Automunge Influence

Anonymous Author(s)
Affiliation
Address
email

Abstract

The Automunge open source python library platform for preparing tabular data pipelines includes automated methods to assemble feature importance evaluation metrics based on shuffle permutation, including metrics associated with source columns as well as relative metrics associated with derived columns originating from the same source column. This paper will introduce a novel method for applying shuffle permutation to evaluate influence of different segments of a numeric feature set's distribution towards predictive accuracy, such as may be helpful to identify hedging targets based on tail influence.

1 Introduction

Wanted to quickly share some detail on a useful method that we've recently added to the Automunge library, intended for use to gauge influence of tail events toward a predictive model by making use of feature importance evaluation by shuffle permutation [2], such as for instance may help evaluate influence to a model from right and left tails of some particular numeric variable's distribution. In the context of a risk assessment, this type of operation could help identify hedging candidates for instance.

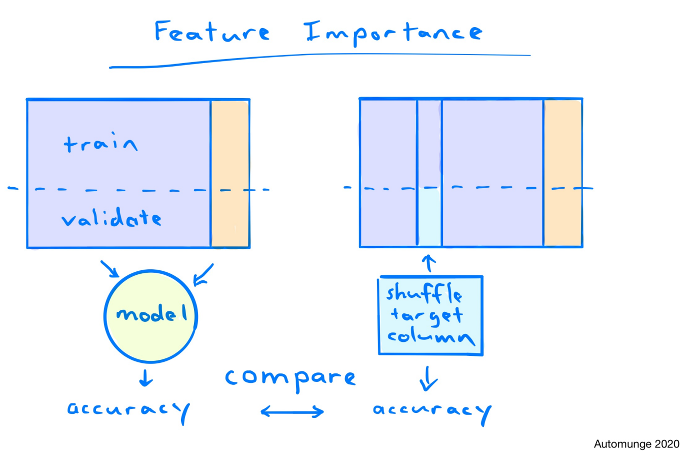

Figure 1: Feature importance by shuffle permutation

2 Tail Bins

The method makes use of the “tail bins” transformation available in the Automunge library as ‘tlbn’. This transform has some relation to the ‘bnep’ family of transforms which grain numerical sets to equal population bins and accept a parameter ‘bincount’ to specify a number of returned bins (which may either be one-hot encoded via the ‘bnep’ family transforms or alternatively ordinal encoded with the ‘bneo’ family transforms). Here is an example subset of these two variants of equal population bins output from application to a common demonstration set from Kaggle:

df_train['LotArea'][0:9]	'bnep' (one-hot encoding equal population bins) bincount = 5					'bneo' (equal population ordinal) bincount = 5
LotArea	LotArea_bnep_0	LotArea_bnep_1	LotArea_bnep_2	LotArea_bnep_3	LotArea_bnep_4	LotArea_bneo
8450	0	1	0	0	0	1
9600	0	0	1	0	0	2
11250	0	0	0	1	0	3
9550	0	0	1	0	0	2
14260	0	0	0	0	1	4
14115	0	0	0	0	1	4
10084	0	0	1	0	0	2
10382	0	0	0	1	0	3
6120	1	0	0	0	0	0

Figure 2: Demonstration of transformations applied to the Kaggle set “House Prices Advanced Regression Techniques”

Here we’re mostly interested in the ‘bnep’ version in order to demonstrate the ‘tlbn’ derivation below. Equal population means that the demarcation for borders between what values will be aggregated into separate bins is determined such as to promote an approximately equal number of entries in each bin (fit to properties of the train set). The implementation makes use of pandas [1] “qcut” in this derivation to derive those intervals, which first and last intervals then converted to open ended (-inf / +inf), and then pandas “cut” to transform the related intervals into associated bins. The resulting aggregated bins are then converted from ordinal to one-hot encoding for a multi-column returned set of bins.

