Discover Feature Engineering, How to Engineer Features and How to Get Good at It

by **Jason Brownlee** on September 26, 2014 in **Machine Learning Process**

Feature engineering is an informal topic, but one that is absolutely known and agreed to be key to success in applied machine learning.

In creating this guide I went wide and deep and synthesized all of the material I could.

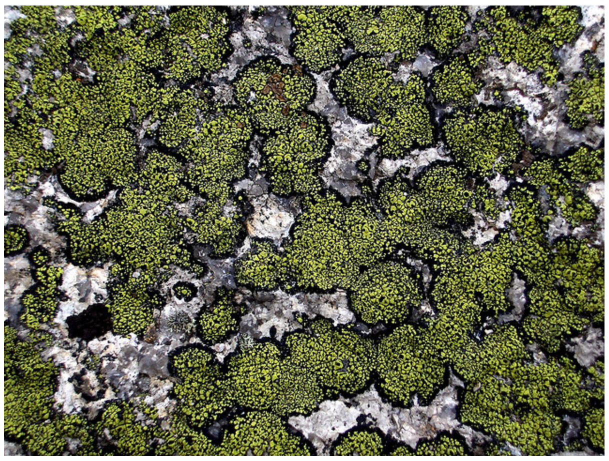
You will discover what feature engineering is, what problem it solves, why it matters, how to engineer features, who is doing it well and where you can go to learn more and get good at it.

If you read one article on feature engineering, I want it to be this one.

> *feature engineering is another topic which doesn't seem to merit any review papers or books, or even chapters in books, but it is absolutely vital to ML success. […] Much of the success of machine learning is actually success in engineering features that a learner can understand.*

— Scott Locklin, in "Neglected machine learning ideas"

Problem that Feature Engineering Solves


Feature engineering is hard.
Photo by Vik Nanda, some rights reserved

When your goal is to get the best possible results from a predictive model, you need to get the most from what you have.

This includes getting the best results from the algorithms you are using. It also involves getting the most out of the data for your algorithms to work with.

How do you get the most out of your data for predictive modeling?

This is the problem that the process and practice of feature engineering solves.

Actually the success of all Machine Learning algorithms depends on how you present the data.

— Mohammad Pezeshki, answer to “What are some general tips on feature selection and engineering that every data scientist should know?”

Importance of Feature Engineering

The features in your data will directly influence the predictive models you use and the results you can achieve.

You can say that: the better the features that you prepare and choose, the better the results you will achieve. It is true, but it also misleading.

The results you achieve are a factor of the model you choose, the data you have available and the features you prepared. Even your framing of the problem and objective measures you’re using to estimate accuracy play a part. Your results are dependent on many inter-dependent properties.

You need great features that describe the structures inherent in your data.

Better features means flexibility.

You can choose “the wrong models” (less than optimal) and still get good results. Most models can pick up on good structure in data. The flexibility of good features will allow you to use less complex models that are faster to run, easier to understand and easier to maintain. This is very desirable.

Better features means simpler models.

With well engineered features, you can choose “the wrong parameters” (less than optimal) and still get good results, for much the same reasons. You do not need to work as hard to pick the right models and the most optimized parameters.

With good features, you are closer to the underlying problem and a representation of all the data you have available and could use to best characterize that underlying problem.

Better features means better results.

The algorithms we used are very standard for Kagglers. […] We spent most of our efforts in feature engineering.

— Xavier Conort, on “Q&A with Xavier Conort” on winning the Flight Quest challenge on Kaggle

What is Feature Engineering?

Here is how I define feature engineering:

Feature engineering is the process of transforming raw data into features that better represent the underlying problem to the predictive models, resulting in improved model accuracy on unseen data.

You can see the dependencies in this definition:

- The performance measures you've chosen (RMSE? AUC?)
- The framing of the problem (classification? regression?)
- The predictive models you're using (SVM?)
- The raw data you have selected and prepared (samples? formatting? cleaning?)

> *feature engineering is manually designing what the input x's should be*

— Tomasz Malisiewicz, answer to "What is feature engineering?"

Feature Engineering is a Representation Problem

Machine learning algorithms learn a solution to a problem from sample data.

In this context, feature engineering asks: what is the best representation of the sample data to learn a solution to your problem?

It's deep. Doing well in machine learning, even in artificial intelligence in general comes back to representation problems. It's hard stuff, perhaps unknowable (or at best intractable) to know the best representation to use, *a priori*.

