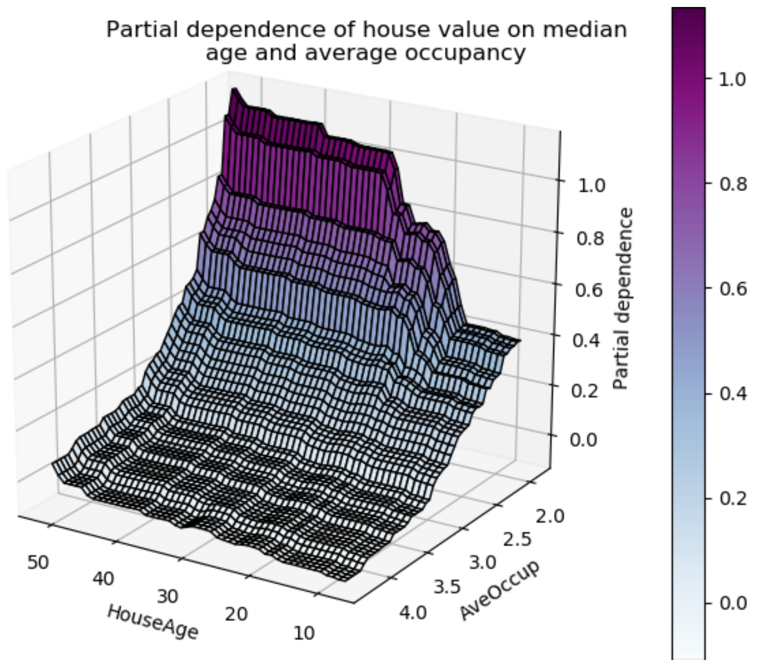
Abhay Pawar Follow
Machine Learning @ Instacart. Blogger. Loves data. www.linkedin.com/in/abhayspawar/
Nov 4 · 7 min read

My secret sauce to be in top 2% of a kaggle competition

Competing in kaggle competitions is fun and addictive! And over the last couple of years, I developed some standard ways to explore features and build better machine learning models. These simple, but powerful techniques helped me get a top 2% rank in Instacart Market Basket Analysis competition and I use them outside of kaggle as well. So, let's get right into it!

One of the most important aspects of building any supervised learning model on numeric data is to understand the features well. Looking at partial dependence plots of a model helps you understand how the model's output changes with any feature.



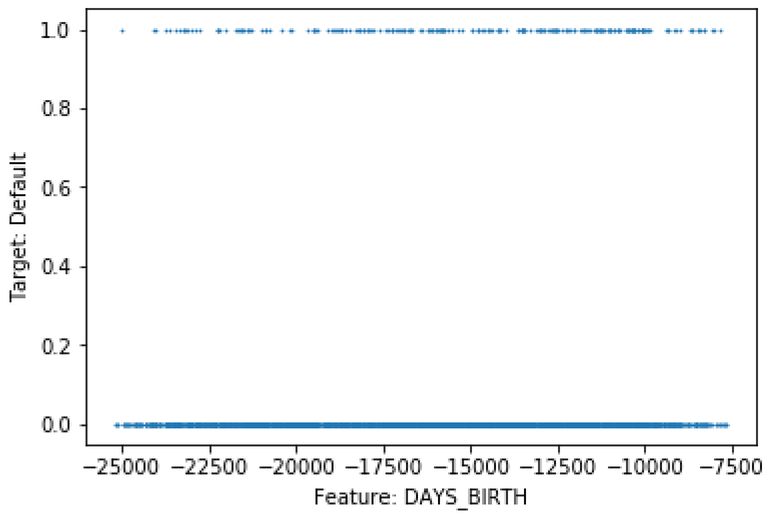
source

But, the problem with these plots is that they are created using a trained model. If we could create these plots from train data directly, it could help us understand the underlying data better. In fact, it can help you with all the following things:

1. Feature understanding
2. Identifying noisy features (**the most interesting part!**)
3. Feature engineering
4. Feature importance
5. Feature debugging
6. Leakage detection and understanding
7. Model monitoring

In order to make it easily accessible, I decided to put these techniques into a python package featexp and in this article, we'll see how it can be used for feature exploration. We'll use the application dataset from Home Credit Default Risk competition on Kaggle. The task of the competition is to predict defaulters using the data given about them.

