Special Topic: TRANSFORMS

PHYS 453

Dr Daugherity

ACU
ABILENE CHRISTIAN UNIVERSITY

Feature Scaling

Problem #1

Consider a dataset that has features with very different scales.

x1 from [0.01 to 0.04]

x2 from [20,000 to 30,000]

Some classifiers don't care (decision trees)

But some will be behave oddly…

- Nearest Neighbors will effectively ignore x1
- Neural Networks will have a hard time converging

The Solution

It is always a good idea to scale your input data.

References:

- User's Guide http://scikit-learn.org/stable/modules/preprocessing.html
- StandardScaler class: http://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.StandardScaler.html
- Intro to ML Chapter 3: https://github.com/amueller/introduction_to_ml_with_python/blob/master/03-unsupervised-learning.ipynb

http://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.StandardScaler.html#sklearn.preprocessing.StandardScaler

StandardScaler will set the mean to zero and variance to one

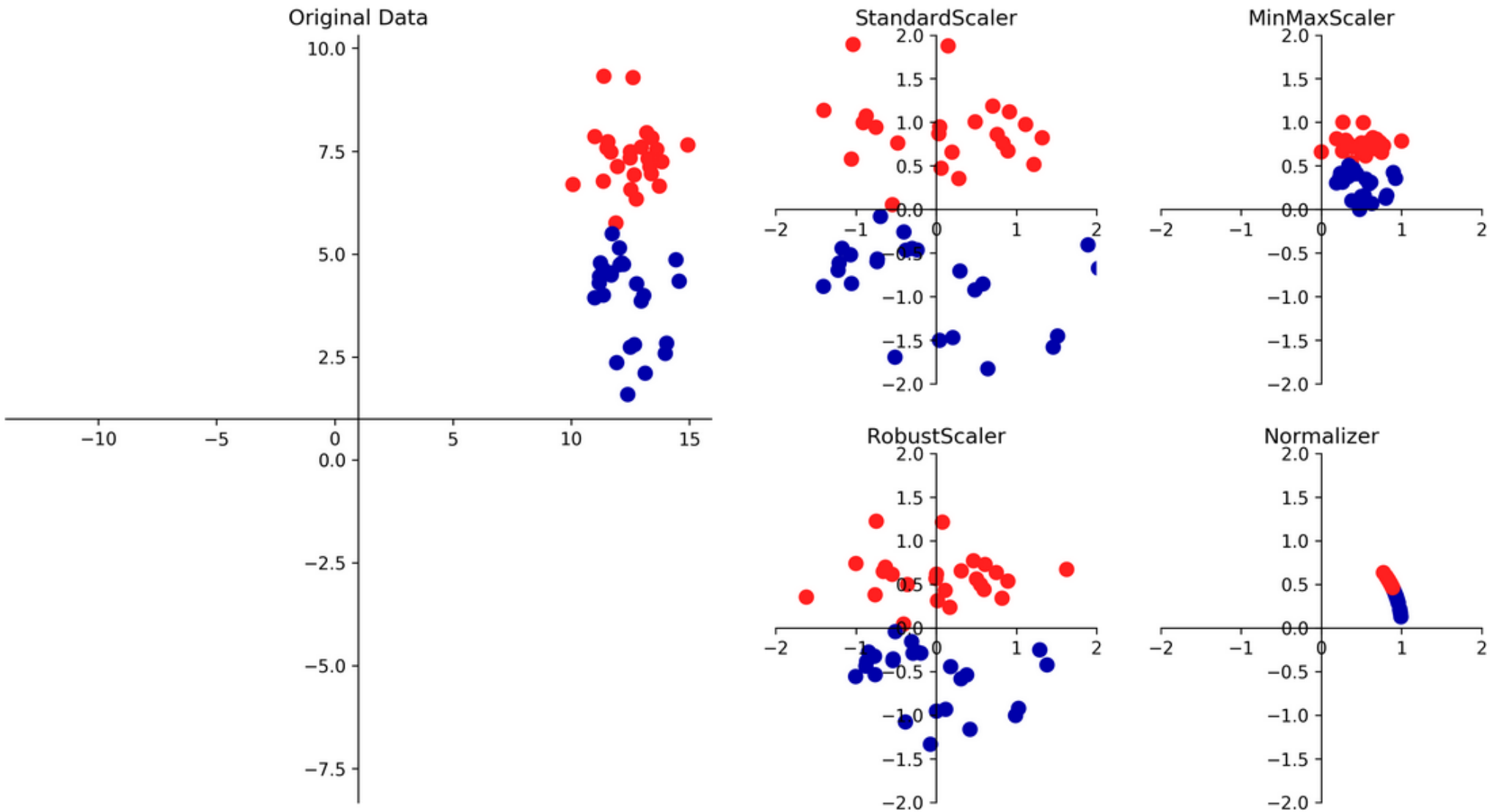
Methods

`fit` (X[, y])	Compute the mean and std to be used for later scaling.
`fit_transform` (X[, y])	Fit to data, then transform it.
`get_params` ([deep])	Get parameters for this estimator.
`inverse_transform` (X[, copy])	Scale back the data to the original representation
`partial_fit` (X[, y])	Online computation of mean and std on X for later scaling.
`set_params` (**params)	Set the parameters of this estimator.
`transform` (X[, y, copy])	Perform standardization by centering and scaling

