Data Preprocessing

Tabular data

Time series data

Text data

Image data

Tabular data

- Most common in data analysis
- Used in business and medical fields
- Easiest to analyze/preprocess

Temperature	Headache	Nausea	Decision (flu)
High	*	No	Yes
Very high	Yes	Yes	Yes
*	No	No	No
High	Yes	Yes	Yes
High	*	Yes	No
Normal	Yes	No	No
Normal	No	Yes	No
*	Yes	*	Yes

Imputation

Impute missing data with...

- numerical: mean, median
- categorical: most common value
- Regression of other variables
- ...or other methods (MICE etc.)

Temperature	Headache	Nausea	Decision (flu)
High	*	No	Yes
Very high	Yes	Yes	Yes
*	No	No	No
High	Yes	Yes	Yes
High	*	Yes	No
Normal	Yes	No	No
Normal	No	Yes	No
*	Yes	*	Yes

Imputation

Example of imputation in `Scikit-learn`

```
from sklearn.impute import SimpleImputer

imputer = SimpleImputer(strategy="median")
imputer.fit(data)
```

Encoder for categorical features

1. Ordinal encoder

```
from sklearn.preprocessing import OrdinalEncoder

ordinal_encoder = OrdinalEncoder()
data_enc = ordinal_encoder.fit_transform(data)
```

Encoder for categorical features

2. Nominal encoder

```
from sklearn.preprocessing import OneHotEncoder

onehot_encoder = OneHotEncoder()
data_1hot = onehot_encoder.fit_transform(data)
```

Tabular data

Time series data

Text data

Image data

Time series data

- Seen in business, economics, environment, energy
- Rows of data are not i.i.d.!
- Must be processed before trained by ML models

	AAPL.High	AAPL.Low	AAPL.Close
2008-09-16	20.4	18.9	20.0
2008-09-17	19.8	18.3	18.3
2008-09-18	19.3	17.2	19.2
2008-09-19	20.6	19.5	20.1
2008-09-22	20.0	18.7	18.7
2008-09-23	19.4	18.1	18.1
2008-09-24	18.7	17.9	18.4
2008-09-25	19.3	18.4	18.8
2008-09-26	18.5	17.6	18.3
2008-09-29	17.1	14.4	15.0
2008-09-30	16.4	15.2	16.2
2008-10-01	16.1	15.3	15.6
2008-10-02	15.5	14.3	14.3

Sliding Window

	AAPL.High
2008-09-16	20.4
2008-09-17	19.8
2008-09-18	19.3
2008-09-19	20.6
2008-09-22	20.0
2008-09-23	19.4
2008-09-24	18.7
2008-09-25	19.3
2008-09-26	18.5
2008-09-29	17.1
2008-09-30	16.4
2008-10-01	16.1
2008-10-02	15.5

Sliding Window

Example of python code:

```python
def rolling(x, order):
    n_points = x.shape[0]
    running = []

    for i in range(n_points-order+1):
        running.append(x[i:i+order])

    return np.array(running)
```

Multivariate Sliding Window

	AAPL.High	AAPL.Low	AAPL.Close
2008-09-16	20.4	18.9	20.0
2008-09-17	19.8	18.3	18.3
2008-09-18	19.3	17.2	19.2
2008-09-19	20.6	19.5	20.1
2008-09-22	20.0	18.7	18.7
2008-09-23	19.4	18.1	18.1
2008-09-24	18.7	17.9	18.4
2008-09-25	19.3	18.4	18.8
2008-09-26	18.5	17.6	18.3
2008-09-29	17.1	14.4	15.0
2008-09-30	16.4	15.2	16.2
2008-10-01	16.1	15.3	15.6
2008-10-02	15.5	14.3	14.3

Tabular data

Time series data

Text data

Image data

Bag-of-Words

	about	bird	heard	is	the	word	you
About the bird, the bird, bird bird bird	1	5	0	0	2	0	0
You heard about the bird	1	1	1	0	1	0	1
The bird is the word	0	1	0	1	2	1	0

Bag-of-Words

Example of Bag-Of-Words in `Scikit-learn`

```python
from sklearn.feature_extraction.text import CountVectorizer

vectorizer = CountVectorizer()
corpus = [
    '2 cups of flour',
    'replace the flour',
    'replace the keyboard in 2 minutes',
    'do you prefer Windows or Mac',
    'the Mac has the most noisy keyboard',
]
X = vectorizer.fit_transform(corpus)
```

Tf-Idf

- **w** = a word
- **D** = a document

$$x_{\mathbf{w},\mathbf{D}} = \mathrm{tf}_{\mathbf{w},\mathbf{D}} \times \log\left(\frac{N}{\mathrm{df}_{\mathbf{w}}}\right)$$

