MMA/MMAI 869
Machine Learning and AI

Advanced Topics

Stephen Thomas

Updated: November 8, 2022

Smith SCHOOL OF BUSINESS | Queen's University

TLDR: Uncle Steve's Guide to Success

- Do ***feature engineering*** to give algorithms more juice
- Follow best practices to ***avoid leakage***:
  - Only fit/train transformers on training
- Remove bad/useless features via ***feature selection***
- Try a few ***different*** ML algorithms
- ***Tune the hyperparameters*** of each algorithm
- Deal with ***class imbalance***
- Use ***cross validation*** on each model pipeline to find best
- Use ***pipelines*** to make coding easier

meh → 

Machine Learning Terminology

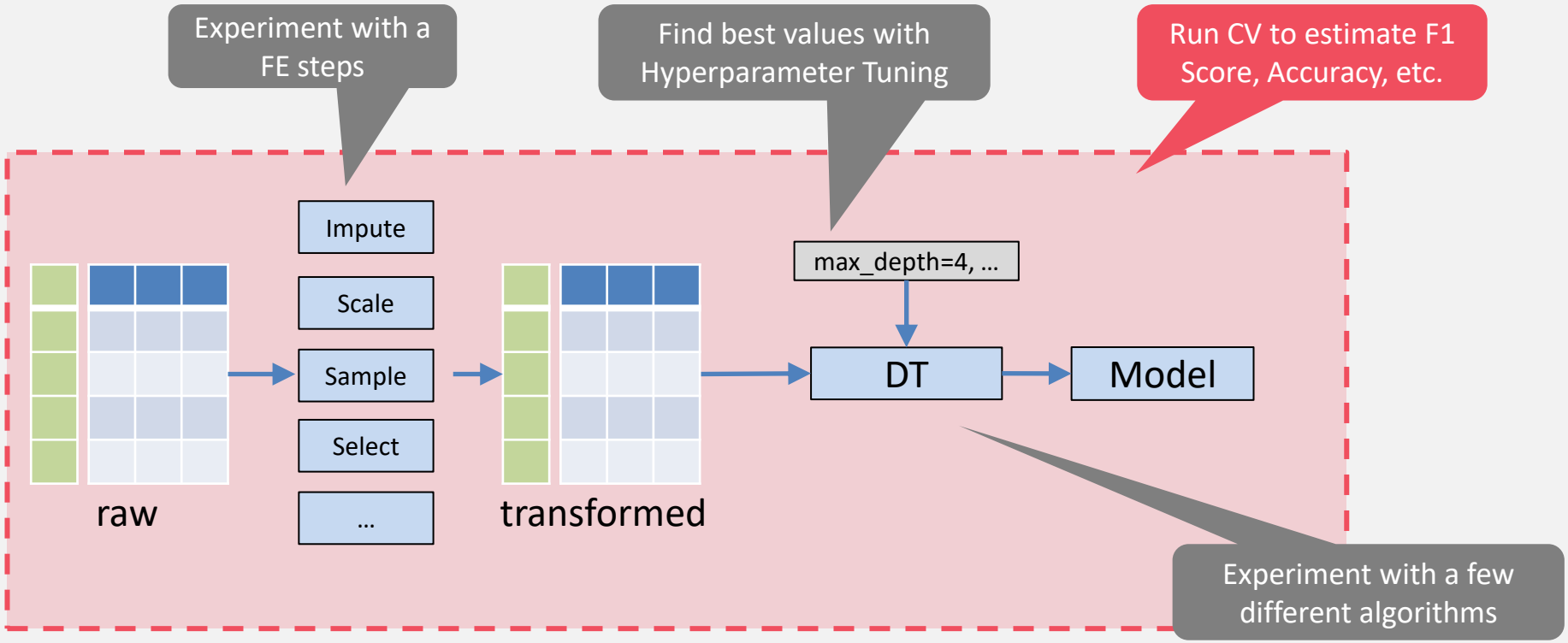
Features
(inputs, independent variables, X, columns)

Target or **Label**
(response, output, dependent variable, Y)

Instances
(rows, cases, records)

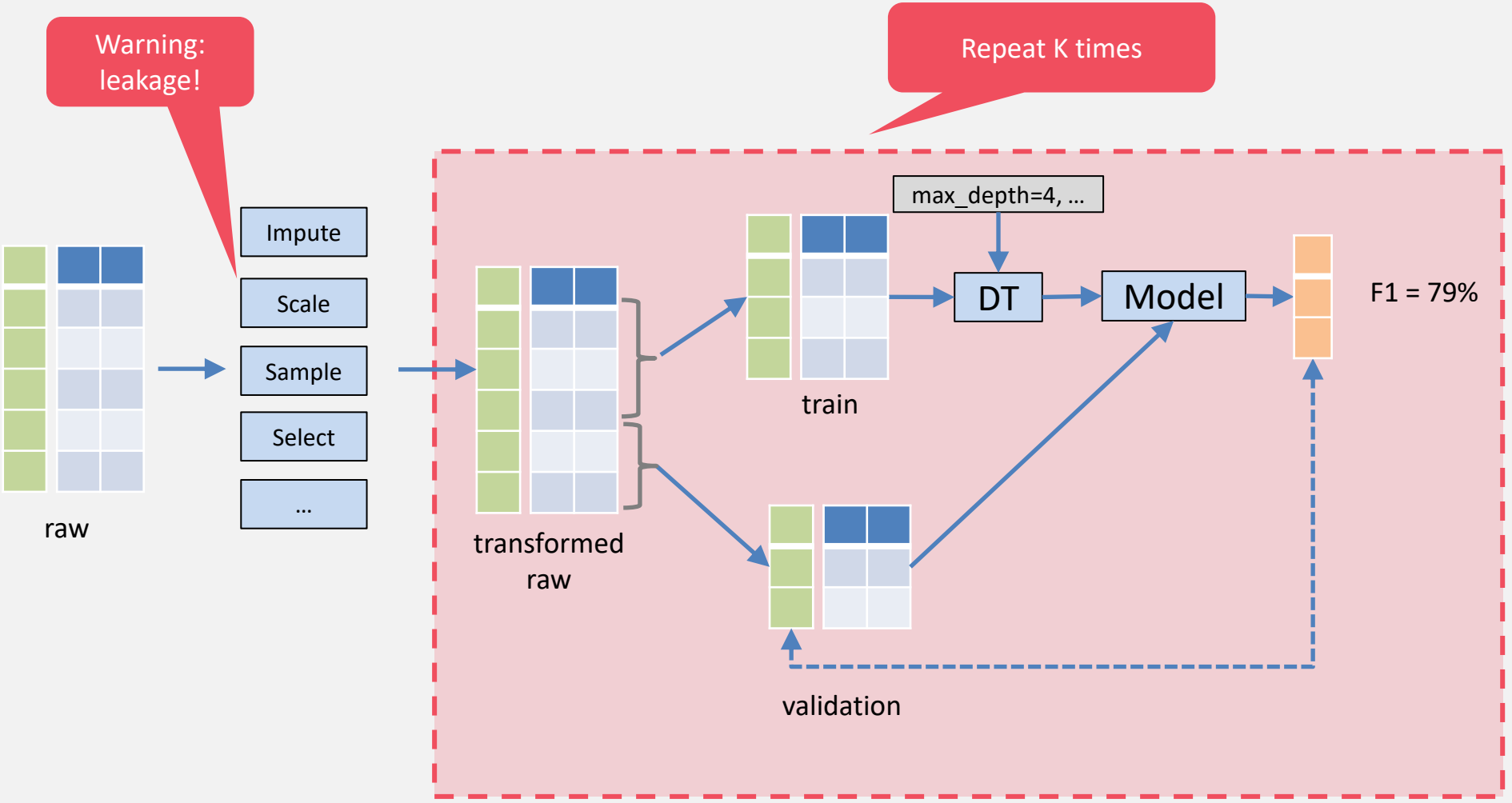
Age	Income	Married	Citizenship	Default
55	36,765	True	Canada	True
66	87,983	True	Canada	True
21	24,354	False	USA	False
24	56,654	True	Canada	False
34	98,324	False	UK	False
36	132,229	False	Germany	True
28	35,000	True	Canada	False
49	50,334	True	Canada	False

Big Picture: Conceptually

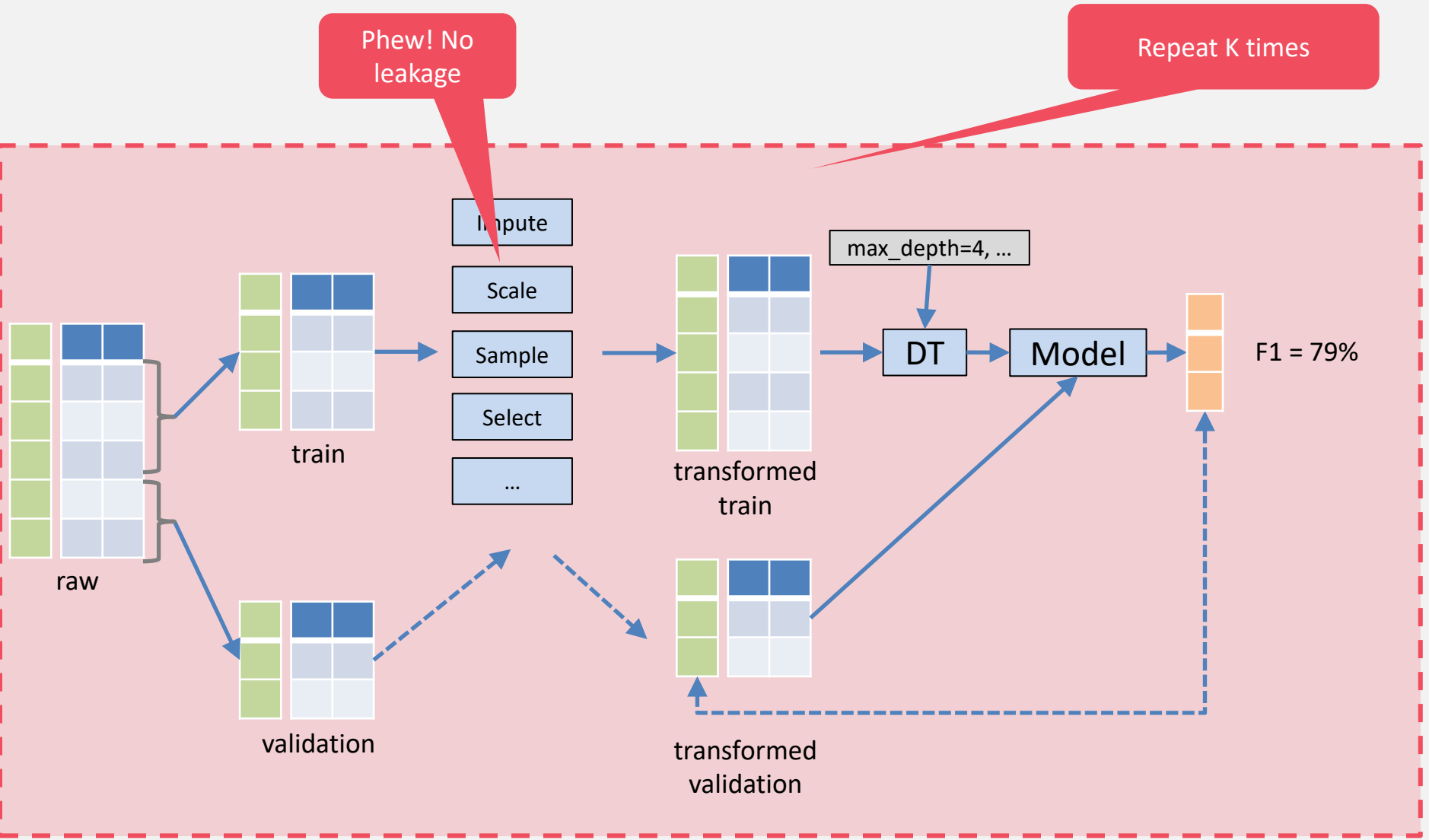


- To estimate F1-Score, need to run it all through CV

Big Picture: Cross-Validation ("FE First")



Big Picture: Cross Validation ("FE in CV")



Big Picture: It's Easy with Pipelines

Smith SCHOOL OF BUSINESS Queen's University

[7]:
```
from sklearn.pipeline import Pipeline, make_pipeline
from sklearn.tree import DecisionTreeClassifier
from sklearn.decomposition import PCA
from sklearn.preprocessing import StandardScaler
from sklearn.feature_selection import RFE
from sklearn.model_selection import GridSearchCV
from sklearn.svm import SVR

pipe1 = make_pipeline(StandardScaler(),
                      PCA(n_components=10),
                      RFE(SVR(kernel="linear")),
                      DecisionTreeClassifier(random_state=223))

param_grid = {
    'standardscaler__with_mean': [True, False],
    'pca__n_components': [5, 10, 20],
    'rfe__n_features_to_select': [None, 10, 20],
    'decisiontreeclassifier__max_depth': [None, 3, 10],
    'decisiontreeclassifier__criterion': ['gini', 'entropy'],
    'decisiontreeclassifier__class_weight':[None, 'balanced'],
}

search = GridSearchCV(pipe1, param_grid,
                      cv=3, n_jobs=5, scoring='f1_micro', return_train_score=True, verbose=2)
```

[8]:
```
search = search.fit(X_train, y_train)
```

Feature Engineering

Feature Selection

Hyper Parameter Tuning

Cross Validation

FEATURE ENGINEERING

Motivation

Why is Steve sad?

Date	Sad?
"2019-07-29"	Yes
"2019-08-05"	No
"2019-08-12"	Yes
"2019-07-28"	No
"2019-08-03"	No
"2019-08-10"	No
…	

Raw feature

Date	Day of Week	Sad?
"2019-07-29"	Monday	Yes
"2019-08-05"	Wednesday	No
"2019-08-12"	Monday	Yes
"2019-07-28"	Sunday	No
"2019-08-03"	Saturday	No
"2019-08-10"	Saturday	No
…		

Engineered feature

Feature Engineering is a Big Deal

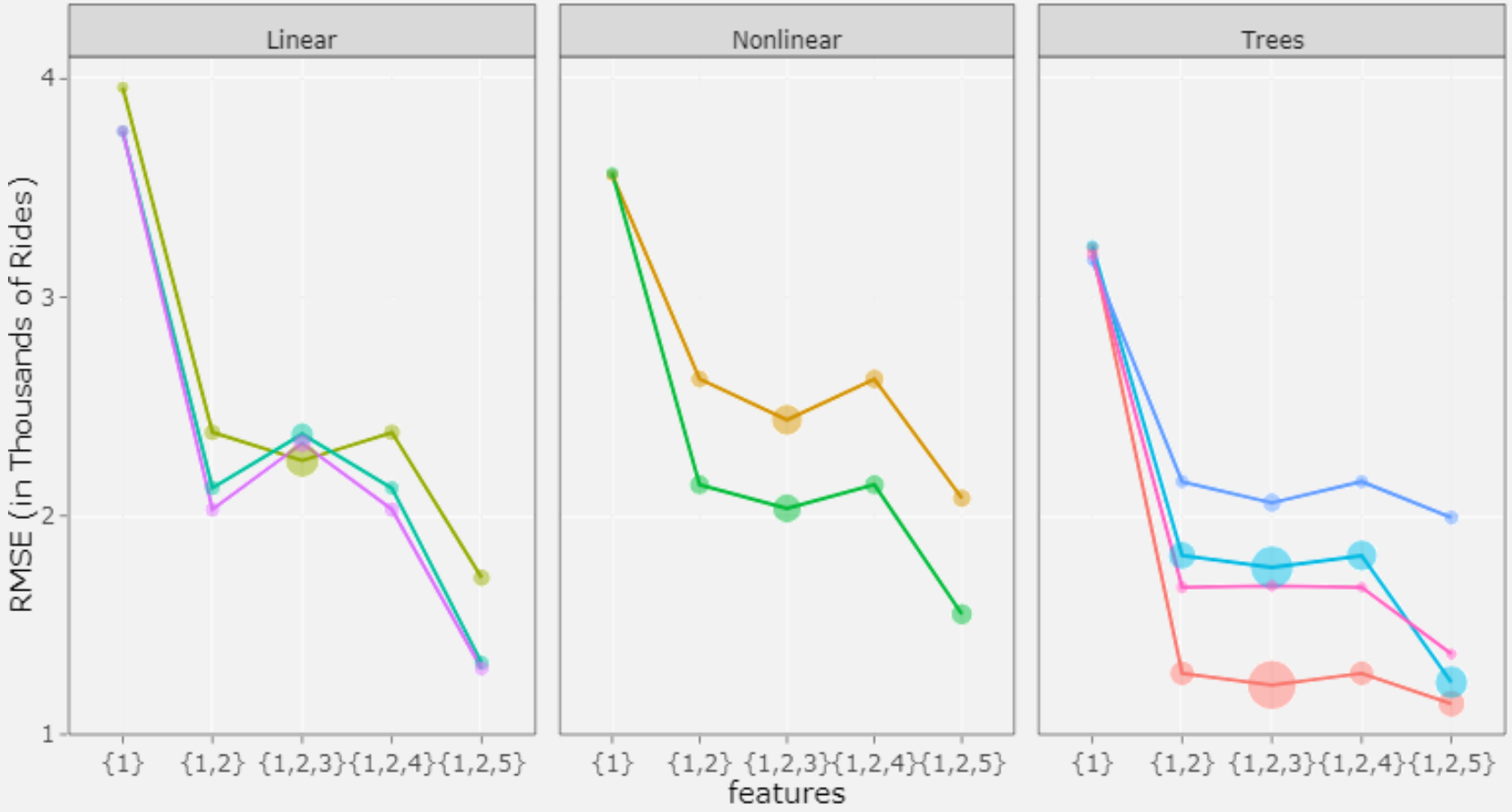
Feature engineering: creating new features to allow machine learning algorithms work better

Uses ***human creativity***

"*Applied machine learning is basically feature engineering.*"
- Andrew Ng

Quick Example

- Predict number of train rides per day in Chicago
- Better and better sets of features tried on three different types of models



Kuhn et al. Feature Engineering. Figure 1.7.

Uncle Steve's Big List of FE Transformations

- Imputation: mean, median, most freq, multiple, …
- Scaling: Log, standardize, minmax, BoxCox, YeoJohnson…
- Discretization/Binning
- Encoding: OHE, ordinal, target, …
- Non-linear transformations: $1/x$, x^2, x^3, $|x|$, 2^x, $\sin(x)$, …
- Linear and non-linear combinations
- Cardinality reduction
- Dimensionality reduction: PCA, SVD, MDS, NMF, Factor analysis, …
- Aggregations
- Splines
- Clustering
- Embeddings
- Dates/times
- Time series: Peaks, mean, max, min, entropy, …
- Text: BOW, topic models, word2vec, embeddings
- Images/Video: Edges, interest points, corners, …
- …

Awesome Tools

Python

- sklearn.preprocessing
- category_encoders
- Featuretools
- autofeat
- Tsfresh
- dirty_cat
- Mlxtend

`sklearn.preprocessing`: Preprocessing and Normalization

The `sklearn.preprocessing` module includes scaling, centering, normalization, binarization methods.

User guide: See the Preprocessing data section for further details.

`preprocessing.Binarizer`(*[, threshold, copy])	Binarize data (set feature values to 0 or 1) according to a threshold.
`preprocessing.FunctionTransformer`([func, ...])	Constructs a transformer from an arbitrary callable.
`preprocessing.KBinsDiscretizer`([n_bins, ...])	Bin continuous data into intervals.
`preprocessing.KernelCenterer`()	Center an arbitrary kernel matrix K.
`preprocessing.LabelBinarizer`(*[, neg_label, ...])	Binarize labels in a one-vs-all fashion.
`preprocessing.LabelEncoder`()	Encode target labels with value between 0 and n_classes-1.
`preprocessing.MultiLabelBinarizer`(*[, ...])	Transform between iterable of iterables and a multilabel format.
`preprocessing.MaxAbsScaler`(*[, copy])	Scale each feature by its maximum absolute value.
`preprocessing.MinMaxScaler`([feature_range, ...])	Transform features by scaling each feature to a given range.
`preprocessing.Normalizer`([norm, copy])	Normalize samples individually to unit norm.
`preprocessing.OneHotEncoder`(*[, categories, ...])	Encode categorical features as a one-hot numeric array.
`preprocessing.OrdinalEncoder`(*[, ...])	Encode categorical features as an integer array.
`preprocessing.PolynomialFeatures`([degree, ...])	Generate polynomial and interaction features.
`preprocessing.PowerTransformer`([method, ...])	Apply a power transform featurewise to make data more Gaussian-like.
`preprocessing.QuantileTransformer`(*[, ...])	Transform features using quantiles information.
`preprocessing.RobustScaler`(*[, ...])	Scale features using statistics that are robust to outliers.
`preprocessing.SplineTransformer`([n_knots, ...])	Generate univariate B-spline bases for features.
`preprocessing.StandardScaler`(*[, copy, ...])	Standardize features by removing the mean and scaling to unit variance.

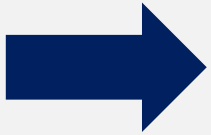
`preprocessing.add_dummy_feature`(X[, value])	Augment dataset with an additional dummy feature.
`preprocessing.binarize`(X, *[, threshold, copy])	Boolean thresholding of array-like or scipy.sparse matrix.
`preprocessing.label_binarize`(y, *, classes)	Binarize labels in a one-vs-all fashion.
`preprocessing.maxabs_scale`(X, *[, axis, copy])	Scale each feature to the [-1, 1] range without breaking the sparsity.
`preprocessing.minmax_scale`(X[, ...])	Transform features by scaling each feature to a given range.
`preprocessing.normalize`(X[, norm, axis, ...])	Scale input vectors individually to unit norm (vector length).
`preprocessing.quantile_transform`(X, *[, ...])	Transform features using quantiles information.
`preprocessing.robust_scale`(X, *[, axis, ...])	Standardize a dataset along any axis.
`preprocessing.scale`(X, *[, axis, with_mean, ...])	Standardize a dataset along any axis.
`preprocessing.power_transform`(X[, method, ...])	Power transforms are a family of parametric, monotonic transformations that are applied to make data more Gaussian-like.

Note: FE Steps are Transformations

- ***Transformations*** take one+ features, and create one+ new features
- Example:

ID	Salary	…
1	56214	
2	15784	
3	100457	
4	36548	
5	92104	
6	8754	

$$X_{new} = \frac{X_{old} - min}{max - min}$$



ID	Salary_MinMax	…
1	0.47	
2	0.09	
3	0.84	
4	0.31	
5	0.78	
6	0.04	

```
1 from sklearn.preprocessing import MinMaxScaler
2
3 scaler = MinMaxScaler()
4 scaler.fit(X_train[['Salary']])
5
6 X_train['Salary_MinMax'] = scaler.transform(X_train[['Salary']])
```

Handling Missing Data

- Usually have some missing data

ID	Age	Gender	Income	Marital Status	Children	Job	Buy?
1		A	150,000		1	Engineer	No
2	27	B	50,000			Teacher	No
3		A	100,000	Married	2		Yes
4	25	B			2	Engineer	Yes
5	35	B		Single	0	Doctor	Yes
6		A	50,000		0	Teacher	No
7	33	B	60,000	Single		Teacher	No
8	20	B	10,000			Student	No

- Missing not at random (MNAR)
  - Value is missing because of the true value itself
- Missing at random (MAR)
  - Missing due to a value of another feature
- Missing completely at random (MCAR)
  - No pattern in when the value is missing

Table from Table 5.2 of Designing Machine Learning Systems, Chip Huyen

Handling Missing Data

- Solutions:
  - Manually fix missing values
  - Use algorithms that are robust to missing data
  - Delete features and/or instances with missing values
  - Simple Imputation
  - Multiple Imputation

ID	Salary	Age	City
1	56214	56	Toronto
2	157784	42	Kingston
3		29	Toronto
4	36548	22	Ottawa
5	0		Kingston
6		87	

Missing Data: Manually Fix

- Sometimes, missing values have a "meaning"
- Replace NaNs with meaningful value
- Ideal situation: easiest fix

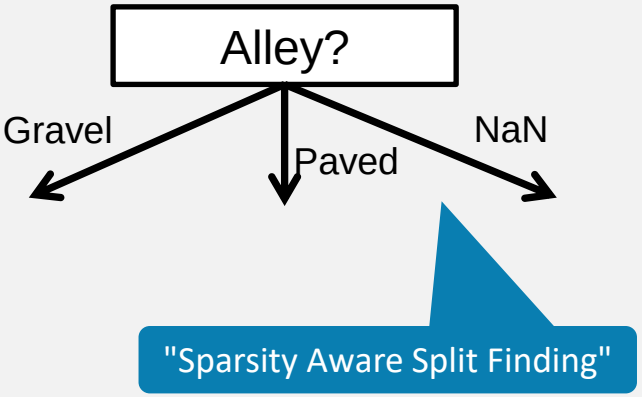
Replace NaNs with "No Alley"

ID	Address	HouseType	Bedrooms	Alley	Price
1	3638 Robson St	Apartment	3	NaN	649155
2	2301 Pine Street	Duplex	4	Gravel	723127
3	1682 Bayfield St	Detached	2	NaN	751975
4	1674 Bayfield St	Single	2	NaN	816670
5	2633 Pape Ave	Single	3	Paved	321588
6	503 Queens Bay	Apartment	3	Paved	922380

House Price Data

Missing Data: Use Robust Algorithms

- Some algorithms can handle missing data internally
  - XGBoost
  - LGBM
  - CatBoost
- You don't have to do anything



ID	Address	HouseType	Bedrooms	Alley	Price
1	3638 Robson St	Apartment	3	NaN	649155
2	2301 Pine Street	Duplex	4	Gravel	723127
3	1682 Bayfield St	Detached	2	NaN	751975
4	1674 Bayfield St	Single	2	NaN	816670
5	2633 Pape Ave	Single	3	Paved	321588
6	503 Queens Bay	Apartment	3	Paved	922380

House Price Data

Missing Data: Delete Features and/or Instances

- Sometimes simplest to delete features and/or instances with missing data
  - If a feature is mostly NaN
  - If only a few instances with NaN

Feature mostly NaN

Instance with NaN

ID	Salary	Age	City	Job Title
1	56214	56	Toronto	NaN
2	157784	42	Kingston	NaN
3	49554	29	Toronto	NaN
4	36548	22	Ottawa	NaN
5	84577	54	Kingston	Sr. Analyst II
6	Nan	87	NaN	NaN

Missing Data: Simple Imputation

- Calculate mean value (on training data)
- Replace NaNs (on training and testing) with mean
- Best for Missing at Random (MAR) data
- Notes:
  - Can also use median
  - For categorical features: use most frequent value
  - Can add missing indicator as well

Income
NaN
6836
2319
1236
5003
886
1442
2978
NaN
5400
NaN
2996
3780
4211
8543

$$\mu = 3802.5$$

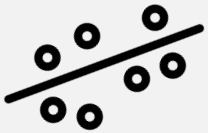
```
1 from sklearn.impute import SimpleImputer
2
3 imputer = SimpleImputer(strategy="mean")
4 imputer = imputer.fit(X_train[['Income']])
5
6 X_train['Income_Imputed'] = imputer.transform(X_train[["Income"]])
7 X_test['Income_Imputed'] = imputer.transform(X_test[['Income']])
```

Income_ Imputed	Income_ Missing
3802.5	1
6836	0
2319	0
1236	0
5003	0
886	0
1442	0
2978	0
3802.5	1
5400	0
3802.5	1
2996	0
3780	0
4211	0
8543	0

Missing Data: Multiple Imputation

- Model trained on training data to predict missing values using other features
  - Linear Regression, KNN, Decision Tree
- Use model to replace NaNs (on training and testing)
- Best for Missing Not at Random (MNAR) data
- Notes:
  - No packages in Python support multiple imputation for categorical
  - MICE package in R does support numeric and categorical
  - IMHO, a lot of trouble for relatively little (or no) gain

Age	Income
34	NaN
54	6836
29	2319
28	1236
53	5003
25	886
32	2442
68	2978
30	NaN
23	5400
51	NaN
47	5996
76	5780
45	4211
30	3543



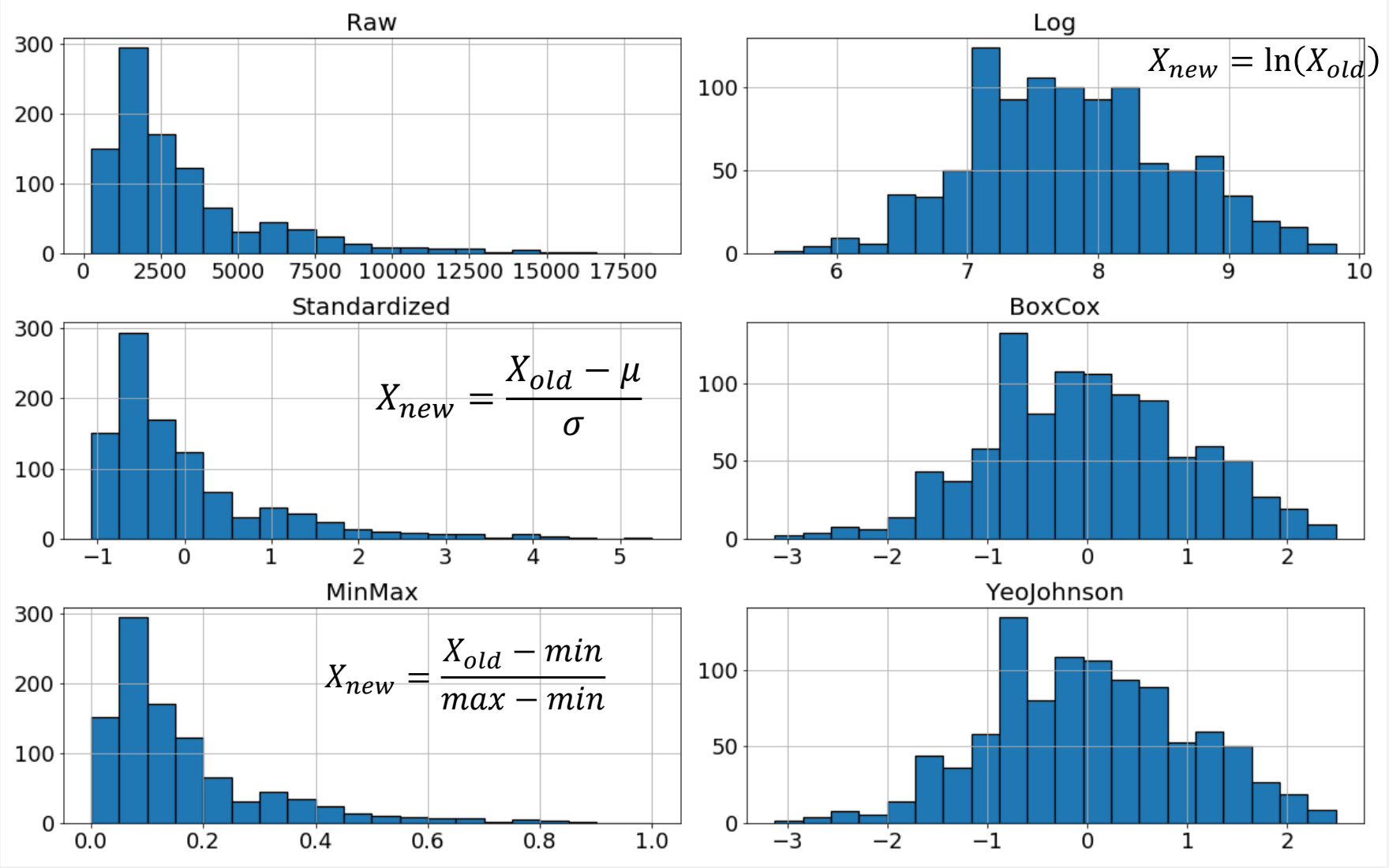
Linear Regression Model

```
from sklearn.experimental import enable_iterative_imputer
from sklearn.impute import IterativeImputer
from sklearn.linear_model import LinearRegression

imputer = IterativeImputer(estimator=LinearRegression())
imputer = imputer.fit(X_train)

imputer.transform(X_train)
```

Age	Income_Imputed
34	3388.3
54	6836
29	2319
28	1236
53	5003
25	886
32	2442
68	2978
30	3154.13
23	5400
51	4383.4
47	5996
76	5780
45	4211
30	3543

Scaling

Scaling: any transformation that changes the values of a feature to either fall within a different range, or have a new distributional shape, or both.



