



Data Science: A Kaggle Walkthrough – Data Transformation and Feature Extraction

MARCH 27, 2016 / BRETT ROMERO / 12 COMMENTS

This article on data transformation and feature extraction is Part IV in a series looking at data science and machine learning by walking through a Kaggle competition. If you have not done so already, you are strongly encouraged to go back and read [Part I](#), [Part II](#) and [Part III](#).

Continuing on the walkthrough, in this part we focus on getting the data we cleaned in [Part III](#) ready for use in the classification algorithm. These steps are often referred to as data transformation and feature extraction.

Data Transformation and Feature Extraction as a Concept

The main purpose of data transformation and feature extraction is to enhance the data in such a way that it increases the likelihood that the classification algorithm will be able to make meaningful predictions. Unlike the steps taken during cleaning, which are designed to address problems with the raw data (missing and erroneous values, formatting issues etc.), these steps change the values and/or structure of the data (data transformation) and add additional features (feature extraction).

As you might imagine, this is quite an open-ended process, and hence a lot of the value that data scientists provide comes in these steps. There is no textbook or walkthrough that can tell you exactly what steps you should take for a given dataset, that knowledge can come only from experience, curiosity and trial and error. However, we can take a look at some common methods to provide a sense of what is possible. Please keep in mind this is not an exhaustive list of options.

Data Transformation

Covering steps taken to modify the data, data transformation is undertaken with the intention to enhance the ability of the classification algorithm to extract information from the data. Below are a few common data transformation methods used.

Bucketing/Binning

A common method for manipulating numeric data, binning or bucketing is when the numerical values in a particular column are converted from a continuous series into

fixed ranges. For example, instead of using the age value of all our users, we could place them into buckets such as 15-20 years old, 21-25 years old and so on.

Typically this technique is used to manage 'noisy data'. To understand what this means, think of the movements of the stock market over time: it goes up and down on an almost daily basis. However, if you are trying to predict the overall direction of the stock market over the next 6 months, these daily movements become kind of irrelevant – what you really want your model to focus on are the movements over longer periods of time. What is more, the essentially random daily movements in stock prices may actually confuse your prediction model – causing less accurate predictions. In this example, the daily movements are the noise and what you want to extract (the longer term direction of the market) is 'the signal'.

The same logic can be applied to any numerical field in your dataset. If you are concerned that small changes in a given value may simply be representing random 'noise', you may want to consider bucketing/binning to remove that noise.

Normalization

Although normalization can take on a large number of meanings depending on the context, the type of normalization being referred to here is the **statistical type** – converting the values of a column into a 'normalized' range. This could be translating heights from centimeter values anywhere from 100cm to 220cm to a scale where 0 represents the average (mean) height for your dataset and -1/+1 represent one **standard deviation** from that average. It could be translating those heights into a range of values from 0 to 1, where 0 is the lowest value in your dataset and 1 is the maximum value. There is a number of other methods that can be used here as well.

This type of transformation is more important for certain types of algorithms than others. For some algorithms – like the one we will be using – this type of transformation is not typically necessary. But for other algorithms, the magnitude of the values in each column will impact the calculations. In these cases, it is optimal to convert ('normalize') the values in each column onto the same scale to ensure each column is treated the equally. For a more detailed explanation on this subject, **this answer** from Quora is a good place to start.

Other Mathematical Transformations


In a similar manner to normalization, there is an almost unlimited number of ways that the numerical values of a given column can be transformed such that they are more suitable for the algorithm being used.

To provide one example, arguably the most common transformation (other than normalization) is to use a logarithm function. This transformation is a commonly used method of dealing with exponential data series (i.e. a column where there a lot of low values and relatively few high values). For those wanting to understand this transformation better, the [Wikipedia page on this topic](#) has a great illustrated example.

As I am hemorrhaging readers at this point, I won't go into detail on the various other transformations possible – the key point is to be aware that there is a large range of possibilities here depending on your needs.