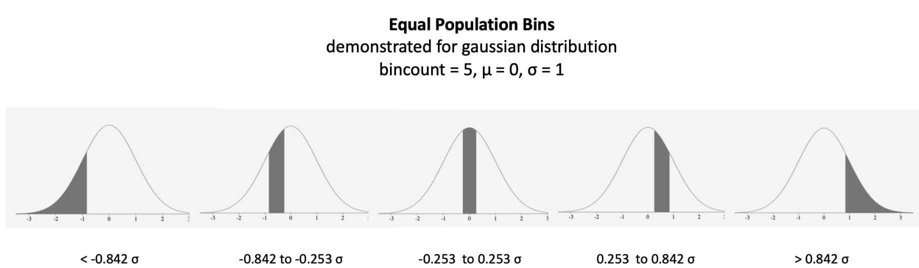

Figure 3: Equal population bins for Gaussian distribution

The ‘tlbn’ transform differs from ‘bnep’ in that the returned bins following this derivation are then converted from boolean activations as to recover the original numeric entries corresponding to each bin. To demonstrate, if a set of entries [2, 1, 7, 4, 9, 10] had been converted to two ‘bnep’ bins with intervals (-inf, 5.5), (5.5, inf), and thus the two columns [1, 1, 0, 1, 0, 0] and [0, 0, 1, 0, 1, 1], the activations in those two columns could then be converted to recapture the corresponding numeric entries as [2, 1, 0, 4, 0, 0] and [0, 0, 7, 0, 9, 10]. The obvious issue with this transformation is the potential overlap between the returned values and the 0 designation for entries that did not fall into a bin. The solution is to scale the entries with a min/max transform such as to fall within a designated range, which we apply to the range 0 to 1, and then to use an arbitrary point outside of that range as a signal to training of infill, which we selected -1 for this purpose. Another complication is that since the first and last intervals are unconstrained for subsequent test data, some of the resulting values may fall outside of the 0 to 1 range. This is ok as long as there is no overlap with the infill signal of -1, so to ensure we scale data such that test data tail events outside of the training data range are transformed after min/max to fall >1, which in practice means that we reverse the order of values returned from the first bin (the one constrained by -inf) by way of a sign variation on the min/max equation.

df_train['LotArea'][0:9]	'bnep' (one-hot encoding equal population bins) bincount = 5					'bneo' (equal population ordinal) bincount = 5
LotArea	LotArea_bnep_0	LotArea_bnep_1	LotArea_bnep_2	LotArea_bnep_3	LotArea_bnep_4	LotArea_bneo
8450	0	1	0	0	0	1
9600	0	0	1	0	0	2
11250	0	0	0	1	0	3
9550	0	0	1	0	0	2
14260	0	0	0	0	1	4
14115	0	0	0	0	1	4
10084	0	0	1	0	0	2
10382	0	0	0	1	0	3
6120	1	0	0	0	0	0

	'tlbn' (tail distribution bins) bincount = 5				
LotArea	LotArea_tlbn_0	LotArea_tlbn_1	LotArea_tlbn_2	LotArea_tlbn_3	LotArea_tlbn_4
8450	-1	0.799767	-1	-1	-1
9600	-1	-1	0.574174	-1	-1
11250	-1	-1	-1	0.523909	-1
9550	-1	-1	0.538582	-1	-1
14260	-1	-1	-1	-1	0.010117
14115	-1	-1	-1	-1	0.009403
10084	-1	-1	0.918707	-1	-1
10382	-1	-1	-1	0.091552	-1
6120	0.165859	-1	-1	-1	-1

Figure 4: 'tlbn' transform in comparison to 'bnep'

