Introduction to machine learning with Python and scikit-learn

Gaël Varoquaux

machine learning in Python

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

1. The machine learning setting
2. Scikit-learn 101
3. Data transformation & pipeline
4. In depth with some estimators
5. Text mining

1 The machine learning setting

Adjusting models for prediction

1 A different statistical-modeling philosophy

- Focus on the output (predictions) of models
  not the components

  Example: $\texttt{attendance} = f(\texttt{context})$

  f could be anything

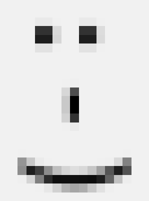
- In practice input data ($\texttt{context}$) is typically multiple "features"
  Example: $\texttt{context} = \{\texttt{temperature}, \texttt{time}, \texttt{weekday}\}$

- Traditional statistical modeling focuses on credible f

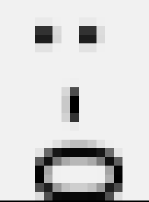
  $$\texttt{attendance} = w_1\ \texttt{temperature} + w_2\ \texttt{time} + w_3\ \texttt{weekday}$$

  Inference and reasonning on model parameters (w_1, w_2, w_3)

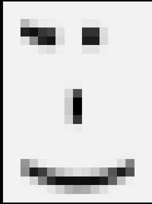
Face recognition



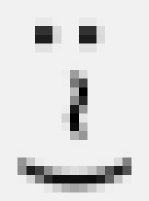
Andrew



Bill

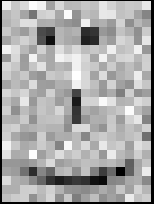


Charles

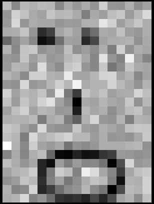


Dave

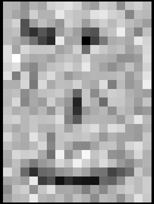
Face recognition



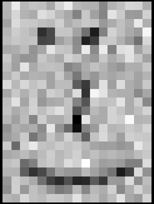
Andrew



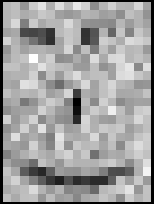
Bill



Charles



Dave



?

A simple method:

1 Store all the known (noisy) images and the names that go with them.

2 From a new (noisy) images, find the image that is most similar.

"Nearest neighbor" method

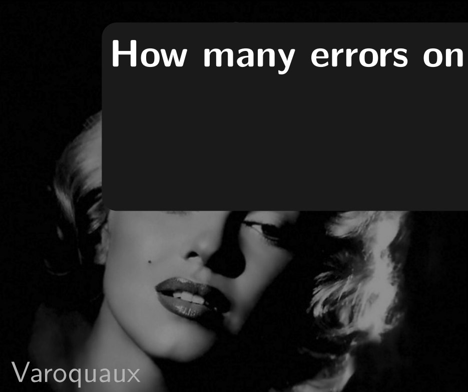
A simple method:

1 Store all the known (noisy) images and the names that go with them.

2 From a new (noisy) images, find the image that is most similar.

"Nearest neighbor" method

How many errors on already-known images?



1 Machine learning in a nutshell

A simple method:

1 Store all the known (noisy) images and the names that go with them.

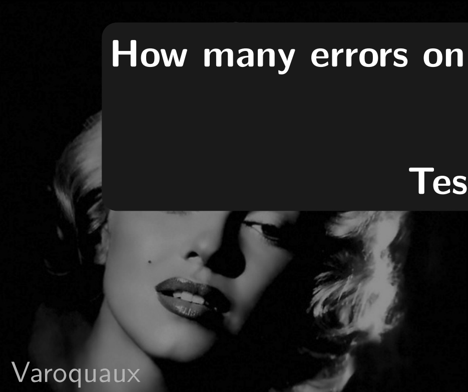
2 From a new (noisy) images, find the image that is most similar.

“Nearest neighbor” method

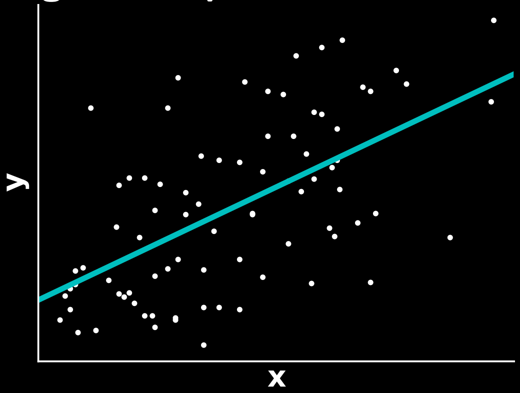
How many errors on already-known images?

... **0: no errors**

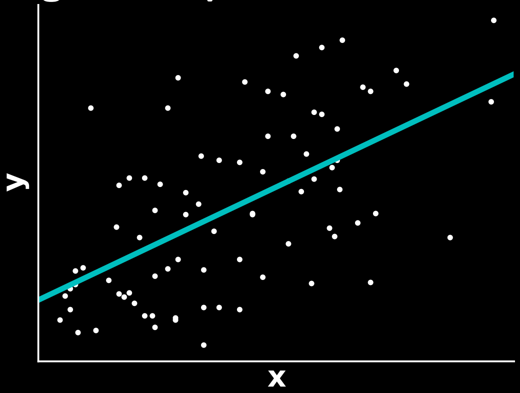
Test data $\neq$ Train data

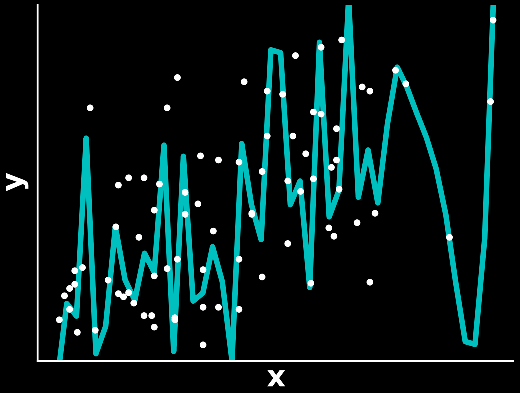


A single descriptor: 1 dimension



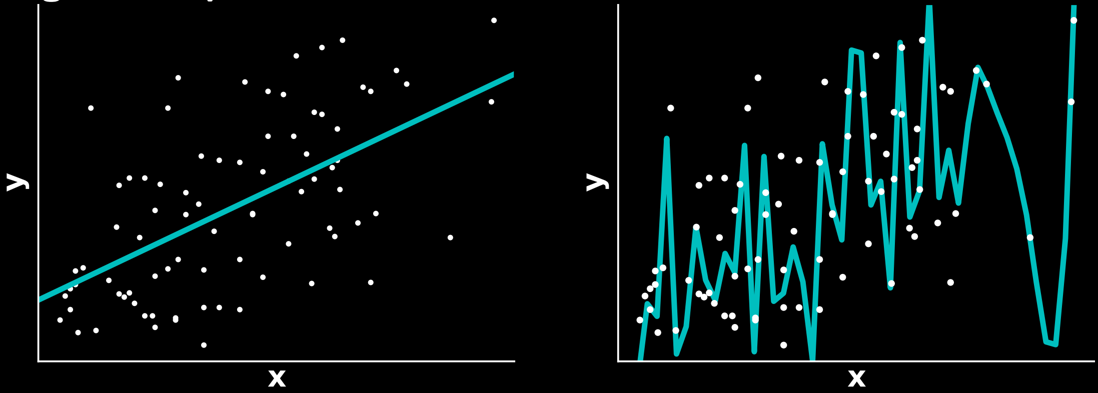
A single descriptor: 1 dimension





Which model to prefer?

