

Practical Data Science

Bauke Brenninkmeijer

Goal

Show some tips, tricks,
and common pitfalls.

whois: Bauke Brenninkmeijer

- MSc in CS and Data Science @Nijmegen
- Data Scientist @ABNAMRO since 2019
 - 1.5 years in Data Management
 - ~1 years in Global Markets
- You might know me from the Data Science Triangle, Python Triangle or some of the other ABN communities.
-  [@baukebrenninkmeijer](https://github.com/baukebrenninkmeijer)

Now you know me

Who the f### are you

Years of experience with data science?

Please raise your hand

1. 1-2
2. 3-5
3. 6-10
4. 10+

What departments are you from?

1. DFC
2. RBB/Mortgages
3. Wealth Banking
4. Commerical banking
5. CISO
6. CADM
7. HR
8. Other

Raise your hand if you know...

1. K-nearest neighbors
2. F1-score
3. Softmax
4. self-attention

Outline

1. Business tips
2. Short tip on models
3. Ordinal/Nominal data encodings
4. Feature Importance with Trees
5. Class Imbalance
6. Order of pre-processing

Business tips

- Understand what they do.
It is essential that a data scientist understands the business case well.
- Read a book about banking (see slides repository).
I didn't initially had to, but it helps so much.
- To know how to handle edge-cases or what to prioritize: ask the stakeholder.
"Korte lijntjes" are essential

Regarding models

Most examples are with Random Forest or Decision Trees.

- Often one of the strongest models (RF/XGB/LGBM)
- Easily implementable with Scikit-learn
- Easily accessible explainability features
- Requires little data preprocessing, like scaling or OHE.

General tip: start with Random Forest

Feature engineering

Often harder than it looks

Ordinal/Nominal variables

Let's discuss categorical encoding vs. one-hot encoding (dummy)

size		small	medium	large
0	→	1	0	0
1		0	1	0
2		0	0	1

When does it matter?

Categorical vs OHE for models

- **Depends on how they optimize.**

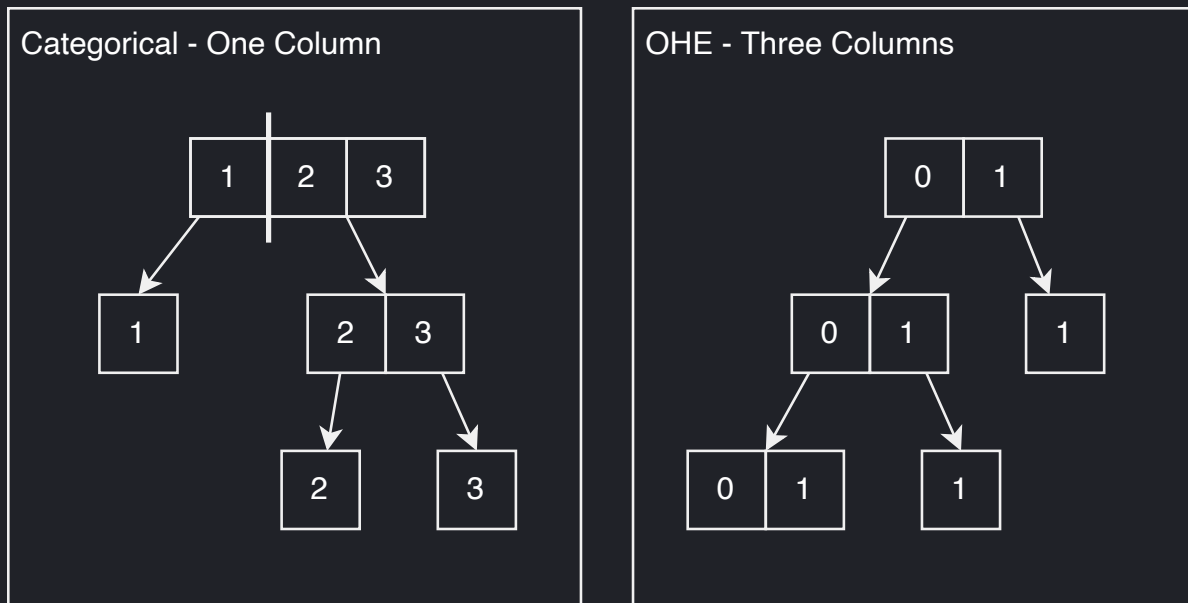
If they use distance metrics on the data, this encoding matters (e.g., K-means, LinReg, NN, SVM).

For example: S and M are closer than S and L.

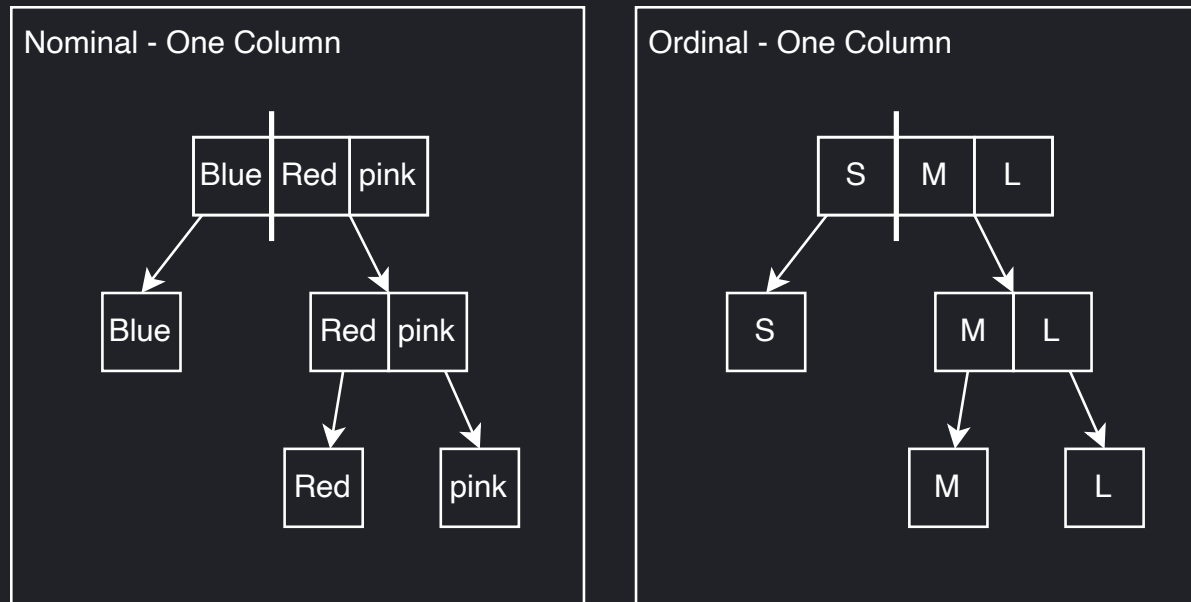
- If other measures are used, such as information gain, sometimes they can be considered equivalent.

E.g., with **trees** with unlimited depth, both encodings are essentially equivalent.

