in the size of the updates, e.g., $O(|\delta_E| + |\delta_L|)$ for our example. The final count aggregation $\gamma_{count(\text{term,spam})}$ only requires us to update the counts per group (formed by term and label) based on the changes in the join result. This

can again be done with linear complexity in the size of the updates given efficient random access to the group counts.

Machine unlearning as view maintenance over dataflow computations. Implementation-wise, we propose to leverage differential computation (McSherry et al., 2013) in *Differential Dataflow* (McSherry, 2021), which generalises incremental view maintenance to dataflow computations over multisets with user-defined functions and recursion. Differential dataflow supports common dataflow operations like map, reduce, join, and filter, automatically parallelises the computation across multiple cores and machines, and consistently updates the results in response to additions and deletions of the inputs. Dataflow computations also allow us to easily add more complex preprocessing functions here, we could for example tokenise complete emails in an `explode` operation, instead of having to store all individual email-term pairs. A (simplified) Rust implementation of our NB training in Differential Dataflow looks as follows:

```
let samples = labeled
  .filter(|l| l.verified)
  .map(|l| (l.mid, l.spam))

let counts_per_feature_and_class = emails
  .explode(|e| { tokenize(e.text).into_iter()
                   .map(|term| (e.mid, term)) })
  .join_map(&samples, |_, term, spam| (term, spam))
  .count()
```

Differential Dataflow now allows us to update both input relations and compute the corresponding changes to the outputs.We could for example delete an existing email via `emails.update(Email { mid: 123, text: "hello world" }, -1)`, which would trigger this email to be removed from the join result and lead to decrements for all the counts related to terms contained in the email and the corresponding class label.

3. Limitations

Our approach is based on the idea of recomputing the parts of the model that have been affected by the user data which we aim to remove. This is efficient if (i) only a small part of the model is affected, and (ii) the recomputation is asymptotically cheaper than full retraining. A general property that a model must have for our approach to work are so-called *sparse computational dependencies* (Bertsekas & Tsitsiklis, 1989; Ewen et al., 2012; Schelter et al., 2013). A formal way to reason about these is via a bipartite dependency graph $G(V_i, V_m, E)$ (Bertsekas & Tsitsiklis, 1989) where the vertex set V_i denotes partitions of our input data, the vertex set V_m denotes partitions of the final model, and an edge $(v_i, v_m) \in E$ between a vertex v_i and a vertex v_m denotes that the input i is necessary to compute the m-th part of the model. This graph allows us to determine which parts of the model need to be recomputed if we remove a particular input v_u. If the graph is sparse (e.g., the vertices in V_i have a low degree), only a small part of the model must be recomputed. If the graph was fully connected, the removal of a single input would trigger a complete retraining of the model.

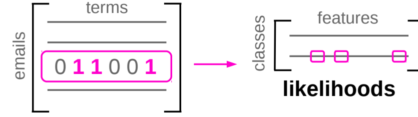

Figure 1. Illustration of the sparse computational dependencies in Naive Bayes. Each (sparse) row of the input affects only a fraction of the model indicated by the non-zero entries in the row.

The dependency graph between the inputs (binary term occurrences in emails) and the model (likelihoods) in our Naive Bayes example is sparse as well as shown in Figure 1. Each estimate of a likelihood $P(x_j = 1|y_c)$ only depends on emails of class y_c which contain the term x_j. That means the number of updates to the model in response to a removal or addition of a sample is linear in the number of non-zero features of this sample.

