Filter

Overview
Introduction
Problem Framing

ML Problem Framing
Understand the problem
Framing an ML problem

Conclusion
Implementing a model
Summary and next steps

Implementing a model

Send feedback

When implementing a model, start simple. Most of the work in ML is on the data side, so getting a full pipeline running for a complex model is harder than iterating on the model itself. After setting up your data pipeline and implementing a simple model that uses a few features, you can iterate on creating a better model.

Simple models provide a good baseline, even if you don't end up launching them. In fact, using a simple model is probably better than you think. Starting simple helps you determine whether or not a complex model is even justified.

Train your own model versus using an already trained model

Trained models exist for a variety of use cases and offer many advantages. However, trained models only really work when the label and features match your dataset exactly. For example, if a trained model uses 25 features and your dataset only includes 24 of them, the trained model will most likely make bad predictions.

Commonly, ML practitioners use matching subsections of inputs from a trained model for fine-tuning or transfer learning. If a trained model doesn't exist for your particular use case, consider using subsections from a trained model when training your own.

Note: If your solution is a generative AI model, you'll almost always fine-tune a pre-trained model instead of training your own.

For information on trained models, see

- Trained models from TensorFlow Hub
- Trained models from Kaggle

Monitoring

During problem framing, consider the monitoring and alerting infrastructure your ML solution needs.

Model deployment

In some cases, a newly trained model might be worse than the model currently in production. If it is, you'll want to prevent it from being released into production and get an alert that your automated deployment has failed.

Training-serving skew

If any of the incoming features used for inference have values that fall outside the distribution range of the data used in training, you'll want to be alerted because it's likely the model will make poor predictions. For example, if your model was trained to predict temperatures for equatorial cities at sea level, then your serving system should alert you of incoming data with latitudes and longitudes, and/or altitudes outside the range the model was trained on. Conversely, the serving system should alert you if the model is making predictions that are outside the distribution range that was seen during training.

Inference server

If you're providing inferences through an RPC system, you'll want to monitor the RPC server itself and get an alert if it stops providing inferences.

Previous
Framing an ML problem
Next
Summary and next steps

Was this helpful?
Send feedback