Tree Splitting Example

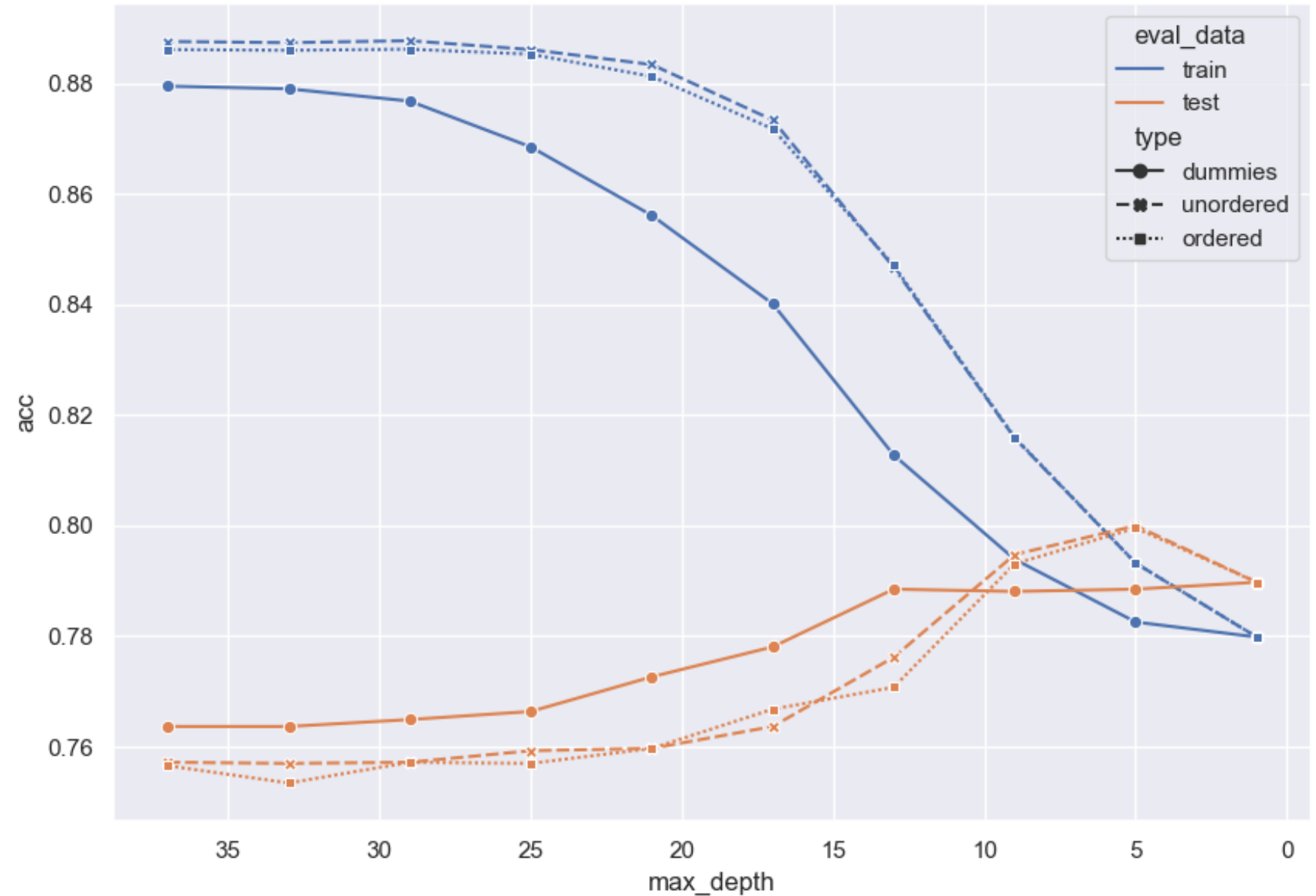


Tree Splitting Example



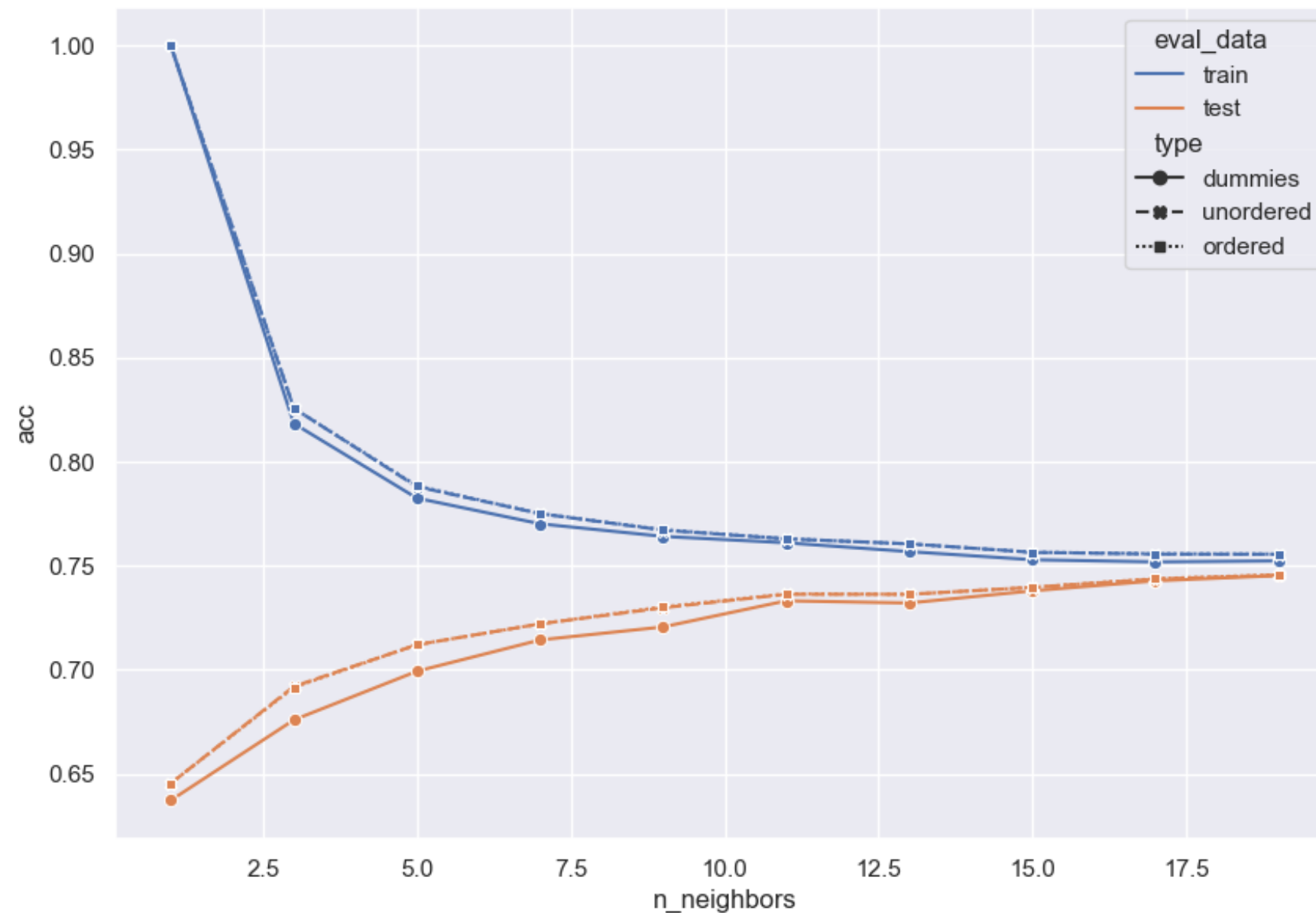
Decision Tree

- OHE worse on train than on test
- When parameters are reduced, the added value of ordered ordinal data becomes clear.



