Training Data Collection Strategies

Learn the training data collection strategies for the machine learning systems you are going to build.

We'll cover the following

Significance of training data

A machine-learning system consists of three main components. They are the training algorithm (*e.g., neural network, decision trees, etc.*), training data and features. The training data is of paramount importance. The model learns directly from the data to create and refine its rules on a given task. Therefore, inadequate, inaccurate, or irrelevant data will render even the most performant algorithms useless. The quality and quantity of training data is a big factor in determining how far you can go in our machine learning optimization task.

> *A lot of the progress in machine learning - and this is an unpopular opinion in academia - is driven by an increase in both computing power and* ***data****. An analogy is to building a space rocket: You need a huge rocket engine, and you need a lot of fuel.*
>
> — Andrew Ng

Most real-world problems fall under the category of supervised learning problems which require labeled training data. This means that it is necessary to strategically think about the collection of labeled data to feed into your learning system.

Let's explore strategies that will help in collecting labeled training data for our learning tasks.

Collection techniques

We will begin by looking at the training data collection techniques.

User's interaction with pre-existing system (online)

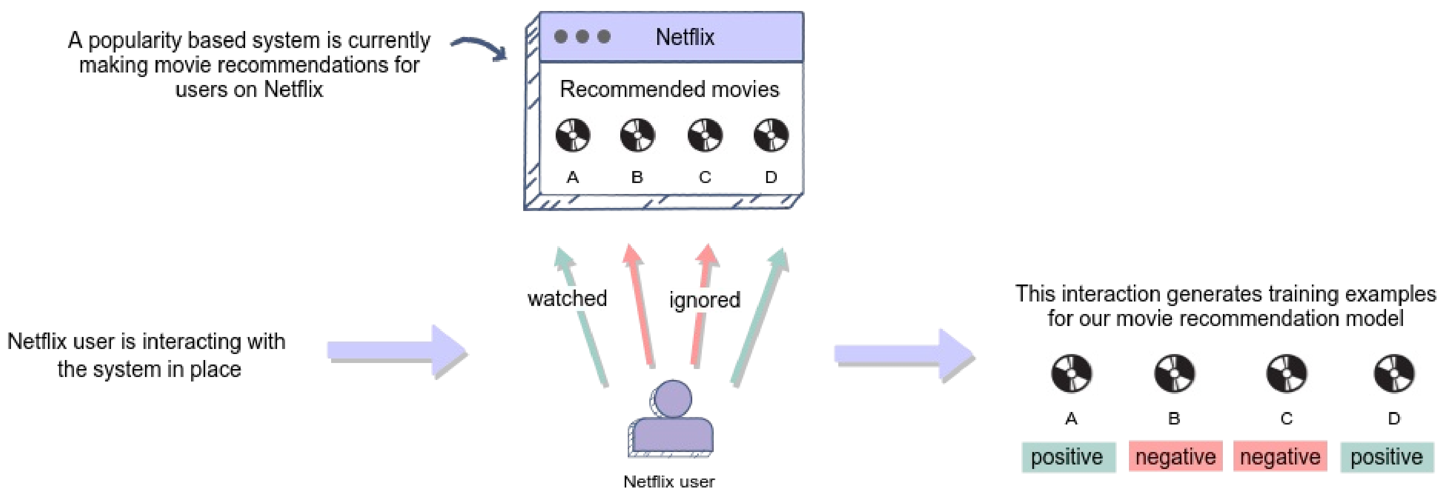
In some cases, the user's interaction with the pre-existing system can generate good quality training data.

> We will refer to this technique as online data collection in the course.

For many cases, the early version built for solving relevance (https://en.wikipedia.org/wiki/Relevance_(information_retrieval)) or ranking (https://en.wikipedia.org/wiki/Learning_to_rank) problem is a rule-based system. With the rule-based system in place, you build an ML system for the task (which is then iteratively improved). So when you build the ML system, you can utilize the user's interaction with the in-place/pre-existing system to generate training data for model training. Let's get a better idea with an example: building a movie recommender system.

Assume that you are asked to recommend movies to a Netflix user. You will be training a model that will predict which movies are more likely to be enjoyed/viewed by the user. You need to collect positive examples *(cases where user liked a particular movie)* as well as negative examples *(cases where the user didn't like a particular movie)* to feed into your model.

Here, the consumer of the system (the Netflix user) can generate training data through their *interactions with the current system* that is being used for recommending movies.


