Problem Statement

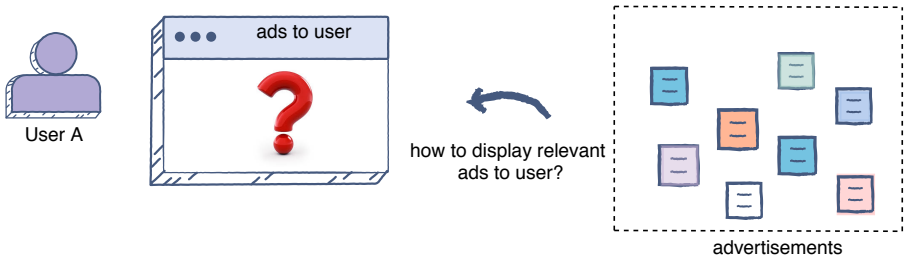
Let's look at the problem statement for the ad prediction system.

We'll cover the following

- Problem statement
- Visualizing the problem
- Interview questions

Problem statement

The interviewer has asked you to build a system to show relevant ads to users.



How to display the most relevant ads to the user?

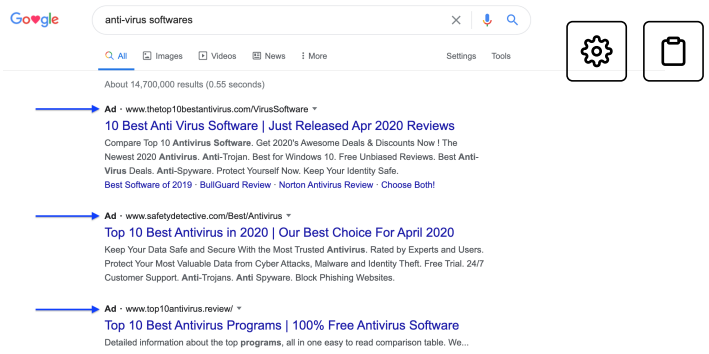
Visualizing the problem

There are two well-known advertising platforms Google and Facebook, that run advertisements paid for by businesses.

Have you ever wondered how Google and Facebook drive new (or existing) customers who are searching for the product or service that you provide to your website using advertisements?

When you type a search query in a google.com (http://google.com) search bar, ads appear in the search results.

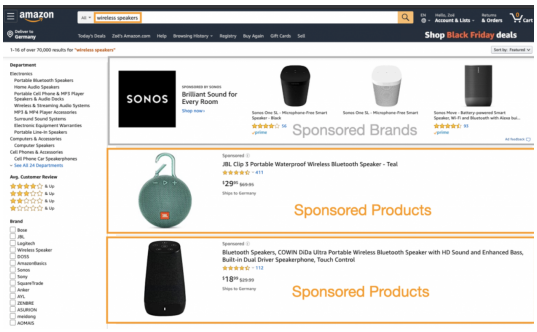
Google's search network allows advertisers to show their business advertisements to users who are actively looking for products or services that the advertiser provides.



Within the Search Network, Google matches a keyword that is relevant to a product or business advertiser, so that when a user issues a query, it triggers an ad that shows up on the search page for a user to click on.

Unlike Google, Facebook (https://www.facebook.com/) ads are mostly shown on users' news feed based on users' interests.

Amazon.com (http://Amazon.com) also shows ads, generally in their search result page based on query context as shown in the picture.



Interview questions

The interviewer can ask the following questions about this problem, narrowing the scope of the question each time.

- How would you build an ML system to predict the probability of engagement for Ads?
- How would you build an Ads relevance system for a search engine?
- How would you build an Ads relevance system for a social network?

📝 Note that the context can be different depending on the type of application in which we are displaying the advertisement.

There are two categories of applications:

Search engine: Here, the query will be part of the context along with the searcher. The system will display ads based on the search query issued by the searcher.

Social media platforms: Here, we do not have a query, but the user information (such as location, demographics, and interests hierarchy) will be part of the context and used to select ads. The system will automatically detect user interest based on the user's historical interactions (using machine learning algorithms) and display ads accordingly.

📝 Most components will be the same for the above two discussed platforms with the main difference being the *context* that is used to select and predict ad engagement.

Let's set up the machine learning problem:

"Predict the probability of engagement of an ad for a given user and context(query, device, etc.)"

Back
Modeling

Next
Metrics

Metrics

Let's look at the online and offline metrics used to judge the performance of an ad's prediction system.

We'll cover the following

- Offline metrics
  - Log Loss
- Online metrics
  - Overall revenue
  - Overall ads engagement rate
  - Counter metrics

The metrics used in our ad prediction system will help select the best machine-learned models to show relevant ads to the user. They should also ensure that these models help the overall improvement of the platform, increase revenue, and provide value for the advertisers.

Like any other optimization problem, there are two types of metrics to measure the effectiveness of our ad prediction system:

1. Offline metrics
2. Online metrics

> 📝 Why are both online and offline metrics important?
>
> Offline metrics are mainly used to compare the models offline quickly and see which one gives the best result. Online metrics are used to validate the model for an end-to-end system to see how the revenue and engagement rate improve before making the final decision to launch the model.

Offline metrics

As we build models, the best way to compare them is to measure prediction accuracy instead of measuring revenue impact directly. The following are a few metrics that enable us to compare the two models better offline.

Log Loss

Let's first go over the *area under the receiver operator curve* (AUC), which is a commonly used metric for model comparison in binary classification tasks. However, given that the system needs well-calibrated prediction scores, AUC, has the following shortcomings in this ad prediction scenario.

1. AUC does not penalize for "how far off" predicted score is from the actual label. For example, let's take two positive examples (i.e., with actual label 1) that have the predicted scores of 0.51 and 0.7 at threshold 0.5. These scores will contribute equally to our loss even though one is much closer to our predicted value.
2. AUC is insensitive to well-calibrated probabilities.

Calibration

Since, in our case, we need the model's predicted score to be well-calibrated to use in Auction, we need a calibration-sensitive metric. Log loss should be able to capture this effectively as Log loss (or more precisely cross-entropy loss) is the measure of our predictive error.

This metric captures to what degree expected probabilities diverge from class labels. As such, it is an absolute measure of quality, which accounts for generating well-calibrated, probabilistic output.

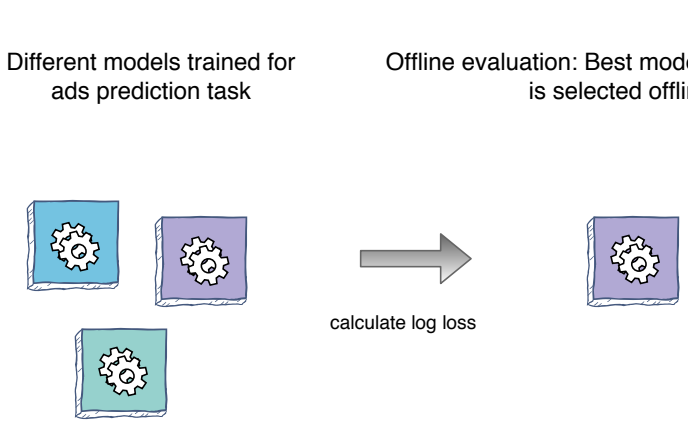
Let's consider a scenario that differentiates why log loss gives a better output compared to AUC. If we multiply all the predicted scores by a factor of 2 and our average prediction rate is double than the empirical rate, AUC won't change but log loss will go down.

In our case, it's a binary classification task; a user engages with ad or not. We use a 0 class label for no-engagement (with an ad) and 1 class label for engagement (with an ad). The formula equals:

$$-\frac{1}{N}\sum_{i=1}^{N}[y_i \log\, p_i + (1-y_i) \log\, (1-p_i)].$$

Here:

- N is the number of observations
- y is a binary indicator (0 or 1) of whether the class label is the correct classification for observation
- p is the model's predicted probability that observation is of class (0 or 1)



Online metrics

For online systems or experiments, the following are good metrics to track:

Overall revenue

This captures the overall revenue generated by the system for the cohorts of the user in either an experiment or, more generally, to measure the overall performance of the system. It's important to call out that just measuring revenue is a very short term approach, as we may not provide enough value to advertisers and they will move away from the system. However, revenue is definitely one critical metric to track. We will discuss other essential metrics to track shortly.

Revenue is basically computed as the sum of the winning bid value (as selected by auction) when the predicted event happens, e.g., if the bid amounts to $0.5 and the user clicks on the ad, the advertiser will be charged $0.5. The business won't be charged if the user doesn't click on the ad.

Overall ads engagement rate

Engagement rate measures the overall action rate, which is selected by the advertiser.

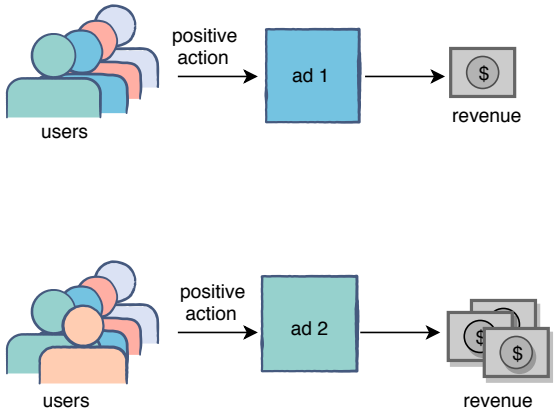
Some of the actions might be:

1. Click rate

This will measure the ratio of user clicks to ads.

2. Downstream action rate

This will measure the action rate of a particular action targeted by the advertiser e.g. add to cart rate, purchase rate, message rate etc.



More positive user engagement on the ad results in more revenue

Counter metrics

It's important to track counter metrics to see if the ads are negatively impacting the platform.

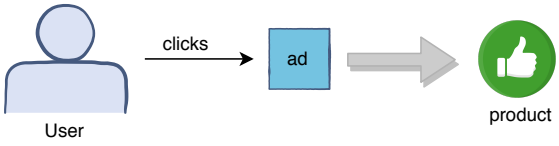
We want the users to keep showing engagement with the platform and ads should not hinder that interest. That is why it's important to measure the impact of ads on the overall platform as well as direct negative feedback provided by users. There is a risk that users can leave the platform if ads degrade the experience significantly.

So, for online ads experiments, we should track key platform metrics, e.g., for search engines, is session success going down significantly because of ads? Are the average queries per user impacted? Are the number of returning users on the platform impacted? These are a few important metrics to track to see if there is a significant negative impact on the platform.

