Fraud Detection - Feature Engineering

- Roger

Categorical features - Standard approaches

One-hot Encoding

One common technique for dealing categorical features is **one-hot-encoding**, i.e. for each level of category, we create a binary feature.

However, in the case of high cardinality (e.g. city ID , region ID, zip codes), this technique leads to infeasibly large number of features.

Tree based models particularly suffer from such a large number of one-hot features because;

1. 1) tree only grows in the direction of zeroes of one hot feature (shown in figure below),
2. 2) if tree wants to split on one-hot (0,1) feature, it's information gain will be low and thus tree will be unable to get splits from the feature near the root if other continuous features are also present.

Label Encoding

Another standard encoding technique is **numerical encoding** (aka **label encoding**) where we assign a random number to each category. This technique infuses some form of order in the data and that order generally doesn't make sense and is random.

We are required to maintain a mapping of category to number in order to later apply same mapping for test data. Maintaining the mapping can sometimes become hard for models in production.

Tree based models suffer from this encoding too, as trees find best split based on values less than or greater than an optimum value. In case of random ordering, less than or greater than does not really make any sense.

Hash Encoding

This commonly used technique converts string type features into a fixed dimension vector using a hash function. The hash function employed in sklearn is the signed 32-bit version of Murmurhash3. A 32 dimensional vector can hold 2^{32} unique combinations.

This becomes useful for high cardinality variables where one hot encoding will give huge number of features. This technique is shown to work really well, particularly for cases where categories have some information in their string pattern.

Pros of hashing:

1. 1) No need to maintain a mapping of category to number mapping. Just need to use the same hash function to encode the test data;
2. 2) Reduction of levels specially in case of high cardinality features, which consequently makes it easier for tree to split;
3. 3) Can combine multiple features to create single hash. This helps in capturing feature interactions.

Cons of hashing:

1. 1) Hash collisions. Different levels of categories can fall into the same bucket. If those categories do not affect target equally, then prediction score might suffer;
2. 2) Need to tune another hyper parameter of number of hash vector dimensions.

Target Encoding

Another option, which is also the prime focus of this lecture is **target encoding** (aka **likelihood encoding**). The idea is to replace each level of the categorical feature with a number and that number is calculated from the distribution of the target labels for that particular level of category.

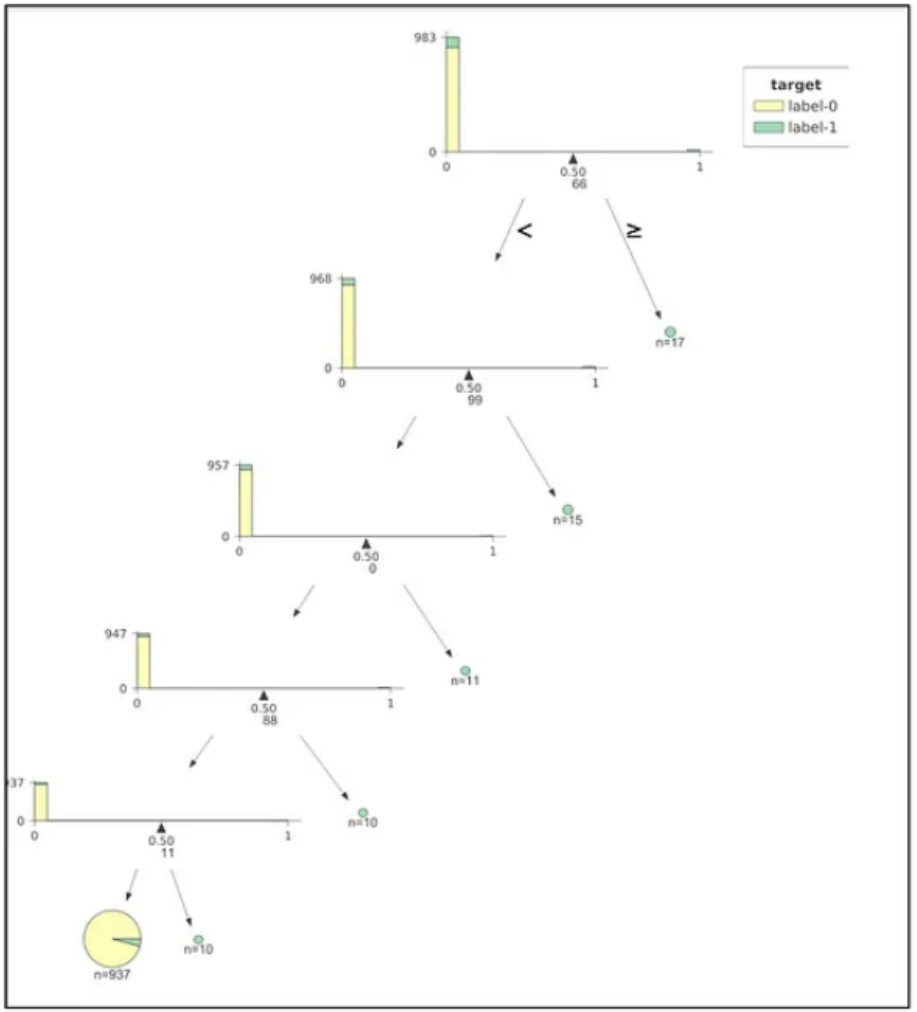
For example, if the levels of categorical feature are red, blue and green. Then replace red with some statistical aggregate (mean, median, variance etc.) of all the target labels where-ever the feature value is red in training data.