One Hot Encoding

Looking at one more example, and the most relevant one for our Kaggle competition, this transformation is one used for categorical data. What this transformation does is take one column with x categories (x must be greater than 2 for this to make sense) and convert it into x columns where each column represents one category in the original column. An illustrated example is shown below:

ID	Gender		ID	Male	Female	Not Specified
1	Male		1	1	0	0
2	Female		2	0	1	0
3	Not Specified		3	0	0	1
4	Not Specified		4	0	0	1
5	Female		5	0	1	0

For those familiar with regression modeling, you may recognize this as the same process of creating dummy variables.

Again there are a few reasons for doing this type of transformation. Some algorithms are structured in such a way that they do not handle categorical data very well –

particularly when the categories do not have an inherent order ([this answer](#) on Stack Overflow does a good job of explaining why). Some other types of algorithms require numerical data to function. The only way to work out whether this transformation will be beneficial is to either read through the documentation for the algorithm you are using or to test it yourself.

Feature Extraction

Often broken down into sub steps of feature construction and feature selection, here we will focus on feature construction. Below are a couple of ways additional features can be constructed and added to your dataset.

Using Hierarchical Information

It will sometimes be the case that data in your dataset represents one level of a particular hierarchy, and that extracting the other implied levels of that hierarchy will provide the model with useful information.

For example, imagine a dataset with a column containing countries. This column allows the algorithm to look for patterns (in combination with all other columns) at the country level. However, by adding a new 'region' column based on the country column (Europe, South Asia, North Africa etc.), you may be providing information to the algorithm that allows it look for patterns across countries.

One of the most common ways to do this is with date fields. Take the date fields in the dataset we are working with as an example. By extracting the day of the week, the month of the year or the hour of the day, we could add important information for the algorithm to use. Maybe people who create their accounts in summer months are more likely to make a booking in a warmer country. Maybe people who were first active late at night are more disorganized travelers and are therefore more likely to make a domestic first booking. Additionally, it could be any combination of these factors that makes the difference (e.g. users first active late at night, in the summer months, on a weekday are more likely to travel to Portugal). The point is not to be able to explain why a factor may be important, but to think of as many factors as possible to test, and allow the algorithm to determine what is important and not important.

Adding External Data

One of the aspects of feature extraction that often gets overlooked is how data can be enriched through the addition of new external data. Using techniques such as [record linkage](#), existing datasets can be greatly expanded by adding new data points for a given record. This new data often provides valuable new information that the algorithm can use to make more accurate predictions.

For example, a training dataset that contains a column with countries could be enriched with demographic data about the country such as population, income per capita or land area – all factors that may allow the algorithm to draw conclusions across similar groups of countries on any of those measures.

Relating this concept to the competition we are working through, consider how much more accurately we could predict a first booking country of a user if we could link the data from their Airbnb profile to data from one of their social media profiles (Facebook, Twitter etc.) or even better, from a Tripadvisor or Expedia account.

The key point here is that it is worth investing time looking for ways to add new and useful data to your existing dataset before moving onto the modeling step. Expanding your dataset in this manner will often produce far bigger improvements in prediction accuracy than the choice of algorithm or the tuning of the algorithm parameters.

The Importance of Domain Knowledge

One of the things that may have occurred to you as you read through the various ways to modify and expand a dataset is how are you supposed to know what will help or not?

This is where knowledge about the data you are using and what it represents becomes so important. This knowledge – referred to as domain knowledge – helps guide this entire process, including what was covered in [Part III](#), cleaning the data.

Understanding how the data was collected helps to provide insight into potential errors in the data that might need to be addressed or shortcomings in the way the

data was sampled (sample selection bias/errors). Understanding the relevant industry or market can also provide a range of insights including:

- what additional information is available to expand your dataset
- what information may help to increase prediction accuracy and what is likely to be irrelevant
- if the model makes intuitive sense (e.g. can you predict the likelihood of a waking up with a headache based on whether someone slept with their shoes on?[\[1\]](#)), and
- if the industry or market is changing in such a way that it is likely to make the model redundant in the near future.

In practical terms, where does this leave aspiring data scientists?

The first thing is to realize that, obviously, it is not possible to be a domain expert for every domain. Acknowledging this limitation is important as it forces a second realization – you will almost always need to seek out this expertise. For most of us that means involving and utilizing people who are domain experts when constructing your dataset and model. Having access to that expertise is likely to be the difference between a model that gets thrown out in 6 months and one that fundamentally improves a business and/or fulfills a customer need.