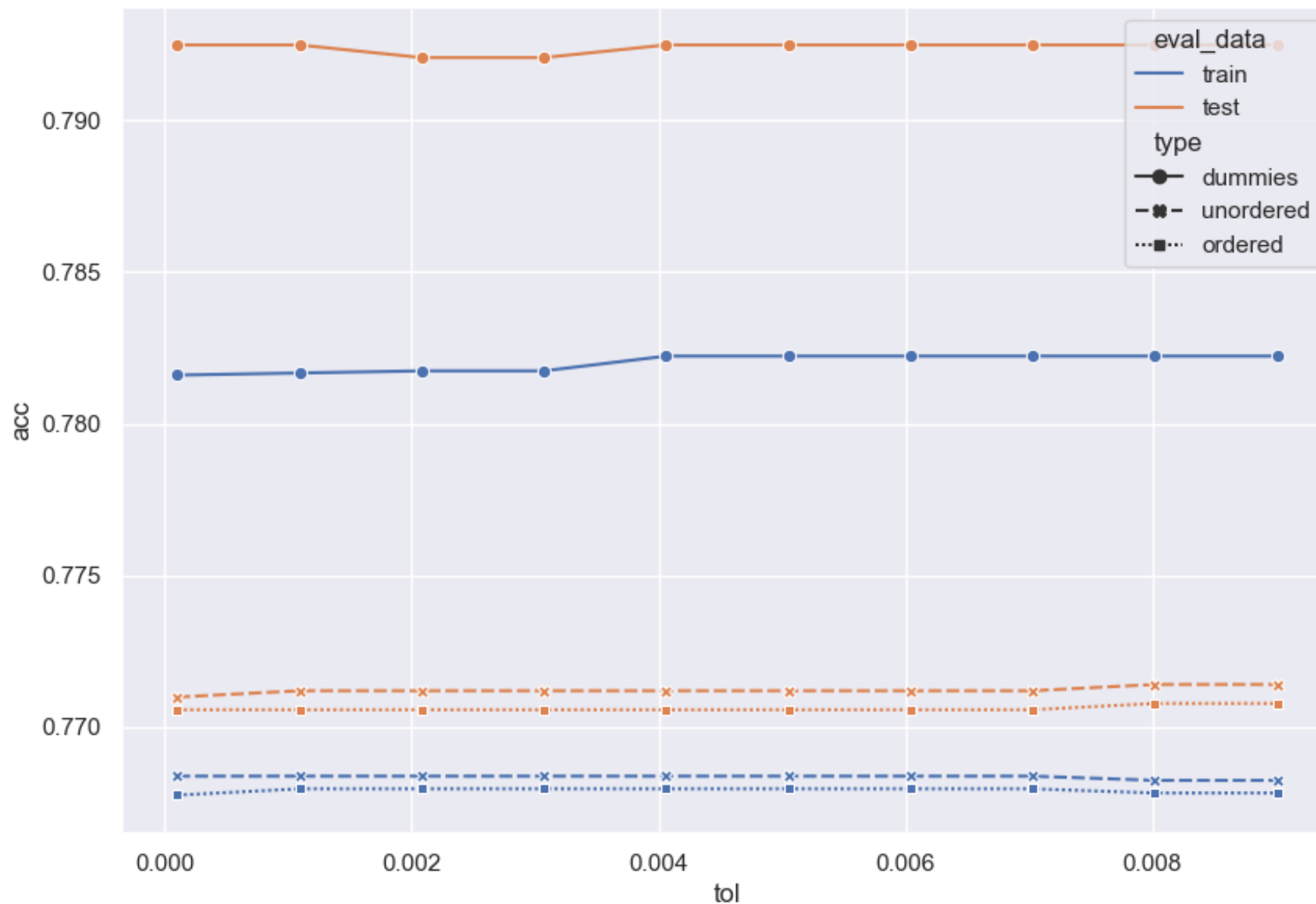
KNN

- Ordinal and Nominal are identical here
- Both outperform OHE



SVM

- With SVM, OHE is better.
- Ordered Categorical is actually the lowest
- Can be caused by an incidental increase in linear separability with random ordering.



What have we learned?

- Ordered categorical representations can help models in constrained situations, such as when reducing overfitting.
- When values of a variable have different importances, we see OHE becomes more powerful.
- When models have unlimited expressive power (i.e. are able to overfit).

Feature Importance with tree-based models

Trees have the awesome `.feature_importance_` attribute.

But

Tree-based models have a strong tendency to overestimate the importance of continuous numerical or high cardinality categorical features.