Below plots show how a decision tree gets constructed for a categorical feature when it is one-hot encoded vs. when it is numerical encoded (a random number for each category). The third plot shows the decision tree with target encoded feature.

For all the plots, a simulated data of 1000 rows with only one categorical feature and a binary target is used. The categorical feature has high cardinality and has 100 different levels labelled from 0 to 99 (keep in mind order doesn't make sense as we are treating it as nominal categorical feature).

The target is dependent on the feature in such a way that categories that are multiple of 11 (0, 11,22,33 .. 99) have target value of 1 and 0 for all other cases. The data has 897 negative and 103 positive labels).

The plots are constructed using dtreeviz library for decision tree visualizations with trees having a constraint of max depth 5.

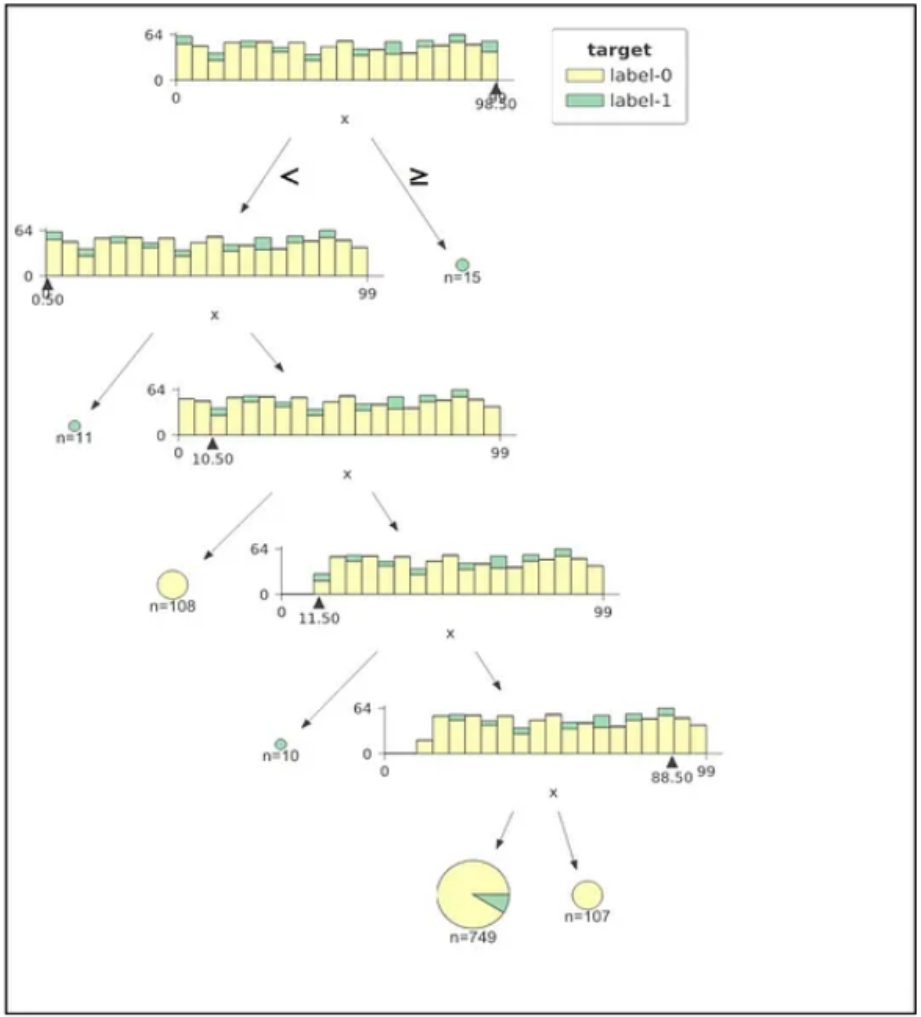


