sqoba

Handling High Cardinality Categoricals via Target Encodings

Anastasios Zouzias (joint work with Immanuel Bayer palaimon.io)

AMLD 2019
Lausanne, January 2019

Goal of Presentation: Target Encodings

Get you familiar with a feature engineering* technique for categoricals

Motivation (personal)

- Target Encoding appears in several submissions / discussions / kernels in machine learning competitions (Kaggle)
- Improve model performance for categorical features
- Easy to describe / implement but tricky to apply properly (regularisation)
- Gradient Boosted Trees implementation (CatBoost by Yandex) is based on mean target encodings

kaggle™

CatBoost

Finance Track: Weight of Evidence encoding

J. R. Statist. Soc. A (1997)
160, *Part* 3, *pp.* 523–541

Statistical Classification Methods in Consumer Credit Scoring: a Review

By D. J. HAND† and W. E. HENLEY

The Open University, Milton Keynes, UK *Rothschilds, London, UK*

[Received March 1995. Final revision October 1996]

SUMMARY

Credit scoring is the term used to describe formal statistical methods used for classifying applicants for credit into 'good' and 'bad' risk classes. Such methods have become increasingly important with the dramatic growth in consumer credit in recent years. A wide range of statistical methods has been applied, though the literature available to the public is limited for reasons of commercial confidentiality. Particular problems arising in the credit scoring context are examined and the statistical methods which have been applied are reviewed.

Keywords: CLASSIFICATION; CONSUMER LOANS; CREDIT CONTROL; CREDIT SCORING; DISCRIMINANT ANALYSIS; FINANCE; REJECT INFERENCE; RISK ASSESSMENT

Review Paper on Consumer Credit Scoring

As can be seen from Table 1, the data are often categorical (typically, continuous variables are categorized), usually with only a few categories, though some, such as postcode, can have many categories. Although the range of statistical techniques for handling multivariate categorical data has widened dramatically in the last 15 years, almost all commercial credit scoring systems use dummy variables (see, for example, Crook *et al.* (1992)). However, the alternative approach of coding categorical variables into numeric form and using continuous data models is becoming more common. For example, one strategy is to use logarithms of likelihood ratios, in this context called *weights of evidence*: the *j*th attribute of the *i*th characteristic is scored as

$$w_{ij} = \ln(p_{ij}/q_{ij}),$$

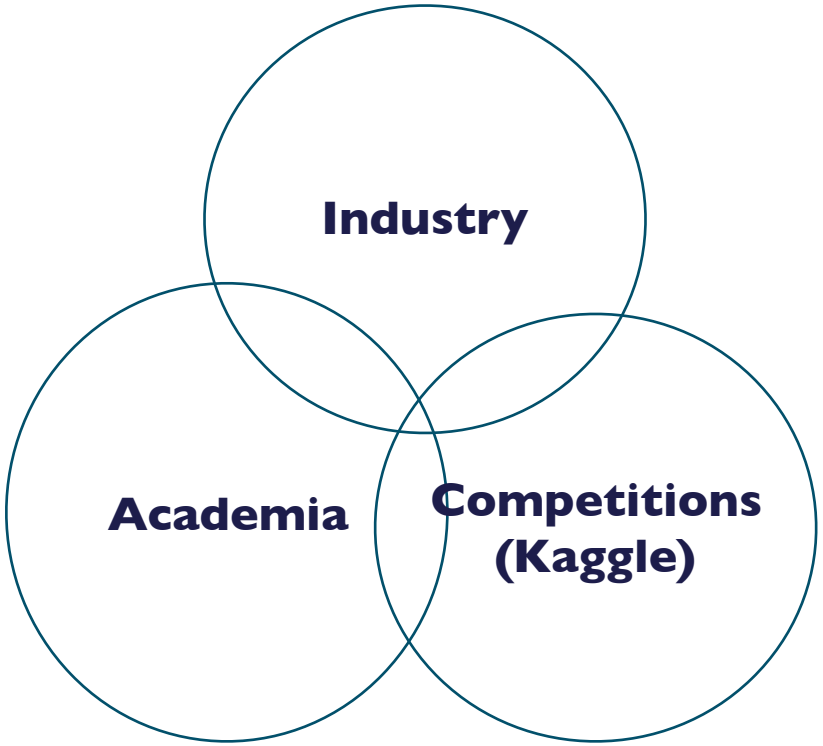
where p_{ij} is the number of good risks in attribute j of characteristic i divided by the total number of good risks (who respond to characteristic i) and q_{ij} is the number of bad risks in attribute j of characteristic i divided by the total number of bad risks (who respond to characteristic i). An alternative approach is to use optimal scaling (Gifi, 1990).

