Machine learning for data science with scikit-learn

Gaël Varoquaux

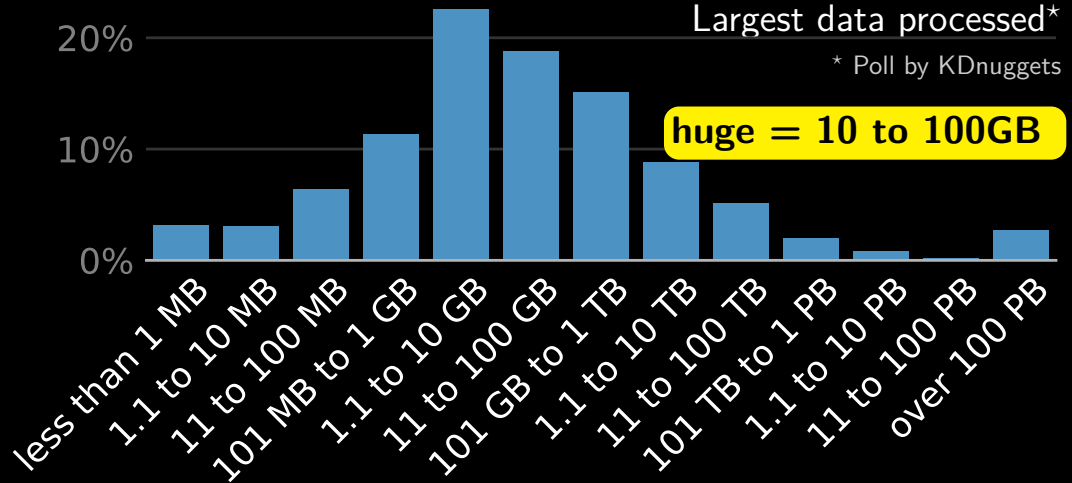
Why is scikit-learn still the #1 package?

Problem settings: tables are very common

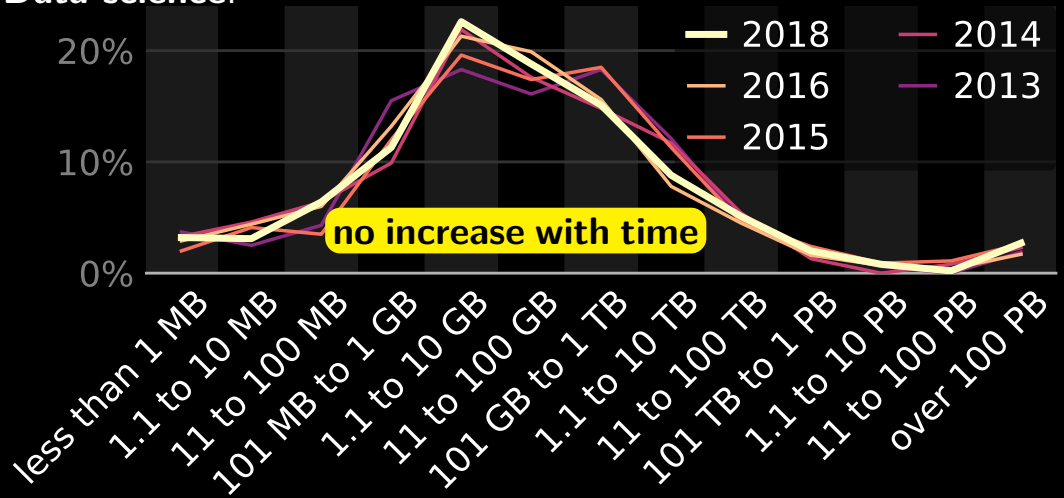
Unnamed: 0	AGY	NAME	MI	JOBCLASS	JC.TITLE	RACE	SEX	EMPTYPE	HIREDT	RATE	HRSWKD	ANNUAL
0	1	781	TEXAS HIGHER EDUCATION COORDINATING BOARD ...	C	7040	PROJECT MANAGER ...	BLACK	FEMALE	URF - UNCLASSIFIED REGULAR FULL-TIME	12/1/22	0.0000	40.0	75000.00
1	2	101	SENATE ...		7104	LEGISLATIVE PROFESSIONAL ...	WHITE	FEMALE	URF - UNCLASSIFIED REGULAR FULL-TIME	2/3/05	0.0000	41.0	65199.96
2	3	241	COMPTROLLER OF PUBLIC ACCOUNTS, JUDICIARY SECT...	J	JD25	JUDGE, RETIRED ...	WHITE	MALE	URP - UNCLASSIFIED REGULAR PART-TIME	2/1/20	75.9615	29.0	114549.84
3	5	771	SCHOOL FOR THE BLIND AND VISUALLY IMPAIRED ...	T	7354	LIBRARY ASSISTANT II ...	WHITE	MALE	CRP - CLASSIFIED REGULAR PART-TIME	10/7/99	0.0000	20.0	21865.68

