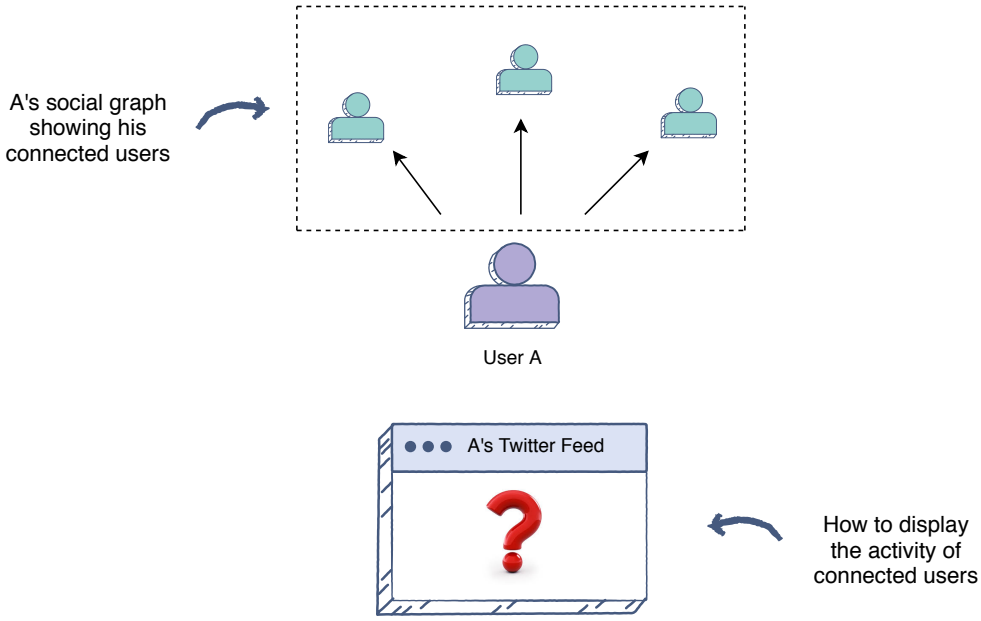
Problem Statement

Let's look at a problem statement that asks you to design a Twitter feed system.

We'll cover the following

Problem statement

The interviewer has asked you to design Twitter feed system that will show the most relevant tweets for a user based on their social graph.



How to display the most relevant content for user A's Twitter feed?

First, let’s develop an understanding of the problem.

Visualizing the problem

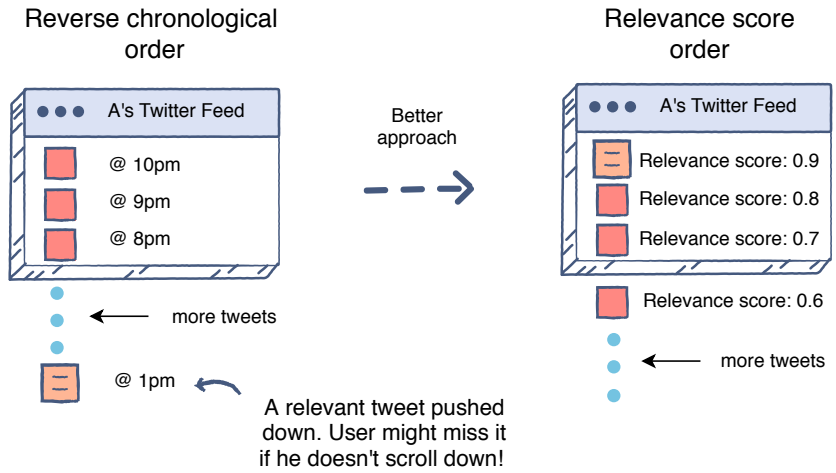
User A is connected to other people/businesses on the Twitter platform. They are interested in knowing the activity of their connections through their feed.

In the past, a rather simplistic approach has been followed for this purpose. All the Tweets generated by their followees since user A's last visit were displayed in **reverse chronological order**.

However, this *reverse-chronological order feed display* often resulted in user A missing out on some Tweets that they would have otherwise found very engaging. Let's see how this happens.

Twitter experiences a large number of daily active users, and as a result, the amount of data generated on Twitter is torrential. Therefore, a potentially engaging Tweet may have gotten pushed further down in the feed because a lot of other Tweets were posted after it.

Hence, to provide a more engaging user experience, it is crucial to rank the most relevant Tweets above the other ones based on user interests and social connections.



Text

Transition from time-based ordering to relevance-based ordering of Twitter feed

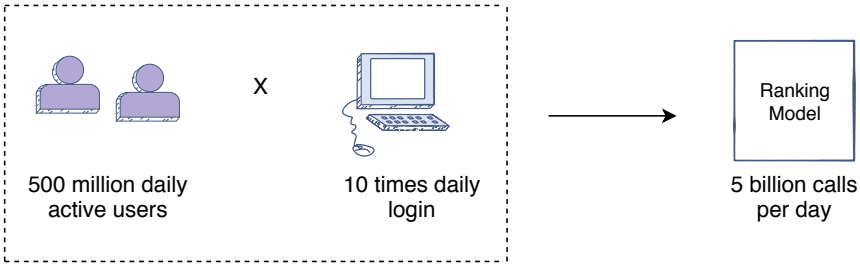
The feed can be improved by displaying activity based on its relevance for the logged-in user. Therefore, the feed order is now based on **relevance ranking**.

Scale of the problem

Now that you know the problem at hand, let's define the scope of the problem:

1. Consider that there are five-hundred million daily active users.
2. On average, every user is connected to one-hundred users.
3. Every user fetches their feed ten times in a day.

Five-hundred million daily active users, each fetching their feed ten times daily, means that your **Tweet ranking system will run five billion times per day**.



The ranking model may receive as many as five billion calls daily

Finally, let's set up the machine learning problem:

"Given a list of tweets, train an ML model that predicts the probability of engagement of tweets and orders them based on that score"

Back
Filtering Results

Next
Metrics

Completed

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

Report an Issue

Ask a Question (https://discuss.educative.io/tag/problem-statement__feed-based-system__grokking-the-machine-learning-interview)

Metrics

Learn about the metrics to capture the success of a feed-ranking system.

We'll cover the following

The feed-ranking system aims to maximize user engagement. So, let's start by looking at all the user actions on a Twitter feed.

User actions

The following are some of the actions that the user will perform on their tweet, categorized as positive and negative actions.

Positive user actions

- Time spent viewing the tweet
- Liking a Tweet
- Retweeting
- Commenting on a Tweet

Negative user actions

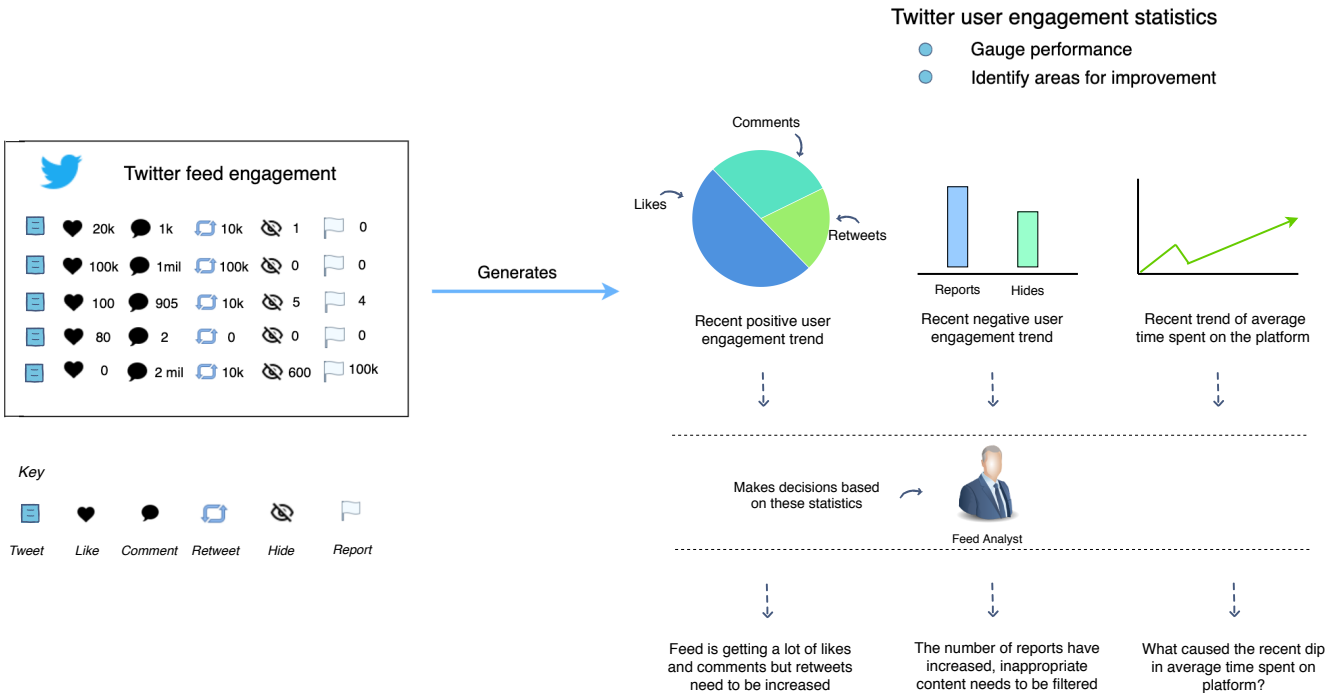
- Hiding a Tweet
- Reporting Tweets as inappropriate

Now, you need to look at these different forms of engagements on the feed to see if your feed ranking system did a good job.

> 📝 All of the metrics we are going to discuss can be used to determine the user engagement with the feed generated by our feed ranking system.

User engagement metrics

The following illustration shows how *feed engagement data* generates useful statistics for gauging user engagement.



User actions on the platform can help generate useful statistics

The above illustration shows different positive and negative engagements on a Twitter feed. Let's see which of these engagements would be a good one to target as your *overall system metric* to optimize for.

Selecting feed optimization metric

An important thing to understand in selecting a topline is that it's scientific as well as a business-driven decision.

The business might want to focus on one aspect of user engagement. For instance, Twitter can decide that the Twitter community needs to engage more actively in a dialogue. So, the topline metric would be to focus more on the **number of comments** on the Tweets. If the average number of comments per user increases over time, it means that the feed system is helping the business objective.

Similarly, Twitter might want to shift its focus to overall engagement. Then their objective will be to increase average overall engagement, i.e., **comments, likes, and retweets**. Alternatively, the business may require to optimize for the time spent on the application. In this case, **time spent on Twitter** will be the feed system metric.

Negative engagement or counter metric

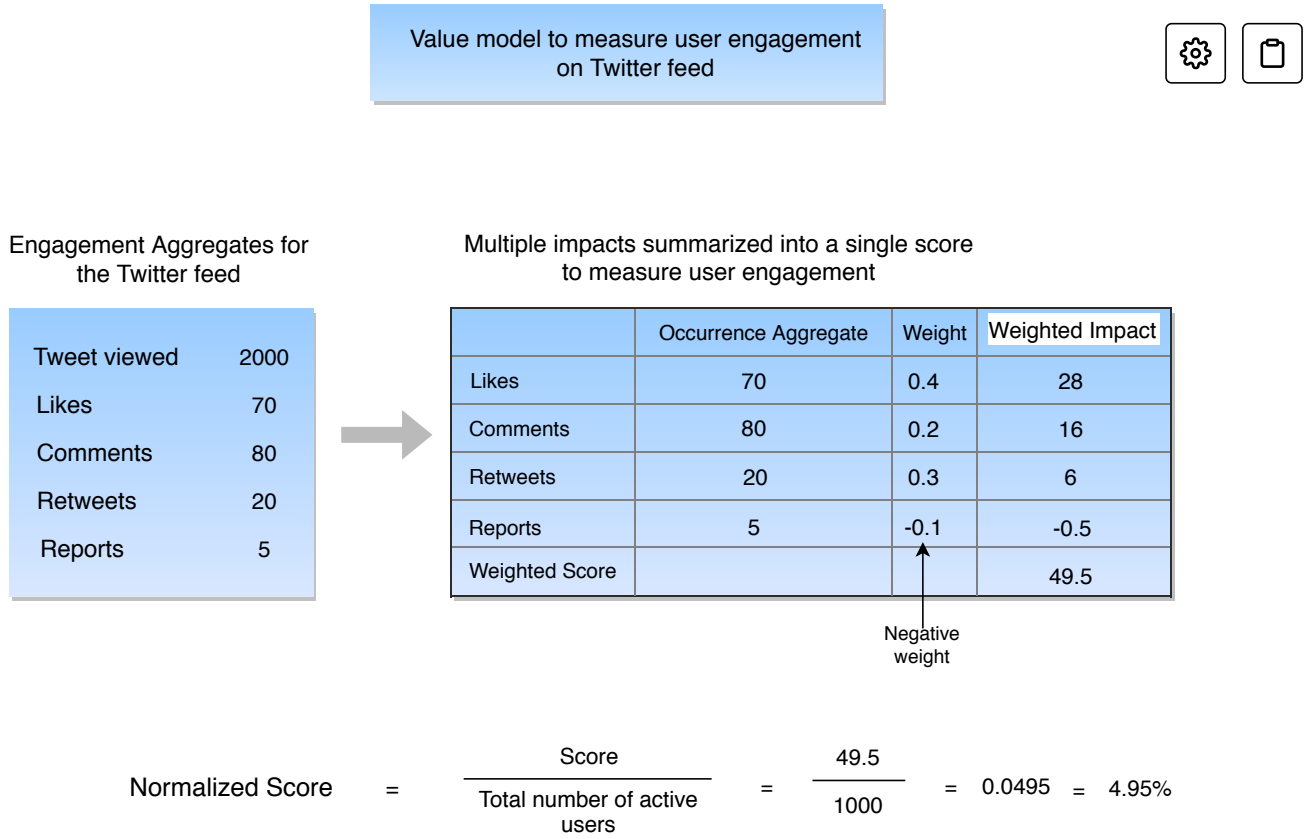
For any system, it's super important to think about counter metrics along with the key, topline ones. In a feed system, users may perform multiple negative actions such as reporting a Tweet as inappropriate, block a user, hide a Tweet, etc. Keeping track of these negative actions and having a metric such as **average negative action** per user is also crucial to measure and track.

Weighted engagement

More often than not, all engagement actions are equally important. However, some might become more important at a particular point in time, based on changing business objectives. For example, to have an engaged audience, the number of comments might be more critical rather than just likes. As a result, we might want to have different weights for each action and then track the overall engagement progress based on that weighted sum. So, the metric would become a **weighted combination of these user actions**. The weighted combination metric can be thought of as a value model. It will summarize multiple impacts (of different forms of user engagements) into a single score. Let's see how this works.

In the following illustration, we are going to use the *weighted combination metric* to measure/score user engagement. Each user action will be assigned a weight according to the impact it should have towards the final score.

> 📝 These weights are assigned, keeping in mind the respective importance of different forms of engagement towards the business objectives.



Using the weighted metric to measure the performance of the feed, i.e., it's user engagement

The user engagements are aggregated across all users' feeds over a specific period of time. In the above diagram, two-thousand tweets were viewed in a day on Twitter. There were a total of seventy likes, eighty comments, twenty retweets, and five reports. The weighted impact of each of these user engagements is calculated by multiplying their occurrence aggregate by their weights. In this instance, Twitter is focusing on increasing "likes" the most. Therefore, "likes" have the highest weight. Note that the negative user action, i.e., "report", has a negative weight to cast its negative impact on the score.

The weighted impacts are then summed up to determine the score. The final step is to normalize the score with the total number of active users. This way, you obtain the engagement per active user, making the score comparable. To explain the importance of normalization, consider the following scenario.

The score calculated above is referred to as score A, in which we considered one-thousand active users. We now calculate the score over a different period where there are only five-hundred active users, referred to as score B. Assume that score B comes out to be less than score A. Now, score A and score B are not comparable. The reason is that the decrease in score B may just be the effect of less active users (i.e., five-hundred active users instead of one-thousand active users).

When it comes to interpretation, a higher score equates to higher user engagement.

The weights can be tweaked to find the desired balance of activity on the platform. For example, if we want to increase the focus on commenting, we can increase its weight. This would put us on the right track, i.e., we would be showing Tweets that get more comments. This will lead to an increase in the overall score, indicating good performance.