1. Decision tree fitted on one hot encoded feature.

In the 1st plot, we can see that decision tree with one-hot encoded features creates many many splits and the tree is all left-sided. The tree is still getting the work done and is able to separate the data.

We observe split values to be 66, 99, 0, 88, 11 respectively from root to leaf. At each split, information gain is low as split focuses only on one level of the categorical feature. If we introduce noisy continuous features, the tree might find more information gain in noise than some one-hot encoded feature and would give those up in feature importance plots, which is sometimes misleading.

This issue is discussed in detail in this blogpost – Are categorical variables getting lost in your random forests? and this kaggle discussion page – Why one-hot-encoding gives worse scores?



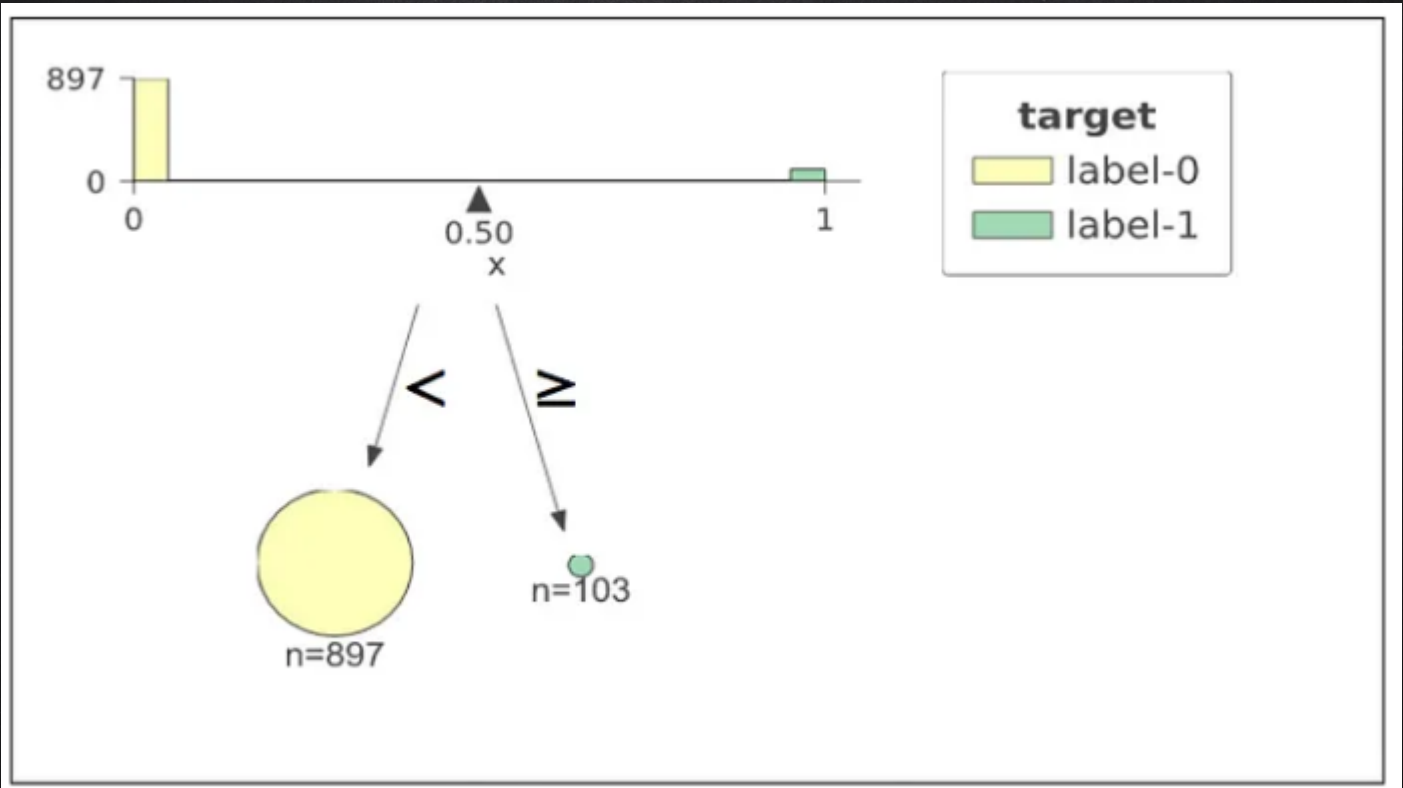
2. Decision tree fitted on numeric encoded feature