Standardization

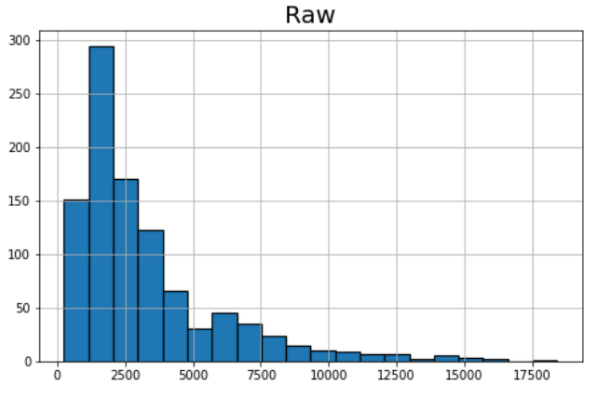
PID	Salary
29	6836
535	2319
695	1236
557	5003
836	886
596	1442
165	2978
918	2359
495	2996
824	3780

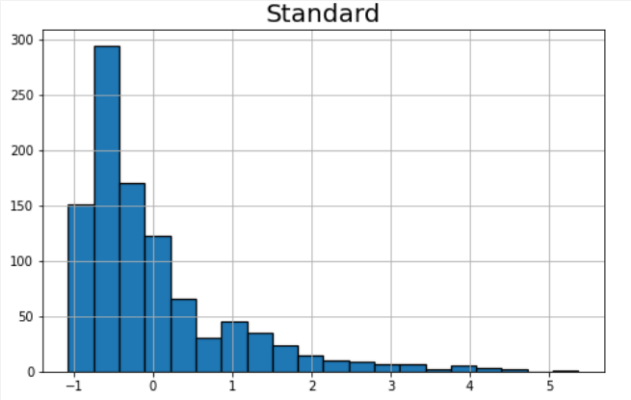
$$X_{new} = \frac{X_{old} - \mu}{\sigma}$$

```
1 from sklearn.preprocessing import StandardScaler
2
3 scaler = StandardScaler()
4 scaler.fit(X_train[['Salary']])
5
6 X_train['Salary_Std'] = scaler.transform(X_train[['Salary']])
```

PID	Salary_Std
29	1.199912
535	-0.359630
695	-0.733547
557	0.567050
836	-0.854388
596	-0.662423
165	-0.132103
918	-0.345819
495	-0.125888
824	0.144796

- Centers and scales
- Usually recommended
- Good for SVM
- Good for NN
- Good for l1, l2 regularization,
- Good for distance metrics (KNN)
- Not needed for tree-based





Min-Max Scaling (Normalization)

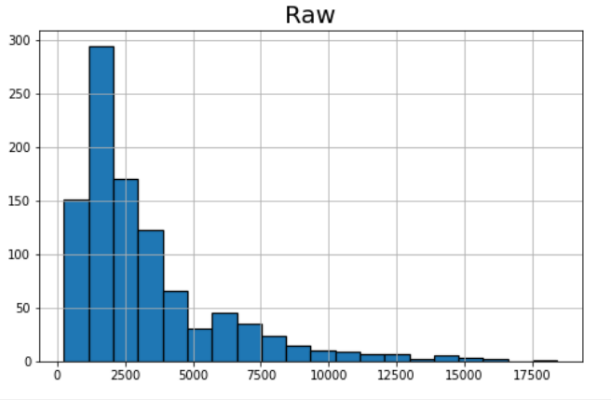
PID	Salary
29	6836
535	2319
695	1236
557	5003
836	886
596	1442
165	2978
918	2359
495	2996
824	3780

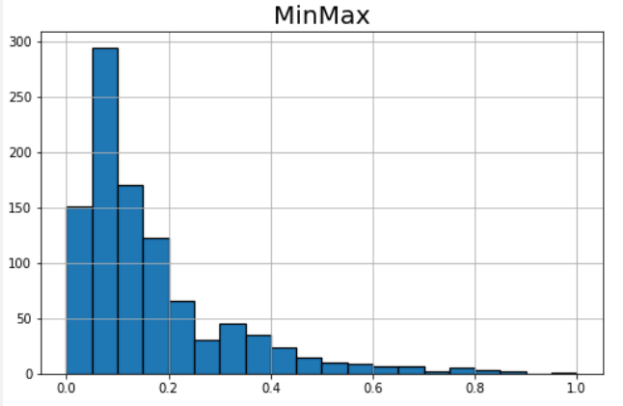
$$X_{new} = \frac{X_{old} - min}{max - min}$$

```
1 from sklearn.preprocessing import MinMaxScaler
2
3 scaler = MinMaxScaler()
4 scaler.fit(X_train[['Salary']])
5
6 X_train['Salary_MinMax'] = scaler.transform(X_train[['Salary']])
```

PID	Salary_MinMax
29	0.362386
535	0.113844
695	0.054253
557	0.261527
836	0.034995
596	0.065588
165	0.150105
918	0.116045
495	0.151095
824	0.194234

- Scales
- Good for NN





Log

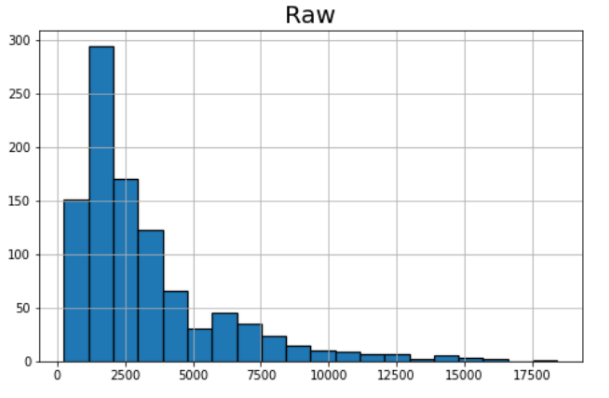
PID	Salary
29	6836
535	2319
695	1236
557	5003
836	886
596	1442
165	2978
918	2359
495	2996
824	3780

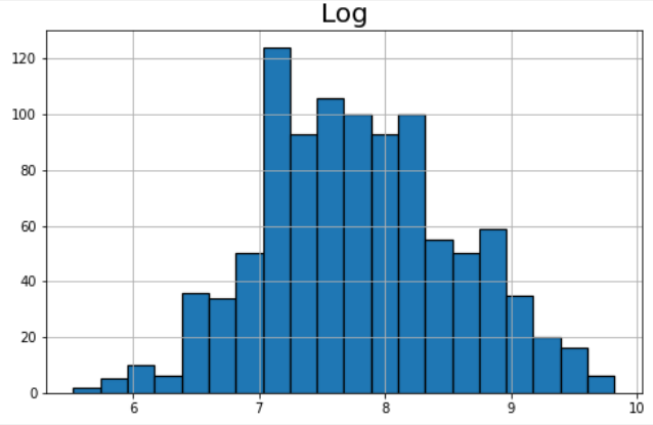
$$X_{new} = \ln(X_{old})$$

```
1 from sklearn.preprocessing import FunctionTransformer
2
3 scaler = FunctionTransformer(np.log1p, validate=True)
4 scaler.fit(X_train[['Salary']])
5 X_train['Salary_Log'] = scaler.transform(X_train[['Salary']])
```

PID	Salary_Log
29	8.830104
535	7.749322
695	7.120444
557	8.517993
836	6.787845
596	7.274480
165	7.999343
918	7.766417
495	8.005367
824	8.237744

- De-skews
- Removes effect of outliers





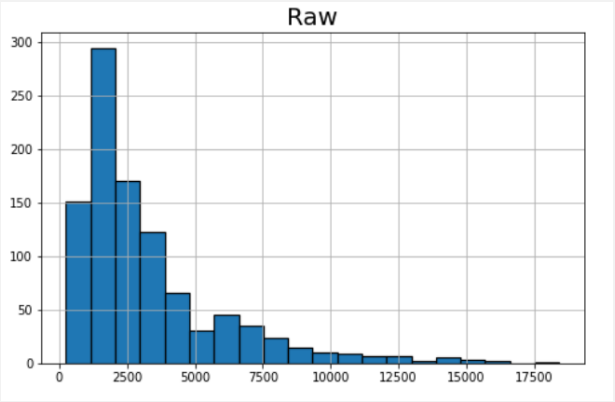
Box-Cox

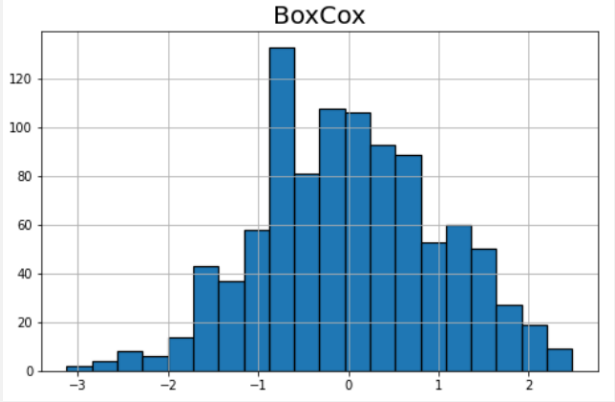
PID	Salary
29	6836
535	2319
695	1236
557	5003
836	886
596	1442
165	2978
918	2359
495	2996
824	3780

```
from sklearn.preprocessing import PowerTransformer

scaler = PowerTransformer(method='box-cox')
scaler.fit(X_train[['Salary']])
X_train['Salary_BoxCox'] = scaler.transform(X_train[['Salary']])
```

PID	Salary_BoxCox
29	1.291730
535	-0.054503
695	-0.893962
557	0.914991
836	-1.355653
596	-0.684355
165	0.267494
918	-0.032279
495	0.275172
824	0.568502

- De-skews
- Removes effect of outliers
- Makes more normal/Gaussian





Exercise

- Create standardization and min-max scalings of the Age feature:

PID	Age	…
1	23	
2	21	
3	45	
4	34	
5	23	
6	34	
7	30	
8	31	

Age_Standard	Age_MinMax

Hint:
Mean = 30.1
Stdev = 7.9

Discretization (Binning)

Discretization: transforming numbers into two or more buckets

CID	Grade	…
1	92	
2	86	
3	89	
4	71	
5	80	
6	62	

CID	Letter	…
1	A+	
2	A	
3	A	
4	B-	
5	A-	
6	C-	

CID	Age	…
1	20	
2	51	
3	84	
4	9	
5	36	
6	37	

CID	Life	…
1	Young	
2	Old	
3	Old	
4	Young	
5	Middle	
6	Middle	

Discretization: Probably Not Good

- Kuhn et al. strongly recommend against discretization
  - Nuance is removed → Harder for model to find pattern
  - Might make model find trend that isn't really there
  - Hard to find cut-off points between buckets

There are a number of problematic issues with turning continuous data categorical. First, it is extremely unlikely that the underlying trend is consistent with the new model. Secondly, when a real trend exists, discretizing the data is most likely making it *harder* for the model to do an effective job since all of the nuance in the data has been removed. Third, there is probably no objective rationale for a specific cut-point. Fourth, when there is no relationship between the outcome and the predictor, there is a substantial increase in the probability that an erroneous trend will be "discovered". This has been widely researched and verified. See Altman (1991), D Altman et al. (1994), and the references therein.

Kenny and Montanari (2013) do an exceptional job of illustrating how this can occur and our example follows their research. Suppose a set of variables are being screened to find which have a relationship with a numeric outcome. Figure 6.6(a) shows an example of a simulated data set with a linear trend with

Encoding

- ***Encoding***: turning categorical features into numeric features
  - One hot encoding/Dummy Coding/Effect coding
  - Ordinal encoding
  - Contrast encoding
  - Hashing
  - Target encoding (aka bin counting)
    - M-estimate
    - Polynomial
    - Target
    - James-Stein
    - …
- Note:
  - Boosting algorithms (LGBM, CatBoost) do their own encoding

One Hot Encoding

OHE: creating a new numeric feature for each level of a categorical

- Very similar to *dummy coding, effect coding*

PID	Province	…
1	ON	
2	QB	
3	AB	
4	ON	
5	ON	
6	BC	
7	AB	
8	BC	

PID	Province_QB	Province_AB	Province_ON	Province_BC	…
1	0	0	1	0	
2	1	0	0	0	
3	0	1	0	0	
4	0	0	1	0	
5	0	0	1	0	
6	0	0	0	1	
7	0	1	0	0	
8	0	0	0	1	

- If there a K levels, then you need K new features
- Best when:
  - small number of levels
  - no order between values

Exercise

- Create a OHE of the School feature for the following table:

PID	Age	School	…
1	23	Queen's	
2	21	Western	
3	45	Queen's	
4	34	Yale	
5	23	Queen's	
6	34	Western	
7	30	Yale	
8	31	Western	

Ordinal Encoding

Ordinal Encoding: creating arbitrary, repeatable mapping from categorical class to an integer

- Note: some people mistakenly call this "Label Encoding"

PID	Temp	…
1	cold	
2	warm	
3	cold	
4	hot	
5	cold	
6	warm	
7	warm	
8	warm	

PID	Temp	…
1	0	
2	1	
3	0	
4	2	
5	0	
6	1	
7	1	
8	1	

Levels given by human expert

PID	Province	…
1	ON	
2	QB	
3	AB	
4	ON	
5	ON	
6	BC	
7	AB	
8	BC	

PID	Province	…
1	3	
2	1	
3	2	
4	3	
5	3	
6	4	
7	2	
8	4	

Possible, but does it make sense?

Target Encoding

Target encoding takes into account the target

- E.g., compute mean target value for each level, then replace the categorical value with this mean

PID	Income	Default
1	Low	0
2	Low	1
3	Low	1
4	Low	1
5	Medium	0
6	Medium	0
7	Medium	0
8	Medium	1
9	Medium	1
10	Medium	1
11	High	0
12	High	0
13	High	0
14	High	0
15	High	1

Level	Target Mean
Low	3/4 = 0.75
Medium	3/6 = 0.50
High	1/5 = 0.20

PID	Income_TE	Default
1	0.75	0
2	0.75	1
3	0.75	1
4	0.75	1
5	0.50	0
6	0.50	0
7	0.50	0
8	0.50	1
9	0.50	1
10	0.50	1
11	0.20	0
12	0.20	0
13	0.20	0
14	0.20	0
15	0.20	1

Exercise

- Create a target encoding of the School feature for the following table:

PID	Age	School	Default
1	23	Queen's	1
2	21	Western	1
3	45	Queen's	0
4	34	Yale	1
5	23	Queen's	0
6	34	Western	1
7	30	Yale	0
8	31	Western	1

Target Encoding: Multiclass

- What if your target contains more than two classes?
  - *aka* multinomial, polynomial
- No problem: just create N new features

PID	Income	Risk
1	Low	1
2	Low	1
3	Low	2
4	Low	2
5	Medium	0
6	Medium	0
7	Medium	1
8	Medium	1
9	Medium	1
10	Medium	2
11	High	0
12	High	0
13	High	0
14	High	1
15	High	1

	0	1	2
Low	0.00	0.50	0.50
Medium	0.33	0.50	0.17
High	0.60	0.40	0.00

Actually, can remove one level

PID	Income_TE_0	Income_TE_1	Income_TE_2	Risk
1	0.00	0.50	0.50	1
2	0.00	0.50	0.50	1
3	0.00	0.50	0.50	2
4	0.00	0.50	0.50	2
5	0.33	0.50	0.17	0
6	0.33	0.50	0.17	0
7	0.33	0.50	0.17	1
8	0.33	0.50	0.17	1
9	0.33	0.50	0.17	1
10	0.33	0.50	0.17	2
11	0.60	0.40	0.00	0
12	0.60	0.40	0.00	0
13	0.60	0.40	0.00	0
14	0.60	0.40	0.00	1
15	0.60	0.40	0.00	1

Target Encoding: Overfitting

- Since Target Encoding uses target, overfitting is a concern
- Techniques to help avoid:
  - Smoothing: add a bit of random noise
  - Weighted average with global mean

PID	Income	Default
1	Low	0
2	Low	1
3	Low	1
4	Low	1
5	Medium	0
6	Medium	0
7	Medium	0
8	Medium	1
9	Medium	1
10	Medium	1
11	High	0
12	High	0
13	High	0
14	High	0
15	High	1

Level	Target Mean
Low	3/4 = 0.75
Medium	3/6 = 0.50
High	1/5 = 0.20

Global Mean = 0.46

PID	Income_TE	Default
1	0.526	0
2	0.526	1
3	0.526	1
4	0.526	1
5	0.476	0
6	0.476	0
7	0.476	0
8	0.476	1
9	0.476	1
10	0.476	1
11	0.400	0
12	0.400	0
13	0.400	0
14	0.400	0
15	0.400	1

Nice Python Package

- Many variants of target encoding

```
import category_encoders as ce

encoder = ce.BackwardDifferenceEncoder(cols=[...])
encoder = ce.BaseNEncoder(cols=[...])
encoder = ce.BinaryEncoder(cols=[...])
encoder = ce.CatBoostEncoder(cols=[...])
encoder = ce.CountEncoder(cols=[...])
encoder = ce.GLMMEncoder(cols=[...])
encoder = ce.HashingEncoder(cols=[...])
encoder = ce.HelmertEncoder(cols=[...])
encoder = ce.JamesSteinEncoder(cols=[...])
encoder = ce.LeaveOneOutEncoder(cols=[...])
encoder = ce.MEstimateEncoder(cols=[...])
encoder = ce.OneHotEncoder(cols=[...])
encoder = ce.OrdinalEncoder(cols=[...])
encoder = ce.SumEncoder(cols=[...])
encoder = ce.PolynomialEncoder(cols=[...])
encoder = ce.TargetEncoder(cols=[...])
encoder = ce.WOEEncoder(cols=[...])

encoder.fit(X, y)
X_cleaned = encoder.transform(X_dirty)
```

High-Cardinality Categorical Features

- Some categorical features have a lot of unique values
  - Many occur rarely
- Can replace rare values with "Other"

Installer Name
Bob
Steve
Steve
Sue
Tom
Steve
Steve
Bob
Meg
Jimmy
Sue
Steve
Steve
Steve
Anup
….

Unique values: 500

Value	Count
Steve	5210
Bob	3655
Jimmy	2104
Sue	563
Meg	15
Anup	8
Tom	3
…	

Installer Name
Bob
Steve
Steve
Sue
Other
Steve
Steve
Bob
Other
Jimmy
Sue
Steve
Steve
Steve
Other
….

Unique values: 6

Datetime Features

"2019-08-14 00:30:02:98" = ☹

TID	Date	…
1	"2019-06-01"	
2	"2019-06-02"	
3	"2019-08-20"	
4	"2019-12-16"	
5	"2020-01-01"	
6	"2019-08-18"	

TID	isWeekend?	Day	Month	Year	Decade	Days Since 2018	…
1	True	Saturday	June	2019	2010s	152	
2	True	Sunday	June	2019	2010s	153	
3	False	Tuesday	August	2019	2010s	232	
4	False	Monday	December	2019	2010s	350	
5	False	Wed	January	2020	2020s	366	
6	True	Sunday	August	2019	2010s	230	

Others?

Non-Linear Mathematical Transformations

- Apply mathematical transformation to a feature
  - sqrt(x)
  - 1/x
  - x^2, x^3
  - |x|
  - 2^x
  - sin(x)
  - log(x)
  - cos(x)

Age	1/Age	sin(Age)
23	0.043	-0.846
21	0.047	0.837
45	0.022	0.851
34	0.029	0.529
23	0.043	-0.846
34	0.029	0.529
30	0.033	-0.988
31	0.032	-0.404

Feature Combinations / Crossings

- Combine pairs of features with mathematics: +, -, ×, ÷

Height	Weight
150	58.2
163	84.2
170	77.0
145	68.1
185	100.5

Height × Weight
8730.0
13724.6
13090.0
9874.5
18592.5

Height ÷ Weight
2.58
1.94
2.21
2.13
1.84

Aggregations

- ***Aggregations*** take one or more (related) instances, and create one new instance.
- Useful when you have several transactions per client, but want to build a model at the client level

Transactions

TID	CID	Date	Spend
1	498	2019-06-01	$10.58
2	658	2019-06-01	$63.54
3	498	2019-06-02	$87.96
4	325	2019-06-03	$56.66
5	987	2019-06-07	$25.00
6	498	2019-06-18	$63.05

Aggregated by Client

CID	Count	Spend_min	Spend_max	Date_max	…
498	10	$5.21	$156.32	2019-08-01	
658	43	$2.32	$63.54	2019-07-25	
325	1	$87.99	$87.99	2019-06-03	
987	19	$56.33	$196.65	2019-08-07	

- pandas groupby() makes this easy

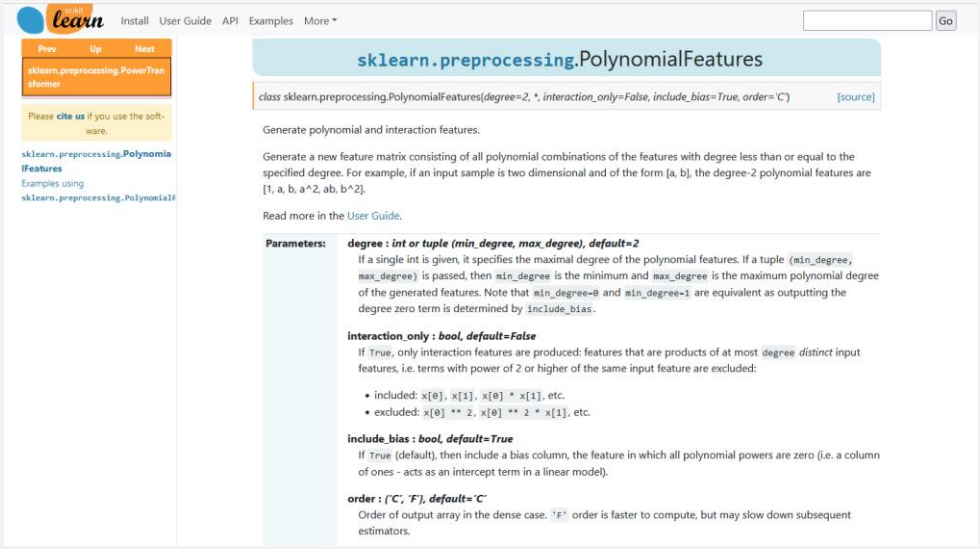
Exercise

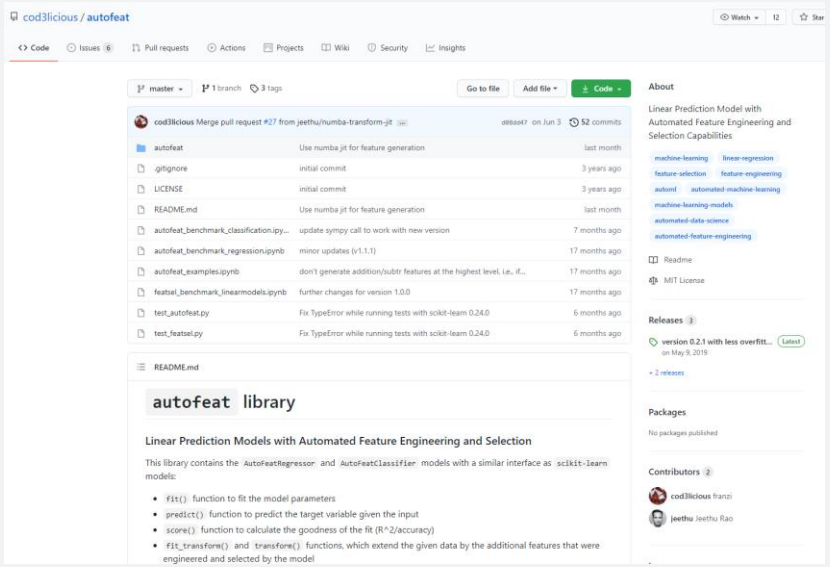
What aggregated features can you think of?