Along with top metrics, it's important to track direct negative feedback by the user on the ad such as providing following feedback on the ad:

1. Hide ad
2. Never see this ad
3. Report ad as inappropriate

These negative sentiments can lead to the perceived notion of the product as negative.



The user clicks on the ad gives a positive impression of the product

All of the metrics discussed above can be used to measure user engagement with ad and advertiser revenue on user engagement.

Back

Problem Statement

Next

Architectural Components

Completed

Architectural Components

Let's see the architectural components of the Ads prediction system.

We'll cover the following

- Architecture
  - Ad selection
  - Ad prediction
  - Auction
    - Pacing
  - Training data generation
- Funnel model approach

Architecture

Let's have a look at the high-level architecture of the system. There will be two main actors involved in our ad prediction system - platform users and advertiser. Let's see how they fit in the architecture:

1. Advertiser flow

Advertisers create ads containing their content as well as targeting, i.e., scenarios in which they want to trigger their ads. A few examples are:

- *Query-based targeting*: This method shows ads to the user based on the query terms. The query terms can be a partial match, full match, expansion, etc.
  For example, whenever a user types a query "machine learning course", the system shows the ML course on educative.io.

- *User-based targeting*: The ads will be subjective to the user based on a specific region, demographic, gender, age, etc.

- *Interest-based targeting*: This method shows interest-based ads. Assume that on Facebook, the advertiser might want to show ads based on certain interest hierarchies. For example, the advertiser might like to show sports-related ads to people interested in sports.

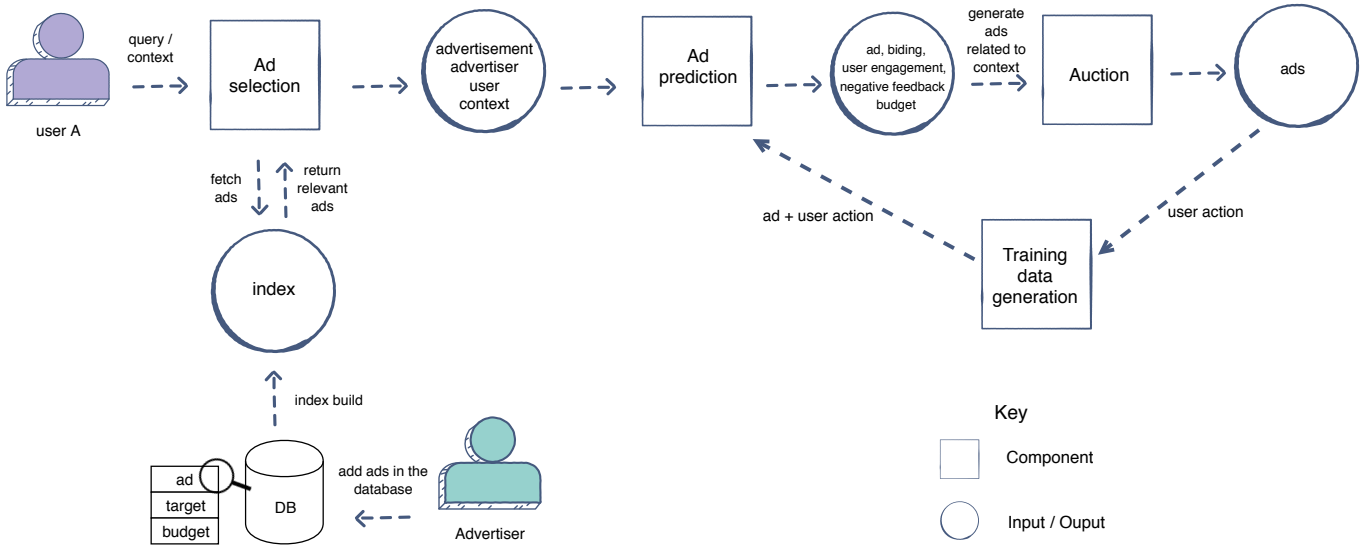
- *Set-based targeting*: This type shows ads to a set of users selected by the advertisers. For example, showing an ad to people who were previous buyers or have spent more than ten minutes on the website. Here, we can expand our set and do seed audience expansion.

2. User flow

As the platform user queries the system, it will look for all the potential ads that can be shown to this user based on different targeting criteria used by the advertiser.

So, the flow of information will have two major steps as described below:

- Advertisers create ads providing targeting information, and the ads are stored in the ads index.
- When a user queries the platform, ads can be selected from the index based on their information (e.g., demographics, interests, etc.) and run through our ads prediction system.



Architectural diagram of ad prediction system

Let's briefly look at each component here. Further explanation will be provided in the following lessons.

Ad selection

The ad selection component will fetch the top k ads based on relevance (subject to the user context) and bid from the ads index.

Ad prediction

The ad prediction component will predict user engagement with the ad (the probability that an action will be taken on the ad if it is shown), given the ad, advertiser, user, and context. Then, it will rank ads based on relevance score and bid.

Auction

The auction mechanism then determines whether these top K relevant ads are shown to the user, the order in which they are shown, and the price the advertisers pay if an action is taken on the ad.

For every ad request, an auction takes place to determine which ads to show. The top relevant ads selected by the ad prediction system are given as input to Auction. Auction then looks at total value based on an ad's bid as well as its relevance score. An ad with the highest total value is the winner of the auction. The total value depends on the following factors:

- **Bid**: The bid an advertiser places for that ad. In other words, the amount the advertiser is willing to pay for a given action such as click or purchase.
- **User engagement rate**: An estimate of user engagement with the ad.
- **Ad quality score**: An assessment of the quality of the ad by taking into account feedback from people viewing or hiding the ad.
- **Budget**: The advertiser's budget for an ad

The estimated user engagement and ad quality rates combined results in the ad relevance score. They can be combined based on different weights as selected by the platform, e.g., if it's important to keep positive feedback high, the ad quality rate will get a higher weight.

The rank order is calculated based on predicted ad score (from the ad prediction component) multipled by the bid. Let's call this the ad rank score.

$$Ad\ rank\ score = Ad\ predicted\ score * bid$$

Ads with the highest ad rank score wins the auction and are shown to the user. Once an ad wins the auction, the cost per engagement (CPE) or cost per click (CPC) will depend on its ad rank score and ad rank score of the ad right below it in rank order, i.e.,

$$CPE = \frac{Ad\ rank\ of\ ad\ below}{Ad\ rank\ score} + 0.01$$

A general principle is that the ad will cost the minimal price that still allows it to win the auction.

Pacing

Pacing an ad means evenly spending the ad budget over the selected time period rather than trying to spend all of it at the start of the campaign.

Remember that whenever the user shows engagement with an ad, the advertiser gets charged the bid amount of the next ad. If the ad rank score is high, the ad set can spend the whole budget at the start of a time period (like the start of a new day, as advertisers generally have daily budgets). This would result in a high cost per click (CPC) when the ad load (the user engagement rate) is high.

Pacing overcomes this by dynamically changing the bid such that the ad set is evenly spread out throughout the day and the advertiser gets maximum return on investment(ROI) on their campaign. This also prevents the overall system from having a significantly high load at the start of the day, and the budget is spent evenly throughout the campaign.

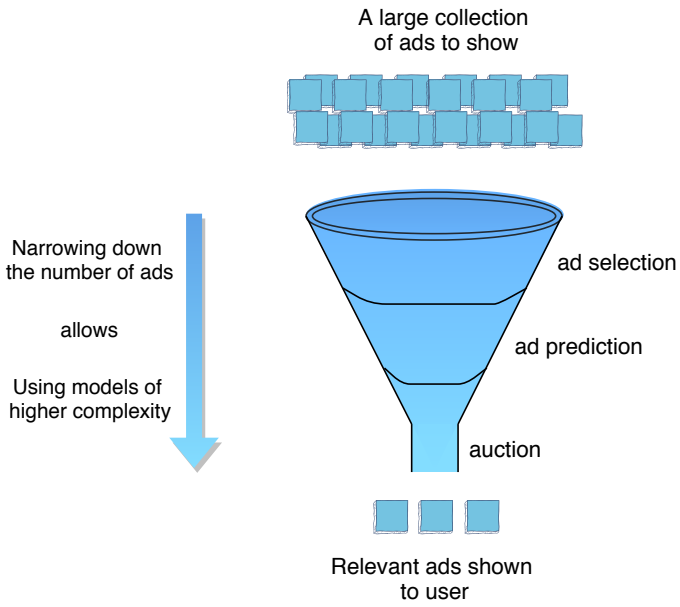
Training data generation

We need to record the action taken on an ad. This component takes user action on the ads (displayed after the auction) and generates positive and negative training examples for the ad prediction component.

Funnel model approach

For a large scale ads prediction system, it's important to quickly select an ad for a user based on either the search query and/or user interests. The scale can be large both in terms of the number of ads in the system and the number of users on the platform. So, it's important to design the system in a way that it can scale well and be extremely performance efficient without compromising on ads quality.

To achieve the above objective, it would make sense to use a funnel approach, we gradually move from a large set of ads to a more precise set for the next step in the funnel.



Funnel approach: get relevant ads for a user

As we go down the funnel (as shown in the diagram), the complexity of the models becomes higher and the set of ads that they run on becomes smaller. It's also important to note that the initial layers are mostly responsible for ads selection. On the other hand, ads prediction is responsible for predicting a well-calibrated engagement and quality score for ads. This predicted score is going to be utilized in the auction as well.

Let's go over an example to see how these components will interact for the search scenario.

- A thirty-year old male user issues a query "machine learning".
- The **ads selection** component selects all the ads that match the targeting criteria (user demographics and query) and uses a simple model to predict the ad's relevance score.
- The **ads selection** component ranks the ads according to r, where $r = bid * relevance$ and sends the top ads to our ads prediction system.
- The **ads prediction** component will go over the selected ads and uses a highly optimized ML model to predict a precise calibrated score.
- The **ads auction** component then runs the auction algorithm based on the bid and predicted ads score to select the top most relevant ads that are shown to the user.

We will zoom into the different segments of this diagram as we explore each segment in the upcoming lessons.

Back
Metrics

Next
Feature Engineering

Completed

Feature Engineering

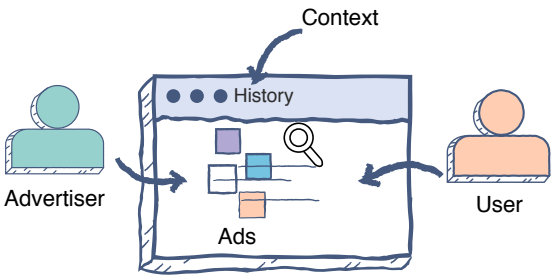
Let's engineer some features for the prediction model.