Problem settings: Mid-sized datasets are very common

Data science:



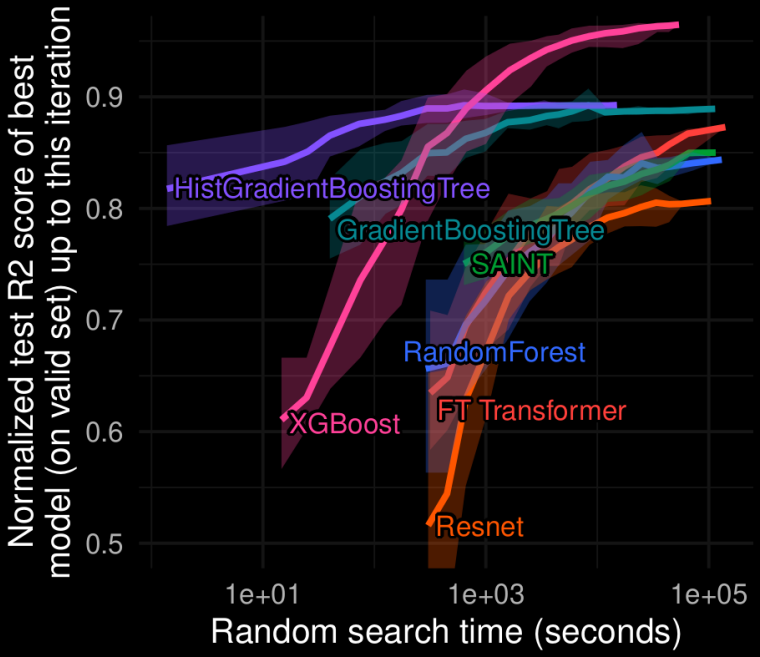
Data science:



Deep learning underperforms on data tables

Tree-based methods outperform tailored deep-learning architectures

[Grinsztajn... 2022]



1 **Scikit-learn reminders**

2 **Data transformation & pipeline**

1 Scikit-learn reminders

A central concept: **the estimator**

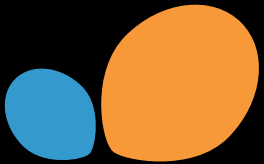
- Instanciated without data
- But specifying the parameters

```
from sklearn.neighbors import KNeighborsClassifier

classifier = KNeighborsClassifier(n_neighbors=2)

classifier.fit(X_train, Y_train)
Y_test = classifier.predict(X_test)
```

`n_neighbors`: model parameters



Training from data

```
estimator.fit(X_train, Y_train)
```

- X a data array with shape $n_{\text{samples}} \times n_{\text{features}}$
- y a numpy 1D array, of ints or float, with shape n_{samples}

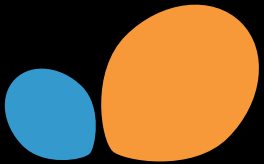
API: using a model

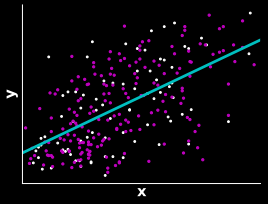
- Prediction: classification, regression