Weight of Evidence encoding of Categoricals

Ăs in many classification problems, there are complementary pressures on the number of variables to be included. Since the data sets are generally large, overfitting problems may not occur. Thus one might seek to use as many variables as possible. However, there are practical limitations: as mentioned above, too many questions or too lengthy a vetting procedure will deter applicants, who will go elsewhere. A

Big data implies no overfitting; no regularisation

Machine Learning: point of views



Gradient Tree Boosting
(xgboost, lightgbm, catboost)

Neural Networks

- **Competitions**: model performance (**this talk**)
- **Industry**: model performance, scalability, interpretability, robustness/simplicity
- **Academia**: Model Performance / Novelty

Handling Categoricals in Machine Learning

Working Example (Postal Code)

age	gender	canton	ZIP	target
34	Male	Zurich	**8043**	1
27	Female	Bern	**3001**	0
54	Male	Luzern	**6000**	1
65	Male	Vaud	**1010**	1
48	N/A	Schwyz	**6413**	0

Cases:

- Models handle categorical **correctly**
- Models handling categorical **incorrectly** (factors in R)
- Models **cannot** not handle categoricals (numerics)

One-Hot Encoding (OHE)

Hierarchical Bayesian Modelling

Numeric Encoding

Entity Embedding (NN)

Target encodings

High Cardinality Categoricals: Label/Numeric Encoding

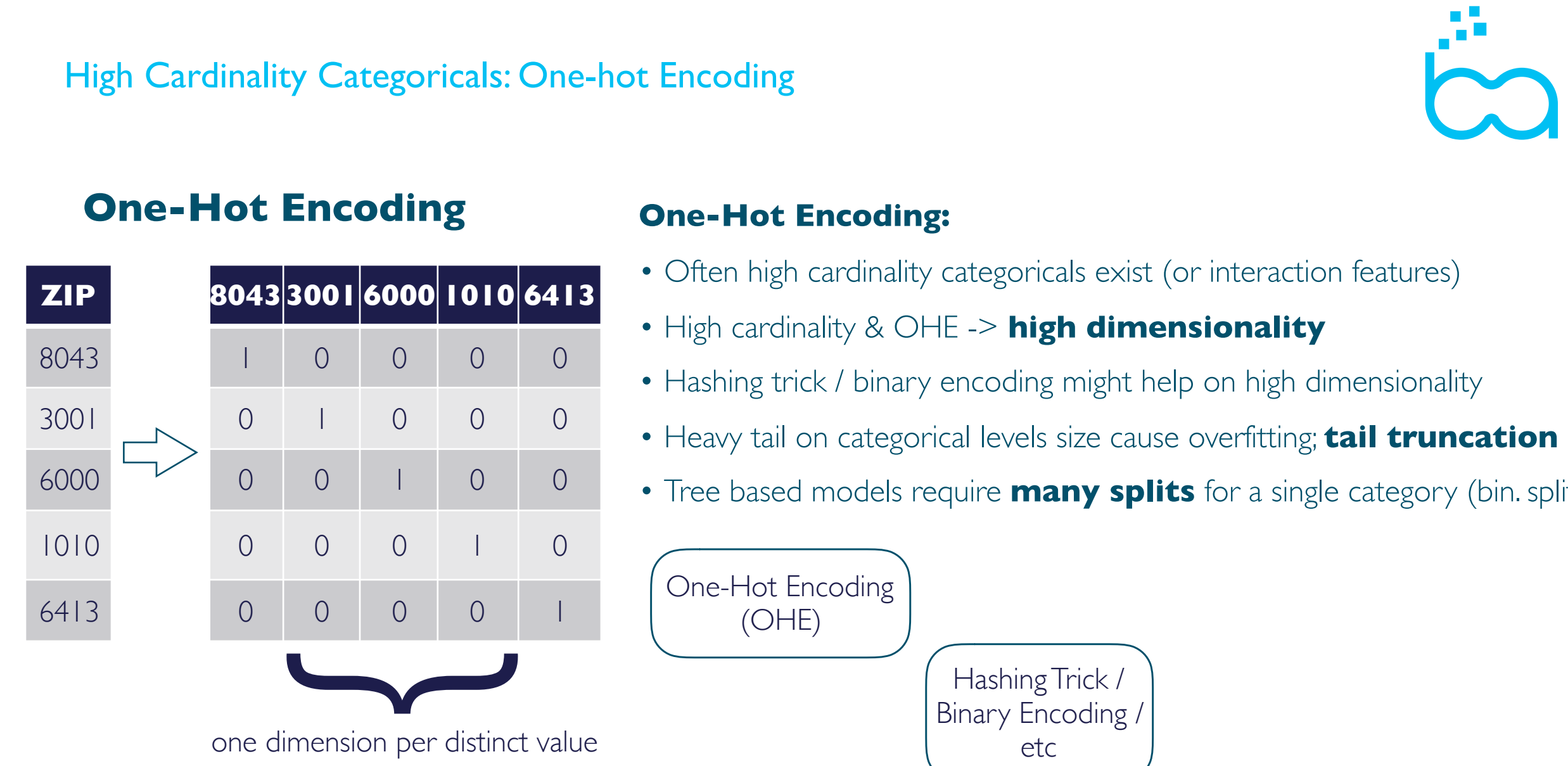
Label Encoding

