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
A different statistical-modeling philosophy

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

  \[ \text{Example: } \text{attendance} = f(\text{context}) \]

  \( f \) could be anything

- In practice input data (context) is typically multiple “features”

  \[ \text{Example: } \text{context} = \{ \text{temperature}, \text{time}, \text{weekday} \} \]

- Traditional statistical modeling focuses on credible \( f \)

  \[ \text{attendance} = w_1 \text{temperature} + w_2 \text{time} + w_3 \text{weekday} \]

  Inference and reasoning on model parameters \( (w_1, w_2, w_3) \)
Machine learning in a nutshell: an example

Face recognition

Andrew  Bill  Charles  Dave
Machine learning in a nutshell: an example

Face recognition

Andrew  Bill  Charles  Dave
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
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) image, find the image that is most similar.

“Nearest neighbor” method

How many errors on already-known images?

Varoquaux
# 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?

<table>
<thead>
<tr>
<th>...</th>
<th>0: no errors</th>
</tr>
</thead>
</table>

Test data \(\neq\) Train data
Machine learning in a nutshell: regression

A single descriptor: 1 dimension
Machine learning in a nutshell: regression

A single descriptor: 1 dimension

Which model to prefer?
Machine learning in a nutshell: regression

A single descriptor: 1 dimension

Problem of “over-fitting”
- Minimizing error is not always the best strategy (learning noise)
- Test data ≠ train data
Machine learning in a nutshell: regression

A single descriptor: 1 dimension

Prefer simple models
Balance the number of parameters to learn with the amount of data

= concept of “regularization”
Machine learning in a nutshell: regression

A single descriptor: 1 dimension

Bias

variance
tradeoff

Prefer simple models = concept of "regularization"

Balance the number of parameters to learn with the amount of data
Machine learning in a nutshell: regression

A single descriptor: 1 dimension

Two descriptors: 2 dimensions

More parameters
Machine learning in a nutshell: regression

A single descriptor: 1 dimension

Two descriptors: 2 dimensions

⇒ Model with more parameters need much more data

“curse of dimensionality”
Some formalism: bias and regularization

**Settings:** data $(X, y)$, prediction $y \sim f(X, w)$

**Our goal:** minimize $\min_w \|y - f(X, w)\|$
Some formalism: bias and regularization

Settings: data \((X, y)\), prediction \(y \sim f(X, w)\)

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

We only can measure \(\|y - f(X, w)\|\)

Prediction is very difficult, especially about the future.

Niels Bohr
**Some formalism:** bias and regularization

**Settings:** data \((X, y)\), prediction \(y \sim f(X, w)\)

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

We only can measure \(\|y - f(X, w)\|\)

**Solution:** bias \(w\) to push toward a plausible solution

In a minimization framework:

\[
\underset{w}{\text{minimize}} \; \|y - f(X, w)\| + p(w)
\]
1 **Summary**: elements of a machine-learning method

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

- **A loss**, or data fit
  A measure of error between \( y_{\text{true}} \) and \( y_{\text{pred}} \)
  Can be given by a noise model

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

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

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

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.
Scikit-learn 101

machine learning in Python

Varoquaux
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

```python
from sklearn import linear_model
classifier = linear_model.LogisticRegression()
classifier.fit(X_train, Y_train)
Y_test = classifier.predict(X_test)
```
A central concept: **the estimator**

- Instanciated without data
- But specifying the parameters

```python
from sklearn.neighbors import KNearestNeighbors

estimator = KNearestNeighbors(n_neighbors=2)
```

`n_neighbors`: model parameters
Training from data

\[ \text{estimator.fit}(X_{\text{train}}, Y_{\text{train}}) \]

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 \( n_{\text{samples}} \)
API: using a model

- Prediction: classification, regression
  \[ Y_{test} = \text{estimator} \cdot \text{predict} (X_{test}) \]