We'll cover the following

- Features for the model
  - Ad specific features
  - Advertiser specific features
  - User specific features
  - Context specific features
  - User-ad cross features
  - User-advertiser cross features

Features are the backbone of any learning system. Let’s think about the main actors or dimensions that will play a key role in our feature engineering process.

1. Ad
2. Advertiser
3. User
4. Context



Actors

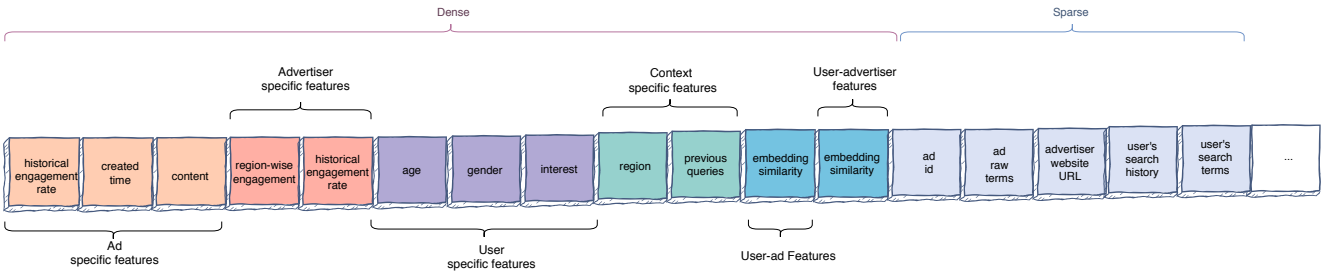
> 📝Context refers to the engagement history, user interests, current location, time and date, as discussed in the previous lessons.

Features for the model

Now it's time to generate features based on these actors. The features would fall into the following categories:

1. Ad specific features
2. Advertiser specific features
3. User specific features
4. Context specific features
5. User-ad cross features
6. User-advertiser cross features

A subset of the features is shown below:



Features in the training data row

Ad specific features

- **ad_id**

  A unique id is assigned to each ad and can be used as a sparse feature. Utilizing ad_id as a sparse feature allows the model to memorize historical engagement for each ad, and it can also be used in interesting cross features for memorization (such as ad_id * user interests). Additionally, we can also generate embeddings during training time for the ad using its id, as we will discuss in the ad prediction section.

- **ad_content_raw_terms**

  Ad terms can also be very useful sparse features. They can tell us a lot about the ad, e.g., a good model can learn from the text's content to identify what the ad is about, such as politics or sports. Raw terms allow the models (especially NN models) to learn such behavior from given raw terms.

- **historical_engagement_rate**

This feature specifies the rate of user engagement with the ad. Here we will measure engagement in different windows such as different times of the day or days of the week. For instance, we can have the following features:

- **ad_engagement_history_last_24_hrs**

  Since ads are short-lived, recent engagement is important. This feature captures the most recent engagement with the ad.

- **ad_engagement_history_last_7_days**

  This captures the activity on the ad on each day of the week. For example, an ad can get more engagement on weekends rather than on weekdays.

> 📝 This feature can tell the model that a particular ad is performing well. This prior engagement data can help predict future engagement behavior.

- **ad_impression**

  This feature records the number of times an ad is viewed. This is helpful since we can train the model on the engagement rate as a feature when we have a reasonable number of ad impressions. We can select the cut-off point until which we want to consider the impressions. For example, we can say that if a particular ad has twenty impressions, we can measure the historical engagement rate.

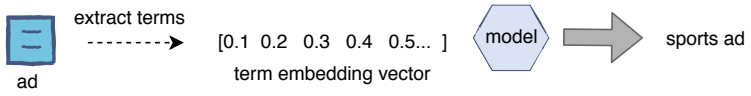
- **ad_negative_engagement_rate**

  This feature keeps a record of negative engagement (hide, report) with the ad.

- **ad_embedding**

  We can generate embedding for an ad given all the data that we know about it e.g. ad terms and engagement data. This embedding is then a dense representation of the ad that can be used in modeling. Please refer to our embedding lesson (https://www.educative.io/collection/page/10370001/6237869033127936/6130870193750016) to see how we can generate this embedding.

  Ad embedding can also be used to detect the ad's category, e.g., it can tell if the ad belongs to sports, etc.



Detect the category/group from the given ad embedding

- **ad_age**

  This feature specifies how old the ad is.

- **ad_bid**

  We can record the bid for the ad specified by the advertiser.

Advertiser specific features

- **advertiser_domain**

  This is a sparse feature that keeps a record of the domain name for an advertiser. This can be used in the same way for memorization and embedding generation as discussed for ad_id.

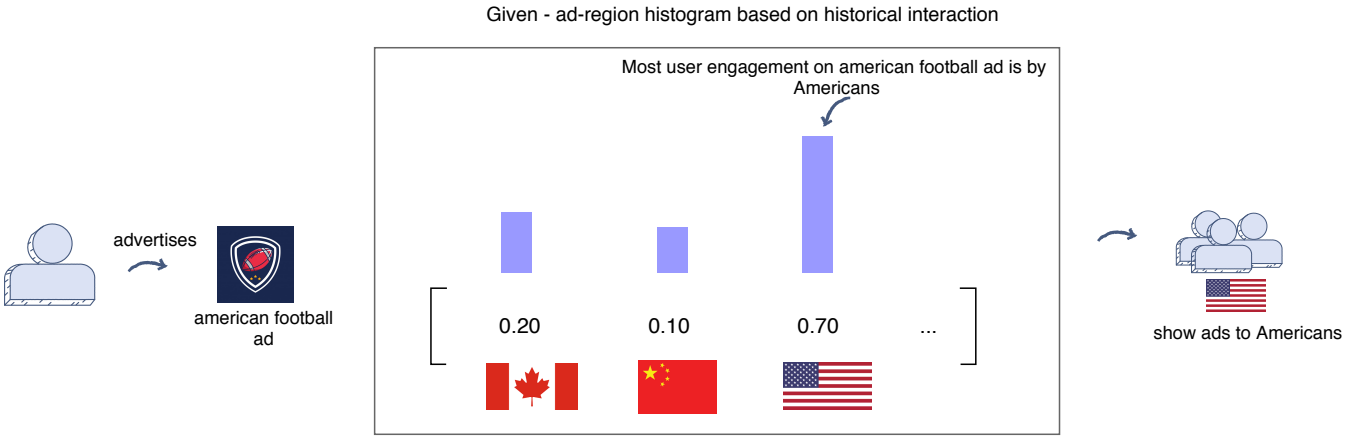
- **historical_engagement_rate**

  This feature specifies the ratio of user engagement with ads posted by a particular advertiser.

- **region_wise_engagement**

  The system should learn to show ads specific to a region from the histogram of engagement based on region. From an advertiser's perspective, the ad posted by an advertiser can be restricted to a specific region based on the ad content.

  For example, an American football ad posted by the advertiser will be most relevant to people living in the United States. This relevance can be predicted using the given histogram of engagement between the users and the ad.



Histogram to show user's engagement region-wise with the ad

User specific features

- **user_previous_search_terms**

This sparse feature specifies what users have searched in the past. This helps in recommending ads based on past user preferences.

- **user_search_terms**

  This sparse feature keeps a record of the user's search query terms.

- **age**

  This feature records the age of the user. It allows the model to learn the kind of ad that is appropriate according to different age groups.

- **gender**

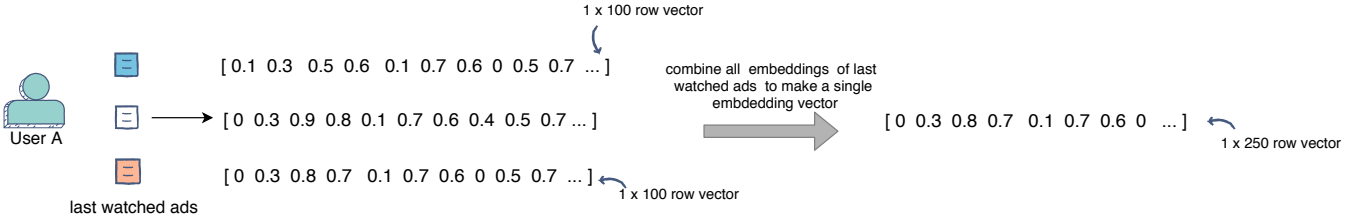
  The model learns about gender-based preferences.

- **language**

  This feature records the language of the user.

- **embedding_last_k_ads**

  The model will learn about the interest of the user using the history of the user's activity on the last k ads that were shown. We can make one embedding by combining embedding vectors of the last k ads that the user engaged with.



embedding of user's last watched ads

- **engagement_content_type**

  This takes into account the content of the ad that the user engages with. Here, we can track which type of ad a user plays around with, such as a video or an image ad.

- **engagement_days**

  This feature captures the user activity on each day of the week. For example, the user might be more active on weekends rather than on weekdays.

- **platform_time_spent**

This feature captures how long the user has been on the platform. This can be useful because we can show a different ad set to the user every hour to maximize their engagement with the ad.

- **region**

  This feature records the country of the user. Users from different geographical regions have different content preferences. This feature can help the model learn regional preferences and tune the recommendations accordingly.

Context specific features

- **current_region**

  This feature keeps track of the geographical location of the user. Note that the context may change if the user travels to other parts of the world. The system should learn the context of the user and show ads accordingly.

- **time**

  This feature will show ads subject to time of the day. The user would be able to see a different set of ads appear throughout the day.

- **device**

  It can be beneficial to observe the device a person is using to view content on. A potential observation could be that a user tends to watch content for shorter bursts on their mobile. However, they he usually chooses to watch on their laptop when they have more free time, so they watch for longer periods consecutively.

  - **screen_size:**

    The size of the screen is an important feature because if the users are using a device with a small screen size, there is a possibility that ads are actually never seen by the users. This is because they don't scroll far down enough to bring the ads in-view.

