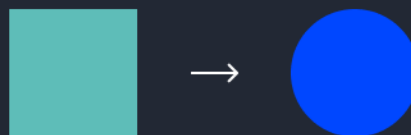


# Feature Engineering Cookbook for Machine Learning



Michael Abehsera

Mar 8, 2019 · 9 min read



When it comes to classic ML feature engineering is one if not the most important factors to improving your scores and speeding up your model without even bothering to tune or get fancy with your model.

There is not a lot of resources and books out there that cover feature engineering in depth, so I wanted to compile a list of code snippets covering most of the techniques I found online and used over time that were critical to most of the projects I worked on. These techniques mostly apply to decision tree and regression-type models (not deep learning).

My goal here was to give the coding examples and not cover the ins and outs on how each technique works and how they impact various models metrics, I am assuming you already heard of most of these techniques and that you will experiment to discover what works best for each project you're working on.

If I missed anything, please mention it in the comments, and I will update the post with a mention.

I use the following datasets in this post:

- Titanic
- News articles

I uploaded everything to this GitHub repo:

<https://github.com/michaelabehsera/feature-engineering-cookbook>

## Getting Started

```
%matplotlib inline
import pandas as pd
import numpy as np
import missingno as msno
from numpy import random
import matplotlib.pyplot as plt
```

### 1. Replacing NaN (or null) Values

Let's take a look at how many null values there are in each column. It seems `Age` and `Cabin` are the most commonly-null features. We will focus on replacing null values for `Age`.

```
df.isnull().sum()

PassengerId      0
Survived          0
Pclass           0
Name             0
Sex              0
Age             177
SibSp            0
Parch            0
Ticket           0
Fare             0
Cabin           687
Embarked         2
dtype: int64
```

## Mean/Median Imputation

```
df['Age'].fillna((df['Age'].mean()), inplace=True)
df['Age'].fillna((df['Age'].median()), inplace=True)
print('Now Age has {} null values.'.format(df.isnull().sum()
['Age']))
```

## Replacing with 0 or -1

```
df['Age'].fillna(value=0, inplace=True)
```

## Replacing with a Random Number. Random Sampling Imputation

We can replace age with random numbers between 1 and 100 as follows.

```
null_rows = df['Age'].isnull()
num_null_rows = sum(null_rows)

rand = random.randint(1, 101, size=num_null_rows)

df.loc>null_rows, 'Age' = rand

df['Age'].plot.hist(title='Distribution of Age - replace null with
random value')

plt.show()
```



A smarter approach would be to replace Age with random samples from the non-null distribution of Age (as, with the previous approach, we would have generated just as many 99-year-olds as 25-year-olds).

```
rand = np.random.choice(df.loc[~null_rows, 'Age'], replace=True,
size=num_null_rows)
df.loc[null_rows, 'Age'] = rand
df['Age'].plot.hist(title='Distribution of Age - replace null with
samples from distribution of Age')
plt.show()
print('Note the lack of 100 year olds in the above plot compared to
the previous plot.')
```

## Indicating Missingness

We could also use an additional 0/1 variable to indicate to our model when Age is missing.

```
df['Age_Missing'] = np.where(df['Age'].isnull(), 1, 0)
```

## Imputation of NA by Values at the End of the Distribution

```
df['Age'].fillna(df.Age.mean() + df.Age.std() * 3, inplace=True)
```

## Replacing With Values of Your Choosing Based on an Assumption.

```
df['Age'].fillna(value='My Unique Value', inplace=True)
```

## Using Regression to Impute Attribute Missing Values

We will use these variables to predict missing values for Age (we could use others, but would need to convert from text).

```
from sklearn.linear_model import LinearRegression
numeric_vars = ['Pclass', 'SibSp', 'Parch', 'Fare']
null_rows = df[numeric_vars + ['Age']].isnull().any(1) # rows where
Age or any feature var. is null
```

Fit regression model to the non-null rows of data, predict the null rows.

```
lr = LinearRegression()
lr.fit(df.loc[~null_rows, numeric_vars], df.loc[~null_rows, 'Age'])
df.loc[null_rows, 'Age'] = lr.predict(df.loc[null_rows,
numeric_vars])
```

## 2. Feature Scaling

### Standard Scaler

```
from sklearn.preprocessing import StandardScaler

x = StandardScaler().fit_transform(x)
```

### MinMax Scaler

```
from sklearn.preprocessing import MinMaxScaler

x = MinMaxScaler().fit_transform(x)
```

### Robust Scaler

```
from sklearn.preprocessing import RobustScaler

x = RobustScaler().fit_transform(x)
```

## 3. Engineering Outliers in Numerical Variables

### Mean/median imputation or random sampling

If we have reasons to believe that the outliers are due to mechanical error or problems during measurement. This means, if the outliers are in nature similar to missing data, then any of the methods discussed for missing data can be applied to replace outliers.

Because the number of outliers is in nature small (otherwise they would not be outliers), it is reasonable to use the mean/median imputation to replace them.

