COMP 642 — Machine Learning Mar 29, 2022

Lecture 20: Machine Learning at Scale

Lecturer: Anshumali Shrivastava
Scribe By: Angela Cao, Steven Cooper, Savinay Kariyappa, Tanmay Mathur

Disclaimer: *These lecture notes are intended to develop the thought process and intuition in machine learning. The materials are not thoroughly reviewed and can contain errors.*

1 Motivation for Machine Learning at Scale

So far in machine learning, we have explored the different types of building blocks that make up a machine learning problem: The Data, The Function, and the Loss Function.

$$D, F, L \tag{1}$$

But in certain applications, before we can even begin to consider an approach to the latter two, we must consider the **scale** of the former, the data.

Consider for example the challenge posed by the Netflix Prize, that of predicting user ratings for movies, based only on a user's prior ratings of different films. At first glance, this might seem simple, but with millions of users, each of whom has watched and rated hundreds of movies, the data set quickly swells to over 100,000,000 data points. How can machine learning handle the data at this scale?

This question is perhaps best approached by asking the follow-up questions, "What resources do we have?" and "How much time is there to train the model?" But even at the largest tech companies running their algorithms on the most advanced machines the world has to offer, a different approach is needed to begin to tackle such problems. Moving from data sets of a thousand or a million to those of a billion or a trillion is like moving between different worlds, and one can NOT do the same thing for data sets of these size. It is hard to compare potential models and loss functions when the data set is huge and we need a way to tune models and validate hypotheses when working in this new world.

2 Random Subsampling

One straightforward approach is to reduce the scope of the data set to a more manageable size (e.g. something that can run in a few hours) through random subsampling of the original dataset. We can then tune and train on the smaller sample until we have a better model that can be scaled up to the whole dataset once ready.

If the original dataset, D, is of size N, draw k random samples to obtain D' where k is a number as big as possible but within the computational budget of our resources.

$$D = \{x_i, y_i\}_{i=1}^{N} \longrightarrow D' = \{x_i, y_i\}_{i=1}^{k} \qquad \text{where} k << N \tag{2}$$

Some disadvantages of this approach are that if you have a class imbalance you might miss data in the rare tails of a dataset's distribution. For certain problems, outliers can be critical, so random sampling might not always be applicable. Unfortunately we also don't always know ahead of time if the model

trained on the smaller set of data will scale when eventually run on the larger dataset.

The hope, however, is that you will learn over time and that the 0th order model created from the subsample is the best start to build upon in subsequent efforts. Development is a constant iterative process and there is no silver bullet or even a final end goal of a "best" algorithm. Development teams are simply continually looking for a better algorithm than the current baseline. Often one team will be at work tuning a production model while other teams are trying to beat its performance in the hopes of one day replacing it.

2.1 Monte Carlo Cross-Validation

In Random Subsampling, known as Monte Carlo cross-validation, the whole dataset is split into a chosen k number of subsets and in each subset, a fixed number of observations is chosen without replacement from the sample and kept aside as test data. With the training set, the model then gets fit while with the test data, a prediction error is obtained from it. [1]

To get the true error estimate obtained from Random Subsampling for the dataset, we get the average of all obtained prediction errors. Suppose E_i is the estimated prediction error for each i^{th} subset. Then the true error estimate, TE can be calculated as [4]

$$TE = \frac{1}{K} \sum_{i=1}^{K} E_i \tag{3}$$

One of the advantages of Random Subsampling is that the size of the train and test sets are independent from k. However, several disadvantages of Random Subsampling are that (1) some samples from the dataset may not be chosen for training or testing, and (2) the process is unsuitable for an imbalanced dataset as we're assuming that the dataset is balanced when we're applying Random Subsampling. [3]

3 Dimensionality Reduction

Another issue caused by large datasets is both the sheer number of possible features. Returning to the Netflix problem, they have 200 million users and thousands of movies meaning there are over a trillion possible combinations. One hot encoding and even embedding this amount of features would result in a VERY big model and just the tokens needed to describe the metadata on a corpus of a hundred thousand words is a lot. Furthermore, one hot encoding on such a large number of features will lead to a very sparse matrix, providing additional challenges. Can we take hundreds of millions of features and reduce them to say a thousand, but still keep a reasonable ability to discriminate between inputs when using the smaller feature set?