1. **Feature Understanding**



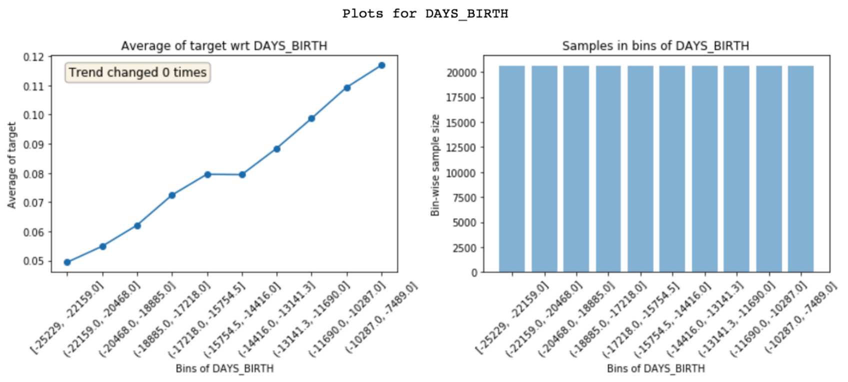
Scatter plot of feature vs. target doesn't help

If dependent variable (target) is binary, scatter plots don't work because all points lie either at 0 or 1. For continuous target, too many data points make it difficult to understand the target vs. feature trend.

Featexp creates better plots which help with this problem. Let’s try it out!

```
from featexp import get_univariate_plots

# Plots drawn for all features if nothing is passed in
get_univariate_plots(data=data_train, target_col='targ
                          features_list=['DAYS_BIRTH'], bin
```



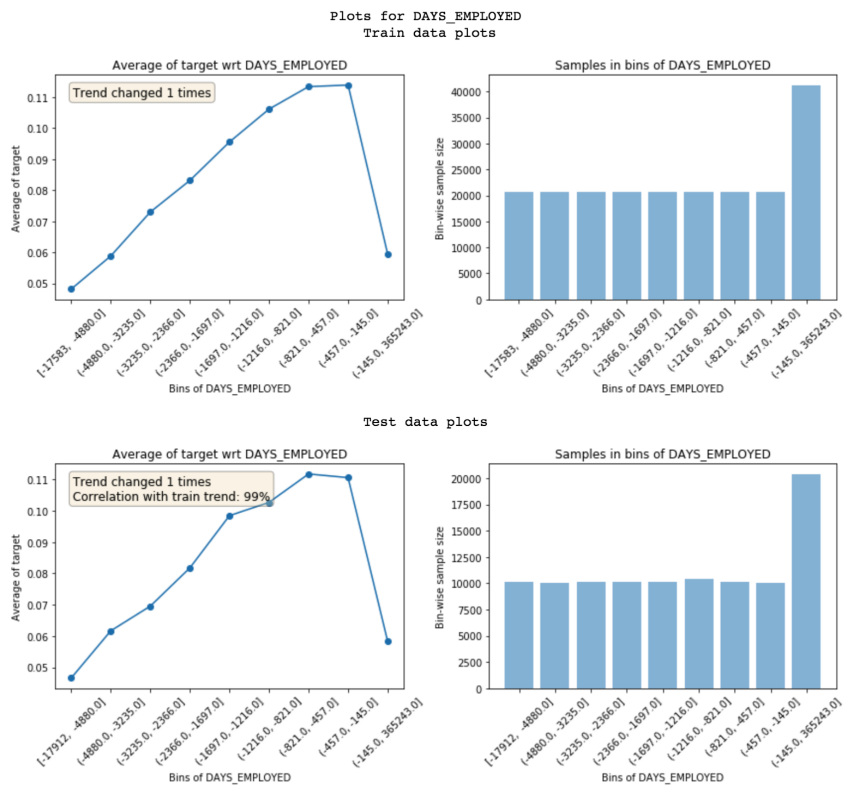
Feature vs. target plot of DAYS_BIRTH (age)

Featexp creates equal population bins (X-axis) of a numeric feature. It then calculates target’s mean in each bin and plots it in the left-hand side plot above. In our case, target’s mean is nothing but default rate. The plot tells us that customers with high negative values for DAYS_BIRTH (higher age) have lower default rates. This makes sense since younger people are usually more likely to default. These plots help us understand what the feature is telling about customers and how it will affect the model. The plot on the right shows number of customers in each bin.