ZIP	ZIP_ENC
80xx	0
30xx	*1*
80xx	0
30xx	*1*
30xx	*1*
34xx	2

Label/Numeric Encoding

- Replace each categorical level with an integer (**implicit ordering**)
- Enforces an arbitrary ordering on categorical
- Many categoricals require more degrees of freedom (colours, industries, postal codes, car brands, etc)

High Cardinality Categoricals: One-hot Encoding



One-Hot Encoding:

- Often high cardinality categoricals exist (or interaction features)
- High cardinality & OHE -> **high dimensionality**
- Hashing trick / binary encoding might help on high dimensionality
- Heavy tail on categorical levels size cause overfitting; **tail truncation**
- Tree based models require **many splits** for a single category (bin. splits)

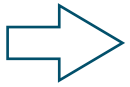
One-Hot Encoding (OHE)

Hashing Trick / Binary Encoding / etc

High Cardinality Categoricals: Mean Target Encoding (no regularisation)

Mean Target Encoding

ZIP	target
80xx	1
30xx	0
80xx	0
30xx	1
30xx	0
34xx	1



ZIP_TE
1/2
1/3
1/2
1/3
1/3
1

Target encoding could extract any property of the group (min, max, std)

Mean Target Encoding (MTE)

- Extract mean target value of group / categorical level (unseen categorical values, use total mean value)
- Danger! Danger! Use of target value; **target leakage / overfitting**
- [+] Require a single dimension/feature per categorical
- [+] Easy to implement
- [-] Must be carefully regularised (next slides)