In the 2nd plot, tree splits on 98.5, 0, 10.5, 11.5, 88.5 respectively from root to leaf. So this tree is also getting to the right answer and finding splits close to multiple of 11.

It is assuming an order in the numbers assigned to categories, which in reality is random assignment. We see that the tree finds it difficult to get to a good decision boundary even for this perfect rule based data.

In practice, this technique often works well with random forest or gradient boosting techniques as long as cardinality is not high. If cardinality is high numerical encoding is likely to make your computational costs explode before you get any benefit.

A workaround to this problem of computational cost is to assign number/rank based on the frequency of that level.



3. Decision tree on target encoded feature

In the 3rd plot, tree with target mean encoding found the perfect division of data in just 1 decision stomp/split. We see that this encoding worked great in our toy example. Let's dig into this further and limit the scope of this lecture to target encoding, target leakage/prediction shift, overfitting caused due to this type of encoding; and finally Catboost's solution to this problem.

Target encoding

Different types of target encoding

Target encoding is substituting the category of k-th training example with one numeric feature equal to some target statistic (e.g. mean, median or max of target).

1. Greedy
2. Holdout
3. K-fold
4. Ordered (the one proposed by Catboost)

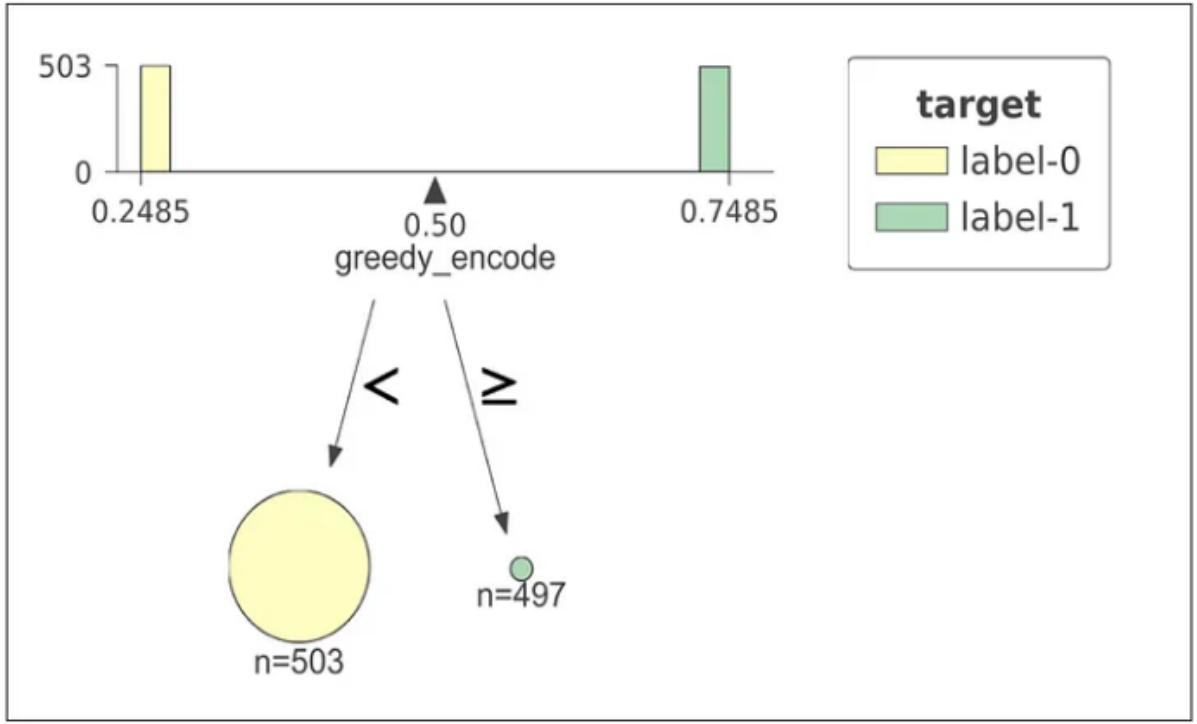
Greedy target encoding

This is the most straightforward approach. Just substitute the category with the average value of target label over the training examples with the same category.