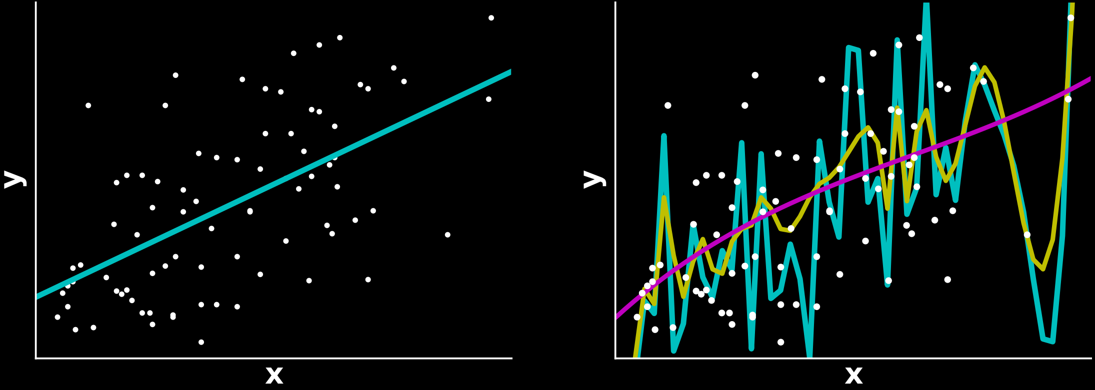
A single descriptor: 1 dimension



Problem of *"over-fitting"*

- Minimizing error is not always the best strategy (learning noise)
- Test data $\neq$ train data

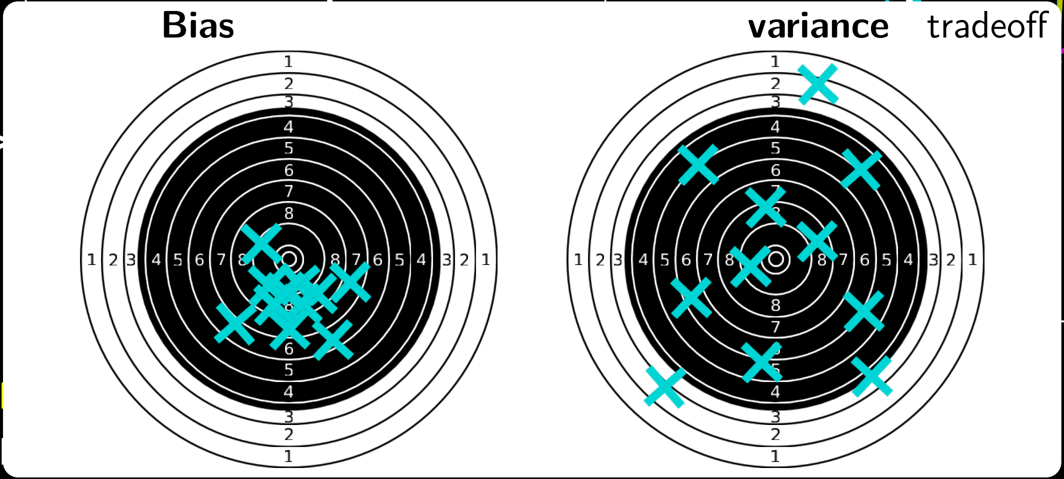
A single descriptor: 1 dimension



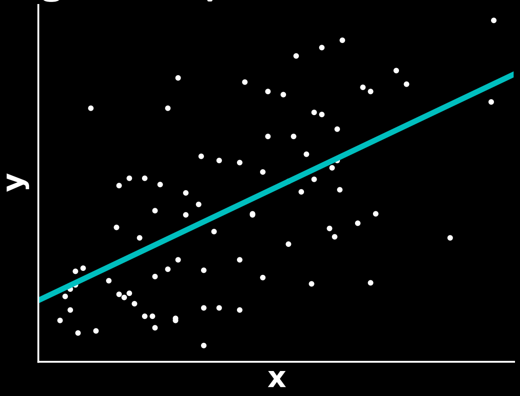
Prefer simple models = concept of *"regularization"*

Balance the number of parameters to learn with the amount of data

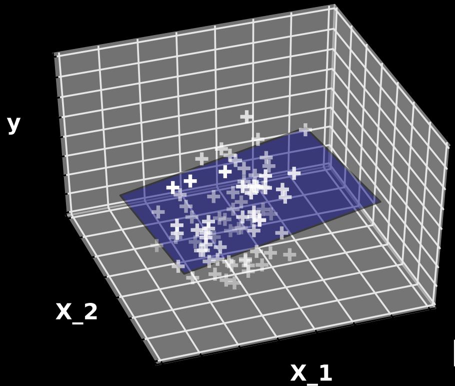
A single descriptor: 1 dimension



A single descriptor: 1 dimension

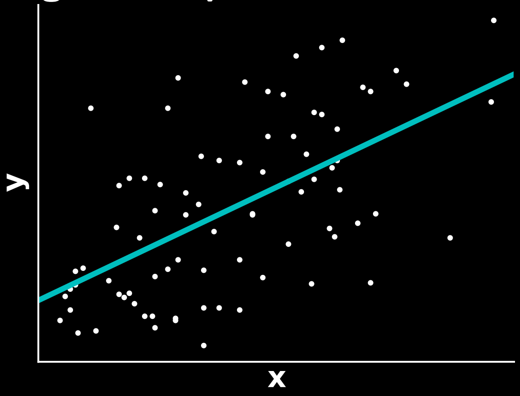


Two descriptors: 2 dimensions

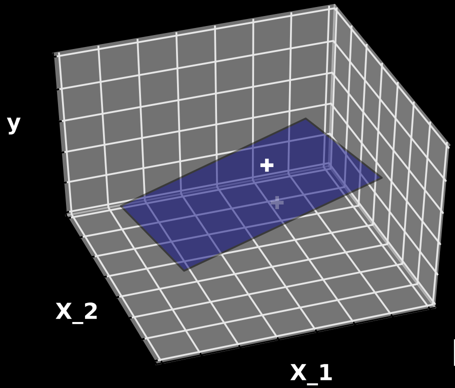


More parameters

A single descriptor: 1 dimension



Two descriptors: 2 dimensions

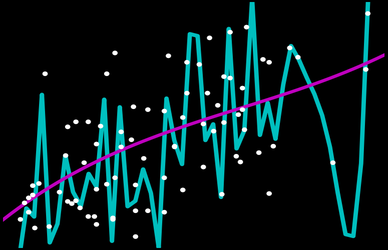


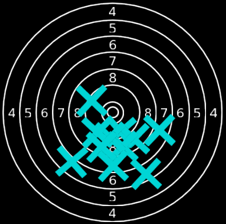
More parameters

⇒ Model with more parameters need much more data

"curse of dimensionality"

1 Some formalism: bias and regularization

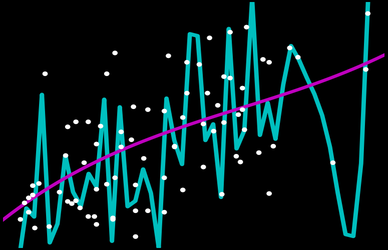


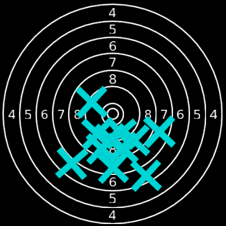


Settings: data $(\mathbf{X}, \mathbf{y})$, prediction $\mathbf{y} \sim f(\mathbf{X}, \mathbf{w})$

Our goal: $\min_{\mathbf{w}} \quad \|\mathbf{y} - f(\mathbf{X}, \mathbf{w})\|$

1 **Some formalism**: bias and regularization





Settings: data $(\mathbf{X}, \mathbf{y})$, prediction $\mathbf{y} \sim f(\mathbf{X}, \mathbf{w})$

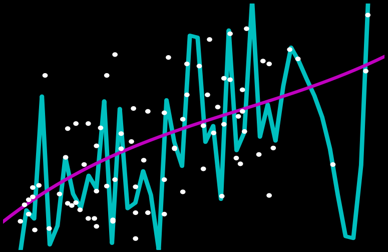
Our goal: $\min_{\mathbf{w}} \mathbb{E}[\|\mathbf{y} - f(\mathbf{X}, \mathbf{w})\|]$

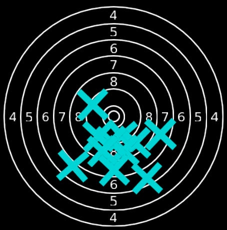
We only can measure $\|\mathbf{y} - f(\mathbf{X}, \mathbf{w})\|$

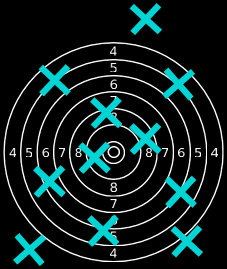
Prediction is very difficult, especially about the future.