Step by Step

After all the theory, let's put some of these techniques into practice.

Transforming Categorical Data

The first step we are going to undertake is some One Hot Encoding – replacing the categorical fields in the dataset with multiple columns representing one value from each column.

To do this, the Scikit Learn library comes with a [One Hot Encoder method](#) that we could use to do these transformations, but it is often instructive to write your own function, particularly if it is a relative simple one like this. The code snippet below creates a simple function to do the encoding for a specified column, and then uses

that function in a loop to convert all the categorical columns (and then delete the original columns).

```
# Home made One Hot Encoding function
def convert_to_binary(df, column_to_convert):
    categories = list(df[column_to_convert].drop_duplicates())

    for category in categories:
        cat_name = str(category).replace(" ", "_").replace("(", "")
        col_name = column_to_convert[:5] + '_' + cat_name[:10]
        df[col_name] = 0
        df.loc[(df[column_to_convert] == category), col_name] = 1

    return df

# One Hot Encoding
print("One Hot Encoding categorical data...")
columns_to_convert = ['gender', 'signup_method', 'signup_f

for column in columns_to_convert:
    df_all = convert_to_binary(df=df_all, column_to_convert=
    df_all.drop(column, axis=1, inplace=True)
```

Creating New Features

From [Part II](#) of this series, one of the things we observed about the training (and test) datasets is that there is not a huge number of columns to work with. This limits what new features we can add based on the existing data. However, two fields that can be used to create some new features are the two date fields – *date_account_created* and *timestamp_first_active*. We want to extract all the information we can out of these two date fields that could potentially differentiate which country someone will make their first booking in. The code for extracting a range of different data points from these two date columns (and then deleting the original date columns) is shown below:

```
# Add new date related fields
print("Adding new fields...")
df_all['day_account_created'] = df_all['date_account_create
df_all['month_account_created'] = df_all['date_account_cre
df_all['quarter_account_created'] = df_all['date_account_cre
df_all['year_account_created'] = df_all['date_account_creat
df_all['hour_first_active'] = df_all['timestamp_first_active'].c
df_all['day_first_active'] = df_all['timestamp_first_active'].dt
```



```
df_all['month_first_active'] = df_all['timestamp_first_active'].dt.month
df_all['quarter_first_active'] = df_all['timestamp_first_active'].dt.quarter
df_all['year_first_active'] = df_all['timestamp_first_active'].dt.year
df_all['created_less_active'] = (df_all['date_account_created'] < df_all['timestamp_first_active'])

# Drop unnecessary columns
columns_to_drop = ['date_account_created', 'timestamp_first_active']
for column in columns_to_drop:
    if column in df_all.columns:
        df_all.drop(column, axis=1, inplace=True)
```

Wrapping Up

In two relatively simple steps, we have changed our training dataset from 14 columns to 163 columns. Although this seems like a lot more information, most of this expansion was caused by the One Hot Encoding, which is not adding more information, but simply expanding out the existing information. We have not added any external data, and I didn't even really investigate what information we could have extracted from the other non-date columns.

Again, this process is open ended, so there is an almost unlimited range of possibilities that we have not even really begun to explore. As such, if you see an additional transformation or have an idea for the addition of a new feature, please feel free to let me know in a comment!

Next Time

In the next piece, we will look at the data in *sessions.csv* that we left aside initially and see how we can add that data to our training dataset.

[1] This is an example of the existence of a confounding factor. A model predicting whether someone will wakeup with a headache based on whether they slept with their shoes on ignores that there is a more logical explanation for the headaches – in this case that both the headaches and sleeping with shoes on are caused by a third factor – going to bed drunk.

Data Science, Technology

◀ DATA ◀ DATA MINING ◀ DATA SCIENCE ◀ DATA TRANSFORMATION
◀ FEATURE EXTRACTION ◀ KAGGLE ◀ MACHINE LEARNING

PREVIOUS POST

Data Science: A Kaggle Walkthrough –
Cleaning Data

NEXT POST

Data Science: A Kaggle Walkthrough – Adding
New Data

Leave a Reply

Your email address will not be published.

Name

Email

Website

Post Comment

© 2018 BRETT ROMERO – UP ↑