machine learning in Python and Julia

1. Machine Learning
   - learning from data
2. Working with scikit-learn in Python
   - about scikit-learn
   - clustering, dimension reduction, nearest neighbors
   - regression and cross validation
3. Clustering and Dimension Reduction in Julia
   - the k-means algorithm
   - principal component analysis
   - training and testing

MCS 507 Lecture 26
Mathematical, Statistical and Scientific Software
Jan Verschelde, 20 October 2023

machine learning in Python and Julia

1. Machine Learning
   - learning from data
2. Working with scikit-learn in Python
   - about scikit-learn
   - clustering, dimension reduction, nearest neighbors
   - regression and cross validation
3. Clustering and Dimension Reduction in Julia
   - the k-means algorithm
   - principal component analysis
   - training and testing

learning from experience

A popular quote from computer scientist Tom Mitchell:

Definition

A program can be said to learn from experience E
with respect to some class of tasks T and performance measure P,
if its performance at tasks in T, as measured by P,
improves with experience E.

Example: classifying pictures of dogs and cats.

The quote above is taken from the following book:
Gavin Hackeling: *Mastering Machine Learning with scikit-learn. Apply effective learning algorithms to real-world problems using scikit-learn.* Packt publishing, 2014.

supervised and unsupervised learning

We distinguish between two types of machine learning:

- Supervised learning:
  given are labeled inputs and outputs,
  the program learns from examples of the right answers.
- Unsupervised learning:
  the program does not learn from labeled data.

Two most common supervised machine learning tasks
are classification and regression.

Examples of unsupervised learning
are clustering and dimensionality reduction.

machine learning in Python and Julia

1. Machine Learning
   - learning from data
2. Working with scikit-learn in Python
   - about scikit-learn
   - clustering, dimension reduction, nearest neighbors
   - regression and cross validation
3. Clustering and Dimension Reduction in Julia
   - the k-means algorithm
   - principal component analysis
   - training and testing

about scikit-learn

```
>>> import sklearn
>>> help(sklearn)

Machine learning module for Python
==================================

sklearn is a Python module integrating classical machine
learning algorithms in the tightly-knit world of scientific
Python packages (numpy, scipy, matplotlib).

It aims to provide simple and efficient solutions to learning
problems that are accessible to everybody and reusable in
various contexts: machine-learning as a versatile tool for
science and engineering.

See http://scikit-learn.org for complete documentation.
```

