Practical Data Science

Bauke Brenninkmeijer

Kaggle Competition for Jr. Data Science • ABN AMRO 2022

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 now me from the Data Science Triangle, Python Triangle or some of the other ABN communities.
- @baukebrenninkmeijer

Now you know me

Who the f### are you

Years of experience with data science?

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

What departments are you from?

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

Do 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)
- 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
0
1
2

→

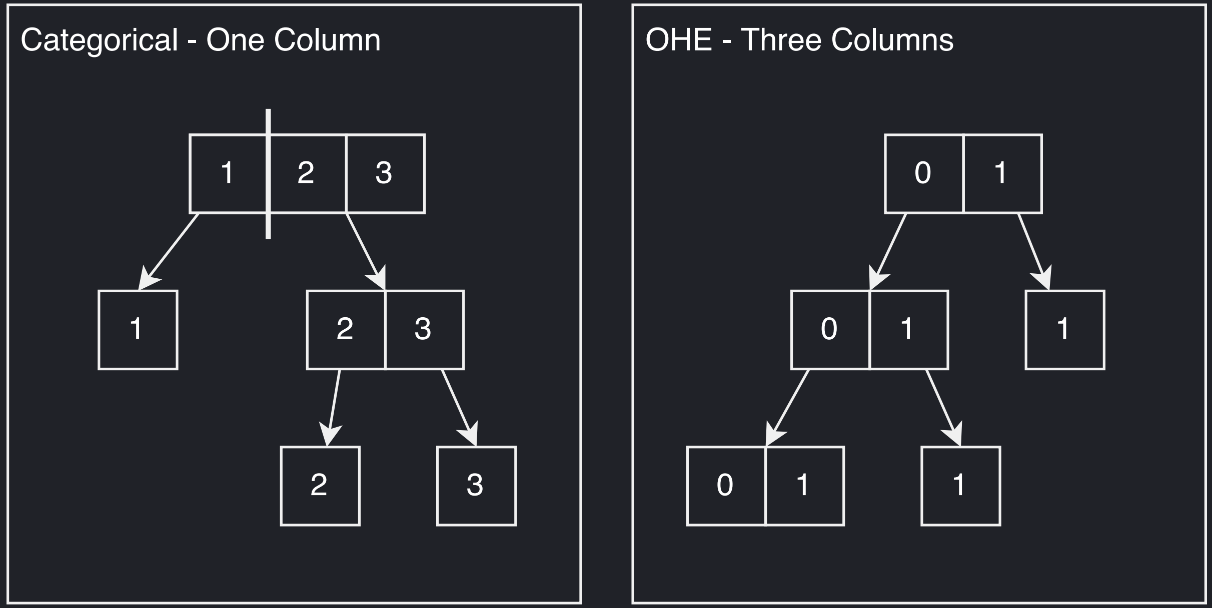
small	medium	large
1	0	0
0	1	0
0	0	1

When does it matter?