2. Identifying noisy features

Noisy features lead to overfitting and identifying them isn’t easy. In featexp, you can pass a test set and compare feature trends in train/test to identify noisy ones. This test set is not the actual test set. Its your local test set/validation set for which you know target.

```
get_univariate_plots(data=data_train, target_col='targ
```



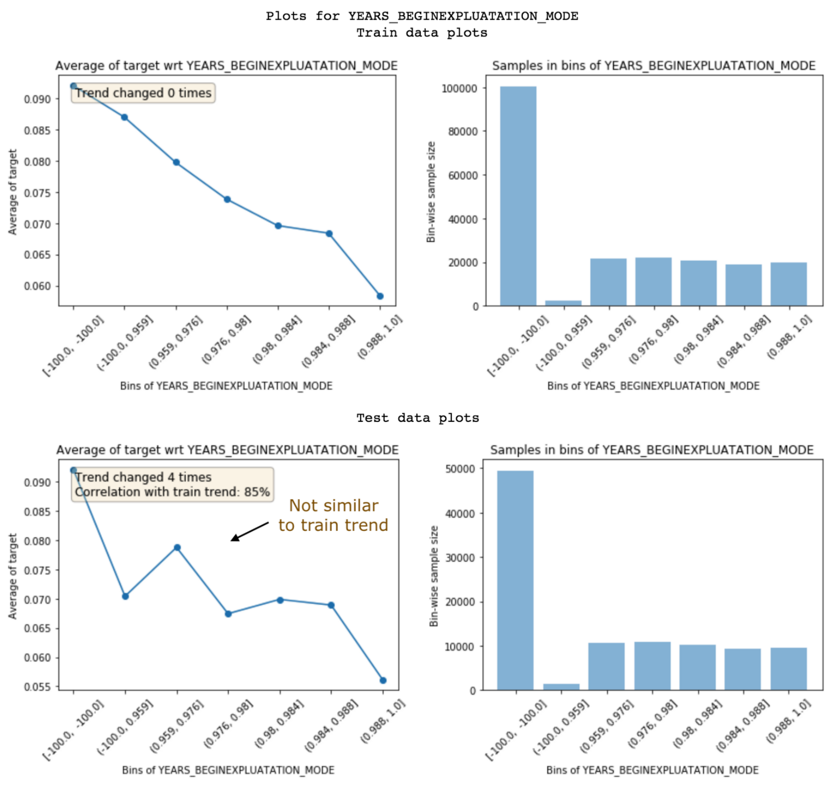
Comparison of feature trends in train and test

Featexp calculates two metrics to display on these plots which help with gauging noisiness:

1. **Trend correlation** (seen in test plot): If a feature doesn't hold same trend w.r.t. target across train and evaluation sets, it can lead to overfitting. This happens because the model is learning something which is not applicable in test data. Trend correlation helps understand how similar train/test trends are and mean target values for bins in train & test are used to calculate it. Feature above has 99% correlation. Doesn't seem noisy!

2. **Trend changes**: Sudden and repeated changes in trend direction could imply noisiness. But, such trend change can also happen because that bin has a very different population in terms of other features and hence, its default rate can't really be compared with other bins.

Feature below is not holding the same trend and hence, has a low trend correlation of 85%. These two metrics can be used to drop noisy features.



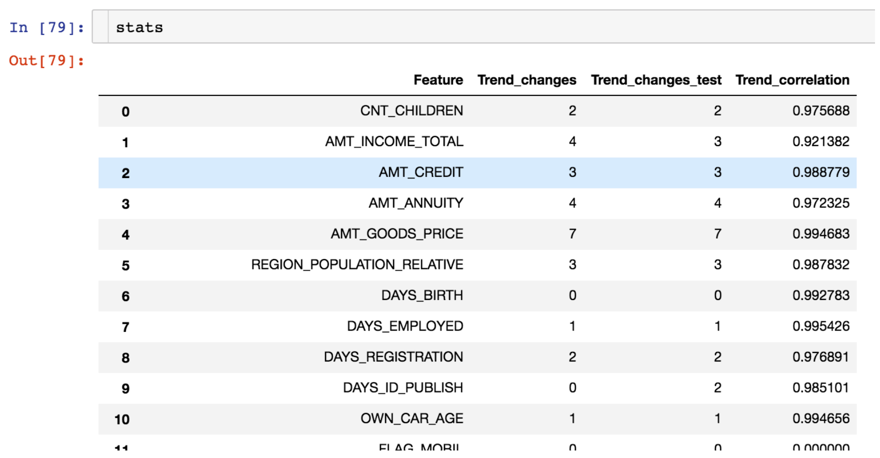
Example of noisy feature

Dropping low trend-correlation features works well when there are a lot of features and they are correlated with each other. It leads to less overfitting and other correlated features avoid information loss. It's also important to not drop too many important features as it might lead to a drop in performance. **Also, you can't identify these noisy features using feature importance because they could be fairly important and still be very noisy!**