## Identify Outliers with Quantiles

```
q25 = df['Age'].quantile(0.25)
q75 = df['Age'].quantile(0.75)
IQR = q75 - q25

# Any value higher than ulimit or below llimit is an outlier

ulimit = q75 + 1.5*IQR
llimit = q25 - 1.5*IQR
print(ulimit, llimit, 'are the ulimit and llimit')
print('Imply Age outliers:')
df['Age'][np.bitwise_or(df['Age'] > ulimit, df['Age'] < llimit)]
```



## Identify Outliers with Mean

Using the mean and standard deviation to detect outliers should only be done with data that is not very skewed. Age is somewhat skewed so this could be an issue.

```
ulimit = np.mean(df['Age']) + 3 * np.std(df['Age'])
llimit = np.mean(df['Age']) - 3 * np.std(df['Age'])
ulimit, llimit
```

```
#out: (73.248081099510756, -13.849845805393123)
```

## Discretization

```
# Re-read data and fill nulls with mean (could use other null-
filling method)

df = pd.read_csv('train.csv')

not_null = ~df['Age'].isnull()

# Get the bin edges using np.histogram

num_bins = 10
_, bin_edges = np.histogram(df['Age'][not_null], bins=num_bins)

# Optionally create labels

# labels = ['Bin_{}'.format(i) for i in range(1, len(intervals))]
labels = [i for i in range(num_bins)]

# Create new feature with pd.cut

df['discrete_Age'] = pd.cut(df['Age'], bins=bin_edges,
labels=labels, include_lowest=True)
```

## Trimming

```
# Let's first remove any missing values
df['Age'].fillna((df['Age'].mean()), inplace=True)

# Get the outlier values
index_of_high_age = df[df.Age > 70].index

# Drop them
df = df.drop(index_of_high_age, axis=0)
```

## Winsorization (Top Coding Bottom Coding)

```
# Get the value of the 99th percentile
ulimit = np.percentile(df.Age.values, 99)

# Get the value of the 1st percentile (bottom 1%)
llimit = np.percentile(df.Age.values, 1)

# Create a copy of the age variable
df['Age_truncated'] = df.Age.copy()

# Replace all values above ulimit with value of ulimit
df.loc[df.Age > ulimit, 'Age_truncated'] = ulimit
```

```
# Replace all values below llimit with value of llimit
df.loc[df.Age < llimit, 'Age_truncated'] = llimit
```

## Rank Transformation (When the Distances Don't Matter so Much)

```
from scipy.stats import rankdata

# This is like sorting the variable and then assigning an index
starting from 1 to each value

rankdata(df['Age'], method='dense')
```

## 4. Engineering Labels, Categorical Variables

### One-Hot-Encoding and Pandas Get Dummies

#### One-Hot-Encoding

```
from sklearn.preprocessing import OneHotEncoder

one = OneHotEncoder(sparse=False).fit_transform(df[['Parch']])

#Convert transformed column from numpy to a DataFrame and merge new
column with older one.

onecol = pd.DataFrame(one)
results = pd.merge(onecol, df, left_index=True, right_index=True)
```

#### Get Dummies

```
# Generally, pd.get_dummies is an easier approach to one-hot
encoding

pd.get_dummies(df['Sex']).head()
```

#### Dropping First

In models which use all features at once (most models apart from tree ensemble models, etc), it is wise to drop the first dummy variable, as it can be derived from the



others (e.g. here we know if someone is Female based on whether they are Male, so we can drop Female).

```
df['Male'] = pd.get_dummies(df['Sex'], drop_first=True)
```

## Mean Encoding

We calculate the mean of the target variable for each class of the variable we wish to encode and replace that variable by these means.

```
# Calculate the mean encoding

means_pclass = df[['Survived']].groupby(df['Pclass']).apply(np.mean)
means_pclass.columns = ['Mean Encoding']
means_pclass

# Merge the encoding into our dataframe (by matching Pclass to the
index of our prob. ratio dataframe)

df = pd.merge(df, means_pclass, left_on=df.Pclass, right_index=True)
```

## Probability Ratio Encoding

```
def probability_ratio(x):
    probability_eq_1 = np.mean(x)
    probability_eq_0 = 1.0 - probability_eq_1
    return probability_eq_1 / probability_eq_0

# Calculate the probability ratio encoding
prob_ratios_pclass =
df[['Survived']].groupby(df['Pclass']).apply(lambda x:
probability_ratio(x))
prob_ratios_pclass.columns = ['Prob Ratio Encoding']
prob_ratios_pclass

# Merge the encoding into our dataframe (by matching Pclass to the
index of our prob. ratio dataframe)
df = pd.merge(df, prob_ratios_pclass, left_on=df.Pclass,
right_index=True)
df.head()
```

## Weight of Evidence Encoding

```
def weight_of_evidence(x):
    probability_eq_1 = np.mean(x)
    probability_eq_0 = 1.0 - probability_eq_1
    return np.log(probability_eq_1 / probability_eq_0)

# Calculate the probability ratio encoding

woe_pclass = df[['Survived']].groupby(df['Pclass']).apply(lambda x:
weight_of_evidence(x))
woe_pclass.columns = ['WOE Encoding']
woe_pclass

df = pd.merge(df, woe_pclass, left_on=df.Pclass, right_index=True)
```