A popularity based system is currently making movie recommendations for users on Netflix
Netflix
Recommended movies
A
B
C
D
watched
ignored
Netflix user is interacting with the system in place
Netflix user
This interaction generates training examples for our movie recommendation model
A
B
C
D
positive
negative
negative
positive

Online data collection through a user's interaction with the system in place

The early version for movie recommendation might be popularity-based, localization-based, rating-based, hand created model or ML-based. The important point here is that we can get training data from the user's interaction with this system. If a user likes/watches a movie recommendation, it will count as a positive training example, but if a user dislikes/ignores a movie recommendation, it will be seen as a negative training example.

The above discussion was one example, but many machine learning systems utilize the current system in place for the generation of training data.

We will discuss training data generation strategies from the current system in multiple problems in this course, such as search ranking, ads relevance and recommendation systems.

Human labelers (offline)

In other cases, the user of the system would not be able to generate training data. Here, you will utilize labelers to generate good quality training data.

We will refer to this technique as offline data collection in the course.

Let's take an example of one such case: an image segmentation system for a self-driving car.

Assume that you are asked to perform image segmentation of the surroundings of a self-driving vehicle. The self-driving car will have a camera to take pictures of its surroundings. You will be training a model that will segment these pictures/images into various objects such as the road, pedestrians, building, sidewalk, signal and so on. For this, you will need to provide accurately segmented images of the self-driving car's surroundings as training data to the model.

Here, the consumer of the system (the person sitting in the self-driving car) can't generate training data for us. They are not interacting with the system in a way that would give segmentation labels for the images captured by the camera.

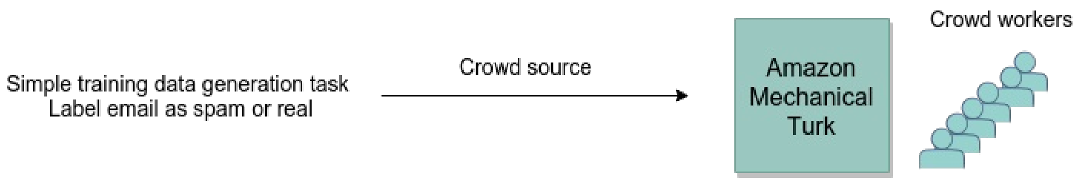
In such a scenario, we need to figure out the person/resource that can generate labeled training data for us.

Three such resources are:

1. **Crowdsourcing**

   As the name implies, in crowdsourcing, we outsource a task to a large group of people. Several crowdsourcing websites, such as Amazon Mechanical Turk (https://en.wikipedia.org/wiki/Amazon_Mechanical_Turk), where we can get quick results by hiring a lot of people for on-demand tasks, are available for this purpose.

   Crowdsourcing can be used to collect training data for relatively simpler tasks. For instance, if you are building an email spam detection system, you would need to label emails as spam or real. This is a simple task that *crowd workers* can easily do without requiring any special training.

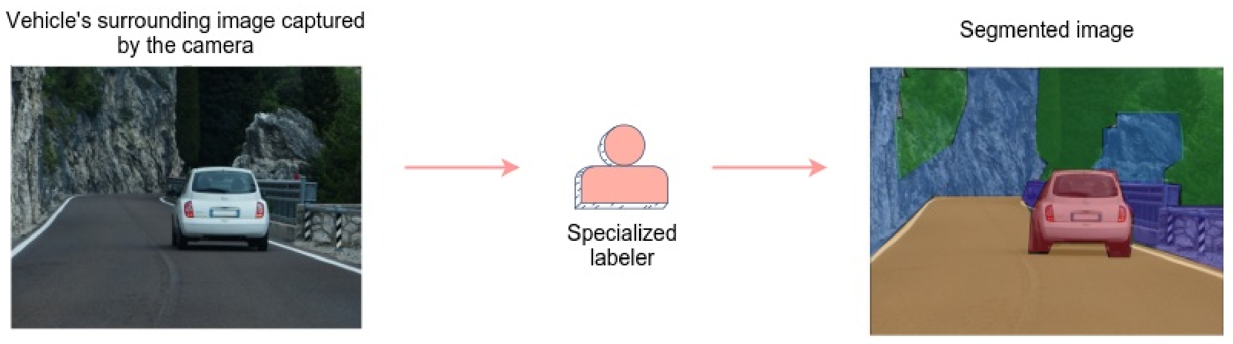

Simple training data generation task
Label email as spam or real
Crowd source
Amazon Mechanical Turk
Crowd workers

Crowdsourcing the task of generating labeled training data

However, there are cases, like when we have privacy concerns, where we cannot utilize crowdsourcing. If we do not want to show user emails to outsiders, we won't be using crowdsourcing. Another case where crowdsourcing falls short is when the task at hand requires specialized training. In these cases, we need to have specialized labelers to do the job.