Tf-Idf

Word	Count		tf		idf	tf×idf	
	Doc 1	Doc 2	Doc 1	Doc 2		Doc 1	Doc 2
I	1	1					
see	1	0					
cat	1	1					
and	2	1					
dog	1	1					
walk	1	1					
bird	0	1					

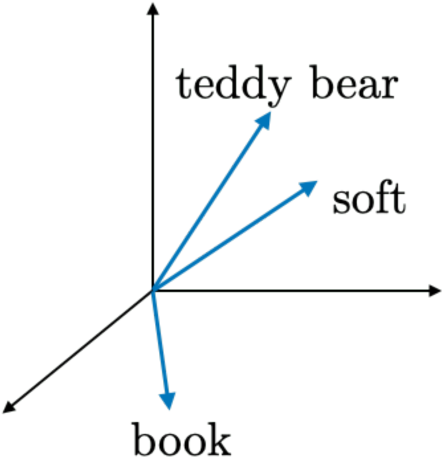
TF-Idf

Example of Tf-idf in `Scikit-learn`

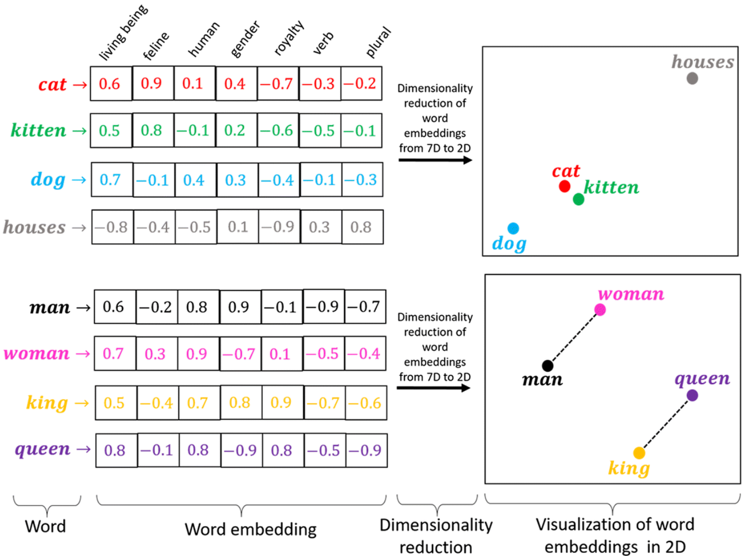
```
from sklearn.feature_extraction.text import TfidfVectorizer

vectorizer = TfidfVectorizer()
corpus = [
    '2 cups of flour',
    'replace the flour',
    'replace the keyboard in 2 minutes',
    'do you prefer Windows or Mac',
    'the Mac has the most noisy keyboard',
]
X = vectorizer.fit_transform(corpus)
```

Capturing word similarity

- Bag-of-Words and Tf-idf do not capture word similary
- Introduce a new encoding technique: **Word2Vec**



Word2Vec



living being
feline
human
gender
royalty
verb
plural
cat → 0.6 0.9 0.1 0.4 −0.7 −0.3 −0.2
kitten → 0.5 0.8 −0.1 0.2 −0.6 −0.5 −0.1
dog → 0.7 −0.1 0.4 0.3 −0.4 −0.1 −0.3
houses → −0.8 −0.4 −0.5 0.1 −0.9 0.3 0.8
man → 0.6 −0.2 0.8 0.9 −0.1 −0.9 −0.7
woman → 0.7 0.3 0.9 −0.7 0.1 −0.5 −0.4
king → 0.5 −0.4 0.7 0.8 0.9 −0.7 −0.6
queen → 0.8 −0.1 0.8 −0.9 0.8 −0.5 −0.9
Dimensionality reduction of word embeddings from 7D to 2D
Dimensionality reduction of word embeddings from 7D to 2D
houses
cat
kitten
dog
woman
man
queen
king
Word
Word embedding
Dimensionality reduction
Visualization of word embeddings in 2D

Encode sentences with Word2Vec

cat →	0.6	0.9	0.1	0.4	−0.7	−0.3	−0.2
kitten →	0.5	0.8	−0.1	0.2	−0.6	−0.5	−0.1
dog →	0.7	−0.1	0.4	0.3	−0.4	−0.1	−0.3
houses →	−0.8	−0.4	−0.5	0.1	−0.9	0.3	0.8

man →	0.6	−0.2	0.8	0.9	−0.1	−0.9	−0.7
woman →	0.7	0.3	0.9	−0.7	0.1	−0.5	−0.4
king →	0.5	−0.4	0.7	0.8	0.9	−0.7	−0.6
queen →	0.8	−0.1	0.8	−0.9	0.8	−0.5	−0.9