Back

Problem Statement

Next

Architectural Components

Completed

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

Report an Issue

Ask a Question

(https://discuss.educative.io/tag/metrics__feed-based-system__grokking-the-machine-learning-interview)

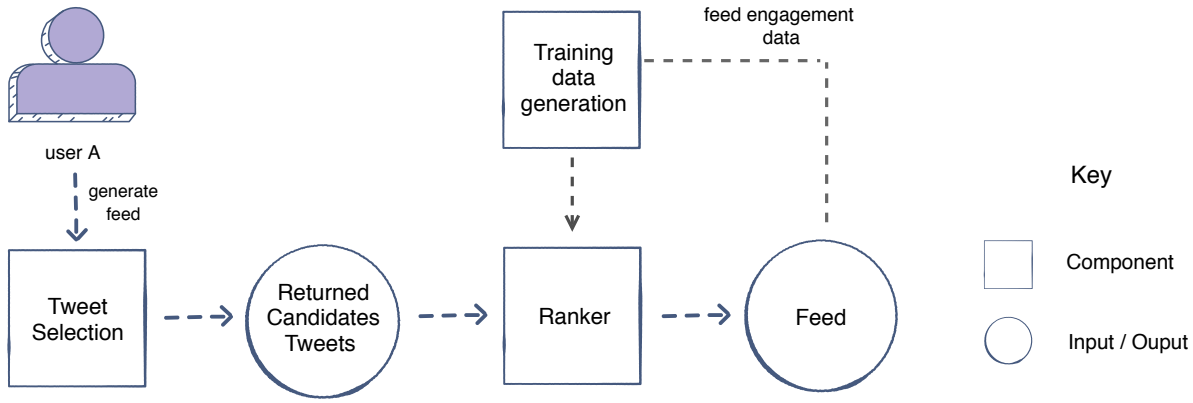
Architectural Components

Have a look at the architectural components of the feed ranking system.

We'll cover the following

Architecture

Let’s have a look at the architectural components that are integral in creating our Twitter feed system.



Architectural diagram of feed based system

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

Tweet selection

This component performs the first step in generating a user’s feed, i.e., it fetches a pool of Tweets from the user’s network (the followees), since their last login. This pool of Tweets is then forwarded to the ranker component.

Training data generation

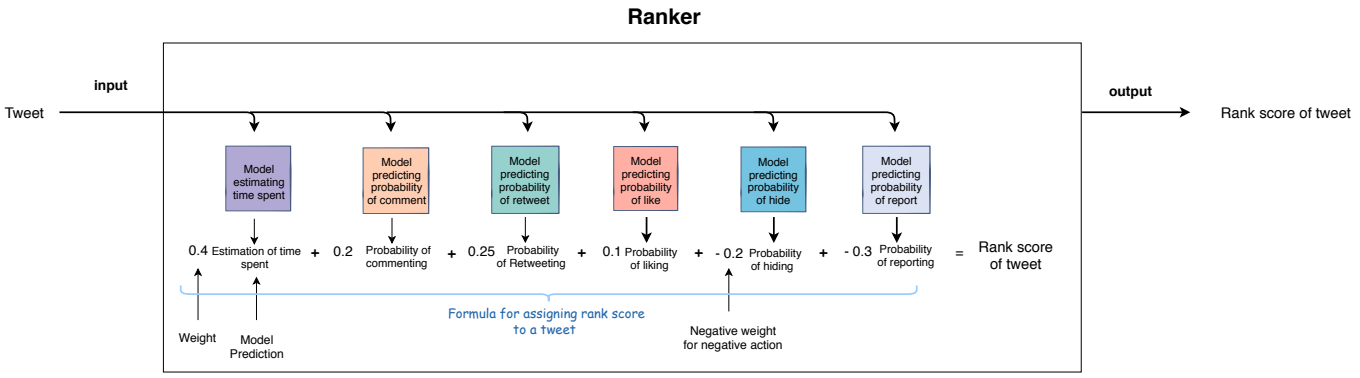
Each user engagement action on the Twitter feed will generate positive and negative training examples for the user engagement prediction models.

Ranker

The ranker component will receive the pool of selected Tweets and predict their probability of engagement. The Tweets will then be ranked according to their predicted engagement probabilities for display on user A's feed. If we zoom into this component, we can:

1. Train a single model to predict the overall engagement on the tweet.
2. Train *separate models*. Each model can *focus on predicting the occurrence probability of a certain user action for the tweet*. There will be a separate predictor for *like*, *comment*, *time spent*, *share*, *hide*, and *report*. The results of these models can be merged, each having a different weight/importance, to generate a rank score. The Tweets will then be ranked according to this score.

   The following illustration gives an idea of how the ranker component combines the outputs of these models to come up with a rank score for a Tweet.



Combining the output of different predictors with weights according to importance of user action

Separately predicting each *user action* allows us to have greater control over the importance we want to give to each action when calculating the rank of the Tweet. We can tweak the weights to display Tweets in such a manner that would align with our current business objectives, i.e., give certain user actions higher/lower weight according to the business needs.

Back

Metrics

Next

Tweet Selection

Completed

Tweet Selection

Let's see how the Tweet selection component fetches Tweets to display on a user's Twitter feed.

We'll cover the following

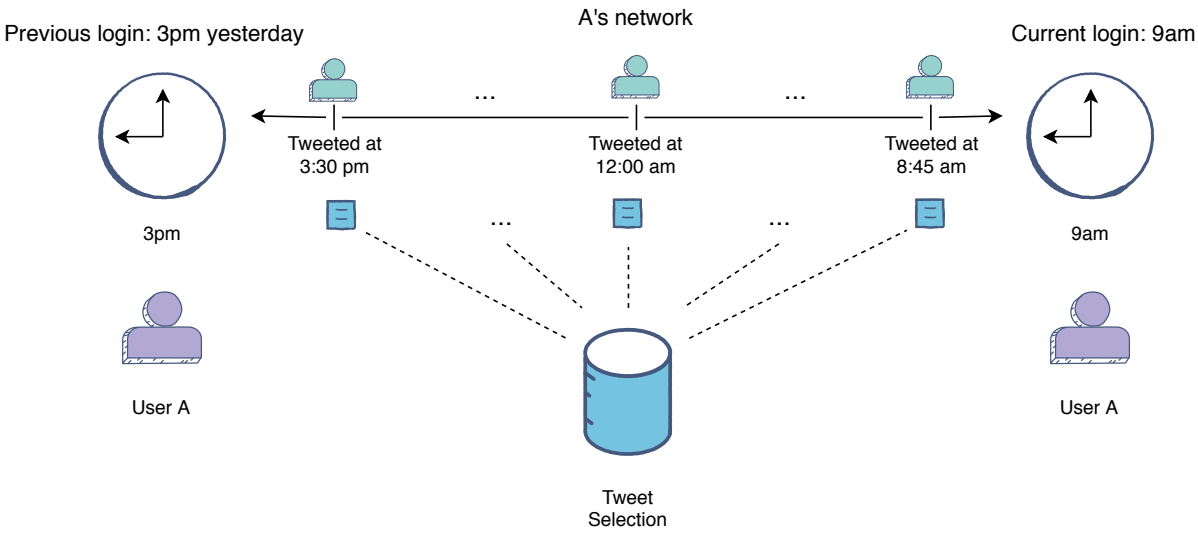
Tweet selection schemes

Let’s see the various schemes Tweet selection component uses to fetch Tweets for a user’s twitter feed.

New Tweets

Consider the following scenario to see how Tweet selection occurs.

User A logs in to Twitter at 9 am to view their Twitter feed. Now, the Tweet selection component has to fetch Tweets for display. It fetches the five-hundred newly generated Tweets by A’s network since the last login at 3 pm yesterday.



Select new Tweets generated by A's network between 3pm to 9 am

> 📝 *A not-so-new Tweet!*
>
> Consider a Tweet that user A has viewed previously. However, by the time the user logs in again, this Tweet has received a much bigger engagement and/or A's network has interacted with it. In this case, it once again becomes relevant to the user due to its recent engagements. Now, even though this Tweet isn't new, the Tweet selection component will select it again so that the user can view the recent engagements on this Tweet.

New Tweets + unseen Tweets

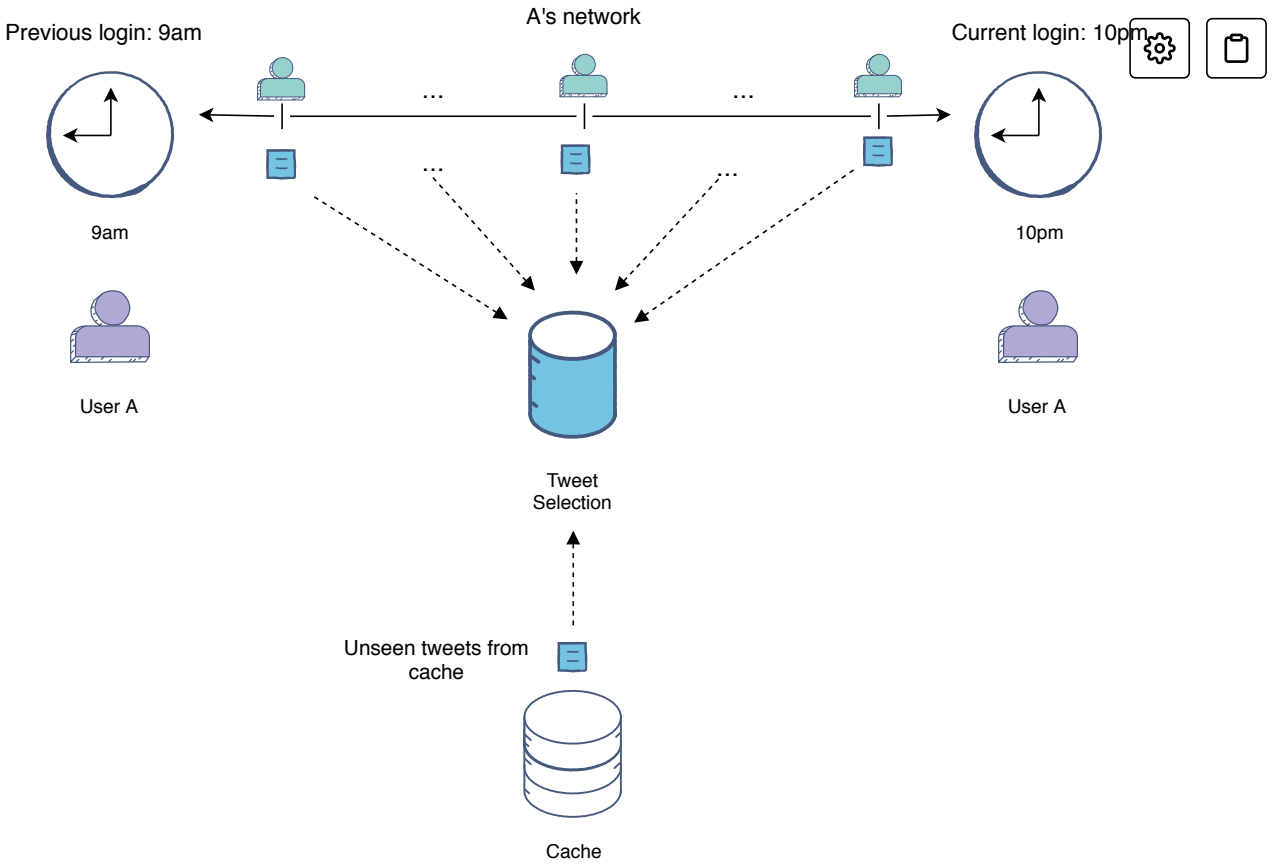
Picking up where we left off, let's see how this component may select a mix of new and old unseen Tweets.

The new Tweets fetched in the previous step are ranked and displayed on user A's feed. They only view the first two-hundred Tweets and log out. The ranked results are also stored in a cache.

Now, user A logs in again at 10 pm. According to the last scheme, the Tweet selection component should select all the new Tweets generated between 9 am and 10 pm. However, this may not be the best idea!

For 9am's feed, the component previously selected a Tweet made at 8:45 am. Since this Tweet was recently posted, it did not have much user engagement at that time. Therefore, it was ranked at the 450^{th} position in the feed. Now, remember that A logged out after only viewing the first two-hundred Tweets and this Tweet remained unread. Since the user's last visit, this unread Tweet gathered a lot of attention in the form of reshares, likes, and comments. The Tweet selection component should now reconsider this Tweet's selection. This time it would be ranked higher, and A will probably view it.

Keeping the above rationale in mind, the Tweet selection component now fetches a mix of newly generated Tweets along with a portion of unseen Tweets from the cache.

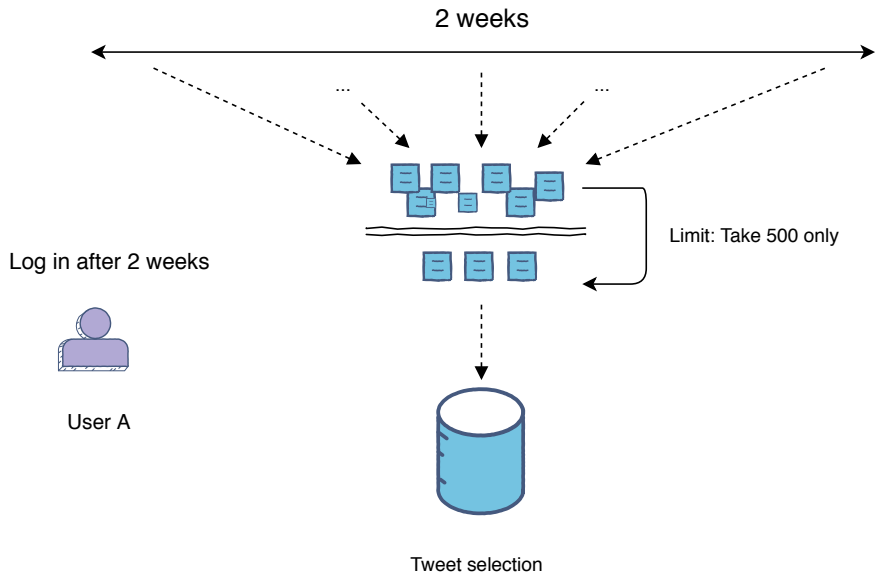


Select a mix of newly generated Tweets and unseen Tweets from the cache

You may encounter the following edge case during Tweet selection.

Edge case: User returning after a while

Consider a scenario, where user *A* might log in after two weeks. A lot of Tweets would have been generated since A's last login. Therefore, the Tweet selection component will impose a limit on the Tweet data it will select. Let's assume that the limit is five-hundred. Now the selected five-hundred Tweets may have been generated over the last two days or throughout the two weeks, depending on the user's network's activity. The main focus is that the pool of Tweets keeps on increasing so a limit needs to be imposed on the number of Tweets that the component will fetch.



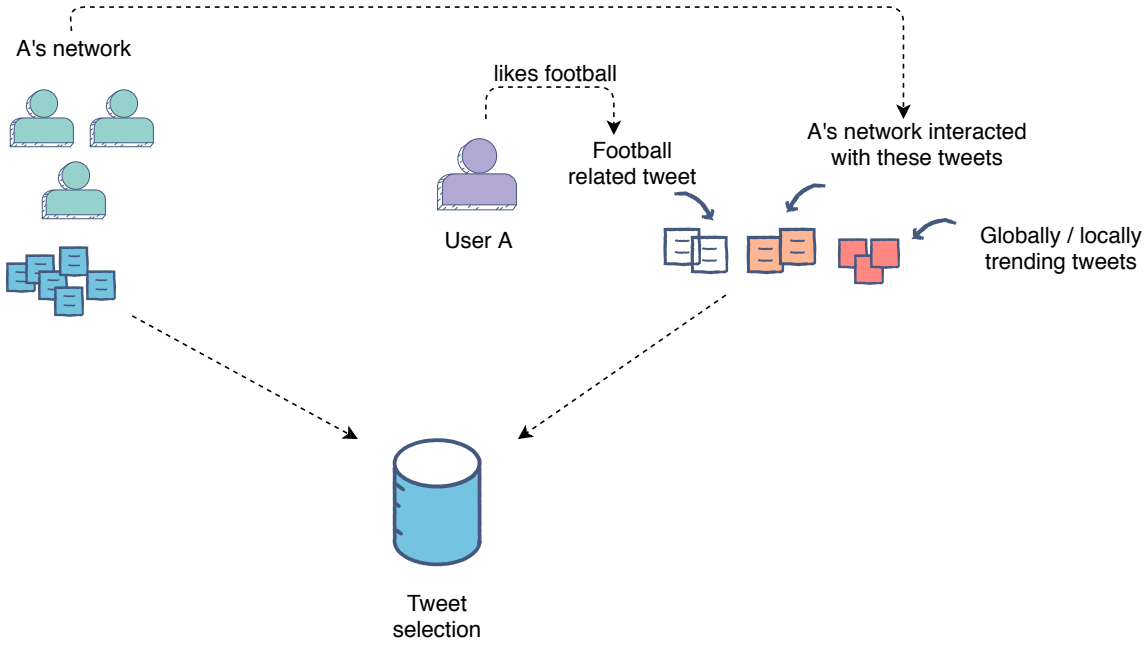
Limit imposed on the number of tweets selected

Network Tweets + interest / popularity-based Tweets

Until now, the Tweet selection component was only fetching Tweets from user A's network of followees. However, there could be Tweets **outside of user A's network** that have a high potential of engaging them. Hence, we arrive at a two-dimensional scheme of selecting network Tweets and potentially engaging Tweets.