Transaction ID	Customer ID	Date	Product ID	Spend
1	498	2019-06-01	5	$10.58
1	498	2019-06-01	2	$63.54
1	498	2019-06-01	3	$87.96
2	658	2019-06-03	4	$56.66
2	658	2019-06-03	7	$25.00
3	498	2019-06-18	9	$63.05
3	498	2019-06-18	9	$144.66
4	125	2019-06-18	5	$76.56
…				

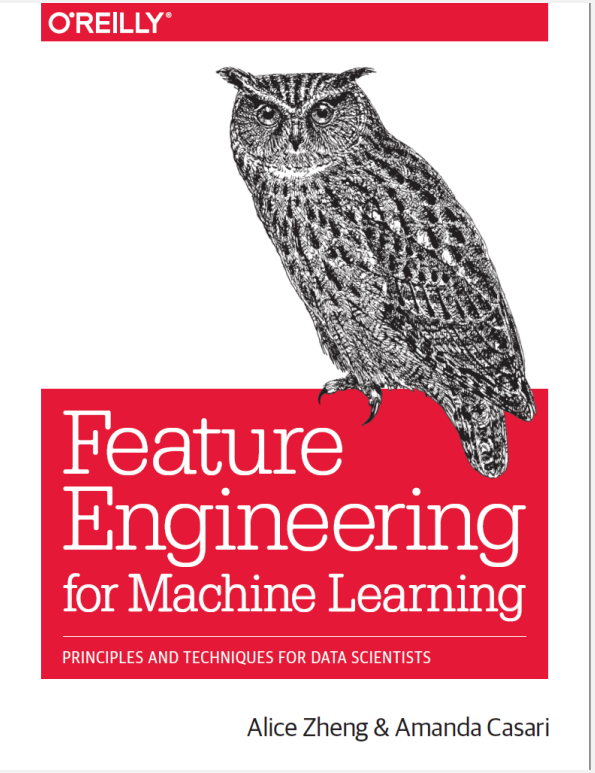
Customer ID	??	??	??	??	…
498					
658					
125					
987					
…					

Packages


learn
Install
User Guide
API
Examples
More
Go
Prev
Up
Next
sklearn.preprocessing.PowerTransformer
Please cite us if you use the software.
sklearn.preprocessing.PolynomialFeatures
Examples using
sklearn.preprocessing.PolynomialF
sklearn.preprocessing.PolynomialFeatures
class sklearn.preprocessing.PolynomialFeatures(degree=2, *, interaction_only=False, include_bias=True, order='C')
[source]
Generate polynomial and interaction features.
Generate a new feature matrix consisting of all polynomial combinations of the features with degree less than or equal to the specified degree. For example, if an input sample is two dimensional and of the form [a, b], the degree-2 polynomial features are [1, a, b, a^2, ab, b^2].
Read more in the User Guide.
Parameters:
degree : int or tuple (min_degree, max_degree), default=2
If a single int is given, it specifies the maximal degree of the polynomial features. If a tuple (min_degree, max_degree) is passed, then min_degree is the minimum and max_degree is the maximum polynomial degree of the generated features. Note that min_degree=0 and min_degree=1 are equivalent as outputting the degree zero term is determined by include_bias.
interaction_only : bool, default=False
If True, only interaction features are produced: features that are products of at most degree distinct input features, i.e. terms with power of 2 or higher of the same input feature are excluded:
included: x[0], x[1], x[0] * x[1], etc.
excluded: x[0] ** 2, x[0] ** 2 * x[1], etc.
include_bias : bool, default=True
If True (default), then include a bias column, the feature in which all polynomial powers are zero (i.e. a column of ones - acts as an intercept term in a linear model).
order : {'C', 'F'}, default='C'
Order of output array in the dense case. 'F' order is faster to compute, but may slow down subsequent estimators.


cod3licious / autofeat
Code
Issues 6
Pull requests
Actions
Projects
Wiki
Security
Insights
master
1 branch
3 tags
Go to file
Add file
Code
cod3licious Merge pull request #27 from jeethu/numba-transform-jit
52 commits
autofeat
Use numba jit for feature generation
last month
.gitignore
initial commit
3 years ago
LICENSE
initial commit
3 years ago
README.md
Use numba jit for feature generation
last month
autofeat_benchmark_classification.ipy...
update sympy call to work with new version
7 months ago
autofeat_benchmark_regression.ipynb
minor updates (v1.1.1)
17 months ago
autofeat_examples.ipynb
don't generate addition/subtr features at the highest level. i.e., if...
17 months ago
featsel_benchmark_linearmodels.ipynb
further changes for version 1.0.0
17 months ago
test_autofeat.py
Fix TypeError while running tests with scikit-learn 0.24.0
6 months ago
test_featsel.py
Fix TypeError while running tests with scikit-learn 0.24.0
6 months ago
README.md
autofeat library
Linear Prediction Models with Automated Feature Engineering and Selection
This library contains the AutoFeatRegressor and AutoFeatClassifier models with a similar interface as scikit-learn models:
fit() function to fit the model parameters
predict() function to predict the target variable given the input
score() function to calculate the goodness of the fit (R^2/accuracy)
fit_transform() and transform() functions, which extend the given data by the additional features that were engineered and selected by the model
About
Linear Prediction Model with Automated Feature Engineering and Selection Capabilities
machine-learning
linear-regression
feature-selection
feature-engineering
automl
automated-machine-learning
machine-learning-models
automated-data-science
automated-feature-engineering
Readme
MIT License
Releases 3
version 0.2.1 with less overfitt...
Latest
on May 9, 2019
+ 2 releases
Packages
No packages published
Contributors 2
cod3licious franzi
jeethu Jeethu Rao

Other Resources



Free online:
http://www.feat.engineering/index.html

FEATURE SELECTION

Feature Selection

- ***Feature selection***: process of selecting a subset of features to use in model training
- Advantages?
- Two kinds of techniques
  - Filter methods — Faster
  - Wrapper methods — Better
- Not necessary for tree-based algorithms

Filter Methods

- Use stats to remove bad/useless features before model training
  - Think of it like a data cleaning step

CID	Province	Favorite Color	Height	Shoe Size	Default
1	ON	Red	152	6.0	1
2	ON	Purple	185	12.0	0
3	ON	Purple	168	10.0	0
4	ON	Blue	143	8.0	1
5	ON	White	150	8.0	1
6	ON	Green	166	8.5	0
7	ON	Green	167	8.0	0
8	ON	Gold	172	11.5	1
9	ON	Yellow	180	12.5	0
10	ON	Red	177	11.0	1

Low Variance

Low Correlation with Target

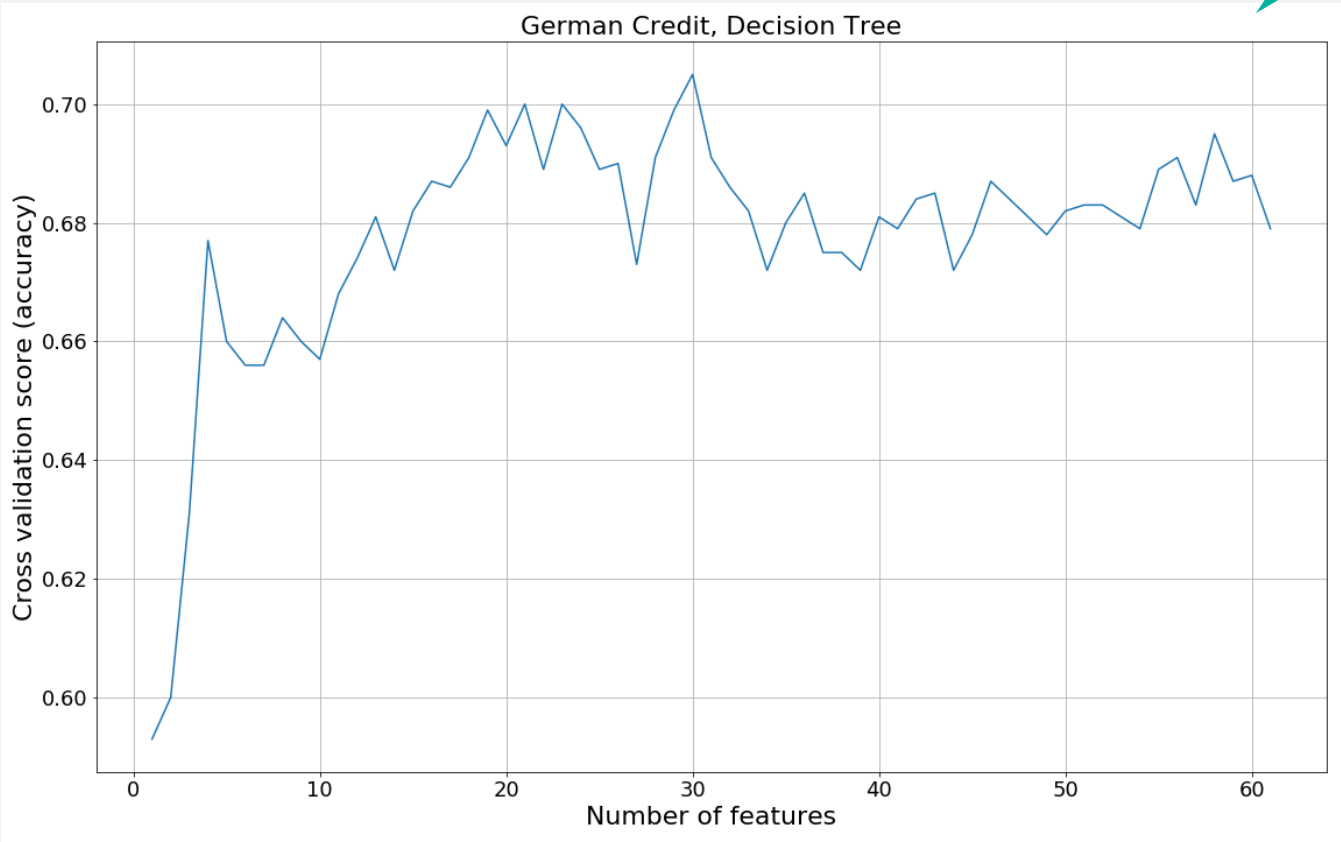
Highly Correlated

- Sklearn: VarianceThreshold()
- Sklearn: SelectKBest()
- Pandas: corr()

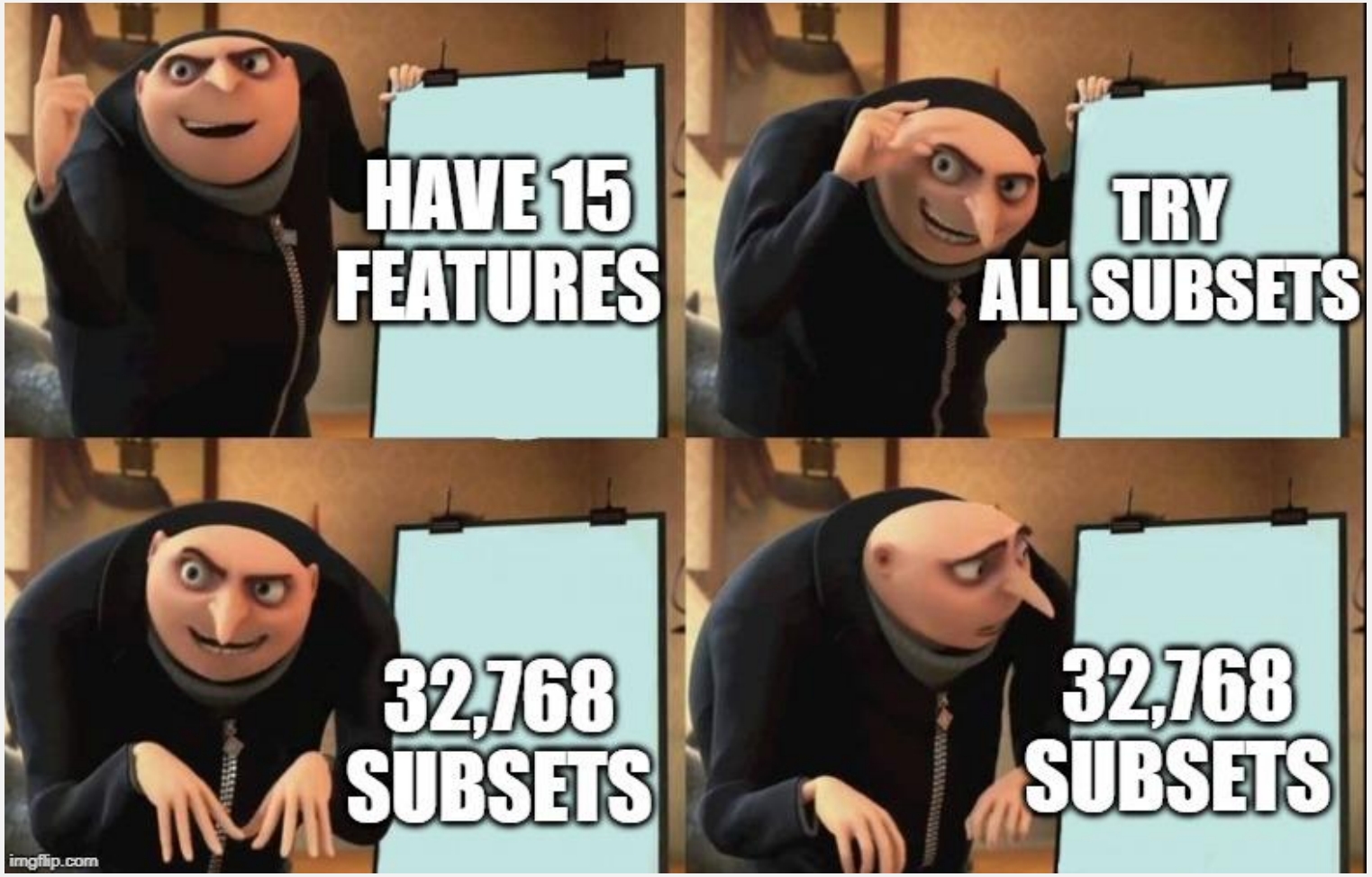
Wrapper Methods

- Find the best subset of features by training and evaluating models on many subsets of features
  - Aka *stepwise selection*
- Build model with all features; assess performance
- Remove the least-important feature, re-build and re-assess
- Repeat
- Choose best subset

Sklearn: RFE



Can't Try Every Possible Subset of Features!



- 5 features → 32 possible subsets
- 15 features → 32,768 possible subsets
- 30 features → 1,073,741,824 possible subsets
- …

HYPERPARAMETER TUNING

Motivation

Hyperparameters

```
[21]  1 clf = DecisionTreeClassifier(criterion='entropy', max_depth=5, min_samples_split=5, random_state=0)
      2
      3 scores = cross_val_score(clf, X, y, cv=10, scoring="accuracy")
      4 print("Mean Accuracy: {:.4f}".format(np.mean(scores)))

      Mean Accuracy: 0.6970
```

Good, but can we do better?

```
[22]  1 clf2 = DecisionTreeClassifier(criterion='entropy', max_depth=10, min_samples_split=5, random_state=0)
      2
      3 scores = cross_val_score(clf2, X, y, cv=10, scoring="accuracy")
      4 print("Mean Accuracy: {:.4f}".format(np.mean(scores)))

      Mean Accuracy: 0.7010
```

Cool! But can we do better?

```
[23]  1 clf3 = DecisionTreeClassifier(criterion='entropy', max_depth=10, min_samples_split=6, random_state=0)
      2
      3 scores = cross_val_score(clf3, X, y, cv=10, scoring="accuracy")
      4 print("Mean Accuracy: {:.4f}".format(np.mean(scores)))

      Mean Accuracy: 0.7040
```

Nice!

- Can keep going forever! Can we automate this?

Hyperparameter Tuning

- ***Hyperparameter tuning***: choosing the best values for hyperparameters
- ***Hyperparameters*** are algorithm's "settings" that are set by the human/data scientist before training begins
  - Not to be confused with "regular" parameters that are learned!
- Methods:
  - Grid Search, Random Search, more advanced….

Algorithm	Hyperparameters	Learned Model Parameters
Decision Tree	criterion, splitter, max_depth, min_samples_split, min_sampels_leaf, min_weight_fraction, max_features, class_weight…	
SVM	C, kernel, degree, gamma, coef0, shrinking, tol, class_weight, …	
Logistic Regression	C, penalty, tol, fit_intercept, class_weight, …	
Neural Network	hidden_layer_sizes, activation_function, alpha, batch_size, learning_rate, max_iter, tol, …	
….		

Grid Search

- Data scientist defines sets of values for each hyperparam
  - Max depth: 5, 10, 15, 20, 25, 30, 35
  - Impurity metric: info gain, gini, variance reduction
  - Min samples: 1, 10, 25, 50, 100
- Grid Search builds model for **every** combo of hyperparameter values
  - How many total combos above?
- Choose the combo that has highest/best performance metric

Example

```
[10]  1 from sklearn.model_selection import GridSearchCV
      2
      3 clf = DecisionTreeClassifier(random_state=42)
      4
      5 params = {'criterion': ('gini', 'entropy'),
      6           'splitter': ('best', 'random'),
      7           'class_weight': ('balanced', None),
      8           'max_depth': [2, 5, 10, 20],
      9           'min_samples_leaf': [1, 5, 10],
     10           'max_features':[0.25, 0.5, 0.75, 1.0]}
     11
     12 search = GridSearchCV(clf, params, scoring='f1_macro', cv=10, verbose=1)
     13 search = search.fit(X, y)
```

```
Fitting 10 folds for each of 384 candidates, totalling 3840 fits
```

```
1 search.best_params_
```

```
{'class_weight': 'balanced',
 'criterion': 'gini',
 'max_depth': 10,
 'max_features': 1.0,
 'min_samples_leaf': 1,
 'splitter': 'best'}
```

+ Code + Text

```
[12]  1 search.best_score_
```

```
0.6631045623915302
```

Table

	class_weight	criterion	max_depth	max_features	min_samples_leaf	splitter	mean_val_score	rank_val_score
66	balanced	gini	10	1.00	1	best	0.663105	1
339	None	entropy	10	0.25	5	random	0.662809	2
145	balanced	entropy	10	0.25	1	random	0.659962	3
280	None	gini	20	0.75	10	best	0.658818	4
364	None	entropy	20	0.25	10	best	0.657111	5
...	...	...	...	...	...	...	...	...
196	None	gini	2	0.25	10	best	0.411765	379
194	None	gini	2	0.25	5	best	0.411765	379
288	None	entropy	2	0.25	1	best	0.411765	379
290	None	entropy	2	0.25	5	best	0.411765	379
192	None	gini	2	0.25	1	best	0.411765	379

384 rows × 8 columns


Just need to
do GridSearch
10,000
combinations
imgflip.com

Random Search

- Data scientist defines *ranges* of values for each hyperparam
  - Max depth: [5 – 35]
  - Impurity metric: [info gain, gini, variance reduction]
  - Min samples: [1 – 100]
- Random Search randomly chooses value in range for each hyperparameter
- Does this N times
- Choose combination that has highest/best performance metric

Example

```
from sklearn.model_selection import RandomizedSearchCV
from scipy.stats import randint, uniform

clf = DecisionTreeClassifier(random_state=42)

params = {"criterion": ["gini", "entropy"],
          "splitter": ["best", "random"],
          "class_weight": ['balanced', None],
          "max_depth": randint(2, 21),
          "min_samples_leaf": randint(1, 11),
          "max_features": uniform(0.0, 1.0)}

search = RandomizedSearchCV(clf, param_distributions=params, n_iter=1000,
                            scoring='f1_macro', cv=10, verbose=1)
search = search.fit(X, y)
```

```
Fitting 10 folds for each of 1000 candidates, totalling 10000 fits
```

```
[15]  search.best_params_
```

```
{'class_weight': None,
 'criterion': 'gini',
 'max_depth': 16,
 'max_features': 0.5352240036603689,
 'min_samples_leaf': 7,
 'splitter': 'random'}
```

```
search.best_score_
```

```
0.6732480845789632
```

Table

	class_weight	criterion	max_depth	max_features	min_samples_leaf	splitter	mean_val_score	rank_val_score
374	None	gini	16	0.535224	7	random	0.673248	1
409	balanced	gini	10	0.374718	1	random	0.672304	2
828	balanced	gini	19	0.136917	8	best	0.670521	3
230	balanced	gini	15	0.544650	4	random	0.667047	4
419	None	entropy	15	0.950268	8	best	0.666441	5
...	...	...	...	...	...	...	...	...
426	None	entropy	2	0.301211	9	random	0.411765	996
773	None	entropy	2	0.318876	6	random	0.411765	996
572	None	gini	2	0.421487	6	random	0.411765	996
832	None	entropy	2	0.793482	9	random	0.411765	996
10	None	entropy	2	0.191799	7	best	0.411765	996

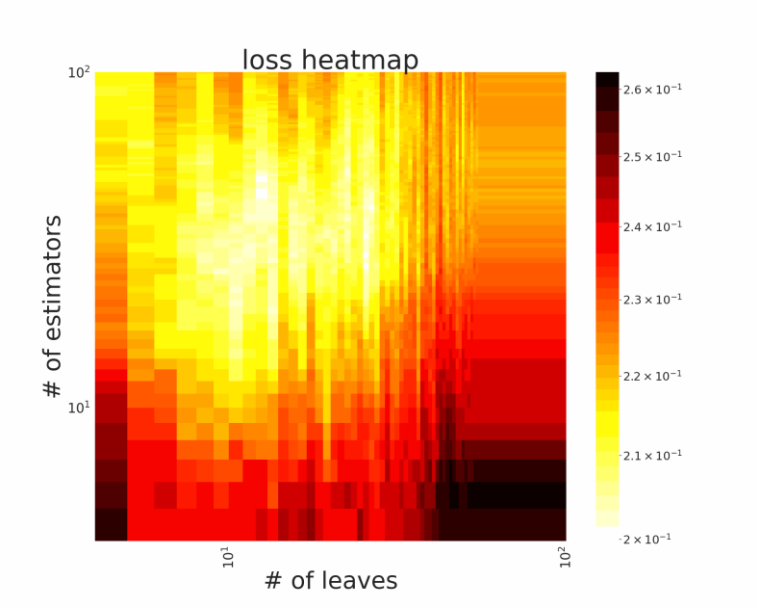
Advanced Search Algorithms

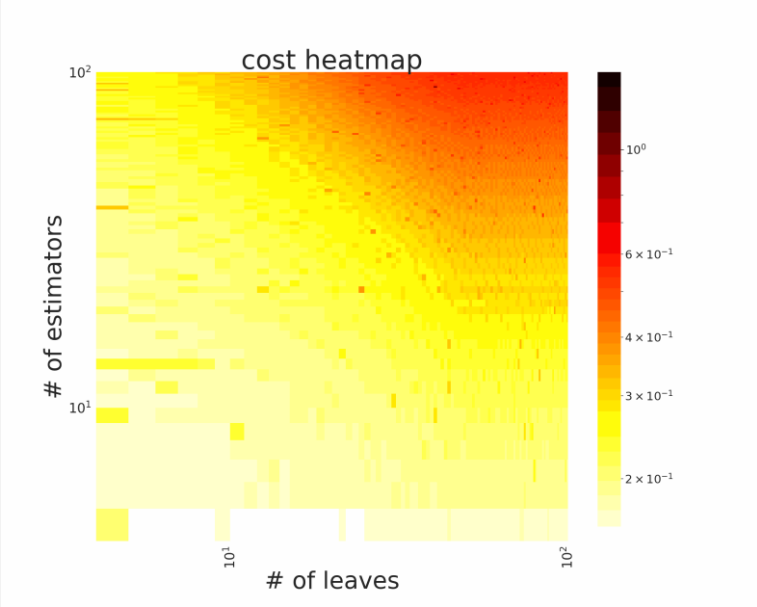
OPTUNA

RAY

FLAML

HYPEROPT



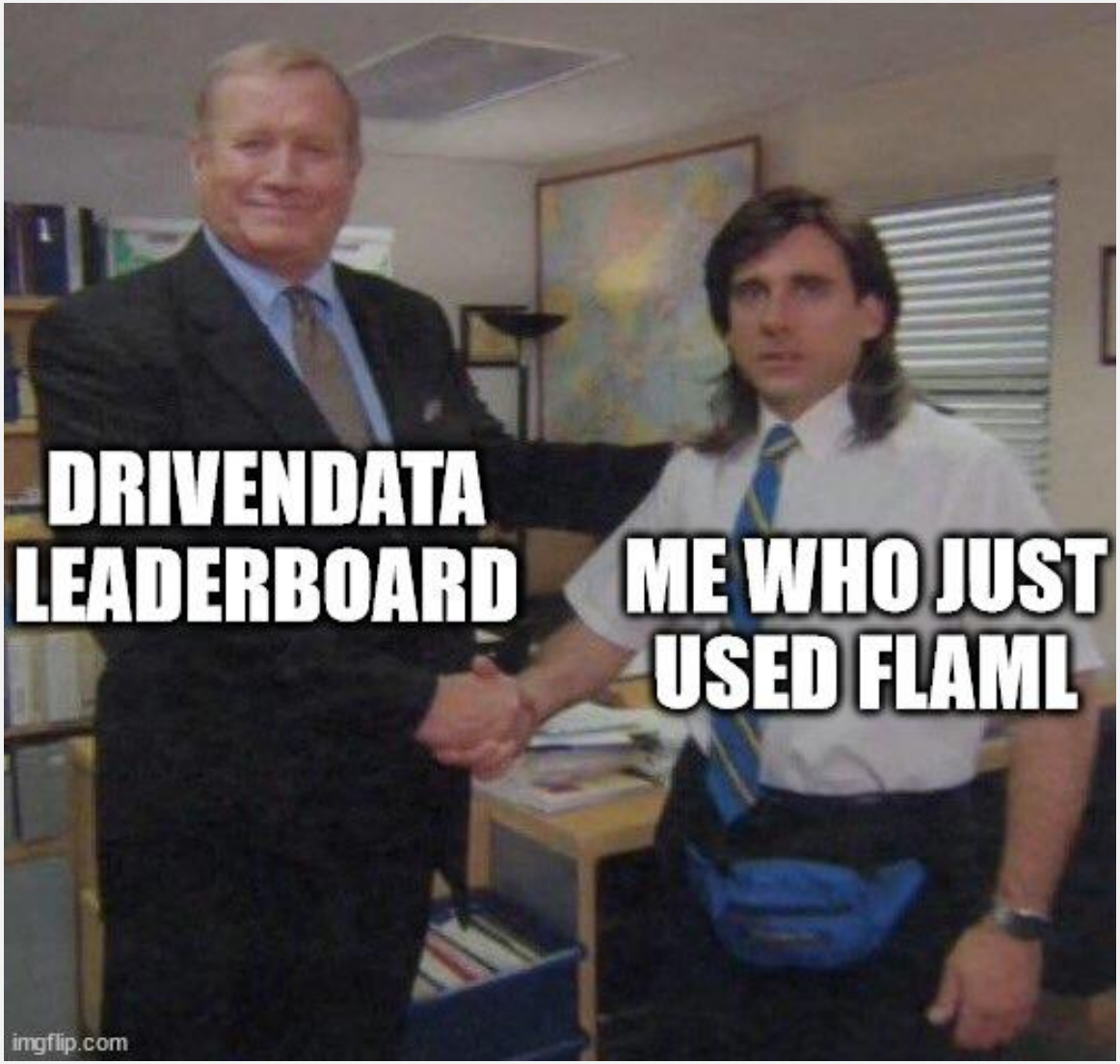


TPE, CmaEs, NSGAII, MOTPE, CFO, …

AutoML Tools

- Many open-source tools exist to help you tune

	Auto sklearn	H2o AutoML	mljar	FLAML	AutoGluon
Developer	Matthias Feurer	H2o	Piotr Płoński	Microsoft	Amazon
ML algorithms included	All sklearn algorithms	RF, Extra Trees, XGBoost, GBM, NN	KNN, DT, RF, Extra Trees, XGBoost, CatBoost, LGBM, NN	RF, Extra Trees, Catboost, XGBoost, LGBM	KNN, RF, Extra Trees, XGBoost, Catboost LGBM, NN
Auto preprocessing?	Lots	Some	Some	Some	Some
Auto FE?	Yes	No	Yes	No	No
Ensemble stacking?	Yes	Yes	Yes	Yes	Yes
Can use GPU?	No	Yes	No	Yes	Yes
Other notes	Operates entirely with sklearn		Also has "Explain" mode for auto EDA	Uses cost-effective hyper parameter search	Tech behind SageMaker; can handle images and text


DRIVENDATA
LEADERBOARD
ME WHO JUST
USED FLAML
imgflip.com

Python Tutorial

https://stream.queensu.ca/Watch/Gq8j6HKn

CLASS IMBALANCE

Imbalanced Data

- ***Imbalanced data***: data sets in which there is a disproportionate ratio of instances in one target class

			Disease
			No
			No
			No
			No
			Yes
			No
			No
			No
			No
			No

In this case:
"No" = ***majority class*** (9 instances)
"Yes" = ***minority class*** (1 instances)

- More common that you think! Examples:
  - Disease screening
  - Fraud detection
  - Spam filtering
  - Customer churn
  - Advertising click-throughs

Solutions

- **Class weighting**
- **Resample the data**
  - **Over sample** the minority class
  - **Generate** synthetic data (SMOTE, ADYSYN)
  - **Under sample** the majority class
  - Packages:
    - Python: imbalanced-learn
    - R: caret

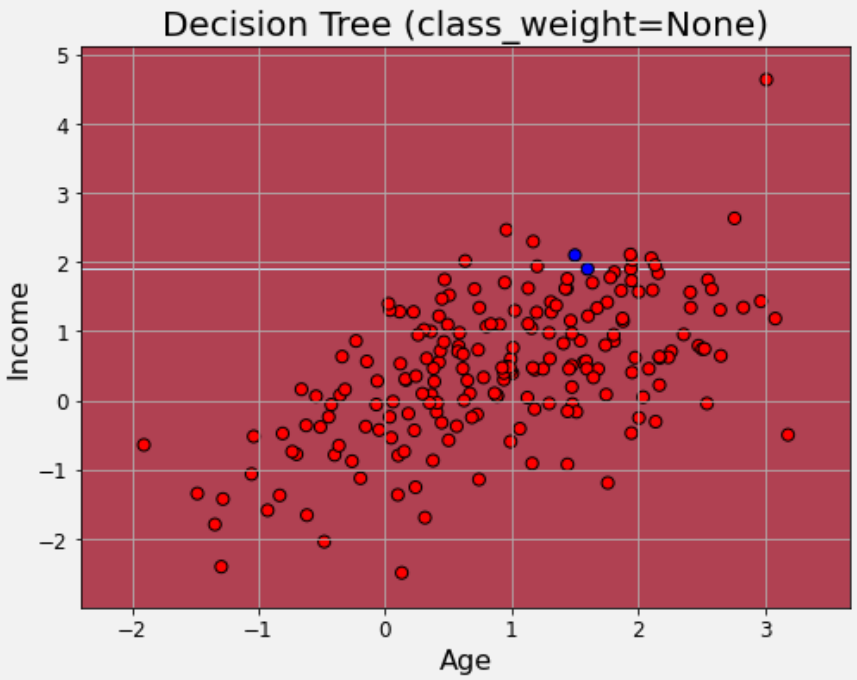
My preferred

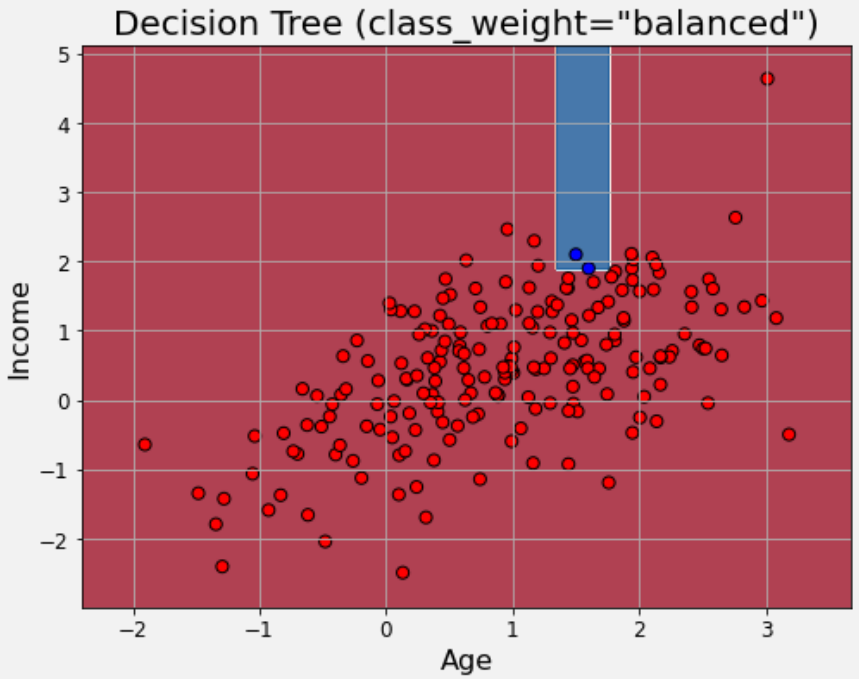
Class Weighting

- Tells algorithm to increase the cost of misclassifying the minority class

```
clf = DecisionTreeClassifier(class_weight = "balanced")
```

- Algorithm will adjust loss function appropriately
- Simple and effective!