Niels Bohr

1 **Some formalism**: bias and regularization







Settings: data $(\mathbf{X}, \mathbf{y})$, prediction $\mathbf{y} \sim f(\mathbf{X}, \mathbf{w})$

Our goal: $\underset{\mathbf{w}}{\text{minimize}}\ \mathbb{E}[\|\mathbf{y} - f(\mathbf{X}, \mathbf{w})\|\ ]$

We only can measure $\|\mathbf{y} - f(\mathbf{X}, \mathbf{w})\|$

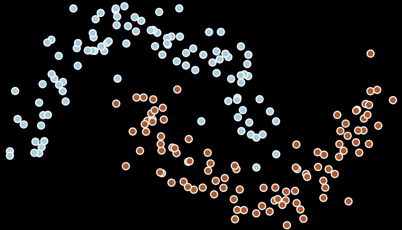
Solution: bias $\mathbf{w}$ to push toward a plausible solution

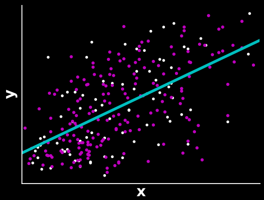
In a minimization framework: $\underset{\mathbf{w}}{\text{minimize}}\ \|\mathbf{y} - f(\mathbf{X}, \mathbf{w})\| + p(\mathbf{w})$

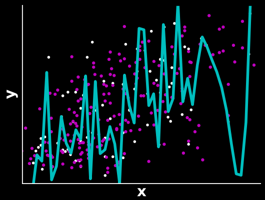
- **A forward model**: $\mathbf{y}_{\text{pred}} = f(\mathbf{X}, \mathbf{w})$
  Numerical rules to go from $\mathbf{X}$ to $\mathbf{y}$

- **A loss**, or data fit
  A measure of error between $\mathbf{y}_{\text{true}}$ and $\mathbf{y}_{\text{pred}}$
  Can be given by a noise model

- **Regularization**:
  Any way of restricting model complexity
  - by choices in the model
  - via a penalty



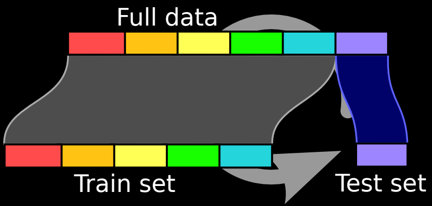


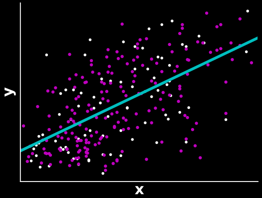


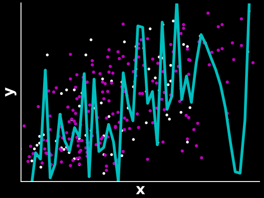
Only performance on new data can evaluate model predictions
(a good model estimates $\mathbb{E}[y|X]$)

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.

2 Scikit-learn 101

machine learning in Python

2 A tool in a wider Python ecosystem

A Python library

- To be combined:
  - `pandas`: dataframes
  - `matplotlib`, `seaborn`: plotting
  - `numpy`: numerical arrays
- Used in scripts or IPython notebooks

Simple usage

```
from sklearn import linear_model
classifier = linear_model.LogisticRegression()
classifier.fit(X_train, Y_train)
Y_test = classifier.predict(X_test)
```

2 API: specifying a model

A central concept: **the estimator**

- Instanciated without data
- But specifying the parameters

```
from sklearn.neighbors import KNearestNeighbors

estimator = KNearestNeighbors(n_neighbors=2)
```

`n_neighbors`: model parameters

Training from data

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

with:

- 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}

2 API: using a model

- Prediction: classification, regression

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

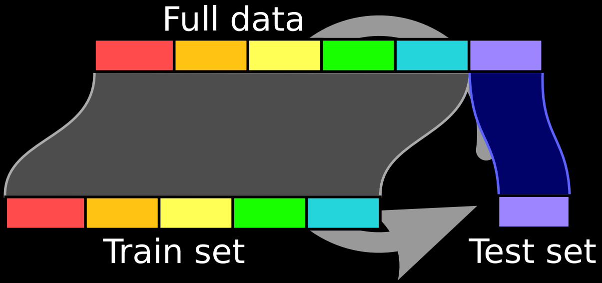
- Transforming: dimension reduction, filter

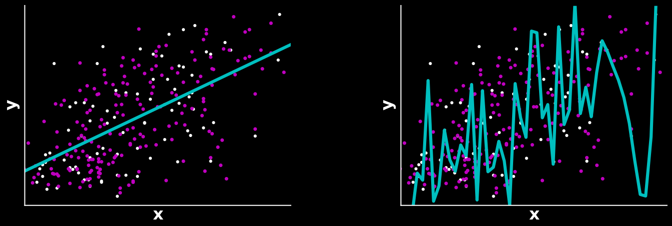
```
X_new = estimator.transform(X_test)
```

- Test score, density estimation

```
test_score = estimator.score(X_test)
```

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





3 Data transformation & pipeline

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

3 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

3 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']])
```

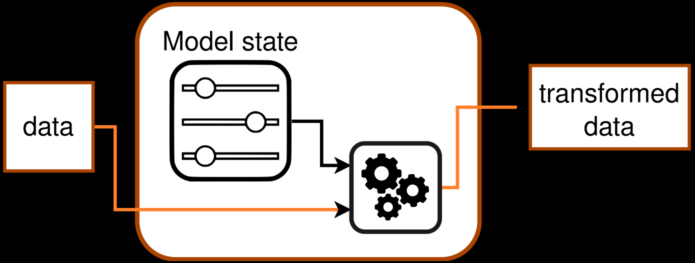
1) store which categories are present
2) encode the data accordingly

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)`



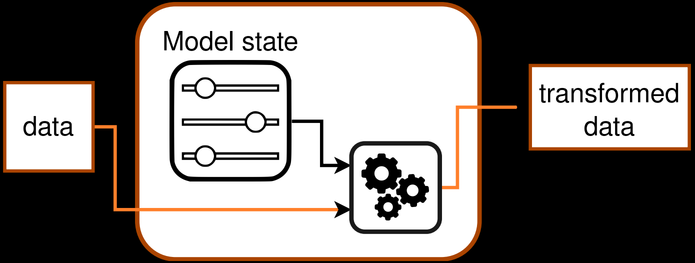
- learning the transformation (`.fit`) $\neq$ applying it (`.transform`)
  - Feature scaling
  - Transforming categorical variables...

Train time

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

3 Data transformations: Transformers

`transformer.transform(data)`



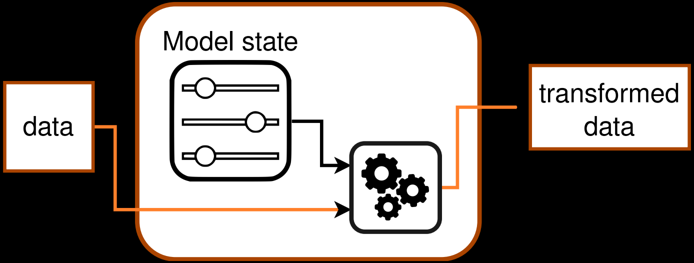
- learning the transformation (`.fit`) $\neq$ applying it (`.transform`)
  - Feature scaling
  - Transforming categorical variables...

Test time

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

3 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)
```

3 Chaining operations: The pipeline

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

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

Questions, difficulties?