> *you have to turn your inputs into things the algorithm can understand*

— Shayne Miel, answer to "What is the intuitive explanation of feature engineering in machine learning?"

Feature Engineering is an Art

It is an art like engineering is an art, like programming is an art, like medicine is an art.

There are well defined procedures that are methodical, provable and understood.

The data is a variable and is different every time. You get good at deciding which procedures to use and when, by practice. By empirical apprenticeship. Like engineering, like programming, like medicine, like machine learning in general.

Mastery of feature engineering comes with hands on practice, and study of what others that are doing well are practicing.

> *…some machine learning projects succeed and some fail. What makes the difference? Easily the most important factor is the features used.*

— Pedro Domingos, in "A Few Useful Things to Know about Machine Learning" (PDF)

Sub-Problems of Feature Engineering

It is common to think of feature engineering as one thing.

For example, for a long time for me, feature engineering was feature construction.

I would think to myself "*I'm doing feature engineering now*" and I would pursue the question "*How can I decompose or aggregate raw data to better describe the underlying problem?*" The goal was right, but the approach was one of a many.

In this section we look at these many approaches and the specific sub-problems that they are intended to address. Each could be an in depth article of their own as they are large and important areas of practice and study.

Feature: An attribute useful for your modeling task

Let's start with data and what is a feature.

Tabular data is described in terms of observations or instances (rows) that are made up of variables or attributes (columns). An attribute could be a feature.

The idea of a feature, separate from an attribute, makes more sense in the context of a problem. A feature is an attribute that is useful or meaningful to your problem. It is an important part of an observation for learning about the structure of the problem that is being modeled.

I use "*meaningful*" to discriminate attributes from features. Some might not. I think there is no such thing as a non-meaningful feature. If a feature has no impact on the problem, it is not part of the problem.

In computer vision, an image is an observation, but a feature could be a line in the image. In natural language processing, a document or a tweet could be an observation, and a phrase or word count could be a feature. In speech recognition, an utterance could be an observation, but a feature might be a single word or phoneme.

Feature Importance: An estimate of the usefulness of a feature

You can objectively estimate the usefulness of features.

This can be helpful as a pre-cursor to selecting features. Features are allocated scores and can then be ranked by their scores. Those features with the highest scores can be selected for inclusion in the training dataset, whereas those remaining can be ignored.

Feature importance scores can also provide you with information that you can use to extract or construct new features, similar but different to those that have been estimated to be useful.

A feature may be important if it is highly correlated with the dependent variable (the thing being predicted). Correlation coefficients and other univariate (each attribute is considered independently) methods are common methods.

More complex predictive modeling algorithms perform feature importance and selection internally while constructing their model. Some examples include MARS, Random Forest and Gradient Boosted Machines. These models can also report on the variable importance determined during the model preparation process.

Feature Extraction: The automatic construction of new features from raw data

Some observations are far too voluminous in their raw state to be modeled by predictive modeling algorithms directly.

Common examples include image, audio, and textual data, but could just as easily include tabular data with millions of attributes.

Feature extraction is a process of automatically reducing the dimensionality of these types of observations into a much smaller set that can be modelled.

For tabular data, this might include projection methods like Principal Component Analysis and unsupervised clustering methods. For image data, this might include line or edge detection. Depending on the domain, image, video and audio observations lend themselves to many of the same types of DSP methods.

Key to feature extraction is that the methods are automatic (although may need to be designed and constructed from simpler methods) and solve the problem of unmanageably high dimensional data, most typically used for analog observations stored in digital formats.

Feature Selection: From many features to a few that are useful

Not all features are created equal.

Those attributes that are irrelevant to the problem need to be removed. There will be some features that will be more important than others to the model accuracy. There will also be features that will be redundant in the context of other features.

Feature selection addresses these problems by automatically selecting a subset that are most useful to the problem.

Feature selection algorithms may use a scoring method to rank and choose features, such as correlation or other feature importance methods.

More advanced methods may search subsets of features by trial and error, creating and evaluating models automatically in pursuit of the objectively most predictive sub-group of features.

There are also methods that bake in feature selection or get it as a side effect of the model. Stepwise regression is an example of an algorithm that automatically performs feature selection as part of the model construction process.