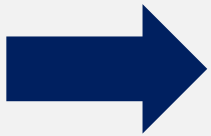


Over Sample

- Randomly duplicate instances from minority class

ID			Target
1			No
2			No
3			No
4			Yes
5			No

80% "No"
20% "Yes"

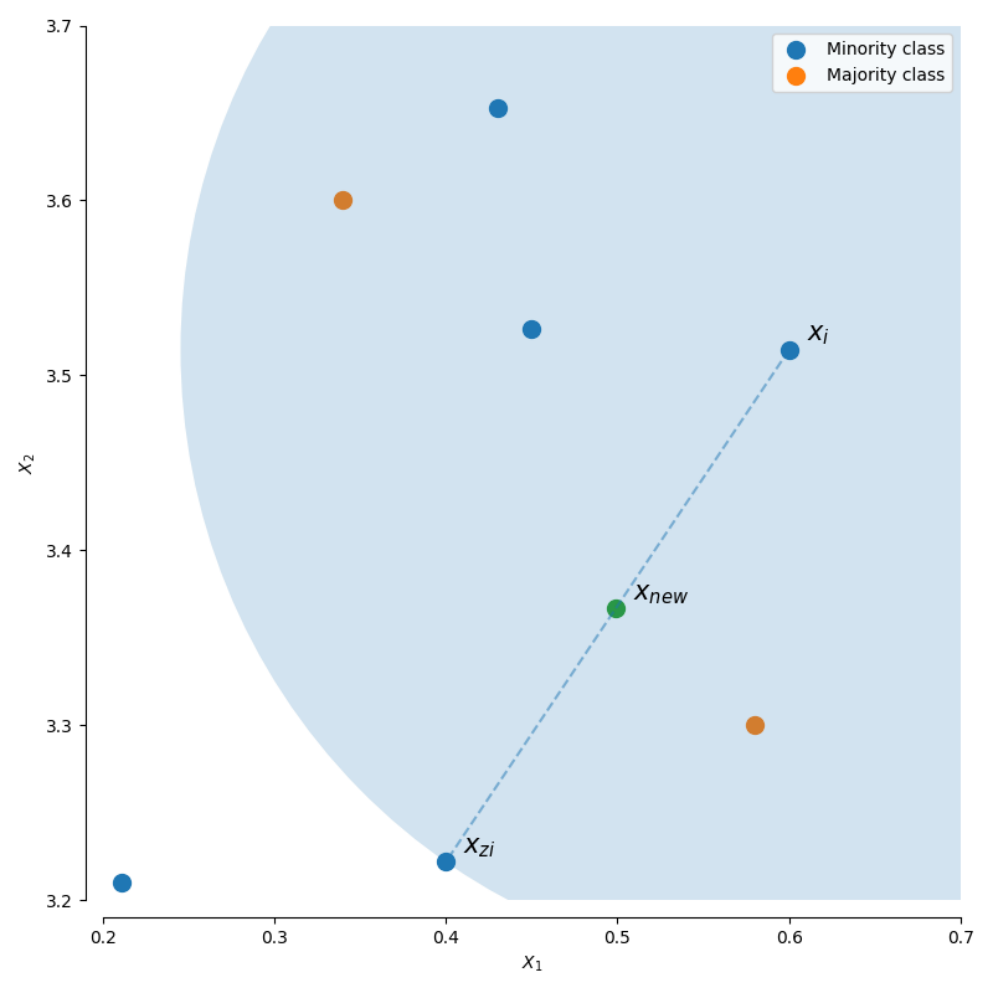


ID			Target
1			No
2			No
3			No
4			Yes
5			No
4			Yes
4			Yes
4			Yes

50% "No"
50% "Yes"

Generate Synthetic Data

- Similar to over sampling, but instead generates artificial data that "is close to" an instance in the minority class
- Common methods: SMOTE, ROSE, ADASYN



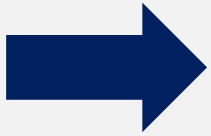
Continuous – interpolation
Categorical – majority level

Under Sample

Randomly remove instances from majority class

ID			Target
1			No
2			No
3			No
4			Yes
5			No
6			Yes

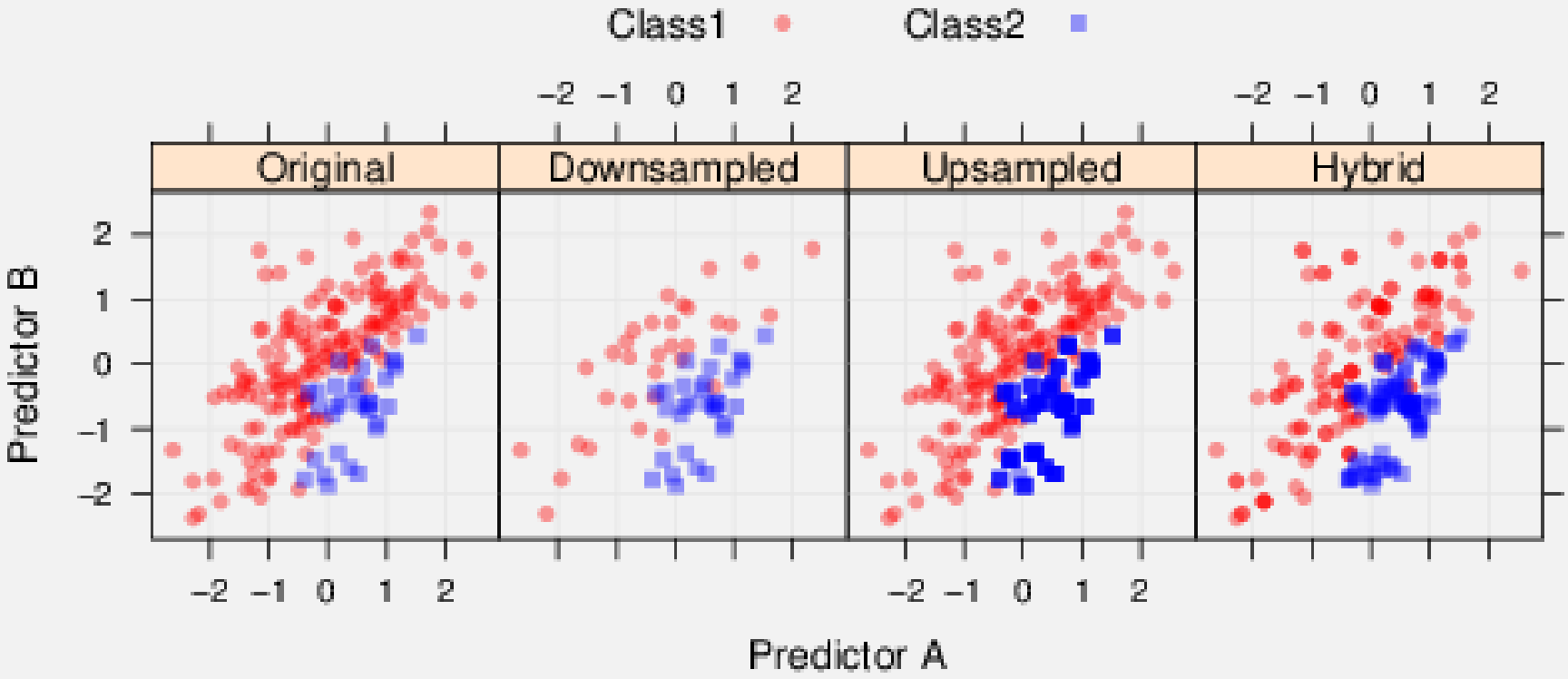
66% "No"
33% "Yes"



ID			Target
1			No
4			Yes
5			No
6			Yes

50% "No"
50% "Yes"

Another Example



Kuhn et al. Figure 16.3.

Exercise

Over sampling *versus* under sampling: when to use which?

Example: Diabetes

[16]:

	Id	num_times_pregnant	plasma_glucose	DBP	triceps_skin	serum_insulin	BMI	pedigree	age	diabetes
0	1	6	148	72	35	0	33.6	0.627	50	1
1	2	1	85	66	29	0	26.6	0.351	31	0
2	3	8	183	64	0	0	23.3	0.672	32	1
3	4	1	89	66	23	94	28.1	0.167	21	0
4	5	0	137	40	35	168	43.1	2.288	33	1

[17]:
```
y_train.value_counts()
y_test.value_counts()
```

[17]:
```
0    400
1    214
Name: diabetes, dtype: int64
```

[17]:
```
0    100
1     54
Name: diabetes, dtype: int64
```

Comparison on Diabetes Dataset

	Method	Neg	True Neg	False Neg	Pos	TP	FP	Accuracy	Recall	Precision	F1	AUC
3	Over SMOTE	105	84	21	49	33	16	0.759740	0.611111	0.673469	0.640777	0.725556
1	Class Weights	117	92	25	37	29	8	0.785714	0.537037	0.783784	0.637363	0.728519
4	Over ADASYN	99	79	20	55	34	21	0.733766	0.629630	0.618182	0.623853	0.709815
2	Over Random	110	86	24	44	30	14	0.753247	0.555556	0.681818	0.612245	0.707778
5	Under Sample	92	73	19	62	35	27	0.701299	0.648148	0.564516	0.603448	0.689074
0	None	116	88	28	38	26	12	0.740260	0.481481	0.684211	0.565217	0.680741

DATA LEAKAGE

Data Leakage

- When you cheat (by accident)
- When training features **contain information they shouldn't**
- Leads to "good" performance in lab, but poor performance in production
- Causes you to select the wrong model
- Examples:
  - ***Target leakage***
  - ***Bad splitting***
  - ***FE before splitting***
  - ***Duplicate instances***
  - ***Groups***

If your results seem too good to be true, then they probably are!

Target Leakage

- Feature determines/interacts with the target

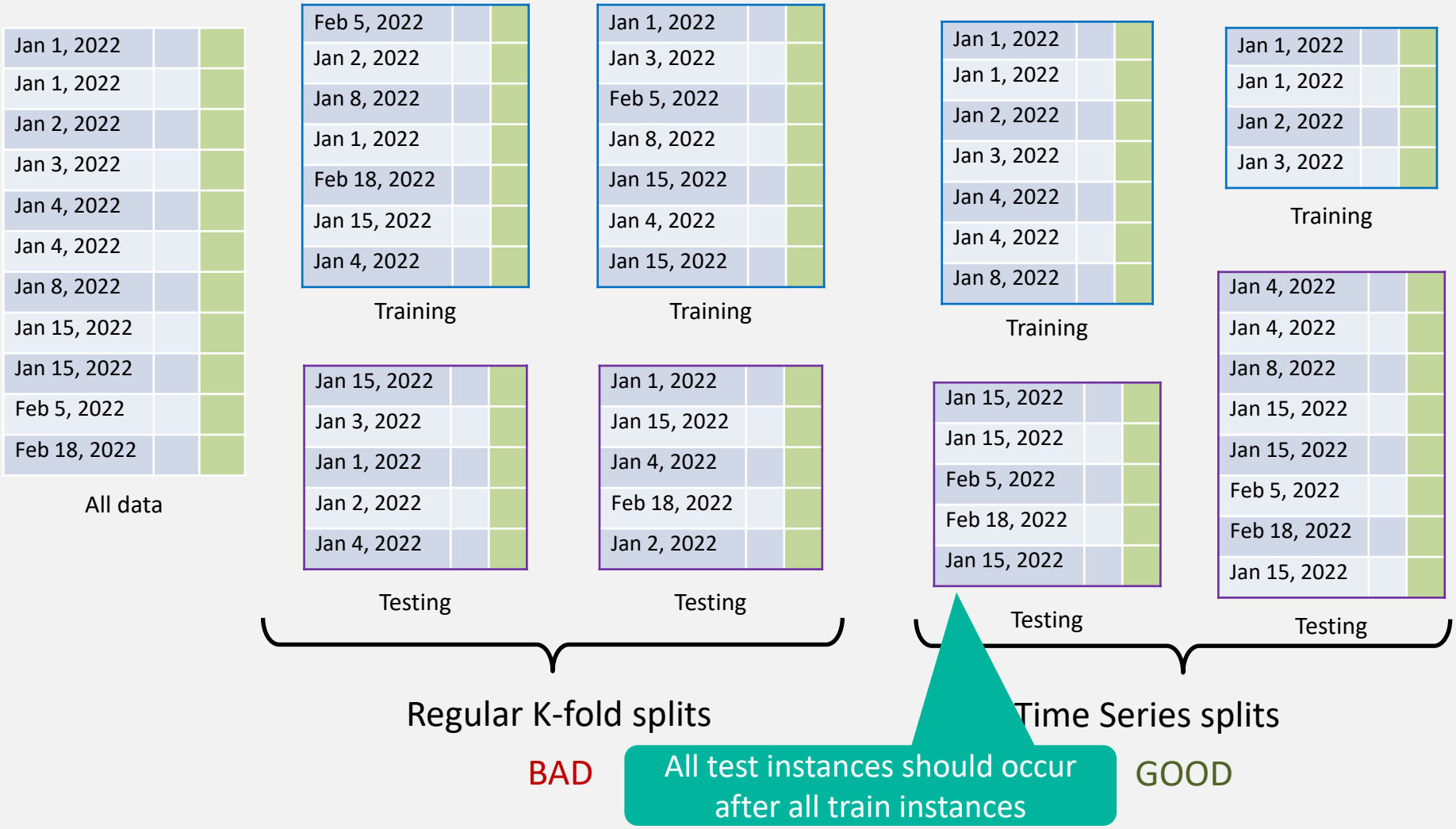
PID	Date of Death	…	Died
1	10/02/2003		Yes
2	-		No
3	3/19/2019		Yes
4	-		No
5	03/03/2013		Yes
6	-		No
7	-		No
8	-		No
9	01/02/1987		Yes
10	-		No

PID	Monthly Salary	…	Yearly Salary
1	$4,000		$48,000
2	…		…
3			
4			
5			
6			
7			
8			
9			
10			

Target Leakage

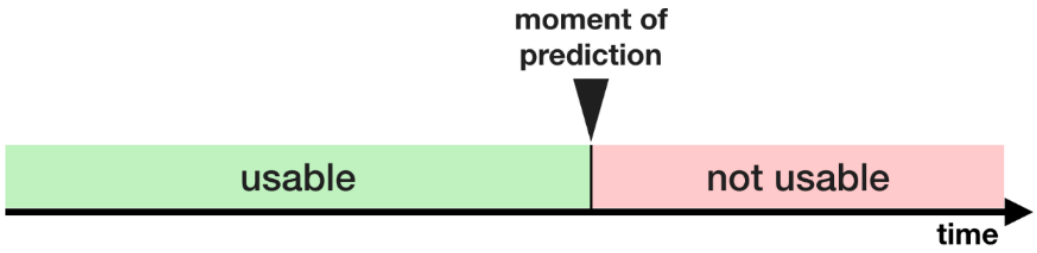
Bad Splitting

- Splitting non-i.i.d. randomly
- E.g., time-correlated data



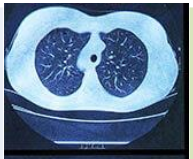
Bad Splitting

- The right way to think about time-data:



Bad Splitting

- E.g., group-correlated data

Image ID	Patient ID	Image	Cancer?
1	1		Yes
2	1		Yes
3	2		No
4	2		No
5	3		No
6	4		Yes

All instances for a given patient should be in the same split

FE Before Splitting

- Training data is influenced by testing data

Example: Scaling before Splitting

```python
from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()
scaler=scaler.fit(X[['Salary']])

X['Salary_Std'] = scaler.transform(X[['Salary']])
X[['ID', 'Salary', 'Salary_Std']]

# Now  split ...
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

PID	Salary
1	1169.0
2	5951.0
3	2096.0
4	7882.0
5	4870.0
6	9055.0
7	2835.0
8	6948.0
9	3059.0
10	5234.0
11	6854.0
12	2255.0
13	7965.0
14	7365.0
15	2006.0
16	10365.0
17	1006.0
18	4532.0
19	6395.0
20	7551.0

$$X_{new} = \frac{X_{old} - \mu}{\sigma}$$

μ =5269.6

σ = 2714.4

PID	Salary_Std
1	-1.511
2	0.251
3	-1.169
4	0.962
5	-0.147
6	1.395
7	-0.897
8	0.618
9	-0.814
10	-0.013
11	0.584
12	-1.111
13	0.993
14	0.772
15	-1.202
16	1.877
17	-1.571
18	-0.272
19	0.415
20	0.840

PID	Salary_Std
1	-1.511
2	0.251
3	-1.169
4	0.962
5	-0.147
6	1.395
7	-0.897
8	0.618
9	-0.814
10	-0.013
11	0.584
12	-1.111
13	0.993
14	0.772
15	-1.202

Training

PID	Salary_Std
16	1.877
17	-1.571
18	-0.272
19	0.415
20	0.840

Testing

The Proper Way

```
# First  split ...
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

scaler = StandardScaler()
scaler=scaler.fit(X_train[['Salary']])

X_train['Salary_Std'] = scaler.transform(X_train[['Salary']])
X_test['Salary_Std'] = scaler.transform(X_test[['Salary']])
```

PID	Salary
1	1169.0
2	5951.0
3	2096.0
4	7882.0
5	4870.0
6	9055.0
7	2835.0
8	6948.0
9	3059.0
10	5234.0
11	6854.0
12	2255.0
13	7965.0
14	7365.0
15	2006.0
16	10365.0
17	1006.0
18	4532.0
19	6395.0
20	7551.0

PID	Salary_Std
1	1169.0
2	5951.0
3	2096.0
4	7882.0
5	4870.0
6	9055.0
7	2835.0
8	6948.0
9	3059.0
10	5234.0
11	6854.0
12	2255.0
13	7965.0
14	7365.0
15	2006.0

Training

$$X_{new} = \frac{X_{old} - \mu}{\sigma}$$

μ =5036.3

σ = 2521.4

PID	Salary_Std
1	-1.534
2	0.363
3	-1.166
4	1.128
5	-0.066
6	1.594
7	-0.873
8	0.758
9	-0.784
10	0.078
11	0.721
12	-1.103
13	1.161
14	0.923
15	-1.202

Training

PID	Salary_Std
16	10365.0
17	1006.0
18	4532.0
19	6395.0
20	7551.0

Testing

PID	Salary_Std
16	2.113
17	-1.598
18	-0.200
19	0.539
20	0.997

Testing

Comparison

Premature

PID	Salary_Std
1	-1.511
2	0.251
3	-1.169
4	0.962
5	-0.147
6	1.395
7	-0.897
8	0.618
9	-0.814
10	-0.013
11	0.584
12	-1.111
13	0.993
14	0.772
15	-1.202

Training

PID	Salary_Std
16	1.877
17	-1.571
18	-0.272
19	0.415
20	0.840

Testing

Proper

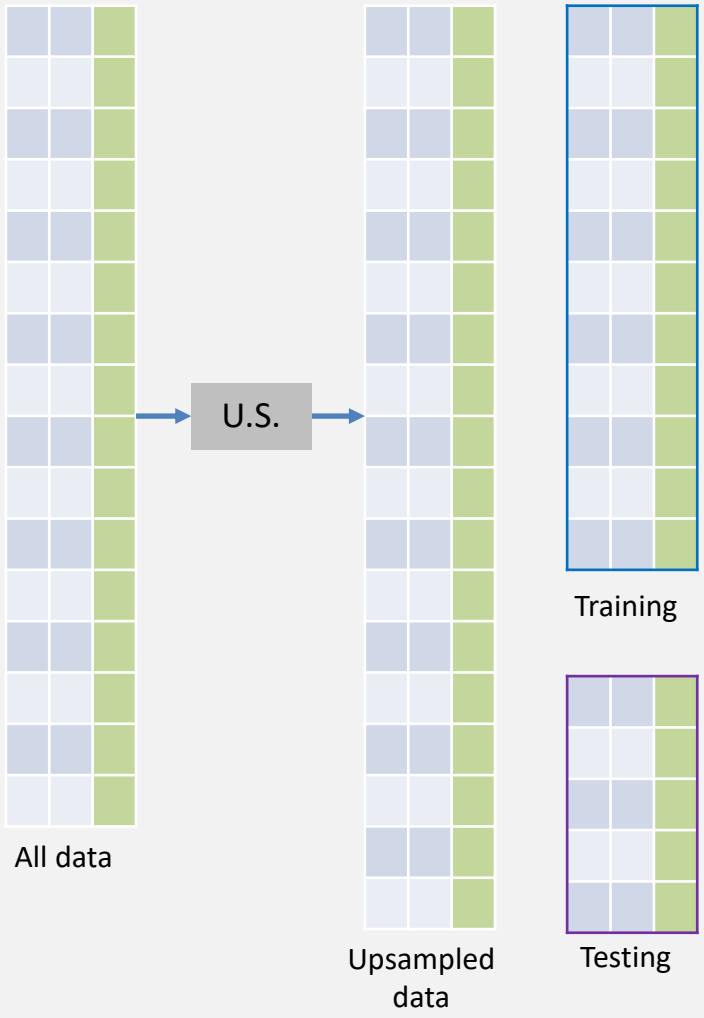
PID	Salary_Std
1	-1.534
2	0.363
3	-1.166
4	1.128
5	-0.066
6	1.594
7	-0.873
8	0.758
9	-0.784
10	0.078
11	0.721
12	-1.103
13	1.161
14	0.923
15	-1.202

Training

PID	Salary_Std
16	2.113
17	-1.598
18	-0.200
19	0.539
20	0.997

Testing

Example: Upsampling before Splitting



...

BAD! Why?

Avoiding and Detecting Leakage

- Train/fit feature transformations on the training ONLY
  - Not on validation/test
- Check feature importances
  - Do top features make sense?
- Do an ablation study
- Understand the meaning of every feature
  - For every feature, ask: "Will I have this feature at prediction time? What values can it have?"

Source: Target Leakage in ML, Yuriy Guts, 2018

PIPELINES

Motivation

```
from sklearn.tree import DecisionTreeClassifier
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA
from sklearn.impute import SimpleImputer
from sklearn.svm import SVR
from sklearn.feature_selection import RFE

imp = SimpleImputer()
scaler = StandardScaler()
pca = PCA(n_components=3)
rfe = RFE(SVR(kernel="linear"))
clf = DecisionTreeClassifier(random_state=42)

imp = imp.fit(X_train)
X_train2 = imp.transform(X_train)

scaler = scaler.fit(X_train2)
X_train3 = scaler.transform(X_train2)

pca = pca.fit(X_train3)
X_train4 = pca.transform(X_train3)

rfe = rfe.fit(X_train4, y_train)
X_train5 = rfe.transform(X_train4)

clf = clf.fit(X_train5, y_train)

X_test2 = imp.transform(X_test)
X_test3 = scaler.transform(X_test2)
X_test4 = pca.transform(X_test3)
X_test5 = rfe.transform(X_test4)

clf.predict(X_test5)

# ....
```

Must manually chain together transformations

Must do all the same transforms to test data

This doesn't even do cross validation; would have to put this al in a loop