2. **Specialized labelers**

   We can hire specialized/trained labelers who can label data for us according to the given ML task. Let's say, you have to build a system for the segmentation of driving images for an autonomous vehicle. The trained labelers will use software, such as Label box, to mark the boundaries of different objects in the driving images. One caveat of using specialized labelers is that training them for a specialized task may be time-consuming and costly. The tasks would be delayed until enough labelers have received training.



Specialized labellers generating training data

Targeted data gathering

Offline training data collection is expensive. So, you need to identify what kind of training data is more important and then target its collection more. To do this, you should see where the system is failing, i.e., areas where the system is unable to predict accurately. Your focus should be to collect training data for these areas.
Continuing with the autonomous vehicle example, you would see where your segmentation system is failing and collect data for those scenarios. For instance, you may find that it performs poorly for night time images and where multiple pedestrians are present. Therefore, you will focus more on gathering and labeling night time images and those with multiple pedestrians.
Have a look at the model debugging and testing lesson (https://www.educative.io/collection/page/10370001/6237869033127936/4824081099653120) to find out how to identify areas where the system is performing poorly.

3. **Open-source datasets**

   Generating training data through manual labelers is an expensive and time-consuming way to gather data. So, we need to supplement it with *open-source datasets* where possible.

For instance, “BDD100K: A Large-scale Diverse Driving Video Database” is an example of an open-source dataset that can be used as training data for this segmentation task. It contains labeled segmented data for driving images.

BDD100K: open source data set for segmentation

> We may use more than one of these methods for ways to generating training data. The search ranking chapter (https://www.educative.io/collection/page/10370001/6237869033127936/5102082685140992) is a good example of this.

Additional creative collection techniques

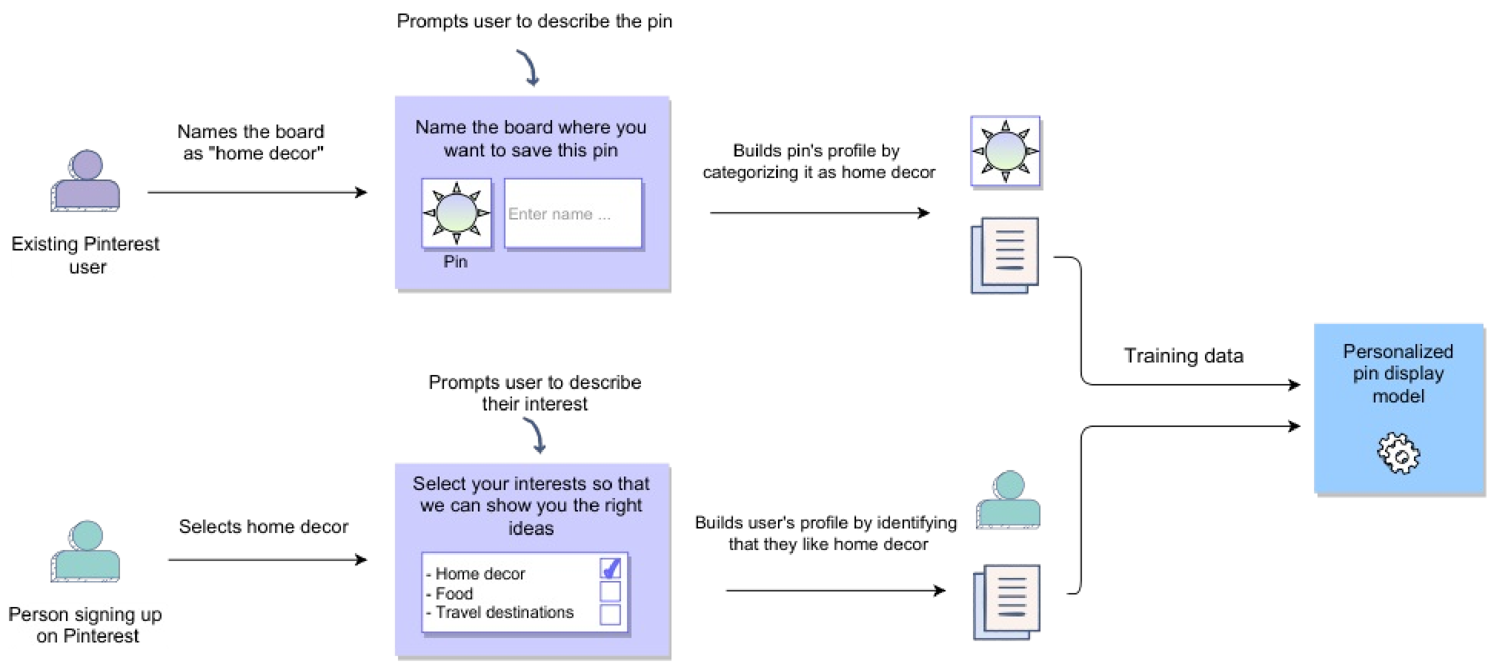
Let’s discuss a few creative ways to *collect and expand* training data.