We are only getting to see the labels of the training data. So, we find the mean encoding of all categories from the training data and map as it is to the test data.

For the cases where some new category is found in test and not available in training data, we substitute values of that category with overall mean of the target.

This works fine as long as we have large amount of training data, categorical features with low cardinality and the target distribution is in training and test data. But this fails to work in the other cases.



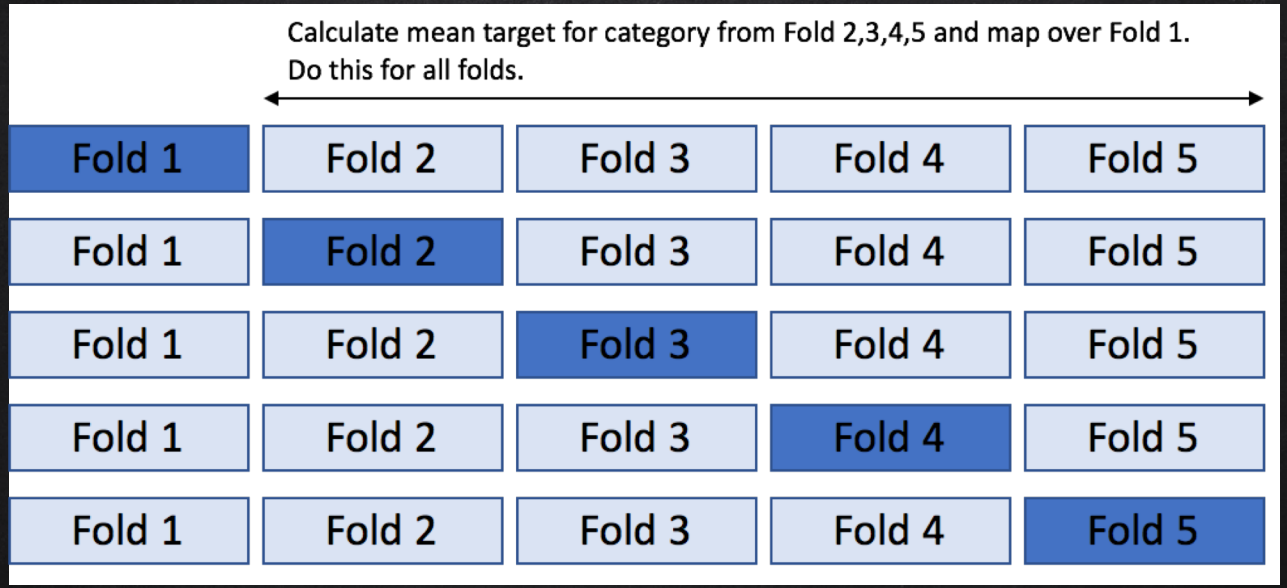
Decision tree on greedy target encoded feature

Let's look at an extreme example to show failure of this encoding technique.

On the left, we see a decision tree plot with perfect split at 0.5 threshold. The training data used for this model has 1000 observations with only one categorical feature having 1000 unique levels. Think of User ID as an example of such unique categorical feature. The labels (0,1) have been assigned randomly to the training data. That means that the target doesn't have any relationship with the feature.

But what decision tree shows us? The tree shows us a perfect split at the very first node, even though in reality the perfect split doesn't exist. The model will show near perfect score on the training data. Is that true? No. Test data might have new User IDs not seen in training data and our model will suck on the test data. This is the case of "overfitting on the training data". Later we will solve this issue.