history of the project

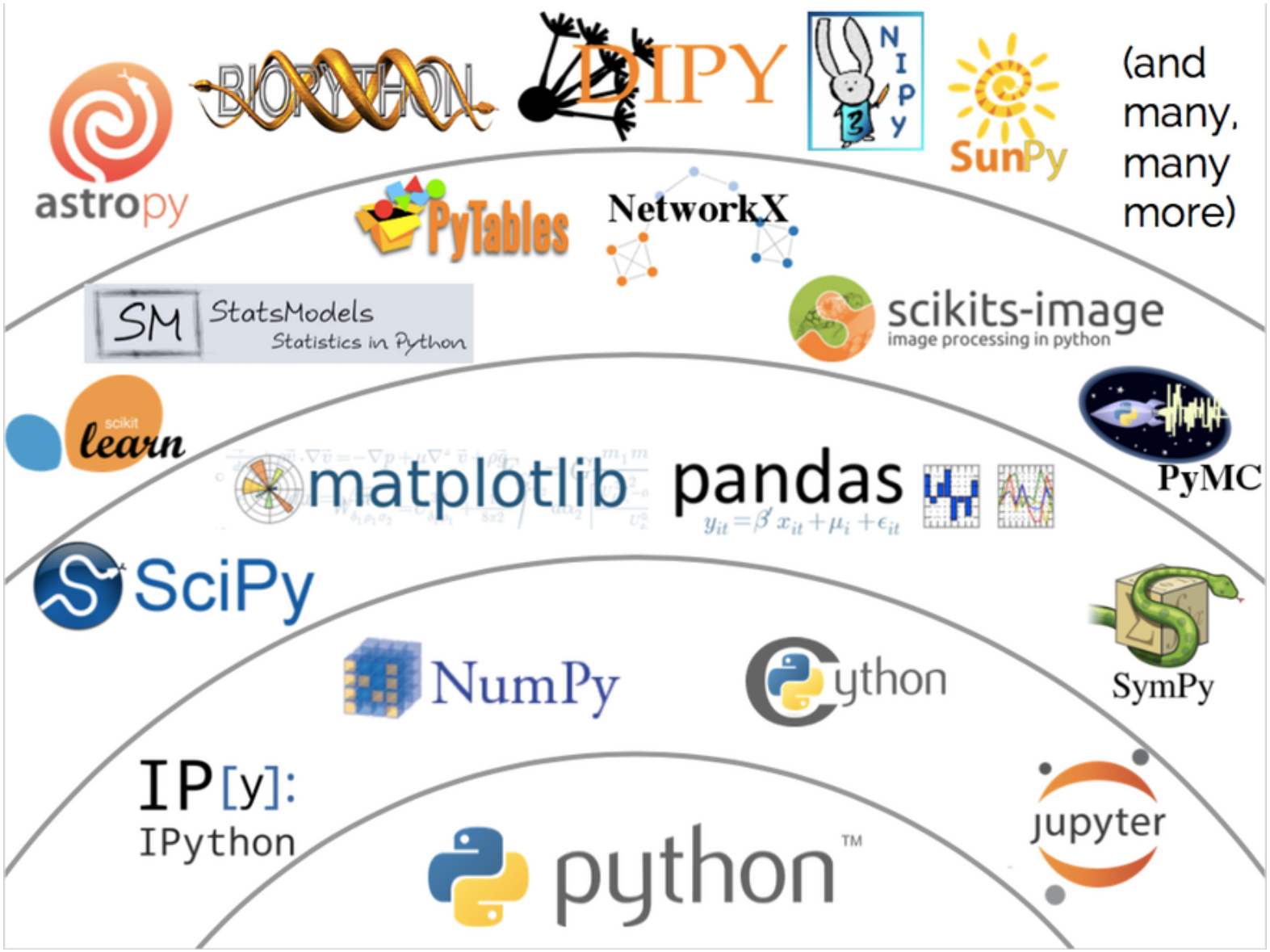
From the scikit-learn-docs pdf (2,503 pages):

- This project was started in 2007 as a Google Summer of Code project by David Cournapeau. Later that year, Matthieu Brucher started work on this project as part of his thesis.
- In 2010 Fabian Pedregosa, Gael Varoquaux, Alexandre Gramfort and Vincent Michel of INRIA took leadership of the project and made the first public release, February the 1st 2010.

F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay **Scikit-learn: Machine Learning in Python.** *Journal of Machine Learning Research* 12: 2825-2830, 2011.

the stack (as of 2015)

picture from the slides of Jake VanderPlas



machine learning in Python and Julia

1. Machine Learning
   - learning from data
2. Working with scikit-learn in Python
   - about scikit-learn
   - clustering, dimension reduction, nearest neighbors
   - regression and cross validation
3. Clustering and Dimension Reduction in Julia
   - the k-means algorithm
   - principal component analysis
   - training and testing

the iris dataset

```
>>> from sklearn import datasets
>>> iris = datasets.load_iris()
>>> type(iris)
<class 'sklearn.utils.Bunch'>
>>> print(iris.DESCR)
.. _iris_dataset:

Iris plants dataset
--------------------
```

The data set comes from 1936 paper of Fisher on the iris plant.