- Transforming: dimension reduction, filter
  \[ X_{new} = \text{estimator} \cdot \text{transform} (X_{test}) \]

- Test score, density estimation
  \[ \text{test\_score} = \text{estimator} \cdot \text{score} (X_{test}) \]
Model evaluation: cross-validation

\[ \text{scores} = \text{cross}_\text{val}_\text{score}(\text{estimator}, \; X, \; y) \]
3 Data transformation & pipeline

Transforming data (pandas dataframes) to numerical matrices (numpy arrays) (preprocessing)
Data tables are not only numbers

```python
import pandas as pd

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

<table>
<thead>
<tr>
<th>Gender</th>
<th>Date Hired</th>
<th>Employee Position</th>
</tr>
</thead>
<tbody>
<tr>
<td>M</td>
<td>09/12/1988</td>
<td>Master Police Officer</td>
</tr>
<tr>
<td>F</td>
<td>06/26/2006</td>
<td>Social Worker III</td>
</tr>
<tr>
<td>M</td>
<td>07/16/2007</td>
<td>Police Officer III</td>
</tr>
<tr>
<td>F</td>
<td>01/26/2000</td>
<td>Library Assistant I</td>
</tr>
</tbody>
</table>

Convert all values to numerical
Data tables are not only numbers

def = pd.read_csv('employee_salary.csv')

<table>
<thead>
<tr>
<th>Gender</th>
<th>Date Hired</th>
<th>Employee Position</th>
</tr>
</thead>
<tbody>
<tr>
<td>M</td>
<td>09/12/1988</td>
<td>Master Police Officer</td>
</tr>
<tr>
<td>F</td>
<td>06/26/2006</td>
<td>Social Worker III</td>
</tr>
<tr>
<td>M</td>
<td>07/16/2007</td>
<td>Police Officer III</td>
</tr>
<tr>
<td>F</td>
<td>01/26/2000</td>
<td>Library Assistant I</td>
</tr>
</tbody>
</table>

Convert all values to numerical

- **Gender** = categorical column: One-hot encode
  
  ```python
  one_hot_enc = sklearn.preprocessing.OneHotEncoder()
  one_hot_enc.fit_transform(df[['Gender']])
  ```

<table>
<thead>
<tr>
<th>Gender (M)</th>
<th>Gender (F)</th>
<th>...</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td></td>
</tr>
</tbody>
</table>
Transformers: fit & transform

One-hot encoder

```python
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`
Data transformations: Transformers

\[ \text{transformer.transform(data)} \]

- Feature scaling
- Transforming categorical variables...

Train time

\[
\begin{align*}
\text{ohe} & = \text{OneHotEncoder}() \\
\text{ohe.fit(X_train, y_train)} \\
\text{X_train_encoded} & = \text{ohe.transform(X_train, y_train)} \\
\text{estimator.fit(X_train_encoded)}
\end{align*}
\]
Data transformations: Transformers

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

- **Model state**
  - data
  - transformed data

- learning the transformation (.fit) ≠ applying it (.transform)
  - Feature scaling
  - Transforming categorical variables...

**Test time**

```python
X_test_encoded = ohe.transform(X_test)
y_pred = estimator.predict(X_test_encoded)
```
Data transformations: Transformers