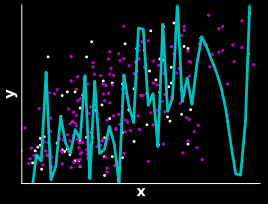
```
Y_test = estimator.predict(X_test)
```

- Transforming: dimension reduction, filter

```
X_new = estimator.transform(X_test)
```





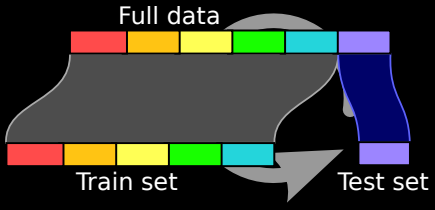


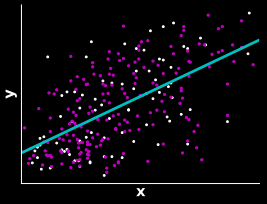
Only performance on new data can evaluate model predictions

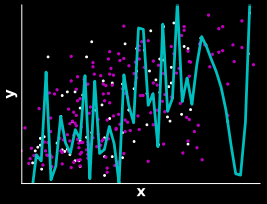
(a good model estimates $\mathbb{E}[y|X]$)

Cross-validation:

- Split the data (leave out 10%)
- Train model on a *train* set
- Evaluate prediction error on *test* set
- Repeat many times







Only performance on new data can evaluate model predictions

(a good model estimates $\mathbb{E}[y|X]$)

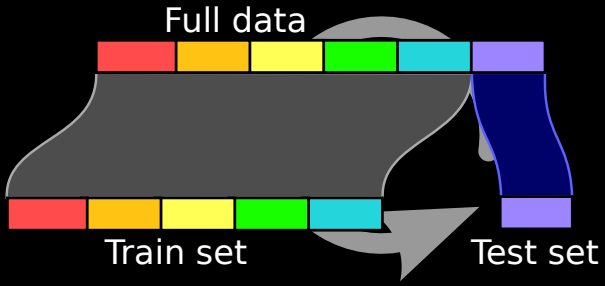
Common errors:

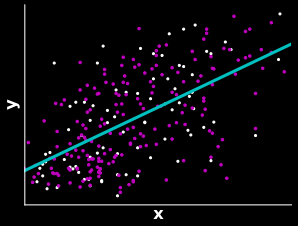
- *All* operations needed to fit the model must be done on *train* set only

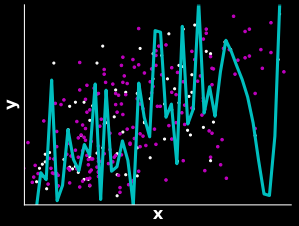
  data reduction, transformation, feature selection, parameter selection

- Testing several models with cross-validation and picking the best gives an optimistic and unreliable estimation of model performance.

```
scores = cross_val_score(estimator, X, y)
```

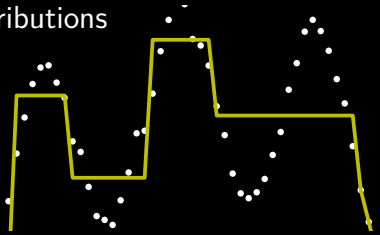


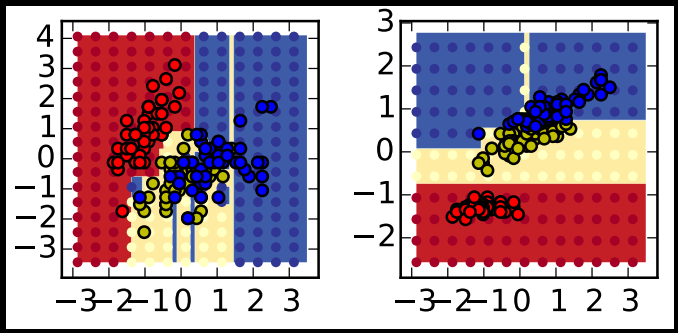




Tree models (*eg* for heterogeneous columnar data)