- **Build the product in a way that it collects data from user**

  We can tweak the functionality of our product in a way that it starts generating training data for our model. Let’s consider an example where people go to explore their interests on Pinterest. You want to show a personalized selection of pins to the new users to kickstart their experience. This requires data that would give you a semantic understanding of the user and the pin. This can be done by tweaking the system in the following way:

  - Ask users to name the board (collection) to which they save each pin. The name of the board will help to categorize the pins according to their content.
  - Ask new users to choose their interests in terms of the board names specified by existing users.

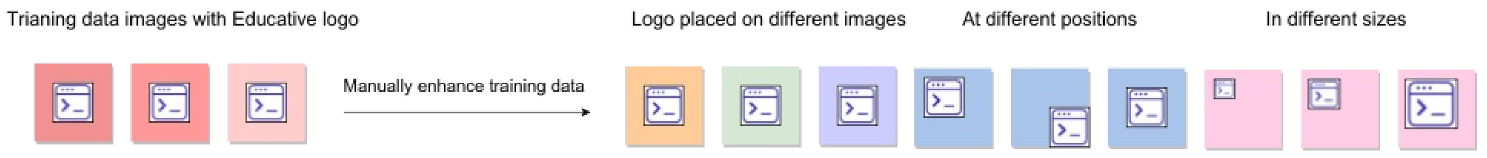
  The first step will help you to build content profiles. Whereas, the second step will help you build user profiles. The model can utilize these to show pins that would interest the user, personalizing the experience.


Prompts user to describe the pin
Names the board as "home decor"
Existing Pinterest user
Name the board where you want to save this pin
Enter name ...
Pin
Builds pin's profile by categorizing it as home decor
Training data
Personalized pin display model
Prompts user to describe their interest
Selects home decor
Person signing up on Pinterest
Select your interests so that we can show you the right ideas
- Home decor
- Food
- Travel destinations
Builds user's profile by identifying that they like home decor

Designing the product in a way that generates training data

- **Creative manual expansion**

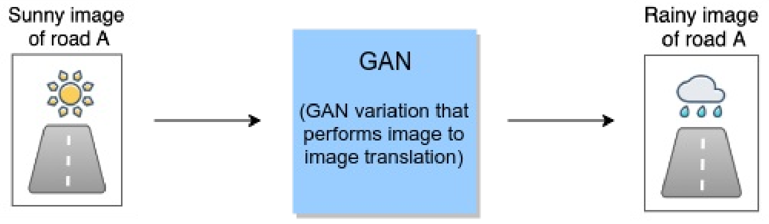
  We can expand training data using creative ways. Assume that we are building a system that detects logos in images (object detection) and we have some images containing the logos we want to detect. We can expand/enhance the training data by manually placing those logos on a different set of images as well. This logo placement can be done in different positions and sizes. The enhanced training data will enable us to build a more robust logo detector model, which will be able to identify logos of all sizes at different positions and from various kinds of images.


Trianing data images with Educative logo
Manually enhance training data
Logo placed on different images
At different positions
In different sizes

Creative manual expansion of training data for logo detection

- **Data expansion using GANs**

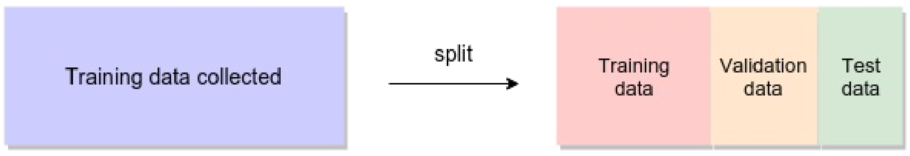
  When working with systems that use visual data, such as object detectors or image segmenters, we can use GANs (generative adversarial networks) to enhance the training data. For instance, consider that we are building an object detector. There are a lot of training images os sunny weather conditions but less of rainy weather. A model trained on this training data may not work well for images with rainy conditions. Here, we can utilize GANs to convert images with sunny weather conditions to rainy weather conditions. This will increase our training data for rainy conditions, resulting in a more robust model.