```
from sklearn.tree import DecisionTreeClassifier
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA
from sklearn.impute import SimpleImputer
from sklearn.svm import SVR
from sklearn.feature_selection import RFE
from sklearn.pipeline import Pipeline, make_pipeline

pipe = make_pipeline(SimpleImputer(),
                     StandardScaler(),
                     PCA(n_components=3),
                     RFE(SVR(kernel="linear")),
                     DecisionTreeClassifier(random_state=42))

pipe.fit(X_train, y_train)
pipe.predict(X_test)
```

predict() calls all Transformers and Estimators properly

fit() calls all Transformers and Estimators properly

Pipelines: The Easier Way

- Way to put all FE, FS, R.S., etc. steps together in one object
- Can easily fit/train transformers on just training data
- Can re-use pipeline easily on new test data; all steps will be applied

[7]:
```
from sklearn.pipeline import Pipeline, make_pipeline
from sklearn.tree import DecisionTreeClassifier
from sklearn.decomposition import PCA
from sklearn.preprocessing import StandardScaler
from sklearn.feature_selection import RFE
from sklearn.model_selection import GridSearchCV
from sklearn.svm import SVR

pipe1 = make_pipeline(StandardScaler(),
                      PCA(n_components=10),
                      RFE(SVR(kernel="linear")),
                      DecisionTreeClassifier(random_state=223))

param_grid = {
    'standardscaler__with_mean': [True, False],
    'pca__n_components': [5, 10, 20],
    'rfe__n_features_to_select': [None, 10, 20],
    'decisiontreeclassifier__max_depth': [None, 3, 10],
    'decisiontreeclassifier__criterion': ['gini', 'entropy'],
    'decisiontreeclassifier__class_weight':[None, 'balanced'],
}

search = GridSearchCV(pipe1, param_grid,
                      cv=3, n_jobs=5, scoring='f1_micro', return_train_score=True, verbose=2)
```

[8]:
```
search = search.fit(X_train, y_train)
```

Feature Engineering

Feature Selection

Hyper Parameter Tuning

Cross Validation

Pipelines

Smith SCHOOL OF BUSINESS Queen's University

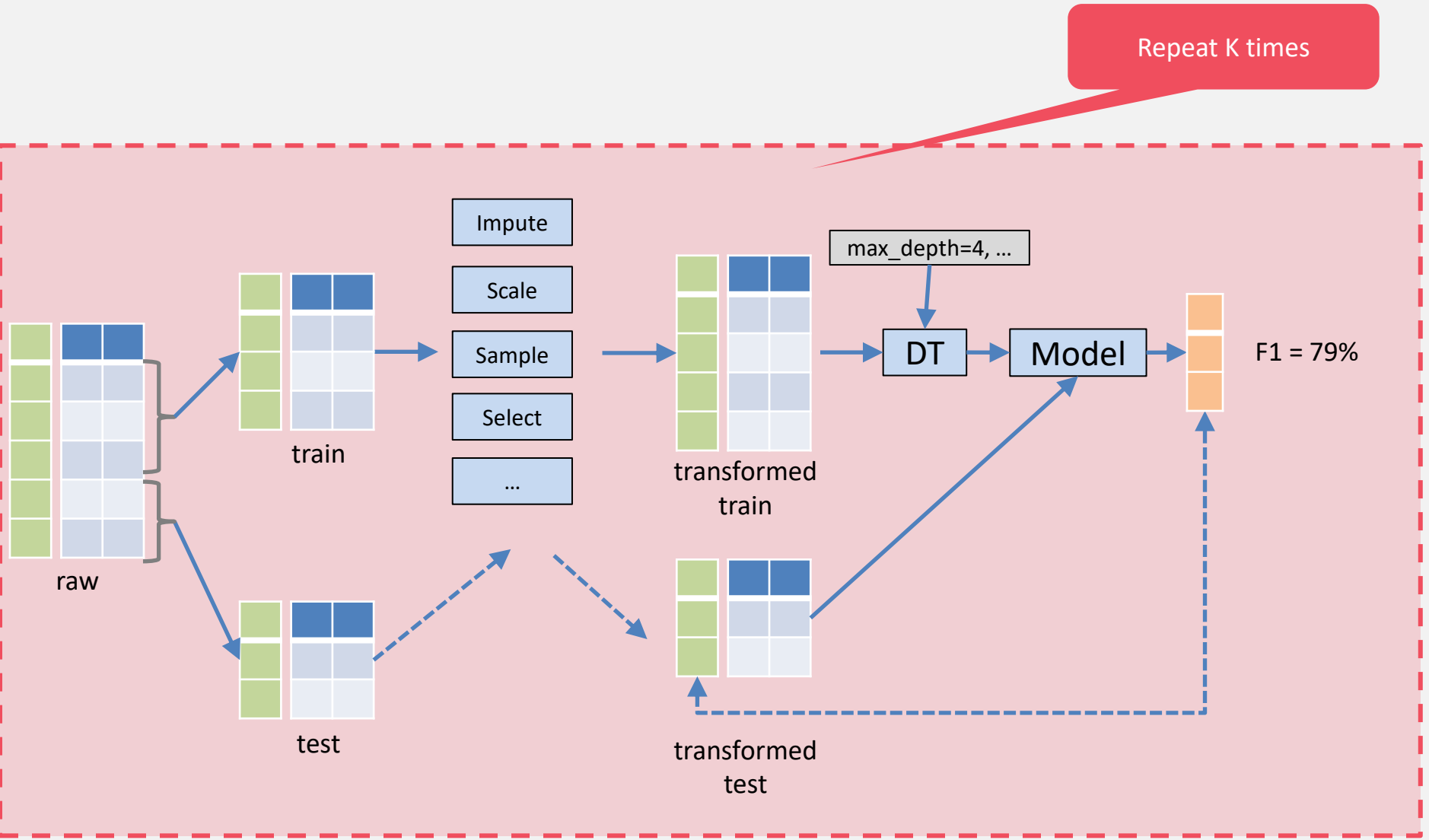
[9]:
```
# Predict testing data by just calling the pipe!
search.predict(X_test)
```

[9]:
```
array([1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 0, 1, 1, 1, 1, 0, 0, 1, 1, 0, 1, 0, 1, 0, 1, 1, 0, 1,
       1, 1, 0, 1, 1, 0, 0, 1, 1, 1, 1, 1, 1, 0, 0, 1, 0, 1, 1, 1, 0, 1,
       1, 1, 0, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0,
       1, 1, 0, 1, 0, 0, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 0, 0, 0,
       0, 1, 0, 1, 1, 0, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 0,
       1, 0, 0, 1, 1, 1, 1, 0, 0, 0, 0, 1, 1, 1, 1, 0, 1, 1, 1, 0, 1, 0,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1,
       0, 1], dtype=int64)
```

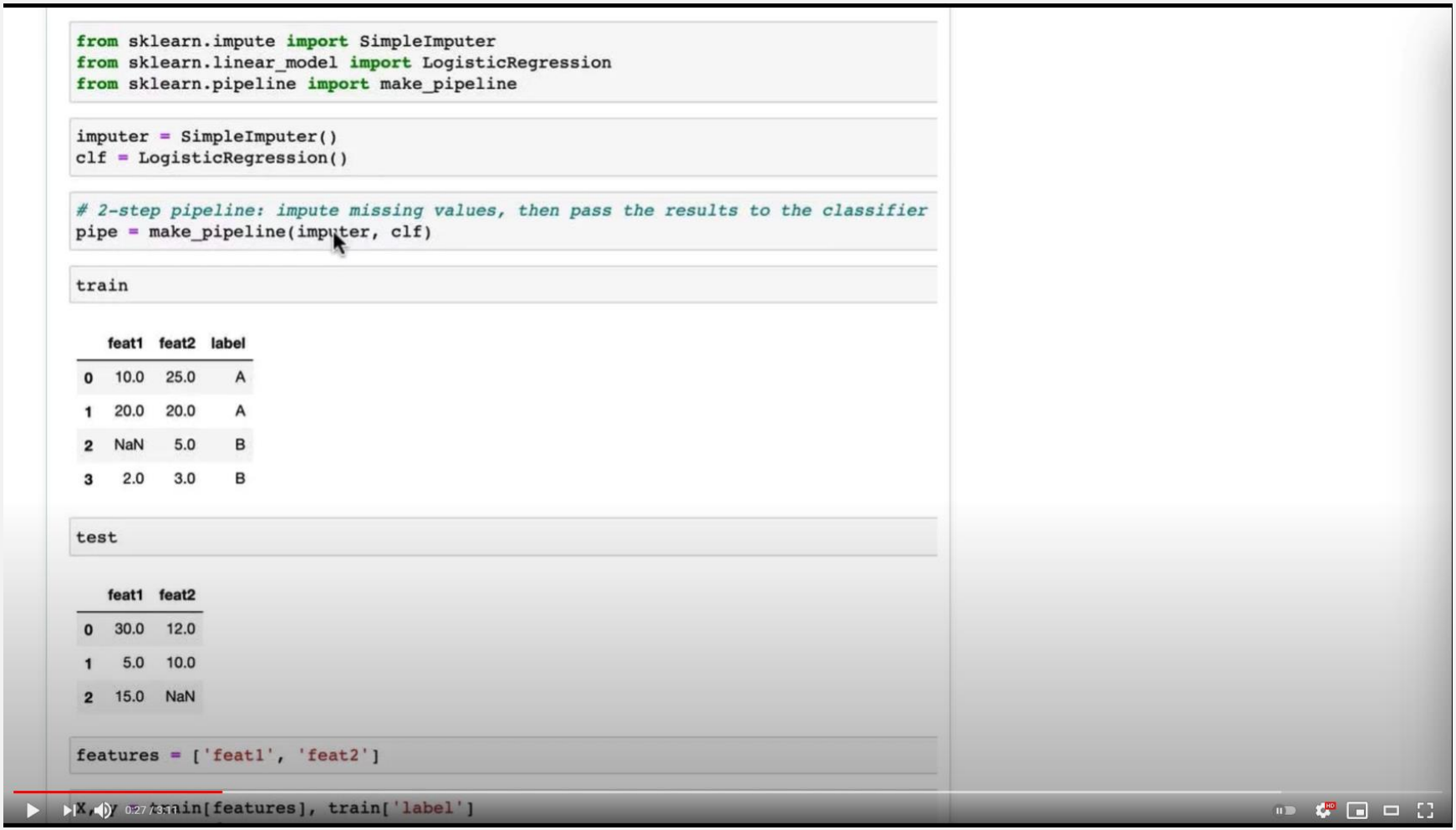
[10]:
```
search.score(X_test, y_test)
```

[10]:
```
0.67
```

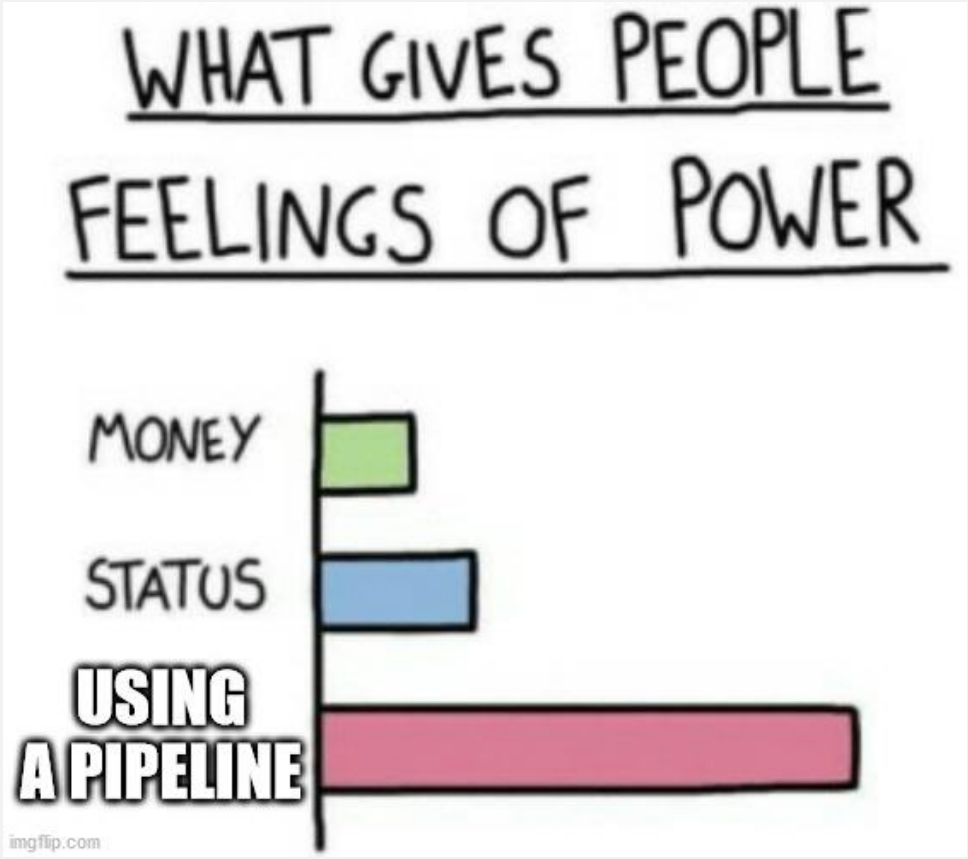
Pipelines Do It all Under the Hood!



Quick Tutorial by Data School



[Use Pipeline to chain together multiple steps](#) (3:11)



Note:

- Pipelines can be hard to learn
- Not strictly necessary for this course

SUMMARY

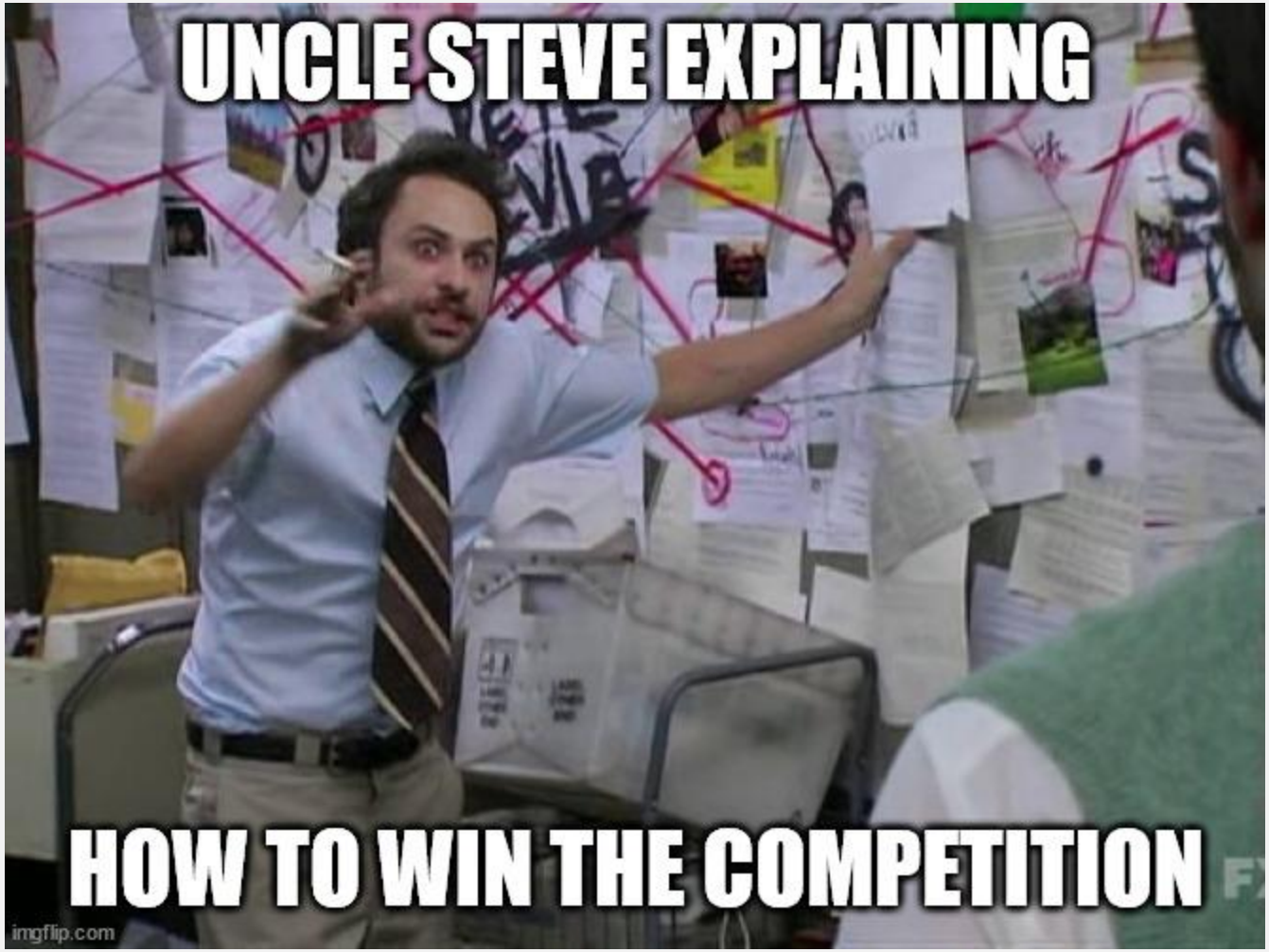
Exercise: Which Leads to Which?

Smith SCHOOL OF BUSINESS Queen's University

ML Task	Benefits
• Prevent data leakage • Feature engineering • Feature selection • Hyperparam tuning • Fixing class imbalance • Using pipelines • Cross validation	• Stop model overfitting • Stop model underfitting • Faster training time • Increase model performance • Easier to prep testing/future data • Easier coding • Obtain accurate estimate of model performance

Summary

- ***Feature engineering***: creating newer, more awesome features
  - Transformations: scaling, binning, encoding, dim reduction, …
  - **If you don't do it**: your model won't have the best information, and it won't achieve as good performance
- ***Feature selection***: selecting only the best features
  - Filter, wrapper
  - **If you don't do it**: extra features laying around, may hurt performance
- ***Hyperparameter tuning***: finding the best values for hyperparameters
  - Grid search, random search
  - **If you don't do it**: Probably leaving a lot of performance on the table
- ***Class imbalance***: when one target class is dominant
  - Class weights or resampling
  - **If you don't do it**: model will over-predict majority class, hurting performance
- ***Data leakage***: when training data has info it couldn't/shouldn't
  - Target leakage, train-test contamination
  - **If you don't control it**: your model will likely overfit, and you will think your model is better than it really is

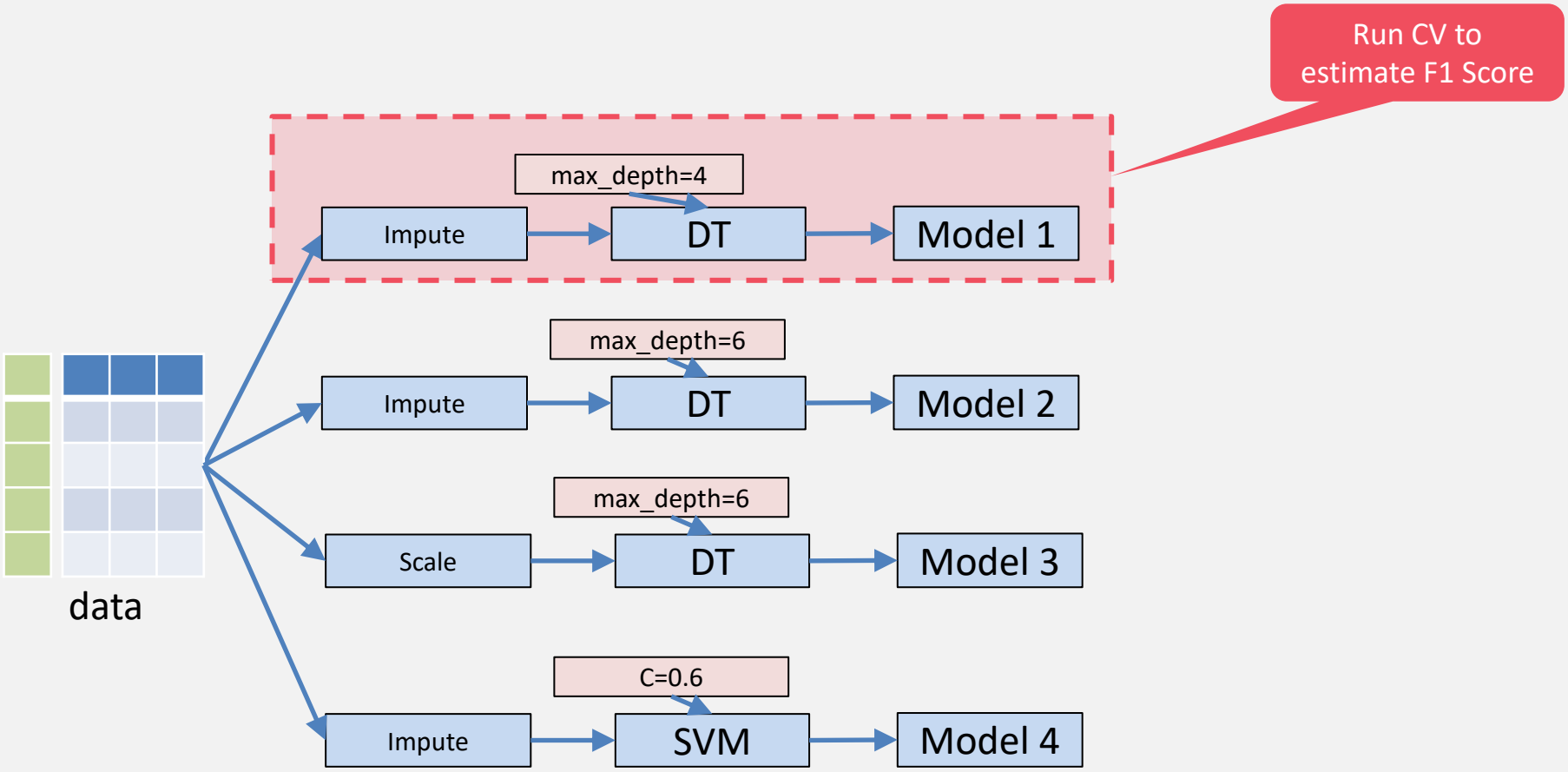

UNCLE STEVE EXPLAINING
HOW TO WIN THE COMPETITION
imgflip.com

APPENDIX

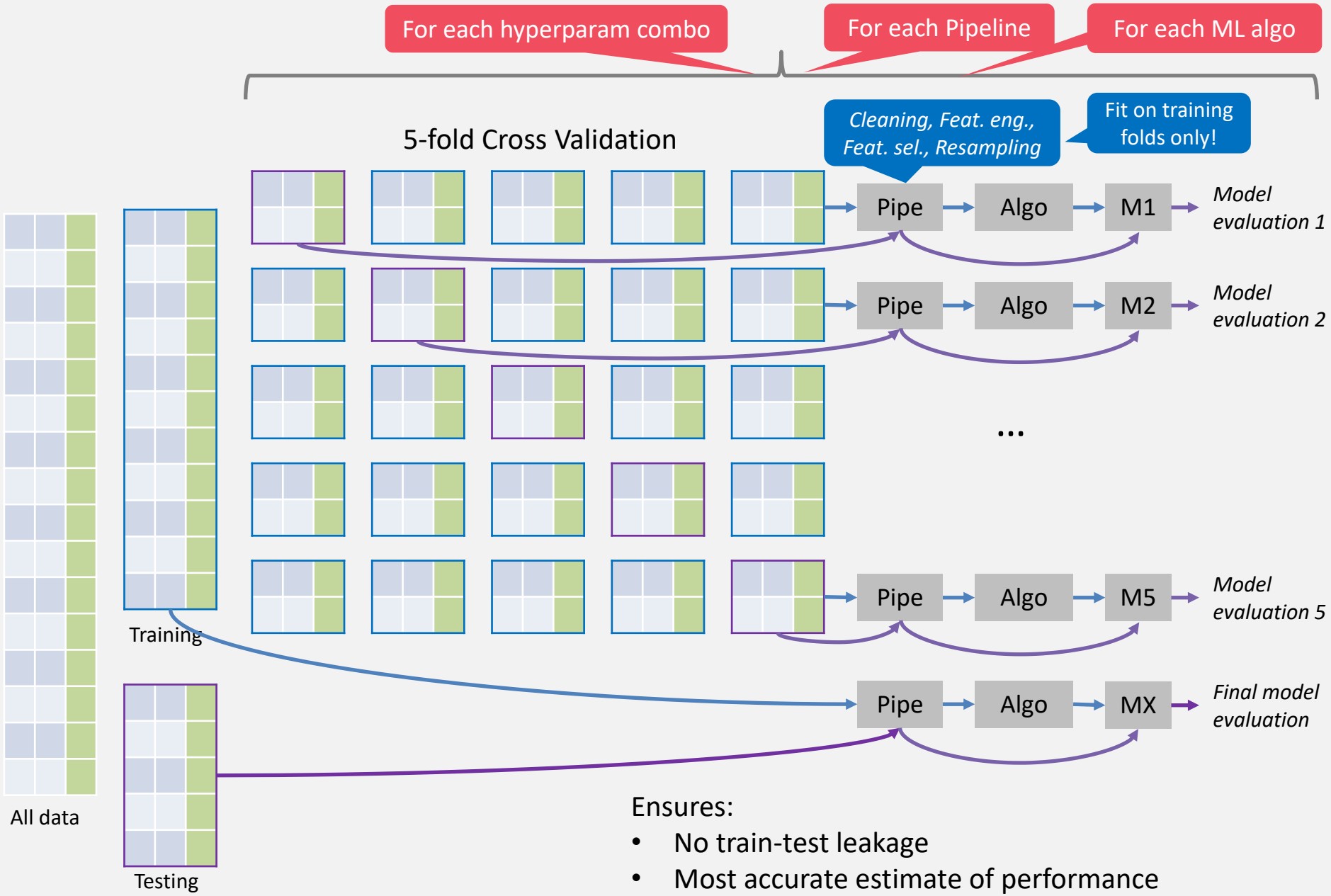
Two Main Goals

1. Make the features consistent with model's assumption
   - Scales
   - Skews
   - Collinearity
   - Normality
   - ….
2. Give the model newer and better features to work with
   - Linear and non-linear combinations of existing features
   - Transactional data aggregation
   - Time series features
   - Date features
   - Image features
   - Text features
   - Audio features
   - Video features
   - …

Big Picture: Conceptually



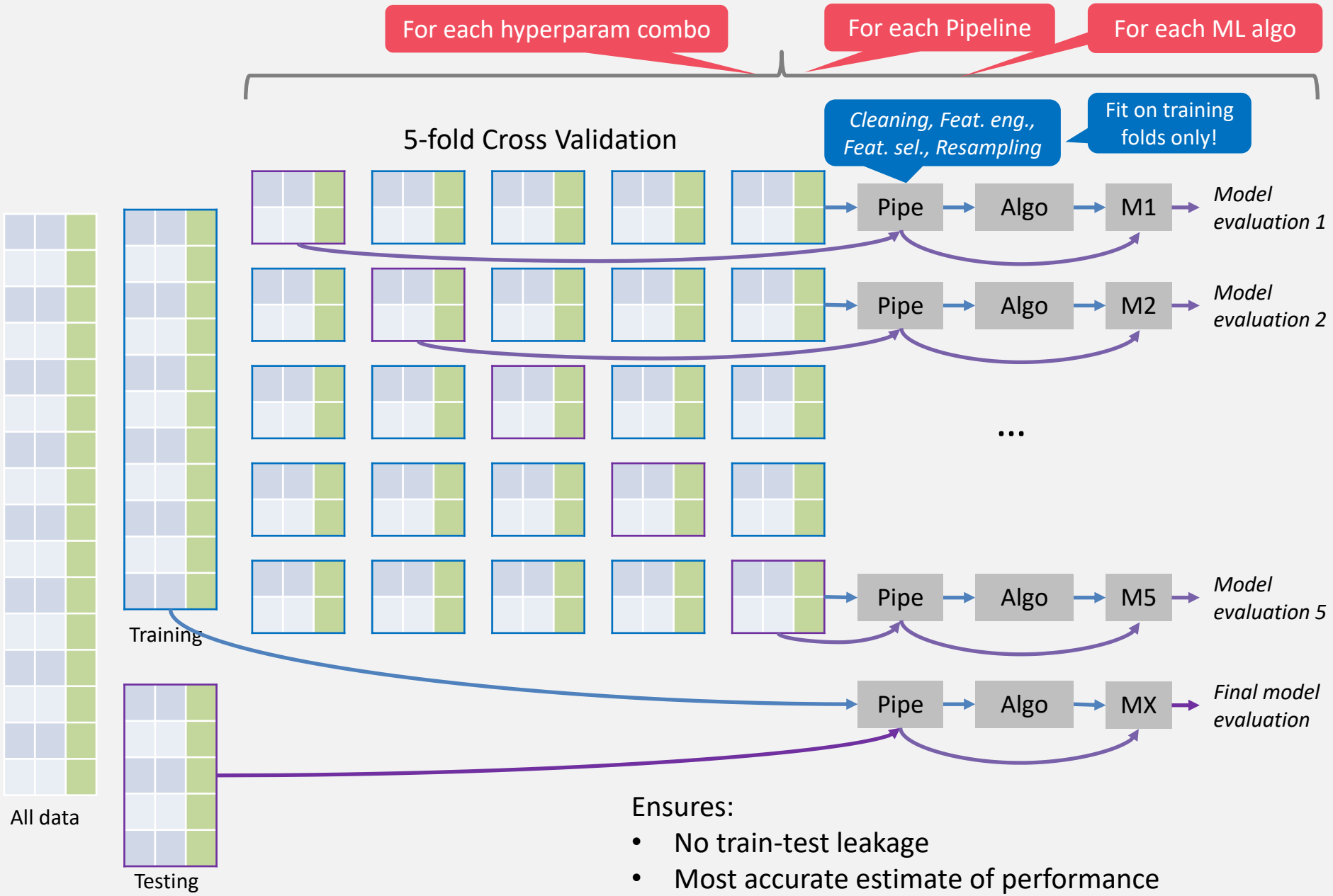
Uncle Steve's Ultimate Training Guide



Ensures:

- No train-test leakage
- Most accurate estimate of performance

Uncle Steve's Guide to Success



Ensures:

- No train-test leakage
- Most accurate estimate of performance

Uncle Steve's Ultimate Training Guide

1. Split data into **training** and **testing**
2. Split **training** into 5 splits of 5 folds each
3. For each of the 5 **splits**:
    1. On the four **training folds**
        1. Feature engineering
        2. Feature selection
        3. Over/under sample
        4. Train model **M`** on transformed **training folds**

    pipeline

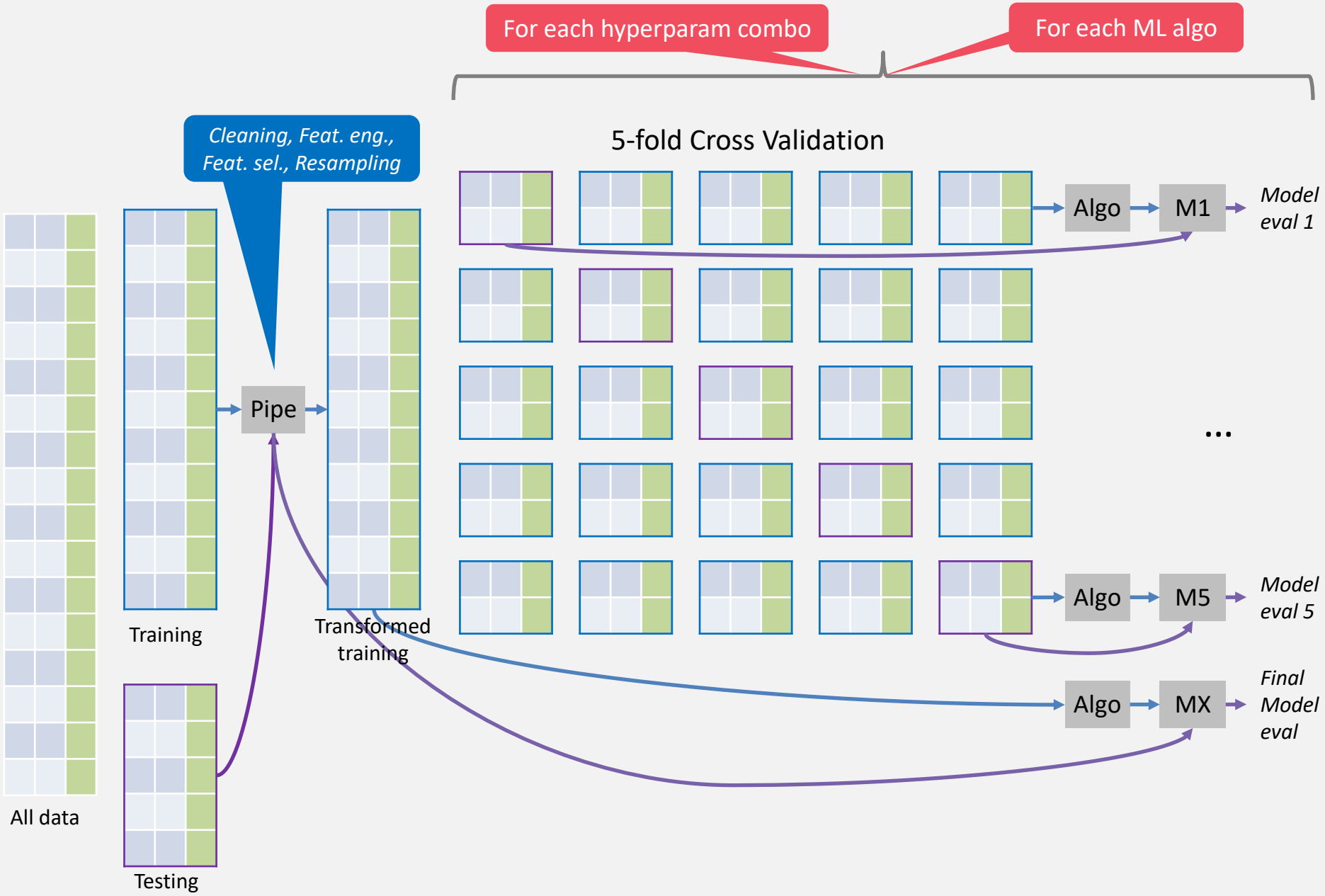
    2. On the **validation** fold:
        1. Transform **validation** data using **pipeline**
        2. Predict **validation** using **M`** and evaluate
4. Combine evaluations from all 5 splits to get mean estimate
5. Decide best pipeline, ML algo, and hyperparam combo
6. On all **training** data**:**
    1. Transform **training** data using **pipeline**
    2. Train model **M**
7. On **testing** data:
    1. Transform **testing** data using **pipeline**
    2. Predict **testing** from **M** and evaluate

For each pipeline

For each hyperparam combo

For each ML algo

Shortcuts Sometimes Necessary/OK



Uncle Steve's Big List of FE Tasks

Transformations

- Scaling
  - Log, standardize, minmax, BoxCox, …
- Discretization
- Linear and non-linear combos
- Encoding
  - OHE, ordinal, target, …
- Dimensionality reduction
  - PCA, SVD, MDS, Factor analysis, …
- Dates/times
- Time series
  - Peaks, mean, max, min, entropy, …
- Text
  - BOW, topic models, word2vec, …
- Images/Video
  - Edges, interest points, corners, …
- …

Aggregations

- Min, max, mean, sum, std, …
- Count, number unique, …
- Time since last, time since first, …
- PercentTrue, Percent > X
- Trend
- Sum last 30 days, last 60 days, …
- Most frequent
- Time between
- …

Example

- Data is ***imbalanced***
  - 1% "Yes", 99% "No
  - E.g., fraud, cancer, spam
- Model always predicts "No"

		Predicted: Yes	Predicted: No	
Actual	Yes	0	10	10
	No	0	990	990
		0	1000	1000

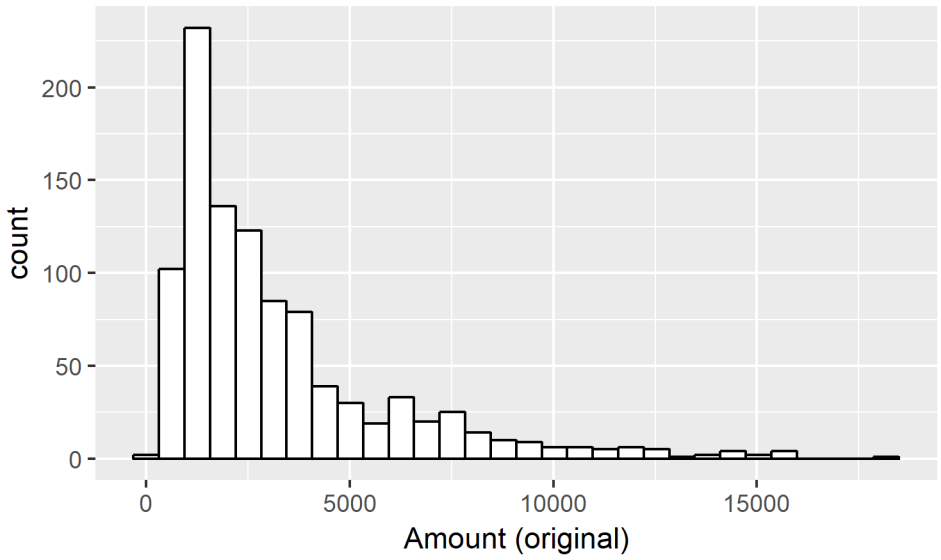
Performance Metrics		
Accuracy	=	98%
Average Accuracy	=	50%
Kappa	=	0%
F1-Score	=	NA
Recall, Sensitivity	=	0%
Specificity	=	100%
Precision	=	NA
NPV	=	99%

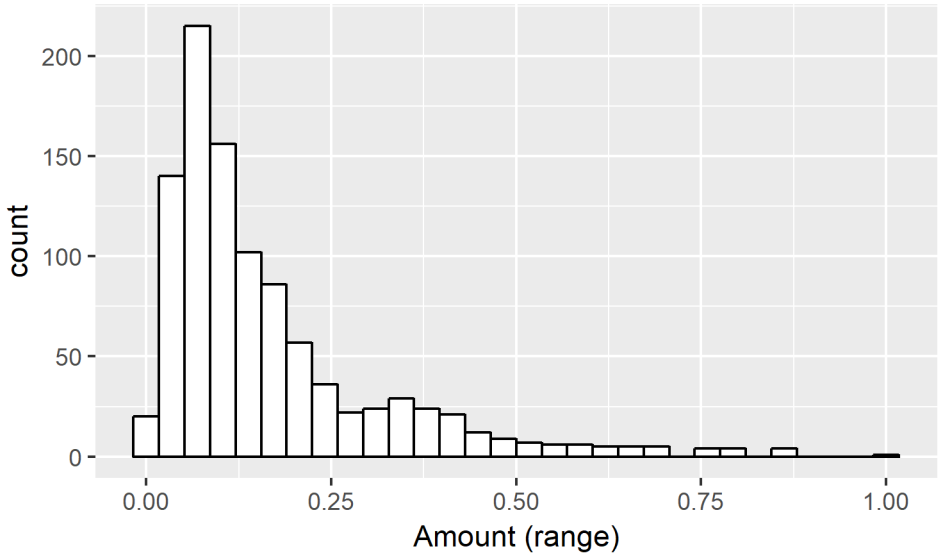
Scaling

- *Standardization*: usually recommended
  - Good for RBF kernel of SVM; l1, l2 regularization, distance metrics
- *Min-max* (aka *normalization*): when features need to have the same positive scale
  - Image processing
  - NNs
- *Log:* Helps de-skew data, and reduce effect of outliers
- *Box-Cox/Yeo-Johnson*: Make the dataset more normal

Range Scaling