An engaging Tweet could be one that:

- aligns with user A's interests
- is locally/globally trending
- engages user A's network



Selecting these Tweets can prove to be very beneficial in two cases:

1. The user has recently joined the platform and follows only a few others. As such, their small network is not able to generate a sufficient number of Tweets for the Tweet selection component (known as the Bootstrap problem).
2. The user likes a Tweet from a person outside of his network and decides to add them to their network. This would increase the discoverability on the platform and help grow the user's network.

> 📝 All of the schemes we have discussed showcase the dynamic nature of the Tweet selection component's task. The Tweets it will select will continue to change each time.

Back

Architectural Components

Next

Feature Engineering

Completed

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

Feature Engineering

Let's engineer some features for the Tweet ranking model.

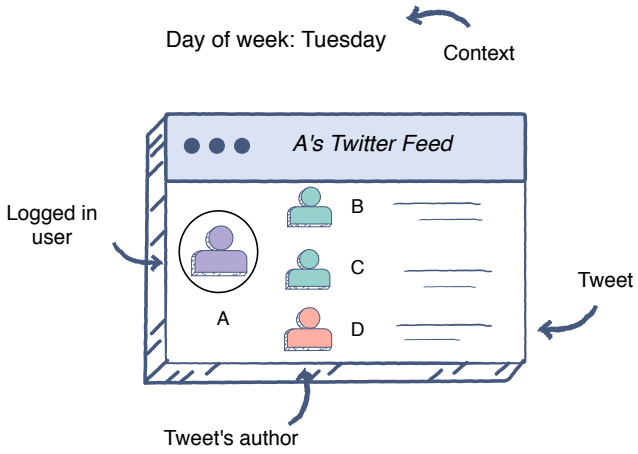
We'll cover the following

The machine learning model is required to predict user engagement on user *A's* Twitter feed. Let's engineer features to help the model make informed predictions.

> 📝 The feature set shown here is the result of one brainstorm session. However, feature engineering is an iterative process. As an exercise, you are encouraged to think about more features.

Let's begin by identifying the four main **actors** in a twitter feed:

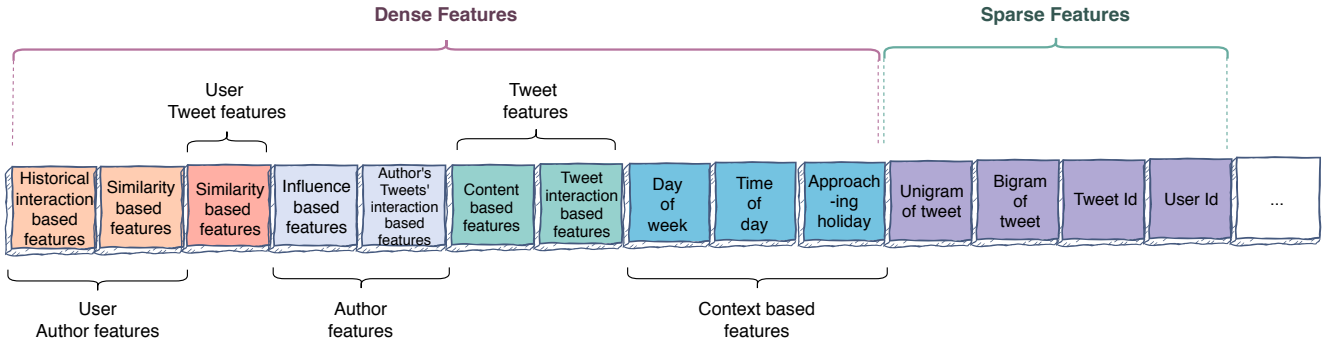
1. The logged-in user
2. The Tweet
3. Tweet's author
4. The context



Actors in a Twitter feed

Features for the model

Now it's time to generate features based on these actors and their interactions. A subset of the features is shown below.



Features in the training data row

Let's discuss these features one by one.

Dense features

We will start by discussing the dense features.

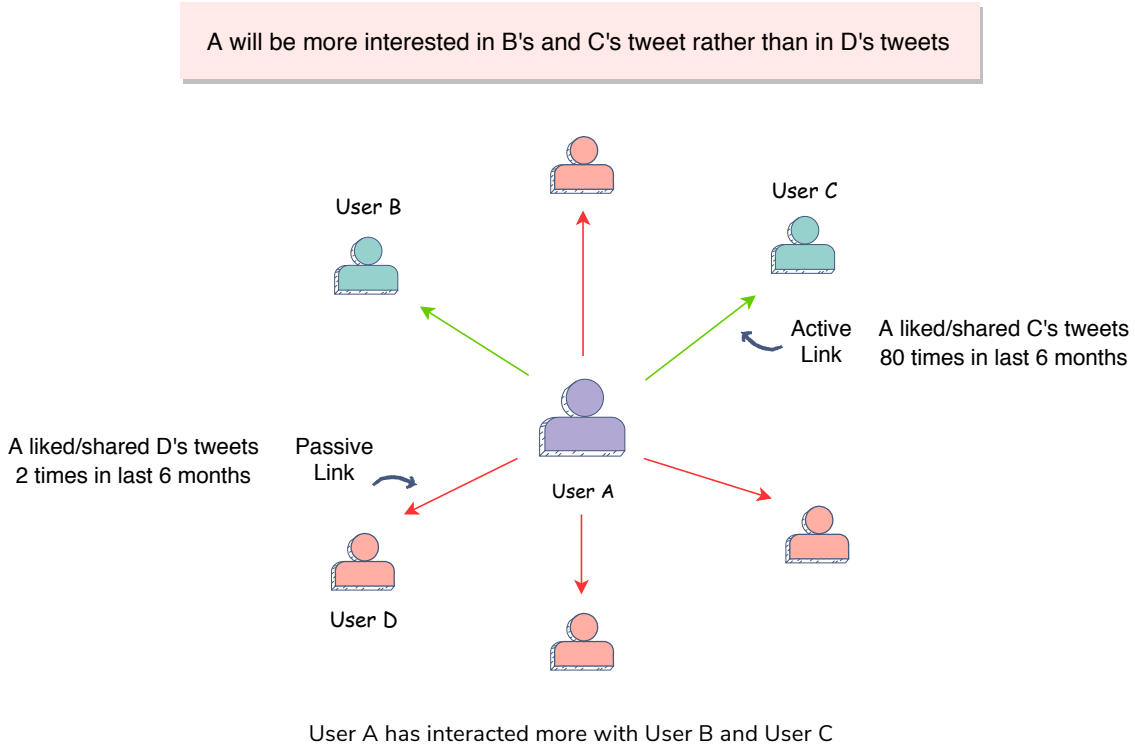
User-author features

These features are based on the logged-in user and the Tweet's author. They will capture the *social relationship* between the user and the author of the Tweet, which is an extremely important factor in ranking the author's Tweets. For example, if a Tweet is authored by a close friend, family member, or someone that user is highly influenced by, there is a high chance that the user would want to interact with the Tweet.

How can you capture this *relationship* in your signals given users are not going to specify them explicitly? Following are a few features that will effectively capture this.

User-author historical interactions

When judging the relevance of a Tweet for a user, the relationship between the user and the Tweet's author plays an important role. It is highly likely that if the user has actively engaged with a followee in the past, they would be more interested to see a post by that person on their feed.



User A has interacted more with User B and User C

Few features based on the above concept can be:

- **author_liked_posts_3months**

  This considers the percentage of an author's Tweets that are liked by the user in the last three months. For example, if the author created twelve posts in the last three months and the user interacted with six of these posts then the feature's value will be:

  $\frac{6}{12}$ = 0.5 or 50%

  This feature shows a more recent trend in the relationship between the user and the author.

- **author_liked_posts_count_1year**

  This considers the number of an author's Tweets that the user interacted with, in the last year. This feature shows a more long term trend in the relationship between the user and the author.

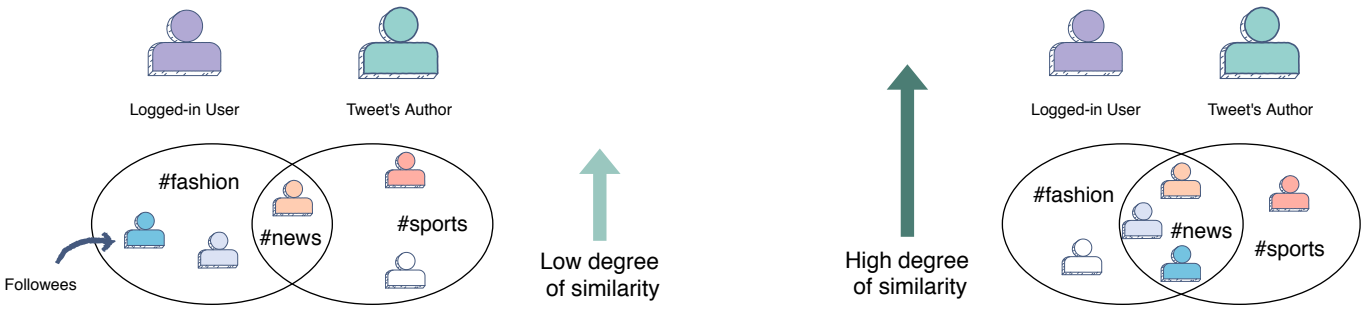
> 📝 Ideally, we should normalize the above features by the total number of Tweets that the user interacted with during these periods. This enables the model to see the real picture by cancelling out the effect of a user's general interaction habits. For instance, let's say user A generally tends to interact (e.g., like or comment) more while user B does not. Now, both user A and B have a hundred interactions on user C's posts. User B's interaction is more significant since they generally interact less. On the other hand, user A's interaction is mostly a result of their tendency to interact more.

User-author similarity #

Another immensely important feature set to predict user engagement focuses on figuring out how similar the logged-in user and the Tweet's author are. A few ways to compute such features include:

- **common_followees**

  This is a simple feature that can show the similarity between the user and the author. For a user-author pair, you will look at the number of users and hashtags that are followed by *both* the user and the author.



Similarity between logged-in user and Tweet's author

- **topic_similarity**

  The user and author similarity can also be judged based on their interests. You can see if they interact with similar topics/hashtags. A simple method to check this is the TF-IDF based similarity between the hashtags:

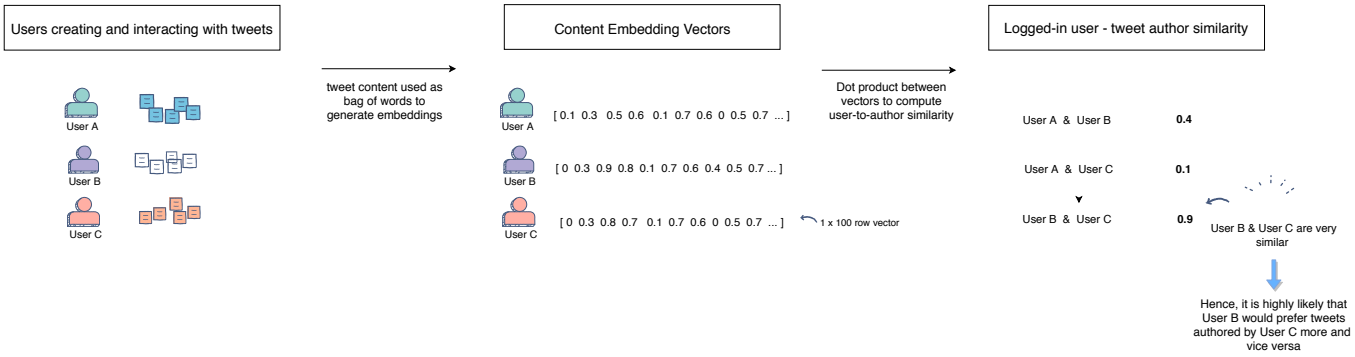
  - followed by the logged-in user and author
  - present in the posts that the logged-in user and author have interacted with in the past
  - used by the author and logged-in user in their posts

  The similarity between their search histories on Twitter can also be used for the same purpose.

- **tweet_content_embedding_similarity**

  AT user is represented by the content that they have generated and interacted with in the past. You can utilize all of that content as a bag-of-words and build an embedding for every user. With an embedding vector for each user, the dot product between them can be used as a fairly good estimate of user-to-author similarity.

> 📝 Embedding helps to reduce the sparsity of vectors generated otherwise.
>
> Have a look at the lesson (https://www.educative.io/collection/page/10370001/6237869033127936/6130870193750016) on embeddings to get an idea on how the embedding can be learned for this scenario.

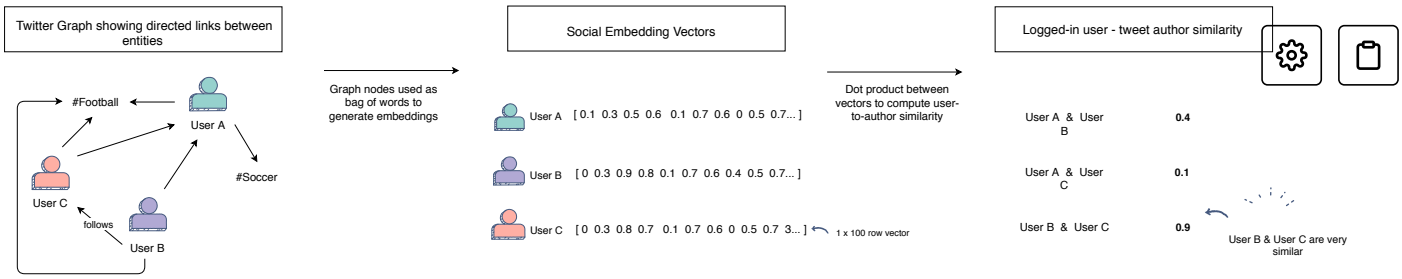


Content embedding similarity between logged-in user and Tweet's author

User B and C are very similar. Hence, it is highly likely that user B will prefer Tweets authored by user C more and vice versa.

- **social_embedding_similarity**

Another way to capture the similarity between the user and the author is to generate embeddings based on the social graph rather than based on the content of Tweets, as we discussed earlier. The basic notion is that people who follow the same or similar topics or influencers are more likely to engage with each other's content. A basic way to train this model is to represent each user with all the other users and topics (user and topic ids) that they follow in the social graph. Essentially, every user is represented by bag-of-ids (rather than bag-of-words), and you use that to train an embedding model. The user-author similarity will then be computed between their social embeddings and used as a signal.



Social similarity between user and author

> 📝 Have a look at the lesson (https://www.educative.io/collection/page/10370001/6237869033127936/6130870193750016) on embeddings to get an idea on how the embedding can be learned for this scenario.

Author features

These features are based on Tweet's author.

Author's degree of influence

A Tweet written by a more *influential* author may be more relevant. There are several ways to measure the author's influence. A few of these ways are shown as features for the model, below.

- **is_verified**

  If an author is verified, it is highly likely that they are somebody important, and they have influence over people.

- **author_social_rank**

  The idea of the author's social_rank is similar to Google's page rank (https://en.wikipedia.org/wiki/PageRank#Algorithm).

  To compute the social rank of each user, you can do a random walk (https://ocw.mit.edu/courses/electrical-engineering-and-computer-science/6-042j-mathematics-for-computer-science-spring-2015/readings/MIT6_042JS15_Session35.pdf) (like in page rank). Each person who follows the author contributes to their rank. However, the contribution of each user is not equal. For example, a user adds more weight if they are followed by a popular celebrity or a verified user.

- **author_num_followers**

  One important feature could be the number of followers the author has. Different inferences can be drawn from different follower counts, as shown below:

Follower count	Inference
150	Personal account with reasonable influence over their social circle
1 thousand	Social media influencer with a good amount of influence over fans
3 million	Celebrity with a great fan following

📝 **Normalised author_num_followers**
You can observe a lot of variation in the follower counts of Twitter users. To bring each user's follower count in a specific range, let's say zero to ten-thousand, you can divide their follower count by the maximum observed follower count (across the platform) and then multiply it with ten-thousand.

- **follower_to_following_ratio**

  When coupled with the number of followers, the follower to following ratio can provide significant insight, regarding:

  1. The type of user account 2.The account's influence
  2. The quality of the account's content (Tweets)

  Let's see how.

Follower to Following Ratio	Follower Count	Inference
≈ 0	< 500	The user has a personal account, mainly following brands and celebrities. The account has average content quality, and its influence is limited to the user's friends and family.