Training data expansion through GANs: converting images in sunny weather condition to rainy weather condition

Train, test, & validation splits

We have looked at various methods to collect training data. Now it is time to split it into three parts and look at the importance of each split.



The collected training data is split into training data, validation data, and test data

The three parts are as follows:

1. **Training data**

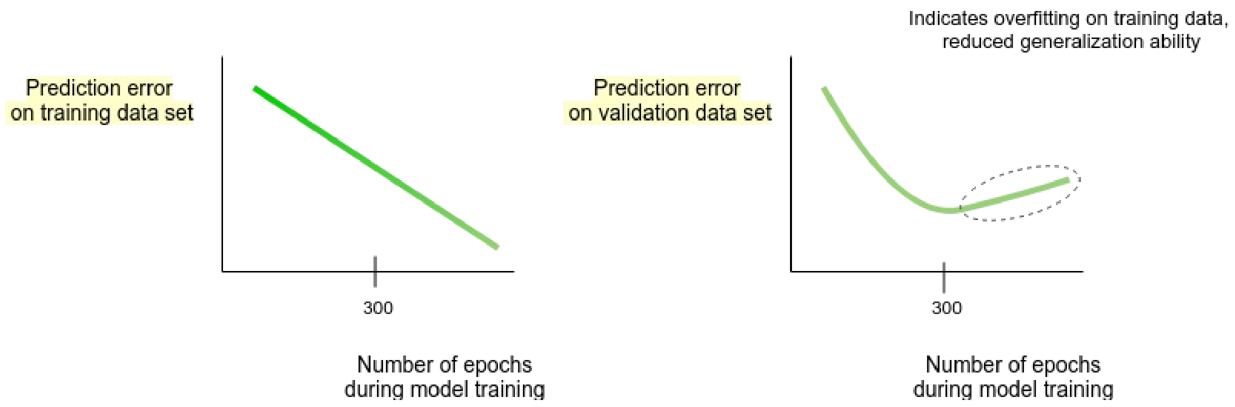
   It helps in training the ML model (fit model parameters).

   Click to expand: model parameters vs. model hyperparameters

2. **Validation data**

   After training the model, we need to tune its hyperparameters (try different model configurations). This process requires *testing the model's performance* on various hyperparameter combinations to select the best one. A simple idea could be to use the training data itself for this task. However, testing the model's performance on the same data that it was trained on would not give an accurate picture of the model's generalization ability. A good performance score may be the result of the model overfitting the training data. So we use the validation set while tuning the hyperparameters.

   Consider a scenario where you are training a neural network. You want to know the optimal number of epochs. The prediction error on the training set will continue to decrease as you keep on experimenting with a higher number of epochs. However, if you compute the error on the validation set, you would find out that maybe after 300 epochs the prediction error starts to increase, indicating that an epoch number higher than 300 will overfit the training data.



The validation set gives true picture of model's generalization ability

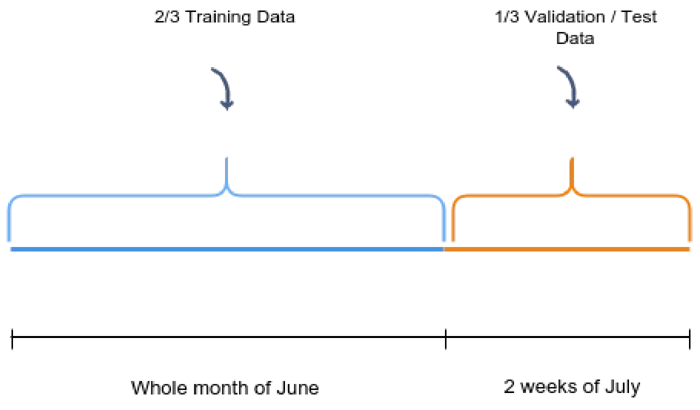
3. **Test data**

Now that we have trained and tuned the model, the final step is to test its performance on data that it has not seen before. In other words, we will be testing the model's generalization ability. The outcome of this test will allow us to make the final choice for model selection. If we had trained several models, we can decide the best ones amongst them and then further see if their performance boost is significant enough to call for an online A/B test.

We pointed out earlier that *we need a data set, which the model has not directly learned from, for this test.*. As such, training data is out of the question. Validation data is also not a suitable choice. It indirectly impacts the model as it's used to tune the model hyperparameters. This is where the test data set comes in. Test data is a completely held out data set that provides an unbiased evaluation of the final model, which has been fit on the training dataset.