```
p <- preprocess(df['Amount'], method = 'range')
df['Amount.range'] <- predict(p, df['Amount'])
```



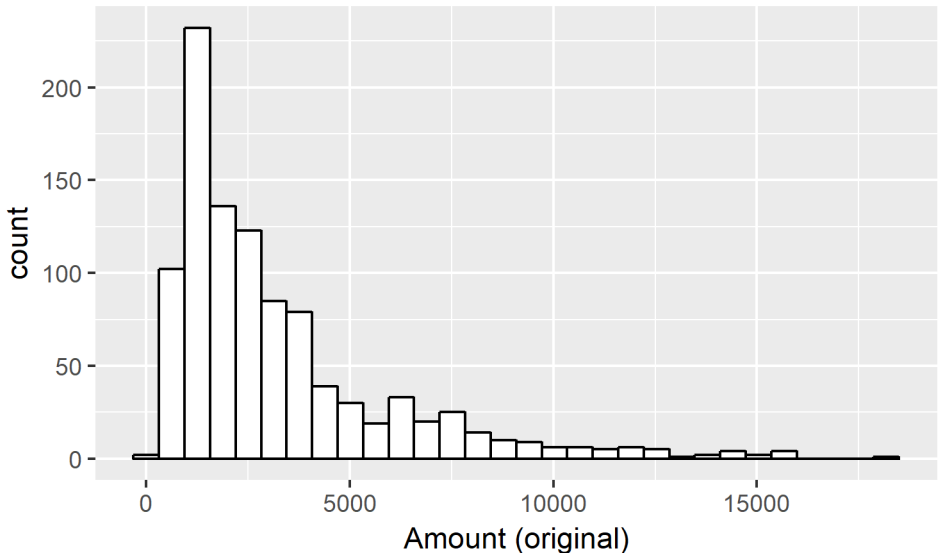


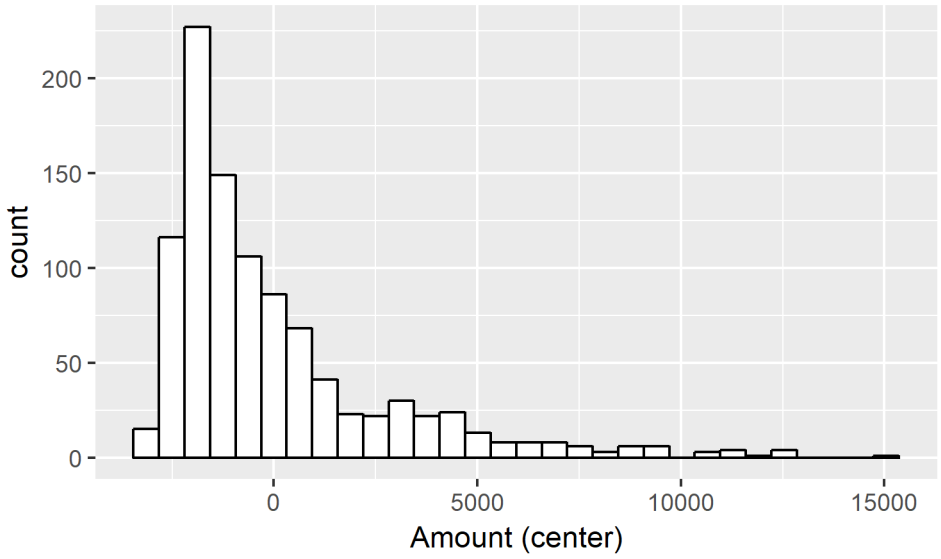
- Aka *min-max scaling*
- Scales data to be within [0, 1]
- Results:
  - Scale changes
  - Distribution shape doesn't change

FeatureEng_GermanCredit.Rmd

Centering

```
p <- preprocess(df['Amount'], method = 'center')
df['Amount.center'] <- predict(p, df['Amount'])
```



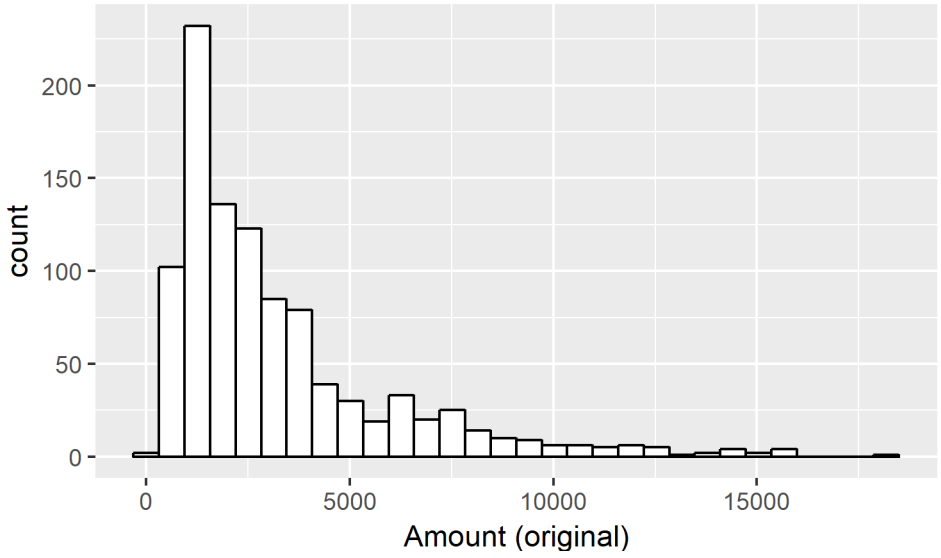


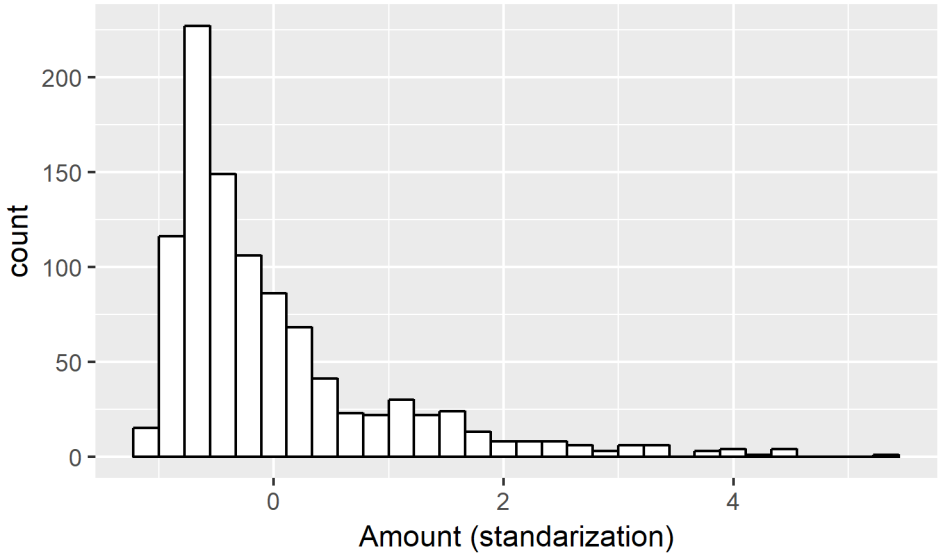
- Centers the feature around zero
- Results:
  - Mean is now 0
  - Scale doesn't change
  - Distribution shape doesn't change

FeatureEng_GermanCredit.Rmd

Standardization

```
p <- preprocess(df['Amount'], method = c('center', 'scale'))
df['Amount.standarization'] <- predict(p, df['Amount'])
```



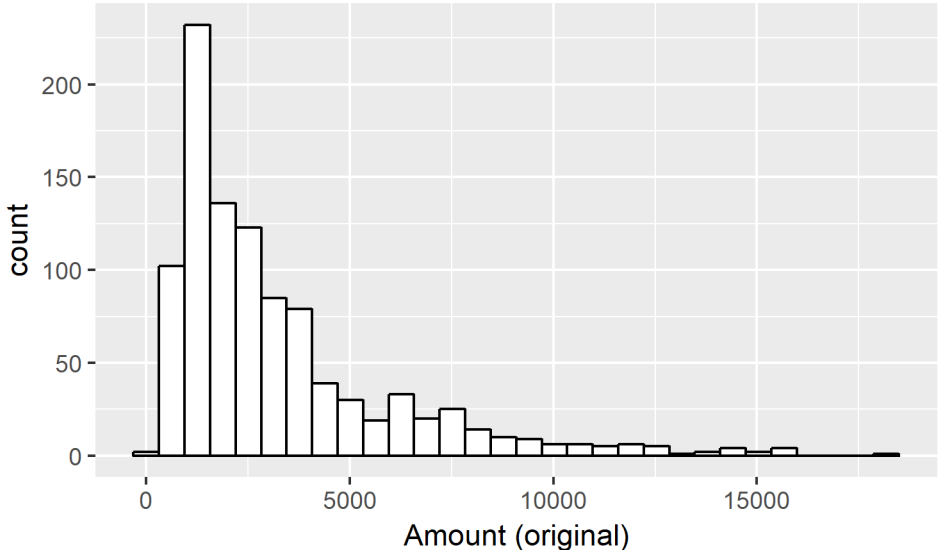


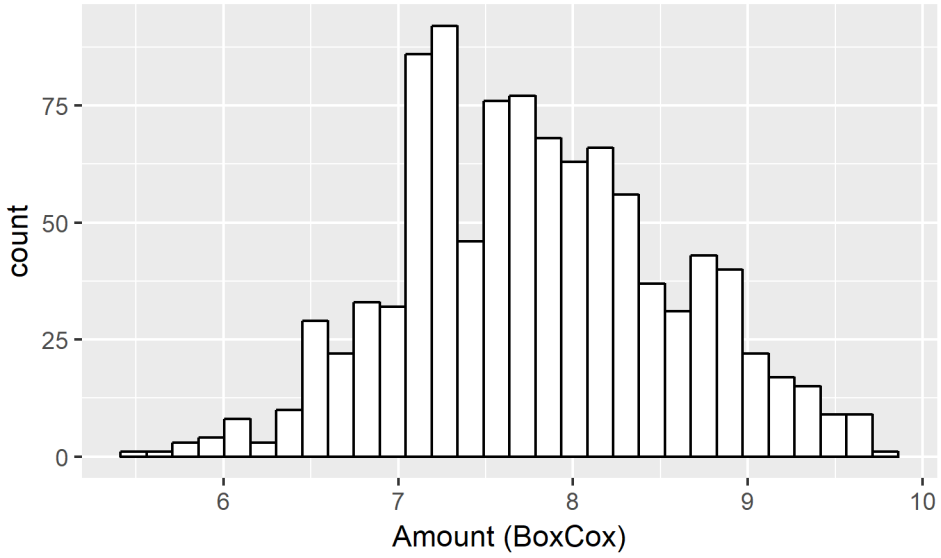
- Centers around zero, and divides by standard deviation
- Results:
  - Scale changes
  - Distribution shape doesn't change

FeatureEng_GermanCredit.Rmd

Box-Cox Transformation

```
p <- preprocess(df['Amount'], method = c('BoxCox'))
df['Amount.BoxCox'] <- predict(p, df['Amount'])
```



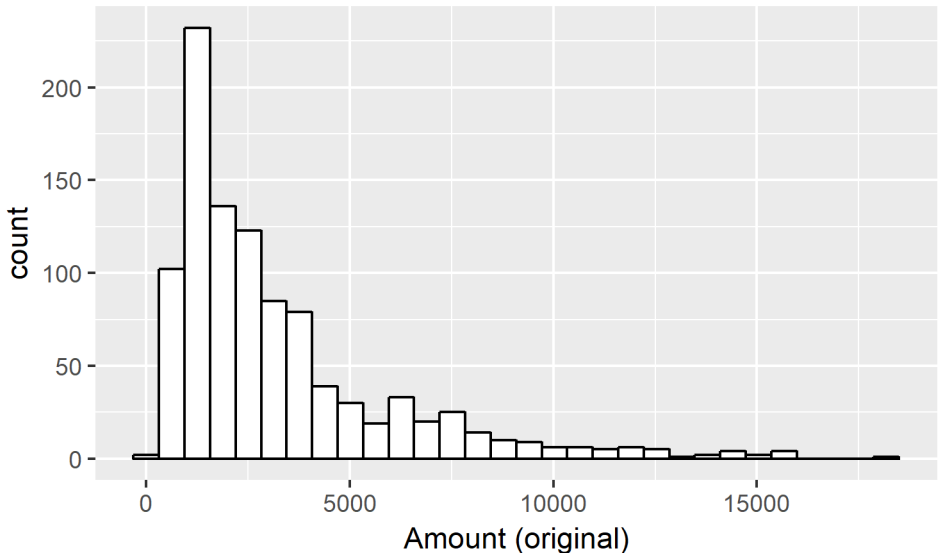


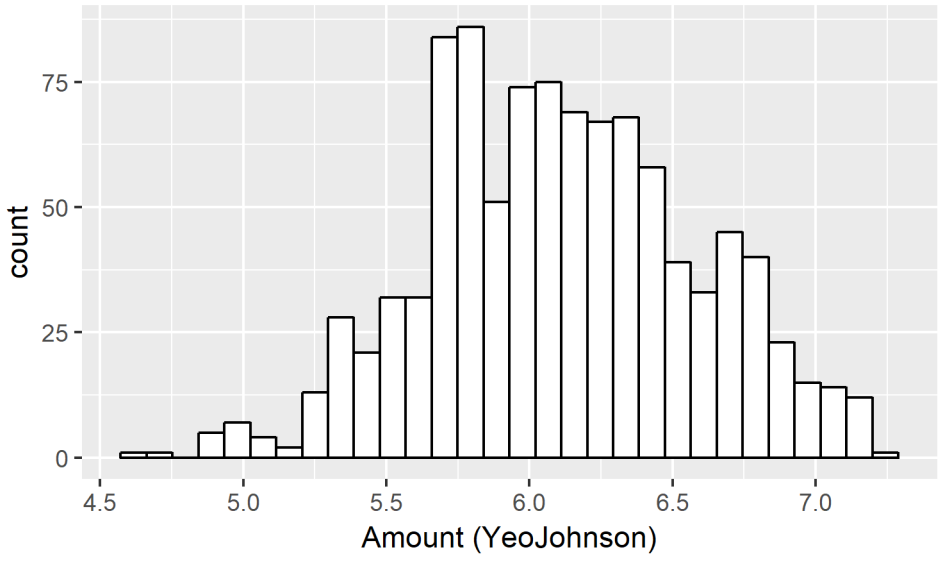
- Reduces skew and transforms into normal distribution
- Results:
  - Scale changes
  - Distribution shape changes

FeatureEng_GermanCredit.Rmd

Yeo-Johnson Transformation

```
p <- preprocess(df['Amount'], method = c('YeoJohnson'))
df['Amount.YeoJohnson'] <- predict(p, df['Amount'])
```





- Reduces skew and transforms into normal distribution
- Like Box-Cox, but can handle negative numbers
- Results:
  - Scale changes
  - Distribution shape changes

Example

- German Credit dataset (62 features)
- caret::sbf() function (Selection by Filtering)

```
filterCtrl <- sbfControl(functions = rfSBF)
r <- sbf(formula, data = df, sbfControl = filterCtrl)
r
```

```
Selection By Filter

Outer resampling method: Bootstrapped (25 reps)

Resampling performance:

 Accuracy  Kappa AccuracySD KappaSD
   0.7475 0.3277    0.01721 0.04169

Using the training set, 31 variables were selected:
   Duration, Amount, InstallmentRatePercentage, Age, ForeignWorker...

During resampling, the top 5 selected variables (out of a possible 55):
   Amount (100%), CheckingAccountStatus.lt.0 (100%), CheckingAccountStatus.none (100%),
CreditHistory.Critical (100%), CreditHistory.NoCredit.AllPaid (100%)

On average, 31.2 variables were selected (min = 25, max = 37)
```

Example

Table 19.2: Cross-validation results for recursive feature selection

	Full set		Reduced set			
	ROC	C.I.	Size	ROC	C.I.	p-value
LDA	0.844	(0.82 – 0.87)	35	0.916	(0.90 – 0.94)	0
RF	0.891	(0.87 – 0.91)	7	0.898	(0.88 – 0.92)	0.1255
SVM	0.889	(0.87 – 0.91)	127	0.891	(0.87 – 0.91)	0.0192
Logistic reg.	0.785	(0.76 – 0.81)	13	0.857	(0.83 – 0.88)	0
N. Bayes	0.798	(0.77 – 0.83)	17	0.832	(0.81 – 0.86)	0.0002
K-NN	0.849	(0.83 – 0.87)	125	0.796	(0.77 – 0.82)	1.0000

The "C.I." column corresponds to 95 % confidence intervals while the p-value column corresponds to a statistical test that evaluates whether the ROC value for the reduced model was a larger than the curve associated with all of the predictors

Kuhn et al.

Forward Selection

- Build model with one feature; assess performance
- Add one more feature; if it helps performance, keep it
- Repeat

```
Create an initial model containing only an intercept term.
repeat
    for each predictor not in the current model do
        Create a candidate model by adding the predictor to the
        current model
        Use a hypothesis test to estimate the statistical significance
        of the new model term
    end
    if the smallest p-value is less than the inclusion threshold then
        Update the current model to include a term corresponding to
        the most statistically significant predictor
    else
        Stop
    end
until no statistically significant predictors remain outside the model
```

Algorithm 19.1: Classical forward selection for linear regression models

Kuhn et al..

Example

- German Credit Data
- KNN
- caret::expand.grid() and caret::train(tuneGrid=)

```
grid <- expand.grid(.kmax = c(5, 10, 25),
                    .distance=c(1, 2),
                    .kernel=c("rectangular", "triangular",
                              "biweight", "cos", "gaussian", "optimal"))

ctrl <- trainControl(method = "repeatedcv",
                     number = 10, repeats = 5,
                      classProbs = TRUE, returnResamp = "all")

kknn_fit <- train(formula,
                  data = train,
                  method = "kknn",
                  metric="Kappa",
                  trControl=ctrl, tuneGrid = grid)
```

Example

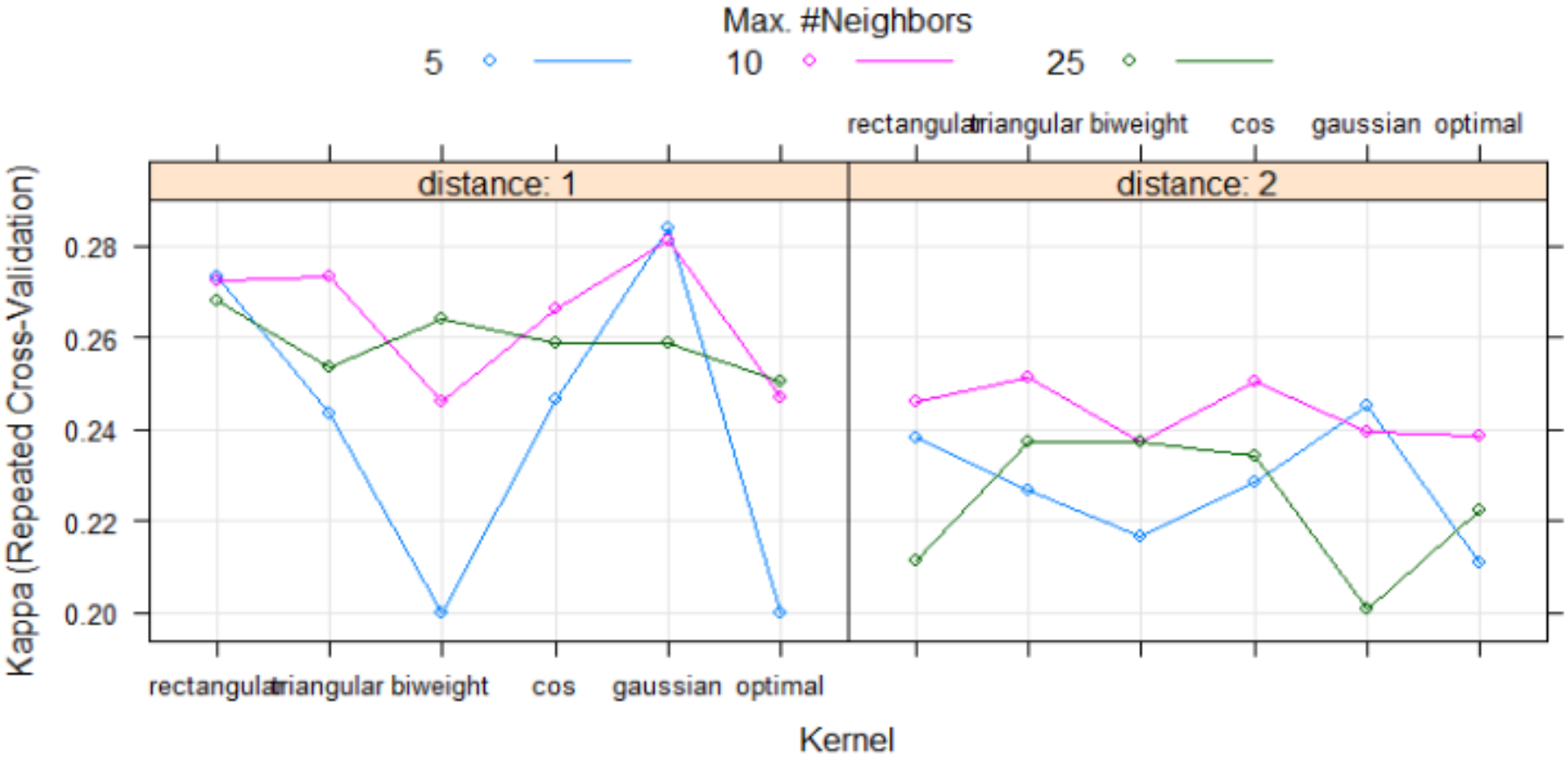
kknn_fit

kmax	distance	kernel	Accuracy	Kappa
5	1	rectangular	0.72875	0.2733849
5	1	triangular	0.70350	0.2434248
5	1	biweight	0.67375	0.1996582
5	1	cos	0.70475	0.2465214
5	1	gaussian	0.73275	0.2840083
5	1	optimal	0.67175	0.1997951
5	2	rectangular	0.71200	0.2380392
5	2	triangular	0.69125	0.2264629
5	2	biweight	0.67700	0.2164267
5	2	cos	0.69225	0.2283583
5	2	gaussian	0.71500	0.2452693
5	2	optimal	0.67025	0.2109859
25	1	triangular	0.73250	0.2535414
25	1	biweight	0.73100	0.2639820
25	1	cos	0.73350	0.2586052
25	1	gaussian	0.73175	0.2589459
25	1	optimal	0.73225	0.2503410
25	2	rectangular	0.72175	0.2111504
25	2	triangular	0.72475	0.2373104
25	2	biweight	0.71200	0.2372264
25	2	cos	0.72400	0.2343167
25	2	gaussian	0.71800	0.2008217
25	2	optimal	0.72200	0.2223402

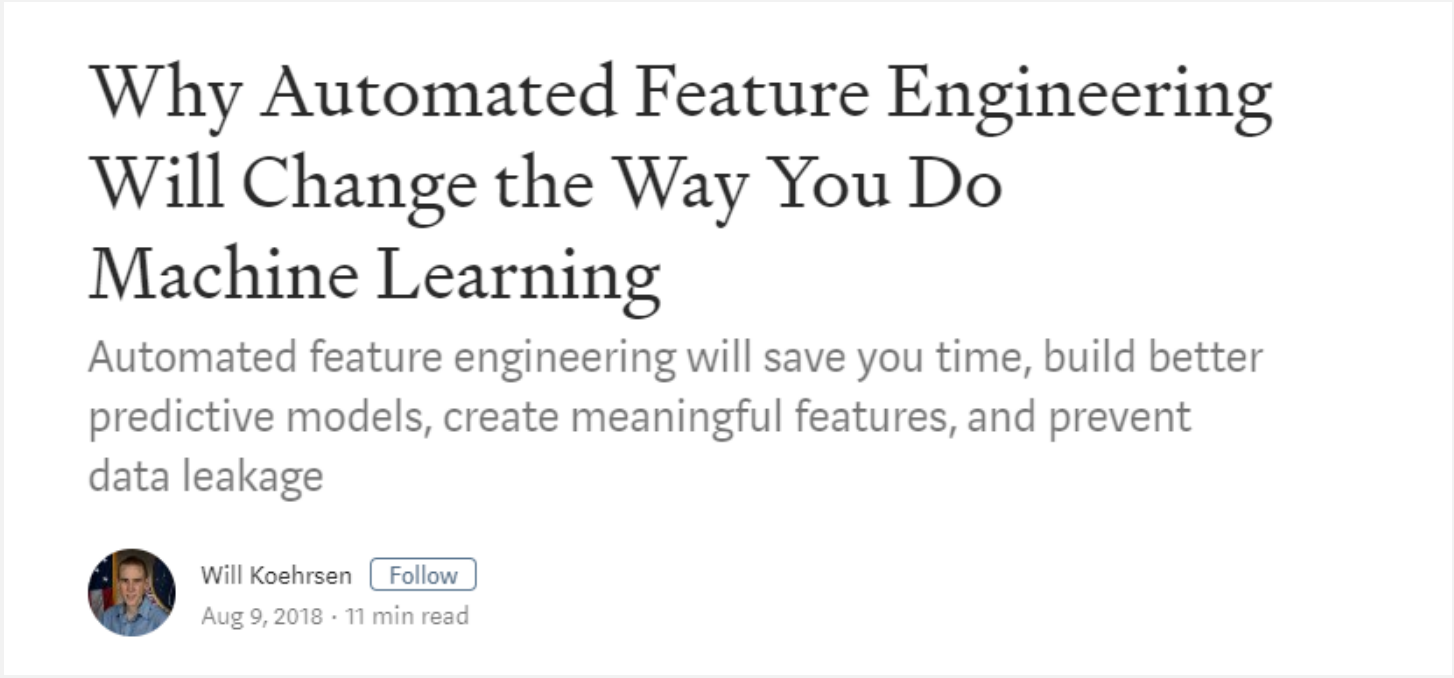
Kappa was used to select the optimal model using the largest value.
The final values used for the model were kmax = 5, distance = 1 and kernel = gaussian.

Example