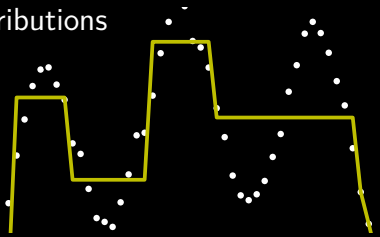
- Decision trees:
  robust to strange data distributions





1 Tree models (*eg* for heterogeneous columnar data)

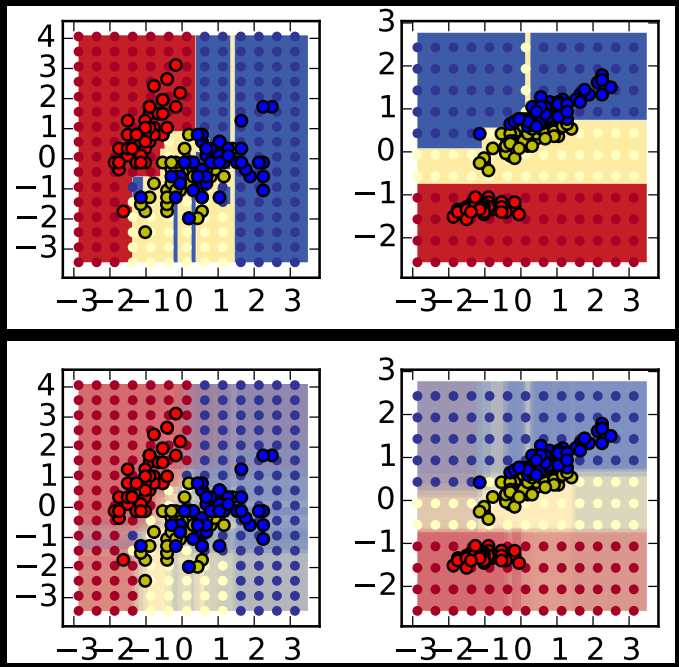
- Decision trees:
  robust to strange data distributions

