Performance and Capacity Considerations

This lesson provides a quick introduction to performance and capacity considerations and discusses why they matter when designing a solution to a machine learning problem.

We'll cover the following

As we work on a machine learning-based system, our goal is generally to improve our metrics (engagement rate, etc.) while ensuring that we meet the capacity and performance requirements.

Major performance and capacity discussions come in during the following two phases of building a machine learning system:

1. *Training time*: How much training data and capacity is needed to build our predictor?
2. *Evaluation time*: What are the Service level agreement(SLA) that we have to meet while serving the model and capacity needs?

We need to consider the performance and capacity along with optimization for the ML task at hand, i.e., measure the complexity of the ML system at th training and evaluation time and use it in the decision process of building our ML system architecture as well as in the selection of the ML modeling technique.

Complexities consideration for an ML system#

Machine learning algorithms have three different types of complexities:

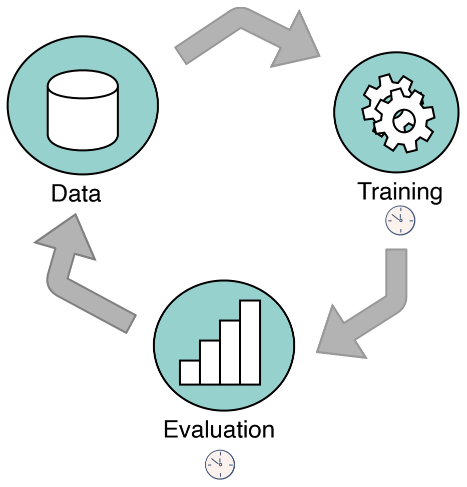
- **Training complexity**

  The training complexity of a machine learning algorithm is the time taken by it to train the model for a given task.
- **Evaluation complexity**

  The evaluation complexity of a machine learning algorithm is the time taken by it to evaluate the input at testing time.
- **Sample complexity**

  The sample complexity of a machine learning algorithm is the total number of training samples required to learn a target function successfully.

> 📝 Sample complexity changes if the model capacity changes. For example, for a deep neural network, the number of training examples has to be considerably larger than decision trees and linear regression.



The measure of training and evaluation time complexity on a sample data in a ML system

Comparison of training and evaluation complexities#

You can see how the training and evaluation complexities can be used to evaluate which model will be best for a given task and resources.

Assume that

- n is the number of the training samples
- f is the number of features
- n_{trees} is the number of trees (for tree-based algorithms)
- n_{l_i} is the number of neurons at i^{th} layer in a neural network
- e is the number of epochs
- d is the max depth of the tree

The training and prediction complexity can be approximated in terms of asymptotic analysis as follows:

Algorithm	Training time	Evaluation time

Algorithm	Training time	Eval[illegible]n
Linear/Logistic Regression (Batch)	$O(nfe)$	$O(f)$
Neural Network	Exponential (varies per implementation)	$O(fn_{l_1} + n_{l_1}n_{l_2} + ...)$
Multiple Additive Regression Trees (MART)	$O(ndfn_{trees})$	$O(fdn_{trees})$

Analysis#

- The evaluation complexity of the *linear regression* algorithm is equal to the complexity of a single-layer neural network-based algorithm. Linear regression is the best choice if we want to save time on training and evaluation. Let's assume the model evaluates one example in 5 μs. For 100k examples, it would take 100k x 5 μs = 500 ms execution time on a single machine.

  For example, for the ad prediction system, the service level agreement(SLA) says that we need to select the relevant ads from the pool of ads in 300 ms. Given this request, we need a fast algorithm. Here linear regression would serve the purpose.

- Relatively *deep neural network* takes a lot more time in both training and evaluation. Its need for training data is also high. However, it's ability to learn complex tasks such as image segmentation and languag understanding, is much higher, and it gives more accurate predictions in comparison to other models. Therefore a deep neural network is a in comparison to other models. Therefore a deep neural network is a

viable choice if it is well suited for the task at hand and capacity isn't a problem.

- *MART* is a tree-based algorithm that has a greater computation cost than linear models, but it is much faster than a deep neural network. Tree-based algorithms are able to generalize well using a moderately-sized training dataset. Therefore, if our training data is limited to a few million examples and capacity/performance is critical, they will be a good choice.