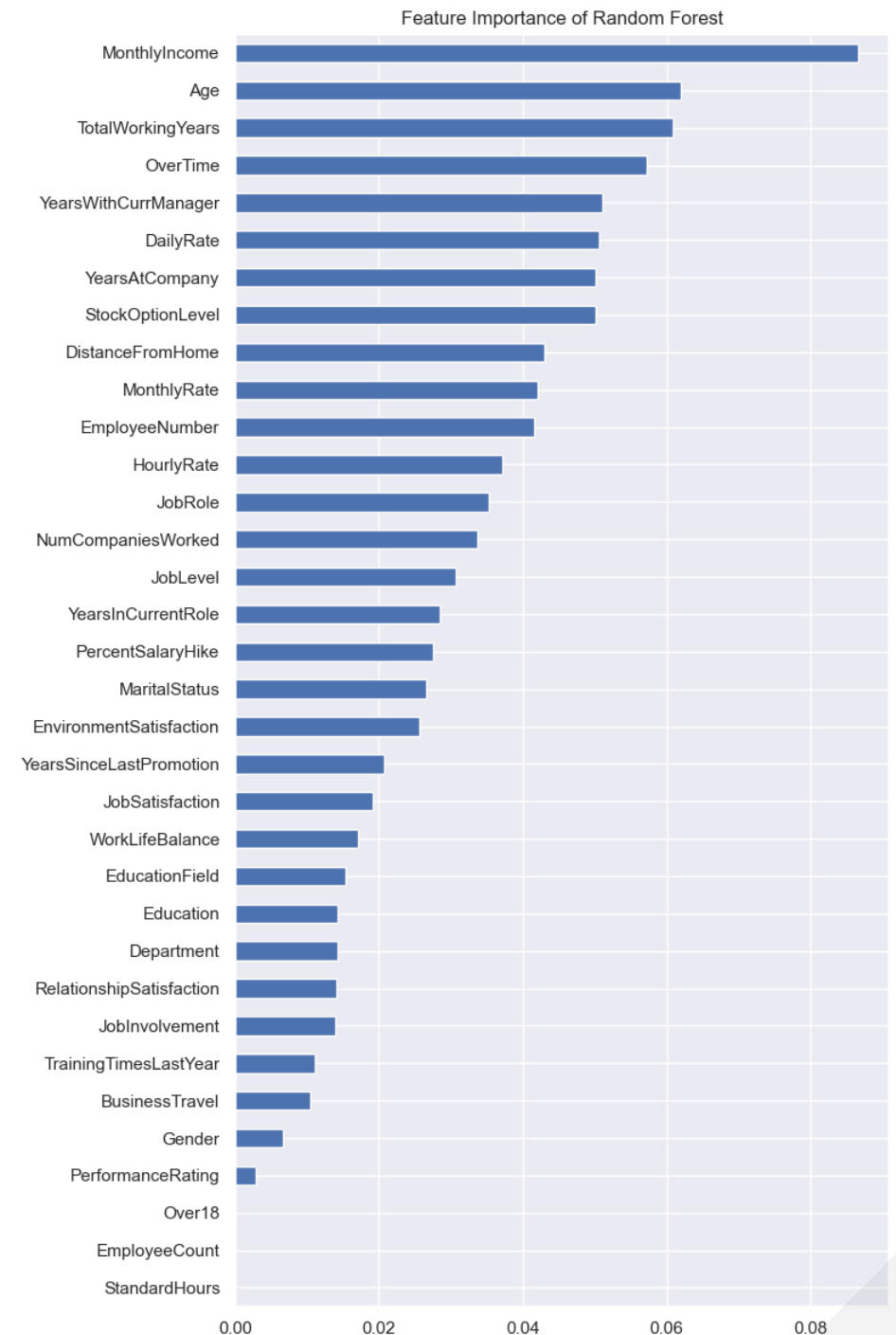
Let's see this in practice

- Binary classification on whether employees will leave the company (attrition)
- Data is mix of discrete and continuous.
- Explanation can be used by senior management to mitigate.

Feature importance

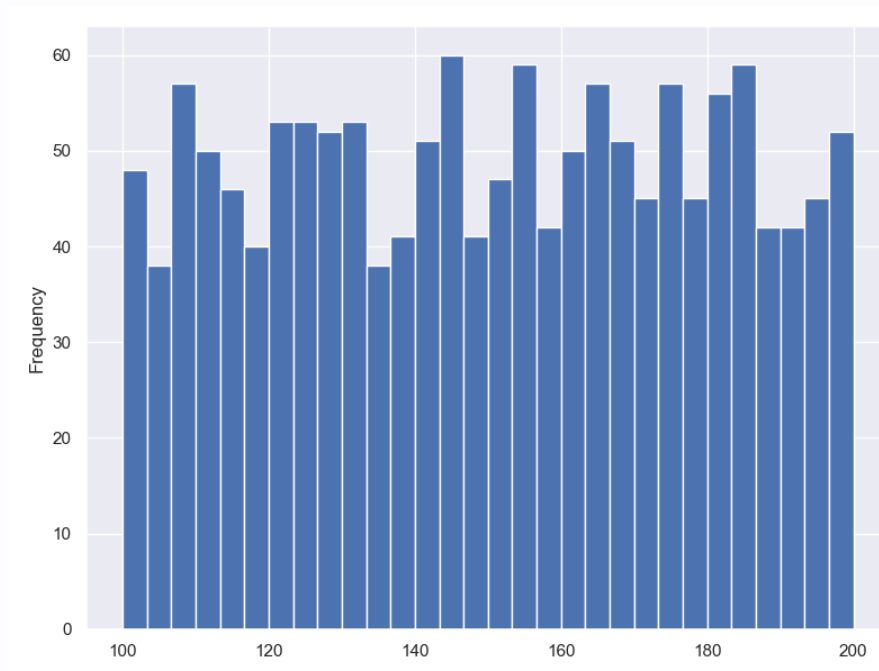
Most important:

- MonthlyIncome
- Age
- WorkingYears



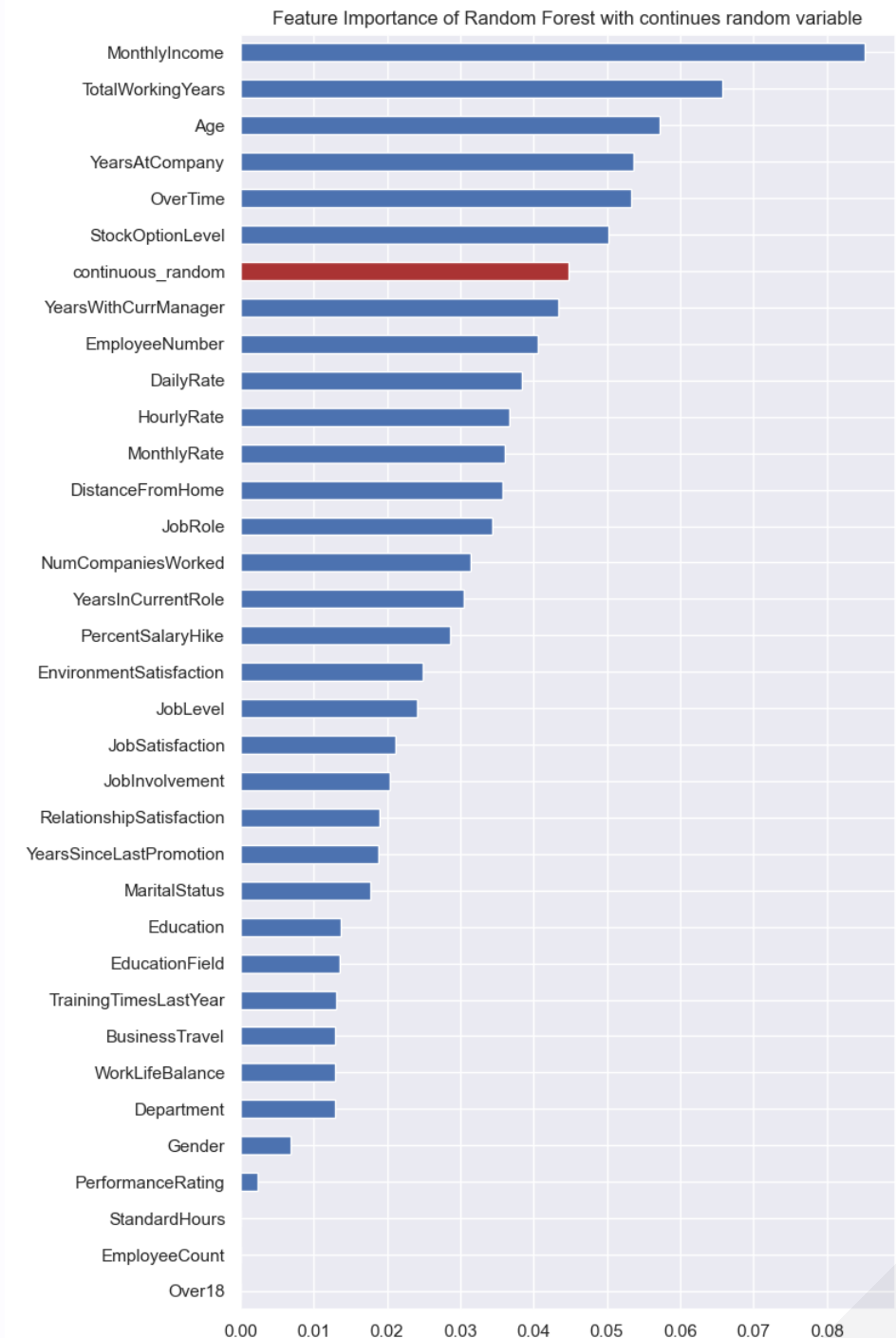
Let's mess with shit

We'll add a single random continuous variable in the range $[100, 200]$.