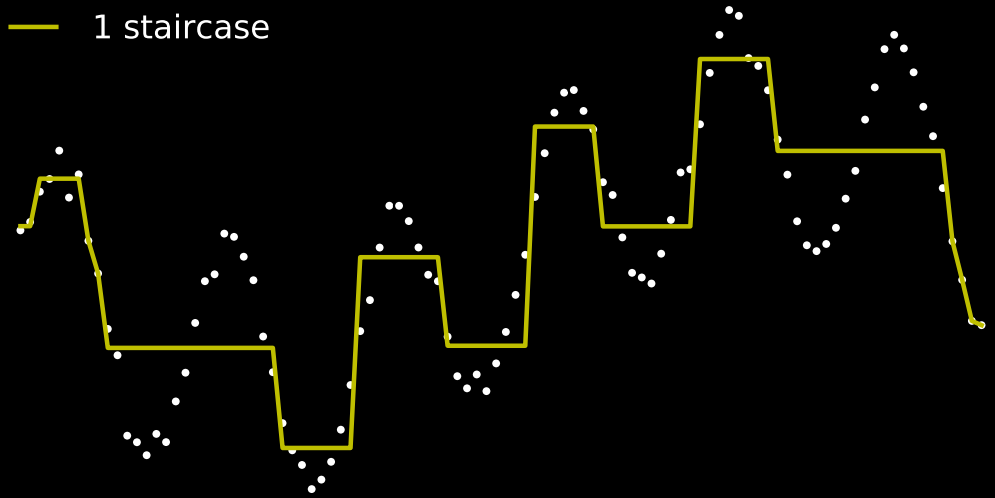


- Ensemble methods:
  combine many trees

Random forests

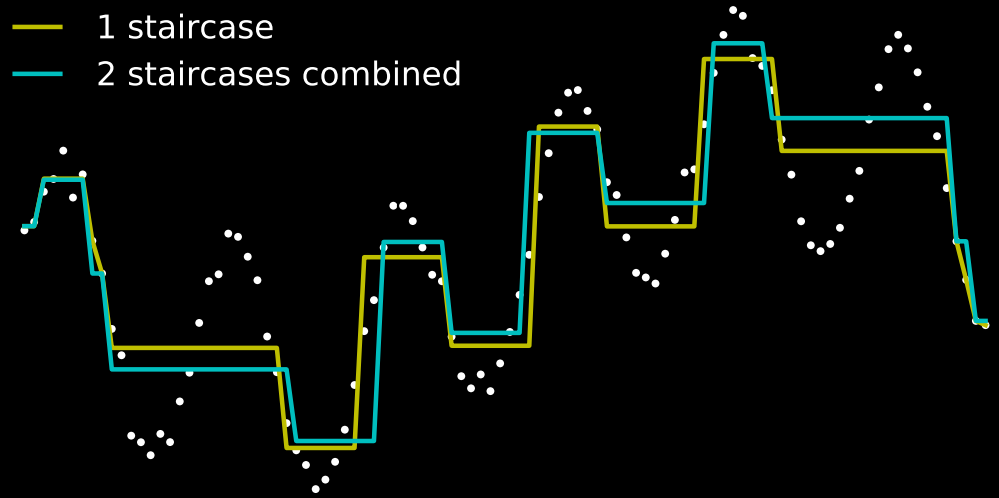
`sklearn.ensemble.RandomForestClassifier`