4 In depth with some estimators

scikit learn

machine learning in Python

$$\text{is\_soup} = .5 \cdot \text{carrot} - 1.2 \cdot \text{flour} - .4 \cdot \text{sugar} + .6 \cdot \text{leak} \ldots$$

- Can handle large number of features
- "interpretable"

Interpretability pitfalls:

- Feature scaling matter:

  features with larger scale → smaller coefficient

- Coefficients are conditional relations

  they must be understand "all other features kept constant"

  eg wage decreases with age, keeping experience constant

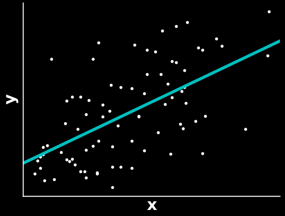
https://scikit-learn.org/stable/auto_examples/inspection/plot_linear_model_coefficient_interpretation.html

$$\text{is\_soup} = .5 \cdot \text{carrot} - 1.2 \cdot \text{flour} - .4 \cdot \text{sugar} + .6 \cdot \text{leak} \ldots$$

- Can handle large number of features
- "interpretable"

Regression:

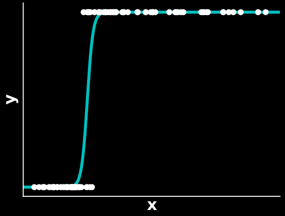
`sklearn.linear_model.Ridge`
`sklearn.linear_model.RidgeCV`



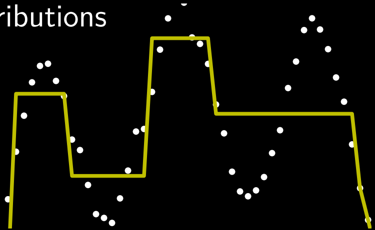
Classification: logistic regression

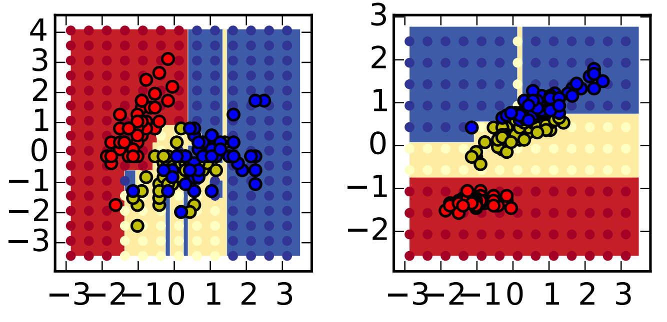
`sklearn.linear_model.LogisticRegression`
`sklearn.linear_model.LogisticRegressionCV`

'l2' and 'l1' penalties different solvers

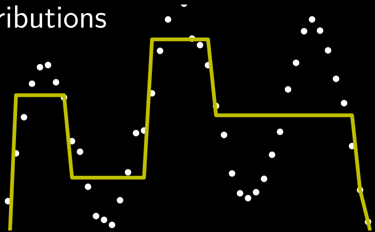


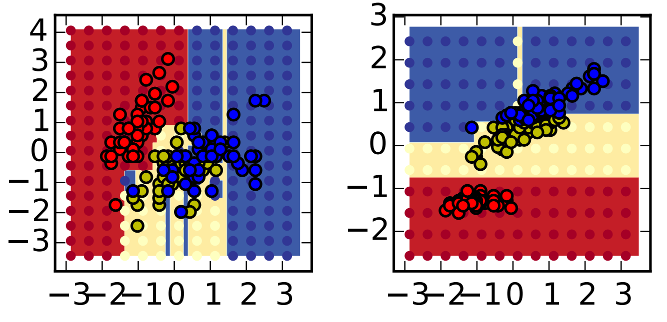
- Decision trees:
  robust to strange data distributions





- Decision trees:
  robust to strange data distributions

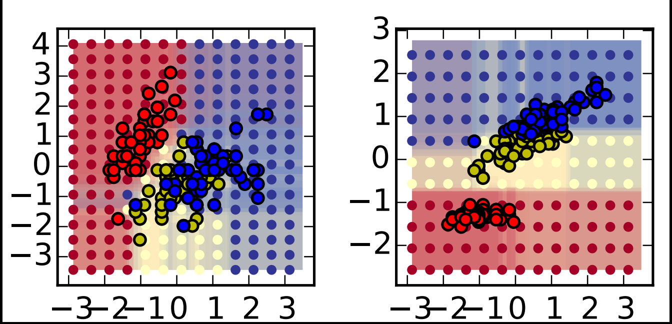




- Ensemble methods:
  combine many trees

  Random forests

  `sklearn.ensemble.RandomForestClassifier`



- Ensemble: combining many trees

Random forests

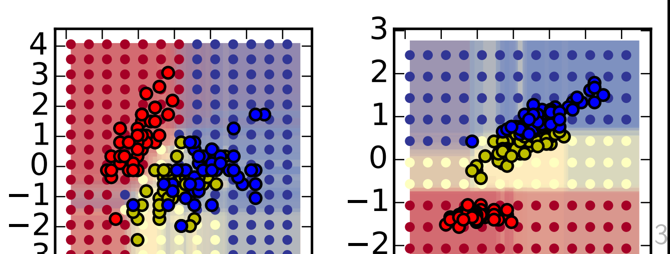
`sklearn.ensemble.RandomForestClassifier`

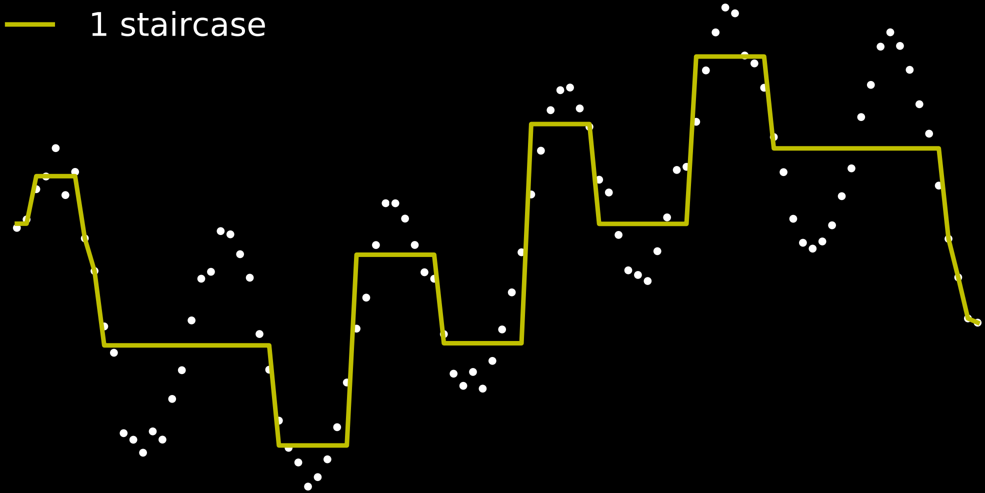
- Build many trees on random pertubation of the data
- Average decisions

More trees –higher `n_estimators` is better but more expensive

Boosted trees

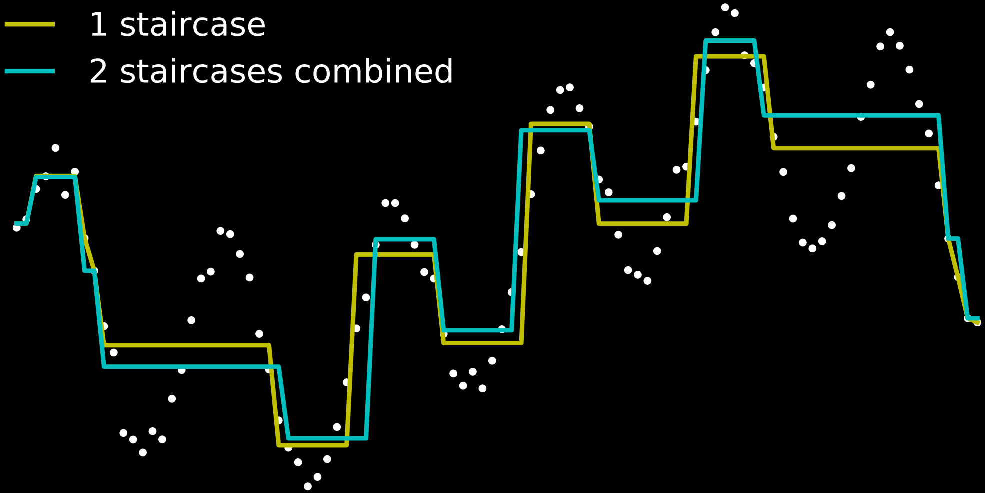
`sklearn.ensemble.HistGradientBoostingClassifier`