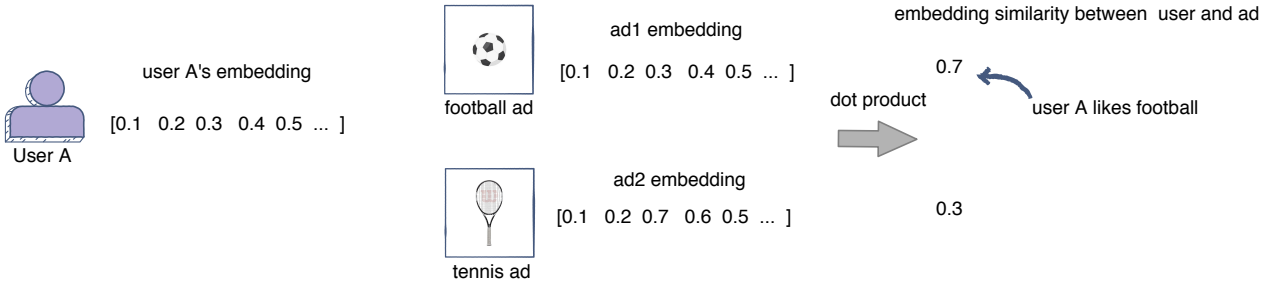
User-ad cross features

- **embedding_similarity**

  Here, we can generate vectors for the user's interest and the ad's content. We can generate embedding vectors for ads based on their content and for the user based on their interactions with the ad. A dot product between these vectors can be calculated to measure their similarity. A high score would equate to a highly relevant ad for the user. Please refer to our

embedding lesson (https://www.educative.io/collection/page/10370001/6237869033127936/6130870193750016) to see a few methods for generating these embeddings.

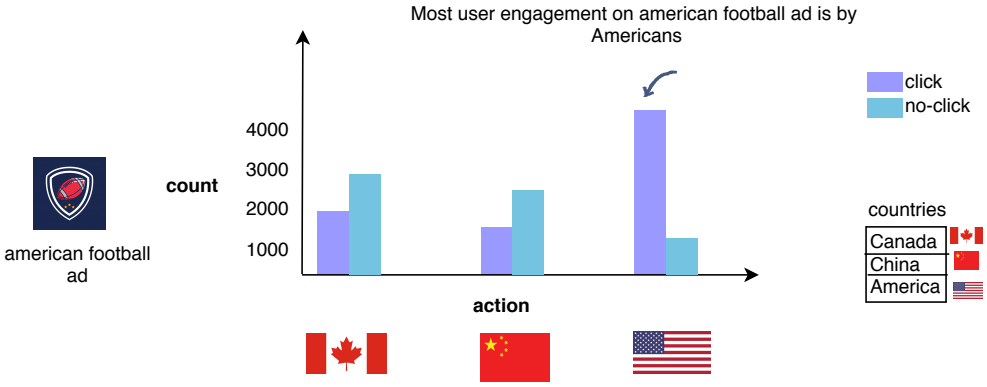
For example, the ad is about tennis and the user is not interested in tennis. Therefore, this feature will have a low similarity score.



User-ad embedding based similarity as a feature for the model

- **region_wise_engagement**

  An ad engagement radius can be another important feature. For example, the ad-user histogram below shows that an American Football ad is mostly viewed by people living in America.

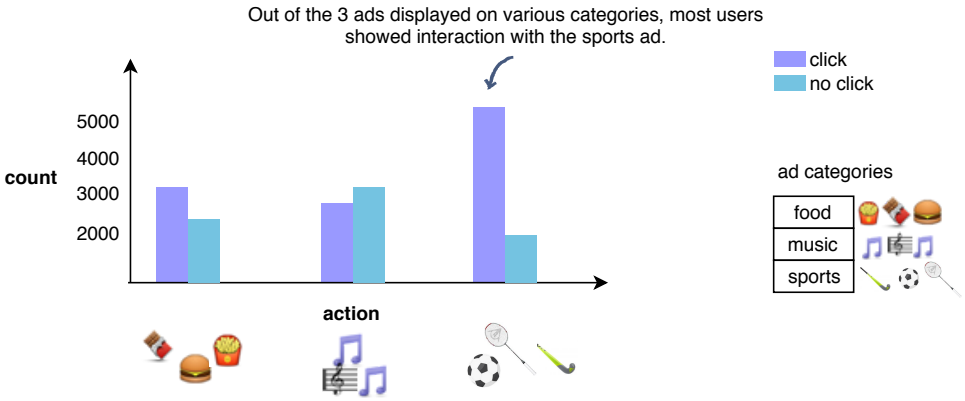


Histogram to show user's engagement region-wise with the ad

- **user_ad_category_histogram**

  This feature observes user engagement on an ad category using a histogram plot showing user engagement on an ad category.

  The following histogram shows that user engagement is the highest on the sports ad.

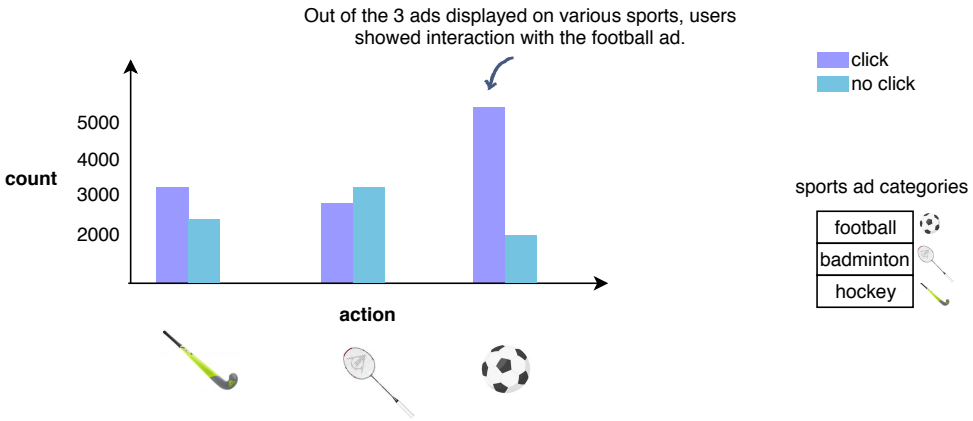


User-ad category histogram based on historical interaction

- **user_ad_subcategory_histogram**

  This feature observes user engagement on an ad subcategory using a histogram plot that shows user engagement on an ad subcategory.

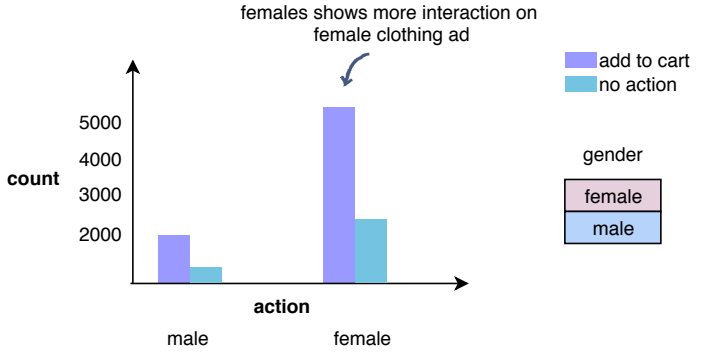
  The following histogram shows that user engagement is the highest on the football ad.



User-ad_sub category histogram based on historical interaction

- **user_gender_ad_histogram**

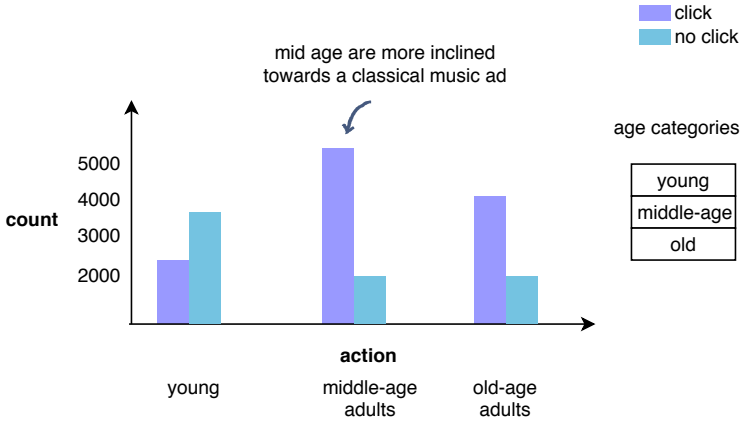
  Some ads can be more appealing to a specific gender. This similarity can be calculated by making a histogram plot showing user engagement gender-wise on an ad. For example, if the ad is for female clothing, it may be primarily of interest to women.



User_gender-ad histogram based on historical interaction

- **user_age_ad_histogram**

  An ad_age histogram can be used to predict user age-wise engagement.