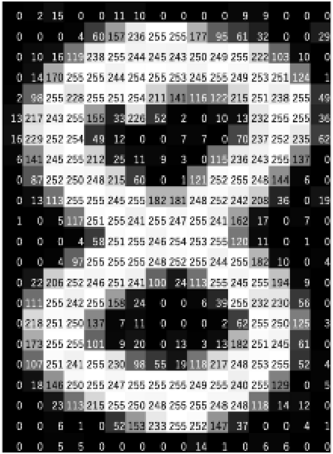
Tabular data

Time series data

Text data

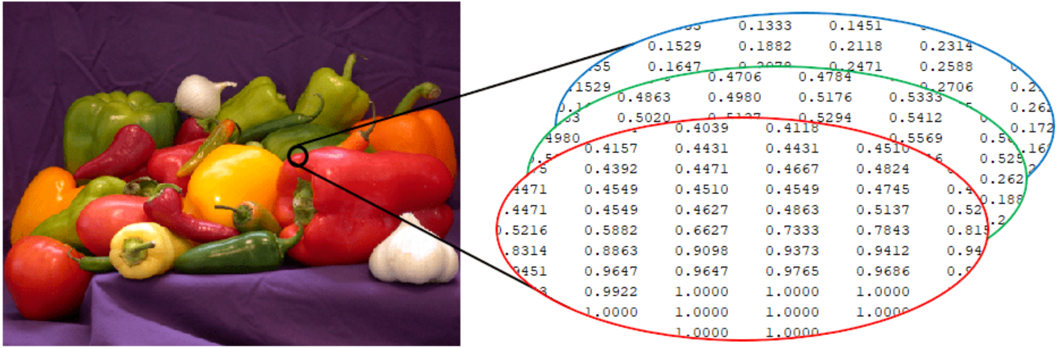
Image data

Pixels



0	2	15	0	0	11	10	0	0	0	0	9	9	0	0	0
0	0	0	4	60	157	236	255	255	177	95	61	32	0	0	29
0	10	16	119	238	255	244	245	243	250	249	255	222	103	10	0
0	14	170	255	255	244	254	255	253	245	255	249	253	251	124	1
2	98	255	228	255	251	254	211	141	116	122	215	251	238	255	49
13	217	243	255	155	33	226	52	2	0	10	13	232	255	255	36
16	229	252	254	49	12	0	0	7	7	0	70	237	252	235	62
6	141	245	255	212	25	11	9	3	0	115	236	243	255	137	0
0	87	252	250	248	215	60	0	1	121	252	255	248	144	6	0
0	13	113	255	255	245	255	182	181	248	252	242	208	36	0	19
1	0	5	117	251	255	241	255	247	255	241	162	17	0	7	0
0	0	0	4	58	251	255	246	254	253	255	120	11	0	1	0
0	0	4	97	255	255	255	248	252	255	244	255	182	10	0	4
0	22	206	252	246	251	241	100	24	113	255	245	255	194	9	0
0	111	255	242	255	158	24	0	0	6	39	255	232	230	56	0
0	218	251	250	137	7	11	0	0	0	2	62	255	250	125	3
0	173	255	255	101	9	20	0	13	3	13	182	251	245	61	0
0	107	251	241	255	230	98	55	19	118	217	248	253	255	52	4
0	18	146	250	255	247	255	255	255	249	255	240	255	129	0	5
0	0	23	113	215	255	250	248	255	255	248	248	118	14	12	0
0	0	6	1	0	52	153	233	255	252	147	37	0	0	4	1
0	0	5	5	0	0	0	0	0	14	1	0	6	6	0	0