- Fit with a tree of depth 10

staircase of 10 constant values

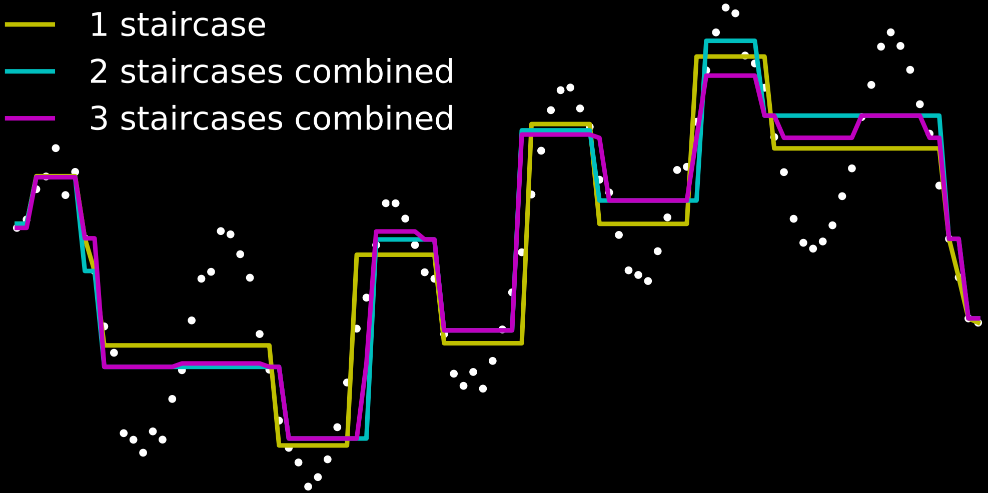


- Fit with a tree of depth 10

staircase of 10 constant values

- Fit a new tree on errors

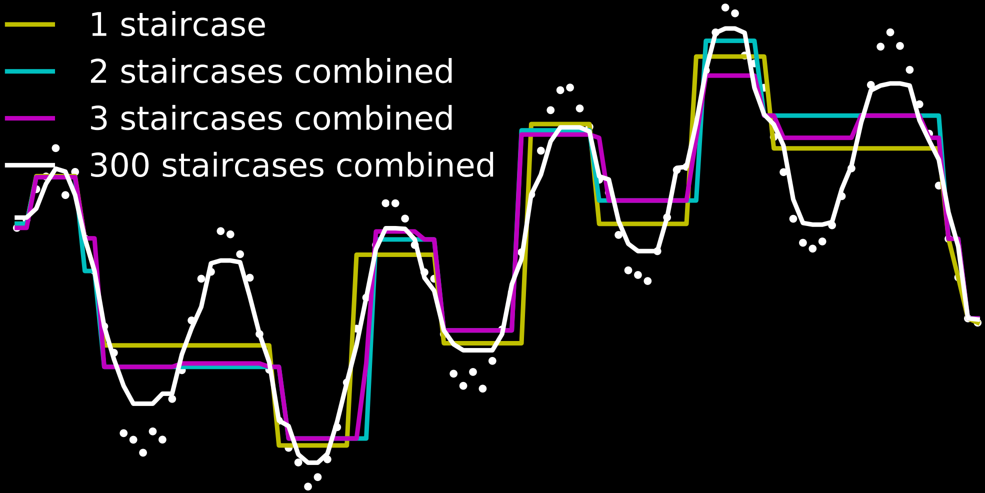
4 Gradient-boosted regression trees



- Fit with a tree of depth 10

staircase of 10 constant values

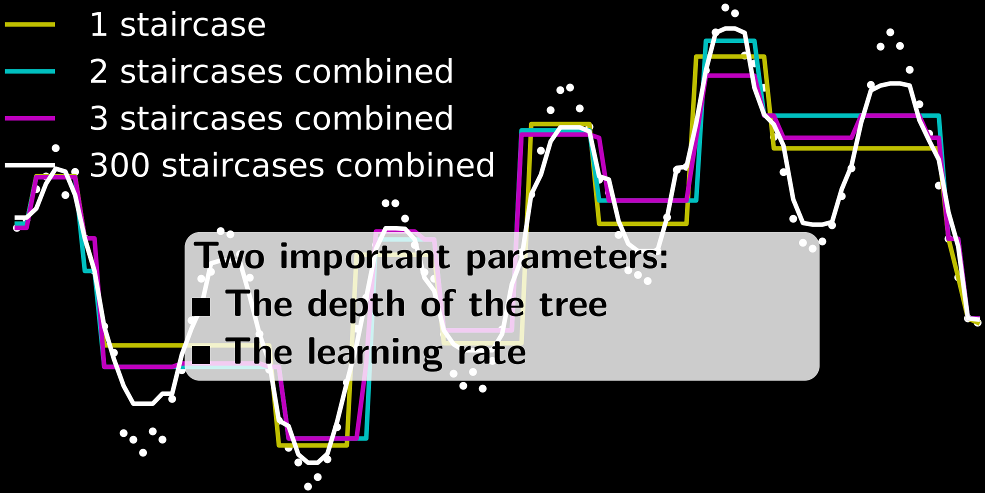
- Fit a new tree on errors



- Fit with a tree of depth 10

staircase of 10 constant values

- Fit a new tree on errors

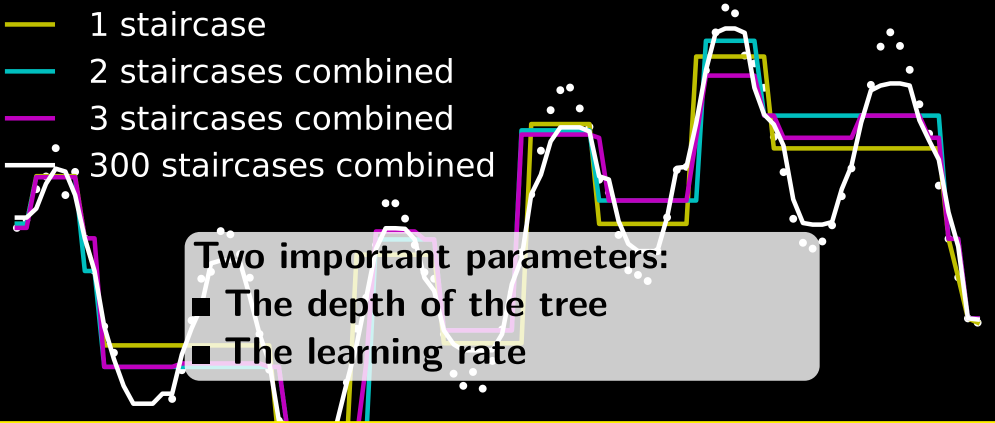


Two important parameters:
- **The depth of the tree**
- **The learning rate**

- Fit with a tree of depth 10

staircase of 10 constant values

- Fit a new tree on errors



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

4 Deep learning and representations

A function to decided if a cat is present?

Deep learning: build the function
by chaining transformations

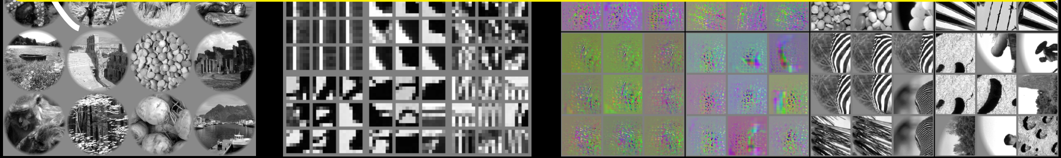
In practice:

- Reuse an existing pretrained architecture
- Use a linear model or tree model
  on an intermediate representation

Software: `keras`

Devil is in details:

same image resolution, same colors



- Great on complex natural signals

Linear estimators

- Can handle large number of features
- Typically a logistic regression

`sklearn.linear_model.SGDClassifier`

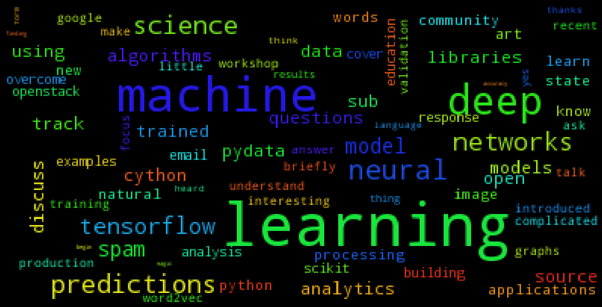
For on-line estimator

Naive Bayes

- Very good for many classes
- On-line estimator

+ chi2 feature selection

5 Text mining



Text as data

173 talks and counting:

How OpenStack makes Python better (and vice-versa)
Introduction to aiohttp
So you think your Python startup is worth $10 million...
SQLAlchemy as the backbone of a Data Science company
Learn Python The Fun Way
Scaling Microservices with Crossbar.io
If you can read this you don't need glasses

Let's find some common topics

173 talks and counting:

How OpenStack makes Python better (and vice-versa)

Introduction to aiohttp

So you think your Python startup is worth $10 million...