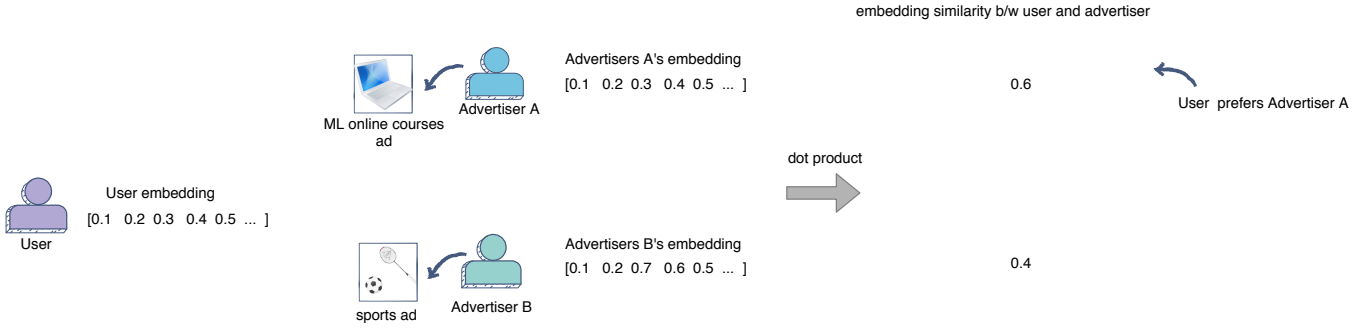


User-ad_age histogram based on historical interaction

User-advertiser cross features

- **embedding_similarity**

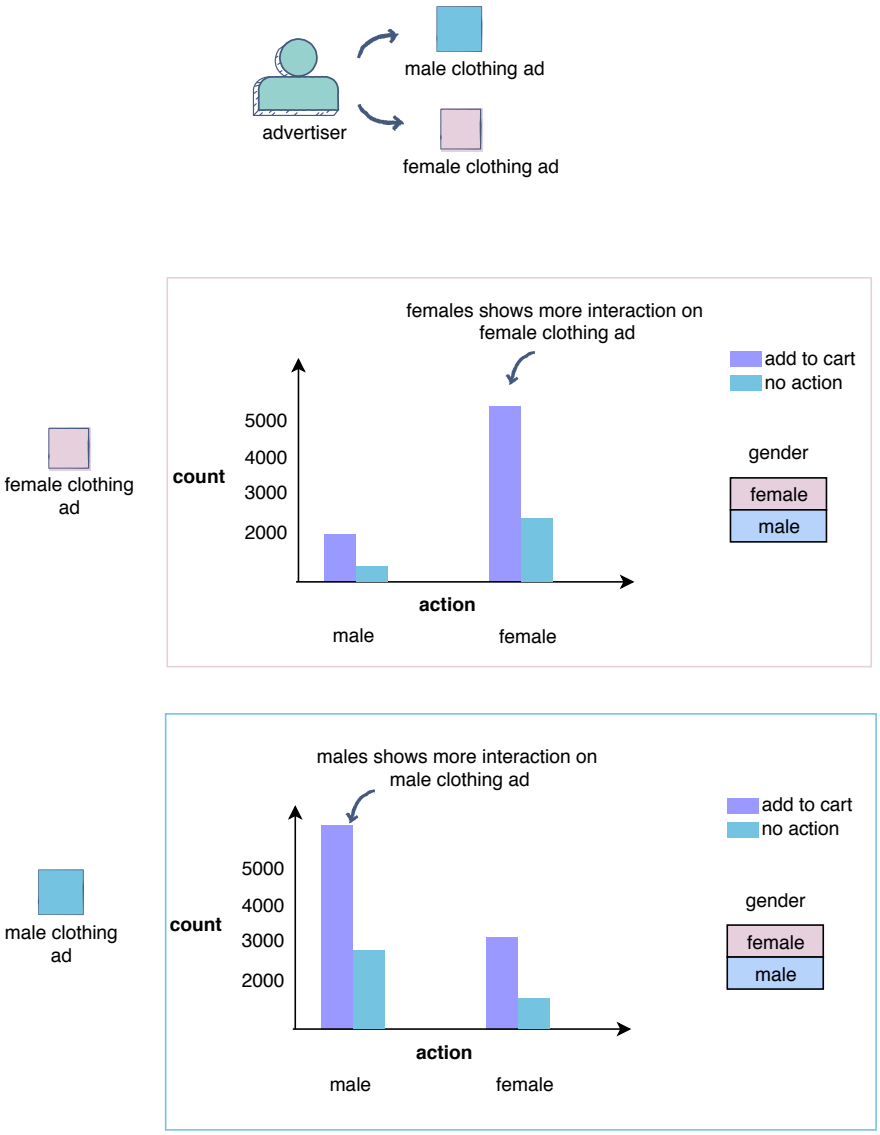
  We can project the advertisement and user in the same embedding space to see how close they are. From the embedding similarity score, we can figure out the type of ads the advertiser shows, and whether the user clicks on those types of ads.



User-advertiser embedding based similarity as a feature for the model

- **user_gender_advertiser_histogram**

  This feature observes gender-wise user engagement on an ad posted by an advertiser using a histogram plot. For example, the advertiser's ad for male clothing might be more engaging for men than women.

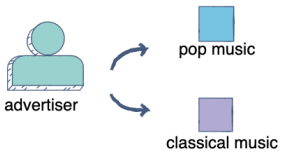


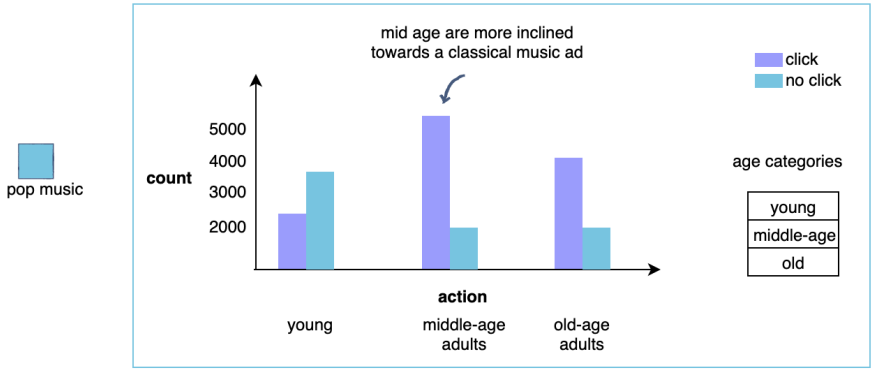
User_gender-advertiser histogram based on historical interaction

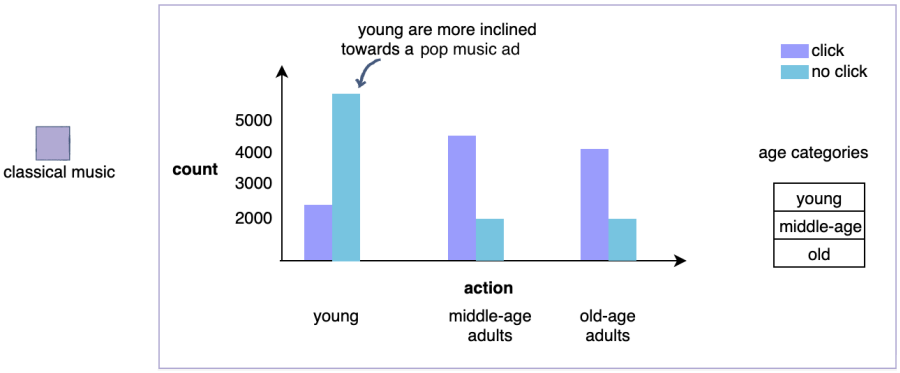
- **user_age_advertiser_histogram**

  This feature observes age-wise user engagement on an ad posted by an advertiser using a histogram plot.

  For example, a young audience might demonstrate a greater inclination towards the advertiser's ad on pop music.







User_age-advertiser histogram based on historical interaction

📝Key consideration for historical engagement features

There are two ways to pass historical data that is partitioned on a specific key (such as day, age, gender, etc.) as features to our models.

The **first approach** is to give the histogram data along with the key value to the model for learning. For example, in the case of daily historical engagement rate, we will give the model a histogram of seven-day user engagement from Sunday to Monday, which looks like [0.4, 0.2, 0.21, 0.25, 0.27, 0.38, 0.42]. In addition, we will pass the current day as a feature as well, e.g., if the day is Sunday the feature value of day will be 1, and for Saturday the feature value will be 7. So, the model will see eight features in total: seven more raw features for each day historical engagement rate and one feature for the current day of the week.

The **second approach** for giving this historical information to our model is to create a single feature that gives the current day engagement rate as a value, e.g., the feature value on Sunday will be 0.4, and on Friday it will be 0.38 for the above data.

It would be better to create features for learning by using both the first and second approaches. The models that are able to learn feature interactions effectively (such as neural networks or decision trees) can learn the current day engagement rate by the features described in the first approach. They might also be able to learn more interesting patterns such as understanding weekend data interactions, previous day interactions, etc. However, it's better to provide the full histogram data for the model to learn new interactions for us. But, for linear models providing the current day engagement data as we did in the second approach is critical because they can't learn these interactions on their own.

So, these are few ideas of features that can help our models predict the engagement rate better.

Back
Architectural Components

Next
Training Data Generation

Completed

69% completed, meet the criteria and claim your course certificate!

Report an Issue

Ask a Question (https://discuss.educative.io/tag/feature-engineering__ad-prediction-system__grokking-the-machine-learning-interview)

Training Data Generation

Let's learn about the techniques for generating training data for the ad prediction system.

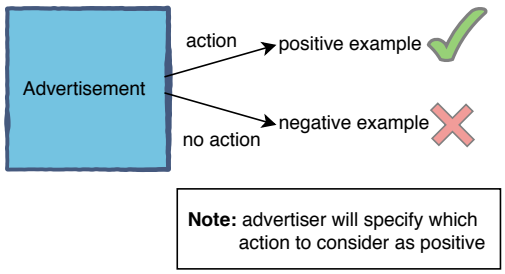
We'll cover the following

- Training data generation through online user engagement
- Balancing positive and negative training examples
- Model recalibration
- Train test split

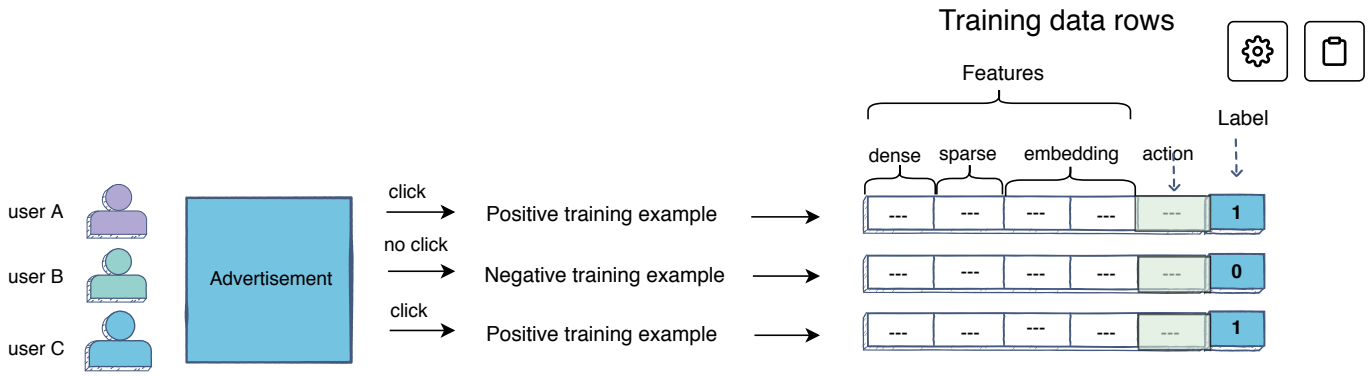
The performance of the user engagement prediction model will depend drastically on the quality and quantity of training data. So let's see how the training data for our model can be generated.

Training data generation through online user engagement

When we show an ad to the user, they can engage with it or ignore it. Positive examples result from users engaging with ads, e.g., clicking or adding an item to their cart. Negative examples result from users ignoring the ads or providing negative feedback on the ad.

