

# Deciding on ML

Before you get elbow-deep in your data, it is important to set yourself up for success. The following section outlines several things to think about before trying to frame a problem for machine learning.

## Start Clearly and Simply

In plain terms, what would you like your ML model to do?

---

### Example Statement

---

We want the ML model to predict how popular a video just uploaded will become in the future.

---

At this point, the statement can be qualitative, but make sure this captures your real goal, not an indirect goal.

## What is Your Ideal Outcome?

Adding your ML model to your system should produce a desirable outcome. This outcome may be quite different from how you assess the model and its quality.

---

### Examples

---

Transcoding	Our ideal outcome is to use fewer resources on transcoding less popular videos. Transcoding is the process of converting the user's uploaded content into a more efficient format that YouTube shows to users. More than 1/3 of videos on YouTube are watched fewer than 10 times! If we can identify these videos and only prepare lower resolution versions of them, we could save a lot of resources.
-------------	--

---

Video Recommendations	Our ideal outcome is to suggest videos that people find useful, entertaining, and worth their time.
-----------------------	---

---

We could have different outcomes from the same model and we could use this prediction to do different things. We want to make sure our outcome states what we actually care about in the product.

Don't limit yourself to metrics for which your product has already been optimizing. Instead, focus on the larger objective of your product or service.

# Success and Failure Metrics

## Quantify It

How will you know if your system has succeeded or failed?

Your success and failure metrics should be phrased independently of your evaluation metrics such as precision, recall, or **AUC**

(/machine-learning/crash-course/classification/roc-and-auc#AUC). Instead, specify the anticipated outcomes. Set your success metrics before you begin, to prevent sunk costs from motivating you to launch a mediocre model.

---

### Examples

Transcoding	A success metric is CPU resource utilization. Success means reducing the CPU cost for transcoding by 35%. Failure means the cost reduction is less than the CPU costs for training and serving the model.
Video Recommendations	A success metric is the number of popular videos properly predicted by the model. Success means predicting 95% of the most popular videos as measured by watch time within 28 days of being uploaded. Failure means the number of popular videos properly predicted is no better than current heuristics.

---

## Are the Metrics Measurable?

A *measurable* metric provides enough information for successful real-world evaluation. For example, a system monitoring the health of an orchard might want to reduce the fraction of sick trees that die. But if you can't measure how many trees are sick, this is not a useful metric.

Ask the following questions:

- How will you measure your metrics?
- When can you measure your metrics?
- How long will it take to know whether your new ML system is a success or failure?

Ideally, you want to fail fast. Watch for too little signal in your data, or data that isn't predictive, to determine if your hypothesis might be wrong. Failing fast will enable you to revise your hypothesis earlier in the process and prevent lost time.

---

### Example

---

You think you can determine which videos will be watched based on a user's location. It seems possible, but when you try it out, the signal is too weak or there is too much noise and it doesn't work. How long will you stick with that hypothesis?

---

Consider engineering and maintenance costs over the long-term.

## Other Failure Scenarios

Watch for failures that are not related to your success metric. For instance, a video suggestion system that always recommends "clickbait" videos is *not* successful at providing a high-quality viewing experience.

## What Output Would You Like the ML Model to Produce?

Revisiting this table, which type of output are you looking for: a number, a label, a cluster, or something else?

---

Type of ML Problem	Description	Example
Classification	Pick one of N labels	cat, dog, horse, or bear
Regression	Predict numerical values	click-through rate
Clustering	Group similar examples	most relevant documents (unsupervised)
Association rule learning	Infer likely association patterns in data	If you buy hamburger buns, you're likely to buy hamburgers (unsupervised)
Structured output	Create complex output	natural language parse trees, image recognition bounding boxes

---

Certain ML problems can fail with one type of output but succeed with another.

## Properties of Good Output

**The Output Must Be Quantifiable with a Clear Definition that the Machine can Produce.**

---

**Example**

---

Did the user enjoy watching the video? vs. Did the user share the video?

---

---

You cannot tell whether a user enjoyed a video without asking the user. If you can't ask the user, you'll need to use a **proxy label** ([/machine-learning/glossary#proxy\\_labels](/machine-learning/glossary#proxy_labels)) instead. That is, you'll need to use a substitute label that will stand in for the real thing. The degree to which users shared the video is a pretty good proxy label. True, sharing isn't a perfect approximation of enjoyment because users might share a video for reasons other than enjoyment (for example, to make fun of a video). However, sharing is quantifiable, trackable, and provides a decent predictive signal.

### The Output Should Be Connected To Your Ideal Outcome.

Your model will optimize the output. Therefore, make sure that the output is truly something you care about. Proxy labels are frequently necessary, since we can't always measure our ideal outcome directly. However, the stronger the connection between our label and our true outcome, the more confident we can be that we're optimizing the right thing.