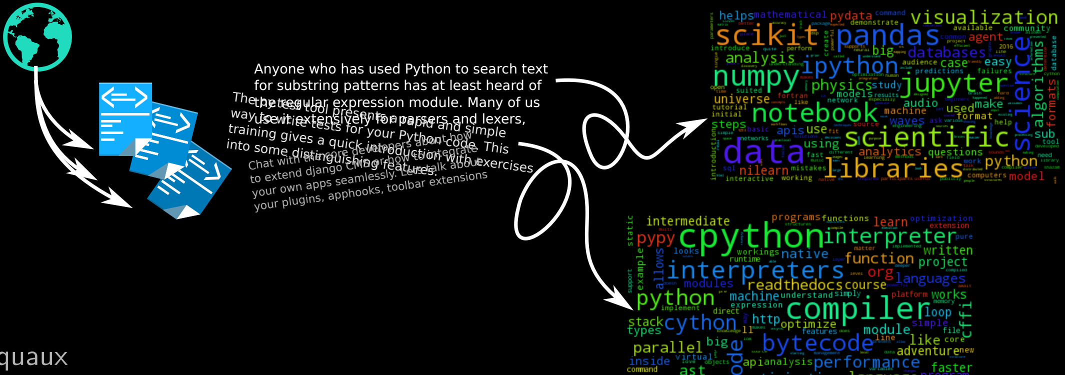
SQLAlchemy as the backbone of a Data Science company

Learn Python The Fun Way

Scaling Microservices with Crossbar.io

If you can read this you don't need glasses

Let's find some common topics



173 talks and counting:

How OpenStack makes Python better (and vice-versa)

Introduction to aiohttp

So you think your Python startup is worth $10 million...

SQLAlchemy as the backbone of a Data Science company

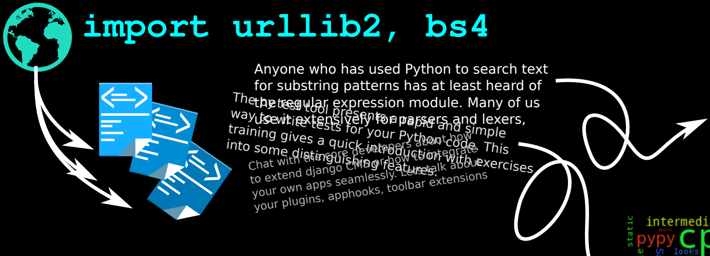
Learn Python The Fun Way

Scaling Microservices with Crossbar.io

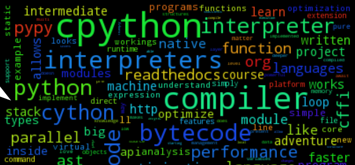
If you can read this you don't need glasses

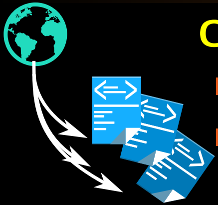
Let's find some common topics

```
import urllib2, bs4
```




```
import sklearn,
    wordcloud
```



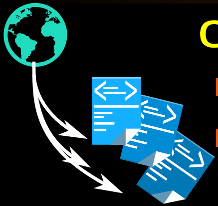


Crawl

- the schedule to get a list of titles and URLs
- talk pages to retrieve abstract and tags

`bs4`: beautiful soup, matchings on the DOM tree

5 Vectorizing webpages to numbers



Crawl

- the schedule to get a list of titles and URLs
- talk pages to retrieve abstract and tags

`bs4`: beautiful soup, matchings on the DOM tree

Common preparation steps

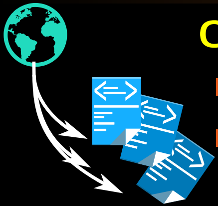
- Normalization

  "Man" $\rightarrow$ "man"

- Stemming

  "consult"
  "consultant" $\rightarrow$ "consult"
  "consulting"

Software: `nltk`, `spacy`

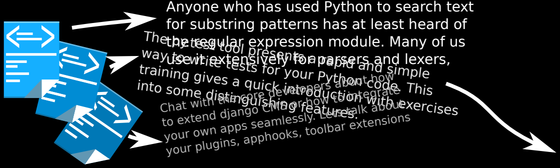


Crawl

- the schedule to get a list of titles and URLs
- talk pages to retrieve abstract and tags

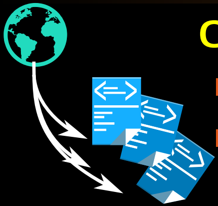
bs4: beautiful soup, matchings on the DOM tree

Vectorize



Term	Freq
a	20
can	10
code	4
is	14
module	3
profiling	2
performance	1
Python	9
the	18

5 Vectorizing webpages to numbers

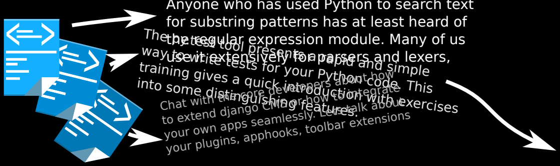


Crawl

- the schedule to get a list of titles and URLs
- talk pages to retrieve abstract and tags

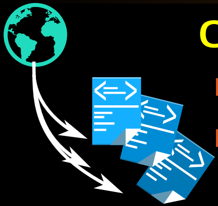
`bs4`: beautiful soup, matchings on the DOM tree

Vectorize



Term	Freq	All docs
a	20	1321
can	10	540
code	4	208
is	14	964
module	3	123
profiling	2	7
performance	1	6
Python	9	191
the	18	1450

5 Vectorizing webpages to numbers

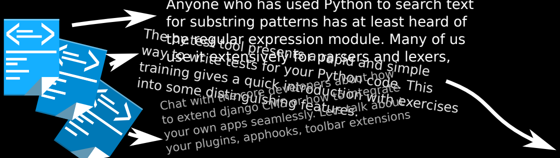


Crawl

- the schedule to get a list of titles and URLs
- talk pages to retrieve abstract and tags

bs4: beautiful soup, matchings on the DOM tree

Vectorize



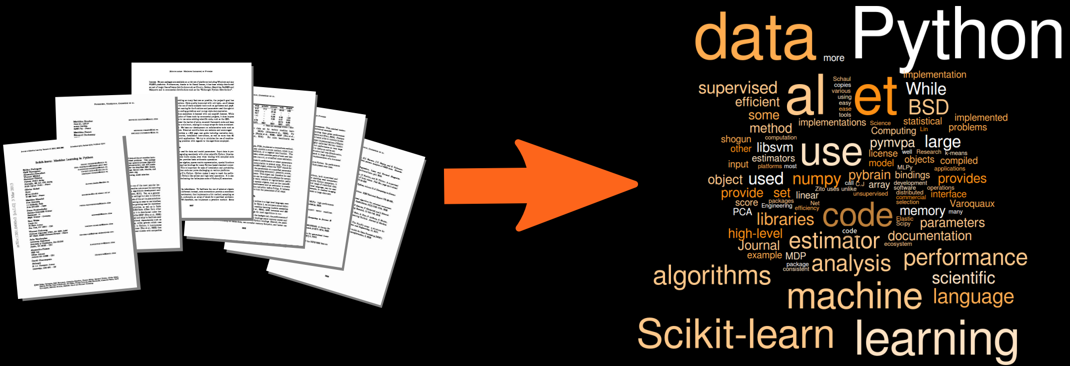
Term	Freq	All docs	Ratio
a	20	1321	.015
can	10	540	.018
code	4	208	.019
is	14	964	.014
module	3	123	.023
profiling	2	7	.286
performance	1	6	.167
Python	9	191	.047
the	18	1450	.012

TF-IDF in scikit-learn

`sklearn.feature_extraction.text.TfidfVectorizer`

From raw data to a sample matrix X

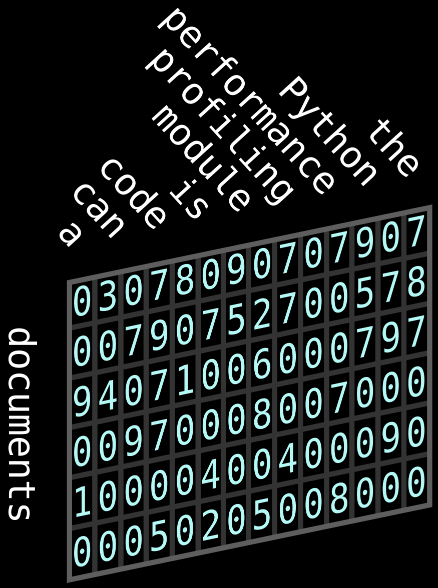
- For text data: counting word occurences
  - Input data: list of documents (string)
  - Output data: numerical matrix