Non-applicability to gradient-based learning. The dependency graph model also explains why our approach is not efficiently applicable to models learnt with gradient descent, as this learning algorithm exhibits *dense computational dependencies*. In general, we learn a supervised model $\mathbf{w}$ on labeled examples $\{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_n, y_n)\}$ as follows using gradient descent: $\mathbf{w}^{(t+1)} = \mathbf{w}^{(t)} - \lambda \sum_{\mathbf{x},y} \nabla_{\mathbf{w}^{(t)}} \ell(\mathbf{x}, y, \mathbf{w}^{(t)})$. We obtain $\mathbf{w}^{(t+1)}$ by updating $\mathbf{w}^{(t)}$ (according to the learning rate λ) in the negative direction of the gradient of the model's loss function, which we compute as the sum of the gradients over the losses ℓ of the individual examples $(\mathbf{x}, y)$. That means that each intermediate model $\mathbf{w}^{(t)}$ is dependent on all inputs with a non-zero gradient and recursively depends on all previous model versions $\mathbf{w}^{(t-1)}, \ldots, \mathbf{w}^{(0)}$, giving rise to a dense computational dependency graph, as sketched in Figure 2.

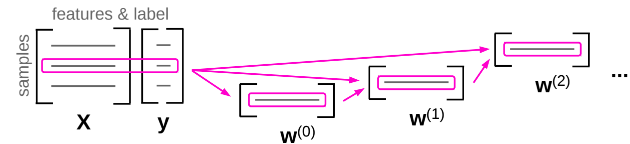

Figure 2. Illustration of the dense computational dependencies in stochastic gradient descent-based learning. Each model iterate $\mathbf{w}^{(t)}$ depends on the whole input and the previous iterate $\mathbf{w}^{(t-1)}$.

This illustrates that our approach works best in case of learning procedures with sparse computational dependencies and no global aggregations. Such approaches include popular and widely used techniques such as nearest neighbor-based methods and tree-based learning, which are widely used in industry (Amazon, 2021).

4. Example: Sequential Recommendation with Vector-Session-kNN

Vector-Session-Knn. Given an evolving session (a sequence of interactions from a set of items $\mathbf{I}$), the goal of session-based recommendation is to accurately predict the next item that the user will interact with. Vector-Session-KNN (Ludewig et al., 2019) (VS-kNN) is a state-of-the-art nearest-neighbor based approach to session-based recommendation, which outperforms current neural approaches for this task.

In VS-kNN, we have a set of historical sessions $\mathbf{H} \in \{0,1\}^{|\mathbf{I}|}$ represented as binary vectors in item space, and an evolving session $\mathbf{s}^{(t)} \in \mathbb{N}^{|\mathbf{I}|}$ at time t, where the values of the non-zero entries denote the insertion order of items. Algorithm 1 describes how VS-kNN computes its recommendations for an evolving session $\mathbf{s}^{(t)}$. First a recency-based sample $\overline{\mathbf{H}}_s$ of size m is taken from all historical sessions $\mathbf{H}_s$ that share at least one item with the evolving sessions. Next, we compute the k closest sessions $\mathbf{N}_s$ from $\overline{\mathbf{H}}_s$ according to the dot product $\pi(\mathbf{s}^{(t)})^\top \mathbf{h}$, which applies an element-wise, non-linear decay function π to the entries denoting the insertion order in the evolving session. All items occurring in these neighboring sessions are finally scored by adding up their similarities (the previously computed decayed dot product) weighted by another non-linear function λ applied to the position $\max(\mathbf{s}^{(t)} \odot \mathbf{n})$ of the most recent shared item between the evolving session $\mathbf{s}^{(t)}$ and the neighbor session $\mathbf{n}$.

Algorithm 1 Vector-Session-kNN.

function VSKNN($\mathbf{s}, \mathbf{H}, \pi, \lambda, m, k$)
- $\mathbf{H}_s \leftarrow$ historical sessions that share at least one item with $\mathbf{s}$
- $\overline{\mathbf{H}}_s \leftarrow$ recency-based sample of size m from $\mathbf{H}_s$
- $\mathbf{N}_s \leftarrow k$ closest sessions $\mathbf{h} \in \overline{\mathbf{H}}_s$ according to $\pi(\mathbf{s}^{(t)})^\top \mathbf{h}$
- **for** item $i \in \mathbf{N}_s$ **do**
  - $\mathbf{r}_i \leftarrow \sum_{\mathbf{n} \in \mathbf{N}_s} \mathbb{1}_n(i) \cdot \pi(\mathbf{s}^{(t)})^\top \mathbf{n} \cdot \lambda(\max(\mathbf{s}^{(t)} \odot \mathbf{n}))$
- **return** item scores $\mathbf{r}$