New feature importances

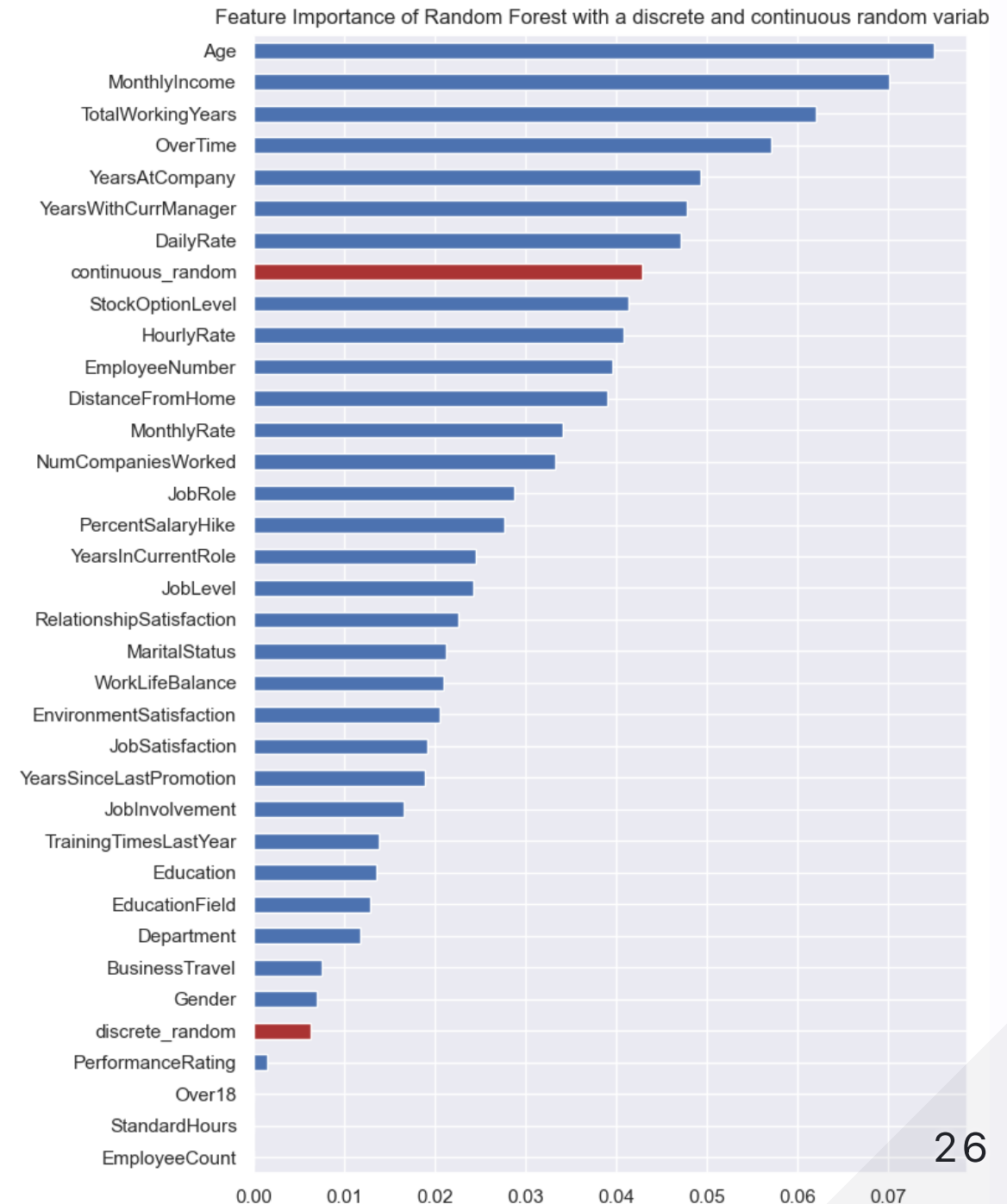
- 7th highest is random...????
- What does this mean for variables below random? No value?



Can go further

Let's also add a discrete
random variable

*Much less important
than the continuous
variable.*





Why?

- Impurity Based Importance (Gini, Entropy or MSE)
- This is biased towards high cardinality features, cause it can split more.
- Can give high importance to unresponsive variables due to overfitting.
- Feature importance is calculated purely on the training data. Does not reflect performance on unseen data.