- Fit with a tree of depth 10

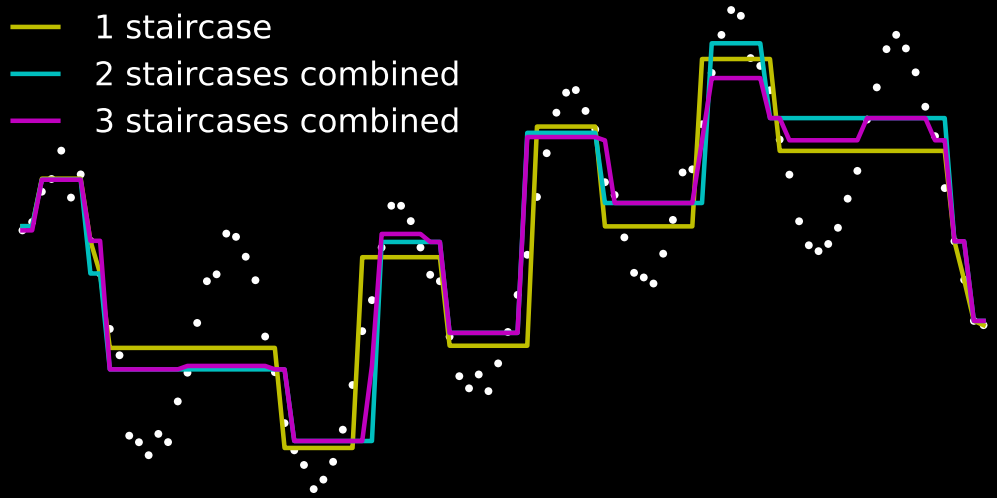
staircase of 10 constant values



- Fit with a tree of depth 10

staircase of 10 constant values

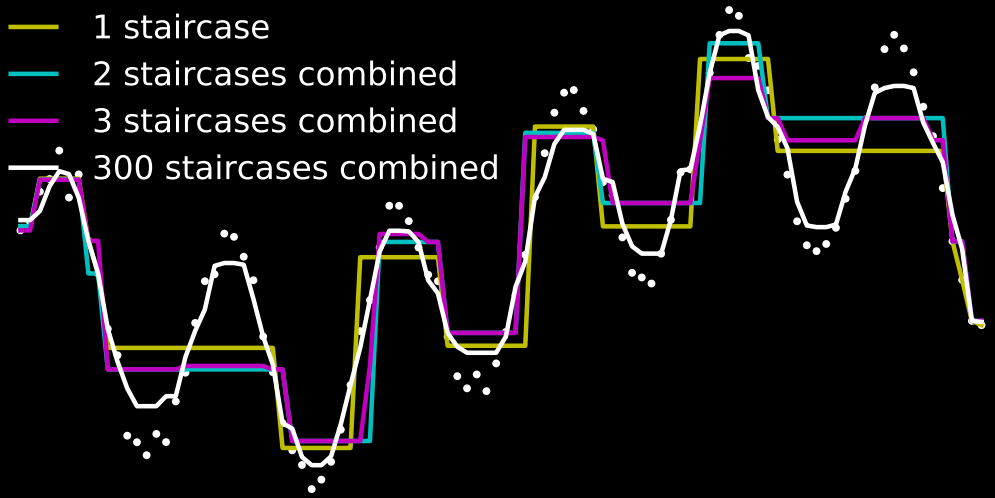
- Fit a new tree on errors – discounted by **learning rate**



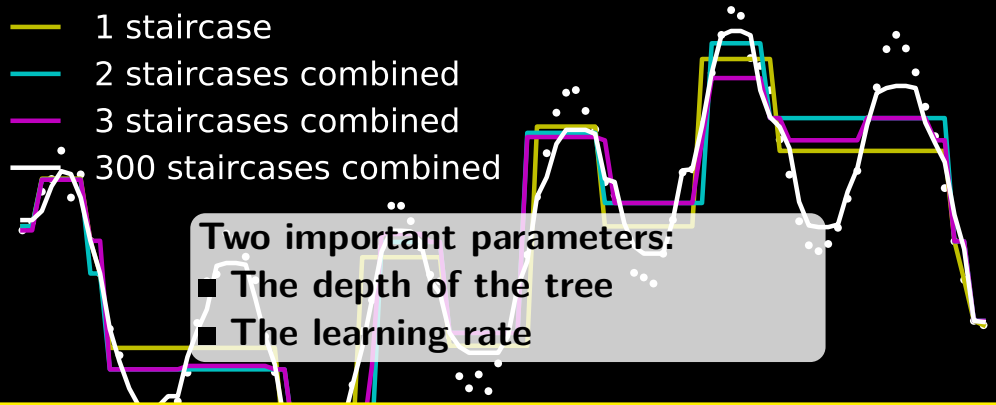
- Fit with a tree of depth 10

staircase of 10 constant values

- Fit a new tree on errors – discounted by **learning rate**



- Fit with a tree of depth 10 — staircase of 10 constant values
- Fit a new tree on errors – discounted by **learning rate**



Two important parameters:

- **The depth of the tree**
- **The learning rate**

`sklearn.ensemble.HistGradientBoostingClassifier`
deals naively with missing values.

- Fit with a tree of depth 10 — staircase of 10 constant values
- Fit a new tree on errors – discounted by **learning rate**

2 Data transformation & pipeline

Transforming data (pandas dataframes)
to numerical matrices (numpy arrays)
(preprocessing)

2 Data tables are not only numbers

```
df = pd.read_csv('employee_salary.csv')
```

Gender	Date Hired	Employee Position Title
M	09/12/1988	Master Police Officer
F	06/26/2006	Social Worker III
M	07/16/2007	Police Officer III
F	01/26/2000	Library Assistant I

Convert all values to numerical

2 Data tables are not only numbers

```
df = pd.read_csv('employee_salary.csv')
```

Gender	Date Hired	Employee Position Title
M	09/12/1988	Master Police Officer
F	06/26/2006	Social Worker III
M	07/16/2007	Police Officer III
F	01/26/2000	Library Assistant I

Convert all values to numerical

- Gender = categorical column: One-hot encode

```
one_hot_enc = sklearn.preprocessing.OneHotEncoder()
one_hot_enc.fit_transform(df[['Gender']])
```