Follower to Following Ratio	Follower Count	Inference
0.5	≈ 3 thousand	The user is a small influencer with low-quality content. They may be using the follow/unfollow method to gain users, i.e., they follow people to receive a follow-back. Once they do, they unfollow them. People may unfollow once they figure out what is going on.
10+	>= 15 thousand	The user is likely to be a micro-celebrity with good quality content. He/she have a wider influence, not limited to their social circle only.
370370	≈ 30 million	The user is Elon Musk!

Historical trend of interactions on the author's Tweets

Another very important set of features is the interaction history on the author's Tweets. If historically, an author's Tweets garnered a lot of attention, then it is highly probable that this will happen in the future, too.

> 📝 A high rate of historical interaction for a user implies that the user posts high-quality content.

Some features to capture the historical trend of interactions for the author's Tweets are as follows:

- **author_engagement_rate_3months**

  The *engagement rate* of the historical Tweets by the author can be a great indicator of future Tweet engagement. To compute the engagement rate in the last three months, we look at how many times different users interacted with the author's Tweets that they viewed.

  Engagement rate: $\frac{Tweet-interactions}{Tweet-views}$

- **author_topic_engagement_rate_3months**

  The engagement rate can be different based on the Tweet's topic. For example, if the author is a sports celebrity, the engagement rate for their family-related Tweets may be different from the engagement rate for their sports-related Tweets.

  We can capture this difference by computing the engagement rate for the author's Tweets per topic. Tweet topics can be identified in the following two ways:

  1. Deduce the Tweet topic by the hashtags used
  2. Predict the Tweet topic based on its content

> 📝 These features don't necessarily have to be based on data from three months, but rather you should utilize different time ranges, e.g., historical engagement rates in last week, month, three months, six months, etc.

User-Tweet features

The similarity between the user's interests and the tweet's topic is also a good indicator of the relevance of a Tweet. For instance, if user A is interested in football, a Tweet regarding football would be very relevant to them.

- **topic_similarity**

  You can use the hashtags and/or the content of the Tweets that the user has either Tweeted or interacted with, in the last six months and compute the TF-IDF similarity with the Tweet itself. This indicates whether the Tweet is based on a topic that the user is interested in.

- **embedding_similarity**

  Another option to find the similarity between the user's interest and the Tweet's content is to generate embeddings for the user and the Tweet. The Tweet's embedding can be made based on the content and hashtags in it. While the user's embedding can be made based on the content and hashtags in the Tweets that they have written or interacted with. A dot product between these embeddings can be calculated to measure their similarity. A high score would equate to a highly relevant Tweet for a user.

Tweet features

These features are based on the Tweet itself.

Features based on Tweet's content

- **Tweet_length**

The length of the Tweet positively correlates with user engagement, especially likes and reshares.

It is generally observed that people have a short attention span and prefer a shorter read. Hence, a more concise Tweet generally increases the chance of getting a like by the user. Also, we know that Twitter restricts the length of the Tweet. So, if a person wants to Retweet a Tweet that has nearly used up the word limit, the person would not be able to add their thoughts on it, which might be off-putting.

- **Tweet_recency**

  The recency of the Tweet is an important factor in determining user engagement, as people are most interested in the latest developments.

- **is_image_video**

  The presence of an image or video makes the tweet more catchy and increases the chances of user engagement.

- **is_URL**

  The presence of a URL may mean that the Tweet:

  1. Calls for action
  2. Provides valuable information

  Hence, such a Tweet might have a higher probability of user engagement.

Features based on Tweet's interaction

You should also utilize the Tweet's interactions as features for our model. Tweets with a greater volume of interaction have a higher probability of engaging the user. For instance, a Tweet with more likes and comments is more relevant to the user, and there is a good chance that the user will like or comment on it too.

- **num_total_interactions**

  The total number of interactions (likes, comments and reshares) on a Tweet can be used as a feature.

  Caveat

  Simply using these interactions as features might give an incomplete picture. Consider a scenario where Bill Gates Tweeted a month ago, and his Tweet received five million engagements over this period. Whereas, another Tweet posted just an hour ago has two-

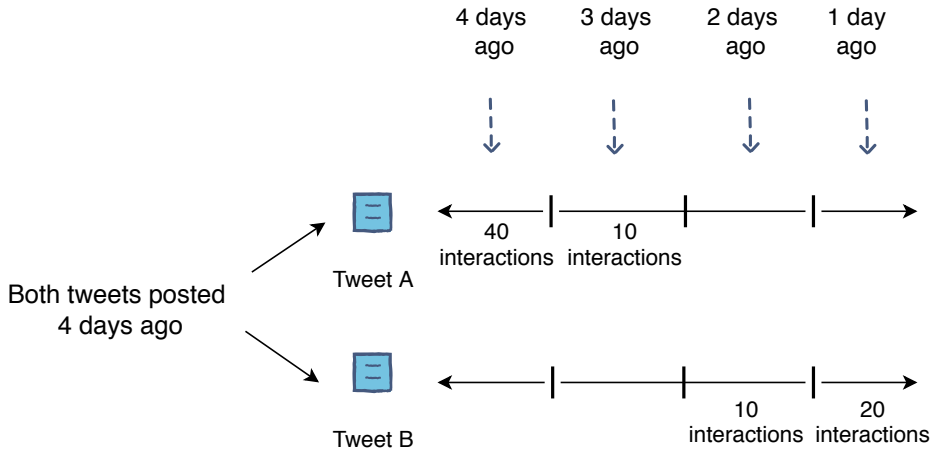
thousand engagements and has become a trending Tweet. Now, if your feature is based on the interaction count only, Bill Gate's Tweet would be considered more relevant, and the quick rise in the popularity of the trending Tweet would not be given its due importance.

To remedy this, we can apply a simple **time decay** model to weight the latest interaction more than the ones that happened some time ago.

> 📝 Time decay can be used in all features where there is a decline in the value of a quantity over time.

One simple model can be to weight every interaction (*like, comment, and retweet*) by $\frac{1}{t+1}$ where t *is the number of days from current time*.

In the above scenario, you saw two Tweets, Tweeted with a lot of time difference. The same scenario can also happen for two Tweets tweeted at the same time, as shown below. Let's see how you can use time decay to see the real value of engagement on these Tweets.



Weighting the interactions with time decay changes the values

Tweet	Total Engagement	Weighted Engagement
A	50	10.5
B	30	13.3

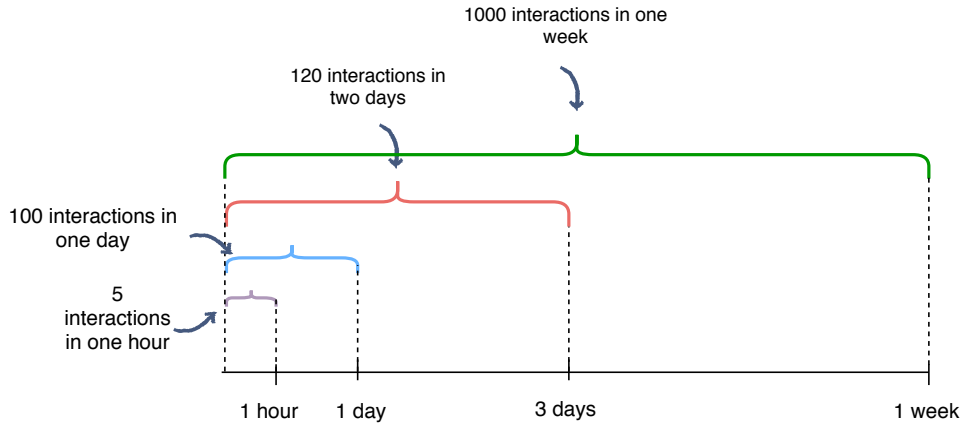
$40 * \frac{1}{4+1} + 10 * \frac{1}{3}$

$10 * \frac{1}{\quad} + 20 *$

Tweet A had a total of fifty interactions, while Tweet B had a total of thirty. However, the interactions on Tweet B were more recent. So, by using time decay, the weighted number of likes on Tweet B became greater than those for Tweet A.

Another remedy is to use different **time windows** to capture the recency of interactions while looking at their numbers. The interaction in each window can be used as a feature:

- **interactions_in_last_1_hour**
- **interactions_in_last_1_day**
- **interactions_in_last_3_days**
- **interactions_in_last_week**



Different overlapping time windows to capture interactions on tweets

Separate features for different engagements

Previously, we discussed combining all interactions. You can also keep them as separate features, given you can predict different events, e.g., the probability of *likes, Retweets and comments*. Some potential features can be:

- **likes_in_last_3_days**
- **comments_in_last_1_day**
- **reshares_in_last_2_hours**

The above three features are looking at their respective forms of interactions from all twitter users. Another set of features can be generated by looking at the interactions on the Tweet made only by user A's network. The intuition behind doing this is that there is a high probability that if a Tweet has more interactions from A's network, then A, having similar tastes, will also interact with that Tweet.

The set of features based on user's network's interactions would then be:

- **likes_in_last_3_days_user's_network_only**
- **comments_in_last_1_day_user's_network_only**
- **reshares_in_last_2_hours_user's_network_only**

Context-based features

These features are based on the context.

- **day_of_week**

  The day of the week can affect the type of Tweet content a user would like to see on their feed.

- **time_of_day**

  Noting the time of the day (coupled with the day of the week) can provide useful information. For example, if a user logs-in on a Monday morning, it is likely that they are at their place of work. Now, it would make sense to show shorter Tweets that they can quickly read. In contrast, if they login in the evening, they are probably at their home and would have more time at their disposal. Therefore, you can show them longer Tweets with video content as well, since the sound from the video would not be bothersome in the home environment.

- **current_user_location**

  The current user location can help us show relevant content. For example, a person may go to San Francisco where a festival is happening (it is the talk of the town). Based on what is popular in a certain area where the user is located, you can show relevant Tweets.

- **season**

  User's viewing preference for different Tweet topics may be patterned according to the four seasons of the year.

- **lastest_k_tag_interactions**

  You can see the latest "k" tags included in the Tweets a user has interacted with to gain valuable insights. For example, assume that the last $k = 5$ Tweets a user interacted with contain the tag "Solar eclipse". From this, we can discern that the user is most likely interested in seeing Tweets centered around the solar eclipse.

- **approaching_holiday**

  Recording the approaching holiday will allow the model to start showing the user more content centred around that holiday. For instance, if Independence Day is approaching, Tweets with more patriotic content would be displayed.

Until now, all the features that we discussed have been dense features.

Sparse features

For some ML models, sparse features can be useful. Following are a few of the sparse features that would prove to be helpful when predicting engagement.

- *unigrams/bigrams of a Tweet*

The presence of certain unigrams or bigrams may increase the probability of engagement for a tweet. For example, during data visualisation, you may observe that Tweets with the bigram "Order now" have higher user engagement. The reason behind this might be that such Tweets are useful and informative for users as they redirect them towards websites where they can purchase things according to their needs.

- *user_id*

  This is a very simple feature used to identify users with very high engagement rates, such as celebrities and influencers.

- *tweets_id*

  This feature is used to identify Tweets that have high engagement and hence have a higher probability of engaging users.

> 📝 Have a look at the embeddings lesson (https://www.educative.io/collection/page/10370001/6237869033127936/6130870193750016) to find out how some of these sparse features can be utilized in our models, via embeddings.

Back

Tweet Selection

Next

Training Data Generation

Completed

Training Data Generation

Let's collect and label training data for the feed ranking ML model.

We'll cover the following

Your user engagement prediction model's performance will depend largely on the quality and quantity of the training data. So, let's see how you can generate training data for your model.

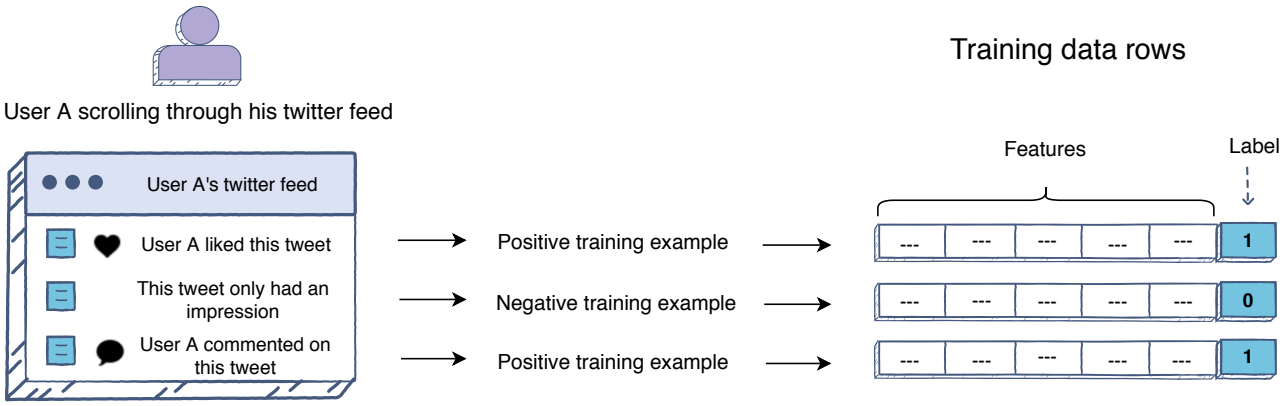
> 📝 Note that the term *training data row* and *training example* will be used interchangeably.

Training data generation through online user engagement

The users' online engagement with Tweets can give us positive and negative training examples. For instance, if you are training a single model to predict user engagement, then all the Tweets that received user engagement would be labeled as positive training examples. Similarly, the Tweets that only have impressions would be labeled as negative training examples.

> 📝 **Impression**: If a Tweet is displayed on a user's Twitter feed, it counts as an impression. It is not necessary that the user reads it or engages with it, scrolling past it also counts as an impression.

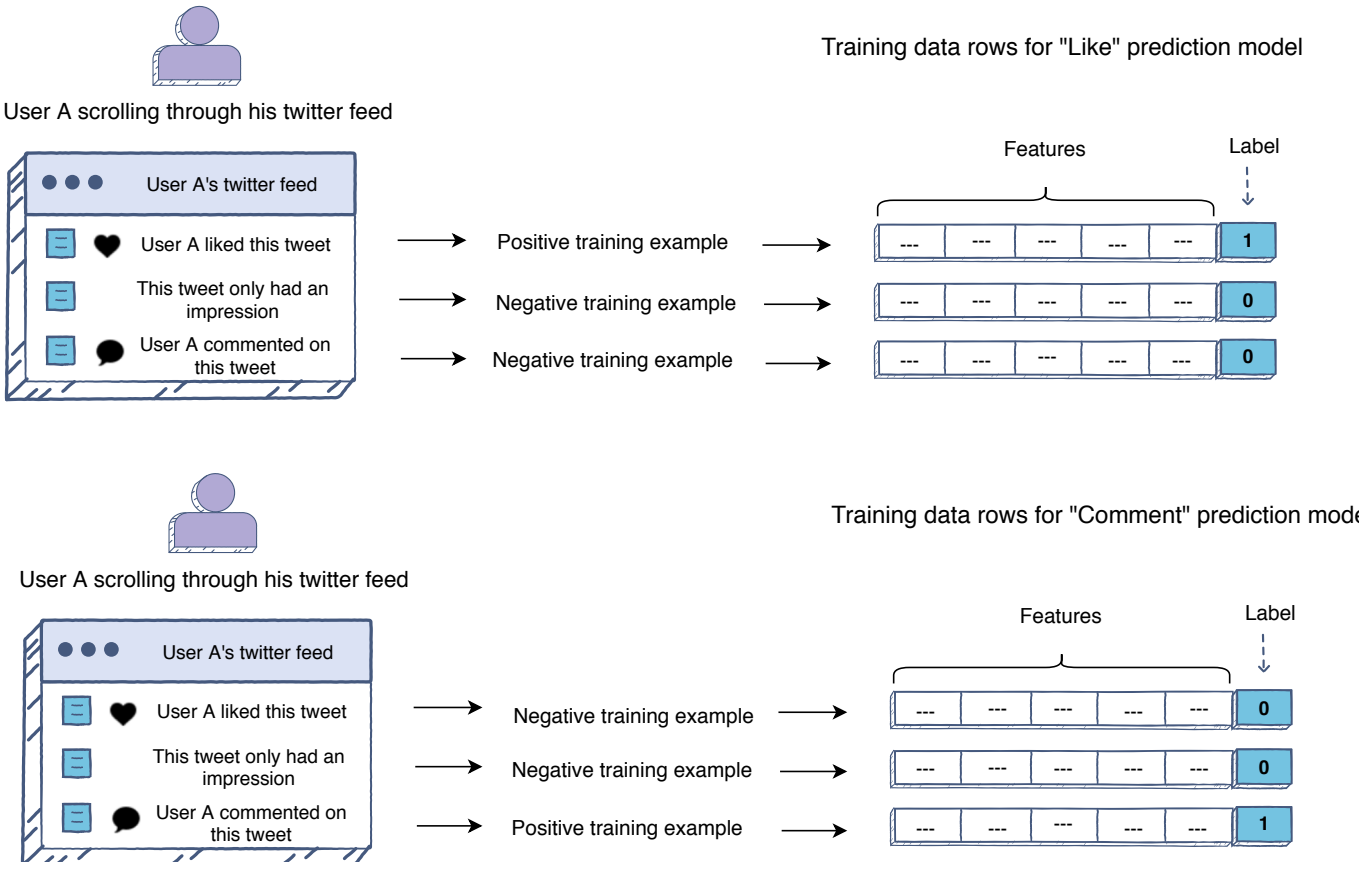
Training data generation for a single model to predict user engagement



Any user engagement counts as positive example

However, as you saw in the architectural components lesson, that you can train different models, each to predict the probability of occurrence of different user actions on a tweet. The following illustration shows how the same user engagement (as above) can be used to generate training data for separate engagement prediction models.