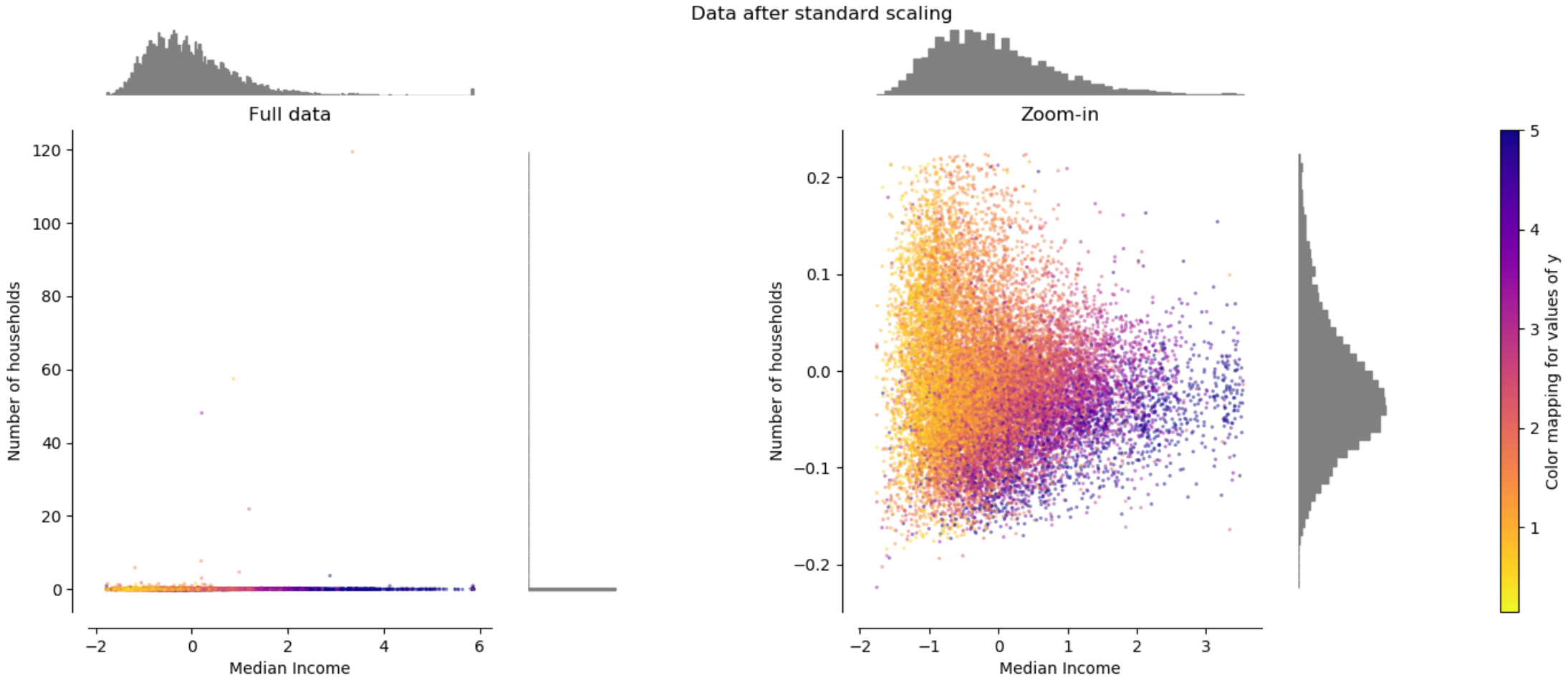
BE CAREFUL to transform all data the same way!

- don’t re-fit your testing data
- don’t forget to transform testing data

https://github.com/amueller/introduction_to_ml_with_python/blob/master/03-unsupervised-learning.ipynb



http://scikit-learn.org/stable/auto_examples/preprocessing/plot_all_scaling.html



PCA

Problem #2

Given a set of features, is it possible to optimize them **before** classifying?

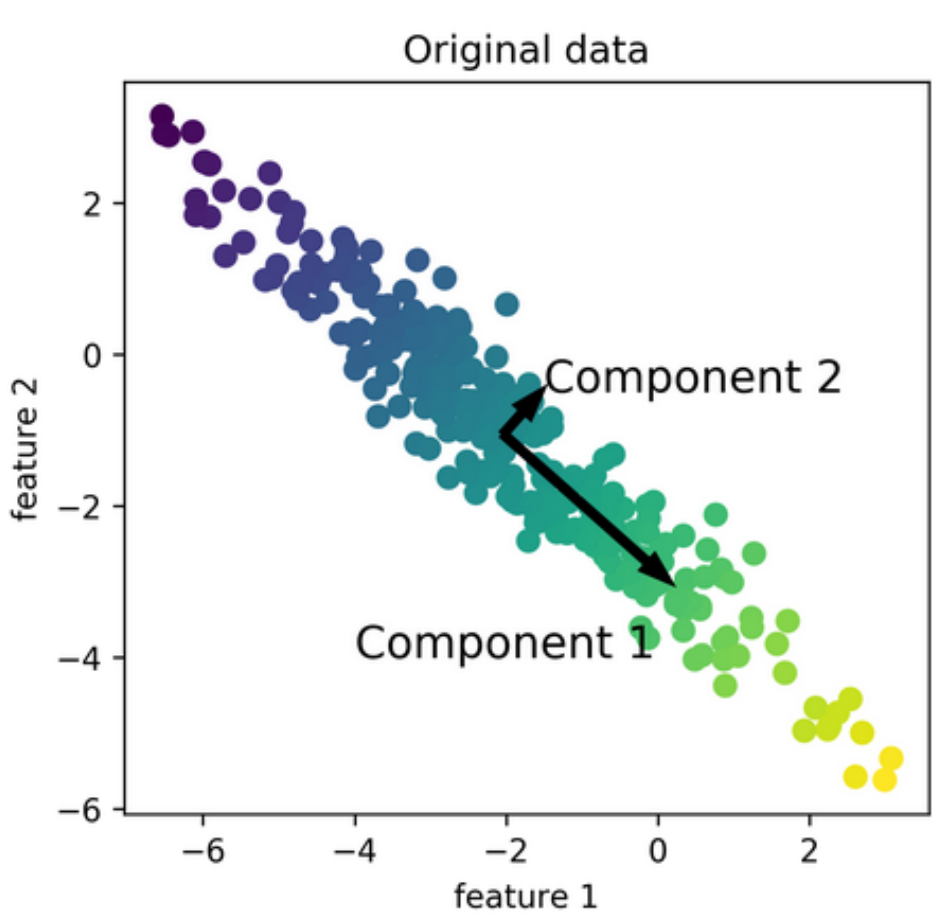
There are a set of preprocessing tricks we can use