The iris is a classic in the field of pattern recognition.

the petal and sepal parts of a flower

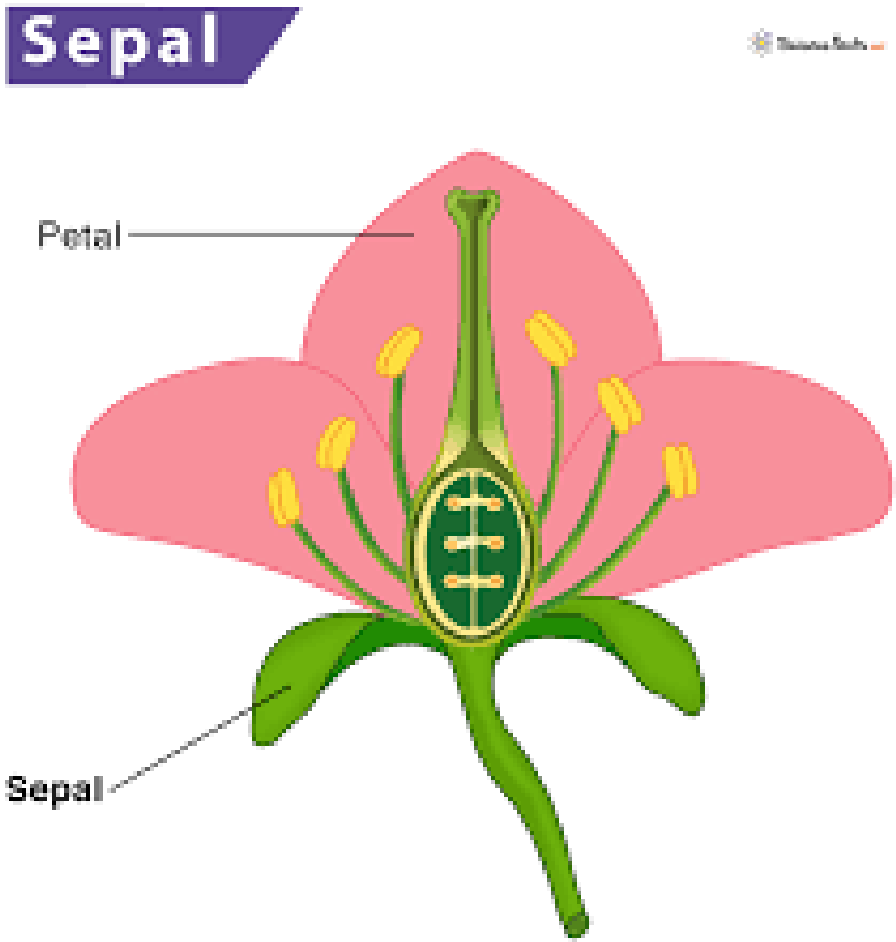


Image from `ScienceFacts.net`.

features and target

The Iris plants data is summarized below.