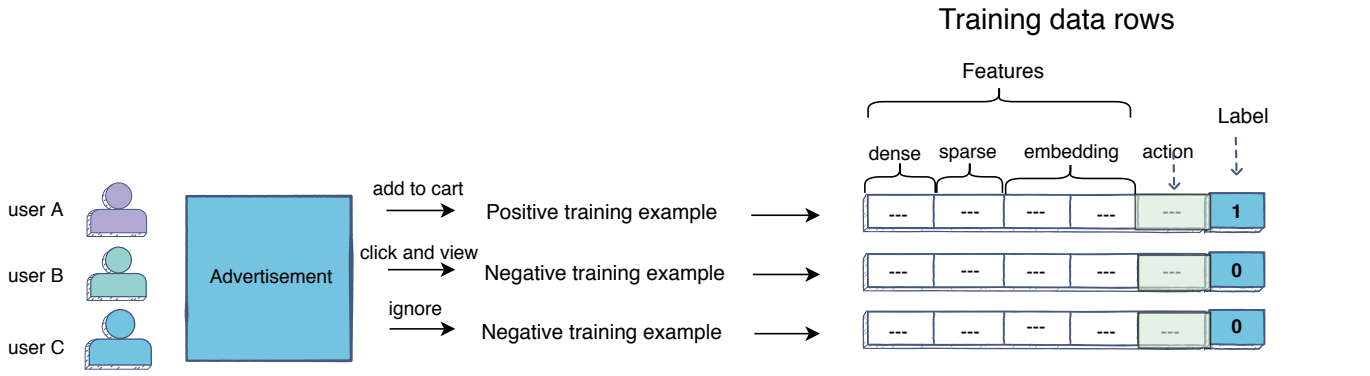


Suppose the advertiser specifies "click" to be counted as a positive action on the ad. In this scenario, a user-click on an ad is considered as a positive training example, and a user ignoring the ad is considered as a negative example.



Click counts as positive example

Suppose the ad refers to an online shopping platform and the advertiser specifies the action “add to cart” to be counted as positive user engagement. Here, if the user clicks to view the ad and does not add items to the cart, it is counted as a negative training example.



"Add to cart" counts as a positive training example.

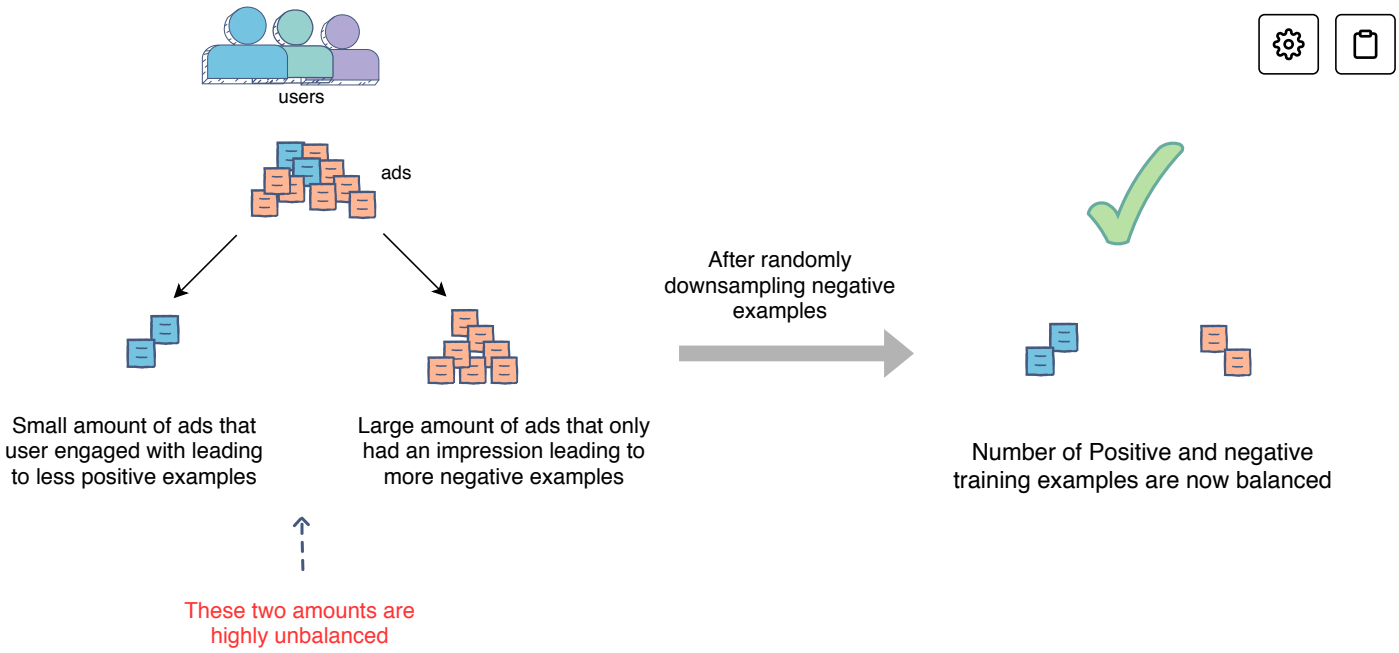
Balancing positive and negative training examples

Users’ engagement with an ad can be fairly low based on the platform e.g. in case of a feed system where people generally browse content and engage with minimal content, it can be as low as 2-3%.

How would this percentage affect the ratio of positive and negative examples on a larger scale?

Let’s look at an extreme example by assuming that one-hundred million ads are viewed collectively by the users in a day with a 2% engagement rate. This will result in roughly *two* million positive examples (where people engage with the ad) and 98 million negative examples (where people ignore the ad).

In order to balance the ratio of positive and negative training samples, we can **randomly down sample** the **negative examples** so that we have a similar number of positive and negative examples.



Model recalibration

Negative downsampling can accelerate training while enabling us to learn from both positive and negative examples. However, our predicted model output will now be in the downsampling space. For instance, consider that if our engagement rate is 5% and we select only 10% negative samples, our average predicted engagement rate will be near 50%. Auction uses this predicted rate to determine order and pricing; therefore it's critical that we recalibrate our score before sending them to auction. The recalibration can be done using:

$$q = \frac{p}{p + (1 - p)/w}$$

Here,

q is the re-calibrated prediction score,

p is the prediction in downsampling space, and

w is the negative downsampling rate.

Train test split

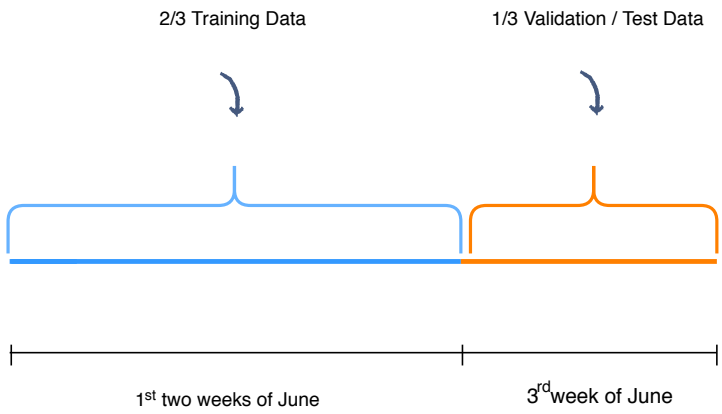
We need to be mindful of the fact that user engagement patterns may differ throughout the week. Hence we will use a week's engagement to capture all patterns during training data generation.

We may randomly select $\frac{2}{3}^{rd}$ or 66.6% training data rows that we have generated and utilize them for training purposes. The rest of the $\frac{1}{3}^{rd}$ or 33.3% can be used for validation and testing of the model.

However, this random splitting would result in utilizing future data for prediction given our data has a time dimension, i.e., we can utilize engagement on historical ads to predict future ad engagement. Hence we will train the model on data from the one-time interval and validate it on the data from its succeeding time interval. This will give a more accurate picture of how our model will perform in a real scenario.

> We are building models with the intent to forecast the user engagement rates on ads.

The following illustration how that we train the model using the engagement data generated in the first two weeks and validate/test it with the data generated in the third week.



Splitting data for training, validation, and testing

Back
Feature Engineering

Next
Ad Selection

Completed

Issue (https://discuss.educative.io/tag/training-data-generation__ad-prediction-system__grokking-the-machine-learning-interview)

Ad Selection

Let's see what role the ad selection component plays in the overall prediction system.

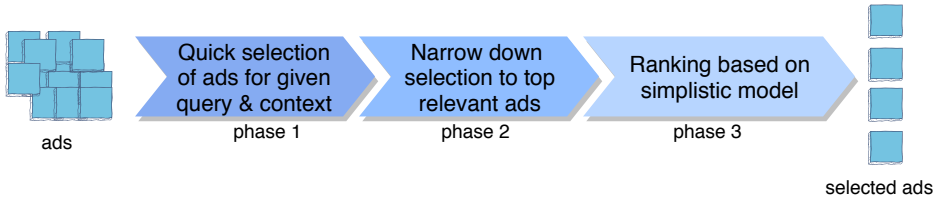
We'll cover the following

- Phase 1: Selection of ads
- Phase 2: Narrow down selection set to top relevant ads
- Phase 3: Ranking of selected ads using a simplistic model
- How does the system scale?

The main goal of the ads selection component is to narrow down the set of ads that are relevant for a given query. In a search-based system, the ads selection component is responsible for retrieving the top relevant ads from the ads database (built using all the active ads in the system) according to the user and query context. In a feed-based system, the ads selection component will select the top k relevant ads based more on user interests than search terms.

Based on our discussions about the funnel-based approach for modeling, it would make sense to structure the ad selection process in the following three phases:

- **Phase 1:** Quick selection of ads for the given query and user context according to selection criteria
- **Phase 2:** Rank these selected ads based on a simple and fast algorithm to trim ads.
- **Phase 3:** Apply the machine learning model on the trimmed ads to select the top ones.



Phases in ML systems

Phase 1: Selection of ads

Advertising platforms can have hundreds of millions of active ads. Our main motivation in this phase is to quickly reduce this space from millions of ads to one ads that can be shown to the current user in the given context (e.g. for a user searching "machine learning" on a mobile device).

The first key requirement to be able to achieve the quick selection objective is to have the ads stored in a system that is fast and enables complex selection criteria. This is where building an in-memory index to store the ads will be massively helpful. Index allows us to fetch ads quickly based on different targeting and relevance information from the user. This is similar to our Search system discussion, where we had to similarly select documents quickly at the selection time to narrow our focus to relevant documents. We will index ads on all possible indexing terms that can be used for selection e.g. targeted terms, city, state, country, region, age etc.

Let's take an example of a search ads system to explain this concept further. Let's say that a male user, aged twenty-five, and located in San Francisco, California is searching stuff related to machine learning. He types in a query "machine learning". In this case, we would want to select all potential ads that we can show to the user, i.e., the ads targeting criteria matches the user's profile.

