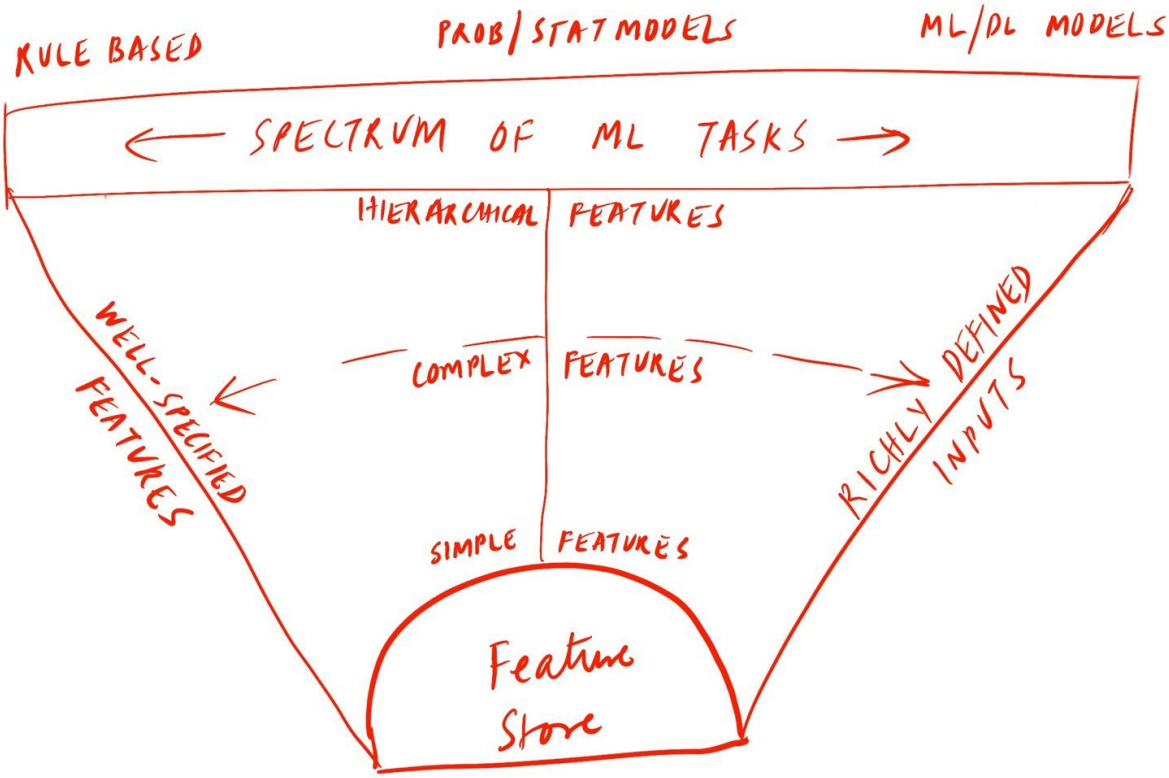
Hierarchical Features and their Importance in Feature Engineering

By Scribble Data

Feature engineering is both a central task in machine learning engineering and is also arguably the most complex task. Data scientists who build models that need to be deployed at large scales, across functional, technical, geographic, demographic and other categories have to reason about how they choose the features for the models. Despite the divergent considerations of different data scientists, they may often rely on variations of a specific feature that occur as a feature hierarchy. In this post, we will explore how hierarchical features can add value to the data science lifecycle, and how feature hierarchies can improve MLOps productivity.



What are Hierarchical Features and why are they important?

Let's take the case of a data science team building customer churn models for a retail or e-commerce company. This data science team may be interested in evaluating location-wise churn hypotheses, i.e., whether customers from a certain location are more likely to churn, than those from other locations. Customers could come either from different localities, cities, states, countries, regions, etc., and as a consequence, our definition of the "location" feature has to incorporate these variations. Let's now consider a different example - a data scientist statistically modeling the failures of a product or service. This data scientist may be interested in building failure prediction models of the product or service for specific demographics, and for specific sub-categories of products. One category of products here could be electronic goods, and under this category, we could have different sub-categories such as smartphones, tablets or laptops. Accordingly, the data scientist may be interested in building separate models for laptops, smartphones or tablets. In the real world, enterprises may be interested in building hundreds or even thousands of such models, each with their own variations in terms of the same core features we discussed.

Hierarchical Features for building better models

In both of these situations, the data science teams require the same essential data in a feature - in the former case, it is location information, and in the latter case, it is the category of product. However, depending on the kind of model they're building, they will require a variation on the feature. This is especially important when different data scientists may approach the same hypothesis from different lenses, although the underlying feature to be computed is roughly the same. Naturally, this opens up the possibility of introducing *hierarchical features* in feature stores. If we were to define these features in a hierarchy where closely related features could be computed together, it could enable data scientists to access a range of features through a single feature specification. This in turn can speed up their model development tasks.

Hierarchical features could be characterized as having intermediary complexity. Very well defined features which can be used for rule-based decision making require considerable clarity in the feature definition and specification, whereas input features to deep learning models such as for computer vision generally don't require as much complexity in feature development workflows.

Hierarchical features for better data science productivity

Let's look at the impact of incorporating hierarchical features in the entire ML lifecycle. It is estimated that up to 80% of the time spent in taking machine learning ideas from concept to working production scale models, is spent on building data pipelines and features, with only the

remaining 20% of the effort spent for generating and validating hypotheses, and in experimenting on and developing machine learning models. This means that time saved in building additional features or on variations on features can contribute significantly to reducing the time required to take models from concept to production. Another reason to consider the hierarchical feature is the importance of having a fall-back option when computing features for different models. When we have partially computed features, these intermediary points in the feature computation can be used for rebuilding features with a specific scope, or during contingencies such as a failover. There are also sound computational reasons for building hierarchical features into feature stores that serve hundreds or thousands of models, such as the ability to federate the computation of models, or borrow feature specifications from relevant models or relevant contexts. We will discuss more of these ideas in forthcoming blog posts.

Required Feature Store Capabilities

Organizations will have numerous models that vary in approach and complexity. The feature engineering subsystem, Feature Store, should have a number capabilities to meet the needs. First, it should support diverse features - simple specification based features to complex model-based features. It should have the required **programming abstractions and flexibility**. Second, it should have **metadata management** capability to track the many levels, versions, and dependencies of the features. Third, it should have a **control interface** to be able to verify, backfill, erase, and reproduce features within any scope. Last, it should be **robust** in architecture to enable scaling and stability.