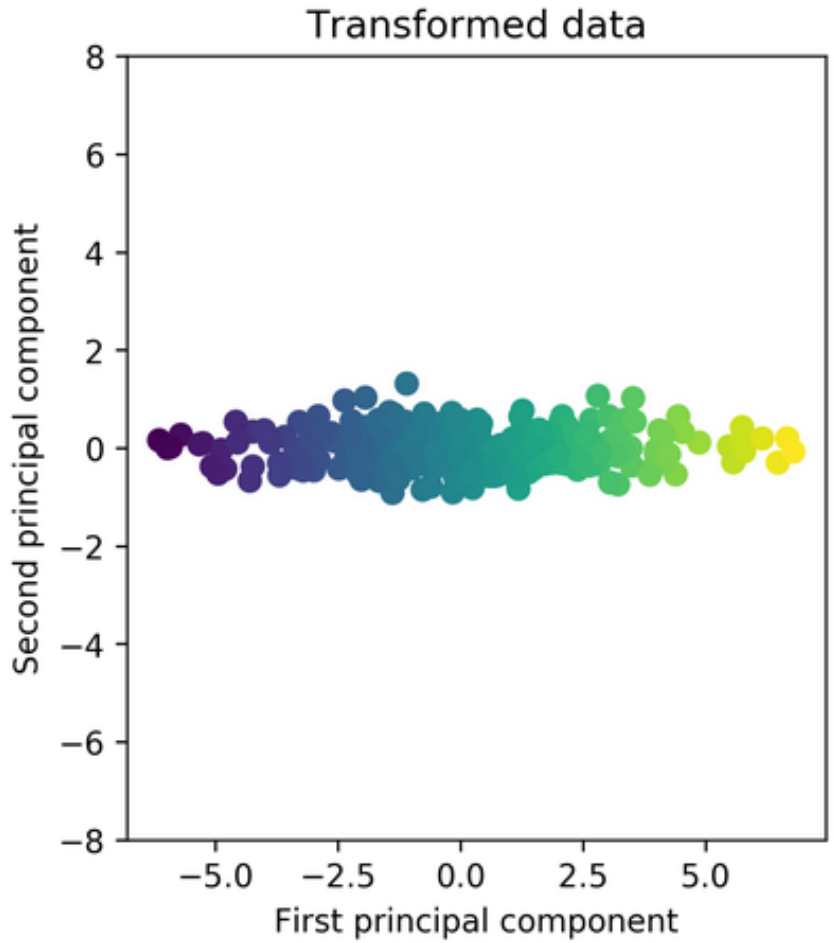
Principal Component Analysis

The algorithm is surprisingly easy:

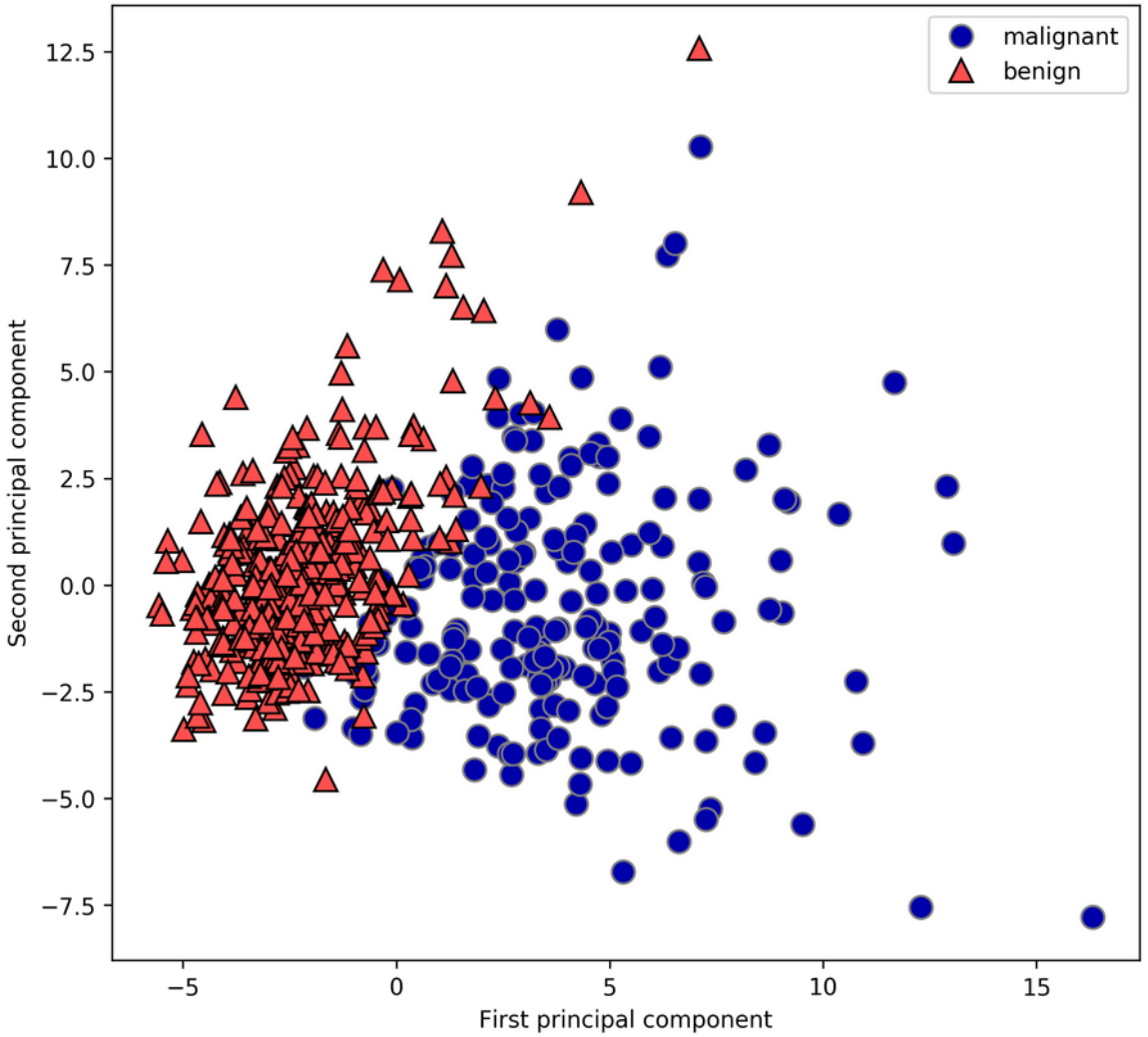
1. normalize the data
2. construct covariance matrix
3. find eigenvalues and eigenvectors