The selection query in this case will look like:

```
(term = "machine learning")
                and
(age = "\*" or age contains "25")
                and
(gender="\*" or gender="male")
                and
(city = "\*" or city = "San Fracisco")
                and
(state="\*" or state="CA")
                and
(status="Active")
                and
(has_budget = true)
```

Selection query for search system

In a feed-based system, the selection of ads will base more on user interests rather than search terms. Let's assume that the same user is interested in Computer science and football, and we want to now fetch ads for his feed. The selection query will look like the following:

```
(interest = "Computer Science" or interest = "Football" or interest = "*")
                                    and
                        (age = "\*" or age contains "25")
                                    and
                        (gender="\*" or gender="male")
                                    and
                    (city = "\*" or city = "San Fracisco")
                                    and
                        (state="\*" or state="CA")
                                    and
                            (status="Active")
                                    and
                            (has_budget = true)
```

Selection query for feed based system

So, the above selection criteria in phase 1 will reduce our space set from all active ads to **ads that are targeted for the current user**.

Phase 2: Narrow down selection set to top relevant ads

The number of ads selected in phase 1 can be quite large depending on the query and the user, e.g., we can have millions of ads targeted for a sports fan. So, running a complex ML model to predict engagement on all these selected ads will be slow and expensive. At this point, it makes sense to narrow down the space using a simplistic approach before we start running complex models.

The eventual ranking of ads is going to be based on (bid * predicted score). We already know the bid at this point and can use the prior engagement score (CPE - cost per engagement) based on the ad, advertiser, ad type, etc. as our predicted score.

$$(\text{bid}) * (\text{prior CPE score})$$

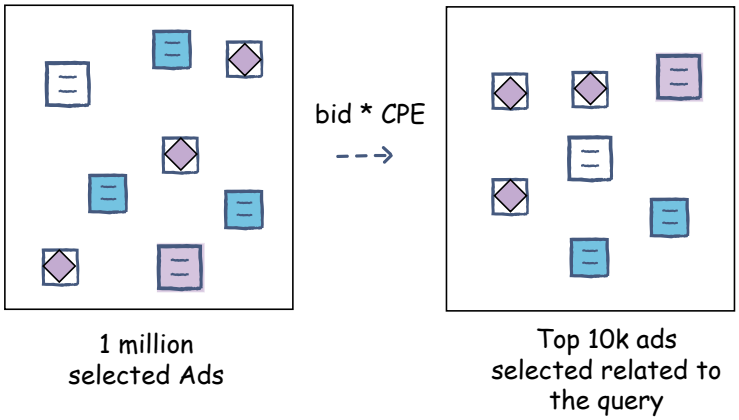
For a new ad or advertiser where the system doesn't have good prior scores, a slightly higher score can be given to ads also called *score boost*. We can use time decay to reduce it. The ranking might look like the following with a new ad boost of 10% to CPE score for the first forty-eight hours of ad with time decay.

```
boost = 0;

if ad_age < 48:
    boost = 0.1 * (1 - ad_age / 48)
    ad_score = (bid) * (1 + boost) * CPE
```

Based on these prior scores, we can narrow down ads from our large selected set to the top k ads. In phase 3, we will use a much stronger predictor than prior scores on the top selected ads from phase 2.

The following diagram shows an example configuration where the selection expression retrieves one million ads from the database. We then select the top ten-thousand ads based on $bid * CPE$ score.



Phase 1 on left with one million ads selected. Phase 2 on the right using bid * prior CPM to narrow it down to top ten-thousand ads

Phase 3: Ranking of selected ads using a simplistic model

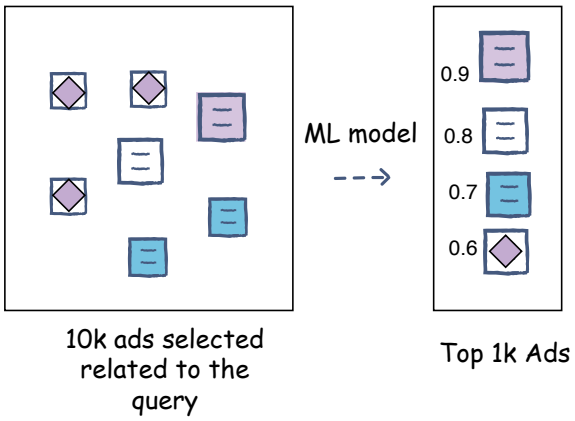
As the ranking in phase 2 is super simplistic, we should still select a sizable ads(e.g., ten thousand) by running a better model as we work towards reducing the ad set. We can run a simplistic and efficient model on ads selected in phase 2 to further narrow it down. The top ads from this phase will be passed to our ad prediction stage to run more complex and accurate models for better user engagement prediction.

We can use either **logistic regression** or **additive trees** based models (such as random forest or boosted decision trees) as they are quite efficient. **Neural network**-based models need more capacity and time so they might not be the best choice at this stage.

The target is to select the top k relevant ads from the set given by phase 2. We will use training data and dense features, which were discussed in previous sections to train this model to predict engagement scores better and minimize our log loss error.

At evaluation time, we will get a new predicted engagement score for ranking of ads. Ads will be sorted based on this prediction CPE, which is the ($bid * CPE$) score, as we did in phase 2. However, our predicted score should be a far better prediction than using historical priors as we did in phase 2.

The following figure shows an example configuration where the top ten-thousand ads from phase 2 are ranked by our ML model to select the top one-thousand ads in phase 3.



Ranking of ads in phase 3 from an efficient ML model of results coming from phase 2

📝 As discussed earlier in the architectural components lessons, the ad set in each phase is narrowed down for the phase below it, thus making it a funnel-based approach. As we progress down the funnel, the complexity of the models increases, and the number of results decrease. Within the ad selection component, we adopt the same funnel based approach. First, we select ads in phase 1, then rank them based on prior scores in phase 2. Finally, we rank them using an efficient model in phase 3 to come up with the top ads that we want to send to the ad prediction stage.

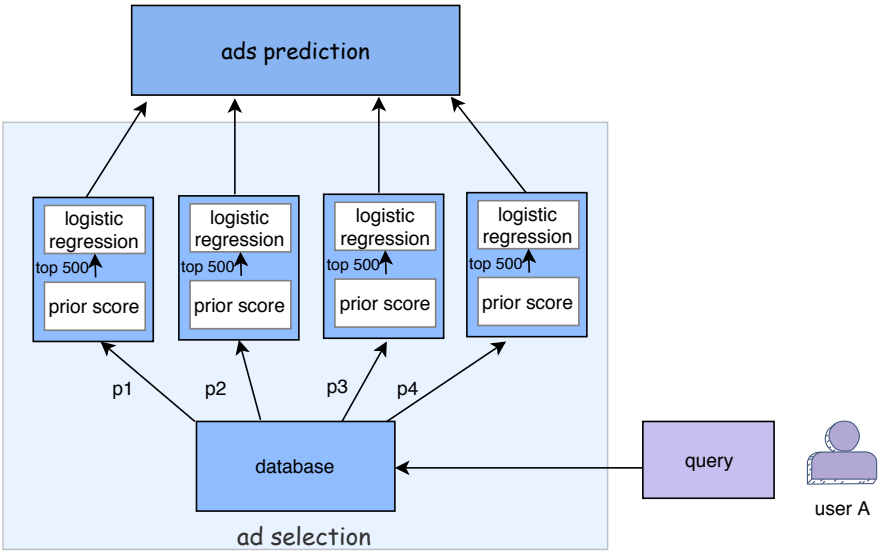
How does the system scale?

Note that our index is sharded, i.e., it runs on multiple machines and every machine selects the top k ads based on the prior score. Each machine then runs a simplistic **logistic regression** model built on dense features (there are no sparse features to keep the model size small) to rank ads.

📝 The number of partitions (shards) depends on the size of the index. A large index results in more partitions as compared to a smaller index. Also, the system load measured in queries per second (QPS) decides how many times the partition is replicated.

Consider a scenario where the size of the index is 1 tera-byte and memory of a single shard is 250 giga-bytes. The index data gets distributed in four partitions. Each partition selects ads that satisfy the selection critera. Then, we use the prior score of the selected ads and select top five-hundred ads based on that score. To further narrow down the ad set, we run logistic regression which returns the top 50 ads. The top fifty ranked ads from each of the four partitions (200 ads in total) are fed to the ad prediction component.

Note that for each level we are selecting top “k” ads. The selection of the number “k” for each level is very arbitrary. It is based on experimentation and system available load/capacity.



Sharded index with four partitions. Each partition select the top five-hundred ads based on a prior score and then run logistic regression to select top fifty ads . The top fifty ads from each partition are fed to ad prediction component

The ad set returned from the ad selection component is further ranked by the ad prediction component.

Back
Training Data Generation

Next
Ad Prediction

Completed

Issue (https://discuss.educative.io/tag/ad-selection__ad-prediction-system__grokking-the-machine-learning-interview)

Ad Prediction

Let's have a look at how the most relevant ads will be predicted from a set of selected ads.

We'll cover the following

- Modeling approach
- Model for online learning
  - Auto non-linear feature generation

The ad prediction component has to make predictions for the final set of candidate selected ads. It needs to be robust and adaptive and should be able to learn from massive data volume.

Let’s go over the best setup and models for this problem.

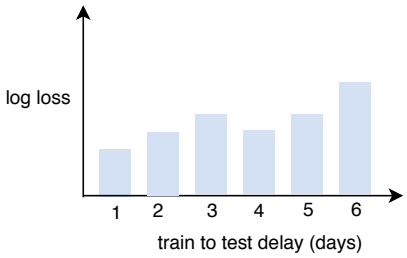
Modeling approach

Ads are generally short-lived. So, our predictive model is going to be deployed in a dynamic environment where the ad set is continuously changing over time.

Given this change in an ad set, keeping the model up to date on the latest ads is important. In other words, model performance will degrade with each passing day if it isn’t refreshed frequently.

Online learning

If we have to plot the log loss of the model, it might look like the graph on the right. Here we are assuming that the model is trained on day one and we are observing log loss on six consecutive days. We can observe the degradation of prediction accuracy (increase of log loss) because of the increased delay between the model training and test set.