## Label Encoding

### Cat.codes

```
# You can use cat.codes to convert the variable into a binary one

df['Sex'] = df['Sex'].astype('category')

# Cat.codes only works if the dtype is 'category'

df['Sex'].cat.codes.head()
```

### Factorize

Also achieves a similar end to `cat.codes`, but gives us the labels of each category.

```
label, val = pd.factorize(df['Sex'])
df['IsFemale'] = label
```

## Binary Encoding

#Can also use np.where to specify which values should be 1 or 0, as follows

```
df['Sex_Binary'] = np.where(df['Sex'].isin(['Male', 'Female']), 1, 0)
```

## 5. Engineering Dates

## Creating Columns Based on Hour/Min...

```
df = pd.read_csv('news_sample.csv')

time = pd.to_datetime(df['time'])
```

Pandas come with packed with DateTime properties which you could check out here: <https://pandas.pydata.org/pandas-docs/stable/api.html#datetimelike-properties>. You can even get a column with microseconds. Here are some of the ones I use most.

- Month
- Min
- Seconds
- Quarter
- Semester
- Day (number)
- Day of the week
- Hr

```
df['Month'] = time.dt.month
df['Day'] = time.dt.day
df['Hour'] = time.dt.hour
df['Minute'] = time.dt.minute
df['Seconds'] = time.dt.second
```

## Creating an isweekend Column

```
df['is_weekend'] = np.where(df['Day'].isin([5,6]), 1, 0)
```

## 6. Engineering Mixed Variables

We've seen that mixed variables are those which values contain both numbers and labels.

How can we engineer this type of variable to use it in machine learning?

What we need to do in these cases is extract the categorical part in one variable and the numerical part in a different variable. Therefore, we obtain 2 variables from the original one.

There is not much to cover here besides giving one fake example.

```
data = ['Apple', 'Banana', '2', '6']

lst_strings = []
lst_int = []

for i in data:
    if i == 'Apple':
        lst_strings.append(i)
    elif i == 'Banana':
        lst_strings.append(i)
    if i == '2':
        lst_int.append(int(i))
    elif i == '6':
        lst_int.append(int(i))
```

## 7. Engineering Rare Labels in Categorical Variables

Let's say there is a small % of values within a large group of categories in a feature, you can grab them and call them “other” in order to reduce the values in the dataset and potentially reduce overfitting.

These observations can be re-categorized by:

- Replacing the rare label by most frequent label
- Grouping the observations that show rare labels into a unique category (with a new label like ‘Rare’, or ‘Other’)

### Replacing the Rare Label by Most Frequent Label

```
val_counts = df['Age'].value_counts()
uncommon_ages = val_counts[val_counts<3].index.values
```

```
most_freq_age = val_counts.index[0]
df.loc[df['Age'].isin(uncommon_ages), 'Age'] = most_freq_age
```

## Grouping the Observations that Show Rare Labels Into a Unique Category (With a New Label like 'Rare', or 'Other')

```
val_counts = df['Age'].value_counts()
uncommon_ages = val_counts[val_counts<3].index.values
df.loc[:, 'Age_is_Rare'] = 0
df.loc[df['Age'].isin(uncommon_ages), 'Age_is_Rare'] = 1
df.loc[:, 'Age_is_Rare'].head()
```

## 8. Transforming Variables

Particularly for linear models, it can be useful to transform input features (or the target) prior to fitting the model, such that their distribution appears normal, or approximately normal (i.e. symmetric and bell-shaped).

### Gaussian Transformation

```
df['Age'].plot.hist()

df['Age Log'] = df['Age'].apply(np.log)

df['Age Log'].plot.hist()
```



## Reciprocal Transformation

```
df['Age Reciprocal'] = 1.0 / df['Age']  
  
df['Age Reciprocal'].plot.hist()
```



## Square Root Transformation

```
df['Age Sqrt'] = np.sqrt(df['Age'])  
  
df['Age Sqrt'].plot.hist()
```



## Exponential Transformation

```
df['Age Exp'] = np.exp(df['Age'])  
  
df['Age Exp'].plot.hist()
```



## Boxcox Transformation

```
from scipy.stats import boxcox  
  
df['Age BoxCox'] = boxcox(df['Age'])[0]  
df['Age BoxCox'].plot.hist()
```



## 9. Interaction Features

Perhaps, for example, older passengers who also paid a higher fare had a *particularly* high chance of not surviving on the Titanic. In such a case we would call that an *interaction* effect between ``Age`` and ``Fare``. To help our model take account of this interaction effect, we can add a new variable ``Age * Fare``.

```
df = pd.read_csv('train.csv')  
  
df['Age_x_Fare'] = df['Age'] * df['Fare']
```

This is of course just one example of an interaction feature, this is a powerful technique and could use some creativity on your part based on the dataset you're dealing with.

[Data Science](#)[Machine Learning](#)[Feature Engineering](#)[Pandas](#)[Scikit Learn](#)[About](#) [Help](#) [Legal](#)

Get the Medium app