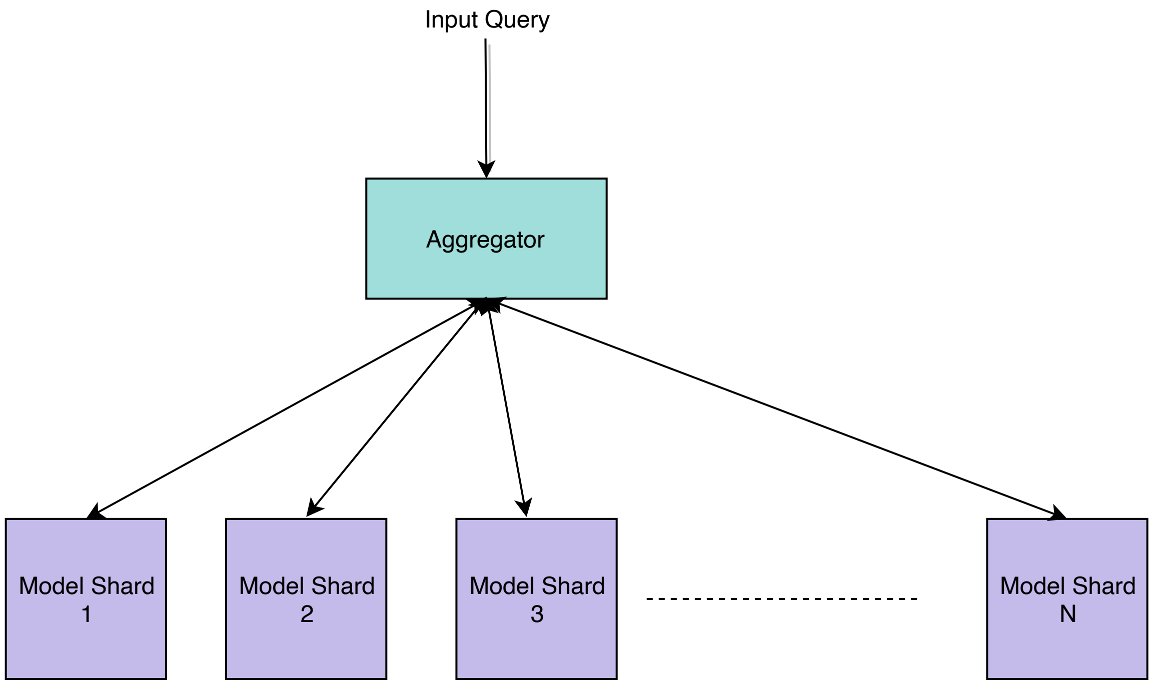
Performance and capacity considerations in large scale system#

Consider that a search system(e.g., Google, Bing) gets a query "computer science" that matches 100 million web pages. The ML-based system wants to respond with the most relevant web pages for the searcher while meeting the system's constraints. These constraints are generally referred to as Service level agreements (SLA). There can be many SLAs around availability and fault tolerance but for our discussion of designing ML systems, *performance* and *capacity* are the most important to think about when designing the system. **Performance** based SLA ensures that we return the results back within a given time frame (e.g. 500ms) for 99% of queries. **Capacity** refers to the load that our system can handle, e.g., the system can support 1000 QPS (queries per second).

If we evaluate every document using a relatively fast model such as tree-based or linear regression and it takes $1\mu s$, our simple model would still take 100s to run for 100 million documents that matched the query "computer science".

This is where distributed systems come in handy; we will distribute the load of a single query among multiple shards, e.g., we can divide t ad n

1000 machines and can still execute our fast model on 100 million documents in 100ms (100s/1000).



Input query sent to multiple shards to distribute load

Let's now consider the scenarios in which we decided that a deep learning model for search ranking is a much better choice and helps improve our search metrics. However, deep learning is significantly slow, assuming that it needs 1ms to evaluate an example. Even now with our 1000 shards, it would still take 100s to rank all the results using this model. Clearly, we are far from our performance SLAs.

If we had unlimited capacity, one way to solve this problem would be to continue to add more shards and bring the number down. However, we cannot have unlimited capacity, and it's important to have the system find the most optimal result given a fixed capacity.

This prompts another important discussion on using a funnel-based approach when designing ML systems for performance and limited capacity.

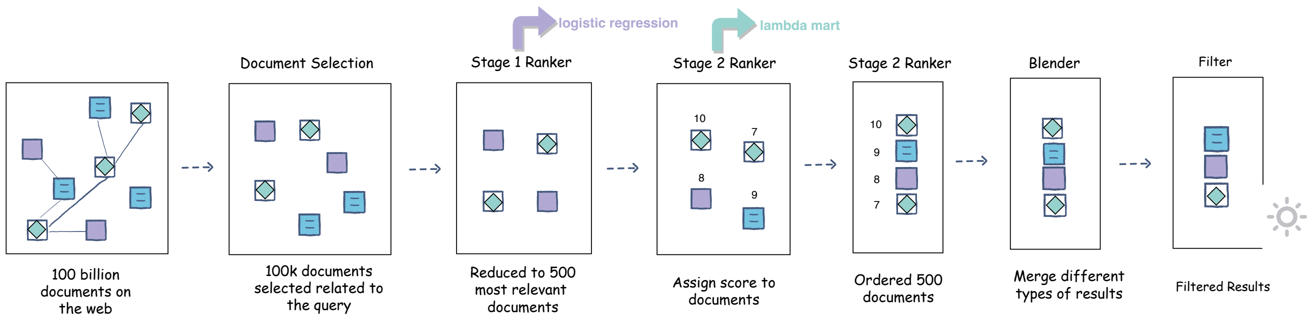
Let's discuss it next.