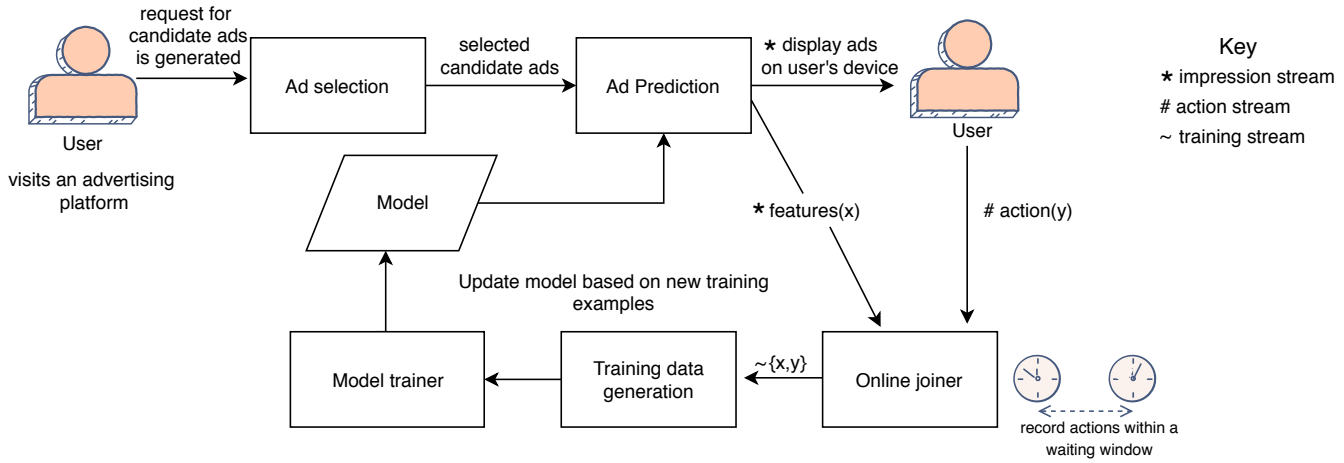
So, keeping models updated with the latest data is important for ad prediction.

One approach to minimize this loss could be refreshing the model more frequently e.g. training it every day on new data accumulated since the previous day.

However, a more practical and efficient approach would be to keep refreshing the model with the latest impressions and engagements after regular internals (e.g. 15 mins, 30 mins, etc.). This is generally referred to as **online learning** or **active learning**.

In this approach, we train a base model and keep adding new examples on top of it with every ad impression and engagement. From an infrastructure perspective, we now need a mechanism that collects the recent ad data and merges it with the current model.

The following figure shows some key components that are critical for enabling online learning. We need a mechanism that generates the latest training examples via an online joiner. Our training data generator will take these examples and generate the right feature set for them. The model trainer will then receive these new examples to refresh the model using stochastic gradient descent. This forms a tightly closed loop where changes in the feature distribution or model output can be detected, learned on, and improved in short successions. Note that the refresh of the model doesn't have to be instantaneous, and we can do it in same batches at a certain frequency, e.g., every 30 mins, 60 mins etc.



Flow of events for ad prediction system

Model for online learning

One model that easily supports online learning and has the ability to update it using stochastic gradient descent using mini-batches is **logistic regression**.

Auto non-linear feature generation

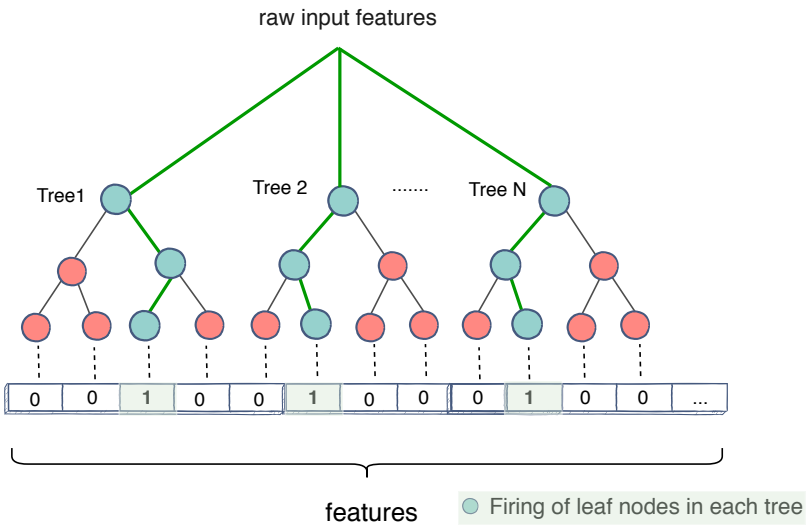
One potential drawback is that simple logistic regression (generalized linear model) relies on manual effort to create complex feature crosses and generating non-linear features. Manually creating such features is cumbersome and will mostly be restricted to modeling the relationship between two or three features. On the other hand, tree and neural network-based models are really good at generating complex relationships among features when optimizing the model.

To overcome this, we can use additive trees and neural networks to find such complex feature crosses and non-linear feature relationships in data, and then these features are input to our logistic regression model.

Additive trees

To capture complex feature crosses and non-linear relationships from the given raw features, additive trees can be extremely handy. We can train a model to predict the probability of engagement using trees and minimize our loss function.

We can represent an additive tree to generate features by taking into account triggered leaf nodes, e.g., let's say that we have one-hundred trees in our ensemble with four leaf nodes each. This will result in 100 * 4 = 400 leaf nodes in which one-hundred nodes will trigger for every example. This binary vector itself captures the prediction (or learning) of our tree ensemble. We can just feed this vector as one input feature to our logistic regression model.

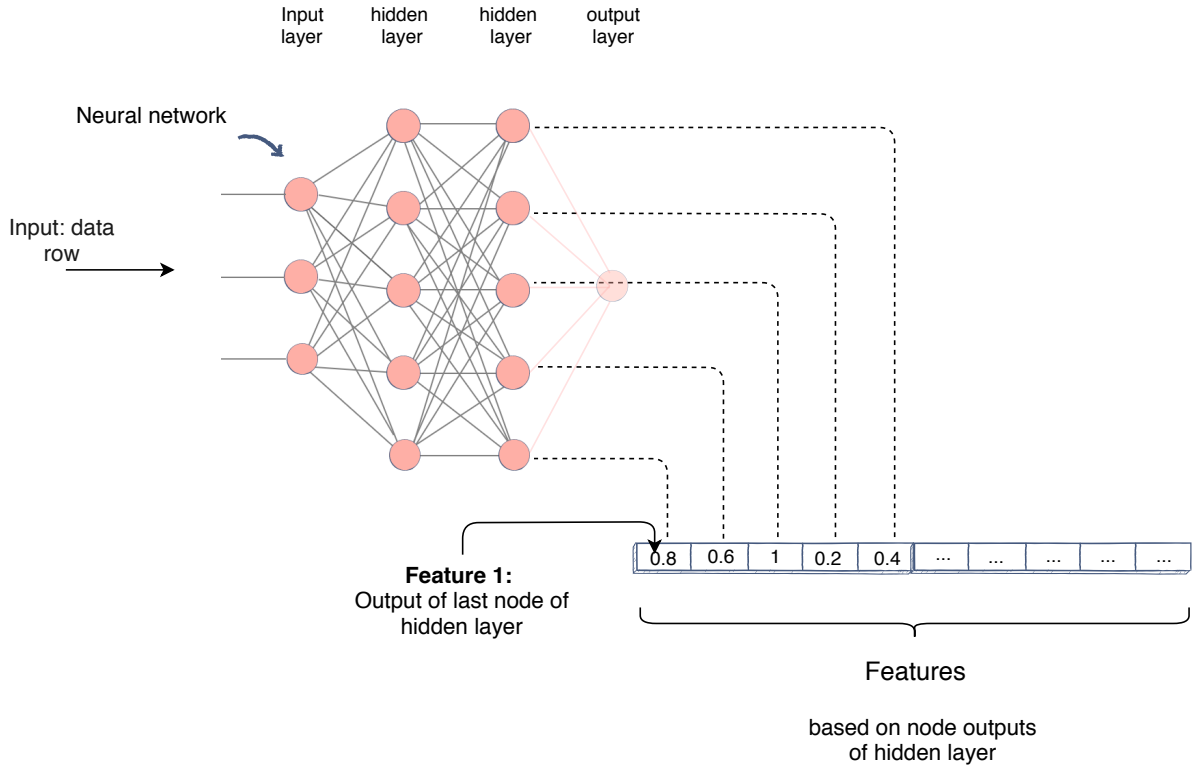


Features from additive trees

Neural Network

A deep neural network can be trained on the raw input features to predict our ads engagement to generate features for our logistic regression model. As discussed in the additives trees section, neural network-based models are also really good at capturing non-linear, complex relationships between features.

The following examples show a network trained with two hidden layers to predict our engagement rate. Once trained, we can use the activation from the last hidden layer to feed in as input feature vector to our logistic regression model.



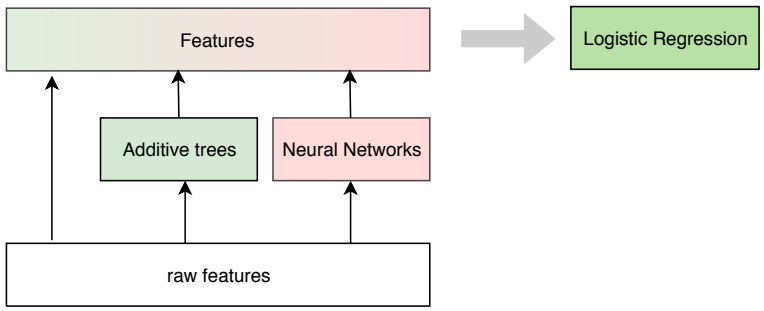
Last hidden layer of neural networks provides features for the logistic regresssion model

Features from neural network

So, let's combine the above two ideas together:

- We train additive trees and neural network to predict non-linear complex relationships among our features. We then use these models to generate features.
- We use raw features and features generated in the previous step to train a logic regression model.

The above two steps together solve both of our problems to capture complex non-linear relationships and also enable us to use online learning to refresh the model frequently for ads that are fast-changing and dynamic in nature.



Training a logistic regression model on the generated features from Trees and Neural Network

This concludes our course on machine learning system design! Hopefully, you are now confident in your ability to answer machine learning design interview questions in an effective and systematic way.

✔ Great! You've completed this lesson.

Not Yet

← Back

Ad Selection

69% completed, meet the criteria and claim your course certificate!

Report an Issue

Ask a Question (https://discuss.educative.io/tag/ad-prediction__ad-prediction-system__grokking-the-machine-learning-interview)