Training data generation for separate models to predict user engagement



When you generate data for the “Like” prediction model, all Tweets that the user has liked would be positive examples, and all the Tweets that they did not like would be negative examples.

> 📝 Note how the comment is still a negative example for the “Like” prediction model.

Similarly, for the “Comment” prediction model, all Tweets that the user commented on would be positive examples, and all the ones they did not comment on would be negative examples.

Balancing positive and negative training examples

Models essentially learn behavior from the data we present them with. Therefore, it’s important for us to provide a good sample of both positive and negative examples to model these interactions between different actors in a system. In the feed-based system scenario, on average, a user engages with as little as approximately 5% of the Tweets that they view per day. How would this percentage affect the ratio of positive and negative examples on a larger scale? How can this be balanced?

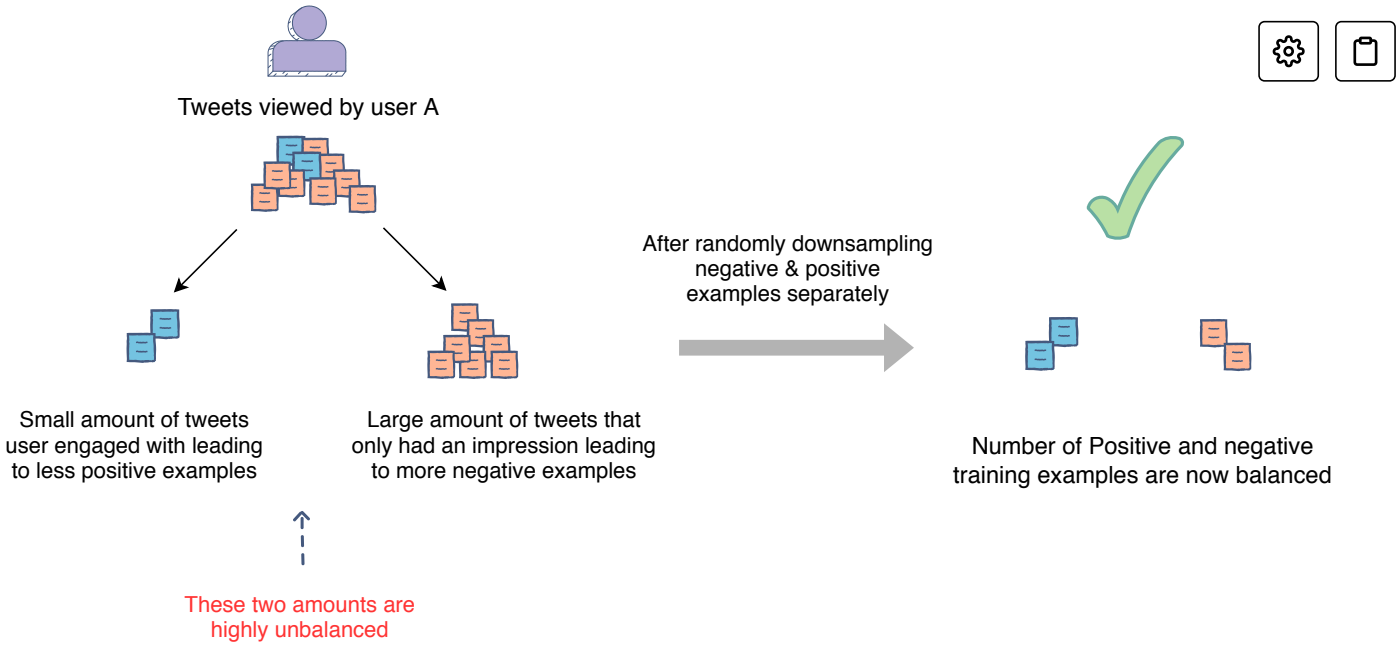
Looking at the bigger picture, assume that one-hundred million Tweets are viewed collectively by the users in a day. With the 5% engagement ratio, you would be able to generate five million positive examples, whereas the remaining ninety-five million would be negative.

Let’s assume that you want to limit your training samples to ten million, given that the training time and cost increases as training data volume increases. If you don’t do any balancing of training data and just randomly select ten million training samples, your models will see only 0.5 million positive examples and 9.5 million negative examples. In this scenario, the models might not be able to learn *key* positive interactions.

Therefore, in order to balance the ratio of positive and negative training samples, you can **randomly downsample**:

- negative examples to five million samples
- positive examples to five million samples

Now, you would have a total of **ten million training examples per day**; five million of which are positive and five million are negative.



> **Note**
> If a model is *well-calibrated*, the distribution of its predicted probability is similar to the distribution of probability observed in the training data. However, as we have changed the sampling of training data, our model output scores will **not** be well-calibrated. For example, for a tweet that got only 5% engagement, the model might predict 50% engagement. This would happen because the model is trained on data that has an equal quantity of negative and positive samples. So the model would think that it is as likely for a tweet to get engagement as it is to be ignored. However, we know from the training that this is not the case.
> Given that the model's scores are only going to be used to rank Tweets among themselves, poor model calibration doesn't matter much in this scenario. We will discuss this in the ads system chapter (https://www.educative.io/collection/page/10370001/6237869033127936/6476235518509056#model-recalibration), where calibrated scores are important, and we need to be mindful of such a sampling technique.

Train test split

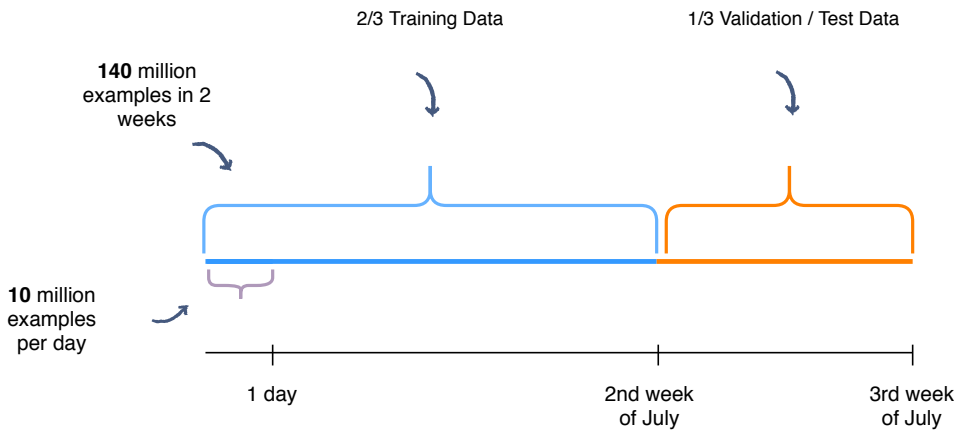
You need to be mindful of the fact that the user engagement patterns may differ throughout the week. Hence, you will use a week's engagement to capture all the patterns during training data generation. At this rate, you would end up with around seventy million rows of training data.

You may randomly select $\frac{2}{3}^{rd}$, or 66.6%, of the seventy million training data rows that you 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 defeats the purpose of training the model on an entire week's data. Also, the data has a time dimension, i.e., we know the engagement on previous Tweets, and we want to predict the engagement on future Tweets ahead of time. Therefore, you will train the model on data from one time interval and validate it on the data with the succeeding time interval. This will give a more accurate picture of how the model will perform in a real scenario.

> 📝 We are building models with the intent to forecast the future.

In the following illustration, we are training the model using data generated from the first and second week of July and data generated in the thirdweek of July for validation and testing purposes.



Splitting data for training, validation, and testing

> 📝 Have a look at the training data collection strategies lesson (https://www.educative.io/collection/page/10370001/6237869033127936/6285523644579840) to get more insights.

Back

Feature Engineering

Next

Ranking

Completed

Ranking

In this lesson, we'll explore different modeling options for the Tweet ranking problem.

We'll cover the following

In the previous lesson, you generated the training data. Now, the task at hand is to predict the probability of different engagement actions for a given Tweet. So, essentially, your models are going to predict the probability of *likes, comments and retweets*, i.e., *P(click), P(comments) and P(retweet)*. Looking at the training data, we know that this is a classification problem where we want to minimize the classification error or maximize area under the curve(AUC).

Modeling options

Some questions to consider are: Which model will work best for this task? How should you set up these models? Should you set up completely different models for each task? Let's go over the modeling options and try to answer these questions. We will also discuss the pros and cons of every approach.

Logistic regression

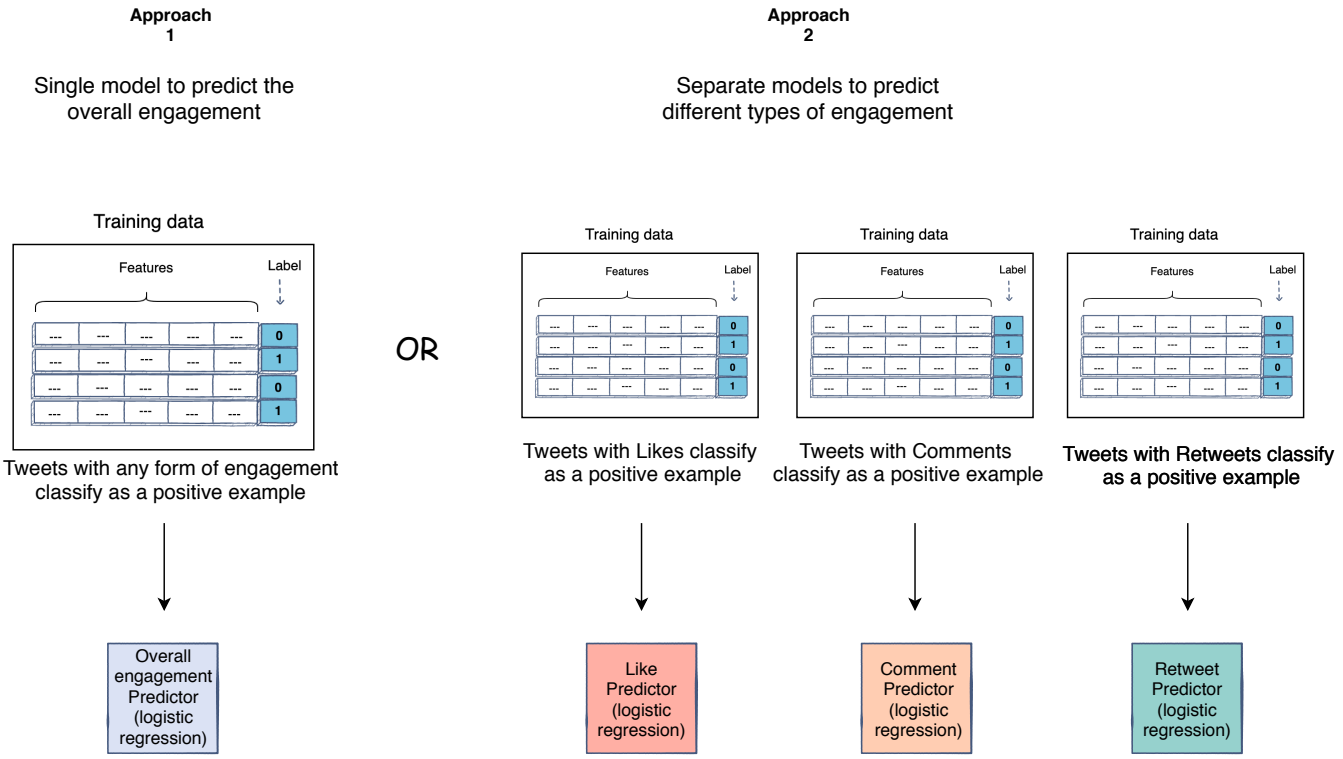
Initially, a simple model that makes sense to train is a logistic regression model with regularization, to predict engagement using the dense features that we discussed in the feature engineering lesson.

A key advantage of using logistic regression is that it is reasonably fast to train. This enables you to test new features fairly quickly to see if they make an impact on the AUC or validation error. Also, it's extremely easy to understand the model. You can see from the feature weights which features have turned out to be more important than others.

A major limitation of the linear model is that it assumes linearity exists between the input features and prediction. Therefore, you have to manually model feature interactions. For example, if you believe that the *day of the week before a major holiday* will have a major impact on your engagement prediction, you will have to create this feature in your training data *manually*. Other models like tree-based and neural networks are able to learn these feature interactions and utilize them effectively for predictions.

Another key question is whether you want to **train a single classifier for overall engagement or separate ones for each engagement action** based on production needs. In a single classifier case, you can train a logistic regression model for predicting the overall engagement on a Tweet. Tweets with any form of engagement will be considered as positive training examples for this approach.

Another approach is to train separate logistic regression models to predict P(like), P(comments) and P(retweet).



These models will utilize the same features. However, the assignments of labels to the training examples will differ. Tweets with likes, comments, or retweets will be considered positive examples for the overall engagement predictor model. Only Tweets with *likes* will be considered as positive examples for the “Like” predictor model. Similarly, only Tweets with *comments* will be considered as positive examples for the “Comment” predictor model and only Tweets with *retweets* will be considered as positive examples for the “Retweet” predictor model.

> 📝 As with any modelling, we should try and tune all the hyperparameters and play with different features to find the best model to fit this problem.

MART

> 📝 MART stands for multiple additive regression trees.

Another modeling option that should be able to outperform logistic regression with dense features is additive tree-based models, e.g., Boosted Decision Trees and Random Forest. Trees are inherently able to utilize non-linear relations between features that aren't readily available to logistic regression as discussed in the above section.

Tree-based models also don't require a large amount of data as they are able to generalize well quickly. So, a few million examples should be good enough to give us an optimized model.

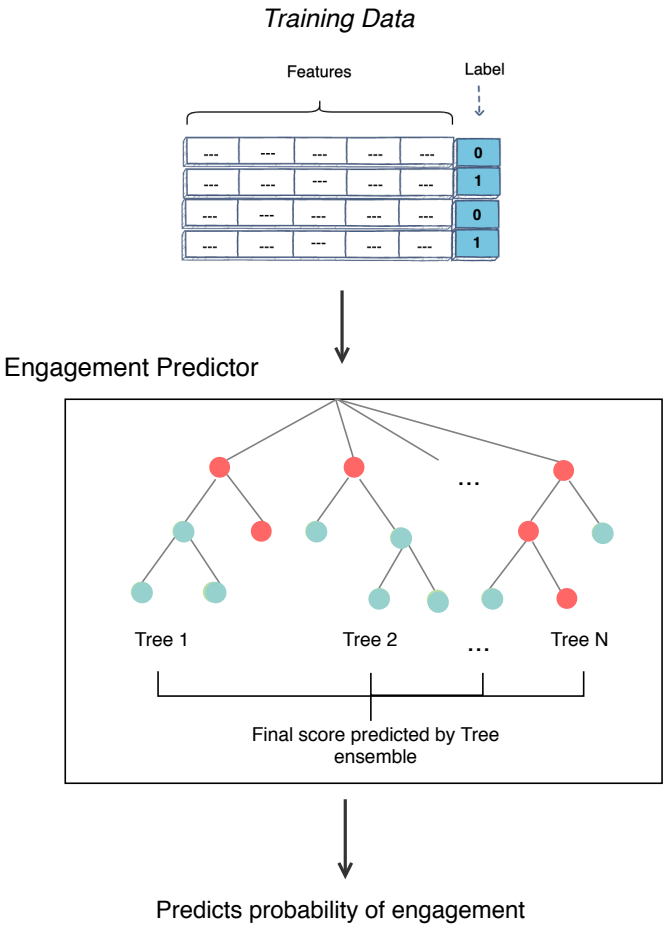
There are various hyperparameters that you might want to play around to get to an optimized model, including

- Number of trees
- Maximum depth
- Minimum samples needed for split
- Maximum features sampled for a split

To choose the best predictor, you should do hyperparameter tuning and select the hyperparameter that minimizes the test error.