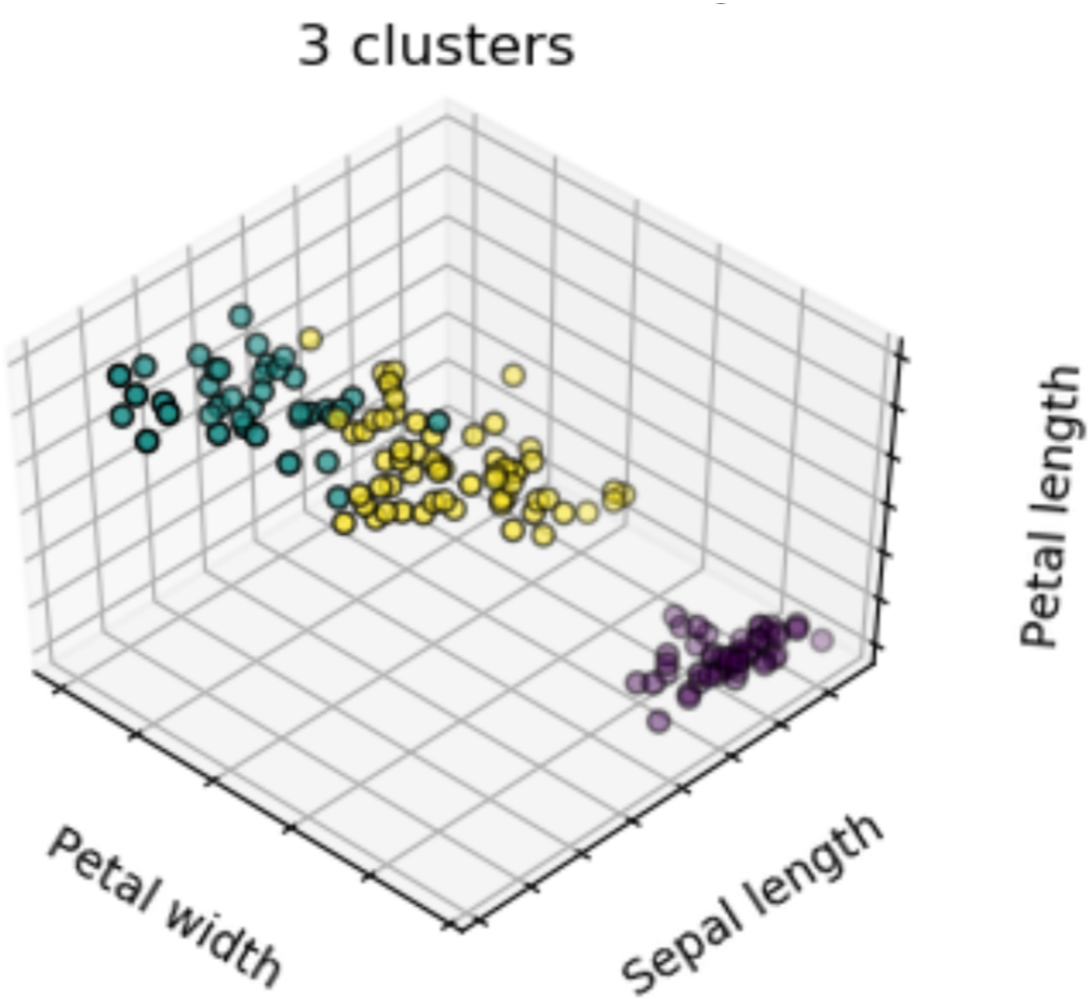
- There are 150 instances, partitioned into three classes. We have 50 instances of each class with names `"Iris-setosa"`, `"Iris-versicolor"`, and `"Iris-virginica"`.
- Each instance has four attributes: sepal length and width, petal length and width (measured in cm).
- We have features and labels:
  1. The attributes of each plant are called *features*, represented by a 4-by-150 floating point matrix.
  2. The names of the plants are called *labels*, represented by an array of 150 strings.

  We can view features as input X and labels as output Y.

examining the data

```
>>> data = iris.data
>>> type(data)
<class 'numpy.ndarray'>
>>> data.shape
(150, 4)
>>> ydata = iris.target
>>> type(ydata)
<class 'numpy.ndarray'>
>>> ydata.shape
(150,)
>>> import numpy as np
>>> np.unique(ydata)
array([0, 1, 2])
```

three clusters



from the scikit-learn-docs, page 181

application of the k-means algorithm

```
>>> from sklearn import cluster
>>> k_means = cluster.KMeans(n_clusters=3)
>>> k_means.fit(data)
KMeans(algorithm='auto', copy_x=True,
       init='k-means++', max_iter=300,
       n_clusters=3, n_init=10, n_jobs=None,
       precompute_distances='auto',
       random_state=None, tol=0.0001,
       verbose=0)
>>> print(k_means.labels_[::10])
[1 1 1 1 1 2 2 2 2 2 0 0 0 0 0]
>>> print(ydata[::10])
[0 0 0 0 0 1 1 1 1 1 2 2 2 2 2]
```

dimension reduction

To plot the 4-dimensional data of the iris data set,
we can use PCA (Principal Component Analysis).
The code comes from page 149 on the book by Hackeling.

```
from sklearn.decomposition import PCA
from sklearn.datasets import load_iris
data = load_iris()
y = data.target
X = data.data
# dimension reduction
pca = PCA(n_components=2)
reduced_X = pca.fit_transform(X)
```