Regularization methods like LASSO and ridge regression may also be considered algorithms with feature selection baked in, as they actively seek to remove or discount the contribution of features as part of the model building process.

Read more in the post: An Introduction to Feature Selection.

Feature Construction: The manual construction of new features from raw data

The best results come down to you, the practitioner, crafting the features.

Feature importance and selection can inform you about the objective utility of features, but those features have to come from somewhere.

You need to manually create them. This requires spending a lot of time with actual sample data (not aggregates) and thinking about the underlying form of the problem, structures in the data and how best to expose them to predictive modeling algorithms.

With tabular data, it often means a mixture of aggregating or combining features to create new features, and decomposing or splitting features to create new features.

With textual data, it often means devising document or context specific indicators relevant to the problem. With image data, it can often mean enormous amounts of time prescribing automatic filters to pick out relevant structures.

This is the part of feature engineering that is often talked the most about as an artform, the part that is attributed the importance and signalled as the differentiator in competitive machine learning.

It is manual, it is slow, it requires lots of human brain power, and it makes a big difference.

> *Feature engineering and feature selection are not mutually exclusive. They are both useful. I'd say feature engineering is more important though, especially because you can't really automate it.*

— Robert Neuhaus, answer to "Which do you think improves accuracy more, feature selection or feature engineering?"

Feature Learning: The automatic identification and use of features in raw data

Can we avoid the manual load of prescribing how to construct or extract features from raw data?

Representation learning or feature learning is an effort towards this goal.

Modern deep learning methods are achieving some success in this area, such as autoencoders and restricted Boltzmann machines. They have been shown to automatically and in a unsupervised or semi-supervised way, learn abstract representations of features (a compressed form), that in turn have supported state-of-the-art results in domains such as speech recognition, image classification, object recognition and other areas.

We do not have automatic feature extraction or construction, yet, and we will probably never have automatic feature engineering.

The abstract representations are prepared automatically, but you cannot understand and leverage what has been learned, other than in a black-box manner. They cannot (yet, or easily) inform you and the process on how to create more similar and different features like those that are doing well, on a given problem or on similar problems in the future. The acquired skill is trapped.

Nevertheless, it's fascinating, exciting and an important and modern part of feature engineering.

Process of Feature Engineering

Feature engineering is best understood in the broader process of applied machine learning.

You need this context.

Process of Machine Learning

The process of applied machine learning (for lack of a better name) that in a broad brush sense involves lots of activities. Up front is problem definition, next is data selection and preparation, in the middle is model preparation, evaluation and tuning and at the end is the presentation of results.

Process descriptions like data mining and KDD help to better understand the tasks and subtasks. You can pick and choose and phrase the process the way you like. I've talked a lot about this before.

A picture relevant to our discussion on feature engineering is the front-middle of this process. It might look something like the following:

1. (tasks before here…)
2. **Select Data**: Integrate data, de-normalize it into a dataset, collect it together.
3. **Preprocess Data**: Format it, clean it, sample it so you can work with it.
4. **Transform Data**: *Feature Engineer happens here*.
5. **Model Data**: Create models, evaluate them and tune them.
6. (tasks after here…)

The traditional idea of "*Transforming Data*" from a raw state to a state suitable for modeling is where feature engineering fits in. Transform data and feature engineering may in fact be synonyms.

This picture helps in a few ways.

You can see that before feature engineering, we are munging out data into a format we can even look at, and just before that we are collating and denormalizing data from databases into some kind of central picture.

We can, and should go back through these steps as we identify new perspectives on the data.

For example, we may have an attribute that is an aggregate field, like a sum. Rather than a single sum, we may decide to create features to describe the quantity by time interval, such as season. We need to step backward in the process through Preprocessing and even Selecting data to get access to the "real raw data" and create this feature.

We can see that feature engineering is followed by modeling.

It suggests a strong interaction with modeling, reminding us of the interplay of devising features and testing them against the coalface of our test harness and final performance measures.

This also suggests we may need to leave the data in a form suitable for the chosen modeling algorithm, such as normalize or standardize the features as a final step. This sounds like a preprocessing step, it probably is, but it helps us consider what types of finishing touches are needed to the data before effective modeling.

Iterative Process of Feature Engineering

Knowing where feature engineering fits into the context of the process of applied machine learning highlights that it does not standalone.

It is an iterative process that interplays with data selection and model evaluation, again and again, until we run out of time on our problem.