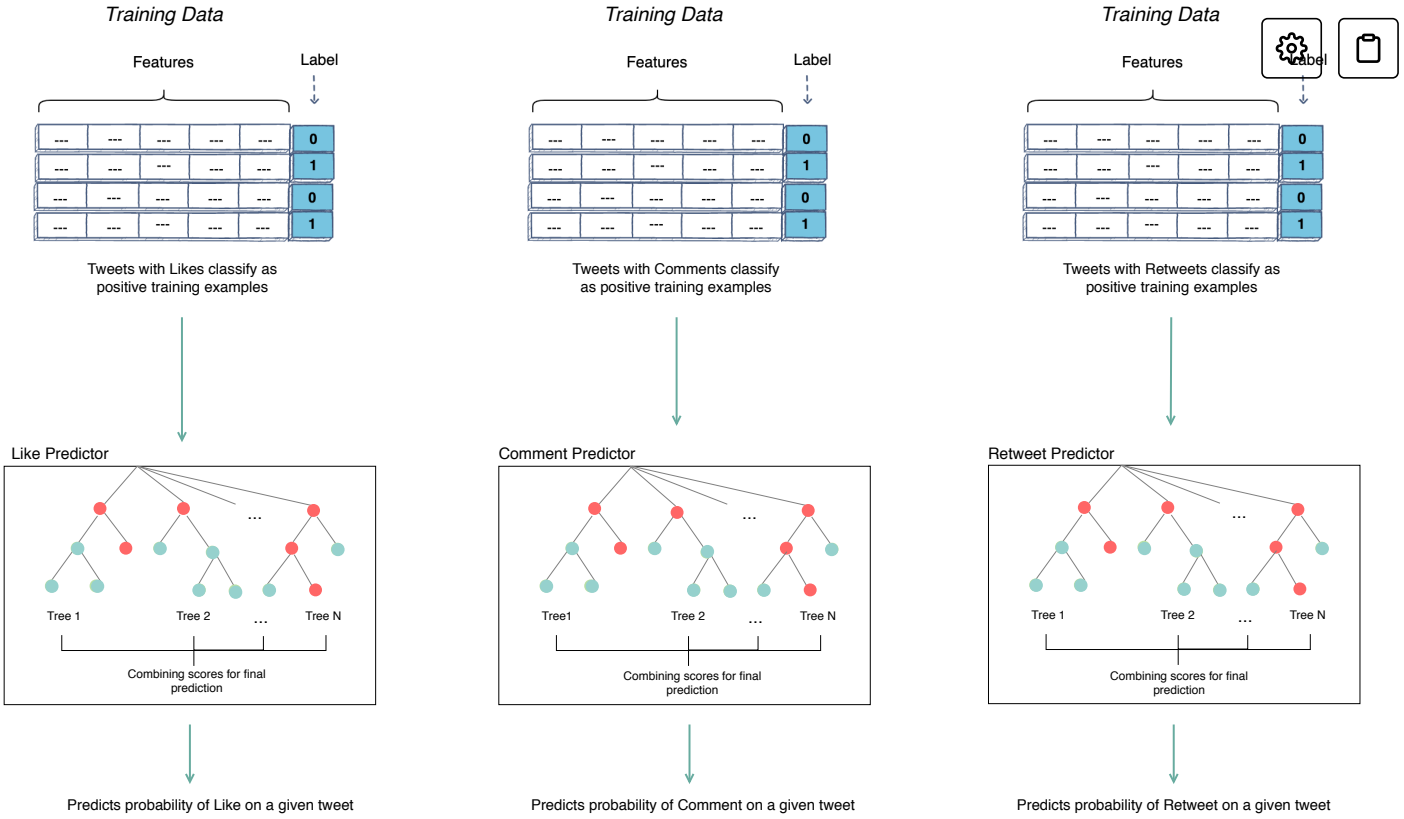
Approach 1

A simple approach is to train a single model to predict the overall engagement.



Approach 2

You could have specialized predictors to predict different kinds of engagement. For example, you can have a *like* predictor, a *comment* predictor and a *reshare* predictor. Once again these models will utilize the same features. However, the assignments of labels to the training examples will differ depending on the model's objective.

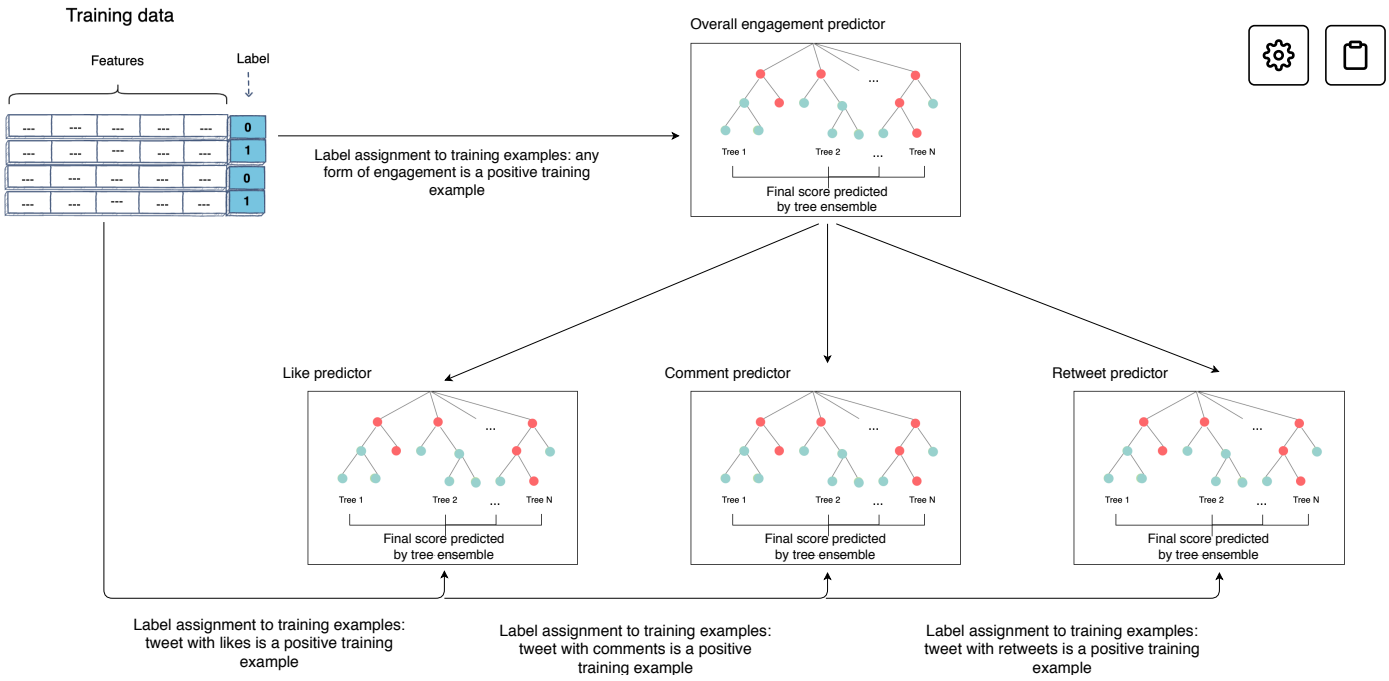


Specialized predictor for each type of user engagement

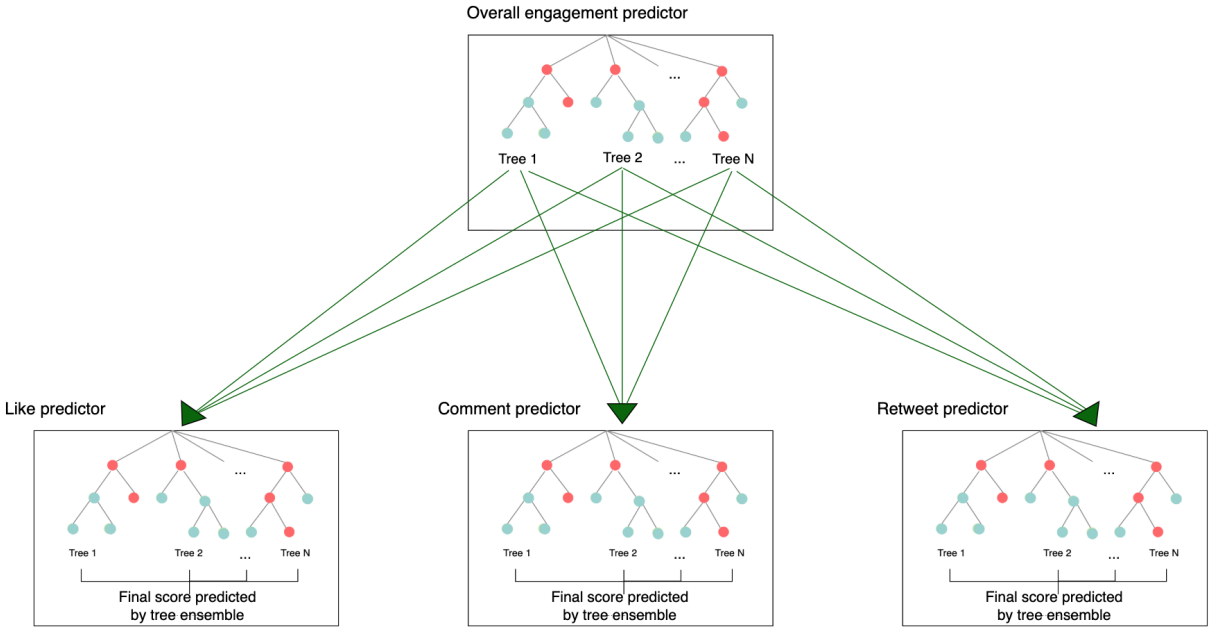
Approach 3

Consider a scenario, where a person reshares a Tweet but does not click the like button. Even though the user didn't actually click on the like button, retweeting generally implies that the user likes the Tweet. The positive training example for the retweet model may prove useful for the like model as well. Hence, you can reuse all positive training examples across every model.

One way to utilize the overall engagement data among each individual predictor of P(like), P(comment) and P(retweet) is to build one common predictor, i.e., P(engagement) and *share its output as input* into all of your predictors.



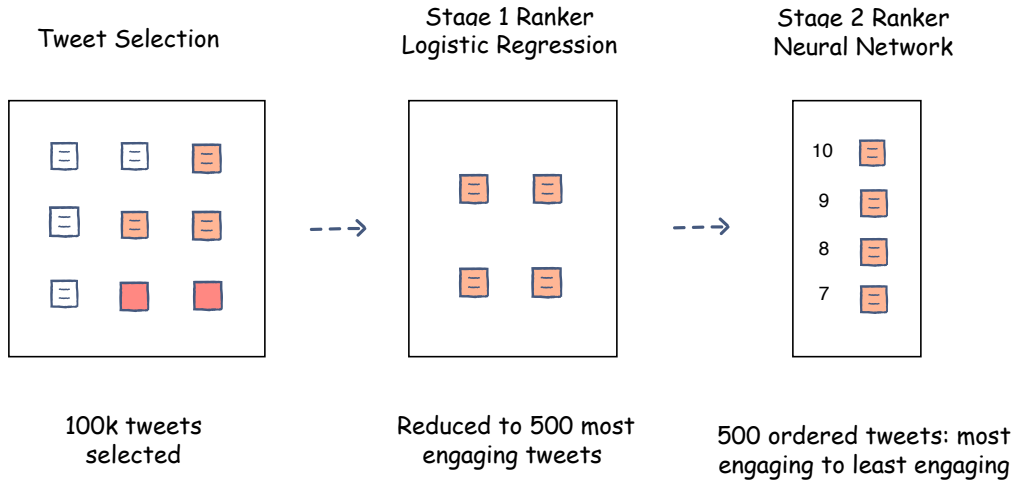
📝 To take **approach three** a notch further, you can use the output of each tree in the "overall engagement predictor" ensemble as features in the individual predictors. This allows for even better learning as the individual model will be able to learn from the output of each individual tree.



Deep learning

With advancements in computational power and the amount of training data you can gather from hundreds of million users, deep learning can be very powerful in predicting user engagement for a given Tweet and showing the user a highly personalized feed.

Having said that, training the model as well as evaluating the model at feed generation time can make this approach computationally very expensive. So, you may want to fall back to the multi-layer approach (https://www.educative.io/collection/page/10370001/6237869033127936/5814461362339840#layered-model-approach), which we discussed in the search ranking chapter, i.e., having a simpler model for stage one ranking and use complex stage two model to obtain the most relevant Tweets ranked at the top of the user's Twitter feed.



The layered model approach

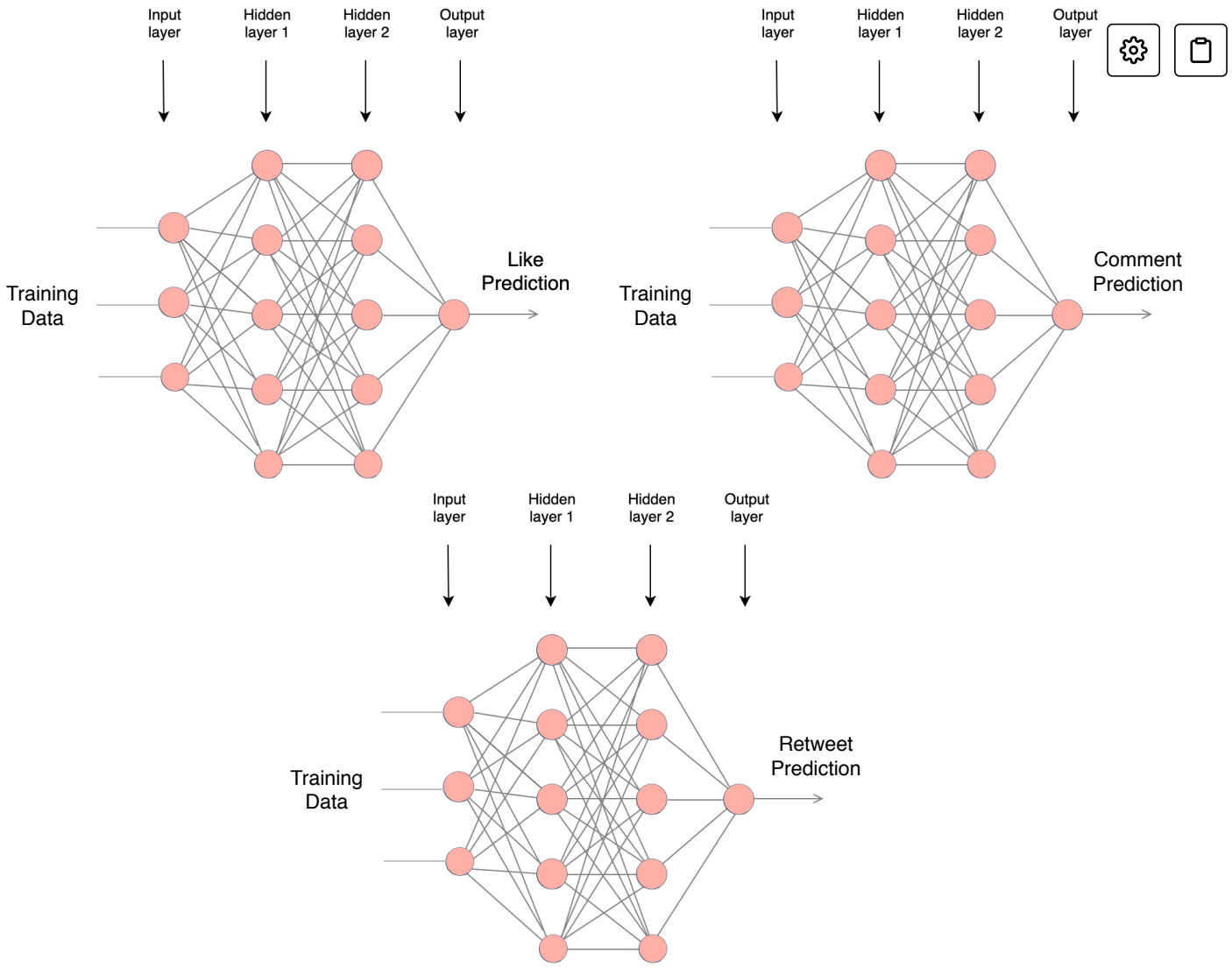
Like with the tree-based approach, there are quite a few hyperparameters that you should tune for deep learning to find the most optimized model that minimizes the test set error. They are:

- Learning rate
- Number of hidden layers
- Batch size
- Number of epochs
- Dropout rate for regularizing model and avoiding overfitting

> 📝 Neural networks are a great choice to add all the raw data and dense features that we used for tree-based models.

Separate neutral networks

One way is to train separate neural nets for each of the P(like), P(comment) and P(retweet).



Separate neural networks to predict probability of likes, comments, and retweets

However, for a very large scale data set, training separate deep neural networks (NNs) can be slow and take a very long time (ten's of hours to days). The following approach of setting up the models for multi-task learning caters to this problem.