* Can be extended to regression, multi-class classification

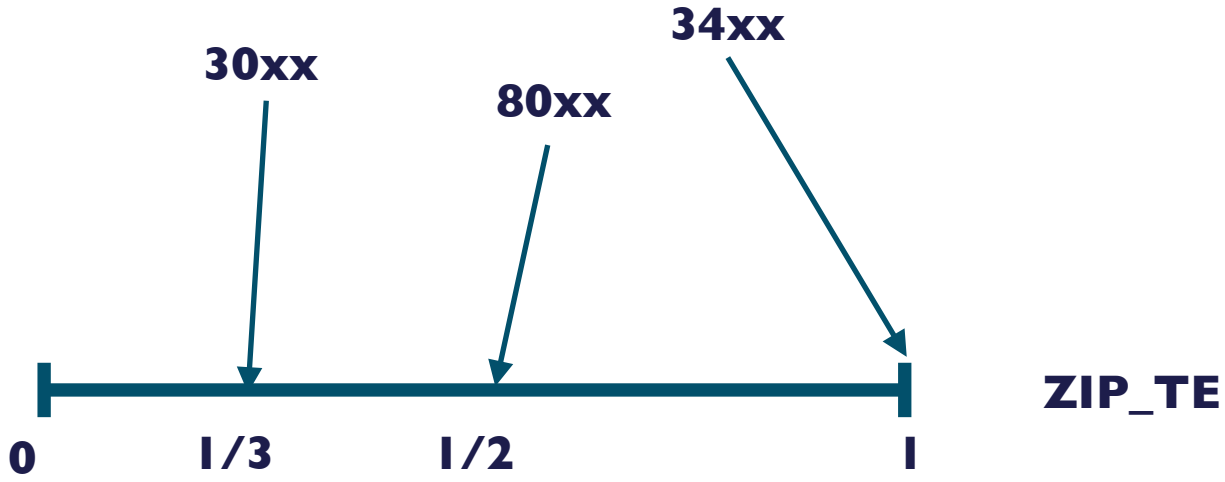
Why Mean Target Encodings work well with GBTs?

Mean Target Encoding

ZIP	target
80xx	1
30xx	0
80xx	0
30xx	1
30xx	0
34xx	1

⇨

ZIP_TE
1/2
1/3
1/2
1/3
1/3
1



Intuition

- Mean TE separates levels of categorical between positive and negative classes.
- Tree based models benefit from above ordering to find the optimal split (cross-entropy, Gini index, MSS) [Fisher58, Breiman84, Ripley96]

An overfitting example in R

```
nLev <- 500
n <- 3000
d <- data.frame(xBad1=sample(paste('level', 1:nLev, sep='_'), n, replace=TRUE),
                xBad2=sample(paste('level', 1:nLev, sep='_'), n, replace=TRUE),
                xGood1=sample(paste('level', 1:nLev, sep='_'), n, replace=TRUE),
                xGood2=sample(paste('level', 1:nLev, sep='_'), n, replace=TRUE))

d$y <- (0.2 * rnorm(nrow(d)) + 0.5 * ifelse(as.numeric(d$xGood1)>nLev/2, 1, -1) +
        0.3 * ifelse(as.numeric(d$xGood2)>nLev/2, 1, -1)) > 0
```

- 2 informative/good categoricals; 2 uninformative/bad categoricals

Process:

- Target encode all categoricals on train split
- Run Logistic Regression on all target encoded numerics.
- Train AUC: **0.99**; Test AUC: **0.85** (models considers xBad1/xBad2 significant)

Regularisation of Mean Target Encodings

Smoothing / Empirical Bayes (EB) / Shrinkage of MTE (James-Stein Estimator)

- Weighted average of MTE and global target mean. (shrink more on small groups)
- EB with global mean as prior. `Double dipping` issue

Additive Laplacian Noise

- Additive noise on each extracted value (differential privacy / stability => generalisation)

K-Fold CV / Stacked Generalisation

- Compute MTE on each fold by using all other folds
- Connection with Stacked Generalisation [Breiman96]
- Owen's LeaveOneOut Target Encoding (attributed by Kaggle competitors)

Bootstrapping / Rolling Mean

- Randomly permute rows; compute MTE on **i^{th}** row based on data up to **(i-1)** row
- Implemented in the Boosted Trees Model: **Catboost** by Yandex (with shrinkage)

Smoothing / Shrinkage / Empirical Bayes (EB)

Smoothing:

- Compute global mean target value: $\mu := 1/2$
- Compute mean for each level, i.e., $\mu_{30xx}(C) = 1/3$
- Regularised mean target is the average of global mean and group/level mean
- **Intuition:** The larger the size of the level (say 30xx) of the categorical (say ZIP), the lower the shrinkage
- **λ(N)** formalises the above intuition. **N** is the size of the level.