The process might look as follows:

1. **Brainstorm features**: Really get into the problem, look at a lot of data, study feature engineering on other problems and see what you can steal.
2. **Devise features**: Depends on your problem, but you may use automatic feature extraction, manual feature construction and mixtures of the two.
3. **Select features**: Use different feature importance scorings and feature selection methods to prepare one or more "views" for your models to operate upon.
4. **Evaluate models**: Estimate model accuracy on unseen data using the chosen features.

You need a well defined problem so that you know when to stop this process and move on to trying other models, other model configurations, ensembles of models, and so on. There are gains to be had later in the pipeline once you plateau on ideas or the accuracy delta.

You need a well considered and designed test harness for objectively estimating model skill on unseen data. It will be the only measure you have of your feature engineering process, and you must trust it not to waste your time.

General Examples of Feature Engineering

Let's make the concepts of feature engineering more concrete.

In this section we will consider tabular data like that you might have in an excel spreadsheet. We will look at some examples of manual feature construction that you might like to consider on your own problems.

When I hear "*feature engineering is critically important*", this is the type of feature engineering I think of. It is the most common form that I am familiar with and practice.

Which of these is best? You cannot know before hand. You must try them and evaluate the results to achieve on your algorithm and performance measures.

Decompose Categorical Attributes

Imagine you have a categorical attribute, like "*Item_Color*" that can be *Red*, *Blue* or *Unknown*.

Unknown may be special, but to a model, it looks like just another colour choice. It might be beneficial to better expose this information.

You could create a new binary feature called "*Has_Color*" and assign it a value of "*1*" when an item has a color and "*0*" when the color is unknown.

Going a step further, you could create a binary feature for each value that *Item_Color* has. This would be three binary attributes: *Is_Red*, *Is_Blue* and *Is_Unknown*.

These additional features could be used instead of the *Item_Color* feature (if you wanted to try a simpler linear model) or in addition to it (if you wanted to get more out of something like a decision tree).

Decompose a Date-Time

A date-time contains a lot of information that can be difficult for a model to take advantage of in it's native form, such as ISO 8601 (i.e. 2014-09-20T20:45:40Z).

If you suspect there are relationships between times and other attributes, you can decompose a date-time into constituent parts that may allow models to discover and exploit these relationships.

For example, you may suspect that there is a relationship between the time of day and other attributes.

You could create a new numerical feature called *Hour_of_Day* for the hour that might help a regression model.

You could create a new ordinal feature called *Part_Of_Day* with 4 values *Morning*, *Midday*, *Afternoon*, *Night* with whatever hour boundaries you think are relevant. This might be useful for a decision tree.

You can use similar approaches to pick out time of week relationships, time of month relationships and various structures of seasonality across a year.

Date-times are rich in structure and if you suspect there is time dependence in your data, take your time and tease them out.

Reframe Numerical Quantities

Your data is very likely to contain quantities, which can be reframed to better expose relevant structures. This may be a transform into a new unit or the decomposition of a rate into time and amount components.

You may have a quantity like a weight, distance or timing. A linear transform may be useful to regression and other scale dependent methods.

For example, you may have *Item_Weight* in grams, with a value like 6289. You could create a new feature with this quantity in kilograms as 6.289 or rounded kilograms like 6. If the domain is shipping data, perhaps kilograms is sufficient or more useful (less noisy) a precision for *Item_Weight*.

The *Item_Weight* could be split into two features: *Item_Weight_Kilograms* and *Item_Weight_Remainder_Grams*, with example values of 6 and 289 respectively.

There may be domain knowledge that items with a weight above 4 incur a higher taxation rate. That magic domain number could be used to create a new binary feature *Item_Above_4kg* with a value of "*1*" for our example of 6289 grams.

You may also have a quantity stored as a rate or an aggregate quantity for an interval. For example, *Num_Customer_Purchases* aggregated over a year.

In this case you may want to go back to the data collection step and create new features in addition to this aggregate and try to expose more temporal structure in the purchases, like perhaps seasonality. For example, the following new binary features could be created: *Purchases_Summer*, *Purchases_Fall*, *Purchases_Winter* and *Purchases_Spring*.

Concrete Examples of Feature Engineering

A great place to study examples of feature engineering is in the results from competitive machine learning.

Competitions typically use data from a real-world problem domain. A write-up of methods and approach is required at the end of a competition. These write-ups give valuable insight into effective real-world machine learning processes and methods.