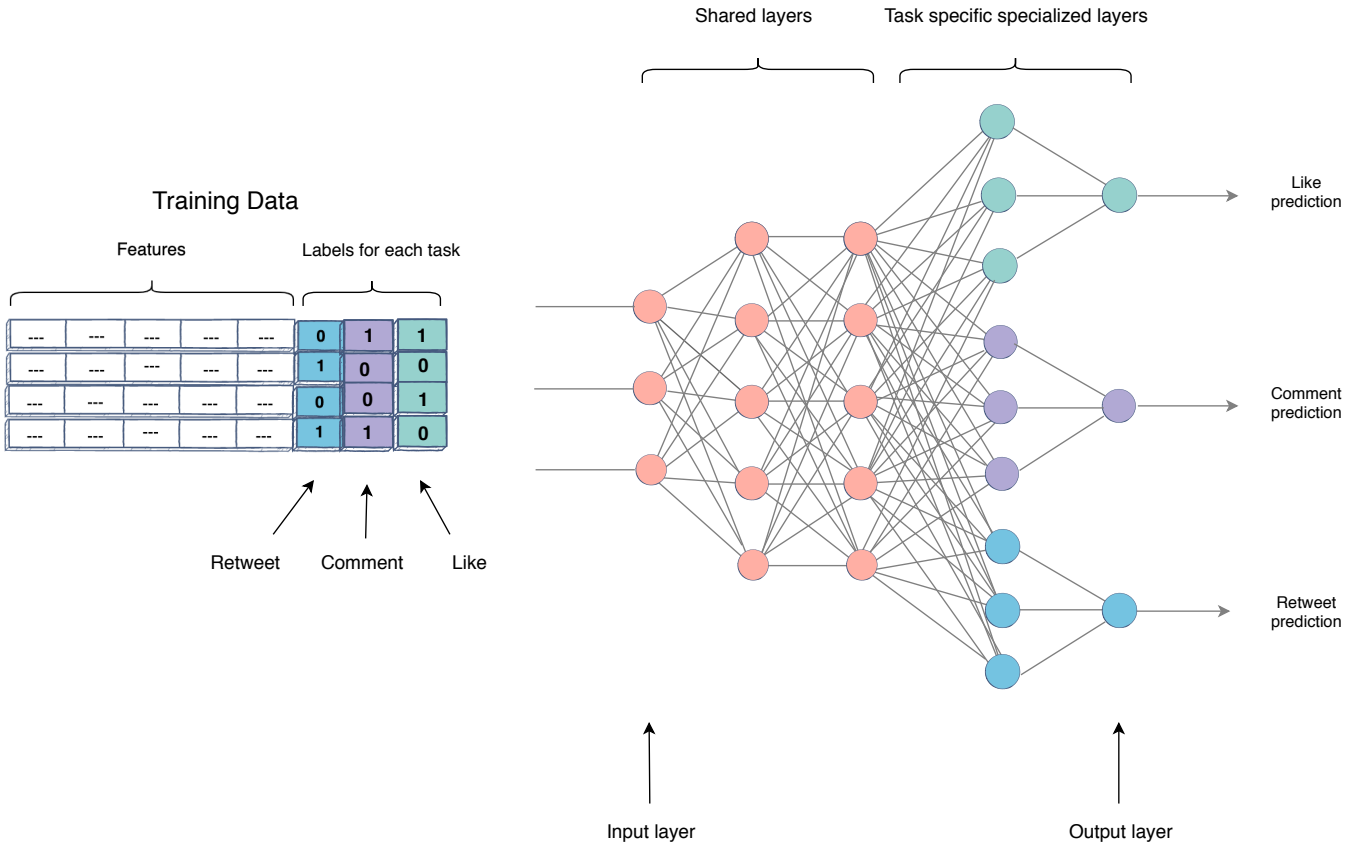
Multi-task neural networks

Likes, comments, and retweets are all different forms of engagement on a Tweet. Therefore, predicting P(like), P(comment) and P (retweet) are similar tasks. When predicting similar tasks, you can use multitask learning. Referring back to the scenario in approach 3 of MART, if you want to predict the probability of a like, it would certainly be helpful for the model to know that this Tweet has been retweeted (knowledge sharing). Hence, you can train a neural network with shared layers (for shared knowledge) appended with specialized layers for each task's prediction. The weights of the shared layers are common to the three tasks. Whereas in the task-specific layer, the network learns information specific to the tasks. The loss function for this model will be the sum of individual losses for all the tasks:

total_loss = like_loss + comment_loss + retweet_loss

The model will then optimize for this joint loss leading to regularization and joint learning.

> 📝 Given the training time is slow for neural networks, training one model (shared layers) would make the overall training time much faster. Moreover, this will also allow us to share all the engagement data across each learning task.



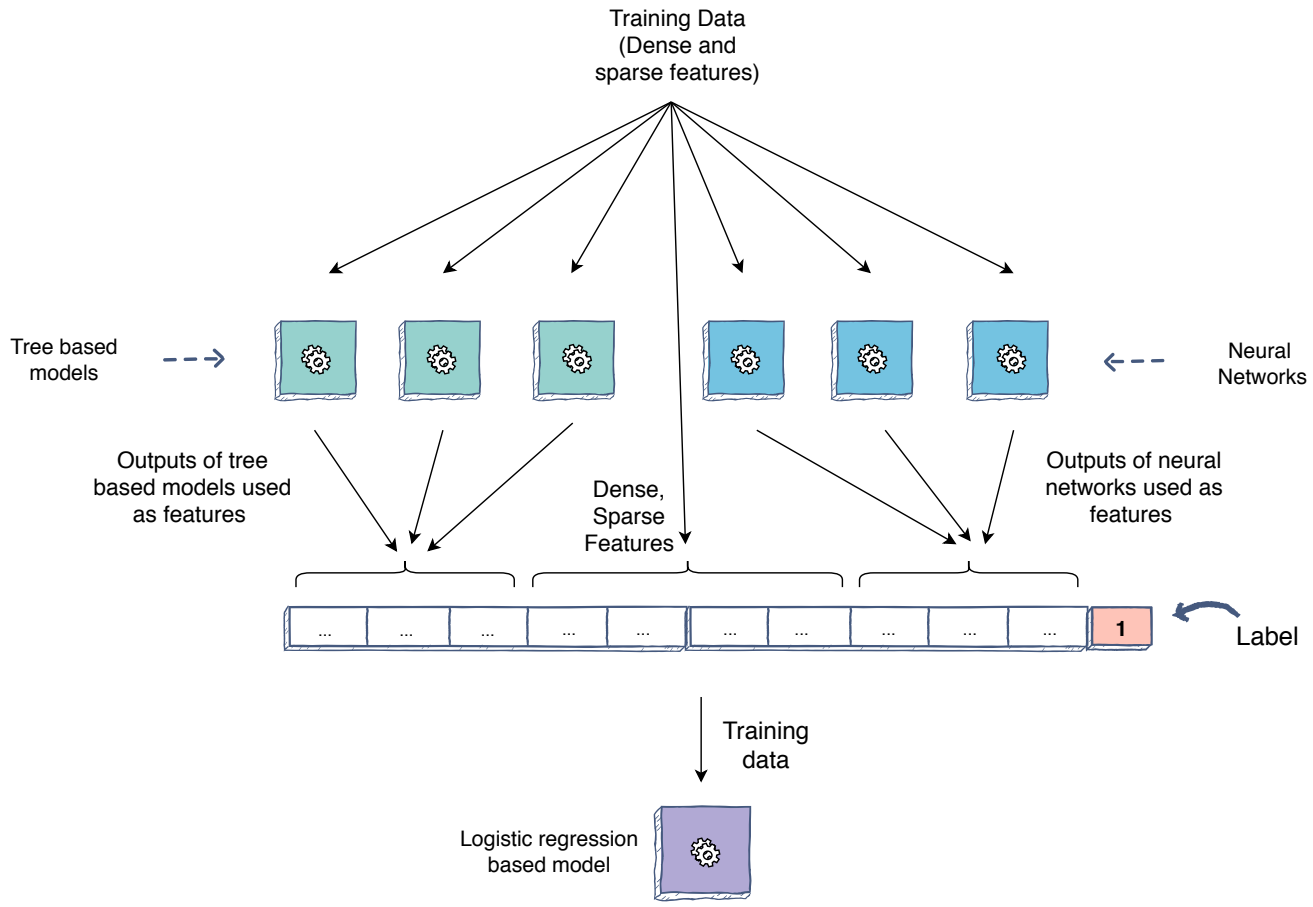
Multi-task neural network configuration: the specialized layers share the initial layers

This approach should be able to perform at least as effective as training separate networks for each task. It should be able to outperform in most cases as we use the shared data and use it across each predictor. Also, one key advantage of using shared layers is that models would be much faster to train than training completely separate deep neural networks for each task.

Stacking models and online learning

One way to outperform the “single model technique approach” is to use multiple models to utilize the power of different techniques by stacking models on top of each other. Let’s go over one such stacking setup that should work out very well for the feed problem.

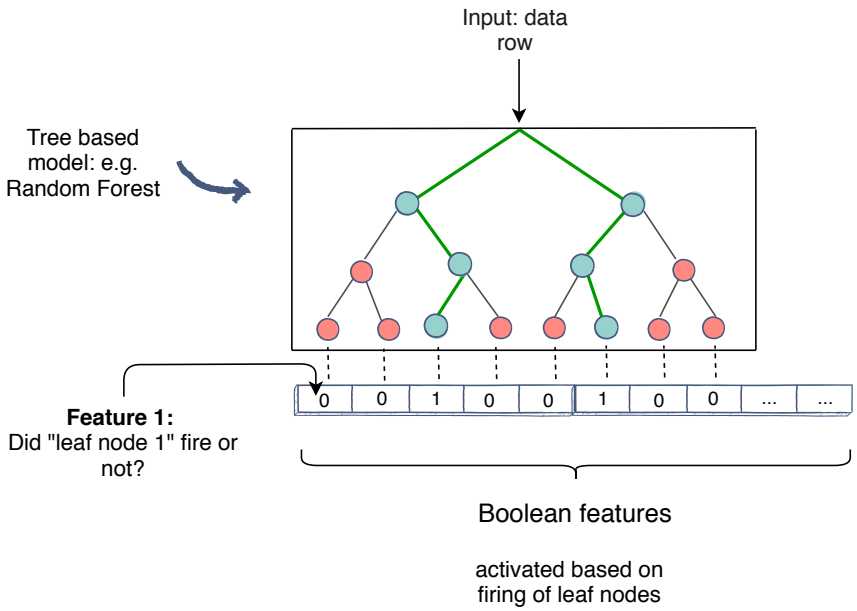
The setup includes training tree-based models and neural networks to generate features that you will utilize in a linear model (logistic regression). The main advantage of doing this is that you can utilize **online learning**, meaning we can continue to update the model with every user action on the live site. You can also use sparse features in our linear model while getting the power of all non-linear relations and functions through features generated by tree-based models and neural networks.



Stacking setup

For trees, you can generate features by using the triggering of leaf nodes, e.g., given two trees with four leaf nodes, each will result in eight features that can be used in logistic regression, as shown below.

How leaf nodes of tree based models produce values for boolean features



Tree based model generating features for the logistic regression model

In the above diagram, each leaf node in the random forest will result in a boolean feature that will be active based on whether the leaf node is triggered or not.

Similarly, for neutral networks, rather than predicting the probability of events, you can just plug-in the output of the last hidden layer as features into the logistic regression models.

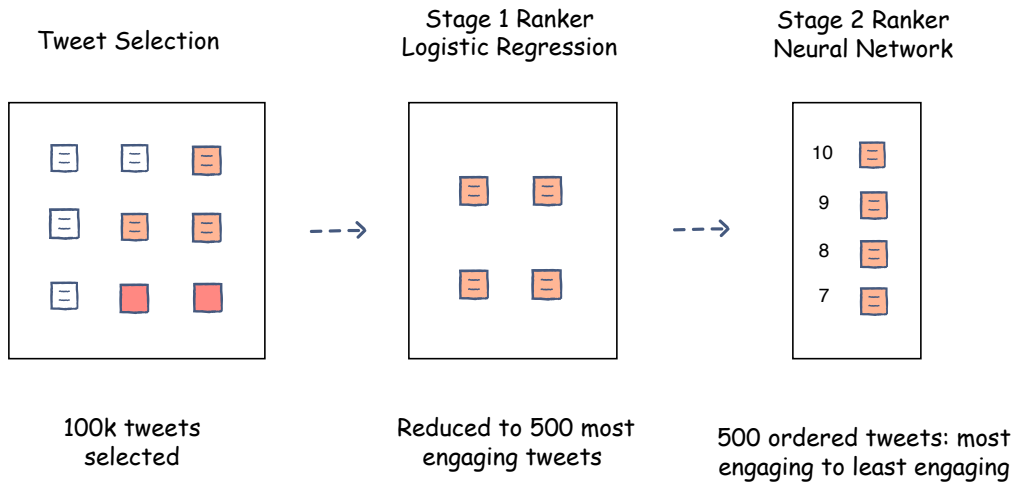
Diversity

Let's learn methods to reduce monotony on the user's Twitter feed.

We'll cover the following

Why do you need diverse Tweets?

Let's assume that you adopted the following modelling option for your Twitter feed ranking system. The Tweet selection component will select one-hundred thousand Tweets for user A's Twitter feed. The stage one ranker will choose the top five-hundred most engaging Tweets for user A. The stage two ranker will then focus on assigning engagement probabilities to these Tweets with a higher degree of accuracy. Finally, the Tweets will be sorted according to the engagement probability scores and will be ready for display on the user's Twitter feed.



The layered model approach

Consider a scenario where the sorted list of Tweets has five consecutive posts by the same author. No, your ranking model hasn't gone bonkers! It has rightfully placed these tweets at the top because:

- The logged-in user and the Tweet's author have frequently interacted with each other's Tweets
- The logged-in user and the Tweet's author have a lot in common like hashtags followed and common followees
- The author is very influential, and their Tweets generally gain a lot of traction

> 📝 This scenario remains the same for any modelling option.

Diversity in Tweets' authors

However, no matter how good of a friend the author is or how interesting their Tweets might be, user A would eventually get bored of seeing Tweets from the same author repeatedly. Hence, you need to introduce diversity with regards to the Tweets' author.

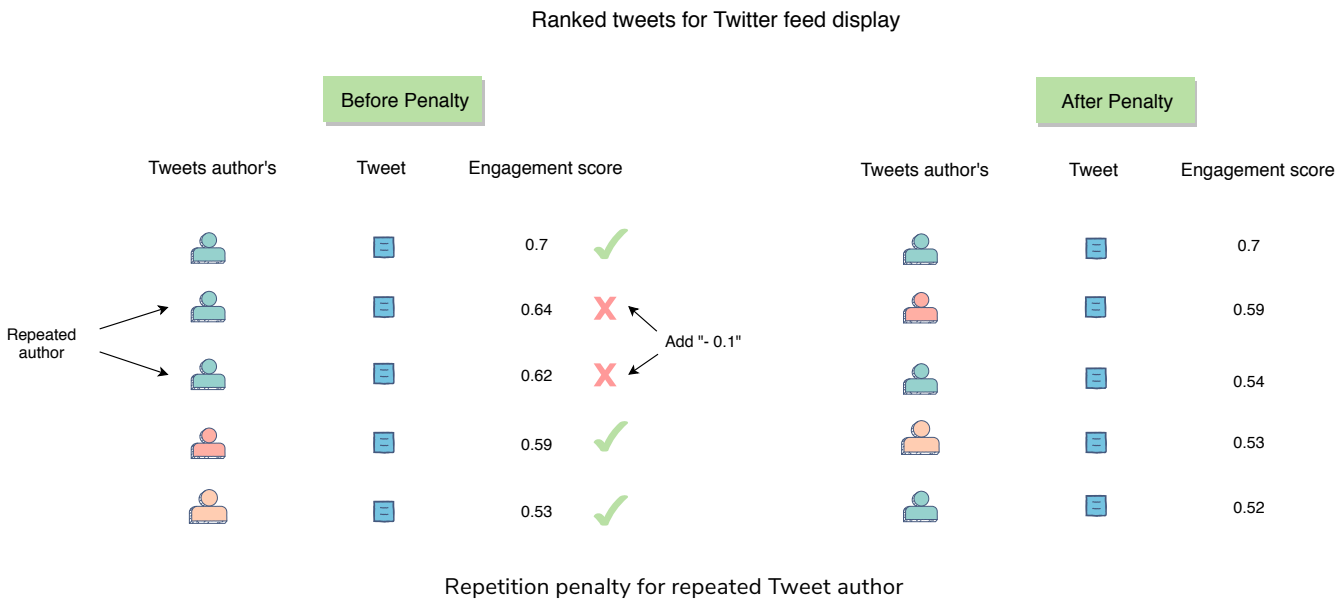
Diversity in tweets' content

Another scenario where we might need to introduce diversity is the Tweet's content. For instance, if your sorted list of Tweets has four consecutive tweets that have videos in them, the user might feel that their feed has too many videos.