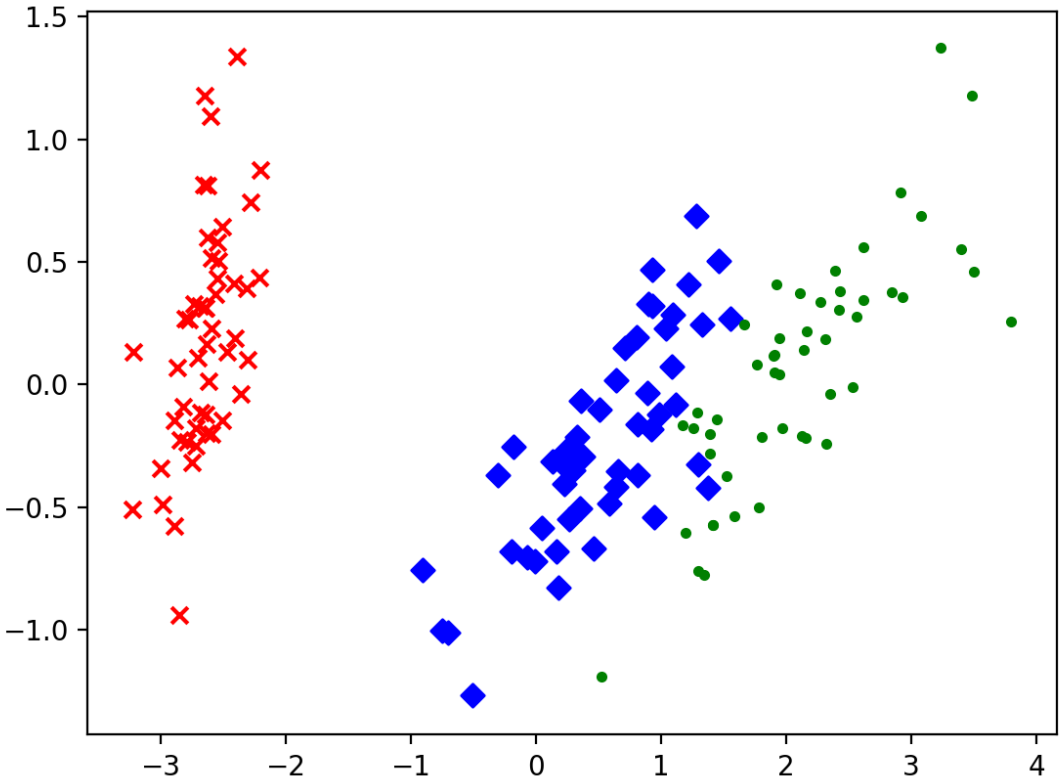
assemble and plot the reduced data

```
red_x, red_y = [], []
blue_x, blue_y = [], []
green_x, green_y = [], []
for i in range(len(reduced_X)):
    if y[i] == 0:
        red_x.append(reduced_X[i][0])
        red_y.append(reduced_X[i][1])
    elif y[i] == 1:
        blue_x.append(reduced_X[i][0])
        blue_y.append(reduced_X[i][1])
    else:
        green_x.append(reduced_X[i][0])
        green_y.append(reduced_X[i][1])
```

plotting with matplotlib

```
import matplotlib.pyplot as plt
plt.scatter(red_x, red_y, c='r', marker='x')
plt.scatter(blue_x, blue_y, c='b', marker='D')
plt.scatter(green_x, green_y, c='g', marker='.')
plt.show()
```

the two dimensional plot



Such a plot may help at determining the number of clusters.

