Data Analytics Data Exploration Classification
Feature Engineering Feature Selection Machine Learning

2019-08-22-Data-competition-From-0-to-1-Part-II

Feature Engineering Techniques

Posted by kaokao on August 22, 2019

Course: github kaokao_proc

← CATALOG

Introduction

Feature engineering, the process creating new input features for machine learning, is one of the most effective ways to improve predictive models.

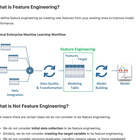
> "Coming up with features is difficult, time-consuming, requires expert knowledge. 'Applied machine learning' is basically feature engineering."—Andrew Ng

> "... some machine learning projects succeed and some fail. What makes the difference? Easily the most important factor is the features used."—Pedro Domingos

What Is Feature Engineering?

We define feature engineering as creating new features from your existing ones to improve model performance.

Typical Enterprise Machine Learning Workflow



What Is Not Feature Engineering?

That means there are certain steps we do not consider to be feature engineering:

- We do not consider **initial data collection** to be feature engineering.
- We do not consider **data sanity and transformation** to be feature engineering.
- We do not consider removing duplicates, handling missing values, or fixing inconsistent classes to be feature engineering. We put these under **data cleaning**.
- We do not consider **scaling or normalization** to be feature engineering because these steps belong inside the cross-validation loop.
- We do not consider dimensionality reduction or PCA to be feature engineering. These steps are belong inside your cross-validation loop.

Feature Engineering cycle

- Hypotheses
  - Domain knowledge
  - Prior experience
  - Intuition
  - ML model feedback
- Validate hypotheses
  - Cross-validation
  - Measurement of desired metrics
  - Avoid leakage

Processing

Imputation

Missing values are one of the most common problems you can encounter when you try to prepare your data for machine learning.

The most simple solution for the missing values is to drop the rows or the entire column. There is not an optimum threshold for dropping but you can use 70% as an example value and try to drop the rows and columns which have missing values with higher than this threshold.

```
threshold = 0.7
#Dropping columns with missing value more than threshold
data = data[data.columns[data.isnull().mean() < threshold]]
#Dropping rows with missing value more than threshold
data = data.loc[data.isnull().mean(axis=1) < threshold]
```

Numerical Imputation

```
#Filling missing value with 0
data = data.fillna(0)
#Filling missing value with mean of the column
data = data.fillna(data.mean())
```

Categorical Imputation

```
#Fill NA/NaN for categorical columns
data[cat_cols] = data[cat_cols].fillna('missing')
y_dummy[y_dummy.isnull()]
#output
data['column_name'].fillna('Other', inplace=True)
```

Handling Outliers

Best way to detect the outliers is to demonstrate the data visually. All other statistical methodologies are open to making mistakes, whereas visualizing the outliers gives a chance to take a decision with high precision.

Outlier Detection with Standard Deviation

```
#Dropping the outliers rows with standard deviation
factor = 3
upper_lim = data['column'].mean() + data['column'].std() * factor
lower_lim = data['column'].mean() - data['column'].std() * factor

data = data[(data['column'] < upper_lim) & (data['column'] > lower_lim)]
```

Outlier Detection with Percentiles

```
#Dropping the outliers rows with percentiles
upper_lim = data['column'].quantile(.95)
lower_lim = data['column'].quantile(.05)

data = data[(data['column'] < upper_lim) & (data['column'] > lower_lim)]

#Dropping the outliers rows with percentiles
upper_lim = data['column'].quantile(.95)
lower_lim = data['column'].quantile(.05)
data.loc[(df[column] > upper_lim),column] = upper_lim
data.loc[(df[column] < lower_lim),column] = lower_lim
```

Log Transform

The benefits of log transform:

- It helps to handle skewed data and after transformation, the distribution becomes more approximate to normal.
- It also decreases the effect of the outliers, due to the normalization of magnitude differences and the model become more robust.

```
# Numerical Binary Encoding
data['log_1'] = data['column1'].map('value1':value1, 'value2':value2)
```

Feature Engineering Techniques

Indicator Variables

The first type of feature engineering involves using indicator variables to isolate key information.

- **Indicator variable from thresholds**: create an indicator variable for age >= 18 to indicate whether a person is an adult or not.
- **Indicator variable from multiple features**: You're predicting real estate prices and you have the features `_bedrooms` and `_bathrooms`. It houses with 0 beds and 2 baths probably means that the house is a mobile home that people live in while they're on vacation. **Indicator variable for special events**: You're modeling weekly sales for an e-commerce site. You can create two indicator variables for the weeks of 12/25 and 01/01.
- **Indicator variable for groups of classes**: You're analyzing WFH coronavirus and your dataset has the categorical feature traffic_source. You could create an indicator variable for paid_traffic by flagging observations with traffic_source values of `SEARCH` or `DISPLAY`.