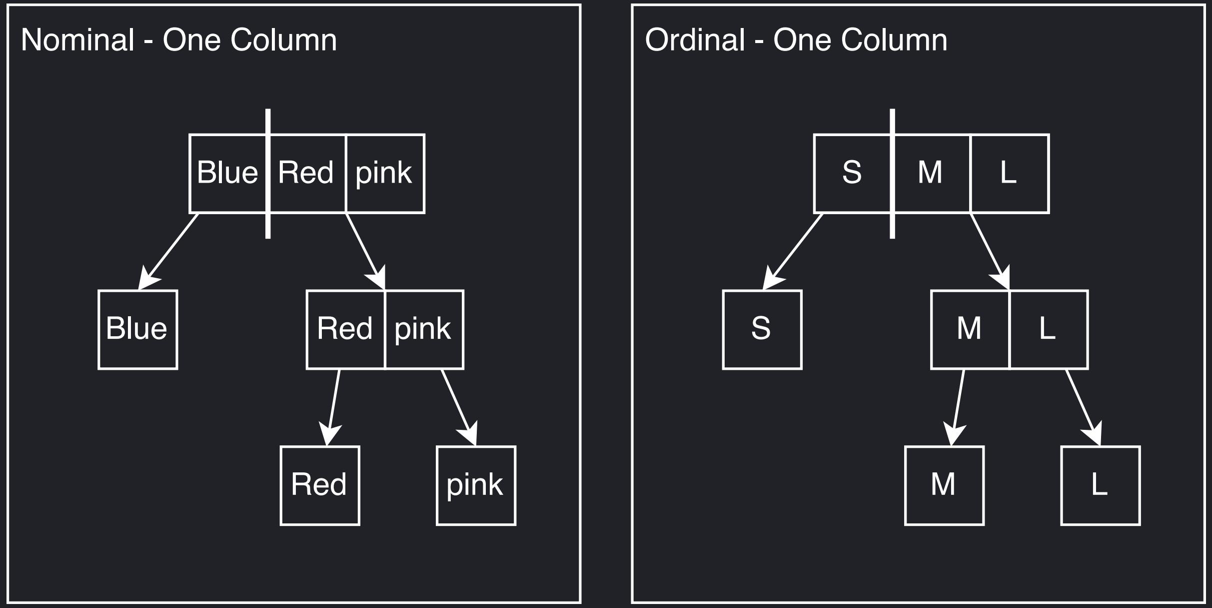
Categorical vs OHE for models

- **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

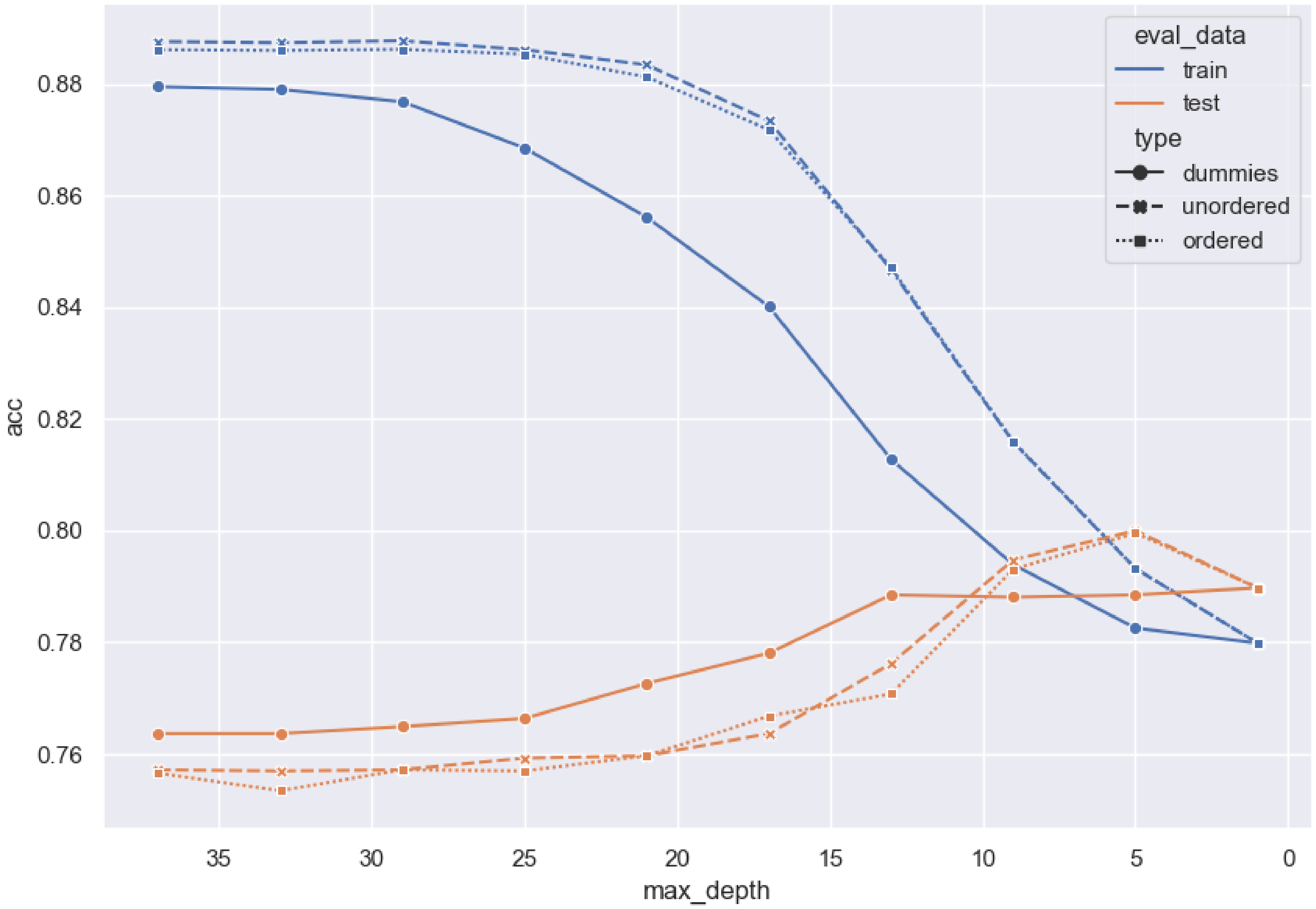


Tree Splitting Example



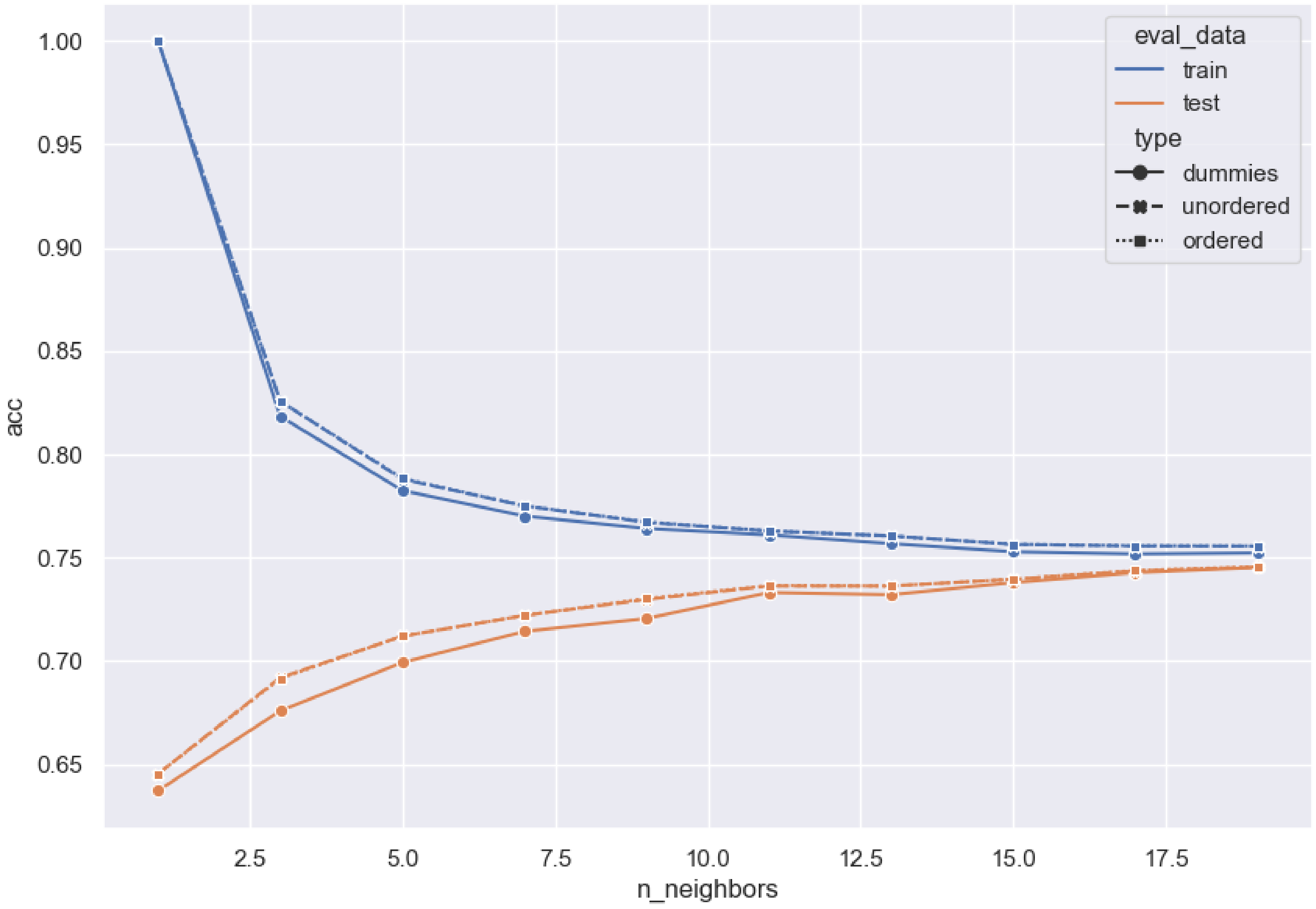