Points to consider during splitting

- The size of each split will depend on your particular scenario. The training data will generally be the largest portion, especially if you are training a model like a deep neural network that requires a lot of training data. Common ratios used for training, validation and test splits are 60%, 20%, 20% or 70%, 15%, 15% respectively.
- While splitting training data, you need to ensure that you are capturing all kinds of patterns in each split. For example, if you are building a movie recommendation system like Netflix, your training data would consist of users' interactions with the movies on the platform. After analysing the data, you may conclude that, generally, the user's interaction patterns differ throughout the week. Different genres and movie lengths are preferred on different days. Hence, you will use the interaction with recommendations throughout a week to capture all patterns in each split.
- Most of the time, we are building models with the intent to forecast the future. Therefore, you need your splits to reflect this intent as well. For instance, in the movie recommendation system example, your data has a time dimension, i.e., you know the users' interactions with previous movie recommendations, and you want to predict their interactions with future recommendations ahead of time. Hence, you will train the model on data from one time interval and validate/test it on the data from its succeeding time interval, as shown in the diagram below. This will provide a more accurate picture of how our model will perform in a real scenario.

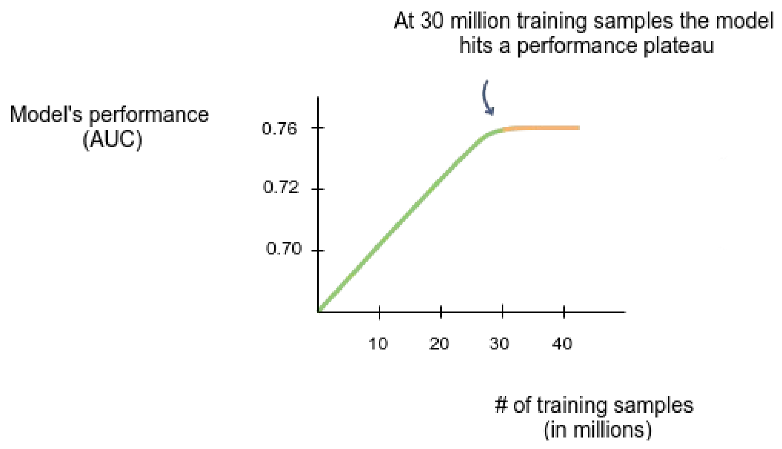


Splitting data for training, validation, and testing

Quantity of training data

As a general guideline, the quantity of the training data required depends on the modeling technique you are using. If you are training a simple linear model, like linear regression, the amount of training data required would be less in comparison to more complex models. If you are training complex models, such as a neural network, the magnitude of data required would be much greater.

Gathering a large amount of training data requires time and effort. Moreover, the model training time and cost also increase as we increase the quantity of training data. To see the optimal amount of training data, you can plot the model's performance against the number of training data samples, as shown below. After a certain quantity of training data, you can observe that there isn't any gain in the model's performance.



Plot of the model's performance on the validation set against the number of training data samples

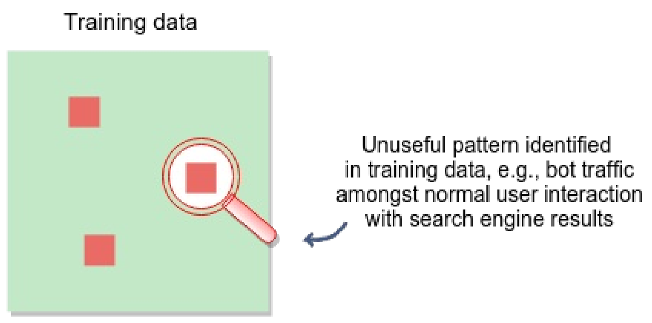
Training data filtering

It is essential to filter your training data since the model is going to learn directly from it. Any discrepancies in the data will affect the learning of the model.

- **Cleaning up data**

  General guidelines are available for data cleaning such as handling missing data, outliers, duplicates and dropping out irrelevant features.

