Rare Feature Engineering Techniques for Machine Learning Competitions

Mohammad Shahebaz

Photo by Franki Chamaki on Unsplash

Creativity has always been the essence of human evolution. May it be the first wheel rolling or the first spark of some insane idea leading to wonderful discoveries in our history. There have been decades of advancements since the stone age and creativity is still admired everywhere.

Every domain now enriches creativity. And, I believe that a data sciences embraces it the most. Right from null-hypothesis, data wrangling to building models there's a significant role of creative insight.

A Kaggle Grandmaster once told me that —

> *"The more you solve problems. The more you get exposed to ideas, challenges and then discover things that actually work for a particular problem"*

That experience part is how you use the data to make informative features for the model to learn better. That's Feature Engineering.

Why is Feature Engineering so important?

Many beginners in data science are moved by how accurate LGBM and XGBoost works. And often they tend to ignore Linear and KNN Regressors.

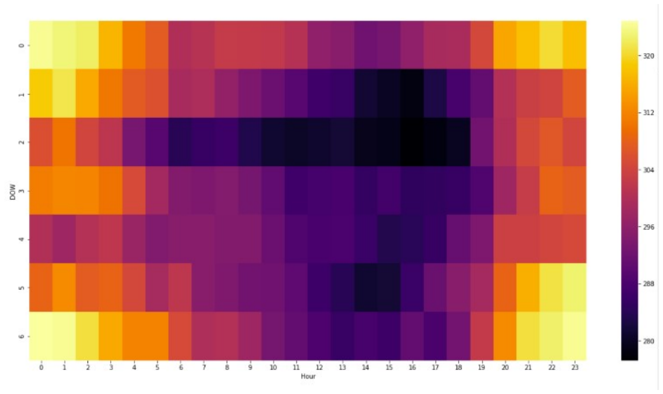
Linear Regression, in certain cases, outperforms even the GBM trees. In fact, I have used some of these models to gain edge in a few competitions.

There is a common aphorism in statistics, attributed to the statistician George Box —

> *All models are wrong but some are useful*

A model is powerful only when it discovers some significant relationship of features with target variable. Now here's where the feature engineering part comes into play, wherein we engineer or create new features so that the model can derive such significant relation.

I once experimented with the **weekend proximity** in a competition at DataHack were we were given dataset to predict the electricity consumption. With heatmaps and exploratory data analysis I plotted the following heatmap.



The heatmap is plotted between the groupby data of Day of Week (**DOW**) and **Hour** of the day. Clearly, there was a patterb at weekends and late night hours of weekend days. The engineered feature—**weekend proximity** improved the score and I ended up winning the competition. Let's cover few more techniques and features which might help you get an edge in your upcoming competitions.

Given below are some of the *rare feature engineering* tricks implemented in the winning solutions of several data science competitions.

1. Transform data to Image
2. Meta-leaks
3. Representation learning features
4. Mean encodings
5. Transforming target variable

Transform data to image pixels

In Microsoft Malware Classification Challenge participants were provided with unprecedented malware dataset. For each file, the raw data contained the hexadecimal representation of the file's binary content. The task was to develop the best mechanism for classifying files in the test set into their respective family affiliations.

Using image representation of raw data as a feature helped the team "say NOOOOO to overfittttting" win the Microsoft Malware Classification Challenge.

The team came with the following approach —

Malwares can be visualized as grayscale images using the byte file. Each byte is from 0 to 255 so it can be easily translated into pixel intensity. However, the standard image processing techniques doesn't work well with other features as n-grams. Later, extraction a grayscale image from asm file rather than the byte file.

The image is shown in below compares the byte image and asm image of the same malware.

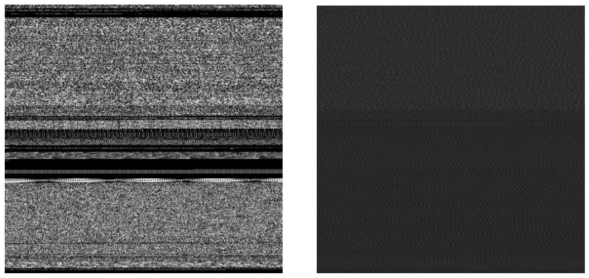


Figure 2. byte image (left) and asm image (right) of the same malware.

Byte image (left) and asm image (right) of the same malware

The team found that the intensities of the first 800 or 1000 pixels in the converted ***asm** file* to ***asm** image* is a good feature for the malware.