K-fold target encoding



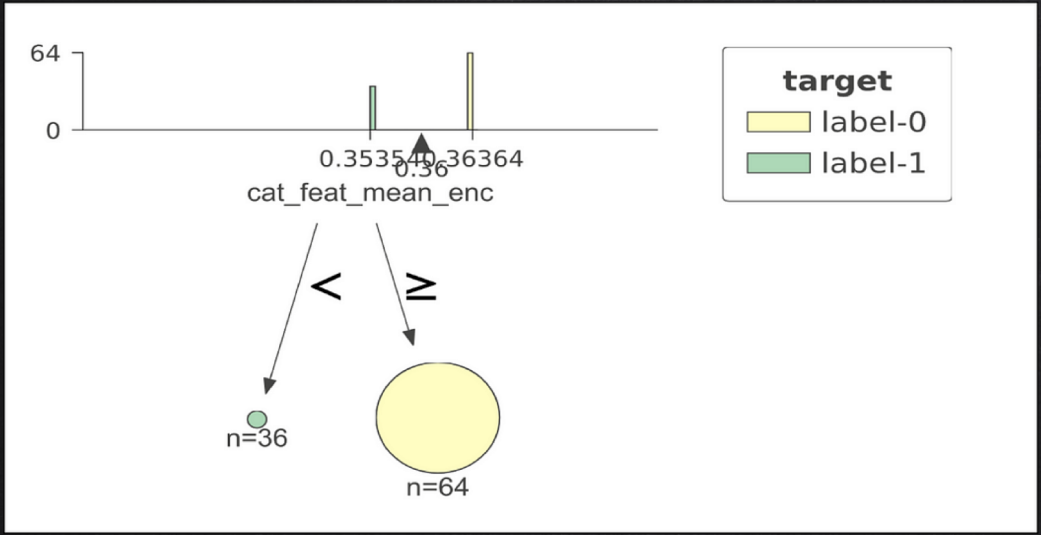
This is the most commonly used approach and solves the issue of overfitting on the training data ("mostly", not always). The idea is similar to k-fold cross validation. We divide the data in K- stratified or random folds, replace the observations present in M-th fold with mean target of data from all others except M-th fold. We are basically trying to 1) Use all of the training data given to us, 2) Not leak the information from self-target label by allowing target information to flow from other fellow observations (same category but other folds). Conventionally, people use this approach with K=5 and so-far it has worked well for small to large datasets.

Holdout target encoding – special case of K–fold when K=2.

Another option is to partition the training dataset into 2 parts. First part to calculate target statistic and the second part to do model training. This way we are not leaking any information from the target of second part which is actually being used for training. But this technique leads to the problem of data reduction as we are basically using less training data than we actually have. With this approach we are not effectively using all the training data we have at our hand. Works when we have large amount of data, still not a great solution though.

Leave one out target encoding – special case of K–fold when K=length of training data.

This is particularly used when we have small dataset. We calculate target statistic for each observation by using labels from all except that particular observation. But this technique can also lead to overfitting in some cases. Let's see one such failure example using the decision tree visualization.



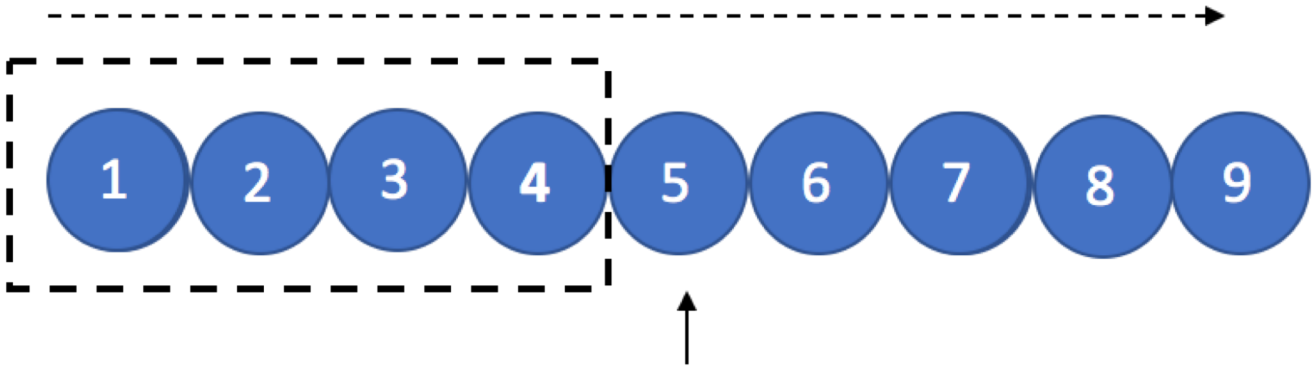
In this example our training data has 100 observations and only 1 feature; call it color and it is always red. It is obvious that this feature is useless. If we try to fit a decision tree with leave one out encoding on this feature, we might end up with a perfect split of the training data (as shown below). This is misleading as this perfect split doesn't exists. The reason it gives this split is because when we use leave one out, it either leaves 0 out or leaves 1 out, giving us only two unique encodings. When we train model on this, it uses those two unique values to find perfect split.