ZIP	target
80xx	1
30xx	0
80xx	0
30xx	1
30xx	0
34xx	1

Mean: 1/2

ZIP_TE
1/2
1/3
1/2
1/3
1/3
1

In general:

$$\mu_j^{\mathrm{EB}}(C) := \lambda(N)\mu_j(C) + (1 - \lambda(N))\mu$$

$$\mu_{30xx}^{\mathrm{EB}}(L) := \lambda(3)\frac{1}{3} + (1 - \lambda(3))\frac{1}{2}$$

Implementations of Target Encodings

R

- **vtreat** package
- h2o package: ***h2o.target_encode_**** methods

Python

- Categorical encodings: https://github.com/scikit-learn-contrib/categorical-encoding
- Regularisation is missing
- Easy to write your own implementation based on **Pandas**

Hyperparameters for Categoricals in Gradient Boosted Trees

CatBoost based on regularised Mean Target Encodings!

Function	XGBoost	CatBoost	Light GBM
Important parameters which control overfitting	1. **learning_rate or eta** – optimal values lie between 0.01-0.2 2. **max_depth** 3. **min_child_weight:** similar to min_child leaf; default is 1	1. **Learning_rate** 2. **Depth** - value can be any integer up to 16. Recommended - [1 to 10] 3. No such feature like min_child_weight 4. **l2-leaf-reg**: L2 regularization coefficient. Used for leaf value calculation (any positive integer allowed)	1. **learning_rate** 2. **max_depth**: default is 20. Important to note that tree still grows leaf-wise. Hence it is important to tune **num_leaves** (number of leaves in a tree) which should be smaller than 2^(max_depth). It is a very important parameter for LGBM 3. **min_data_in_leaf**: default=20, alias= min_data, min_child_samples
Parameters for categorical values	Not Available	1. **cat_features**: It denotes the index of categorical features 2. **one_hot_max_size**: Use one-hot encoding for all features with number of different values less than or equal to the given parameter value (max – 255)	1. **categorical_feature:** specify the categorical features we want to use for training our model
Parameters for controlling speed	1. **colsample_bytree**: subsample ratio of columns 2. **subsample**: subsample ratio of the training instance 3. **n_estimators**: maximum number of decision trees; high value can lead to overfitting	1. **rsm:** Random subspace method. The percentage of features to use at each split selection 2. No such parameter to subset data 3. **iterations**: maximum number of trees that can be built; high value can lead to overfitting	1. **feature_fraction:** fraction of features to be taken for each iteration 2. **bagging_fraction:** data to be used for each iteration and is generally used to speed up the training and avoid overfitting 3. **num_iterations:** number of boosting iterations to be performed; default=100

LightGBM splits categorical by using **gradient vector** & optimal splits [Fisher58] (2-means on 1 dim.)