> *What is an **asm file**? Program written in assembly language, a low level programming language that can be converted to machine language*

Although they claimed that they didn't understand why it worked. The performance of using this feature alone was not that impressive, but when it was used together with the other n-gram features, the performance is improved significantly. The image representation helped the model to differentiate the clean versus the malware files more efficiently. Thus, improving the score.

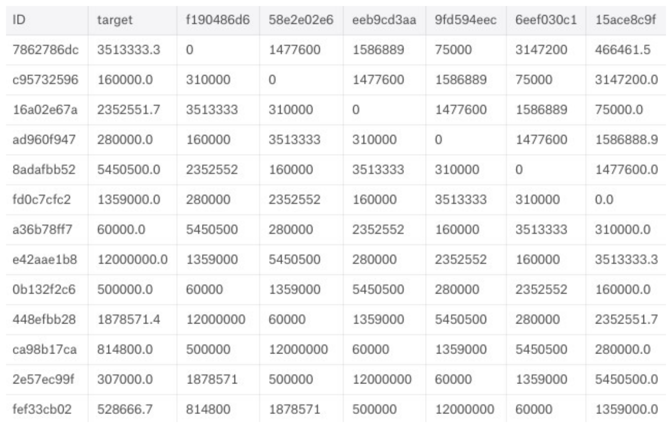
Converting raw data to image and then using pixels as feature. It was one of the amazing feature engineering happened in Kaggle Competitions.

For a detailed explanation, please refer the complete winning solution code.

Meta-Leaks

> *When a processed feature explains the target unreasonably well without application of any ML. It's called Data Leakage.*

A recent competition at Kaggle, Santander Value Prediction Challenge had a leak that explained the target with brute-force searching of sequence on both rows and columns.



ID	target	f190486d6	58e2e02e6	eeb9cd3aa	9fd594eec	6eef030c1	15ace8c9f
7862786dc	3513333.3	0	1477600	1586889	75000	3147200	466461.5
c95732596	160000.0	310000	0	1477600	1586889	75000	3147200.0
16a02e67a	2352551.7	3513333	310000	0	1477600	1586889	75000.0
ad960f947	280000.0	160000	3513333	310000	0	1477600	1586888.9
8adafbb52	5450500.0	2352552	160000	3513333	310000	0	1477600.0
fd0c7cfc2	1359000.0	280000	2352552	160000	3513333	310000	0.0
a36b78ff7	60000.0	5450500	280000	2352552	160000	3513333	310000.0
e42aae1b8	12000000.0	1359000	5450500	280000	2352552	160000	3513333.3
0b132f2c6	500000.0	60000	1359000	5450500	280000	2352552	160000.0
448efbb28	1878571.4	12000000	60000	1359000	5450500	280000	2352551.7
ca98b17ca	814800.0	500000	12000000	60000	1359000	5450500	280000.0
2e57ec99f	307000.0	1878571	500000	12000000	60000	1359000	5450500.0
fef33cb02	528666.7	814800	1878571	500000	12000000	60000	1359000.0

Data Leak at Santander (Credits: Giba)

The target variable is clearly leaked in `f190486` column. Without application of any machine learning, I managed to score about `0.57` RMLSE on the leaderboard. The leak was discovered nearly 20 days before the competition deadline but the host wished to continue the competition assuming this to be a data property.

Although, this a rare case of target leak, your model can extract good patterns from features like *file names, image meta-data and also in ID variable*.

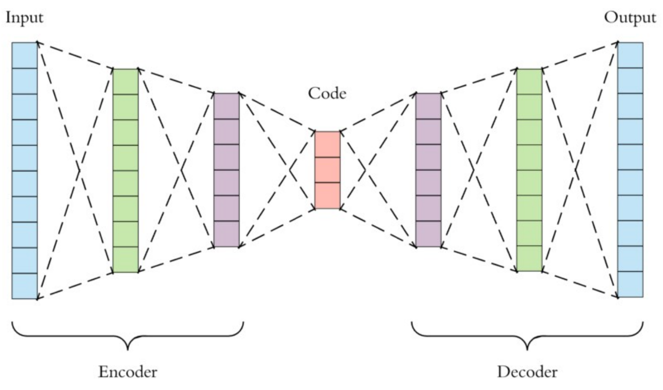
So, do spend little time on EDA on IDs and other features. You might end up discovering a leak and exploit it fully in positive potential

Note that these engineering is not practical in real data science problem but is only specific to competitions where the problem organizers fail to prevent data leakage.