Decision Tree

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



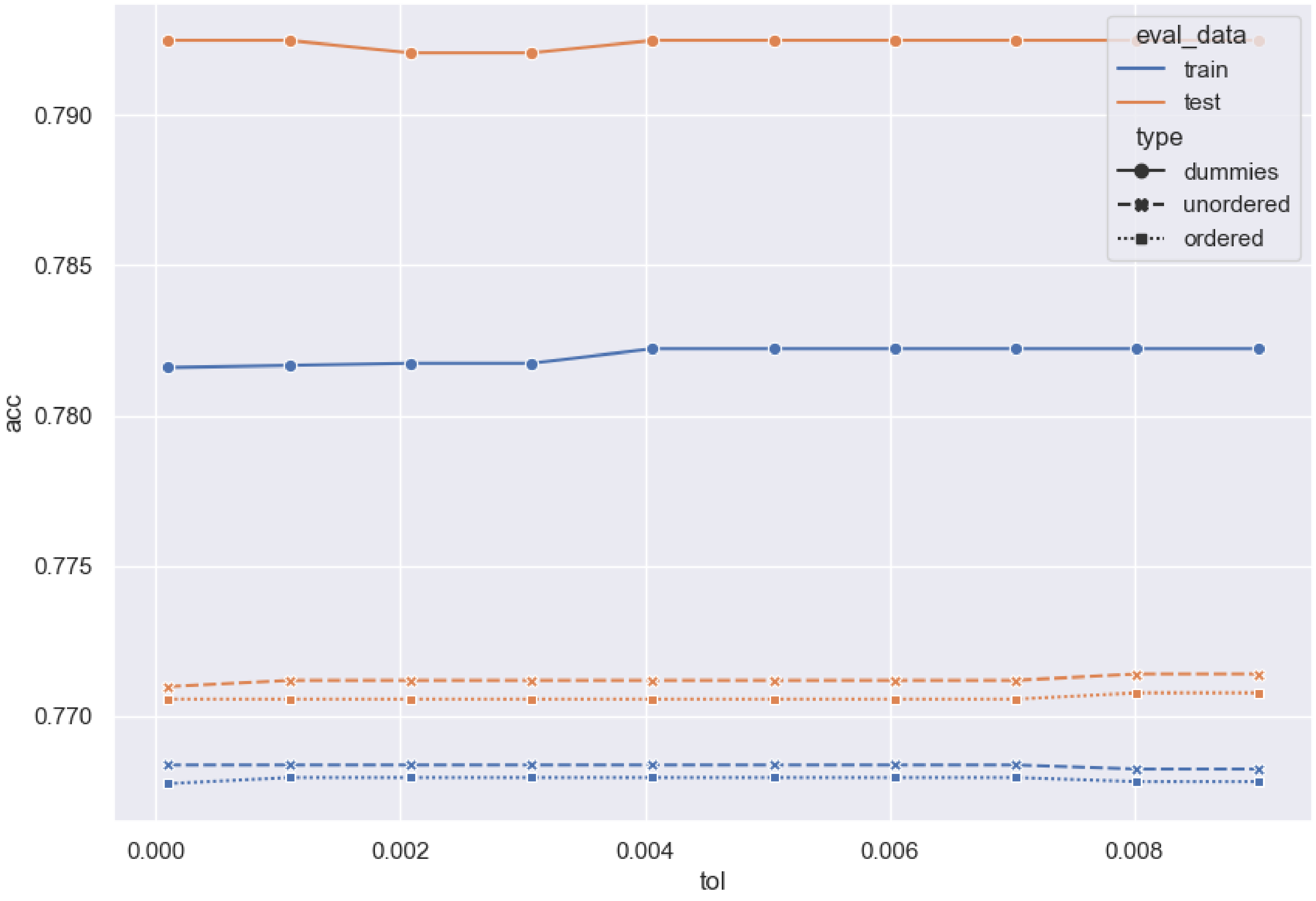
KNN

- 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 powerfull.
- 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.