Using test data from a different time period works better because then you would be making sure if feature trend holds over time.

get_trend_stats() function in featexp returns a dataframe with trend correlation and changes for each feature.

```python
from featexp import get_trend_stats
stats = get_trend_stats(data=data_train, target_col='t
```

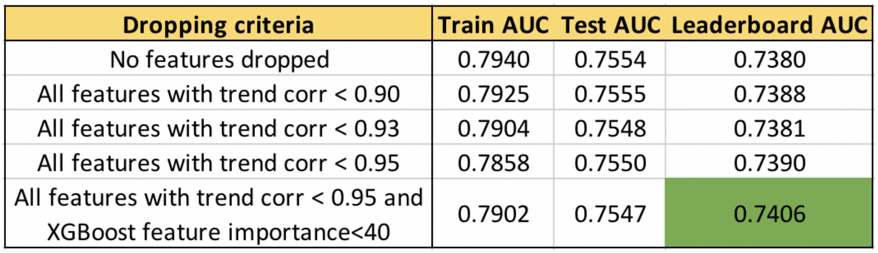


```
In [79]: stats
Out[79]:
```

	Feature	Trend_changes	Trend_changes_test	Trend_correlation
0	CNT_CHILDREN	2	2	0.975688
1	AMT_INCOME_TOTAL	4	3	0.921382
2	AMT_CREDIT	3	3	0.988779
3	AMT_ANNUITY	4	4	0.972325
4	AMT_GOODS_PRICE	7	7	0.994683
5	REGION_POPULATION_RELATIVE	3	3	0.987832
6	DAYS_BIRTH	0	0	0.992783
7	DAYS_EMPLOYED	1	1	0.995426
8	DAYS_REGISTRATION	2	2	0.976891
9	DAYS_ID_PUBLISH	0	2	0.985101
10	OWN_CAR_AGE	1	1	0.994656
11	FLAG_MOBIL	0	0	0.000000

Dataframe returned by get_trend_stats()

Let's actually try dropping features with low trend-correlation in our data and see how results improve.



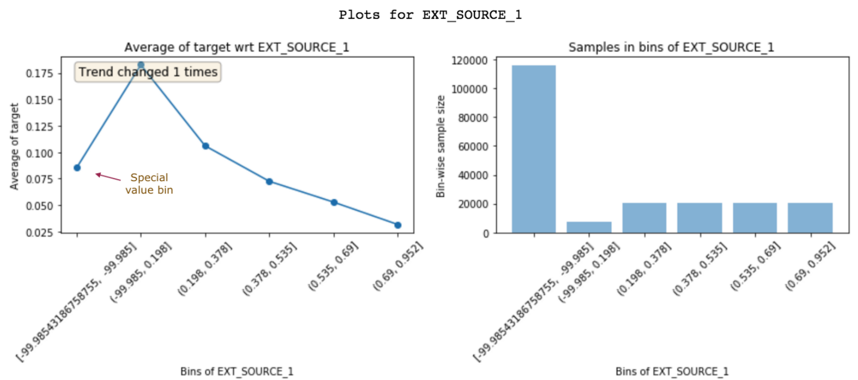
Dropping criteria	Train AUC	Test AUC	Leaderboard AUC
No features dropped	0.7940	0.7554	0.7380
All features with trend corr < 0.90	0.7925	0.7555	0.7388
All features with trend corr < 0.93	0.7904	0.7548	0.7381
All features with trend corr < 0.95	0.7858	0.7550	0.7390
All features with trend corr < 0.95 and XGBoost feature importance<40	0.7902	0.7547	0.7406

AUC for different feature selections using trend-correlation

We can see that higher the trend-correlation threshold to drop features, higher is the leaderboard (LB) AUC. Not dropping important features further improves LB AUC to 0.74. It's also interesting and concerning that test AUC doesn't change as much as LB AUC. Getting your validation strategy right such that local test AUC follows LB AUC is also important. Whole code can be found in featexp_demo notebook.