the setup

```
>>> from sklearn import datasets
>>> import numpy as np
>>> iris = datasets.load_iris()
>>> iris_X = iris.data
>>> iris_y = iris.target
>>> np.random.seed(0)
>>> indices = np.random.permutation(len(iris_X))
>>> iris_X_train = iris_X[indices[:-10]]
>>> iris_y_train = iris_y[indices[:-10]]
>>> iris_X_test = iris_X[indices[-10:]]
>>> iris_y_test = iris_y[indices[-10:]]
```

running the classification algorithm

```
>>> from sklearn.neighbors import KNeighborsClassifier
>>> knn = KNeighborsClassifier()
>>> knn.fit(iris_X_train, iris_y_train)
KNeighborsClassifier(algorithm='auto', leaf_size=30,
                     metric='minkowski', metric_params=None,
                     n_jobs=None, n_neighbors=5, p=2,
                     weights='uniform')
>>> KNeighborsClassifier(algorithm='auto',
... leaf_size=30, metric='minkowski',
... metric_params=None, n_jobs=None, n_neighbors=5, p=2,
... weights='uniform')
KNeighborsClassifier(algorithm='auto', leaf_size=30,
                     metric='minkowski', metric_params=None,
                     n_jobs=None, n_neighbors=5, p=2,
                     weights='uniform')
```

predicting and verification

```
>>> knn.predict(iris_X_test)
array([1, 2, 1, 0, 0, 0, 2, 1, 2, 0])
>>> iris_y_test
array([1, 1, 1, 0, 0, 0, 2, 1, 2, 0])
>>>
```

machine learning in Python and Julia

1. Machine Learning
   - learning from data
2. Working with scikit-learn in Python
   - about scikit-learn
   - clustering, dimension reduction, nearest neighbors
   - regression and cross validation
3. Clustering and Dimension Reduction in Julia
   - the k-means algorithm
   - principal component analysis
   - training and testing

house prices dataset of California

```
>>> from sklearn.datasets \
import fetch_california_housing
>>> housing = fetch_california_housing()
>>> print(housing.DESCR)
```

There are 20640 instances with 8 attributes.
The medial house value is usually the target.

The data was derived from the 1990 U.S. census.