3.1 One Hot Encoding

The typical representation of features in a document classification task is known as the "Bag of Words Model". In this model, the distinct token from the documents is extracted to represent the features of the documents in both training and testing datasets.

Machine learning algorithms are programmed to use vectors or tensors and the features extracted above are represented in what is known as the term-document incidence matrix. The matrix is a "one-hot encoding" representation of the features where the frequency or the incidence information of each token

is represented. [2]

Consider the following Example of three documents:
• Lewis likes movies.
• Sebastian likes to make jokes.
• Max likes jokes and movies.

The term document matrix will be as follows:

Lewis	likes	movies	Sebastian	to	make	jokes	Max	and
1	1	1	0	0	0	0	0	0
0	1	0	1	1	1	1	0	0
0	1	1	0	0	0	1	1	1

However, for extremely big datasets this incidence matrix can quickly become onerously large and the data will be inherently sparse as the average row will only contain a few entries for columns and vice versa. Therefore it would be desirable to reduce the number of features in scope before applying machine learning algorithms.

3.2 Principal Component Analysis

One approach to reducing dimensionality to obtain a smaller feature set would be to perform Principal Component Analysis (PCA) in the hopes of getting a much smaller amount of new uncorrelated variables that preserves the original data's variation. But because the dataset is so large, the resources to perform PCA itself is so computationally and memory expensive as to be a non-starter. i.e. it would be more cost effective to simply run machine learning. Although PCA is the mathematically optimal dimensionality reduction, assuming linear transformations, it is simply too costly to calculate and train end-to-end for billion scale data. Again, a billion is a different world from one thousand and we don't have the budget to learn the "best" reduction.

Random sampling of the features themselves, or only taking the most frequently seen features are other possible approaches, but both face the same demerit of excluding potentially useful information that is then lost to the model forever.

3.3 Feature Hashing

The most popular approach to handle these issues is through a process known as feature hashing. In feature hashing, a large number, e.g. a thousand, features from the larger feature set are combined into one element through randomly matching the original features to new features in the new smaller vector. Their information is combined with the other features that are mapped to the same output node. Since the mapping is random, the result is still reasonable though some information loss will always be inevitable as there is "no free lunch" in dimensionality reduction.

$$\frac{1}{1k} x_i^T x_j \approx \frac{1}{1M} X_i^T X_j \tag{4}$$

As seen above, this approach still preserves pairwise geometry and doesn't require the computationally expensive operation of computing a thousand eigenvectors required by PCA. If the original vector is dense, there will be a lot of overlap which can hurt performance but if the vector is sparse (as is often the case when dealing with extremely large datasets) feature hashing performs quite well.

3.4 Hashing

Hashing involves using a simple equation that only requires two fixed integers (e.g. a and b) but will map any input into a set number of outputs. For a 1000, the hashing equation could be written as:

$$h(x) = (ax + b) \mod 1000 \tag{5}$$

With sparse input vectors there is a very low likelihood that sparse data will collide and change the result leading to a very high probability that the inner product will match, and thus the geometry is preserved.

Feature hashing also has the added benefit of helping to preserve privacy since the individual user information is combined together before being fed to the model.

Returning to our Netflix example, a large majority of people will always like the popular titles (e.g. Shawshank Redemption) but there will still be enough heterogeneity in the combined vector to be useful in making individualized predictions for different users.

3.5 Hybrid Approach

A hybrid approach involves starting with and preserving the most important and frequently observed features (e.g. 1,000) and then feature hashing the rest (e.g. 1,001 - 1,000,000). This ensures that the information on the very frequent and likely critical features is preserved, while also not ignoring the information contained in the less common features.

3.6 Data Processing and Data Cleaning

Another useful way to reduce the size of the data and thus the complexity of the machine learning to follow, is to perform data processing and data cleaning on the dataset first. This is not a trivial step and for practitioners data process, data cleaning and data management can be where the majority of their efforts are dedicated.