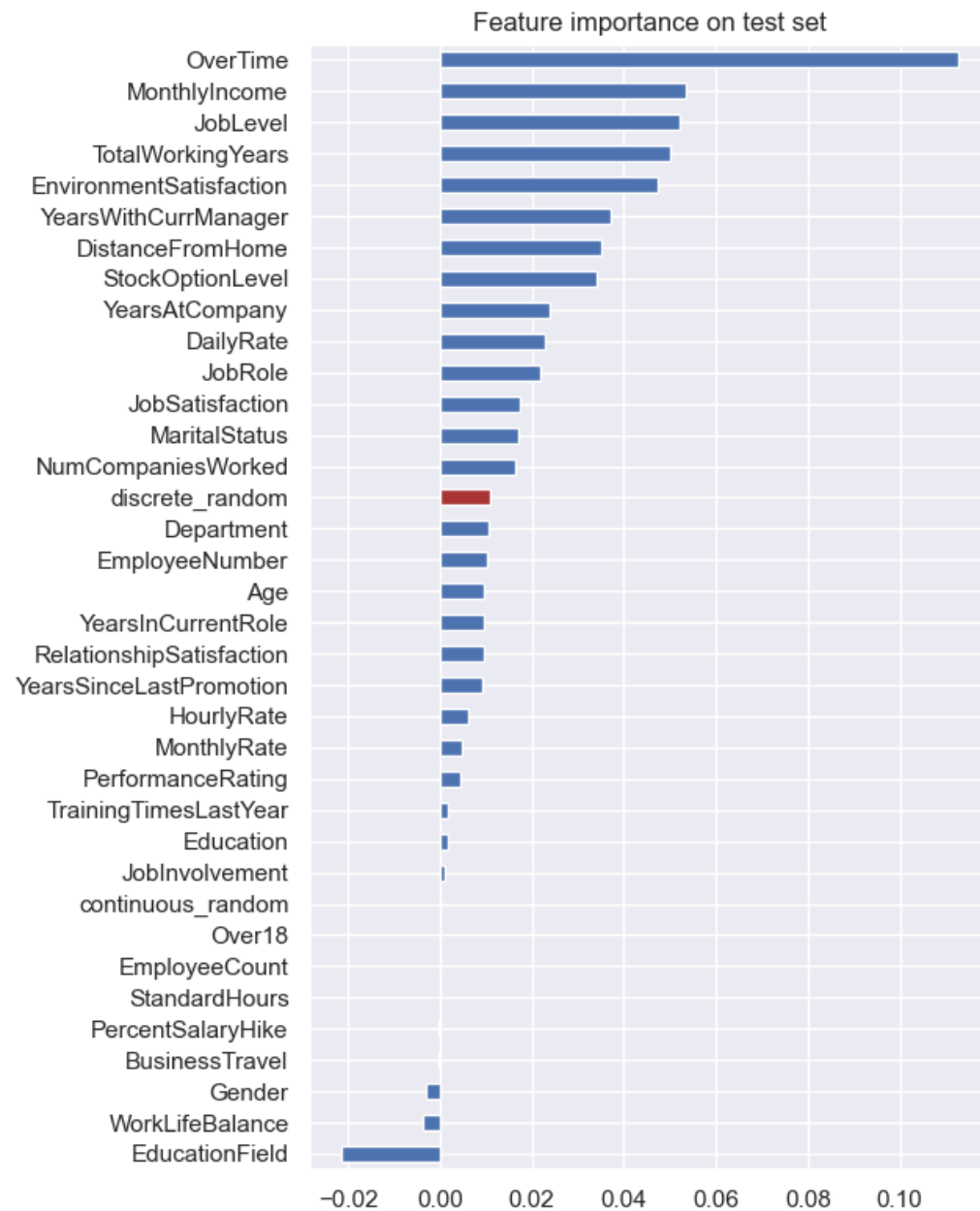
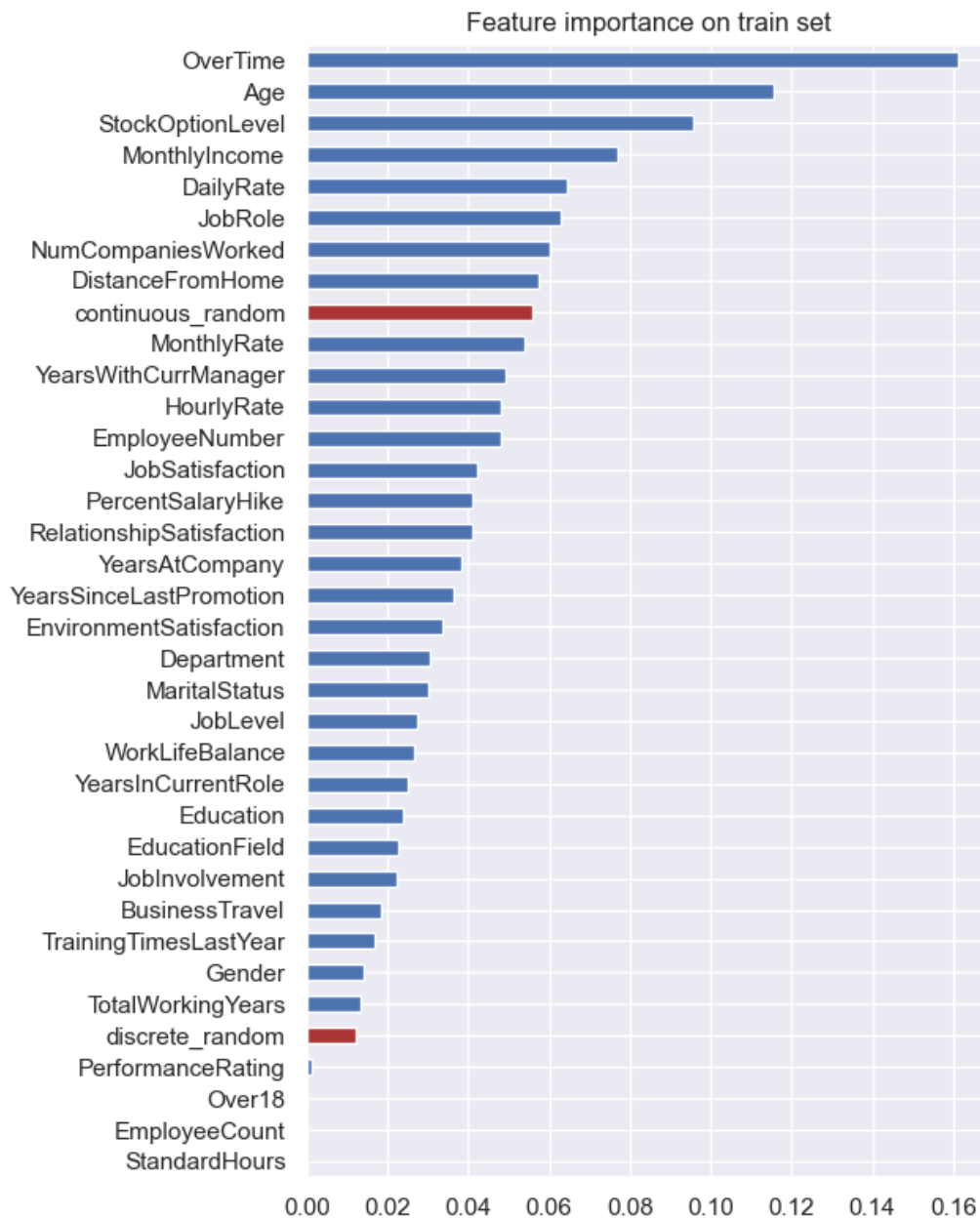
The solution: Permutation Importance

- Model agnostic way to determine importance of features for trained model.
- Can be applied on unseen data.
- Calculates impact of variable based on how much the model performance decreases

Algo

1. Calculate baseline score (`.score` or custom metric)
2. Shuffles a feature and recomputes score
3.  performance ==  importance

What happens with correlated columns?



Class Imbalance

Techniques typically used:

- Oversampling
- Undersampling
- Smote

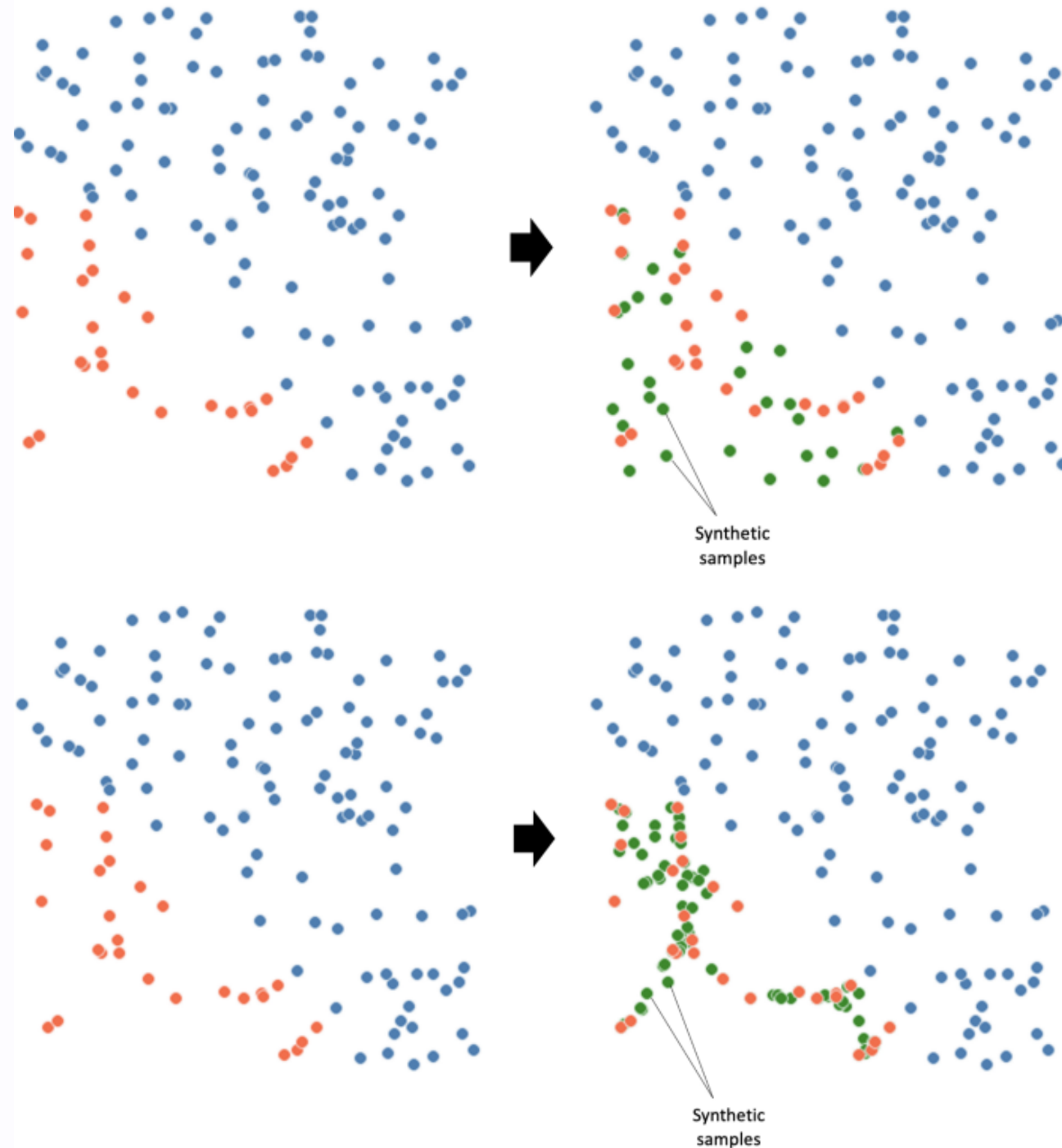
These methods increase complexity with often limited results.