Interaction Features

This type of feature engineering involves highlighting interactions between two or more features. Thus, we hope features can be combined to provide more information than the sum of the individuals. Do you ever `gender` x `home?` to `height` and `gender` x `partner?` to `height` in the scenarios?

Note: We don't recommend using an automated loop to create interactions for all your features. This leads to feature explosion.

- **Sum of two features**
- **Difference between two features**
- **Product of two features**
- **Quotient of two features**

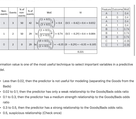
Feature Representation

Your data won't always come in the ideal format. You should consider if you'd gain information by representing the same feature in a different way.

- **Date and time features**: Let's say you have the feature `purchase_datetime`. It might be useful to break this down into `purchase_hour` and `purchase_day_of_week`. Remember aggregation observations to create features such as `purchase_count_last_7_days`, `purchase_avg_last_14_days`, `purchase_day_of` `purchase_week_mean` etc.

```
from datetime import date

data = pd.DataFrame({'date':
'01-01-2000',
'01-02-2000',
'23-03-2000',
'23-06-2000',
'12-12-2000',
'24-12-2000',
'11-08-2000'
})
#Converting string to date
data['date'] = pd.to_datetime(data.date, format="%d-%m-%Y")

#Extracting year
data['year'] = data['date'].dt.year

#Extracting month
data['month'] = data['date'].dt.month

#Finding the difference between the dates
data['period_passed'] = date.today().year - data['date'].dt.year

#Extracting second if the timestamp exists
data['second'] = data['date'].dt.second # output will be all zeros as the timestamp was

#Extracting the weekday name of the date
data['day_name'] = data['date'].dt.dayofweek
```

- **Binning (分箱)**: Binning can be applied on both categorical and numerical data. When there is a feature with many classes that have low sample counts, You can try grouping similar classes and then grouping the remaining ones into a single 'Other' class (**Grouping sparse classes**).



The main motivation of binning is to make the model more robust and prevent overfitting, however, it has a cost to the performance. Every time you bin something, you sacrifice information and make your data more regularized.

```
# Numerical Binning Example
data['fare'] = pd.qcut(data['fare'], q=4, labels=['Low', 'Med', 'High', 'Very'])

#Categorical Binning Example
data['Country'].value_counts()
data['Country'].isin(['USA','Mexico','Russia',
data['Country'].isin(['Germany','Italy',
data['Country'].isin(['Japan','China','India',
data['Country'].isin(['Brazil',
choices = ['Europe', 'Asia', 'North America', 'South America']

data['continent'] = np.select(conditions, choices, default='Other') # Grouping sparse c
```

For numerical columns, except for some obvious overfitting cases, binning might be rationalized to have lots of algorithms, due to its effect on model performance. However, for categorical columns, the labels with low frequencies probably affect the robustness of statistical models negatively.

- **One Hot encoding**: Transform categories into individual binary (0 or 1) features

A	B	C	D
0	0	1	0
0	1	0	0
1	0	0	0
0	0	0	1

Feature	Feature_A	Feature_B	Feature_C	Feature_D
A	1	0	0	0
B	0	1	0	0
C	0	0	1	0
D	0	0	0	1

- **Frequency Encoding**: Encoding of categorical levels of feature to values between 0 and 1 based on their relative frequency

Feature
[0,0,0,0,0,0,0,0]
[1,1,1,0,0,0,0,0]
[1,1,1,1,1,1,1,0]

Feature	Encoded Feature
[0,0,0,0,0,0,0,0]	0.00
[1,1,1,0,0,0,0,0]	0.38
[1,1,1,1,1,1,1,0]	0.88

- **Target mean encoding**

A	B	C	D
0.75	0.5	0.67	0.5

Feature	Outcome	MeanEncode
A	1	0.75
A	1	0.75
A	1	0.75
A	0	0.75
B	1	0.50
B	0	0.50
C	1	0.67
C	1	0.67
C	0	0.67
D	1	0.50
D	0	0.50

Supplement

Weighted Average It is better to calculate weighted average of the overall mean of the training set and the mean of the level:

$$f(n) = \mathrm{mean}(\mathrm{level}) * (1 - f(n)) + \mathrm{mean}\,(\mathrm{dataset}) \qquad (1)$$

Where $f(n) = \frac{1}{1+e^{(-\frac{n-k}{f})}}$, k = frequency, f = inflection point, f = steepness. We can get graph in desmos calculator.