```
plot(kknn_fit)
```



Automated Feature Engineering



Why Automated Feature Engineering Will Change the Way You Do Machine Learning

Automated feature engineering will save you time, build better predictive models, create meaningful features, and prevent data leakage

Will Koehrsen Follow

Aug 9, 2018 · 11 min read

https://towardsdatascience.com/why-automated-feature-engineering-will-change-the-way-you-do-machine-learning-5c15bf188b96

DataRobot

H2O.ai

Featuretools

MODEL EVALUATION/SELECTION

The Analytics Process

Data Preparation

Organize and clean the data

Feature Engineering

Normalization
Discretization
Coding
Temporal, text, image

Feature Selection

Filter
Wrapper

Analysis / Modeling

Conduct the analysis using the appropriate techniques

Model Training

Algorithms
Ensembles
Parameter tuning

Model Selection/ Evaluation

Cross validation
A/B Testing

Definition

- ***Model evaluation***: evaluating different models to see which is best
- Common technique:
  - Perform feature engineering
  - Perform feature selection
  - For each model you want to try (e.g., DT, RF, NN, KNN):
    - Perform parameter tuning
    - Run CV on final, best model
    - Save range of performances across folds
  - Compare range of performances for each model
  - Select best one

Example

- German Credit dataset
- Use caret::resamples() to compare models

```
ctrl <- trainControl(method = "repeatedcv",
                     number = 10, repeats = 5, classProbs = TRUE)

rpartFit <- train(formula, data = train, "rpart",
                  preProc=c('nzv', 'center', 'scale'),
                  trControl = ctrl, tuneLength = 10, metric="Kappa")

svmFit <- train(formula, data = train, "svmLinear3",
                  preProc=c('nzv', 'center', 'scale'),
                  trControl = ctrl, tuneLength = 10, metric="Kappa")

nbFit <- train(formula, data = train, "naive_bayes",
                  preProc=c('nzv', 'center', 'scale'),
                  trControl = ctrl, tuneLength = 10, metric="Kappa")

rfFit <- train(formula, data = train, "parRF",
                  preProc=c('nzv', 'center', 'scale'),
                  trControl = ctrl, tuneLength = 10, metric="Kappa")

gbmFit <- train(formula, data = train, "gbm",
                preProc=c('nzv', 'center', 'scale'),
                trControl = ctrl, tuneLength = 10, metric="Kappa", verbose=F)

kknnFit <- train(formula, data = train, "kknn",
                 preProc=c('nzv', 'center', 'scale'),
                 trControl = ctrl, tuneLength = 10, metric="Kappa")
```

Example

- German Credit dataset
- Use caret::resamples() to compare models

```
resamps <- resamples(list(rpart = rpartFit,
                          svm = svmFit,
                          nb = nbFit,
                          rf = rfFit,
                          gbm = gbmFit,
                          kknn = kknnFit))
```

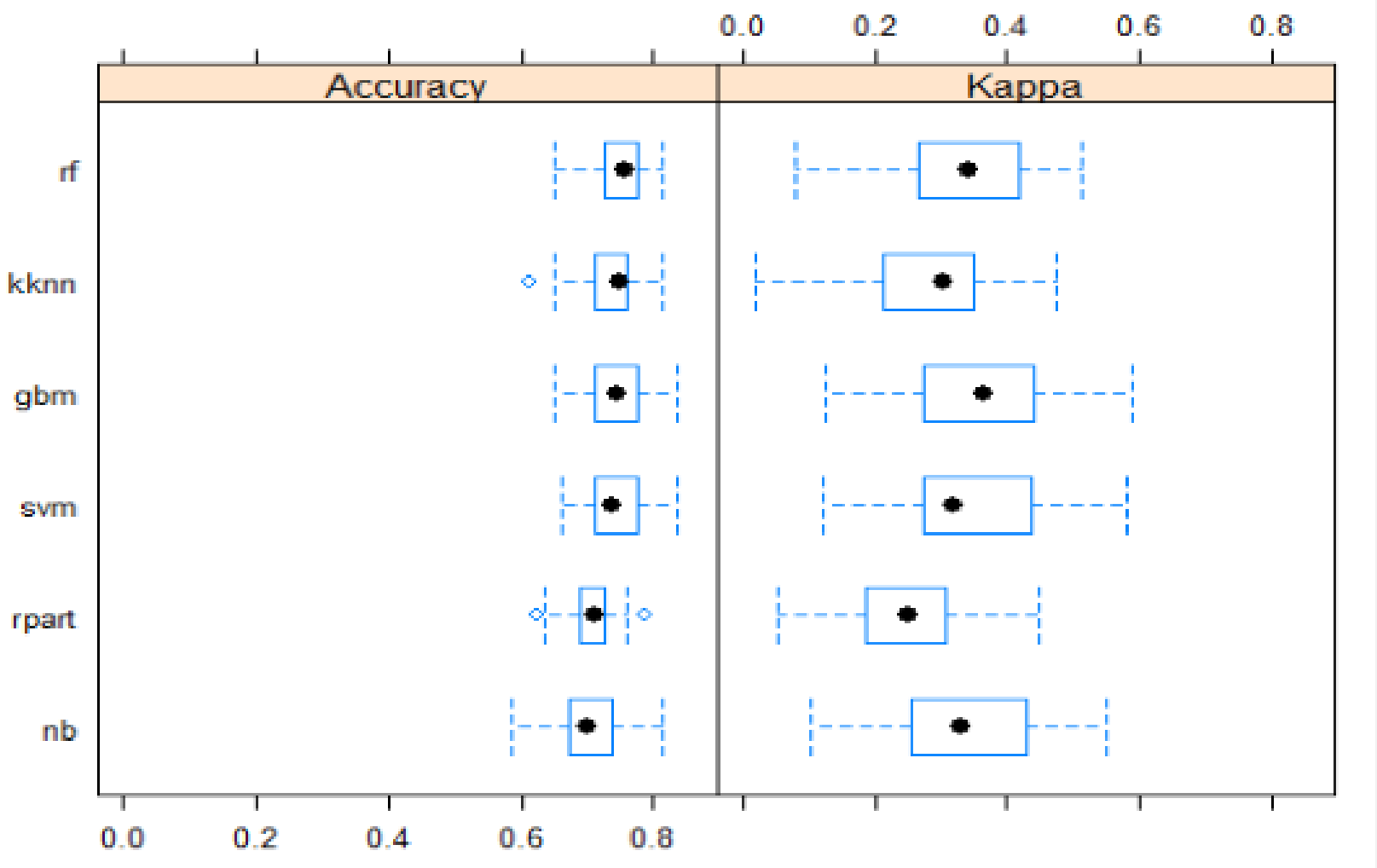
Example

```
summary(resamps)
```

```
Models: rpart, svm, nb, rf, gbm, kknn
Number of resamples: 50

Accuracy
        Min. 1st Qu.  Median    Mean 3rd Qu.   Max. NA's
rpart 0.6250  0.6875 0.71250 0.71025  0.7250 0.7875    0
svm   0.6625  0.7125 0.73750 0.74250  0.7750 0.8375    0
nb    0.5875  0.6750 0.70000 0.70350  0.7375 0.8125    0
rf    0.6500  0.7250 0.75625 0.75300  0.7750 0.8125    0
gbm   0.6500  0.7125 0.74375 0.74050  0.7750 0.8375    0
kknn  0.6125  0.7125 0.75000 0.73625  0.7625 0.8125    0

Kappa
             Min.   1st Qu.    Median      Mean   3rd Qu.      Max. NA's
rpart 0.05405405 0.1885311 0.2516234 0.2496412 0.3072289 0.4480519    0
svm   0.12162162 0.2721519 0.3181818 0.3404345 0.4350527 0.5779221    0
nb    0.10493827 0.2568303 0.3306452 0.3331034 0.4228406 0.5481928    0
rf    0.07894737 0.2687349 0.3421053 0.3391160 0.4178082 0.5129870    0
gbm   0.12500000 0.2731929 0.3632221 0.3485287 0.4328560 0.5886076    0
kknn  0.01898734 0.2184066 0.3027778 0.2906152 0.3493151 0.4753086    0
```

Example

```
bwplot(resamps)
```



Scaling

- Scaling: any transformation that changes the value of a feature to either fall within a different range, or have a new distributional shape, or both
- E.g., ***range scaling***:

$$a_i' = \frac{a_i - min(a)}{max(a) - min(a)} \times (high - low) + low$$

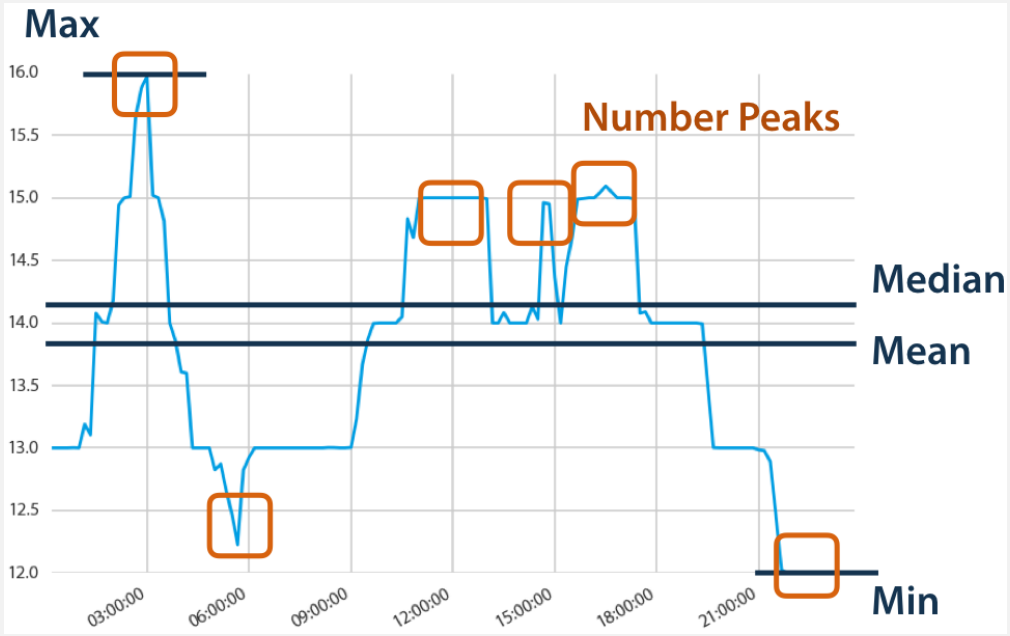
- Other common methods:
  - Normalization – makes the distribution more normal
  - Standardization
  - Box-Cox transformation
  - Yeo-Johnson transformation

Extracting Features from Time Series

- Extract features that describe characteristic of time series
  - Max, min, median, length, mean change, range count, linear trend, …

TID	P1	P2	P3	P4	P5	P6	P7	P8	…
1	266.4	145.9	183.1	119.3	180.3	168.5	231.8	224.5	
2	265.0	150.9	179.1	110.3	173.3	163.5	224.8	216.5	
3	259.1	149.9	181.1	119.3	183.3	171.5	221.8	209.5	
4	250.3	148.9	173.1	127.3	179.3	176.5	221.8	215.5	
5	248.0	154.9	172.1	121.3	174.3	182.5	212.8	209.5	
6	250,9	152.9	173.1	124.3	174.3	188.5	202.8	205.5	

TID	Max	Min	Median	Peaks	…
1	336.5	94.3	122.9	4	
2	332.5	97.3	129.9	6	
3	342.5	107.3	136.9	7	
4	345.5	116.3	126.9	2	
5	336.5	120.3	133.9	9	
6	333.5	115.3	131.9	4	



Source: https://tsfresh.readthedocs.io/en/latest/text/introduction.html

Extracting Features from Text

- Key words, key phrases, key topics
- Lots of options: Bag of Words, Topic embeddings, Word2Vec, …
  - Will discuss in future course

ID	Date	Text
1	2259	My dog ate my homework.
2	6836	The cat ate my sandwich.
3	7722	A dolphin ate the homework and the sandwich.

ID	Date	a	and	ate	cat	dolphin	dog	homework	my	sandwich	the
1	2259	0	0	1	0	0	1	1	2	0	0
2	6836	0	0	1	1	0	0	0	1	1	1
3	7722	1	1	1	0	1	0	1	0	1	2

Number of times "the" occurs in document 3

Feature Hashing

Feature Hashing: converting each level to a bucket using a hash function.

- Aka "the hashing trick"

Race	1	2	3	4	5	6	7	8
Asian	1	0	0	0	0	0	0	0
Asian	1	0	0	0	0	0	0	0
Asian	1	0	0	0	0	0	0	0
Hispanic	0	0	0	0	0	1	0	0
Hispanic	0	0	0	0	0	1	0	0
Hispanic	0	0	0	0	0	1	0	0
Other	1	0	0	0	0	0	0	0
Other	1	0	0	0	0	0	0	0
Other	1	0	0	0	0	0	0	0
0	0	1	0	0	0	0	0	0
0	0	1	0	0	0	0	0	0
0	0	1	0	0	0	0	0	0
AfricanAmerican	0	1	0	0	0	0	0	0
AfricanAmerican	0	1	0	0	0	0	0	0
AfricanAmerican	0	1	0	0	0	0	0	0
Caucasian	0	0	0	0	0	1	0	0
Caucasian	0	0	0	0	0	1	0	0
Caucasian	0	0	0	0	0	1	0	0

Original Feature

8 New "Hashed" Features

Aside: What is a Hash Function?

- It's a function just like any other…
  - my_function(x) = 2x + 4
    - my_function(8) = 2*8 + 4 = 20
    - my_function(0) = 2*0 + 4 = 4
- … that usually takes strings as inputs and produces numbers …
  - my_hash("cheese") = 21
  - my_hash("soda") = 45
- … and the produced numbers are within a certain range …
  - my_hash(x) will always produce values between 0 and 100
- … and are always repeatable:
  - my_hash("cheese") = 21
  - my_hash("cheese") = 21
  - my_hash("cheese") = 21

Aside: What is a Hash Function?

- Most hash functions convert each character in the string to a number, then add the numbers, and then mod the result
- $myhash1("cheese")$
$$= (c + h + e + e + s + e)\%10$$
$$= (3 + 8 + 5 + 5 + 19 + 5)\%10$$
$$= (45)\%10$$
$$= 5$$
- $myhash2("cheese")$
$$= (c + h + e + e + s + e) * 5673 \,\%\, 10$$
- $myhash3("cheese")$
$$= (c * h + e + e/s + e) * 56744 \,\%\, 10$$

Aside: What is a Hash Function?

- Two different inputs might produce the same output
  - $myhash1("cheese")$=88
  - $myhash1("chocolate")$=88
  - $myhash1("pizza")$=88
- Can't go backwards
  - Knowing that the hash function returned 88, you don't know what the input was

Aside: What is a Hash Function?

- Good for "binning" things
  - You have a list of 1 million user names
  - Want to randomly group them into 10 buckets, repeatably
- Good for finding duplicate things
  - If two things hash to the same value, then they might be duplicates

Aside: Hashing One Feature

Race
Asian
Asian
Asian
Hispanic
Hispanic
Hispanic
Other
Other
Other
0
0
0
AfricanAmerican
AfricanAmerican
AfricanAmerican
Caucasian
Caucasian
Caucasian

Original Feature

1	2	3	4	5	6	7	8
1	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0
0	0	0	0	0	1	0	0
0	0	0	0	0	1	0	0
0	0	0	0	0	1	0	0
1	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0
0	1	0	0	0	0	0	0
0	1	0	0	0	0	0	0
0	1	0	0	0	0	0	0
0	1	0	0	0	0	0	0
0	1	0	0	0	0	0	0
0	1	0	0	0	0	0	0
0	0	0	0	0	1	0	0
0	0	0	0	0	1	0	0
0	0	0	0	0	1	0	0

8 New "Hashed" Features

Mapping:

race0	raceAfricanAmerican	raceOther	raceHispanic	raceCaucasian	raceAsian
2	2	1	6	6	1

Aside: Hashing Two Features

Race	Gender
Asian	Female
Asian	Male
Asian	Female
Hispanic	Female
Hispanic	Male
Hispanic	Male
Other	Female
Other	Female
Other	Male
0	Male
0	Male
0	Male
AfricanAmerican	Female
AfricanAmerican	Female
AfricanAmerican	Male
Caucasian	Male
Caucasian	Male
Caucasian	Female

Original Features

1	2	3	4	5	6	7	8
1	0	0	0	1	0	0	0
2	0	0	0	0	0	0	0
1	0	0	0	1	0	0	0
0	0	0	0	1	1	0	0
1	0	0	0	0	1	0	0
1	0	0	0	0	1	0	0
1	0	0	0	1	0	0	0
1	0	0	0	1	0	0	0
2	0	0	0	0	0	0	0
1	1	0	0	0	0	0	0
1	1	0	0	0	0	0	0
1	1	0	0	0	0	0	0
0	1	0	0	1	0	0	0
0	1	0	0	1	0	0	0
1	1	0	0	0	0	0	0
1	0	0	0	0	1	0	0
1	0	0	0	0	1	0	0
0	0	0	0	1	1	0	0

8 New "Hashed" Features

Mapping:

race0	raceAfricanAmerican	raceOther	raceHispanic	raceCaucasian	raceAsian
2	2	1	6	6	1

genderMale	genderFemale
1	5

Aside: Hashing Three Features

Race	Gender	Age
Asian	Female	[40-50)
Asian	Male	[60-70)
Asian	Female	[30-40)
Hispanic	Female	[50-60)
Hispanic	Male	[70-80)
Hispanic	Male	[80-90)
Other	Female	[60-70)
Other	Female	[70-80)
Other	Male	[40-50)
0	Male	[50-60)
0	Male	[50-60)
0	Male	[70-80)
AfricanAmerican	Female	[50-60)
AfricanAmerican	Female	[20-30)
AfricanAmerican	Male	[60-70)
Caucasian	Male	[70-80)
Caucasian	Male	[50-60)
Caucasian	Female	[80-90)

1	2	3	4	5	6	7	8
1	0	1	0	1	0	0	0
2	0	0	0	0	0	0	1
1	0	0	0	1	1	0	0
1	0	0	0	1	1	0	0
1	0	1	0	0	1	0	0
1	0	0	0	0	1	1	0
1	0	0	0	1	0	0	1
1	0	1	0	1	0	0	0
2	0	1	0	0	0	0	0
2	1	0	0	0	0	0	0
2	1	0	0	0	0	0	0
1	1	1	0	0	0	0	0
1	1	0	0	1	0	0	0
0	1	0	0	1	0	1	0
1	1	0	0	0	0	0	1
1	0	1	0	0	1	0	0
2	0	0	0	0	1	0	0
0	0	0	0	1	1	1	0

Original Features

8 New "Hashed" Features

Mapping:

race0	raceAfricanAmerican	raceOther	raceHispanic	raceCaucasian	raceAsian
2	2	1	6	6	1

genderMale	genderFemale
1	5

age[20-30)	age[30-40)	age[40-50)	age[50-60)	age[60-70)	age[70-80)	age[80-90)
7	6	3	1	8	3	7

Aside: Feature Hashing

- Pros
  - Can save memory
  - Can save developer time – no need to provide list of every possible value
  - Easily handle missing data and streaming data
  - Fast and simple
- Cons
  - No exact inverse mapping → loss of interpretability
  - Hash collisions are possible → loss of information

Contrasts

- If you have ordered data, OHE loses information
  - E.g., "Low", "Medium", "High"
- Solution: ***contrasts*** are encodings with ordered numeric values

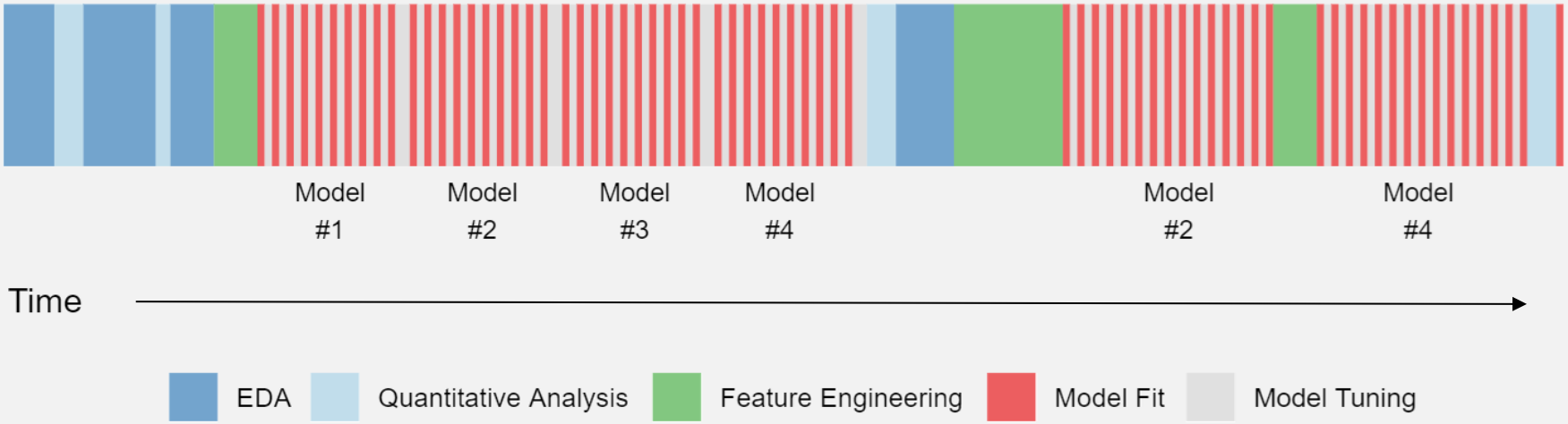
PID	Income	…
1	Low	
2	Low	
3	Medium	
4	High	
5	Medium	
6	High	

PID	Income	…
1	-0.71	
2	-0.71	
3	0	
4	0.71	
5	0	
6	0.71	

- The contrast levels add up to 0
- Values chosen by some cool mathematics; details omitted here

How Much Time on What?

- ***Varies tremendously per project!***
- Here's one example:



Part science, part art.

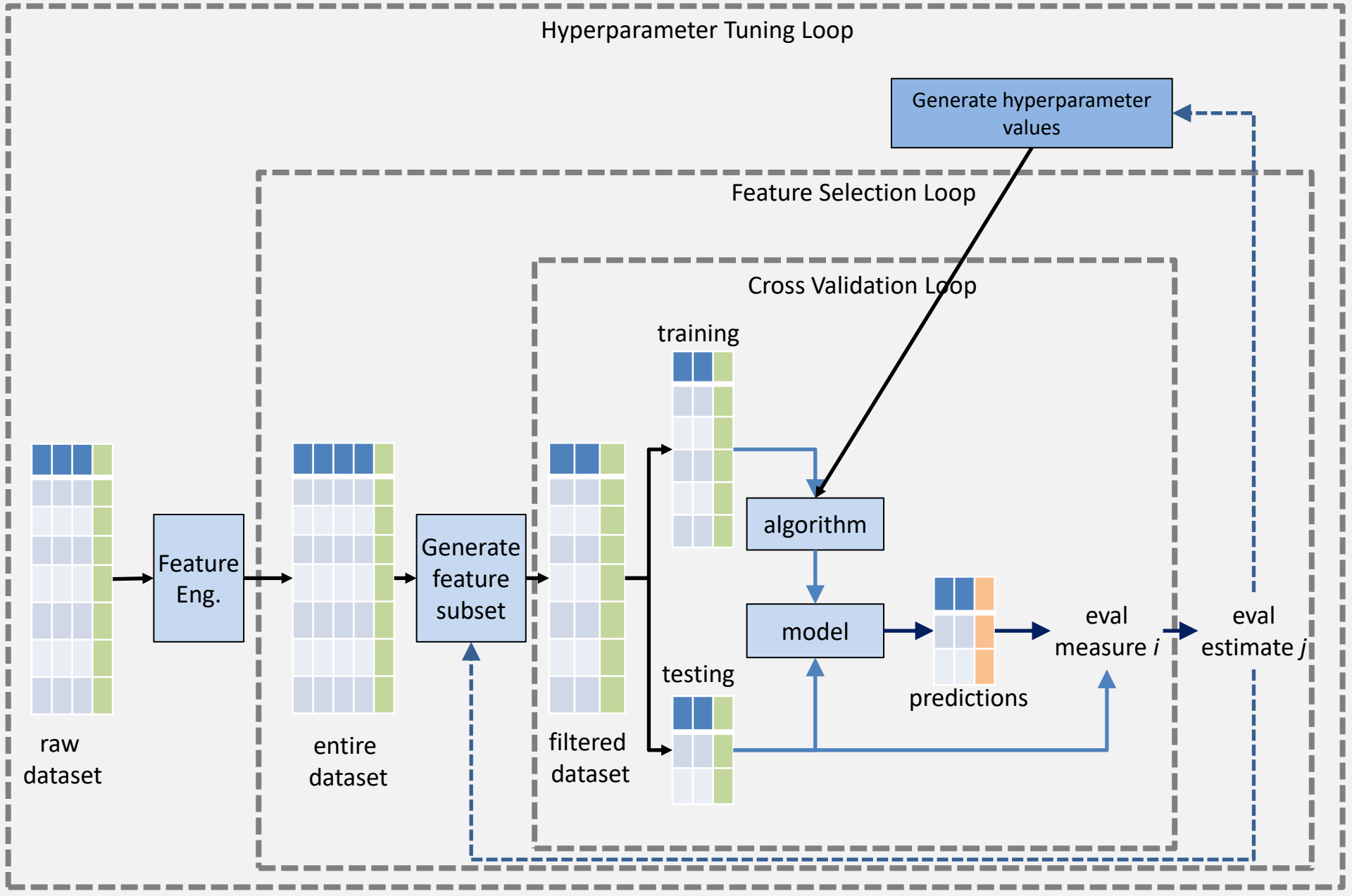
Kuhn et al. *Feature Engineering*. Fig 1.4.

Caret's Grid Search

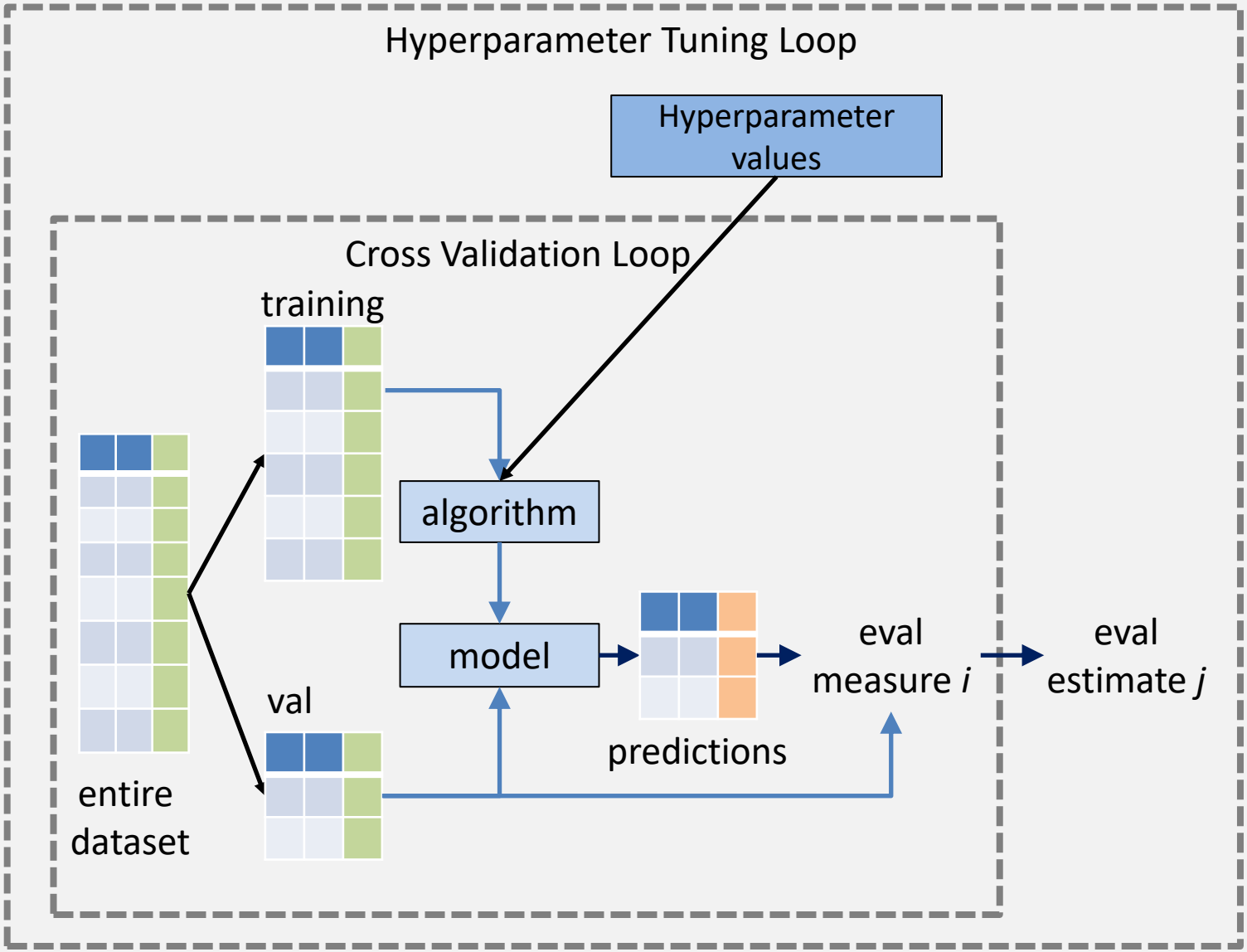
```
Define sets of model parameter values to evaluate
for each parameter set do
    for each resampling iteration do
        Hold–out specific samples
        [Optional] Pre–process the data
        Fit the model on the remainder
        Predict the hold–out samples
    end
    Calculate the average performance across hold–out predictions
end
Determine the optimal parameter set
Fit the final model to all the training data using the optimal parameter set
```

Caret Docs

Total, Crazy Loop

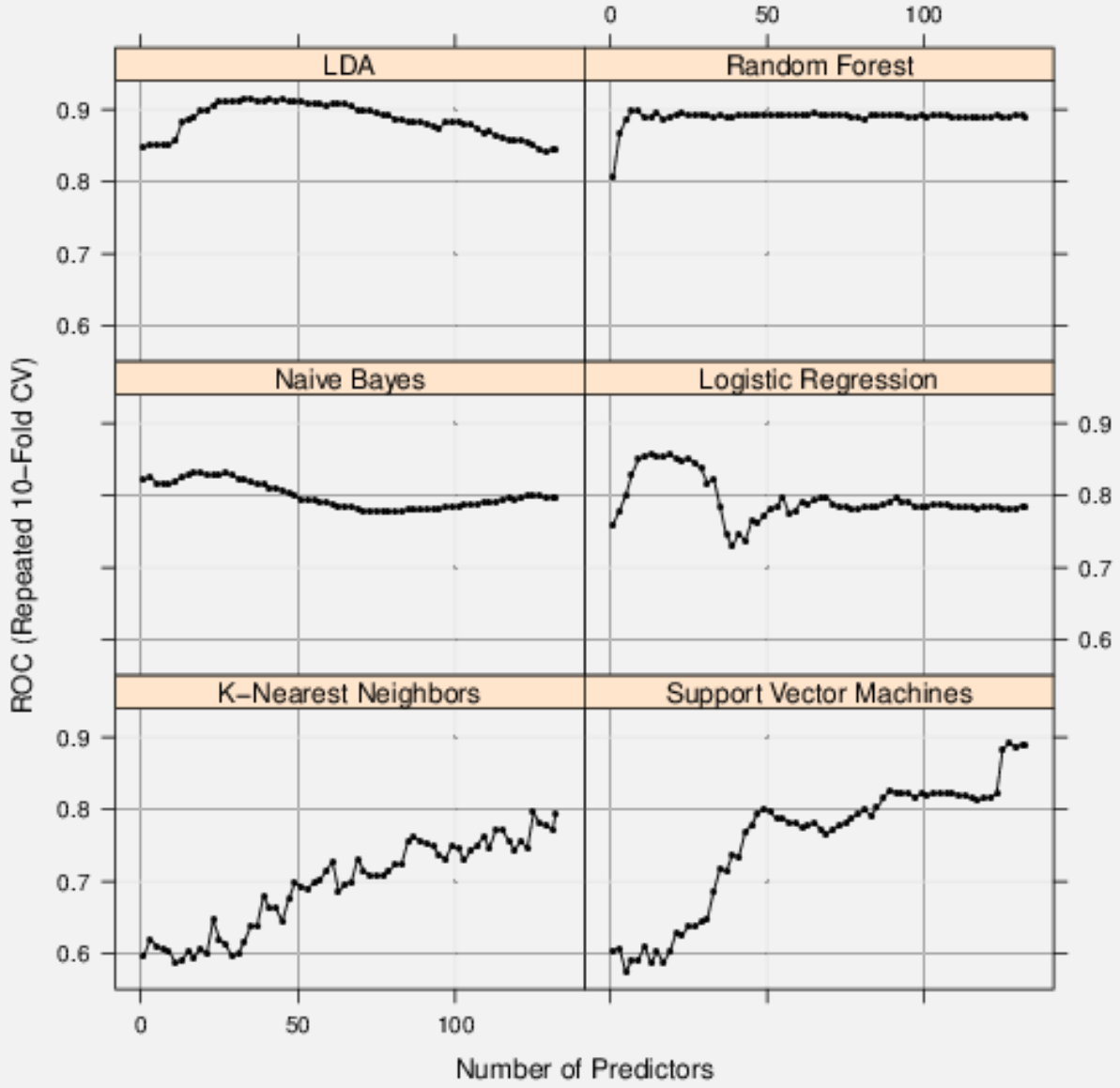


General Process



Example

Alzheimer's dataset



Kuhn et al. Figure 19.5

Backward Selection, aka RFE

- Build model with all features; assess performance
- Remove the least-important feature, re-build and re-assess
  - If performance is the same, drop the feature
- Repeat

1 Tune/train the model on the training set using all P predictors
2 Calculate model performance
3 Calculate variable importance or rankings
4 for *each subset size* S_i, $i = 1 \ldots S$ **do**
5 | Keep the S_i most important variables
6 | [Optional] Pre-process the data
7 | Tune/train the model on the training set using S_i predictors
8 | Calculate model performance
9 | [Optional] Recalculate the rankings for each predictor
10 **end**
11 Calculate the performance profile over the S_i
12 Determine the appropriate number of predictors (i.e. the S_i associated with the best performance)
13 Fit the final model based on the optimal S_i

Algorithm 19.2: Backward selection via the recursive feature elimination algorithm of Guyon et al. (2002)

Kuhn et al.

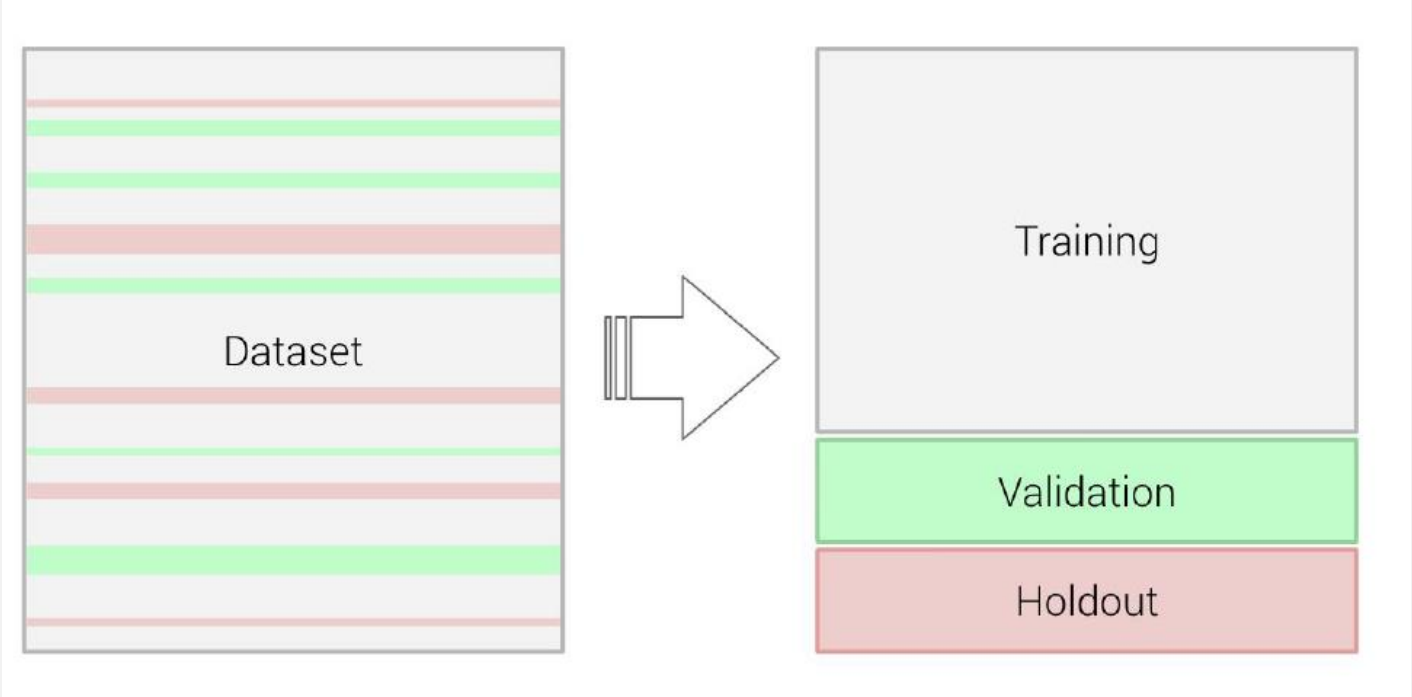
Feature-Feature Correlation Filter

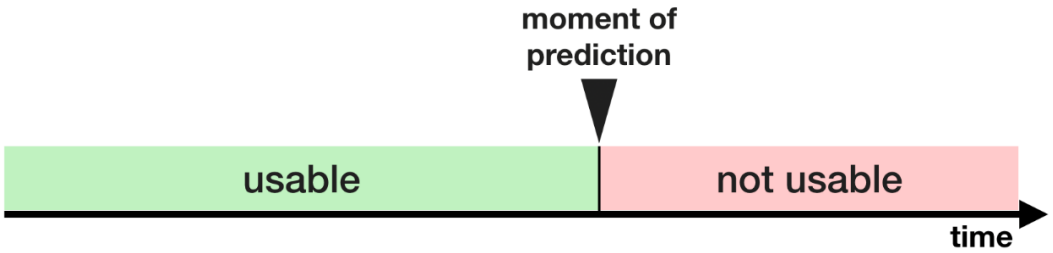
If two or more features are correlated, remove all but one.

Trip ID	Distance Driven	Gas Spend	…
1	52	20.80	
2	254	109.65	
3	100	29.56	
4	98	39.20	
5	547	269.65	
6	325	130.00	
7	923	478.51	
8	42	13.20	
9	874	350.11	
10	365	146.08	
…			

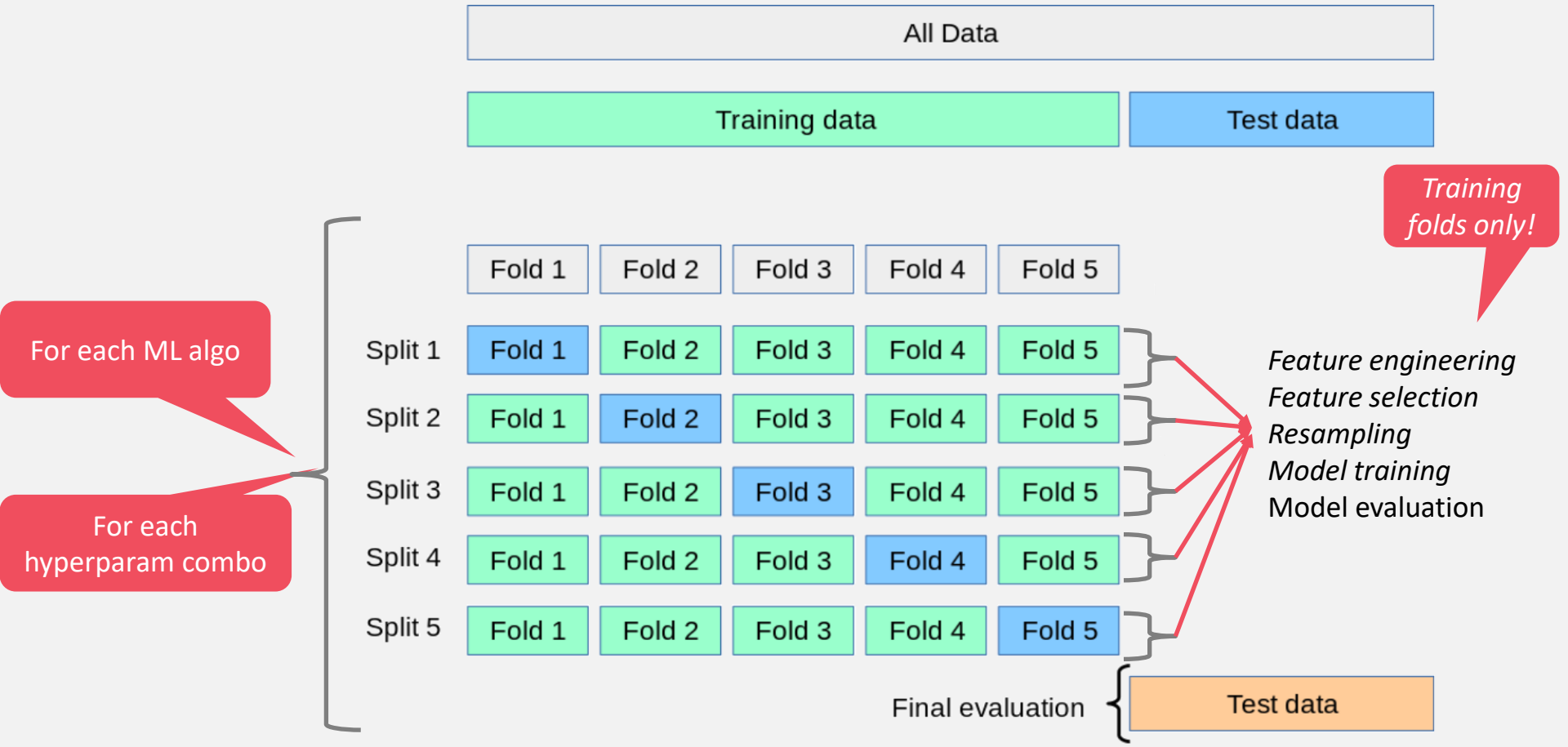
- Numeric-Numeric: Pearson
- Categorical-Categorical: Chi-squared
- Numeric-Categorical: ANOVA

Time Leakage





How It All Fits Together (Simple)



https://scikit-learn.org/stable/modules/cross_validation.html#cross-validation

Example

Smith SCHOOL OF BUSINESS Queen's University

[21]:
```
from imblearn.over_sampling import RandomOverSampler