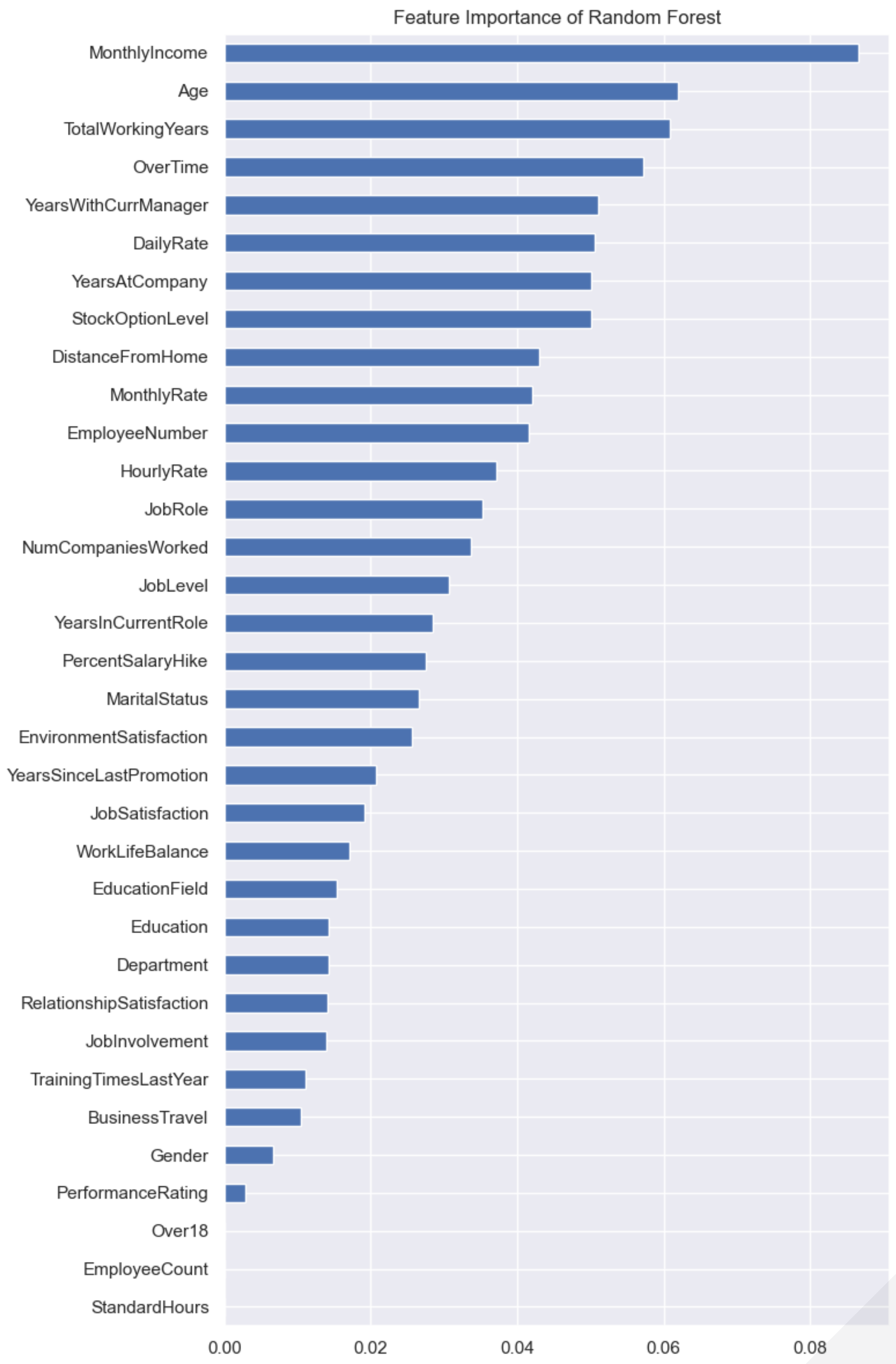
Let's see this in practice

- `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

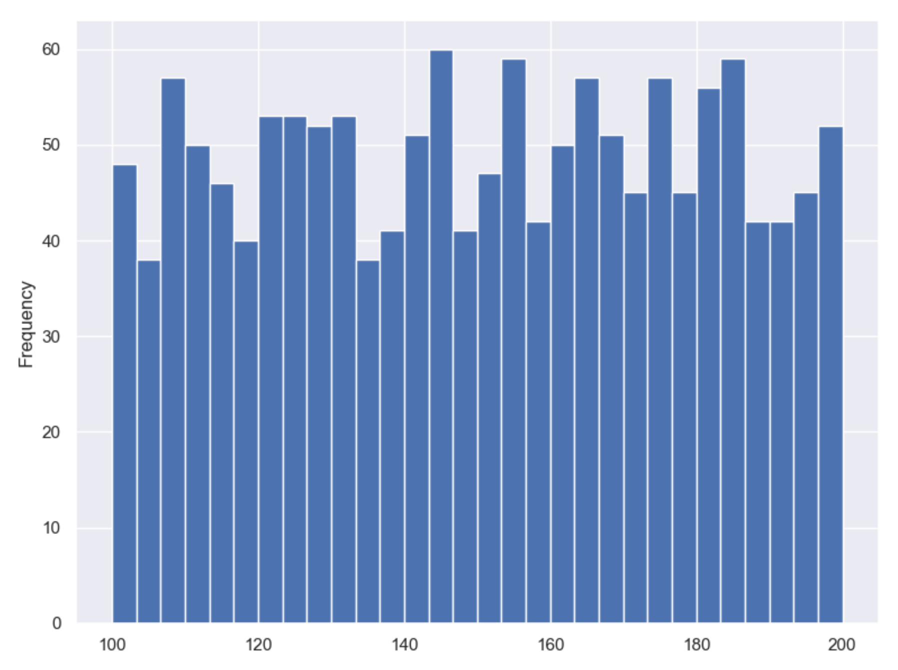
Most important:

- `MonthlyIncome`
- `Age`
- `WorkingYears`



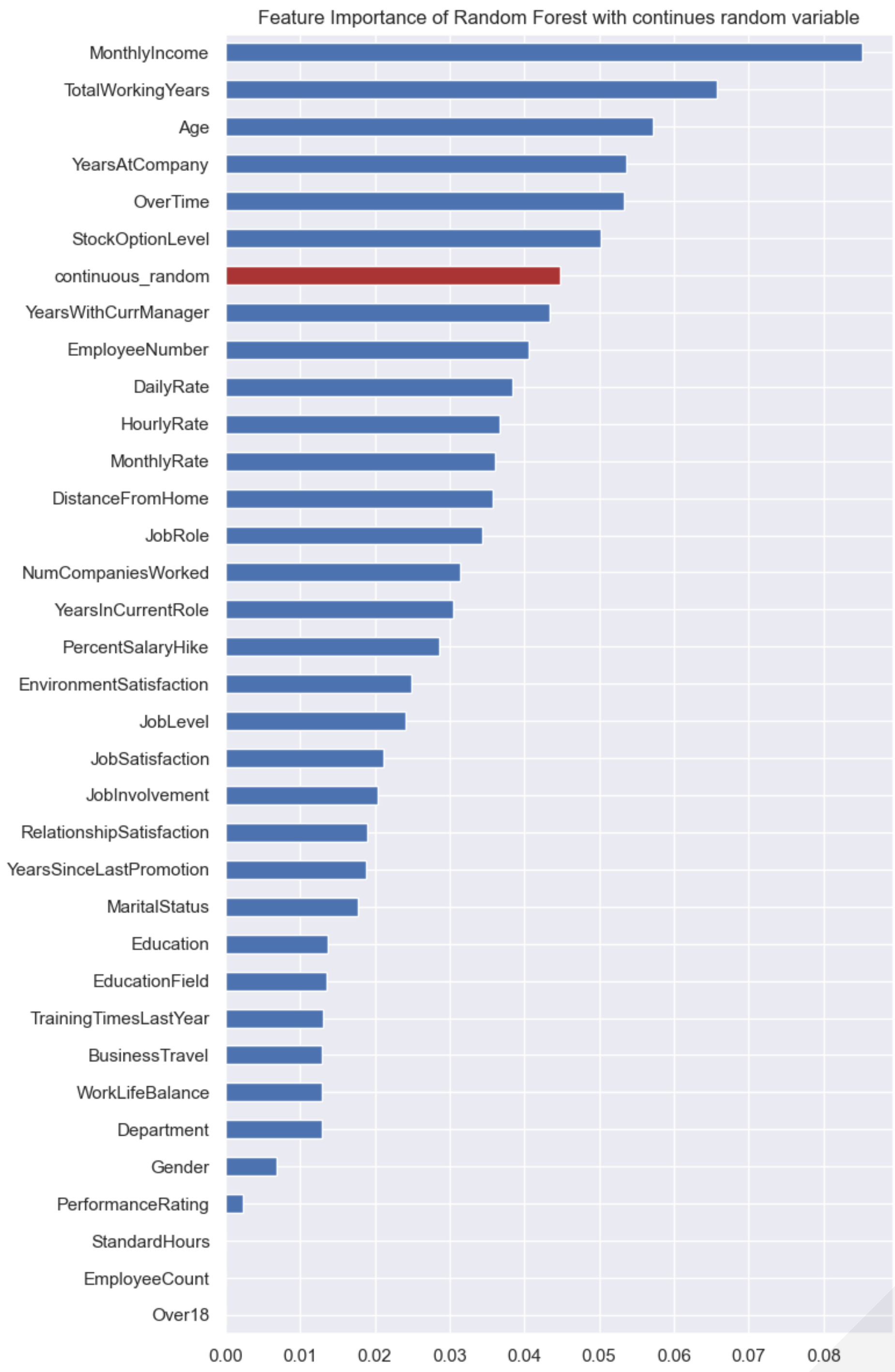
Let's mess with shit

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



New feature importances

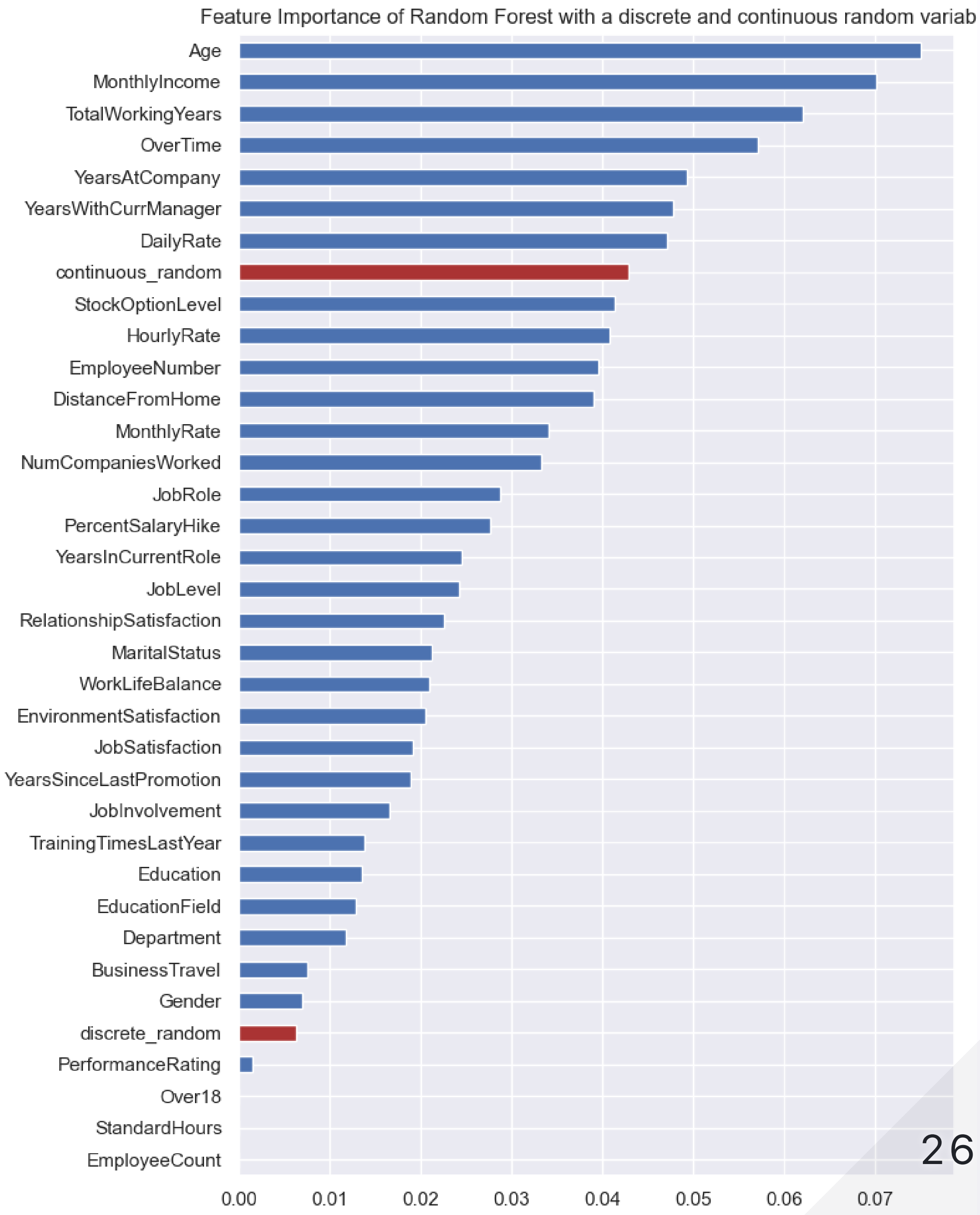
- 7th highest is random...????
- What does this mean for variables below random? No value?



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 unpredictive variables due to `overfitting`.
- Feature importance is calculated purely on the training data. Does not reflect performance on unseen data.