To avoid overfitting we could use leave-one-out schema

Feature	Outcome
A	1
A	1
A	1
A	0
B	1
B	0
C	1
C	1
C	0
D	1
D	0

LOOencode
0.66
0.66
0.66
0.66
0.50
0.50
1.00
1.00
0.50
0.50
1.00

Weight of Evidence $WoE = \ln \frac{P(D=1|x_i)}{P(D=0|x_i)}$

To avoid division by zero:

$$WoE_i = \ln\left(\frac{\text{Number of non-events in a group}_i + 0.5/\text{Number of non-events}}{\text{Number of events in a group}_i + 0.5/\text{Number of events}}\right) \qquad (2)$$

Information Value $IV = \sum_i^N (\%\ \text{non-events}_i - \%\ \text{events}_i) * WoE_i$

Age	Events	Non-Events	% of Events	% of Non-Events	WoE	IV
10	1	0	$\frac{1+0.5}{10}$ = 0.15	$\frac{0+0.5}{90}$ = 0.006	$\ln(0.15/0.006) = 3.22$	$(0.15-0.006)*3.22 = 0.46$
15	2	10	0.25	0.12	$\ln(0.25/0.12) = 0.73$	$(0.25-0.12)*0.73 = 0.09$
20	3	30	$\frac{3+0.5}{10}$ = 0.35	$\frac{30+0.5}{90}$ = 0.33	$\ln(0.35/0.33) = 0.06$	$(0.35-0.33)*0.06 = 0.001$
25	4	40	0.45	0.45	$\ln(0.45/0.45) = 0.00$	$(0.45-0.45)*0.00 = 0.00$
30	5	10	0.55	0.12	$\ln(0.55/0.12) = 1.52$	$(0.55-0.12)*1.52 = 0.65$
						IV = 1.20

Information value is one of the most useful technique to select important variables in a predictive model.

- Less than 0.02, then the predictor is not useful for modeling (separating the Goods from the Bads)
- 0.02 to 0.1, then the predictor has only a weak relationship to the Goods/Bads odds ratio
- 0.1 to 0.3, then the predictor has a medium strength relationship to the Goods/Bads odds ratio.
- 0.3 to 0.5, then the predictor has a strong relationship to the Goods/Bads odds ratio.
- 0.5, suspicious relationship (Check once)

Information value increases as bins / groups increases for an independent variable. We have to be careful when there are more than 20 bins as some bins may have a very low number of events and non-events.

Textual Data

- Bag of Words: extract tokens from text and use their occurrences (or TFIDF weights) as features:
  - CountVectorizer, TfidfVectorizer
- NLP techniques:
  - Stemming/lemmatizing words (not always necessary):
  - Convert all words to lower case
  - Stemming for English words:
  - pmorph
- Deep Learning for textual data:
  - Word2Vec embeddings, bag of pre-trained vzs
  - Hate computer "semantic distance" between tokens/words

External Data

An underused type of feature engineering is bringing in external data. This can lead to some of the biggest breakthroughs in performance.

- **External APIs**: `modis_data` etc
- **Geocoding**: Let's say you have `street_address`, `city`, and `state`. Well, you can use geocoding APIs to get the latitude and longitude. This will allow you to calculate features such as `travel_distance_miles`, `dist` etc.
- **Other sources of the same data**

Error Analysis (Post-Modeling)

This final type of feature engineering left Lizzie falls under a pseudo-post-analysis step. This is performed after training your first model.

Error analysis is a broad term that refers to analyzing the mistakes/flaws of high error observations. Form your model and iterating on your next steps for improvement.

- **Start with large errors**
- **Segment by classes**
- **Unsupervised clustering**
- **Ask colleagues or domain experts**

Conclusion

Good features to engineer...

- Can be computed for future observations.
- Are available in production.
- Are informed by domain knowledge or exploratory analysis.
- Must have the potential to be predictive. Don't just create features for the sake of it.
- **Never touch the target variable.** （非常必要）

Reference

- https://towardsdatascience.com/feature-engineering-for-machine-learning-3a5e293a5114
- https://medium.com/open-machine-learning-course/open-machine-learning-course-topic-6-feature-engineering-and-feature-selection-8b94f870706a

PREVIOUS
2019-08-20-2-BEST-OPTIMIZATION-CODE

NEXT
2019-08-20-1-BEST-OPTIMIZATION-WORK-PROCESS

分享至

未登录用户 ×

点赞 0人

登录 Kaokaoer 登录

暂无数据

未登录一下能怎么样？

FEATURED TAGS

Data Analytics Data Exploration Feature Engineering Processing EDA Machine

FRIENDS

Kaokaoer Kaokaoer

Copyright © 2019 Kaokaoer
Theme by Gitbook Gitbook 3