---

Output	Ideal Outcome
Predict whether the user will share the article.	Show the user articles they will like.
Predict whether the video will be popular.	Suggest videos that people find useful and worth their time.
Predict whether the user will install an app from an app store.	Find apps the user will use often and enjoy.
<b>Make sure there is a strong connection between the two!</b>	

---

### Can You Obtain Example Outputs to Use for Training Data?

*How and from what source?*

Supervised machine learning relies on labeled data. If it is difficult to obtain example outputs for training, you may need to revisit your responses to past exercises to reformulate your problem and goals so you can train a model on your data. Your output examples may need to be engineered, as in the example above, which turns watch time into a percentile.

### Using the Output

Your model can make predictions at either of two points:

- In real time, in response to user activity (online).

- As a batch and then cached (offline).

How will your product use the predictions? Remember, you want to use ML to make decisions, not just predictions. How will you turn your model's predicted outcome into a decision?

---

### Example

---

We'll predict a video's popularity when the new video is uploaded. The outcome will help determine the transcoding algorithm for the video.

---

Think about how the implementation would look in pseudocode. For example, say you are building a model to predict the likelihood a user will click on a notification sent by your app, and you want to only send notifications to users likely to click them. Your pseudocode could look like the following:

```
click_probability = call_model(user)
if click_probability > 0.05:
    send_notification(user)
```

The preceding step helps turn your model's predictions into decisions.

Next, figure out where in your architecture this code would live. Doing so will help you address the following important questions:

- What data does your code have access to when it needs to call the model?
- What are your latency requirements? Do you need to run quickly to avoid lagging in your UI, or are you running without a user waiting for your model?

The requirements discovered by answering the preceding questions can affect the features used in your model. You can only train on features that you would also have access to when you call the model; otherwise, there's no point in training on those features since you can't use your learned knowledge at serving time. Additionally, make sure you can get each feature within your latency requirements. Using data from remote services may make certain features expensive from a latency standpoint. For example, if you wanted to use the current weather as a feature for your camera app, you might find that the latency from looking up the current weather is too high and causes unacceptable slowdown in your app.

Lastly, be wary of using out-of-date data. Note that training data is sometimes several days old. When running on live traffic, you might encounter cases where data you'd like to use

isn't yet available. Perhaps your database only updates a user's history every 30 minutes, so you don't have access to their most recent data when making your prediction.

## Bad Objectives

When set up properly, ML systems are very good at pursuing the objectives that they're given. Conversely, ML systems can produce unintended outcomes when given the wrong objectives. Therefore, carefully consider how the objectives of the system will help solve your problem.

Think back to our model that predicts the next video someone will watch on YouTube. Expand each section below by clicking the plus icon to learn why each successive proposed objective is not good for this problem.

### Maximize Click Rate

Users may click on something but then not stay on it very long. This optimizes clickbait, so maybe we should try something else.

### Maximize Watch Time

Users may watch a long time, but then exit the session.

---

#### Example

A Minecraft video gets a 0.1% audience that watches video for 3 hours, 8% of whom watch another video for 5 minutes, while the rest quit watching altogether. The system is maximizing watch time, so the users' "watch next" list will consist solely of long Minecraft videos.

---

It is important to note that multiple short watches can be just as good as one long watch and can even increase the overall session watch time. On to the next objective!

### Maximize Session Watch Time

This model still favors longer videos, which is still a problem. This model does mine particular interests really well. For example, if a user watches a video of LeBron James dunking, the system will show them every LeBron dunking video. Ever. The video recommendation system is *really* good at that, but user experience suffers. Each person can see YouTube as the place to go to watch a specific type of video. Diversity suffers.

## Increase Diversity & Maximize Session Watch Time

What problems do you think might arise from this objective? Keep in mind Goodhart's law ([https://wikipedia.org/wiki/Goodhart%27s\\_law](https://wikipedia.org/wiki/Goodhart%27s_law)) , "When a measure becomes a target, it ceases to be a good measure."

## Heuristics

How might you solve your problem without ML?

Suppose you need to deliver a product tomorrow, and you have only time enough to hard-code the business logic. You could try a heuristic (non-ML solution) like the following:

---

### Example

---


Consider people who uploaded popular videos in the past. Assume that new videos uploaded by these people will also become popular.

---


The preceding heuristic might not be the world's greatest heuristic, but it does provide a baseline. Never launch a fancy ML model that can't beat a heuristic. The exercise of making a heuristic often will help you identify good signals in your ML model.

Non-ML solutions can sometimes be simpler to maintain than ML solutions.

[Previous](#)

 [Hard ML Problems](/machine-learning/problem-framing/hard) (/machine-learning/problem-framing/hard)

[Next](#)

[Try it Yourself](/machine-learning/problem-framing/try-it/framing-exercise) (/machine-learning/problem-framing/try-it/framing-exercise) 

Except as otherwise noted, the content of this page is licensed under the [Creative Commons Attribution 4.0 License](https://creativecommons.org/licenses/by/4.0/) (<https://creativecommons.org/licenses/by/4.0/>), and code samples are licensed under the [Apache 2.0 License](https://www.apache.org/licenses/LICENSE-2.0) (<https://www.apache.org/licenses/LICENSE-2.0>). For details, see the [Google Developers Site Policies](https://developers.google.com/site-policies) (<https://developers.google.com/site-policies>). Java is a registered trademark of Oracle and/or its affiliates.

Last updated 2020-02-05 UTC.