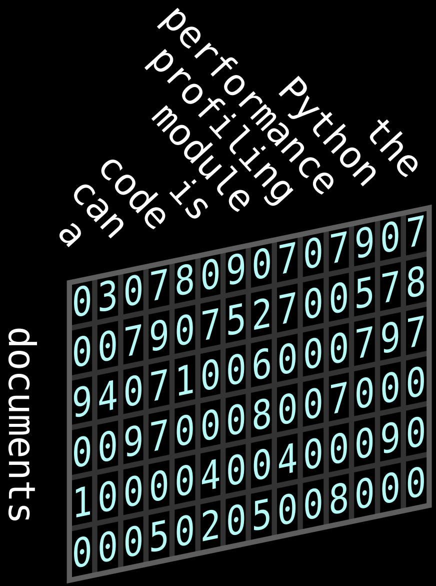
From raw data to a sample matrix X

- For text data: counting word occurences
  - Input data: list of documents (string)
  - Output data: numerical matrix

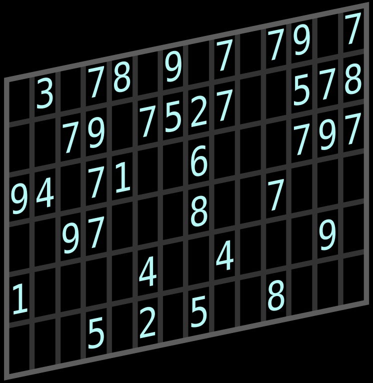
```
from sklearn.feature_extraction.text import
    TifdfVectorizer
vectorizer = TfidfVectorizer()

X = vectorizer.fit_transform(documents)
```



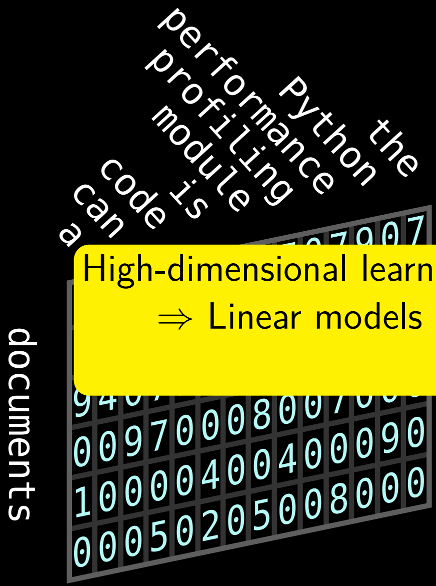
Term-document matrix



Term-document matrix



Can be a sparse matrix



High-dimensional learning problem

⇒ Linear models

(*eg* `LogisticRegression`)

Term-document matrix

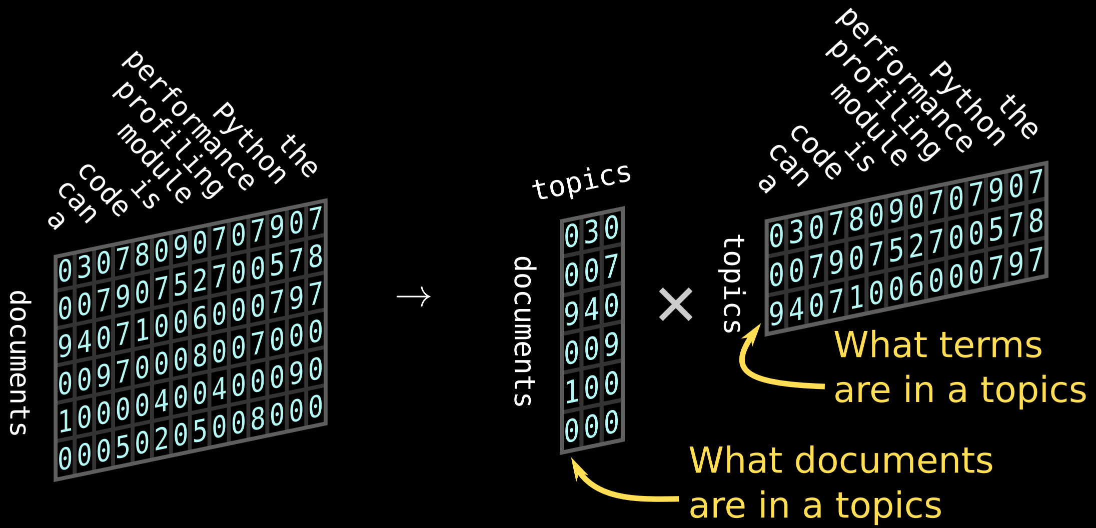
Can be a sparse matrix

Semantics

Semantics

Relations between words

5 Topic modeling: matrix factorization



A matrix factorization

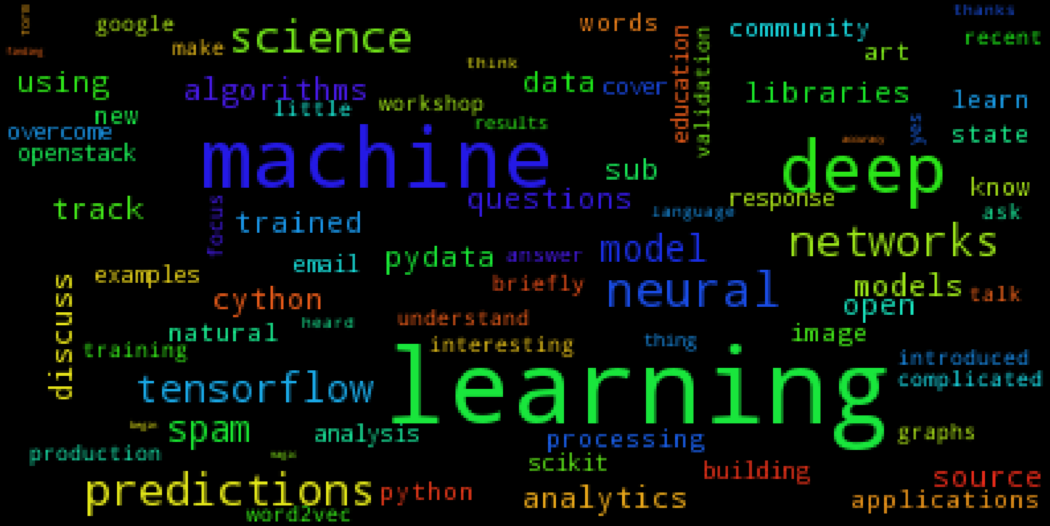
Often with non-negative constraints

`sklearn.decompositions.NMF`

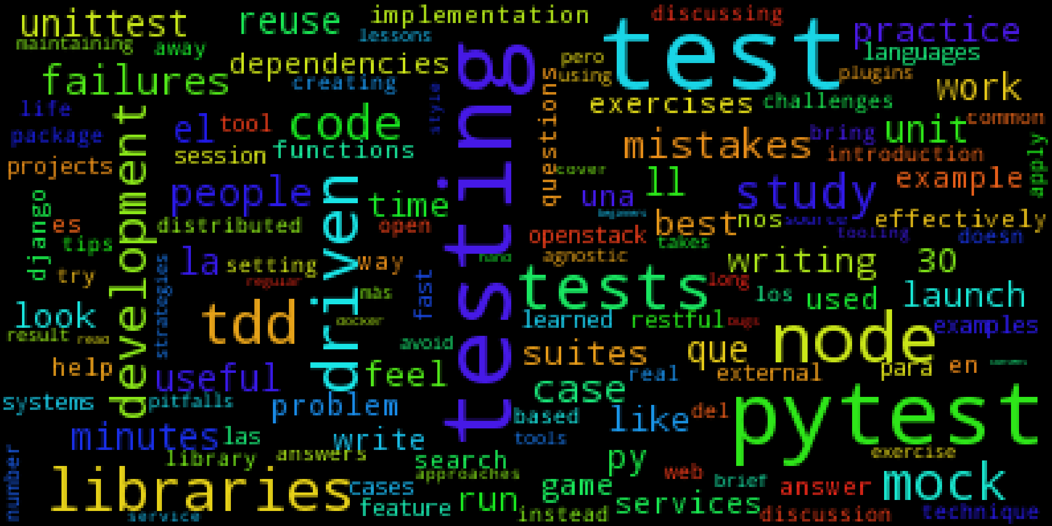
Topic 1


python
general
development
programming
software
architecture
practice
design
track
talk
best
code
use
performance
beginners
business
educational
language
clean
using
data
source
tooling
structures
things
microservices
education
learn
case
iot
devops
developers
writing
tools
way
ll
help
like
features
infrastructure
distributed
library
service
openstack
application
security
finally
functional
simple
lang
open
does
classes
systems
project
work
used
web
real
problems
fun
example
easy
class
examples
different
want
try
recent
basic
science
discuss
internet
descriptors
world
practical
restful
short
dynamic
based
module
game
review
right
people
years
cross
really
api
good
scaling
languages
packages
time
new
community
make
documentation
important
platform
just
start
don
metrics
working
study
concepts
patching