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

- Learning the transformation (`.fit`) ≠ 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)
```
Chaining operations: The pipeline

\[ \text{Pipeline} = \text{transformation1} \rightarrow (\text{transformation2} \ldots \rightarrow) \text{predictor} \]

\[ \text{pipe} = \text{make_pipeline(ohe, estimator)} \]

Replace:

\[
\begin{align*}
\text{ohe} &= \text{OneHotEncoder}() \\
\text{ohe.fit}(\text{X_train}, \text{y_train}) \\
\text{X_train_encoded} &= \text{ohe.transform}(\text{X_train}, \text{y_train}) \\
\text{estimator.fit}(\text{X_train_encoded})
\end{align*}
\]

with:

\[
\begin{align*}
\text{pipe.fit}(\text{X_train}, \text{y_train}) \\
\text{pipe.predict}(\text{X_test})
\end{align*}
\]
### Data tables: dates

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

<table>
<thead>
<tr>
<th>Gender</th>
<th>Date Hired</th>
<th>Employee Position</th>
</tr>
</thead>
<tbody>
<tr>
<td>M</td>
<td>09/12/1988</td>
<td>Master Police Officer</td>
</tr>
<tr>
<td>F</td>
<td>06/26/2006</td>
<td>Social Worker III</td>
</tr>
<tr>
<td>M</td>
<td>07/16/2007</td>
<td>Police Officer III</td>
</tr>
<tr>
<td>F</td>
<td>01/26/2000</td>
<td>Library Assistant I</td>
</tr>
</tbody>
</table>

Convert all values to numerical

- Date: use pandas’ datetime support

```python
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

```python
from dirty_cat import DatetimeEncoder

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

month, day, hour, dayofweek
```
Transformers: dates

Simplified object for dates – The dirty_cat module

DatetimeEncoder: features for different time regularity

```python
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`
Transformers: General case

For dates: FunctionTransformer

```python
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
ColumnTransformer: assembling

Applies different transformers to columns

These can be complex pipelines

```python
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
ColumnTransformer: assembling

Applies different transformers to columns

These can be complex pipelines

```python
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

```python
model = make_pipeline(column_trans, HistGradientBoostingClassifier)
scores = cross_val_score(model, df, y)
```
ColumnTransformer: assembling

Applies different transformers to columns

These can be complex pipelines

```python
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

```python
from dirty_cat import TableVectorizer

tab_vec = TableVectorizer()

X = tab_vec.fit_transform(df)
```

"Automagic" choices: defaults can be improved
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
is_soup = 0.5 \cdot \text{carrot} - 1.2 \cdot \text{flour} - 0.4 \cdot \text{sugar} + 0.6 \cdot \text{leak} \ldots

- Can handle large number of features
- "interpretable"

Interpretability pitfalls:
- Feature scaling matter:
  features with larger scale $\rightarrow$ smaller coefficient

- Coefficients are \textit{conditional} relations
  they must be understand "all other features kept constant"
  eg wage decreases with age, keeping experience constant

Linear models

is_soup = 0.5 \cdot \text{carrot} - 1.2 \cdot \text{flour} - 0.4 \cdot \text{sugar} + 0.6 \cdot \text{leak} \ldots

- Can handle large number of features
- “interpretable”

**Regression:**

```python
from sklearn.linear_model import Ridge
from sklearn.linear_model import RidgeCV
```

**Classification:** logistic regression