Representation Learning Features

A traditional data science competitor is well aware of the basic feature-engineering methods like—Label Encoding, One-Hot Encoding, Binning and other feature methods that work well. However, these methods are quite common and pretty much used by everyone these days.

To stand out from the crowd and take yourself higher on the leaderboard clever model tuning and feature engineering is required. One of these clever feature engineering techniques is offered by representation learning using AutoEncoders.



Auto Encoder. Source — TDS

> *Autoencoder is simply put a representation learning model that learns the input and then regenerates the input itself.*

> *Example: It's like exposing a human to a picture of a cat and then asking him to draw that cat after a while.*

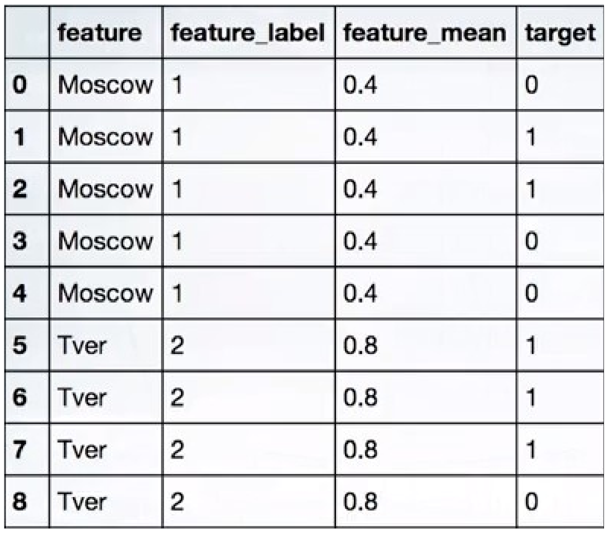
The intuition is to extract the best-observed features in the learning process. In above example, a human will surely sketch two eyes, triangular ears and perhaps whiskers. And, later model can signify the importance of the presence of whiskers in classifying the image as a cat.

Michael Jahrer used the same implementation at Porto Seguro's Safe Driver Prediction competition where representation learning was a key factor in winning of the competition.

Mean Encoding

Mean encoding is no longer that rare. But for a beginner starting out with machine learning, its a quite a handy trick to use at solving problems for greater accuracy. When target's value counts from the training data is used to replace the categorical values its called Target Encoding. When a statistical measure like mean is used to encode categorical values then its Mean Encoding.

Let's take the number of labels into account along with the target variable to encode the labels giving information of value_counts occurrence per class of target variable.



	feature	feature_label	feature_mean	target
0	Moscow	1	0.4	0
1	Moscow	1	0.4	1
2	Moscow	1	0.4	1
3	Moscow	1	0.4	0
4	Moscow	1	0.4	0
5	Tver	2	0.8	1
6	Tver	2	0.8	1
7	Tver	2	0.8	1
8	Tver	2	0.8	0

The **feature_label** is simply Label Encoding implementation of scikit-learn library. While the **feature_mean** is the

No. of true targets under the label Moscow / Total Number of targets under the label Moscow

that is 2/5 = 0.4.

Similarly, for ***Tver*** categorical value —

m = No. of true targets under the label Tver = 3

n = Total Number of targets under the label Tver= 4

Now, encoding for Tver is given by

m/n = 3/4 = 0.75 (0.8 Approx)

When is Mean-Encoding preferred over other encoding methods?

- In case of a high cardinal features, mean encoding could prove to be a much simpler alternative.

> *High Cardinality : When number of unique elements of a column is significanltly high.*

- In such case, the label encoding will result in continuous numbers till the range of the cardinality. Thus, adding noisy labels and encodings to the feature resulting in poor accuracies. And, the condition worsens for one-hotencoding as the dimensionality of the dataset increases with one-hot encoded features. In such scenario, mean encoding are a way to go.

There are various other implementations of the mean encoding. Mean encoding are prone to overfitting to trainset because we are leaking enough information of target in terms of means while encoding our categorical features. The practical method to benefit from target encoding is to use it along a proper regularization technique.