Topic 2


machine
learning
deep
networks
neural
science
predictions
tensorflow
algorithms
questions
model
models
trained
cython
spam
analytics
processing
libraries
community
data
cover
education
validation
using
track
discuss
openstack
overcome
new
google
make
words
art
recent
learn
state
know
ask
response
language
sub
workshop
little
results
think
pydata
answer
email
examples
briefly
understand
natural
heard
training
interesting
thing
image
open
talk
introduced
complicated
graphs
source
applications
building
scikit
analysis
production
python
word2vec
focus
thanks

Topic 3


testing
test
pytest
node
libraries
driven
development
tests
tdd
mock
unittest
reuse
implementation
failures
dependencies
code
practice
work
unit
mistakes
study
people
exercises
languages
writing
launch
used
suites
que
case
like
useful
minutes
problem
feel
time
write
search
run
services
game
examples
example
effectively
best
openstack
session
functions
tool
el
una
la
look
help
systems
projects
package
life
maintaining
away
creating
discussing
plugins
challenges
common
introduction
apply
question
using
pero
distributed
open
setting
way
tips
es
django
try
result
fast
strategies
pitfalls
library
answers
approaches
cases
feature
instead
discussion
technique
answer
web
brief
py
tools
based
del
exercise
external
real
para
en
learned
restful
los
long
30
ll
nos
agnostic
takes
tooling
doesn
las
number
service
avoid
lessons
discover
style

5 On the EuroPython abstracts

Distributional semantics: meaning of words

"You shall know a word by the company it keeps"

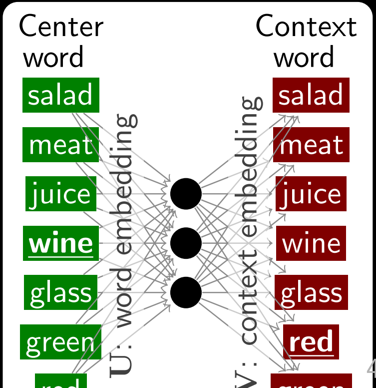
Firth, 1957

Example: **A glass of red _ _ _, please**

Could be **wine** maybe **juice**?

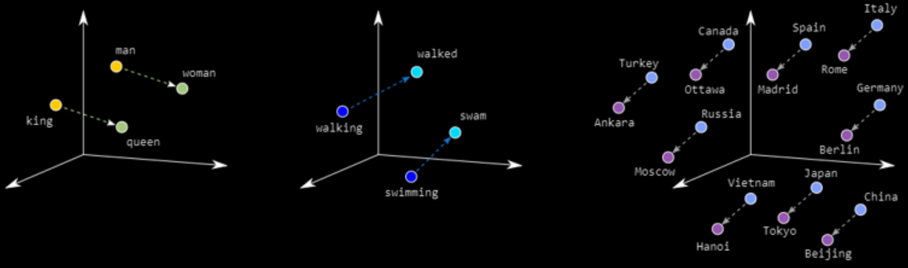
wine and **juice** have related meanings

Embed words in vector space
so that close-by vectors correspond to
equally-likely contexts



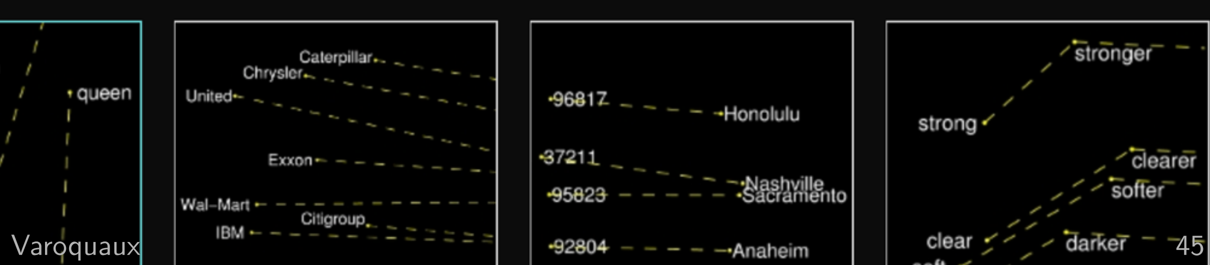
Word2vec

FastText: robust to typos and new words



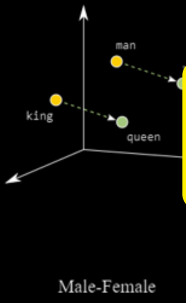
Male-Female Verb Tense Country-Capital

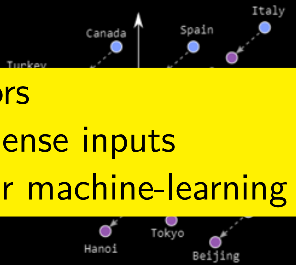
GloVe



Word2vec

FastText: robust to typos and new words





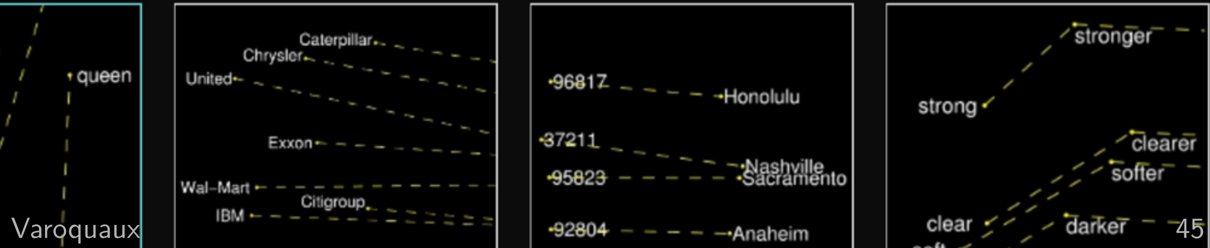
Transform words into vectors

⇒ Low-dimensional, dense inputs

⇒ richer machine-learning models

Software: gensim, fasttext

GloVe



Sequence models

Right language models: predict the next word

Recurrent Neural Network:

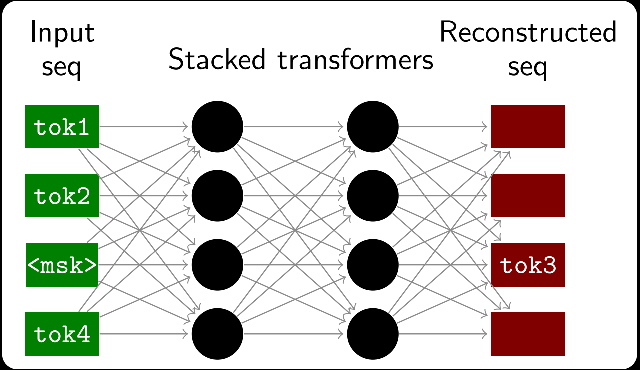
$$\text{Probability}((n+1)^{th}\text{ word }|n^{th},(n-1)^{th},\ldots)$$
$$= f\left(n^{th}\text{ word, Probability}(n^{th}\text{ word }|(n-1)^{th},(n-2)^{th},\ldots)\right)$$

Challenge: long-distance links $\Rightarrow$ LSTM

Also for left language models

Importance of language models predicting words
Difficulty of capturing long-distance relationships

Masked language models



Extracts internal representations of word sequences

Software: Huggingface transformers

for longer texts, grammatical structure, distant syntax

5 References I

[Gribonval(2011)] R. Gribonval.
Should penalized least squares regression be interpreted as maximum a posteriori estimation?
IEEE Transactions on Signal Processing, 59(5):2405–2410, 2011.