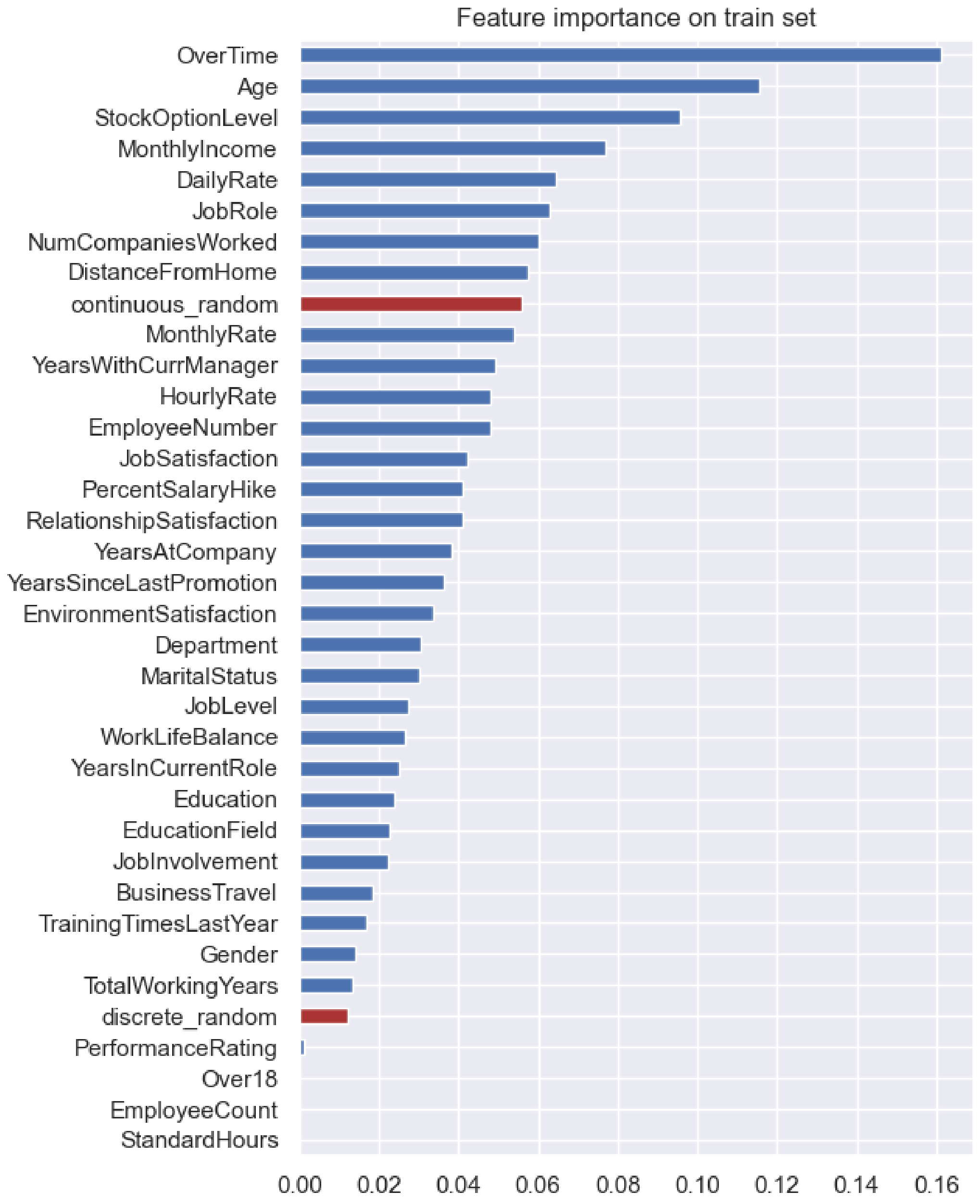
The solution: Permutation Importance

- Model agnostig 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

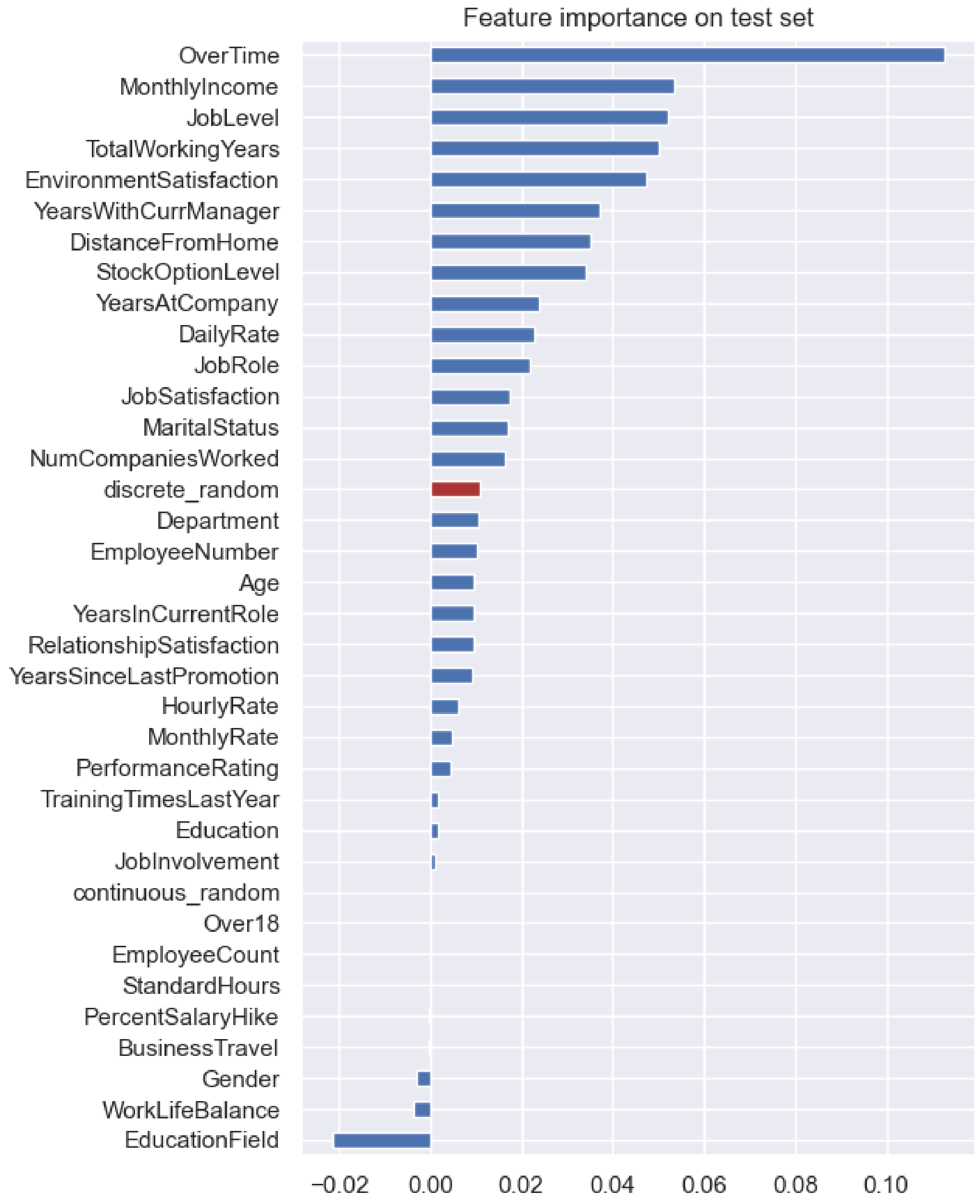
Algo

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

What happens with correlated columns?



Feature importance on train set
OverTime
Age
StockOptionLevel
MonthlyIncome
DailyRate
JobRole
NumCompaniesWorked
DistanceFromHome
continuous_random
MonthlyRate
YearsWithCurrManager
HourlyRate
EmployeeNumber
JobSatisfaction
PercentSalaryHike
RelationshipSatisfaction
YearsAtCompany
YearsSinceLastPromotion
EnvironmentSatisfaction
Department
MaritalStatus
JobLevel
WorkLifeBalance
YearsInCurrentRole
Education
EducationField
JobInvolvement
BusinessTravel
TrainingTimesLastYear
Gender
TotalWorkingYears
discrete_random
PerformanceRating
Over18
EmployeeCount
StandardHours
0.00
0.02
0.04
0.06
0.08
0.10
0.12
0.14
0.16



Feature importance on test set
OverTime
MonthlyIncome
JobLevel
TotalWorkingYears
EnvironmentSatisfaction
YearsWithCurrManager
DistanceFromHome
StockOptionLevel
YearsAtCompany
DailyRate
JobRole
JobSatisfaction
MaritalStatus
NumCompaniesWorked
discrete_random
Department
EmployeeNumber
Age
YearsInCurrentRole
RelationshipSatisfaction
YearsSinceLastPromotion
HourlyRate
MonthlyRate
PerformanceRating
TrainingTimesLastYear
Education
JobInvolvement
continuous_random
Over18
EmployeeCount
StandardHours
PercentSalaryHike
BusinessTravel
Gender
WorkLifeBalance
EducationField
−0.02
0.00
0.02
0.04
0.06
0.08
0.10