But there is another intuitive way!

SMOTE

- Synthetic Minority
Oversampling
Technique
- Creates synthetic
points to increase the
number of
observations in
minority class

Class Imbalance • Practical Data Science • Bauke
Brenninkmeijer



Oversampling

- Fix class imbalance by looking more at the minority class
- I.e., duplicate minority data points



Undersampling

- Only look at the same number of data points in the majority class, as there are in the minority class.
- I.e., drop part of the data



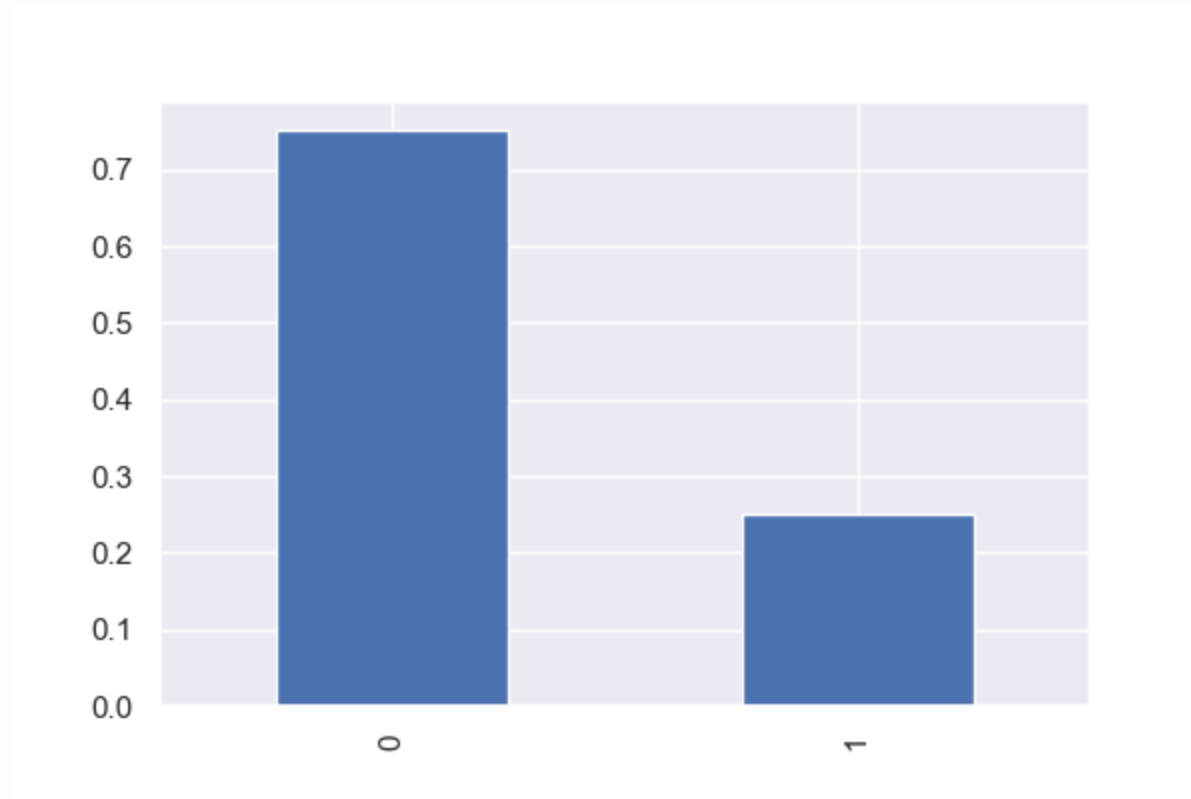
Class Weights

- Default part of sklearn
- Allows one to penalise minority errors more
- Assign weights to classes such that the weighted sums is equal.