- "Principal Components" are the eigenvectors in order from largest to smallest eigenvalues
- You can discard the less important eigenvectors to reduce the number of dimensions
- *Maximizes* local variance

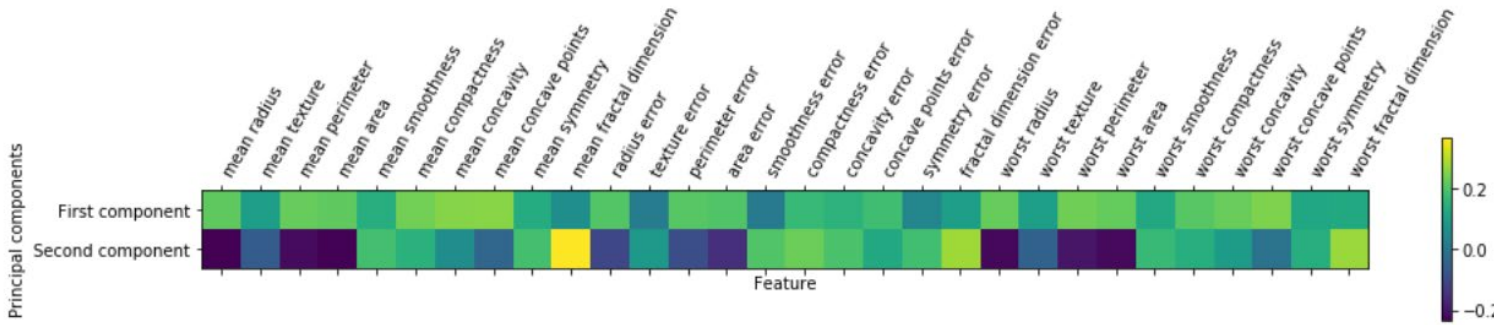

Original data
feature 2
feature 1
Component 2
Component 1
2
0
−2
−4
−6
−6
−4
−2
0
2


Transformed data
Second principal component
First principal component
8
6
4
2
0
−2
−4
−6
−8
−5.0
−2.5
0.0
2.5
5.0

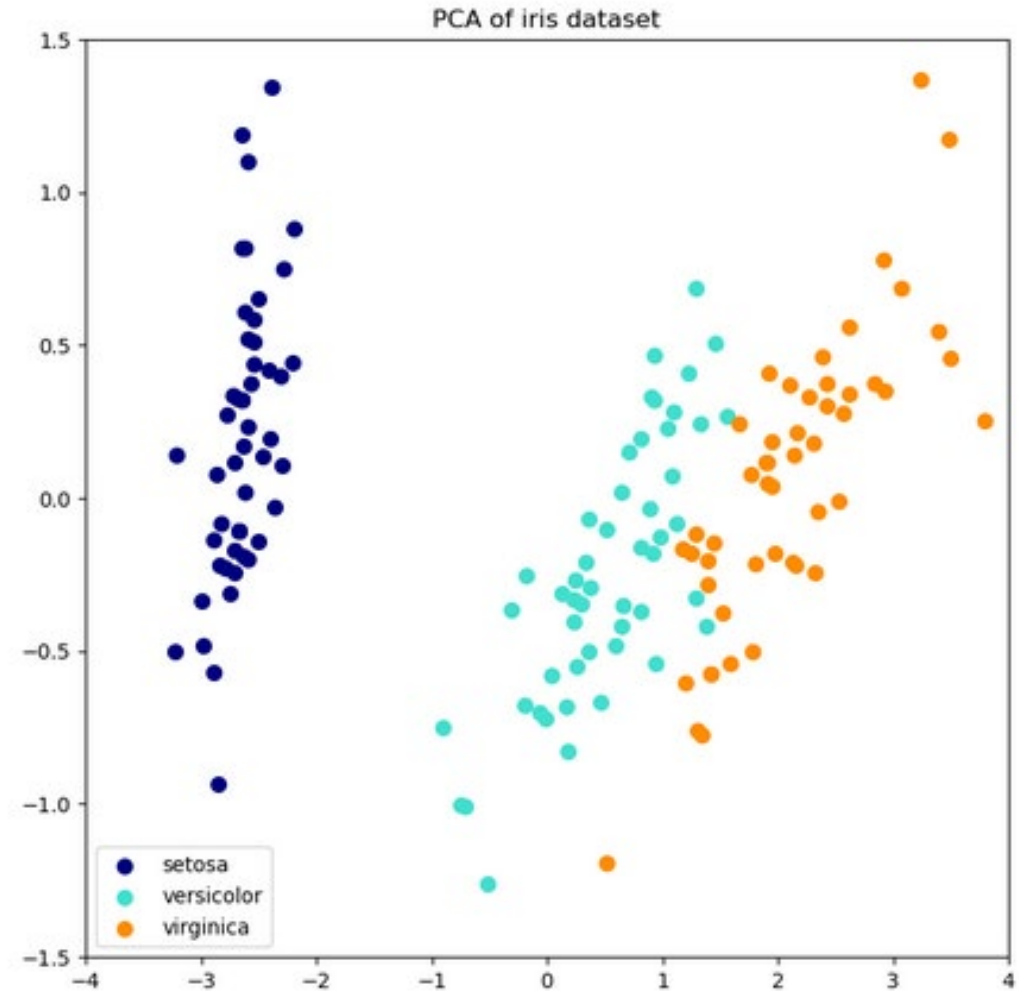
https://github.com/amueller/introduction_to_ml_with_python/blob/master/03-unsupervised-learning.ipynb