```python
from sklearn.linear_model import LogisticRegression
from sklearn.linear_model import LogisticRegressionCV
```

'\ell_2' and '\ell_1' penalties different solvers
Tree models (e.g. for heterogeneous columnar data)

- Decision trees: robust to strange data distributions

Ensemble methods: combine many trees
Random forests
sklearn.ensemble.RandomForestClassifier
Tree models (e.g., for heterogeneous columnar data)

- Decision trees: robust to strange data distributions

- Ensemble methods: combine many trees

  **Random forests**

  `sklearn.ensemble.RandomForestClassifier`
Tree ensembles

- Ensemble: combining many trees

Random forests

- `sklearn.ensemble.RandomForestClassifier`
- Build many trees on random perturbation of the data
- Average decisions
- More trees – higher `n_estimators` is better but more expensive

Boosted trees

- `sklearn.ensemble.HistGradientBoostingClassifier`
Gradient-boosted regression trees

- Fit with a tree of depth 10

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

sklearn.ensemble.HistGradientBoostingClassifier deals naively with missing values.
Gradient-boosted regression trees

1 staircase
2 staircases combined

Fit with a tree of depth 10
Fit a new tree on errors

staircase of 10 constant values

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

sklearn.ensemble.HistGradientBoostingClassifier deals naively with missing values.
Gradient-boosted regression trees

- 1 staircase
- 2 staircases combined
- 3 staircases combined

- 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.
Gradient-boosted regression trees

- 1 staircase
- 2 staircases combined
- 3 staircases combined
- 300 staircases combined

- Fit with a tree of depth 10
- Fit a new tree on errors

staircase of 10 constant values
Gradient-boosted regression trees

- 1 staircase
- 2 staircases combined
- 3 staircases combined
- 300 staircases combined

2 staircases combined
3 staircases combined
300 staircases combined

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
Gradient-boosted regression trees

- 1 staircase
- 2 staircases combined
- 3 staircases combined
- 300 staircases combined

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

Varoquaux
Kern

32
A function to decide if a cat is present?

Very non-local and complex function.

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
Deep learning and representations

**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
4 For text data

**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
Text mining
Text as data
Scraping the EuroPython abstracts

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
Scraping the EuroPython abstracts

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
Let's find some common topics

import urllib2, bs4

import sklearn, wordcloud

Anyone who has used Python to search text for substring patterns has at least heard of the regular expression module. Many of us use it extensively for parsers and lexers, training gives a quick introduction with exercises to extend django CMS. Let's talk about your plugins, apphooks, toolbar extensions.

import urllib2, bs4
import sklearn, wordcloud

Anyone who has used Python to search text for substring patterns has at least heard of the regular expression module. Many of us use it extensively for parsers and lexers, training gives a quick introduction with exercises to extend django CMS. Let's talk about your plugins, apphooks, toolbar extensions.
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
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" → "man"

- Stemming
  "consult"
  "consultant" → "consult"
  "consulting"

Software: nltk, spacy
Vectorizing webpages to numbers

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

**Vectorize**

bs4: beautiful soup, matchings on the DOM tree

Anyone who has used Python to search text for substring patterns has at least heard of the regular expression module. Many of us use it extensively to parse and lexers, training gives a quick introduction to this topic.

Chat with the core developers about how to extend django CMS or how to integrate your own apps seamlessly. Let's talk about your plugins, apphooks, toolbar extensions, profiling performance, and more.

<table>
<thead>
<tr>
<th>Term</th>
<th>Freq</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>20</td>
</tr>
<tr>
<td>can</td>
<td>10</td>
</tr>
<tr>
<td>code</td>
<td>4</td>
</tr>
<tr>
<td>is</td>
<td>14</td>
</tr>
<tr>
<td>module</td>
<td>3</td>
</tr>
<tr>
<td>profiling</td>
<td>2</td>
</tr>
<tr>
<td>performance</td>
<td>1</td>
</tr>
<tr>
<td>Python</td>
<td>9</td>
</tr>
<tr>
<td>the</td>
<td>18</td>
</tr>
</tbody>
</table>
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

Anyone who has used Python to search text for substring patterns has at least heard of the regular expression module. Many of us use it extensively to parse and lexers, training gives a quick introduction tool. This into some distinguishing features. Chat with django core developers to extend your own apps seamlessly. Your plugins, apphooks, toolbar extensions

<table>
<thead>
<tr>
<th>Term</th>
<th>Freq</th>
<th>All docs</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>20</td>
<td>1321</td>
</tr>
<tr>
<td>can</td>
<td>10</td>
<td>540</td>
</tr>
<tr>
<td>code</td>
<td>4</td>
<td>208</td>
</tr>
<tr>
<td>is</td>
<td>14</td>
<td>964</td>
</tr>
<tr>
<td>module</td>
<td>3</td>
<td>123</td>
</tr>
<tr>
<td>profiling</td>
<td>2</td>
<td>7</td>
</tr>
<tr>
<td>performance</td>
<td>1</td>
<td>6</td>
</tr>
<tr>
<td>Python</td>
<td>9</td>
<td>191</td>
</tr>
<tr>
<td>the</td>
<td>18</td>
<td>1450</td>
</tr>
</tbody>
</table>

Varoquaux
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

Anyone who has used Python to search text for substring patterns has at least heard of the regular expression module. Many of us use it extensively for parsers and lexers, training gives a quick introduction. This... Chat with the core developers about how to extend django CMS or how to integrate your own apps seamlessly. Let's talk about your plugins, apphooks, toolbar extensions.

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

TF-IDF in scikit-learn

sklearn.feature_extraction.text.TfidfVectorizer

Varoquaux
Vectorizing

From raw data to a sample matrix \( X \)

- For text data: counting word occurrences
  - Input data: list of documents (string)
  - Output data: numerical matrix
From raw data to a sample matrix $X$

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