Ordered target encoding

This is not a commonly used approach. But this is the core idea behind Catboost library, which works really well for categorical data as shown in this paper.

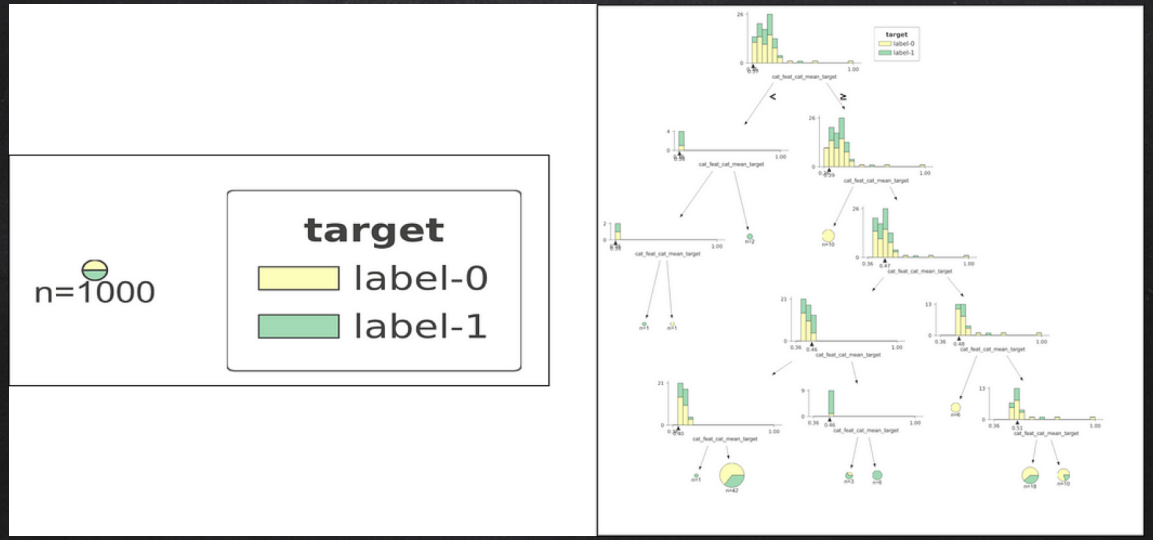
This technique is motivated from the validation techniques used to handle the time series data. When the distribution of target is not consistent with time, we need to take special care to not leak any information from the future while tuning hyper-parameters. Different ways to do hyper-parameter tuning or splitting validating set in time series data are discussed in this blogpost – Time Series Nested Cross-Validation.

The basic idea is to use **out of time validation** approaches as traditional k-fold methods do not work well in such cases. Similar to that, ordered target encoding relies on the concept of artificial time dependency. We calculate target statistic for each example by only using the target from the history. To generate the proxy concept of time, we randomly shuffle the data. To simplify, we randomize the rows and take running average of target label grouped by each category.



Running mean calculation.
Numbers are assigned randomly to each observation. Only 1-4 are used to find encoding for 5

But there is a problem in taking group-wise running averages. For the initial few rows in the randomized data, the mean encoding will have high variance as it only saw a few points from the history. But as it sees more data, the running average starts to get stable. This instability due to randomly chosen initial data points makes the initial part of dataset to have weak estimates. Catboost handles this problem by using more than one random permutations. It trains several models simultaneously, each one is trained on its own permutation. All the models share the same tree forest i.e the same tree structures. But leaf values of these models are different. Before building the next tree, CatBoost selects one of the models, which will be used to select tree structure. The structure is selected using this model, and is then used to calculate leaf values for all the models.



Let's see if the ordered encoding solves the failure cases we discussed for the leave one out and greedy target encoding techniques.

Case 1 (left plot). When we have all unique categories and greedy approach fails. We see that ordered encoding just can't simply find any good split at the first place. Case 2 (right plot). When we have all same category. Leave one out fails in this by giving perfect split in the first node. But ordered encoding struggles to find a good model as it should be.