ros = RandomOverSampler(random_state=0)

X_resampled, y_resampled = ros.fit_resample(X_train, y_train)
X_resampled.shape
y_resampled.shape
np.bincount(y_resampled)

evals_d.append(quick_evaluate_with_dt(X_resampled, X_test, y_resampled, y_test, 'Over Random'))
```

[21]:
```
(800, 8)
```

[21]:
```
(800,)
```

[21]:
```
array([400, 400], dtype=int64)
```

Example

Smith SCHOOL OF BUSINESS Queen's University

[22]:
```
from imblearn.over_sampling import SMOTE

X_resampled, y_resampled = SMOTE(random_state=0).fit_resample(X_train, y_train)

X_resampled.shape
y_resampled.shape
np.bincount(y_resampled)

evals_d.append(quick_evaluate_with_dt(X_resampled, X_test, y_resampled, y_test, 'Over SMOTE'))
```

[22]:
```
(800, 8)
```

[22]:
```
(800,)
```

[22]:
```
array([400, 400], dtype=int64)
```

[23]:
```
from imblearn.over_sampling import ADASYN

X_resampled, y_resampled = ADASYN(random_state=0).fit_resample(X_train, y_train)

X_resampled.shape
y_resampled.shape
np.bincount(y_resampled)

evals_d.append(quick_evaluate_with_dt(X_resampled, X_test, y_resampled, y_test,  'Over ADASYN'))
```

[23]:
```
(775, 8)
```

[23]:
```
(775,)
```

[23]:
```
array([400, 375], dtype=int64)
```

Time Leakage

Jan 1, 2022
Jan 1, 2022
Jan 2, 2022
Jan 3, 2022
Jan 4, 2022
Jan 4, 2022
Jan 8, 2022
Jan 15, 2022
Jan 15, 2022
Feb 5, 2022
Feb 18, 2022

All data

Feb 5, 2022
Jan 2, 2022
Jan 8, 2022
Jan 1, 2022
Feb 18, 2022
Jan 15, 2022
Jan 4, 2022

Training

Jan 15, 2022
Jan 3, 2022
Jan 1, 2022
Jan 2, 2022
Jan 4, 2022

Testing

Jan 1, 2022
Jan 3, 2022
Feb 5, 2022
Jan 8, 2022
Jan 15, 2022
Jan 4, 2022
Jan 15, 2022

Training

Jan 1, 2022
Jan 15, 2022
Jan 4, 2022
Feb 18, 2022
Jan 2, 2022

Testing

Regular K-fold splits

BAD

Jan 1, 2022
Jan 1, 2022
Jan 2, 2022
Jan 3, 2022
Jan 4, 2022
Jan 4, 2022
Jan 8, 2022

Training

Jan 15, 2022
Jan 15, 2022
Feb 5, 2022
Feb 18, 2022
Jan 15, 2022

Testing

Jan 1, 2022
Jan 1, 2022
Jan 2, 2022
Jan 3, 2022

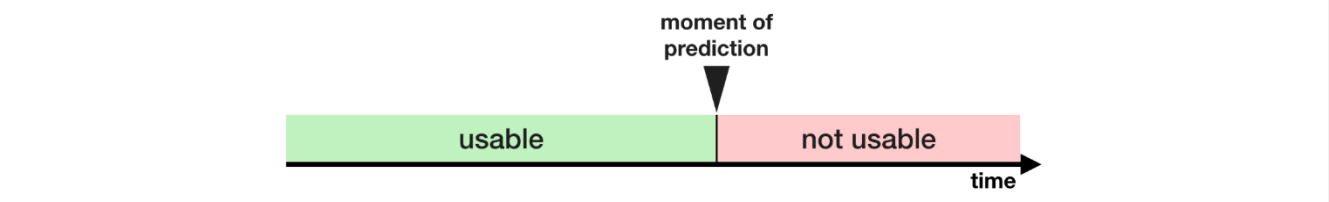
Training

Jan 4, 2022
Jan 4, 2022
Jan 8, 2022
Jan 15, 2022
Jan 15, 2022
Feb 5, 2022
Feb 18, 2022
Jan 15, 2022

Testing

Time Series splits

GOOD



Example

```
[24]: from imblearn.under_sampling import RandomUnderSampler

      rus = RandomUnderSampler(random_state=0)
      X_resampled, y_resampled = rus.fit_resample(X_train, y_train)

      X_resampled.shape
      y_resampled.shape
      np.bincount(y_resampled)

      evals_d.append(quick_evaluate_with_dt(X_resampled, X_test, y_resampled, y_test,  'Under Sample'))

[24]: (428, 8)

[24]: (428,)

[24]: array([214, 214], dtype=int64)
```

Feature-Target Correlation Filter

If feature is not at all correlated with target, can remove:

CID	Favorite Color	…	Default
1	Red		1
2	Purple		0
3	Purple		0
4	Blue		1
5	White		1
6	Green		0
7	Green		0
8	Gold		1
9	Yellow		0
10	Red		1
…			

- Numeric-Numeric: Pearson
- Categorical-Categorical: Chi-squared
- Numeric-Categorical: ANOVA

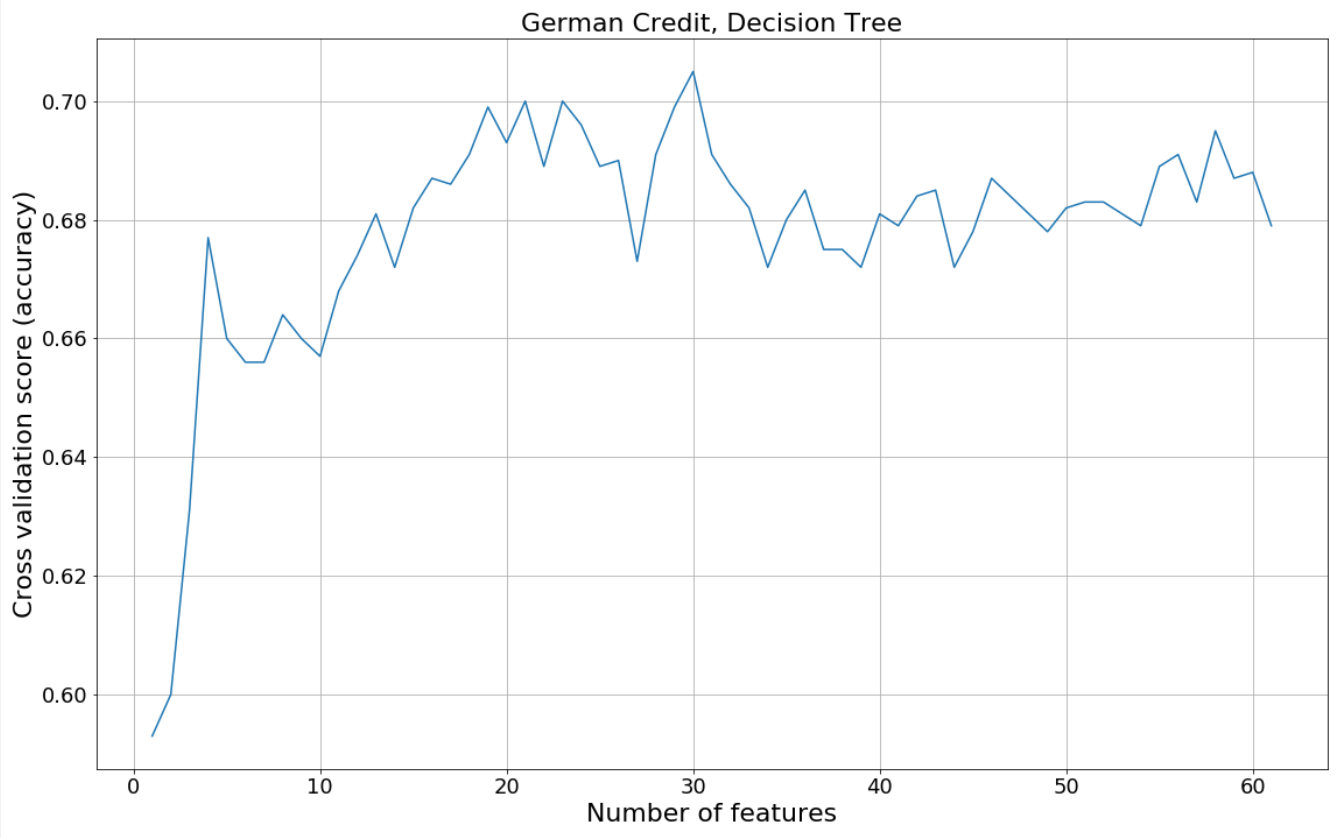
Variance Threshold Filter

Remove features with zero variance or low variance.

CID	Province	…
1	ON	
2	ON	
3	ON	
4	ON	
5	ON	
6	ON	
7	ON	
8	ON	
9	ON	
10	ON	
…		

Backward Selection, aka RFE

- Build model with all features; assess performance
- Remove the least-important feature, re-build and re-assess
  - If performance is the same, drop the feature
- Repeat



German Credit Example

[6]:
```
X_train.shape
```

[6]:
```
(800, 61)
```

[7]:
```
clf = DecisionTreeClassifier(random_state=0)
clf = clf.fit(X_train, y_train)
y_pred = clf.predict(X_test)
print(classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

           0       0.46      0.49      0.48        59
           1       0.78      0.76      0.77       141

    accuracy                           0.68       200
   macro avg       0.62      0.63      0.62       200
weighted avg       0.69      0.68      0.68       200
```

German Credit Example

[8]:
```
from sklearn.feature_selection import VarianceThreshold
sel = VarianceThreshold(threshold=(0.1));
sel = sel.fit(X_train);

X_train_new = sel.transform(X_train)
X_test_new = sel.transform(X_test)

X_train_new.shape

clf = DecisionTreeClassifier(random_state=0)
clf = clf.fit(X_train_new, y_train)
y_pred = clf.predict(X_test_new)
print(classification_report(y_test, y_pred))
```

[8]:
```
(800, 35)
              precision    recall  f1-score   support

           0       0.53      0.58      0.55        59
           1       0.82      0.79      0.80       141

    accuracy                           0.73       200
   macro avg       0.67      0.68      0.68       200
weighted avg       0.73      0.72      0.73       200
```

German Credit Example

[10]:
```
from sklearn.feature_selection import SelectKBest, chi2

sel = SelectKBest(chi2, k=10)
sel = sel.fit(X_train, y_train)

X_train_new = sel.transform(X_train)
X_test_new = sel.transform(X_test)

X_train_new.shape

clf = DecisionTreeClassifier(random_state=0)
clf.fit(X_train_new, y_train)
y_pred = clf.predict(X_test_new)
print(classification_report(y_test, y_pred))
```

[10]:
```
(800, 10)
```

[10]:
```
DecisionTreeClassifier(random_state=0)
              precision    recall  f1-score   support

           0       0.58      0.51      0.54        59
           1       0.80      0.84      0.82       141

    accuracy                           0.74       200
   macro avg       0.69      0.68      0.68       200
weighted avg       0.74      0.74      0.74       200
```

German Credit Example

[11]:
```
clf = DecisionTreeClassifier(random_state=0)
sel = RFE(estimator=clf, n_features_to_select=10)
sel = sel.fit(X_train, y_train)

X_train_new = sel.transform(X_train)
X_test_new = sel.transform(X_test)

X_train_new.shape

clf = DecisionTreeClassifier(random_state=0)
clf.fit(X_train_new, y_train)
y_pred = clf.predict(X_test_new)
print(classification_report(y_test, y_pred))
```

[11]:
```
(800, 10)
```

[11]:
```
DecisionTreeClassifier(random_state=0)
```

	precision	recall	f1-score	support
0	0.45	0.51	0.48	59
1	0.78	0.74	0.76	141
accuracy			0.68	200
macro avg	0.62	0.63	0.62	200
weighted avg	0.69	0.68	0.68	200