```python
from sklearn.feature_extraction.text import TfidfVectorizer
vectorizer = TfidfVectorizer()
X = vectorizer.fit_transform(documents)
```
The term-document matrix

documents

the Python performance profiling module is code can a

can code is

Can be a sparse matrix

High-dimensional learning problem

⇒ Linear models (e.g., LogisticRegression)

G Varoquaux
## The term-document matrix

The term-document matrix can be a sparse matrix. Can be an area for high-dimensional learning problems such as linear models (e.g., LogisticRegression).

### Example Term-document Matrix

<table>
<thead>
<tr>
<th>Documents</th>
<th>Code</th>
<th>Module</th>
<th>Performance</th>
</tr>
</thead>
<tbody>
<tr>
<td>Documents</td>
<td>03078090707907</td>
<td>00790752700578</td>
<td>94071006000797</td>
</tr>
<tr>
<td>Code</td>
<td>00970008007000</td>
<td>10000400400090</td>
<td>00050205008000</td>
</tr>
</tbody>
</table>

The Python performance profiling module can be code.
The term-document matrix

- High-dimensional learning problem
  \[\Rightarrow\text{Linear models} \quad (eg \text{LogisticRegression})\]

- Can be a sparse matrix

- Python performance profiling module is code can
Semantics
Semantics
Relations between words
5 Topic modeling: matrix factorization

A matrix factorization

Often with non-negative constraints

What terms are in a topics
What documents are in a topics

The Python performance profiling module is code can a

sklearn.decompositions.NMF
On the EuroPython abstracts

Topic 1
On the EuroPython abstracts

Topic 2
On the EuroPython abstracts

Topic 3

[Word cloud image]
On the EuroPython abstracts
Semantics and word embeddings

Distributional semantics:
“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

Center word
U: word embedding
salad
meat
juice
wine
glass
green
red
Context word
V: context embedding
salad
meat
juice
wine
glass
green
red

Varoquaux
Precomputed word embeddings

Trained on huge corpora

Word2vec

FastText: robust to typos and new words

Transform words into vectors

⇒ Low-dimensional, dense inputs

⇒ richer machine-learning models

Software: gensim, fasttext
Precomputed word embeddings

Trained on huge corpora

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
Traditional sequence models

Right language models: predict the next word

Recurrent Neural Network:
\[
\text{Probability}\left((n + 1)^{th} \text{ word} \mid n^{th}, (n - 1)^{th}, \ldots\right) = f\left(n^{th} \text{ word}, \text{Probability}\left(n^{th} \text{ word} \mid (n - 1)^{th}, (n - 2)^{th}, \ldots\right)\right)
\]

Challenge: long-distance links \(\Rightarrow\) LSTM

Also for left language models

Importance of language models predicting words

Difficulty of capturing long-distance relationships
Transformers

Masked language models

Extracts internal representations of word sequences

Software: Huggingface transformers

for longer texts, grammatical structure, distant syntax

Should penalized least squares regression be interpreted as maximum a posteriori estimation?