The cancer data set has 30 features. PCA lets us visualize a 2D representation of the data.



http://scikit-learn.org/stable/auto_examples/decomposition/plot_incremental_pca.html




```
pca.components_
```

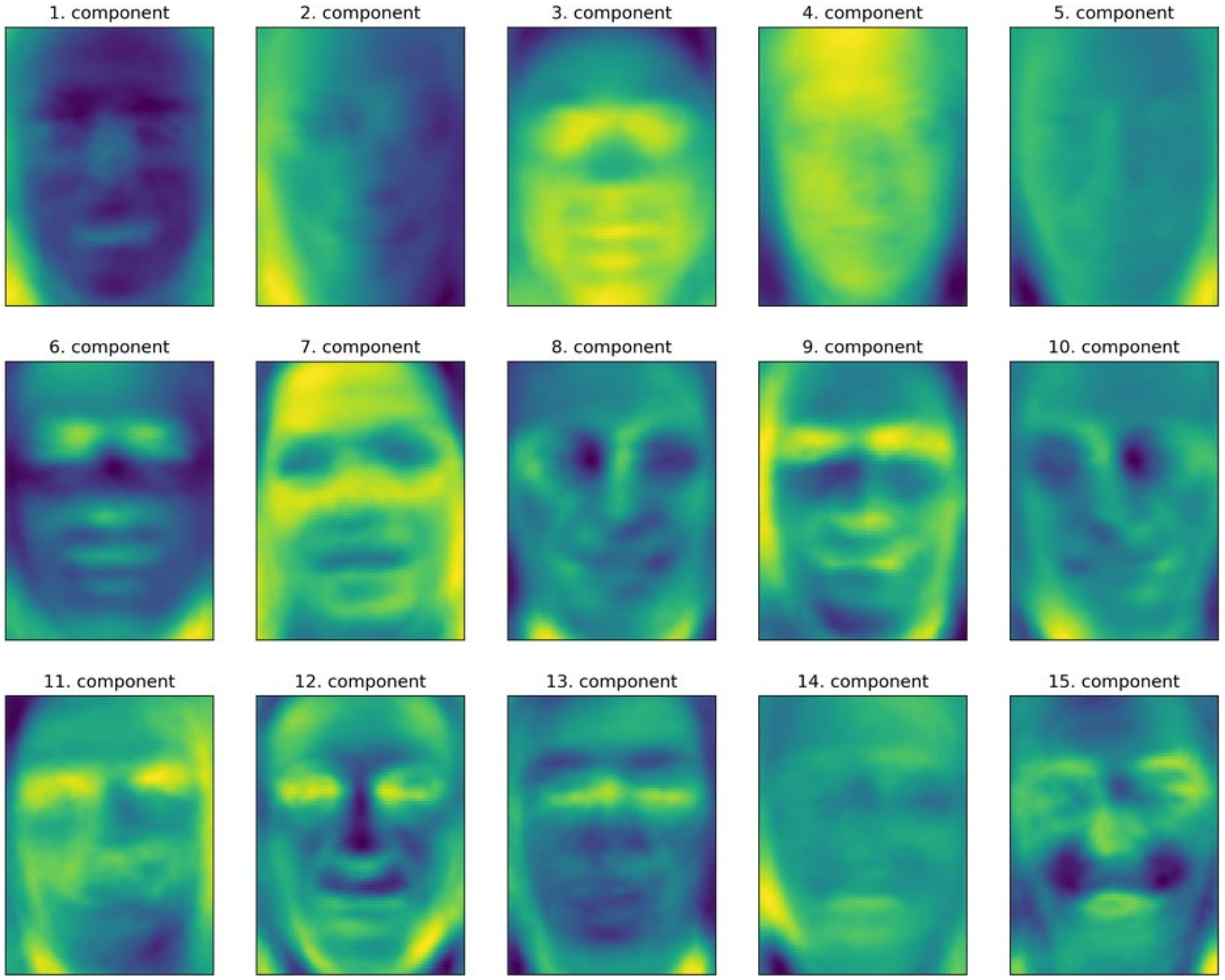
```
array([[ 0.36138659, -0.08452251,  0.85667061,  0.3582892 ],
       [ 0.65658877,  0.73016143, -0.17337266, -0.07548102]])
```

Eigenfaces!

If we plot the biggest PCA components for face images we get the eigenfaces!

Every face could be represented as a linear combination of these.

We can also use this trick for completing partial images. Fit the eigenfaces to the portion we've got then predict missing piece



Pipelines

https://scikit-learn.org/stable/modules/compose.html#pipeline

6.1.1. Pipeline: chaining estimators

`Pipeline` can be used to chain multiple estimators into one. This is useful as there is often a fixed sequence of steps in processing the data, for example feature selection, normalization and classification. `Pipeline` serves multiple purposes here:

Convenience and encapsulation

You only have to call fit and predict once on your data to fit a whole sequence of estimators.

Joint parameter selection

You can grid search over parameters of all estimators in the pipeline at once.

Safety

Pipelines help avoid leaking statistics from your test data into the trained model in cross-validation, by ensuring that the same samples are used to train the transformers and predictors.