Gender (M)	Gender (F)	...
1	0	
0	1	
1	0	
0	1	

One-hot encoder

```
one_hot_enc.fit(df[['Gender']])
X = one_hot_enc.transform(df[['Gender']])
```

1) store which categories are present

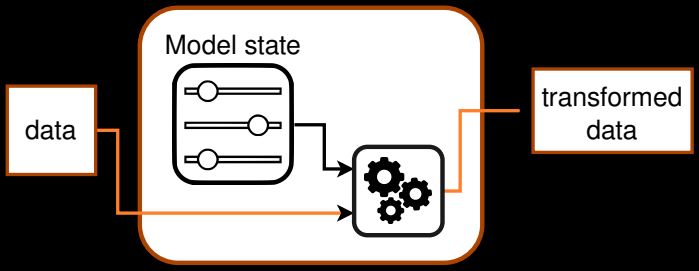
2) encode the data accordingly

Prefer to `pd.get_dummies` because columns are defined from train set, and do not change with test set

Separating fitting from transforming

- Avoids data leakage
- Can be used in a `Pipeline` and `cross_val_score`

`transformer.transform(data)`

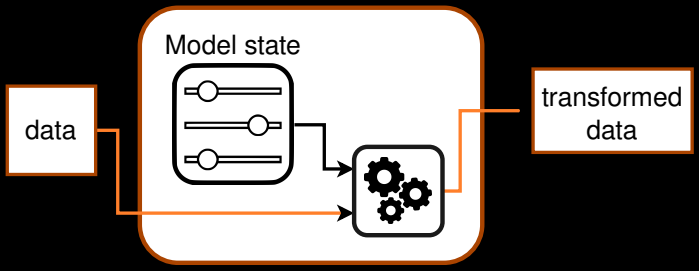


- learning the transformation (`.fit`) $\neq$ applying it (`.transform`)
  - Feature scaling
  - Transforming categorical variables...

Train time

```
ohe = OneHotEncoder()
ohe.fit(X_train, y_train)
X_train_encoded = ohe.transform(X_train, y_train)
estimator.fit(X_train_encoded)
```

```
transformer.transform(data)
```



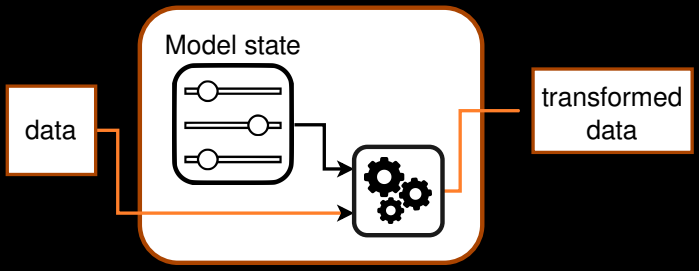
- learning the transformation (`.fit`) $\neq$ applying it (`.transform`)
  - Feature scaling
  - Transforming categorical variables...

Test time

```
X_test_encoded = ohe.transform(X_test)
y_pred = estimator.predict(X_test_encoded)
```

2 Data transformations: Transformers

`transformer.transform(data)`



- learning the transformation (`.fit`) $\neq$ applying it (`.transform`)
  - Feature scaling
  - Transforming categorical variables...

```
ohe = OneHotEncoder()
ohe.fit(X_train, y_train)
X_train_encoded = ohe.
    transform(X_train,
    y_train)
estimator.fit(
    X_train_encoded)
```

```
X_test_encoded = ohe.
    transform(X_test)
y_pred = estimator.predict(
    X_test_encoded)
```

Pipeline = transformation1 → (transformation2 ... →) predictor

```
pipe = make_pipeline(ohe, estimator)
```

Replace:

```
ohe = OneHotEncoder()
ohe.fit(X_train, y_train)
X_train_encoded = ohe.
    transform(X_train,
    y_train)
estimator.fit(
    X_train_encoded)
```

```
X_test_encoded = ohe.
    transform(X_test)
y_pred = estimator.predict(
    X_test_encoded)
```

with:

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

```
pipe.predict(X_test)
```

```
df = pd.read_csv('employee_salary.csv')
```

Gender	Date Hired	Employee Position Title
M	09/12/1988	Master Police Officer
F	06/26/2006	Social Worker III
M	07/16/2007	Police Officer III
F	01/26/2000	Library Assistant I

Convert all values to numerical

- Date: use pandas' datetime support

```
dates = pd.to_datetime(df['Date First Hired'])
# the values hold the data in secs
dates.values.astype(float)
```

Simplified object for dates – The `dirty_cat` module

`DatetimeEncoder`: features for different time regularity

```
from dirty_cat import DatetimeEncoder

date_trans = DatetimeEncoder()
X = date_trans.fit_transform(df['Date First Hired'])
```

month, day, hour, dayofweek

Simplified object for dates – The `dirty_cat` module

`DatetimeEncoder`: features for different time regularity

```
from dirty_cat import DatetimeEncoder

date_trans = DatetimeEncoder()
X = date_trans.fit_transform(df['Date First Hired'])
```

month, day, hour, dayofweek

Installing a new package

In the notebook: `%pip install dirty-cat`

For dates: FunctionTransformer

```
def date2num(date_str):
    out = pd.to_datetime(date_str).values.astype(np.float)
    return out.reshape((-1, 1)) # 2D output

date_trans = preprocessing.FunctionTransformer(
    func=date2num, validate=False)
X = date_trans.transform(df['Date First Hired']
```

Separating fitting from transforming

- Avoids data leakage
- Can be used in a `Pipeline` and `cross_val_score`

Applies different transformers to columns

- These can be complex pipelines

```
column_trans = compose.make_column_transformer(
      (one_hot_enc, ['Gender', 'Employee Position Title']),
      (date_trans, 'Date First Hired'),
   )

X = column_trans.fit_transform(df)
```

From DataFrame to array
with heterogeneous preprocessing & feature engineering

Applies different transformers to columns

- These can be complex pipelines

```
column_trans = compose.make_column_transformer(
      (one_hot_enc, ['Gender', 'Employee Position Title']),
      (date_trans, 'Date First Hired'),
   )

X = column_trans.fit_transform(df)
```

Benefit: model evaluation on dataframe

```
model = make_pipeline(column_trans, HistGradientBoostingClassifier)
scores = cross_val_score(model, df, y)
```

Applies different transformers to columns

- These can be complex pipelines

```
column_trans = compose.make_column_transformer(
      (one_hot_enc, ['Gender', 'Employee Position Title']),
      (date_trans, 'Date First Hired'),
   )

X = column_trans.fit_transform(df)
```

Simplified object – The `dirty_cat` module

`TableVectorizer`: applies transformers depending on columns types

```
from dirty_cat import TableVectorizer
tab_vec = TableVectorizer()

X = tab_vec.fit_transform(df)
```

"Automagic" choices: defaults can be improved

scikit-learn MOOC

`https://inria.github.io/scikit-learn-mooc`

Free, didactic, hands one, from zero to hero

More important sklearn-bits

Set your hyperparameters

- GridSearchCV, RandomSearchCV

  HalvingGridSearchCV and HalvingRandomSearchCV progressively more data

Model inspection

- `sklearn.inspection.permutation_importance`

Unsupervised learning

- PCA = data reduction
- K-means: clustering

Learn to read the docs

The MOOC

Module 1. The Predictive Modeling Pipeline

1. Tabular data exploration

   Getting familiar with Python dataframes

2. Fitting a scikit-learn model on numerical data

   Getting familiar with scikit-learn

3. Handling categorical data

   Getting familiar with data transformations

We will go over some "theory" and cover practice after

3 References I

L. Grinsztajn, E. Oyallon, and G. Varoquaux. Why do tree-based models still outperform deep learning on typical tabular data? In *Thirty-sixth Conference on Neural Information Processing Systems Datasets and Benchmarks Track*, 2022.