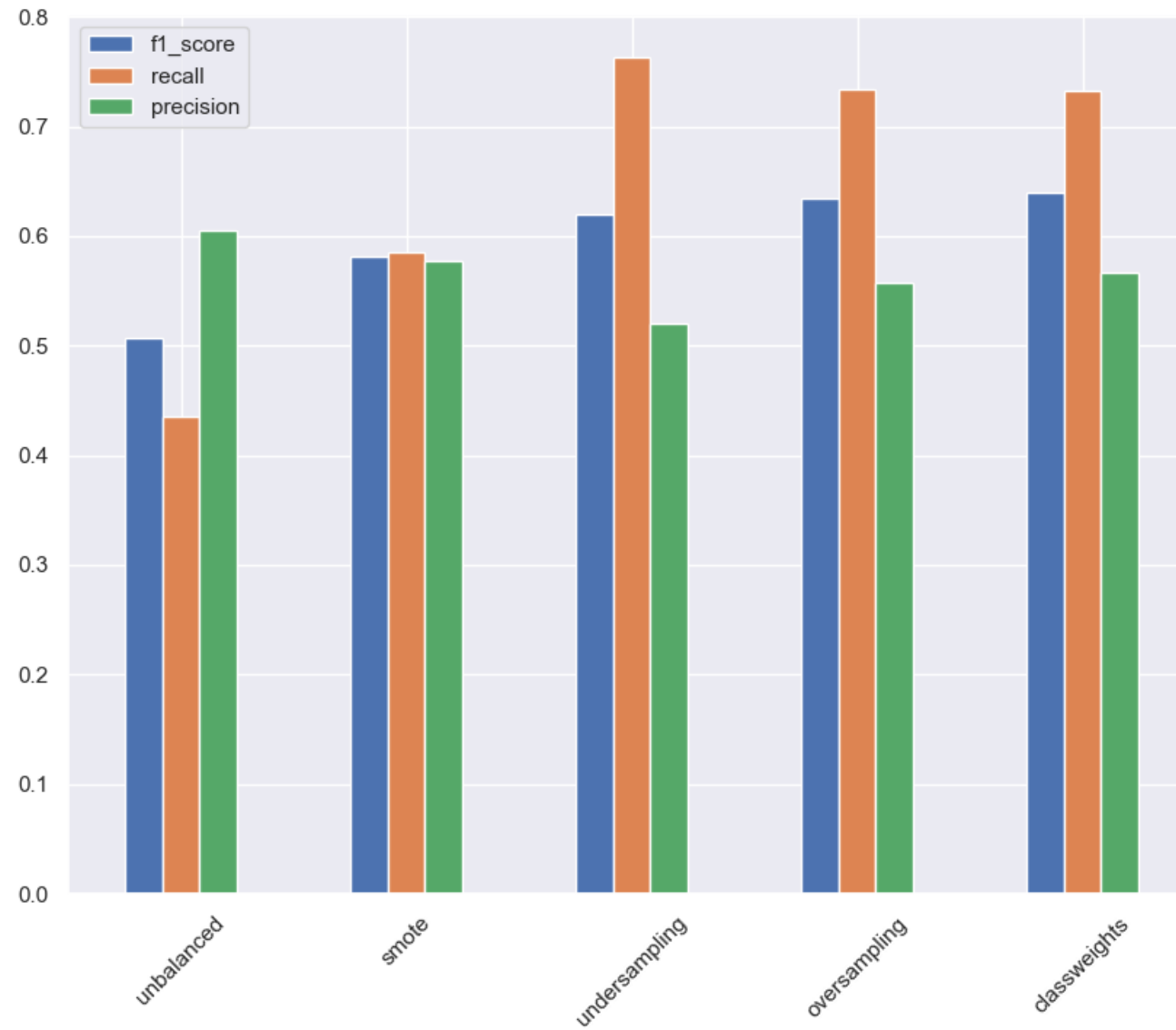
$$n_{minority} \times w_{minority} = n_{majority} \times w_{majority}$$

We'll look at a dataset of employees interested in switching jobs

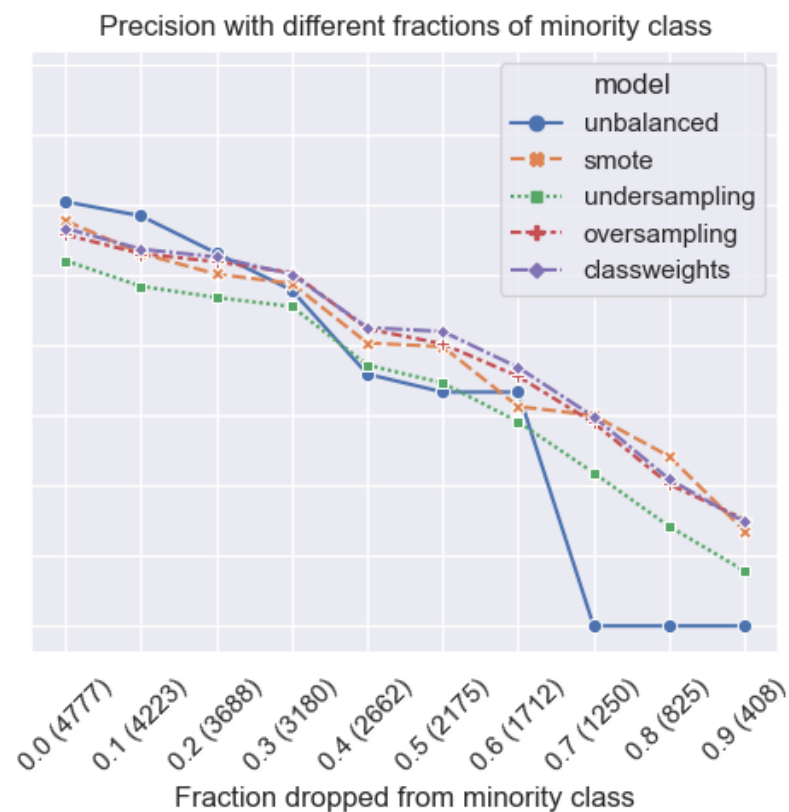
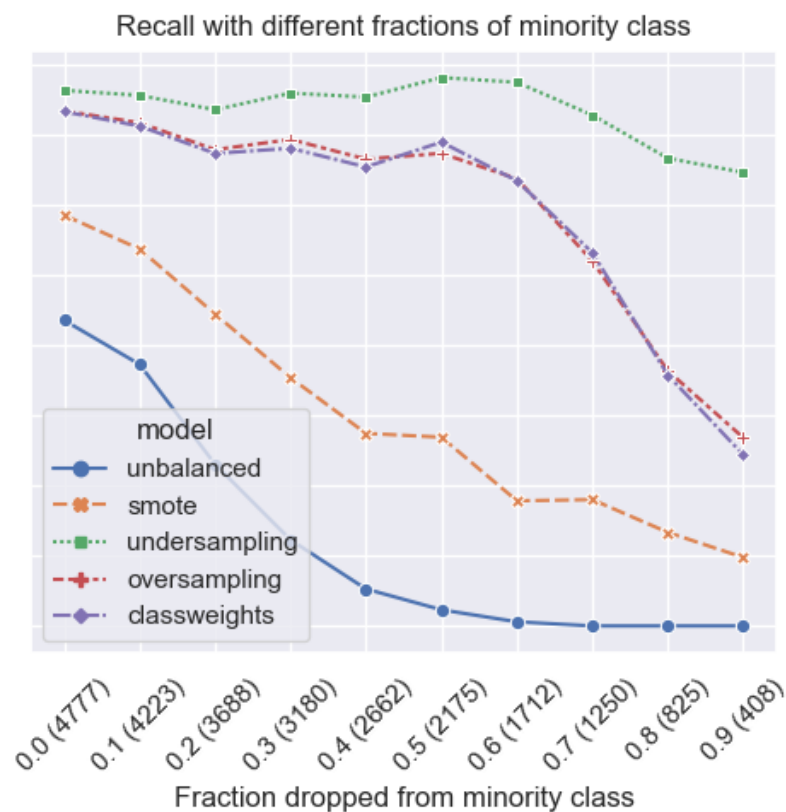
Target is imbalanced:



We train a classifier on SMOTE, undersampling, oversampling and class weights and compare the results.



What happens if we make the data more imbalanced?



Take away

- Class weights are a useful addition to the imbalanced learning toolbox.
- Very similar to oversampling in performance
- Try several imbalanced learning solutions and see what works!

Order of pre-processing

Order of pre-processing

- **Never** do distribution based transformations **before** splitting train/test
- Splitting should (almost) always be the first step, to prevent information leakage

```
# includes metrics from future test set
df = StandardScaler().fit_transform(df)

x_train, x_test, y_train, y_test = train_test_split(
    df.drop('target', axis=1),
    df.target,
)
```

What we should do:

```
x_train, x_test, y_train, y_test = train_test_split(
    df.drop('target', axis=1),
    df.target,
)

scaler = StandardScaler()
x_train = scaler.fit_transform(x_train)
x_test = scaler.transform(x_test)
```

Goes for

- StandardScaler
- Min-max scaler
- Quantile scaler
- MaxAbsScaler
- etc.

Thank you for listening.

Questions?