All estimators in a pipeline, except the last one, must be transformers (i.e. must have a transform method). The last estimator may be any type (transformer, classifier, etc.).

Example

https://scikit-learn.org/stable/auto_examples/compose/plot_digits_pipe.html#sphx-glr-auto-examples-compose-plot-digits-pipe-py

```
# Define a pipeline to search for the best combination of PCA truncation
# and classifier regularization.
pca = PCA()
# Define a Standard Scaler to normalize inputs
scaler = StandardScaler()

# set the tolerance to a large value to make the example faster
logistic = LogisticRegression(max_iter=10000, tol=0.1)
pipe = Pipeline(steps=[("scaler", scaler), ("pca", pca), ("logistic", logistic)])

X_digits, y_digits = datasets.load_digits(return_X_y=True)
# Parameters of pipelines can be set using '__' separated parameter names:
param_grid = {
    "pca__n_components": [5, 15, 30, 45, 60],
    "logistic__C": np.logspace(-4, 4, 4),
}
search = GridSearchCV(pipe, param_grid, n_jobs=2)
search.fit(X_digits, y_digits)
print("Best parameter (CV score=%0.3f):" % search.best_score_)
print(search.best_params_)
```

Pipeline

- Best practice
- Avoids silly bugs and data leakage
- Slight change to grid search, worth it for speed and safety

Manifold Learning

Manifold Learning

http://scikit-learn.org/stable/modules/manifold.html

This is technically our first look at unsupervised learning.

Can we do even better? Transform the features in a way that makes the (unlabeled) data form clusters in simple 2D or 3D space.

These tricks are implemented in sklearn as “Manifold Learning”. Let’s look at one of the newer and most magical methods.

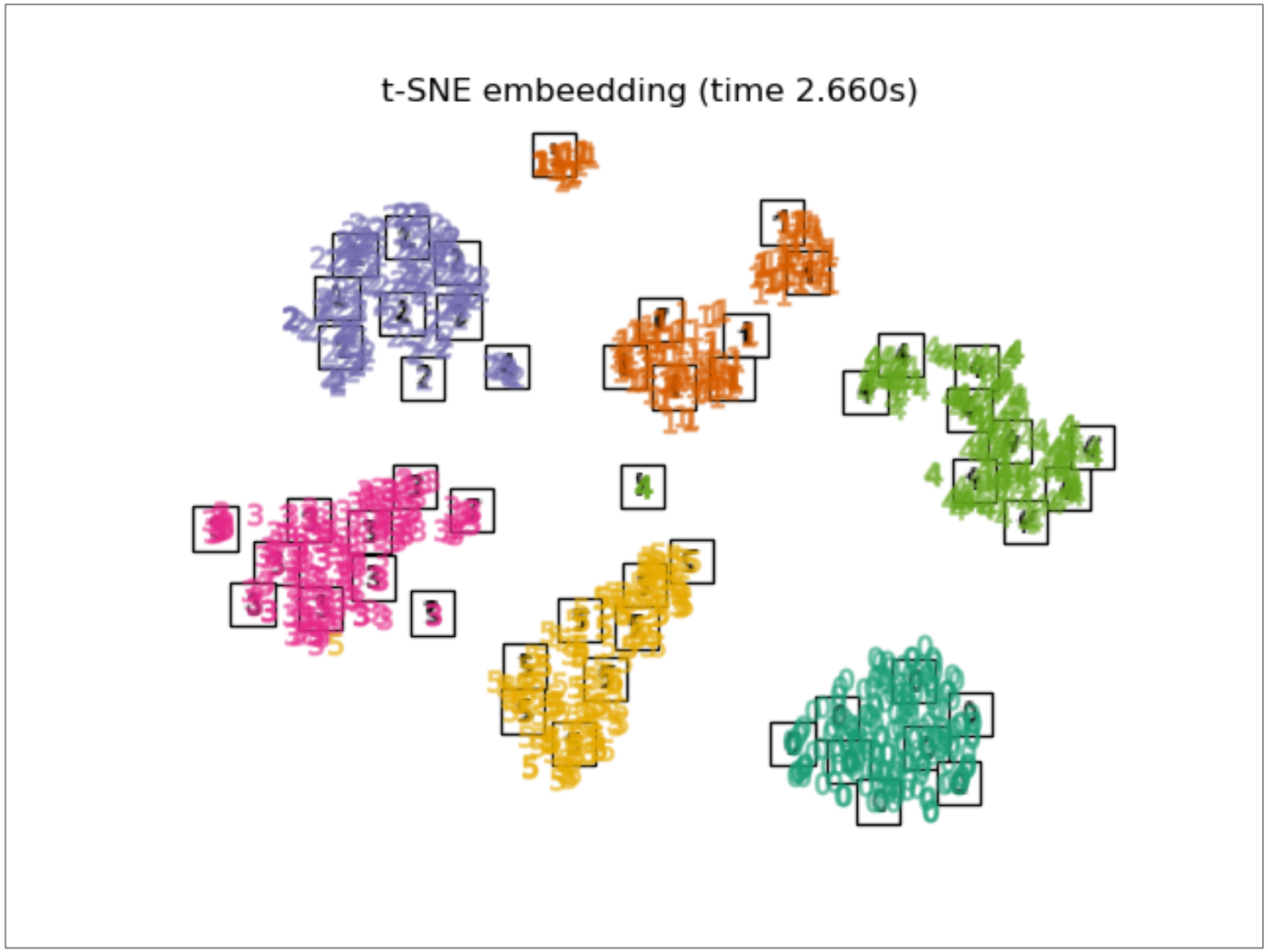
tSNE

t-distributed Stochastic Neighbor Embedding

Imagine we want to construct a 2D map of high-dimensional points. We can do this by attaching “springs” to each point in 2D space where the springs get smaller for points that are close neighbors