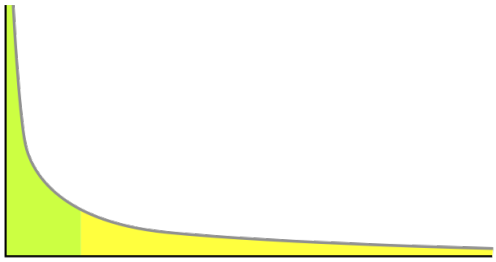
In the corpus of webpages for example, while there are trillions of pages they can be practically reduced to a much smaller million by removing duplicates and near duplicates. This is not always an easy task and to reiterate a lot of work happens before you get to the machine learning step (sometimes over 90%).

4 Caching

One final trick to handling large datasets is **caching**, i.e., simply memorizing the prior searches and most common results.

4.1 Caching and the Power Law

The real world is much more ordered than the random possibilities in mathematics and often follows a power law probability distribution (often referred to by the Pareto principle or ”80-20 rule” stating that 80% of outcomes are due to 20% of causes). In reality for many phenomena, a very few things happen very frequently and a great many things only happen rarely. This can be seen in people’s favorite movies, website traffic, common words etc. and has been observed again and again in real-world data. Out of millions of movies, there are only a few thousand that are frequently seen by the audience. Similarly, out of a billion websites, there are only a few hundred daily visited by people.



Example Power Law Distribution [5]

As illustrated in the above plot the green shaded region represent a small fraction of the individual features but a large number of the contributions with the yellow shaded region representing the long tail. We can leverage the knowledge of power law distributions in our machine learning algorithms, since if we just remember the head of the data we are most of the way there.

To do so, simply save common queries and memorize the most popular results (i.e. caching). Treat the most important results differently than the rest to capture the ”head” of the distributions and stop once you reach the point of diminishing returns for subsequent resources. More complex algorithms can then be used on the remaining dataset to capture the information in the tail.

Caching is a popular approach and can often be observed when using many websites. Unusual queries can take a while to run the first time, but the second time when they are repeated they run in an instant since the previous results have been cached. When you have queried the algorithm a second time the algorithm realizes this is no longer a rare event and the algorithm simply returns the cached result.

4.2 Caching in Action

Let’s take an example of an Amazon Product search, “Nintendo Switch” accounted for 1.26 million monthly searches in search volume last year. Now, rather than running a neural network every time someone asks for a Nintendo Switch Amazon caches the result in order to save unnecessary computes. Similarly, Google caches the search queries with each search. It is evident that when you are searching for trending topic results appear much faster compared to when you are searching for a non-trivial topic.

Therefore, it makes sense for the engineers to handle frequently asked queries and non-frequently asked queries differently. They can cache the results for frequent topics and run the model for non-frequent ones. Caching the head of the distribution will enable frequently asked queries to be answered efficiently. By doing this exercise we are compressing our use case, enabling us to reach 90 percent of our users quickly. So, it is very important to remember: “if it is required again, better cache it”.

5 Summary and Application

Sampling, feature hashing and caching represent the trinity of big data management approaches for machine learning and their combined application can bring problems from the world of impossibility to a manageable size. This is necessary because there are many applications with massive numbers of features and huge sets of data.

Often the data generation rate is so fast that programs have to run even faster to keep up (e.g. thousands of tweets per second). For problems that must handle this data in real time (e.g. hate speech detection on tweets) systems therefore are designed to work in a hierarchy, quickly filtering the vast majority of the data and only apply more complex algorithms and processes to the 1% of the data that fails to pass the first argument.

6 Coming Soon

In our next class, we will learn that without parrallelism, machine learning isn't going anywhere!

References

[1] Hajare Akash. Hold out method random sub-sampling method. https://blog.ineuron.ai/Hold-Out-Method-Random-Sub-Sampling-Method-3MLDEXAZML.

[2] Wikipedia contributors. "feature hashing — Wikipedia, the free encyclopedia", "2021". "[Online; accessed 4-April-2022]".

[3] Satyam Kumar. Understanding 8 types of cross-validation. https://towardsdatascience.com/understanding-8-types-of-cross-validation-80c935a4976d.

[4] Penn State: Eberly College of Science. 2.2 - cross-validation. https://online.stat.psu.edu/stat508/lesson/2/2.2.

[5] Hay Kranen PD. The long tail. 2006.