Class Imbalance

Techniques typically used:

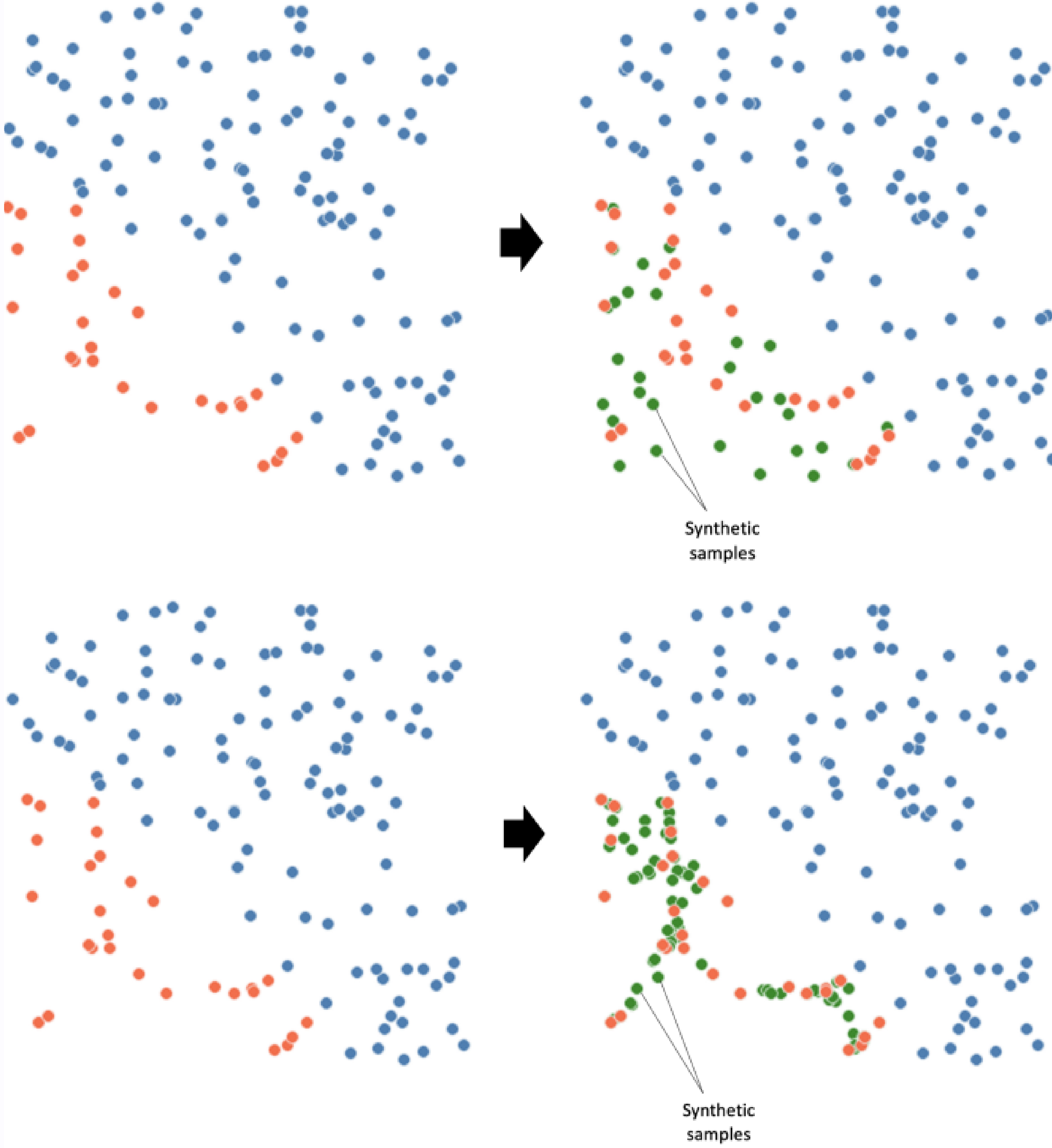
- Oversampling
- Undersampling
- Smote

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



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

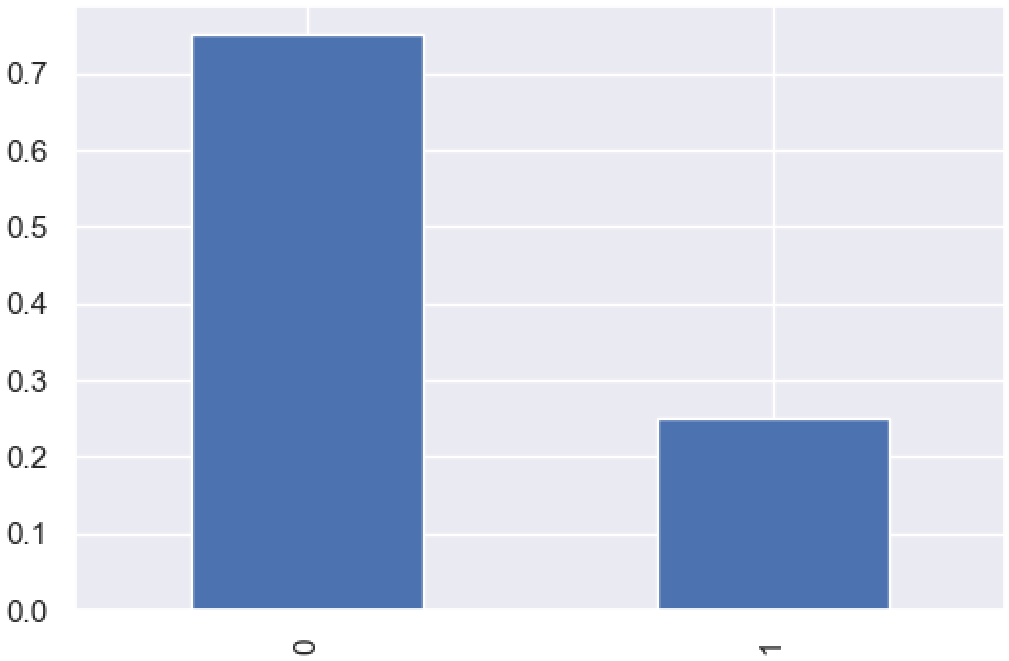
Class Weights

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

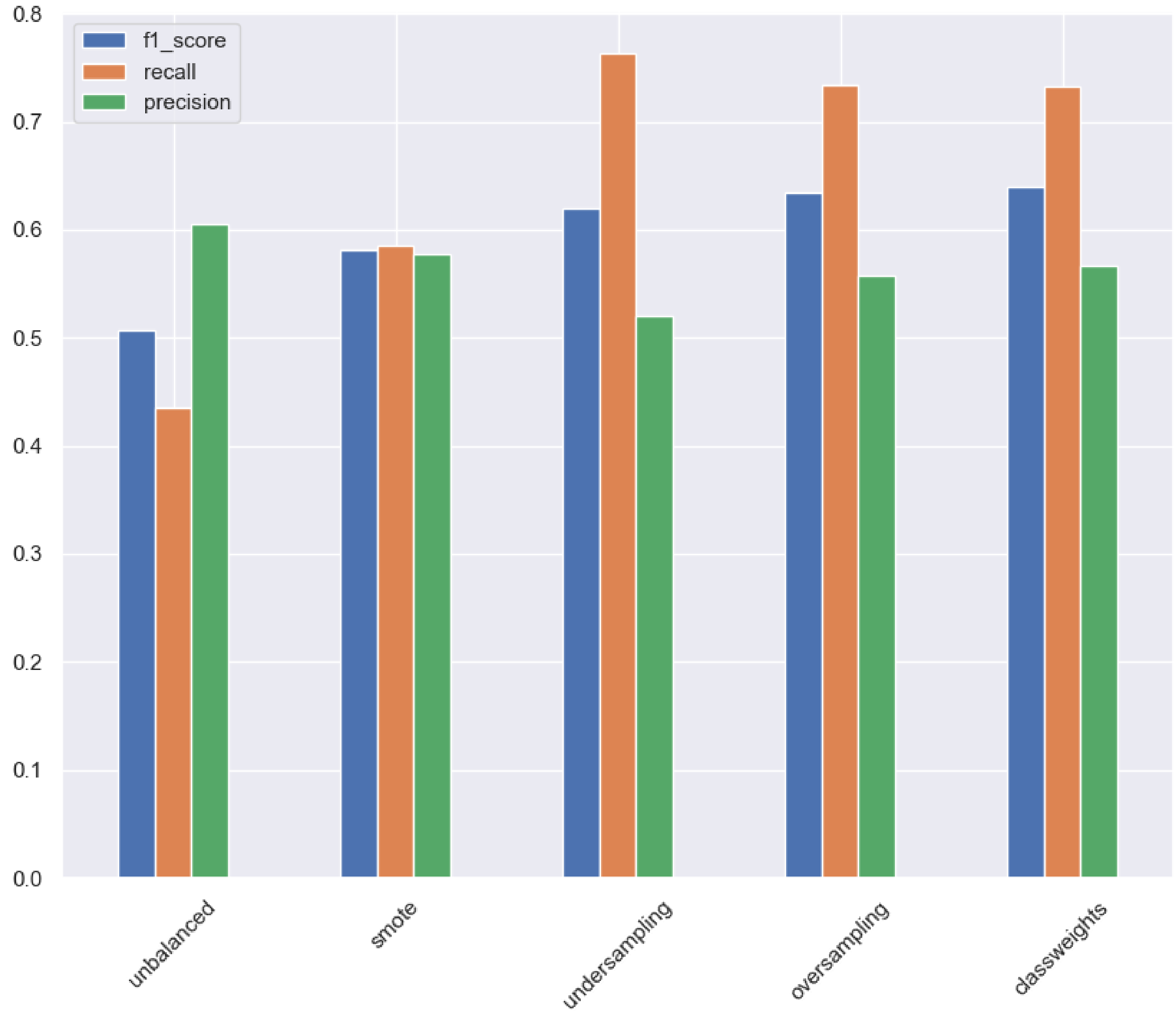
$$n_{minority} \times w_{minority} = n_{majority} \times w_{majority}$$

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

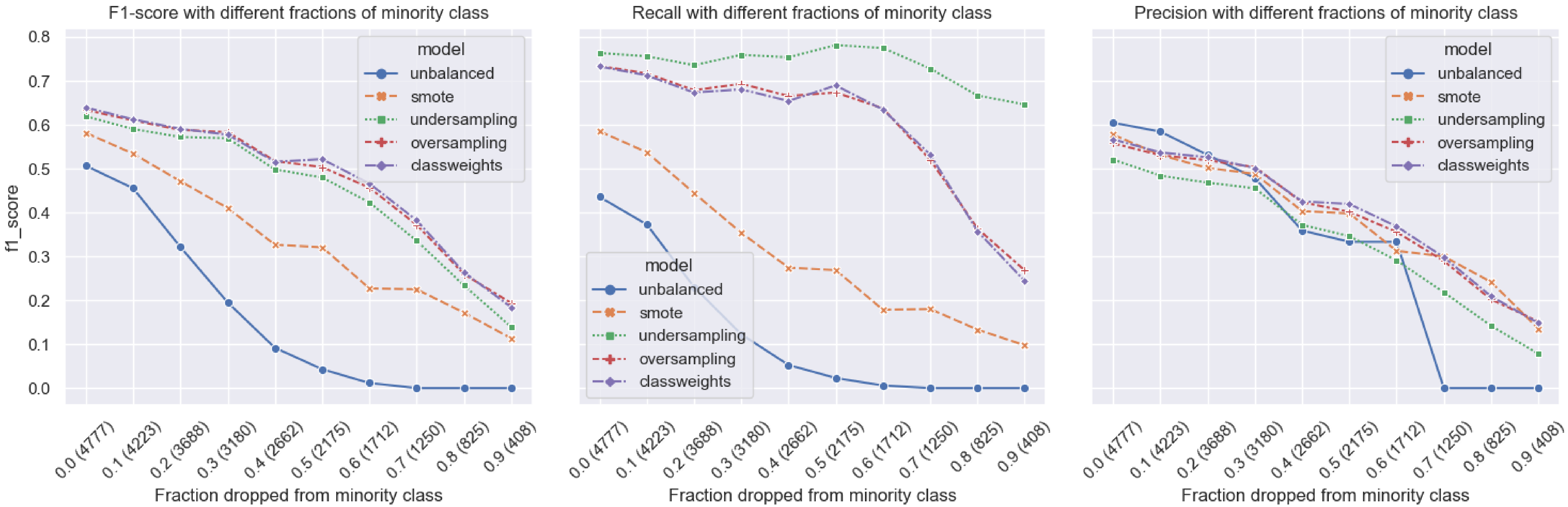
Target is imbalanced:



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


0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0.0
f1_score
recall
precision
unbalanced
smote
undersampling
oversampling
classweights

What happens if we make the data more imbalanced?



F1-score with different fractions of minority class
Recall with different fractions of minority class
Precision with different fractions of minority class
f1_score
model
unbalanced
smote
undersampling
oversampling
classweights
0.0 (4777)
0.1 (4223)
0.2 (3688)
0.3 (3180)
0.4 (2662)
0.5 (2175)
0.6 (1712)
0.7 (1250)
0.8 (825)
0.9 (408)
Fraction dropped from minority class

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?