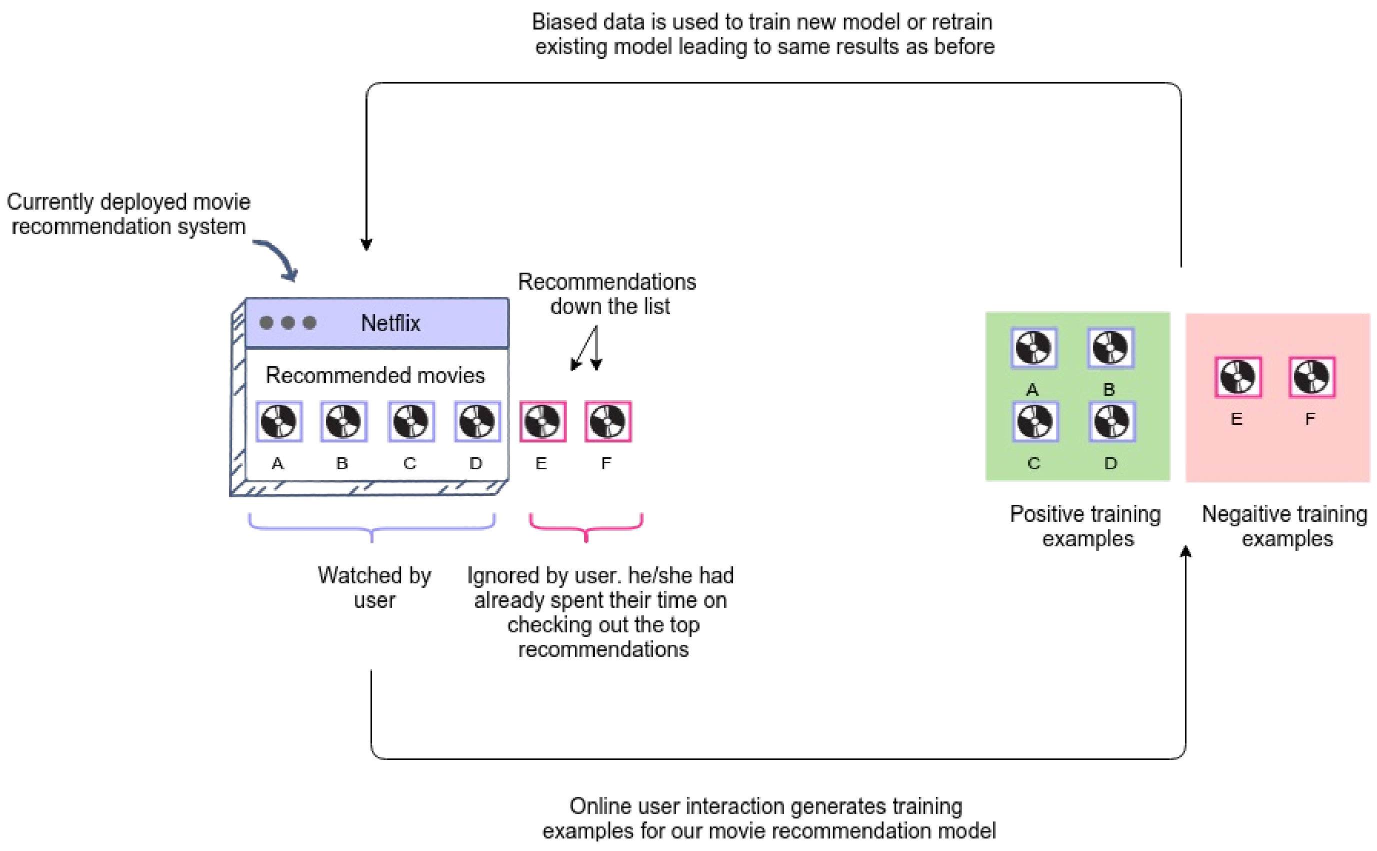
  Apart from this, you need to analyze the data with regards to the given task to identify patterns that are not useful. For instance, consider that you are building a search engine's result ranking system. Cases where the user clicks on a search result are considered positive examples. In comparison, those with just an impression are considered negative examples. You might see that the training data consist of a lot of bot traffic apart from the real user interactions. Bot traffic would just contain impressions and no clicks. This would introduce a lot of wrong negative examples. So we need to exclude them from the training data so that our model doesn't learn from wrong examples.


Training data
Unuseful pattern identified in training data, e.g., bot traffic amongst normal user interaction with search engine results

- **Removing bias**

  When we are generating training data through online user engagement, it may become biased. Removing this bias is critical. Let's see why by taking the example of a movie recommender system like Netflix.