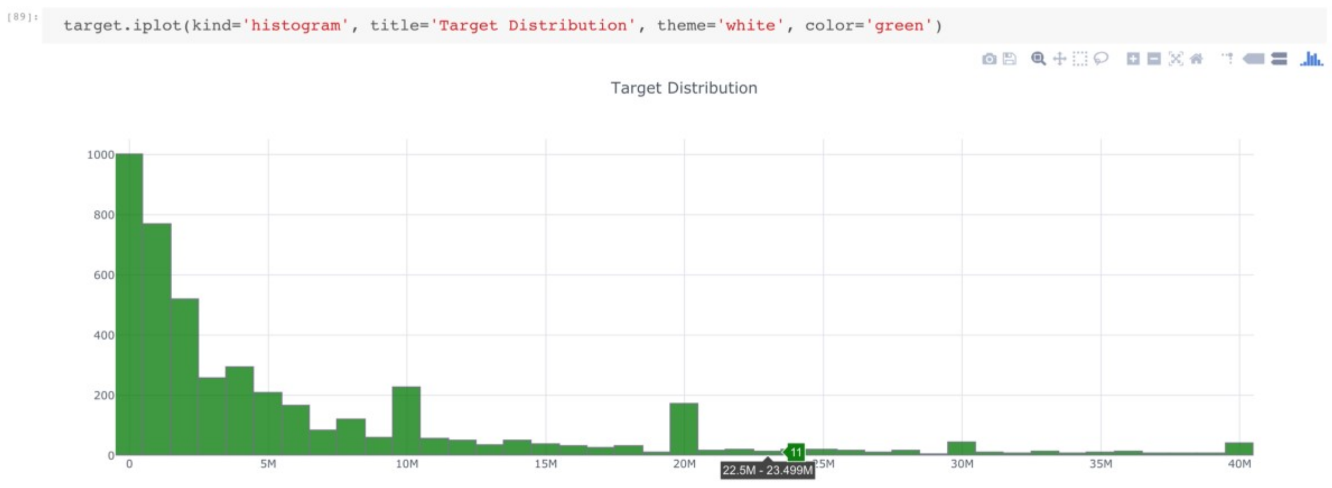
There are lot ways to regularize mean encodings;

- Regularization using CV loop methods
- Regularization Smoothing
- Regularization Expanding mean

The complete explanation of each method is beyond the scope of this article. For detailed explanation head over to the following video at Coursera.

Transforming Target Variable

Not, strictly feature engineering. But, when you are given a highly skewed data the models often predict the values that weight towards the skewed side.

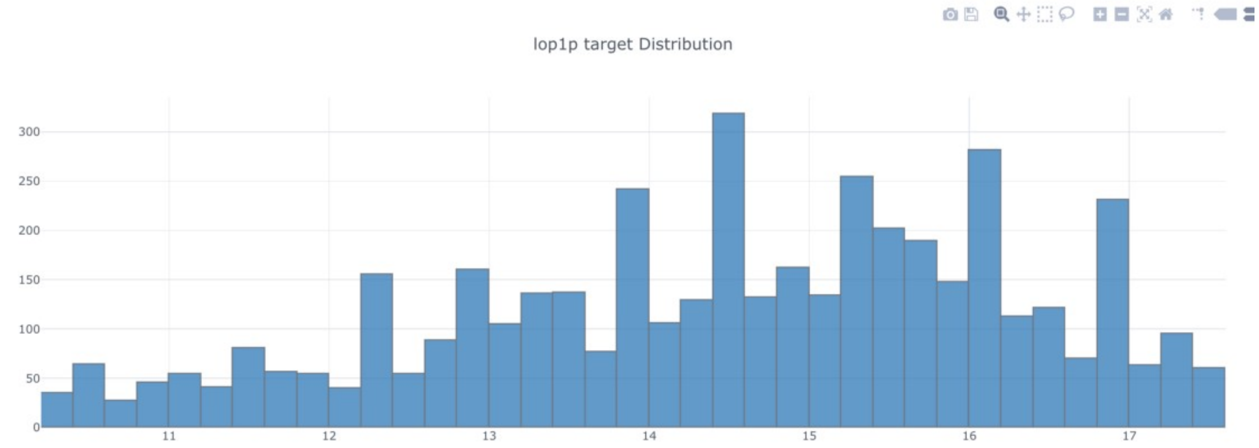


```
[89]: target.iplot(kind='histogram', title='Target Distribution', theme='white', color='green')
```

Target distribution

Just, transforming the target variable wit h `log(1+target)` gives more near-to gaussian distribution.

```
[90]: log1p_target.iplot(kind='histogram', title='lop1p target Distribution', theme='white', color='blue')
```



log1p target transformation

Note that you should again reverse transform the predicted values using `predictions = np.exmp1(log_predictions)` to get expected prediction values for submission.

That's all for now. Do let me know if you have others tricks for feature engineering up your sleeve.