Introducing the repetition penalty

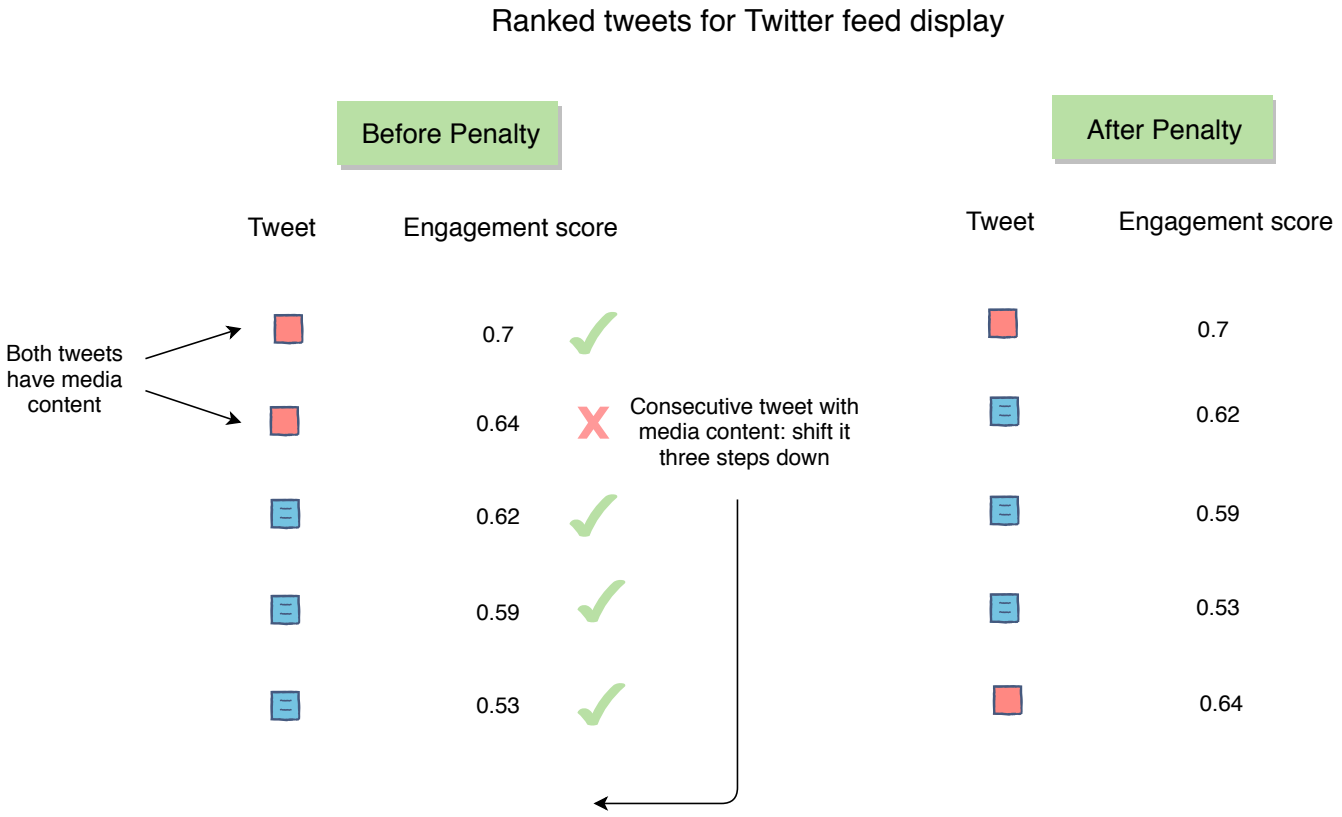
To rid the Twitter feed from a monotonous and repetitive outlook, we will introduce a repetition penalty for repeated Tweet authors and media content in the Tweet.

One way to introduce a repetition penalty could be to add a negative weight to the Tweet's score upon repetition. For instance, in the following diagram, whenever you see the author being repeated, you add a negative weight of -0.1 to the Tweet's score.



Repetition penalty for repeated Tweet author

Another way to achieve the same effect is to bring the Tweet with repetition three steps down in the sorted list. For instance, in the following diagram, when you observe that two consecutive Tweets have media content in them, you bring the latter down by three steps.



Repetition penalty for consecutive Tweets with media content

Back

Ranking

Next

Online Experimentation

Completed

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

Online Experimentation

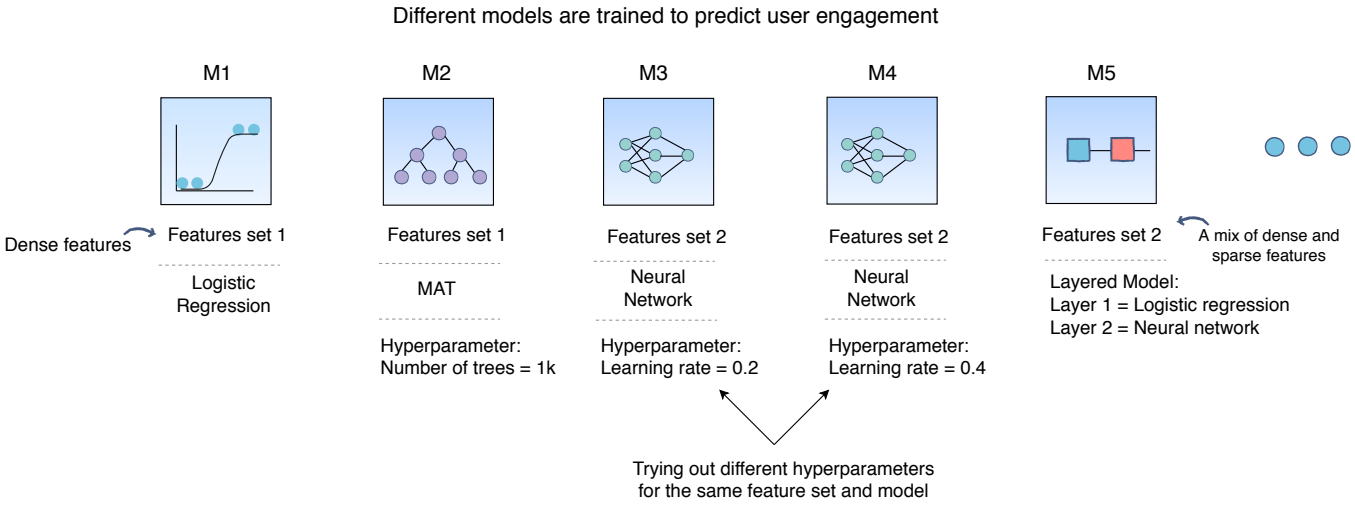
Let's see how to evaluate the model's performance through online experimentation.

We'll cover the following

Let’s look at the steps from training the model to deploying it.

Step 1: Training different models

Earlier, in the training data generation lesson, we discussed a method of splitting the training data for training and validation purposes. After the split, the training data is utilized to train, say, **fifteen** different models, each with a different combination of hyperparameters, features, and machine learning algorithms.

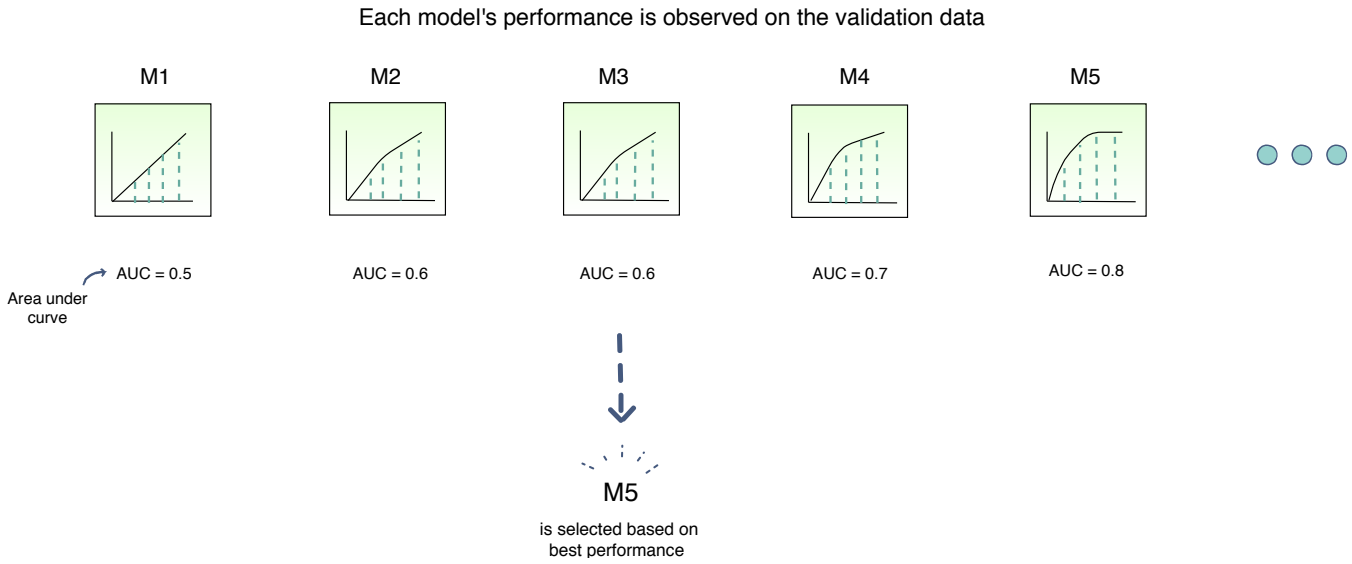


Different models are trained to predict user engagement

The above diagram shows different models that you can train for our tweet engagement prediction problem. Several combinations of feature sets, modeling options, and hyperparameters are tried.

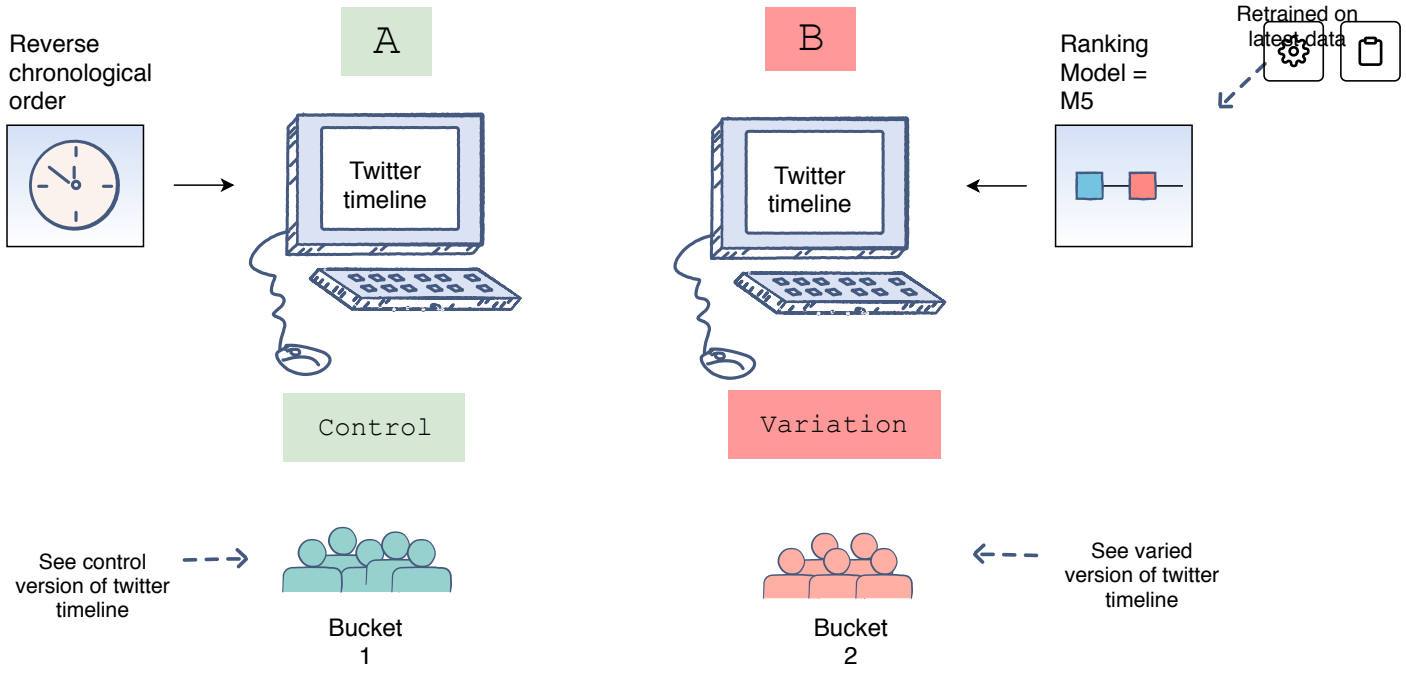
Step 2: Validating models offline

Once these **fifteen** models have been trained, you will use the validation data to select the best model offline. The use of unseen validation data will serve as a sanity check for these models. It will allow us to see if these models can generalise well on unseen data.



Step 3: Online experimentation

Now that you have selected the best model offline, you will use A/B testing to compare the performance of this model with the currently deployed model, which displays the feed in reverse chronological order. You will select 1% of the five-hundred million active users, i.e., five million users for the A/B test. Two buckets of these users will be created each having 2.5 million users. Bucket one users will be shown twitter timelines according to the time-based model; this will be the control group. Bucket two users will be shown the Twitter timeline according to the new ranking model.



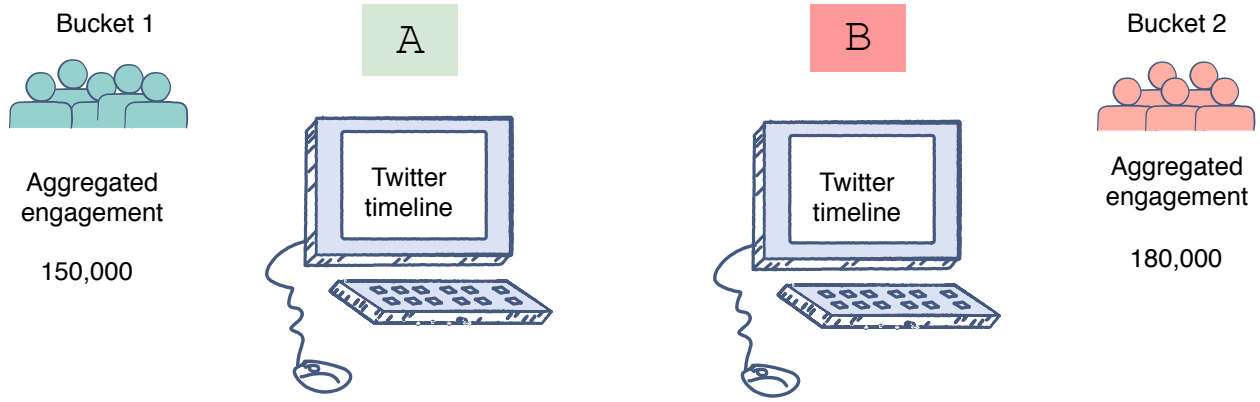
Bucket one users see the control version, whereas Bucket two users see the varied version of the Twitter timeline

However, before you perform this A/B test, you need to retrain the ranking model.

Recall that you withheld the most recent partition of the training data to use for validation and testing. This was done to check if the model would be able to predict future engagements on tweets given the historical data. However, now that you have performed the validation and testing, you need to retrain the model using the recent partitions of training data so that it *captures the most recent phenomena*.

Step 4: To deploy or not to deploy

The results of the A/B tests will help decide whether you should deploy the new ranking model across the platform.



Engagement aggregates for both buckets of users

You can observe that the Twitter feeds generated by the new ranking model had thirty (180k-150k) more engagements.

Increase in engagement (*gain*) = $\frac{180{,}000-150{,}000}{150{,}000}$ * 100 = 20%

This model is clearly able to outperform the current production, or live state. You should use statistical significance (like p-value) to ensure that the gain is real.

> 📝 Learn more about online experimentation in this lesson (https://www.educative.io/collection/page/10370001/6237869033127936/5766564994351104).

Another aspect to consider when deciding to launch the model on production, especially for *smaller gains*, is the increase in complexity. If the new model increases the complexity of the system significantly without any significant gains, you should not deploy it.

> 📝 You should go for complex solutions (based on new features, or data, etc.) only if you anticipate it to bring larger gains in the future.

To wrap up, if, after an A/B experiment, you see an engagement gain by the model that is statistically significant and worth the complexity it adds to the system, it makes sense to replace the current live system with the new model.

Back

Diversity

Next

Problem Statement

Completed