In this section we touch on a few examples of interesting and notable post-competition write-ups that focus on feature engineering.

Predicting Student Test Performance in KDD Cup 2010

The KDD Cup is a machine learning competition held for attendees of the ACM Special Interest Group on Knowledge Discovery and Data Mining conferences, each year.

In 2010, the focus of the competition was the problem of modeling how students learn. A corpus of student results on algebraic problems was provided to be used to predict those students' future performance.

The winner of the competition were a group of students and academics at the National Taiwan University. Their approach is described in the paper "Feature Engineering and Classifier Ensemble for KDD Cup 2010".

The paper credits feature engineering as a key method in winning. Feature engineering simplified the structure of the problem at the expense of creating millions of binary features. The simple structure allowed the team to use highly performant but very simple linear methods to achieve the winning predictive model.

The paper provides details of how specific temporal and other non-linearities in the problem structure were reduced to simple composite binary indicators.

This is an extreme and instructive example of what is possible with simple attribute decomposition.

Predicting Patient Admittance in the Heritage Health Prize

The heritage health prize was a 3 million dollar prize awarded to the team who could best predict which patients would be admitted to hospital within the next year.

The prize had milestone awards each year where the top teams would be awarded a prize and their processes and methods made public.

I remember reading the papers released at the first of the three milestones and being impressed with the amount of feature engineering involved.

Specifically, the paper "Round 1 Milestone Prize: How We Did It – Team Market Makers" by Phil Brierley, David Vogel and Randy Axelrod. Most competitions involve vast amounts of feature engineering, but it struck me how clearly this paper made the point.

The paper provides both tables of attributes and SQL required to construct the attributes.

The paper gives some great real-world examples of feature engineering by simple decomposition. There are a lot of counts, mins, maxes, lots of binary attributes, and discretized numerical attributes. Very simple methods used to great effect.

More Resources on Feature Engineering

We have covered a lot of ground in this article and I hope you have a much greater appreciation of what feature engineering is, where it fits in, and how to do it.

This is really the start of your journey. You need to practice feature engineering and you need to study great practitioners of feature engineering.

This section provides some resources that might help you on your journey.

Books

I cannot find any books or book chapters on the topic.

There are however some great books on feature extraction. If you are working with digital representations of analog observations like images, video, sound or text, you might like to dive deeper into some feature extraction literature.

- Feature Extraction, Construction and Selection: A Data Mining Perspective
- Feature Extraction: Foundations and Applications (I like this book)
- Feature Extraction & Image Processing for Computer Vision, Third Edition

There are also lots of books on feature selection. If you are working to reduce your features by removing those that are redundant or irrelevant, dive deeper into feature selection.

- Feature Selection for Knowledge Discovery and Data Mining
- Computational Methods of Feature Selection

Papers and Slides

It is a hard topic to find papers on.

Again, there are plenty of papers of feature extraction and chapters in books of feature selection, but not much of feature engineering. Also feature engineering has a meaning in software engineering as well, one that is not relevant to our discussion.

Here are some generally relevant papers:

- JMLR Special Issue on Variable and Feature Selection

Here are some generally relevant and interesting slides:

- Feature Engineering (PDF), Knowledge Discover and Data Mining 1, by Roman Kern, Knowledge Technologies Institute
- Feature Engineering and Selection (PDF), CS 294: Practical Machine Learning, Berkeley
- Feature Engineering Studio, Course Lecture Slides and Materials, Columbia
- Feature Engineering (PDF), Leon Bottou, Princeton

Links

There blog posts here and there. The most useful links are tutorials that work through a problem and clearly articulate the intentional feature engineering.

Below are some generally interesting links:

- Feature Engineering: How to perform feature engineering on the Titanic competition (a getting started competition on Kaggle). There is more data munging than feature engineering, but it's still instructive.
- ~~IPython Notebook by Guibing Guo, dedicated to explaining feature engineering. A bit messy, but worth a skim~~. (link appears broken, sorry.)

Videos

There are a few videos on the topic of feature engineering. The best by far is titled "Feature Engineering" by Ryan Baker. It's short (9 minutes or so) and I recommend watching it for some good practical tips.

If you think I missed a key concept or resource, please leave a comment.

Update 2015: I notice that there is now a Wikipedia article on Feature Engineering and it copies large chunks of this post. Oh well.