Vector-Session-Knn in Differential Dataflow. The algorithm can be expressed as a series of joins and aggregations in differential dataflow. A major challenge here is that differential computations are modeled on multisets without order, while VS-kNN requires sequences as input. We solve this issue by expressing the insertion order of items in an evolving session via the cardinalities of items in the multiset, e.g., the item at position three in the sequence has a cardinality of three in our multiset. The decayed dot product $\pi(\mathbf{s}^{(t)})^\top \mathbf{h}$ between a session $\mathbf{s}^{(t)}$ and a historical session $\mathbf{h}$ can thus be modeled by an aggregation over a semi-join between the items in the sessions.

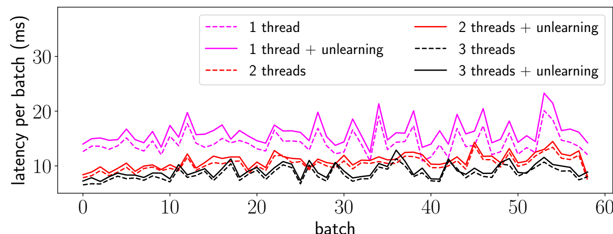

Figure 3. Mean latencies per batch of 100 sessions for a state-of-the-art sequential recommendation model on a dataset of more than 1M clicks from an ecommerce platform. Concurrent unlearning requests only cause minimal overhead for model maintenance.

Preliminary experimental results. We conduct a preliminary evaluation of our differential VS-kNN implementation on a proprietary dataset of 200K real world sessions with 1.1M clicks from an industry partner of ours (a large European ecommerce platform), which we used in a recent study on real world recommender systems (Kersbergen & Schelter, 2021). We compute session-based recommendations for 6K sessions with 30K clicks from the subsequent day. We process batches of 100 evolving sessions from the test set at a time with m=100 and k=20, and predict the next items for a session length from 1 to 10 for all the evolving sessions in a batch. We measure the mean prediction latency in milliseconds for each batch, with a growing number of threads. We compare the execution of the workload without unlearning to a second run with unlearning operations, where we delete 10 historical sessions from the training set concurrently with the recommendation computation for each batch.

Figure 3 illustrates the resulting mean latencies per batch for the workload without unlearning (the dotted lines). We find that a batch of 100 sessions can be processed in about 15 milliseconds with a single thread, and increasing the number of threads reduces this time to less than 10 milliseconds. We see that the additional unlearning operations only marginally increase the recommendation latency per batch, typically around two milliseconds for the single threaded implementation, and only around one millisecond for the multithreaded execution.

5. Next Steps

The preliminary experimental results confirm our proposed approach of modelling the training as differential dataflow to incorporate low-latency unlearning operations (deletions) during the maintenance of the recommendation system. To the best of our knowledge, there exist no other recommendation systems which can incorporate such low-latency unlearning operations. We are currently exploring for a variety of nearest-neighbor and tree-based ML models whether they can also be efficiently modeled as differential computations.

References

Amazon. Algorithms in amazon sagemaker. https://docs.aws.amazon.com/sagemaker/latest/dg/algos.html, 2021.

Andrews, P., Kalro, A., Mehanna, H., and Sidorov, A. Productionizing machine learning pipelines at scale. *ML Systems workshop at ICML*, 2016.

Bertsekas, D. P. and Tsitsiklis, J. N. *Parallel and distributed computation: numerical methods*, volume 23. Prentice hall Englewood Cliffs, NJ, 1989.