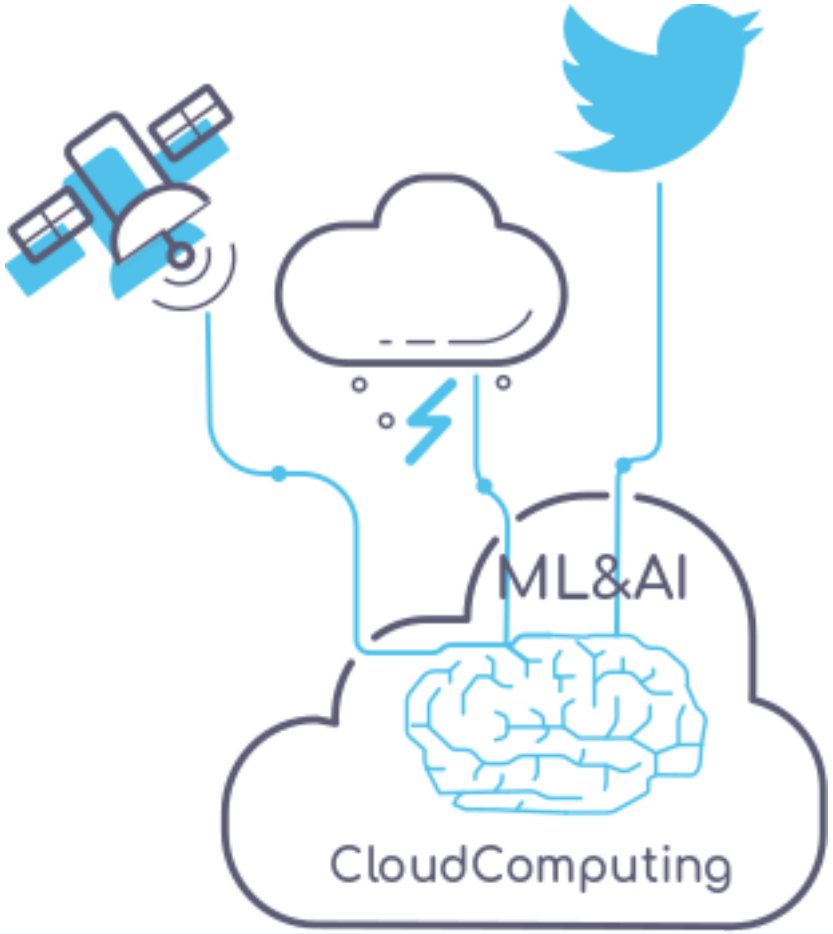
Extracted from: https://towardsdatascience.com/catboost-vs-light-gbm-vs-xgboost-5f93620723db

Conclusions & Discussion

- (Mean) Target encodings appear where model performance is the main focus
- Top performing models in Kaggle: feature engineering & GBTs
- Regularisation of Target Encodings cannot be avoided & tricky to be done right
- Target Encodings have many connections to fundamental topics in Statistical Learning (Empirical Bayes, Stability, Model Stacking, Bootstrapping)
- Rise of GBT models that handle categoricals natively…

palaimon.io

Konstanz University Spin-off



Questions

Thank you

11.2 Categorical Feature Support

- LightGBM offers good accuracy with integer-encoded categorical features. LightGBM applies Fisher (1958) to find the optimal split over categories as described here. This often performs better than one-hot encoding.
- Use `categorical_feature` to specify the categorical features. Refer to the parameter `categorical_feature` in Parameters.
- Categorical features must be encoded as non-negative integers (`int`) less than `Int32.MaxValue` (2147483647). It is best to use a contiguous range of integers started from zero.
- Use `min_data_per_group`, `cat_smooth` to deal with over-fitting (when `#data` is small or `#category` is large).
- For a categorical feature with high cardinality (`#category` is large), it often works best to treat the feature as numeric, either by simply ignoring the categorical interpretation of the integers or by embedding the categories in a low-dimensional numeric space.

```
# Modelling
m1 <- glm(y ~ xBad1_catB + xBad2_catB + xGood1_catB + xGood2_catB,
          data=dTrainTreated,
          family=binomial(link='logit'))
```

```
##
## Call:
## glm(formula = y ~ xBad1_catB + xBad2_catB + xGood1_catB + xGood2_catB,
##     family = binomial(link = "logit"), data = dTrainTreated)
##
## Deviance Residuals:
##      Min        1Q    Median        3Q       Max
## -2.65876  -0.00186   0.00000   0.00212   2.72722
##
## Coefficients:
##             Estimate Std. Error z value Pr(>|z|)
## (Intercept) -0.09887    0.12843  -0.770    0.441
## xBad1_catB   0.70055    0.15909   4.404 1.06e-05 ***
## xBad2_catB   0.98752    0.17038   5.796 6.80e-09 ***
## xGood1_catB  1.16785    0.09741  11.989  < 2e-16 ***
## xGood2_catB  1.21988    0.16537   7.377 1.62e-13 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```