Other Feature Engineering Tactics

Window based aggregations and linkages: Almost always, it helps to explicitly create window based aggregations from the user activity data and pass those are features to your classification models. Some examples of these features are:

- number of other transactions from the IP used in the current transaction over last 4 weeks,
- number of logins from the user in last 24 hours,
- country user logged in previous time,
- time when the user performed transaction for the first time,
- location of last order delivery,
- etc.

You can imagine that there can be hundreds of such variables. You can initially start with simpler ones and a single window.

You can imagine that there can be hundreds of such variables. You can initially start with simpler ones and a single window.

Why are such aggregations important?

We personally don't know the fraudsters and what kind of loopholes or behavior patterns they exhibit. Did they steal someone's credit card and are placing many orders in short span of time from your e-commerce store? or Did they find someone's username-password and trying to place orders from their legitimate account to a new location. In the former case, you would see "number of times a card is used in transaction within last 24 hours" *(a potential window aggregate)* to be very high, much higher than an average legitimate transaction. In the latter case, you would see location of order delivery different than the location of last order delivery. The "distance between current and previous order location" *(a potential linkage)* would be high, much higher than any legitimate transaction.

Supplement your data with external information: You can improve the accuracy and reliability of your raw activity data by adding new and supplemental information from third-party sources. Some raw features can be expanded to supplement with behavioral, demographic or geographic information. Classic example is IP address that can be easily enriched to geolocation, subnet, network, VPN, reputation score and all that supplemental can improve your classification models.

These are good primers for building a decent supervised classification model, it can outperform anomaly detection or semi supervised model. This can be further improved by leveraging the graphical structure of user activities. One way to keep things simple but still use complex relations that neural network can learn is to train a network on data and **extract embeddings from the pre-trained network as yet-another-feature for your tree based classifier**. You can do following:

Neural network based feature engineering:

1. **RNN, Transformer based network for sequential information**: The user activities are distributed across time and you get time-series style data. So use RNN or Transformer style architecture for capturing sequential information. If done well, it would do the similar job as **window based aggregations**. For example, the network can learn from the sequence of sign-ins about how to detect anomalous behaviors that can help in detecting account-takeover fraud.

2. Graph network for neighborhood and linkages: The user activities are connected with each other across several attributes. Generally, we use same email address to register in multiple websites, use same card for transactions in various places and use one or two devices (e.g. Mac and iPhone). Similar story is true for fraudsters, but due to their ill will, their behavior would be different than normal users. For e.g. instead of using same card for transactions, they might rotate several cards and use several email addresses to register, but their device might be remain same. You would see multiple of those fraudulent transactions **connected** with device as common thread. Graph networks can leverage such connections and provide better information that you might otherwise miss.

```python
def add_datepart(df, field_name, prefix=None, drop=True, time=False):
    "Helper function that adds columns relevant to a date in the column `field_name` of `df`."
    make_date(df, field_name)
    field = df[field_name]
    prefix = ifnone(prefix, re.sub('[Dd]ate$', '', field_name))
    attr = ['Year', 'Month', 'Week', 'Day', 'Dayofweek', 'Dayofyear', 'Is_month_end', 'Is_month_start',
            'Is_quarter_end', 'Is_quarter_start', 'Is_year_end', 'Is_year_start']
    if time: attr = attr + ['Hour', 'Minute', 'Second']
    # Pandas removed `dt.week` in v1.1.10
    week = field.dt.isocalendar().week.astype(field.dt.day.dtype) if hasattr(field.dt, 'isocalendar') else field.dt.week
    for n in attr: df[prefix + n] = getattr(field.dt, n.lower()) if n != 'Week' else week
    mask = ~field.isna()
    df[prefix + 'Elapsed'] = np.where(mask,field.values.astype(np.int64) // 10 ** 9,np.nan)
    if drop: df.drop(field_name, axis=1, inplace=True)
    return df
```

Reference:
https://github.com/fastai/fastai/blob/785ec908c7f75a272c718dd0df53822bdf5e9e0e/fastai/tabular/core.py#L26

Reference

https://web.archive.org/web/20200924113639/https://roamanalytics.com/2016/10/28/are-categorical-variables-getting-lost-in-your-random-forests/

https://www.kaggle.com/c/zillow-prize-1/discussion/38793

https://towardsdatascience.com/getting-deeper-into-categorical-encodings-for-machine-learning-2312acd347c8