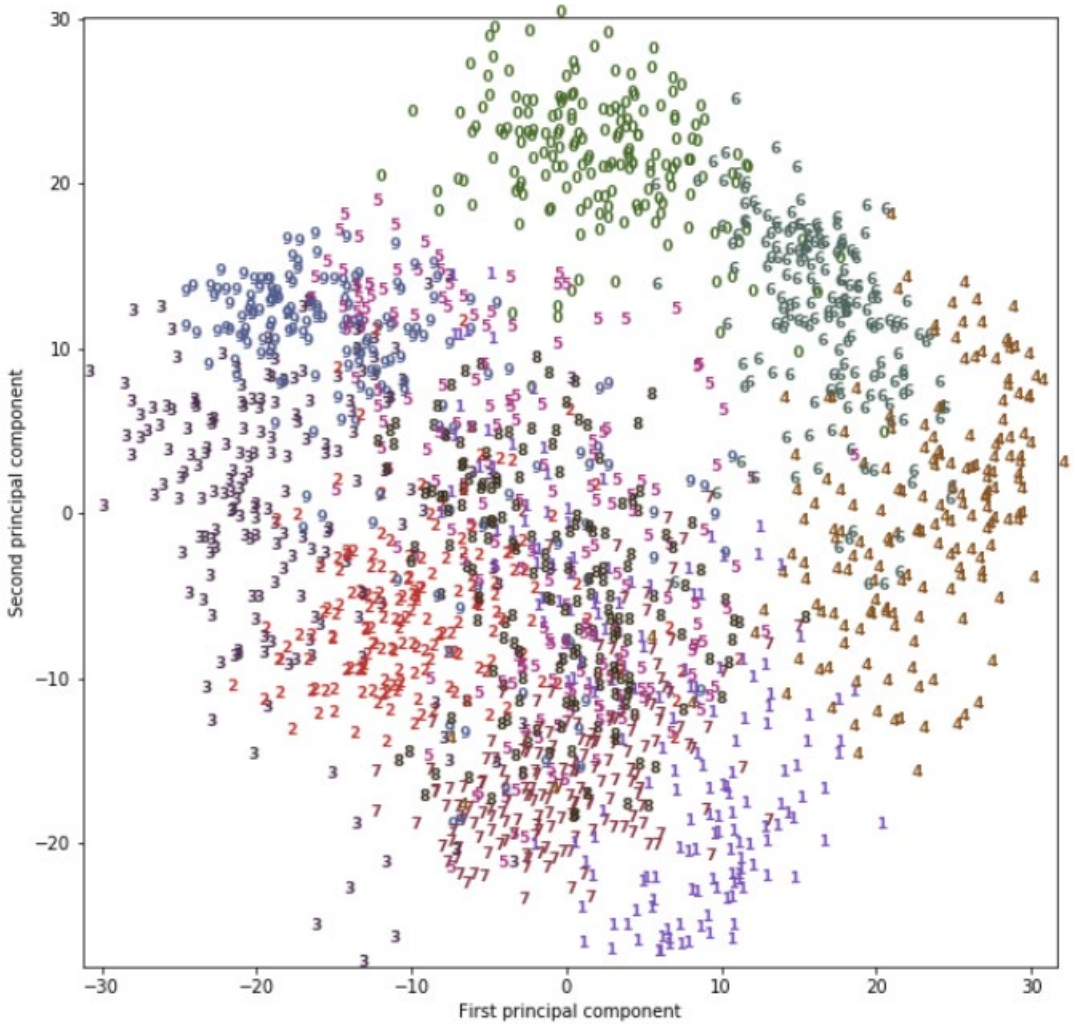
- https://distill.pub/2016/misread-tsne/
- https://www.youtube.com/watch?v=RJVL80Gg3lA&list=UUtXKDgv1AVoG88PLl8nGXmw (algorithm starts at 10:30)

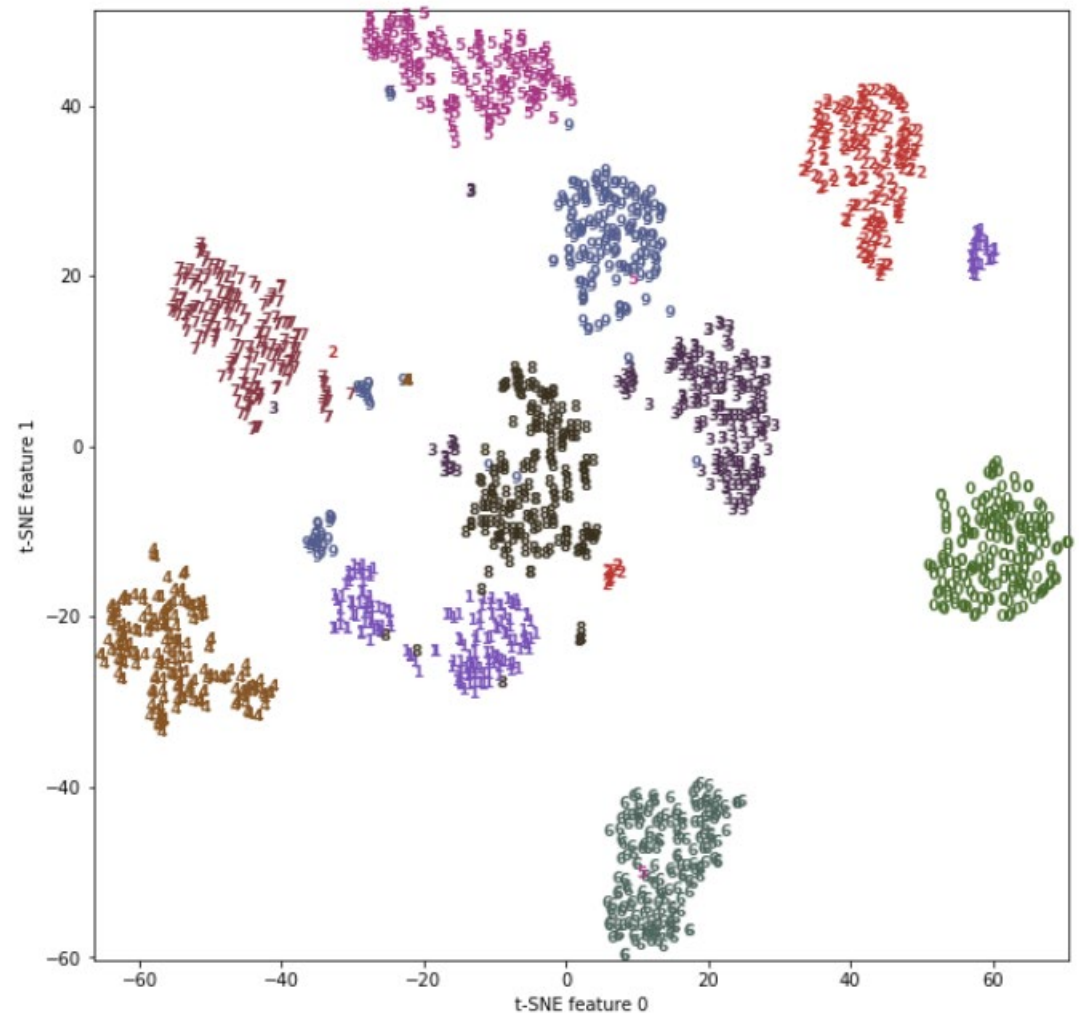
https://scikit-learn.org/stable/auto_examples/manifold/plot_lle_digits.html