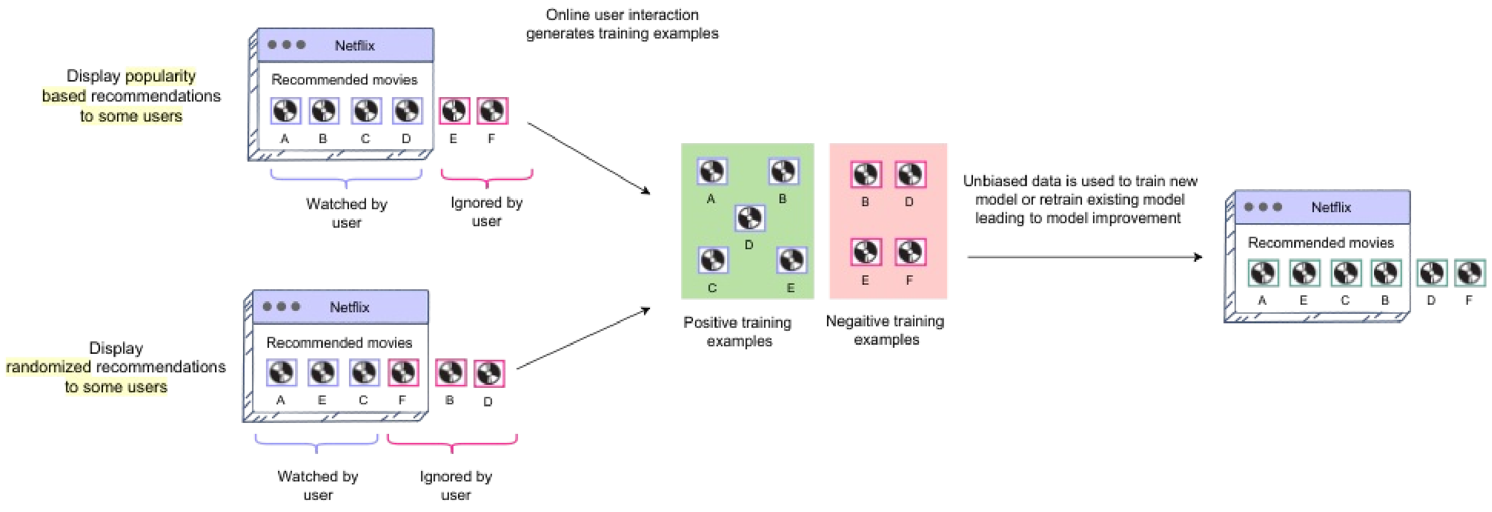
  The pre-existing recommender is showing recommendations based on popularity. As such, the popular movies always appear first and new movies, although they are good, always appear later on as they have less user engagement. Ideally, the user should go over the whole recommendation list so that he/she can discover the good, new movies as well. This would allow us to classify them as positive training examples so that the model can learn to put them on top, once re-trained. However, due to the user's time constraints, he/she would only interact with the topmost recommendations resulting in the generation of biased training data. The model hence trained, will continue considering the previous top recommendation to be the top recommendation this time too. Hence, the "rich getting richer" cycle will continue.



Biased data is used to train new model or retrain existing model leading to same results as before
Currently deployed movie recommendation system
Netflix
Recommended movies
A
B
C
D
E
F
Recommendations down the list
Watched by user
Ignored by user. he/she had already spent their time on checking out the top recommendations
A
B
C
D
Positive training examples
E
F
Negaitive training examples
Online user interaction generates training examples for our movie recommendation model

Biased training data generated through online user engagement

In order to break this cycle, we need to employ an exploration technique that explores the whole content pool (all movies available on Netflix). Therefore, we show "randomized" recommendations instead of "popular first" for a small portion of traffic for gathering training data. The users' engagement with the randomized recommendations provides us with unbiased training data. This data really helps in removing the current positional and engagement bias of the system.


Display popularity based recommendations to some users
Netflix
Recommended movies
A B C D E F
Watched by user
Ignored by user
Online user interaction generates training examples
Display randomized recommendations to some users
Netflix
Recommended movies
A E C F B D
Watched by user
Ignored by user
A B D C E
Positive training examples
B D E F
Negaitive training examples
Unbiased data is used to train new model or retrain existing model leading to model improvement
Netflix
Recommended movies
A E C B D F

Removing bias through exploration technique

- **Bootstrapping new items**

  Sometimes we are dealing with systems in which new items are added frequently. The new items may not garner a lot of attention, so we need to boost them to increase their visibility. For example, in the movie recommendation system, new movies face the cold start problem (https://en.wikipedia.org/wiki/Cold_start_(recommender_systems)). We can boost new movies by recommending them based on their similarity with the user's already watched movies, instead of waiting for the new movies to catch the attention of a user by themselves. Similarly, we may be building a system to display ads, and the new ads face the cold start problem. We can boost them by increasing their relevance scores a little, thereby artificially increasing their chance of being viewed by a person.