Blakeley, J. A., Larson, P.-A., and Tompa, F. W. Efficiently updating materialized views. *ACM SIGMOD Record*, 15 (2):61–71, 1986.

Böse, J.-H., Flunkert, V., Gasthaus, J., Januschowski, T., Lange, D., Salinas, D., Schelter, S., Seeger, M., and Wang, Y. Probabilistic demand forecasting at scale. *VLDB*, 10 (12):1694–1705, 2017.

Cao, Y. and Yang, J. Towards making systems forget with machine unlearning. *IEEE Symposium on Security and Privacy*, pp. 463–480, 2015.

Cauwenberghs, G. and Poggio, T. Incremental and decremental support vector machine learning. *NeurIPS*, pp. 409–415, 2001.

Ewen, S., Tzoumas, K., Kaufmann, M., and Markl, V. Spinning fast iterative data flows. *PVLDB*, 5(11):1268–1279, 2012.

GDPR.eu. Article 17: Right to be forgotten. https://gdpr.eu/article-17-right-to-be-forgotten, a.

GDPR.eu. Recital 74: Responsibility and liability of the controller. https://gdpr.eu/recital-74-responsibility-and-liability-of-the-controller/, b.

Ginart, A., Guan, M. Y., Valiant, G., and Zou, J. Making AI forget you: Data deletion in machine learning. *NeurIPS*, 2019.

Google. Deletion in the google cloud. https://cloud.google.com/security/deletion, 2021.

Karasuyama, M. and Takeuchi, I. Multiple incremental decremental learning of support vector machines. *NeurIPS*, pp. 907–915, 2009.

Kersbergen, B. and Schelter, S. Learnings from a retail recommendation system on billions of interactions at bol.com. *ICDE*, 2021.

Ludewig, M., Mauro, N., Latifi, S., and Jannach, D. Performance comparison of neural and non-neural approaches to session-based recommendation. In *Proceedings of the 13th ACM Conference on Recommender Systems*, pp. 462–466, 2019.

McSherry, F. Differential dataflow. https://github.com/TimelyDataflow/differential-dataflow, 2021.

McSherry, F., Murray, D. G., Isaacs, R., and Isard, M. Differential dataflow. *CIDR*, 2013.

Neel, S., Roth, A., and Sharifi-Malvajerdi, S. Descent-to-delete: Gradient-based methods for machine unlearning, 2020.

Olston, C., Fiedel, N., Gorovoy, K., Harmsen, J., Lao, L., Li, F., Rajashekhar, V., Ramesh, S., and Soyke, J. Tensorflow-serving: Flexible, high-performance ml serving. *ML Systems workshop at NeurIPS*, 2017.

Schelter, S. "amnesia"–a selection of machine learning models that can forget user data very fast. *CIDR*, 2020.

Schelter, S., Ewen, S., Tzoumas, K., and Markl, V. All roads lead to rome: optimistic recovery for distributed iterative data processing. *CIKM*, pp. 1919–1928, 2013.

Schelter, S., Biessmann, F., Januschowski, T., Salinas, D., Seufert, S., and Szarvas, G. On challenges in machine learning model management. *IEEE Data Engineering Bulletin*, 2018.

Sculley, D., Holt, G., Golovin, D., Davydov, E., Phillips, T., Ebner, D., Chaudhary, V., Young, M., Crespo, J.-F., and Dennison, D. Hidden technical debt in machine learning systems. In *NeurIPS*, pp. 2503–2511, 2015.

Shastri, S., Banakar, V., Wasserman, M., Kumar, A., and Chidambaram, V. Understanding and benchmarking the impact of gdpr on database systems. *PVLDB*, 2020.

Stoyanovich, J., Howe, B., and Jagadish, H. Responsible data management. *VLDB*, 13(12):3474–3489, 2020.

Wu, Y., Dobriban, E., and Davidson, S. B. Deltagrad: Rapid retraining of machine learning models, 2020.