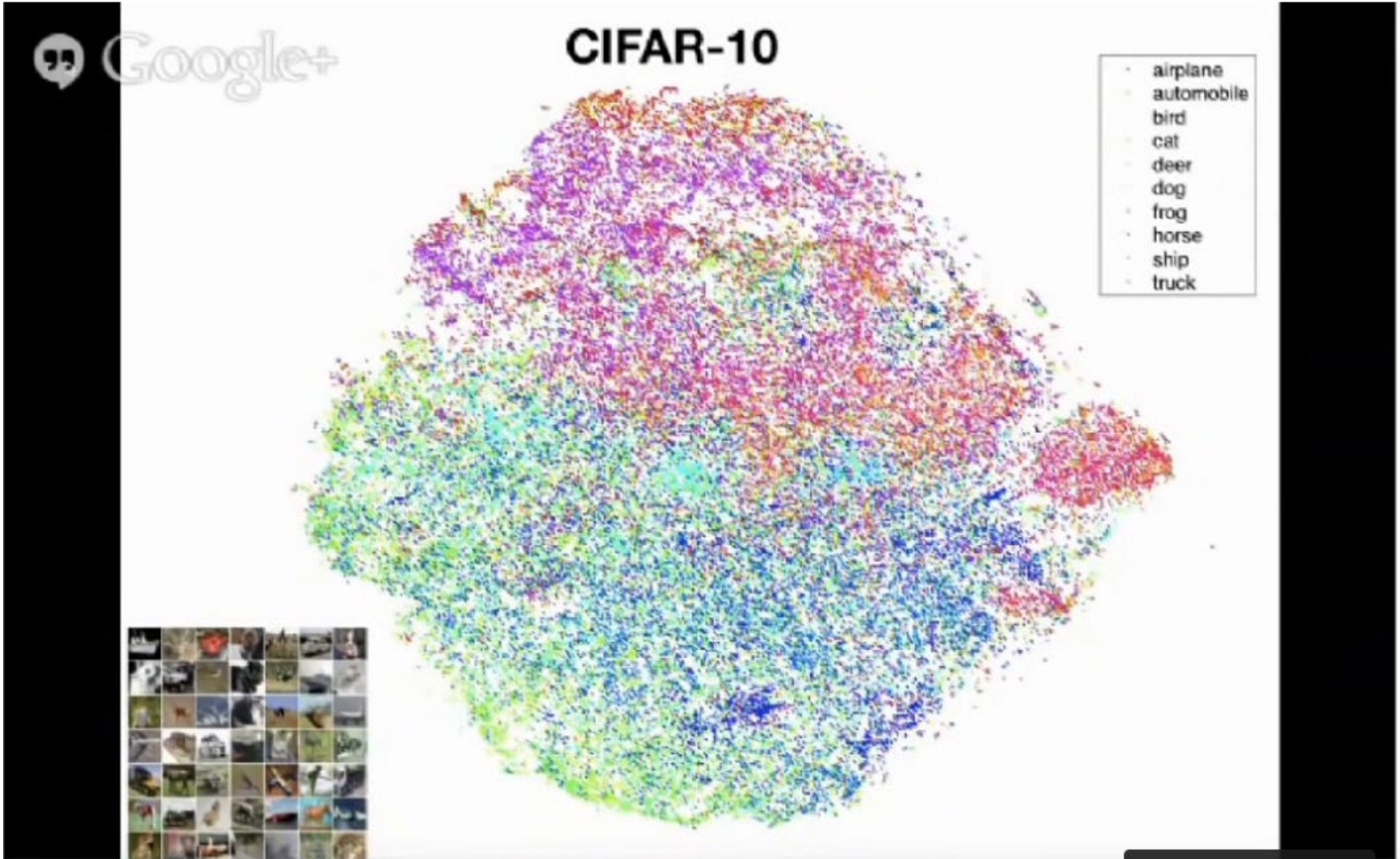
T-SNE on the digits dataset (only using digits 0-5) makes a 2D plot showing clusters of "similar" without knowing the right answers!

https://github.com/amueller/introduction_to_ml_with_python/blob/master/03-unsupervised-learning.ipynb





https://www.youtube.com/watch?v=RJVL80Gg3lA



Best Practices

- Scale your data
- Use pipelines for speed and safety
- Grid search to tune