The purpose of this type of transformation is to convert a single column numerical set to a collection of numerical sets corresponding to different segments of the original set's distribution while maintaining entry correspondence with the other rows of the associated tabular data set from which it originated. By segregating the different portions of a distribution into distinct columns, it becomes possible to conduct a feature importance evaluation which measures the relative importance of each of these distribution segments in relation to the others towards predictive accuracy of a target label set, which can be performed either in automunge(.) by activating the featureselection parameter or postmunge(.) with the featureeval parameter. The reports generated from these operations, such as those automunge(.) reports returned in postprocess_dict['FS_sorted'] or postmunge(.) reports returned in postreports_dict['FS_sorted'], demonstrate two derived metrics associated with a column's feature importance. The first metric 'metric' describes the importance of source columns towards predictive accuracy by shuffling all columns derived from each target source column to derive the accuracy metric. For our 'tlbn' method we're actually interested in the second derived metric 'metric2', which describes the relative importance of each column derived from a single source column towards predictive accuracy achieved by shuffling all but the target column derived from the same source column to derive the accuracy metric (where for the first metric a larger value implies greater source column importance, and for the second metric a smaller value implies greater relative importance between associated derived columns). We can view the derived columns sorted by the second metric by inspecting the associated FS_sorted report returned from automunge(.) as postprocess_dict['FS_sorted']['metric2_column_key']['(source column header)'], with the order of returned 'tlbn' columns, as whose suffix numbering designation would indicate order of values from the distribution, could indicate whether tail portions of the distribution (first and last bins) have greater relative importance to predictive accuracy of our model.

```
{'LotArea_tlbn_4': 0.00013460335610815388,
 'LotArea_tlbn_0': 0.0004405612269814396,
 'LotArea_tlbn_2': 0.0006103751135602131,
 'LotArea_tlbn_1': 0.0006640267200068717,
 'LotArea_tlbn_3': 0.0006707872807621973}
```

Figure 5: 'metric2' for relative importance between features derived from same source column

Here we see that in this demonstration based on the Kaggle House Prices competition, in which the target label set is a collection of sale prices for home sales and the feature sets are the corresponding home properties, the target feature of our evaluation 'LotArea' does appear to have tail sensitivity. Specifically, we see that in our five bin returned set (0–4), the top bin (containing the largest LotAreas) appears to be the most influential to predictive accuracy of our home price model based on the smallest metric2 value, and the second most influential is the bottom bin (containing the smallest LotAreas), which also agrees with intuition for this application. We should note that while this type of evaluation does demonstrate importance of distribution segments from a target numeric feature set to predictive accuracy, one thing it doesn't do is demonstrate the risk/reward paradigm of that influence (e.g. do increasing lot areas increase or decrease home sale prices). In addition to simple correlation, there is another method that we recommend as a useful stress test heuristic to detect fragility [3]. However, correlation is not reliable in big data applications, and the stress test fragility heuristic may be blind to multimodal distributions. The methods demonstrated here could be considered an alternative to such heuristics which make use of the power of machine learning to navigate exotic distributions, within the context of simple application under automation by way of the Automunge platform.

3 One More Thing

Oh and one more thing. Please note that similar feature importance evaluation for impact to predictive accuracy can also be performed for categoric feature sets by simply applying a one-hot encoding transform ('text') and generating a feature importance report in which the sorted metric2 result will similarly demonstrate importance of different categoric entries toward predictive accuracy. Here's an example from the same data set in which we assign the 'GarageCars' feature to one-hot encoding and perform feature importance which identifies that two-car garages are the most influential to home sale price predictive accuracy, again derived from a mean squared log error metric since home sale price predictions are a regression application.

```
{'GarageCars_2.0': 0.0002802533007064678,
 'GarageCars_3.0': 0.0002815128610716977,
 'GarageCars_1.0': 0.0005032697665753316,
 'GarageCars_0.0': 0.0005496799599492563,
 'GarageCars_4.0': 0.0005553286982610262}
```

Figure 6: 'metric2' for relative importance between features derived from same source column

Broader Impact

Feature importance is a common form of supporting explainable AI. The broader impact of this work is realized as extension of feature importance metrics from current thresholds of influence of aggregate feature sets to a more granular evaluation of segments within a feature set's distribution. We thus believe the primary impact of this work will be net positive as to support more transparent explainability for tabular data applications.

References

[1] McKinney, Wes. Data structures for statistical computing in python, *Proceedings of the 9th Python in Science Conference*, pp. 51–56, 2010

[2] Parr, Terrence & Turgutlu, Kerem & Csiszar, Christopher & Howard, Jeremy. Beware Default Random Forest Importances, *Explained.ai (blog)* March 26, 2018 `https://explained.ai/rf-importance/`

[3] Taleb, Nassim & Canetti, Elie & Kinda, Tidiane & Loukoianova, Elena & Schmieder, Christian (2012) A New Heuristic Measure of Fragility and Tail Risks: Application to Stress Testing, *IMF Working Paper*, August 2012