3. Feature Engineering

The insights that you get by looking at these plots help with creating better features. Just having a better understanding of data can lead to better feature engineering. But, in addition to this, it can also help you in improving the existing features. Let's look at another feature EXT_SOURCE_1:



Feature vs. target plot of EXT_SOURCE_1

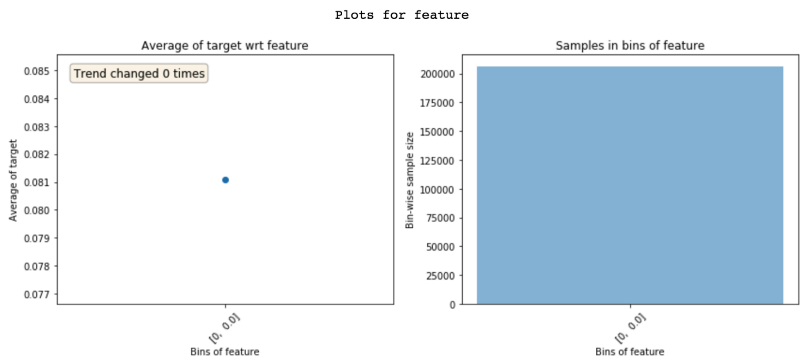
Customers having a high value of EXT_SOURCE_1 have low default rates. But, the first bin (~8% default rate) isn't following the feature trend (goes up and then down). It has only negative values around -99.985 and a large population. This probably implies that these are special values and hence, don't follow the feature trend. Fortunately, non-linear models won't have a problem learning this relationship. But, for linear models like logistic regression, such special values and nulls (which will be shown as a separate bin) should be imputed with a value from a bin with similar default rate instead of simply imputing with feature mean.

4. Feature importance

Featexp also helps you with gauging feature importance. DAYS_BIRTH and EXT_SOURCE_1 both have a good trend. But, population for EXT_SOURCE_1 is concentrated in special value bin which tells that it might not be as important as DAYS_BIRTH. Based on XGBoost model's feature importance, DAYS_BIRTH is actually more important than EXT_SOURCE_1.

5. Feature debugging

Looking at Featexp's plots helps you in capturing bugs in complex feature engineering codes by doing these two things:



Zero variation features show only a single bin

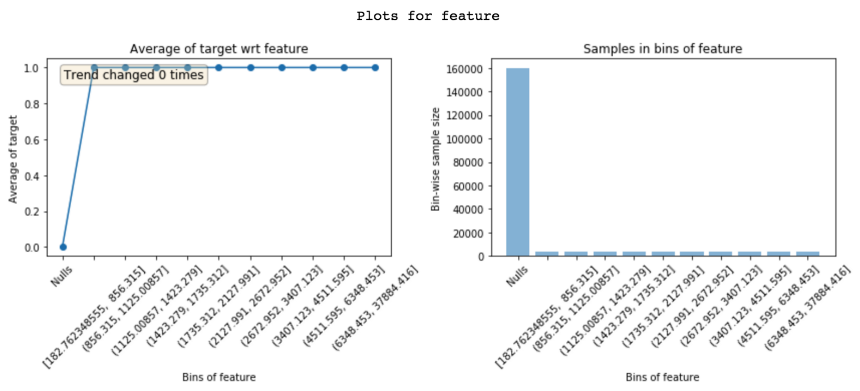
1. Checking if the feature's population distribution looks right. I've personally encountered extreme cases like above numerous times due to minor bugs.

2. Always hypothesize what the feature trend will look like before looking at these plots. Feature trend not looking like what you expected might hint towards some problem. **And frankly, this process of hypothesizing trends makes building ML models much more fun!**

6. Leakage Detection

Data leakage from target to features leads to overfitting. Leaky features have high feature importance. But, understanding why leakage is happening in a feature is difficult. Looking at featexp plots can help you with that.

The feature below has 0% default rate in 'Nulls' bin and 100% in all other bins. Clearly, this is an extreme case of leakage. This feature has a value only when the customer has defaulted. Based on what the feature is, this could be because of a bug or the feature is actually populated only for defaulters (in which case it should be dropped). **Knowing what the problem is with leaky feature leads to quicker debugging.**



Understanding why a feature is leaky

7. Model Monitoring

Since featexp calculates trend correlation between two data sets, it can be easily used for model monitoring. Every time the model is re-trained, the new train data can be compared with a well-tested train data (typically train data from the first time you built the model). Trend correlation can help you monitor if anything has changed in feature w.r.t. its relationship with target.

• • •

Doing these simple things have always helped me in building better models in real life and on kaggle. With featexp it takes 15 minutes to look at these plots and it's definitely worth it as you won't be flying blind after that.

What other tricks and tips do you find useful for exploring features? I'm always looking for feedback. Let me know in comments or reach out to me at abhayspawar@gmail.com. Thank you for reading!