Layered/funnel based modeling approach#

To manage both the performance and capacity of a system, one reasonable approach that's commonly used is to start with a relatively fast model when you have the most number of documents e.g. 100 million documents in case of the query "computer science" for search. In every later stage, we continue to increase the complexity (i.e. more optimized model in prediction) and execution time but now the model needs to run on a reduce number of documents e.g. our first stage could use a linear model and final stage can use a deep neural network. If we apply deep neural network for only top 500 documents, with 1ms evaluation time per document, we would need 500ms on a single machine. With five shards we can do it in around 100ms.

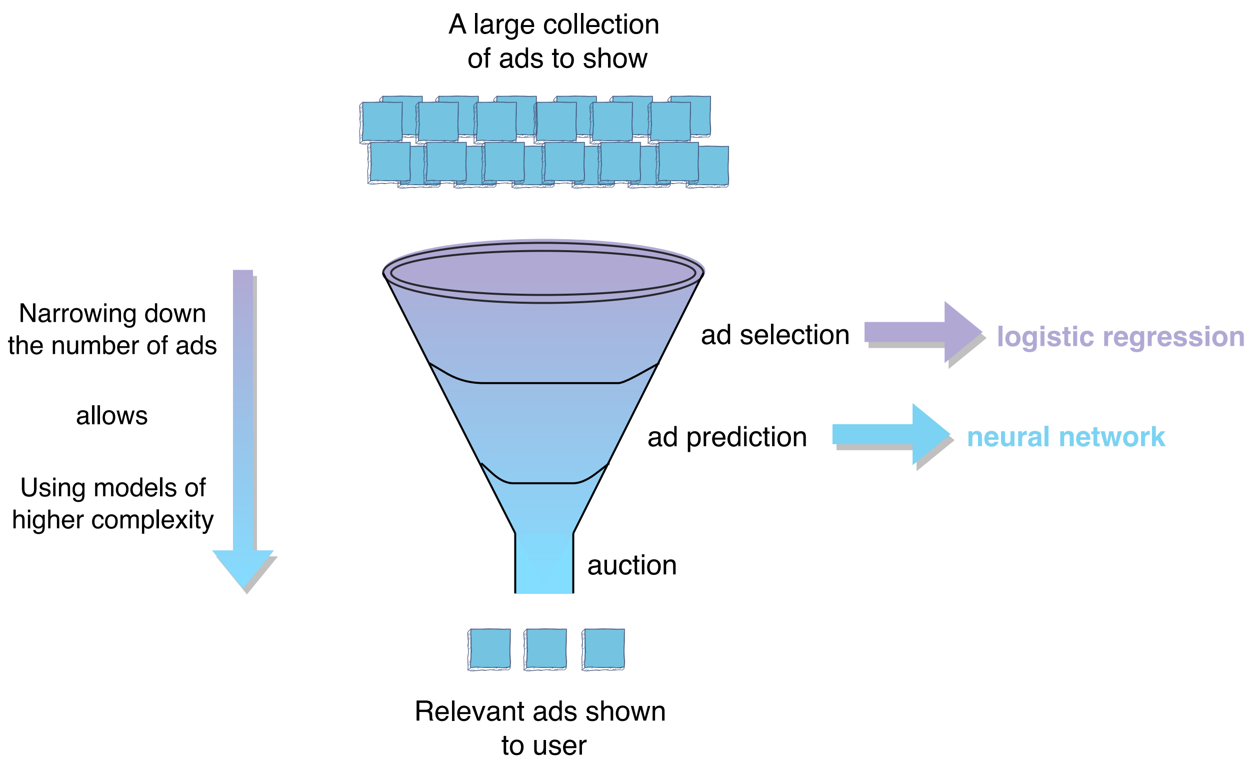
> 📝 In ML systems like search ranking, recommendation, and ad prediction, the layered/funnel approach to modeling is the right way to solve for scale and relevance while keeping performance high and capacity in check.

The following figure shows an illustration of this multi-layer funnel approach that we will explain in detail in the search ranking system design problem.


logistic regression
lambda mart
Document Selection
Stage 1 Ranker
Stage 2 Ranker
Stage 2 Ranker
Blender
Filter
10
7
8
9
10
9
8
7
100 billion documents on the web
100k documents selected related to the query
Reduced to 500 most relevant documents
Assign score to documents
Ordered 500 documents
Merge different types of results
Filtered Results

Layered model approach for search ranking system

Similarly, the following is an illustration of how an ads relevance system would look in this funnel based approach. We will discuss this in more detail in the ad prediction system chapter.


A large collection of ads to show
Narrowing down the number of ads
allows
Using models of higher complexity
ad selection
logistic regression
ad prediction
neural network
auction
Relevant ads shown to user

Funnel approach for an ad prediction system