RGB images

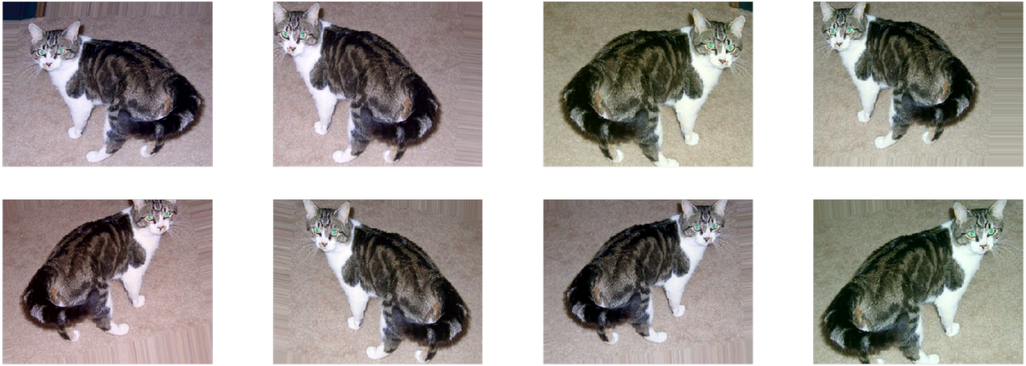


0.1333 0.1451
0.1529 0.1882 0.2118 0.2314
0.1647 0.2471 0.2588
0.4706 0.4784
0.2706
0.4863 0.4980 0.5176 0.5333
0.5020 0.5294 0.5412
0.4039 0.4118
0.5569
0.4157 0.4431 0.4431
0.4392 0.4471 0.4667 0.4824
0.4549 0.4510 0.4549 0.4745
0.4549 0.4627 0.4863 0.5137
0.5882 0.6627 0.7333 0.7843
0.8863 0.9098 0.9373 0.9412
0.9647 0.9647 0.9765 0.9686
0.9922 1.0000 1.0000 1.0000
1.0000 1.0000 1.0000
1.0000 1.0000

Importing images

```
import numpy as np
from PIL import Image

im = Image.open('my_image.png')
im_arr = np.array(im)
im_arr.shape
Out: (487, 650, 3)
```

Image Augmentation



More training data!

Feature Normalization

- We can transform each feature (each column in the dataset) to have mean $= 0$ and standard deviation $= 1$

$$x \to \frac{x - \mathrm{mean}(x)}{\mathrm{s.\,d.}(x)}$$

  - Prevent `OverflowError` by making x small
  - Some method, such as PCA, only works well when the data is centered at the origin

- For image data, we can simply do

$$x \to \frac{x}{255}$$

Feature Normalization

- For a BoWs or Tf-idf vector of a document, normalize by its length:

$$x \to \frac{x}{\sqrt{x_1^2 + x_2^2 + \ldots + x_d^2}} \quad x = (x_1, x_2, \ldots, x_d)$$

  If you use `TfidfVectorizer`, the normalization is done automatically

Feature Normalization

- You must split the data into train+val+test sets before any preprocessing!
- This is to prevent ”data leakage” from the training set to test set
- Transform test set using mean and s.d. **of the training set**

$$x \to \frac{x - \mathrm{mean}_{\mathrm{train}}(x)}{\mathrm{s.\,d.}_{\mathrm{train}}(x)}$$

Feature Normalization

Wrong!

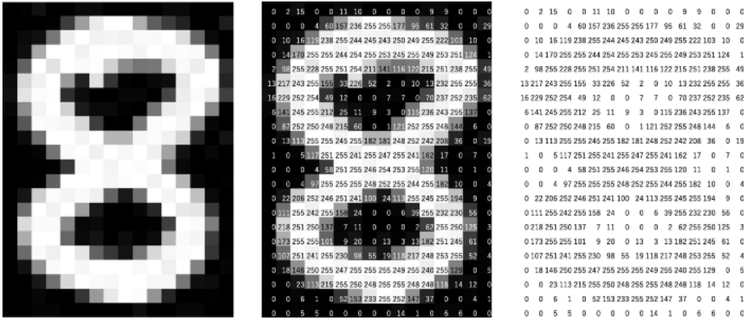
```
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split

scaler = StandardScaler()

data_norm = scaler.fit_transform(data)
train, test = train_test_split(data_norm)
```

Feature Normalization

Correct.

```python
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split

train, test = train_test_split(data)
scaler = StandardScaler()

train_norm = scaler.fit_transform(train)
test_norm = scaler.transform(test)
```

Dimensionality reduction



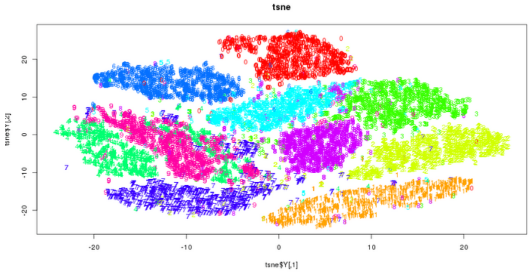
- How can we visualize high dimensional data?
- There are various methods to reduce the data to 2D (e.g. PCA)

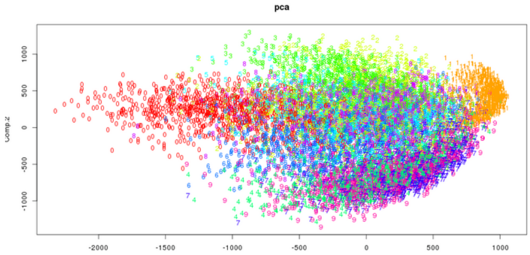
t-SNE

Example of t-SNE in `Scikit-learn`

```
from sklearn.manifold import TSNE

t_sne = manifold.TSNE(n_components=2)
embedding = t_sne.fit_transform(data)
```

t-SNE and PCA on MNIST data





UMAP

Example of UMAP

```python
from umap import UMAP

um = umap.UMAP(n_neighbors=5)
embedding = um.fit_transform(data)
```

UMAP on MNIST data