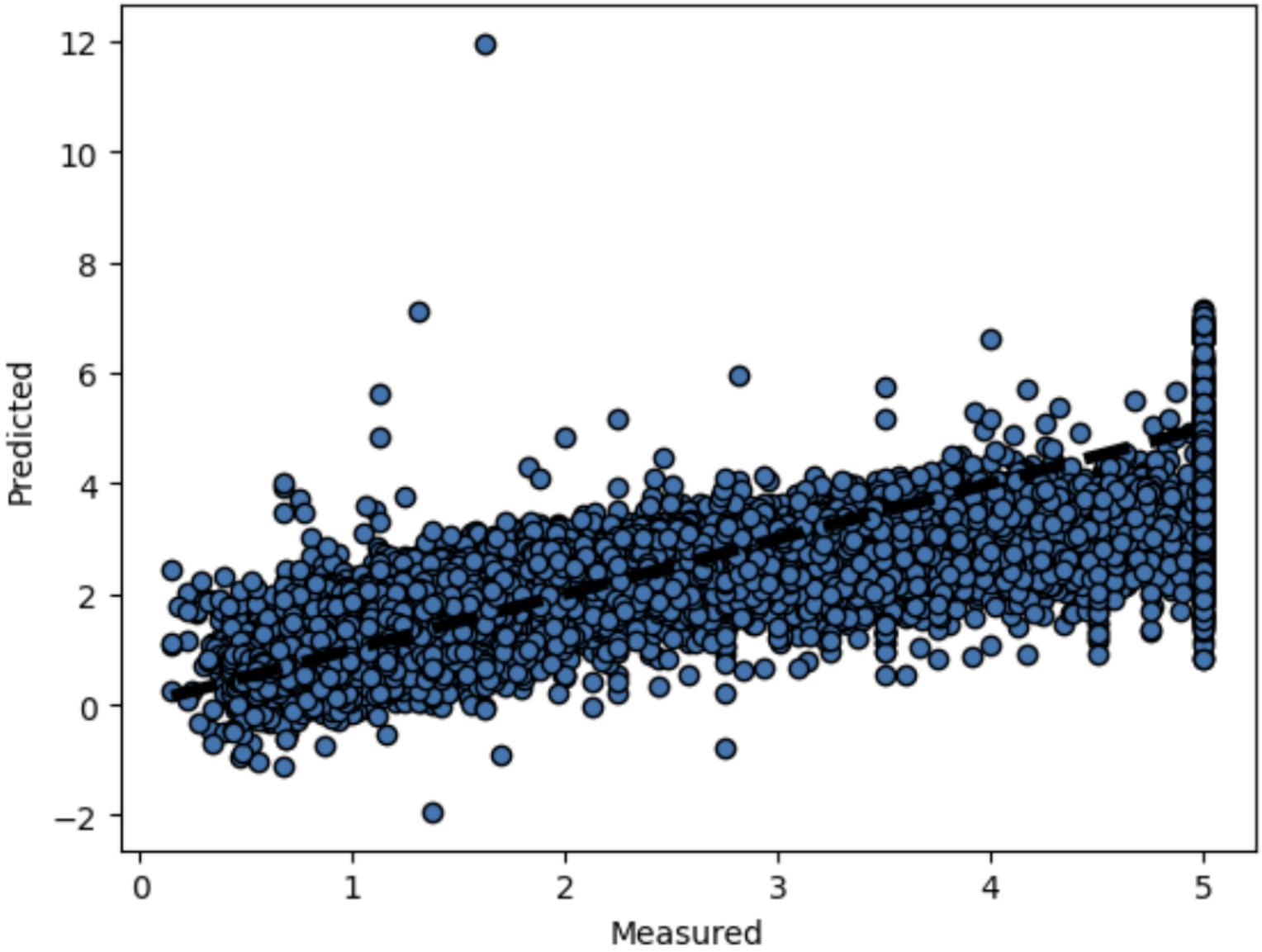
linear regression

```
from sklearn import datasets
from sklearn.model_selection import cross_val_predict
from sklearn import linear_model
import matplotlib.pyplot as plt
lr = linear_model.LinearRegression()
from sklearn.datasets import fetch_california_housing
housing = fetch_california_housing()
y = housing.target

# cross_val_predict returns an array of the same
# size as `y` where each entry is a prediction
# obtained by cross validation

predicted = cross_val_predict(lr, housing.data, y, \
    cv=10)
```

the plotting instructions

```
fig, ax = plt.subplots()
ax.scatter(y, predicted, edgecolors=(0, 0, 0))
ax.plot([y.min(), y.max()], [y.min(), y.max()], \
    'k--', lw=4)
ax.set_xlabel('Measured')
ax.set_ylabel('Predicted')
plt.show()
```

visualization of predicted errors



machine learning in Python and Julia

1. Machine Learning
   - learning from data
2. Working with scikit-learn in Python
   - about scikit-learn
   - clustering, dimension reduction, nearest neighbors
   - regression and cross validation
3. Clustering and Dimension Reduction in Julia
   - the k-means algorithm
   - principal component analysis
   - training and testing

the Iris plants data in Julia

```
using MLDatasets: Iris

F, L = Iris(as_df=false)[:]
```

Then:

- The `F` is a 4-by-150 matrix of `Float64` elements.
- The `L` is a 150-element vector of elements of type `String`.

the k-means algorithm in Julia

The k-means algorithm will be illustrated on the Iris plants data:

1. The features are in the 4-by-150 matrix `F`.
2. The labels are in the 150-element vector `L`.

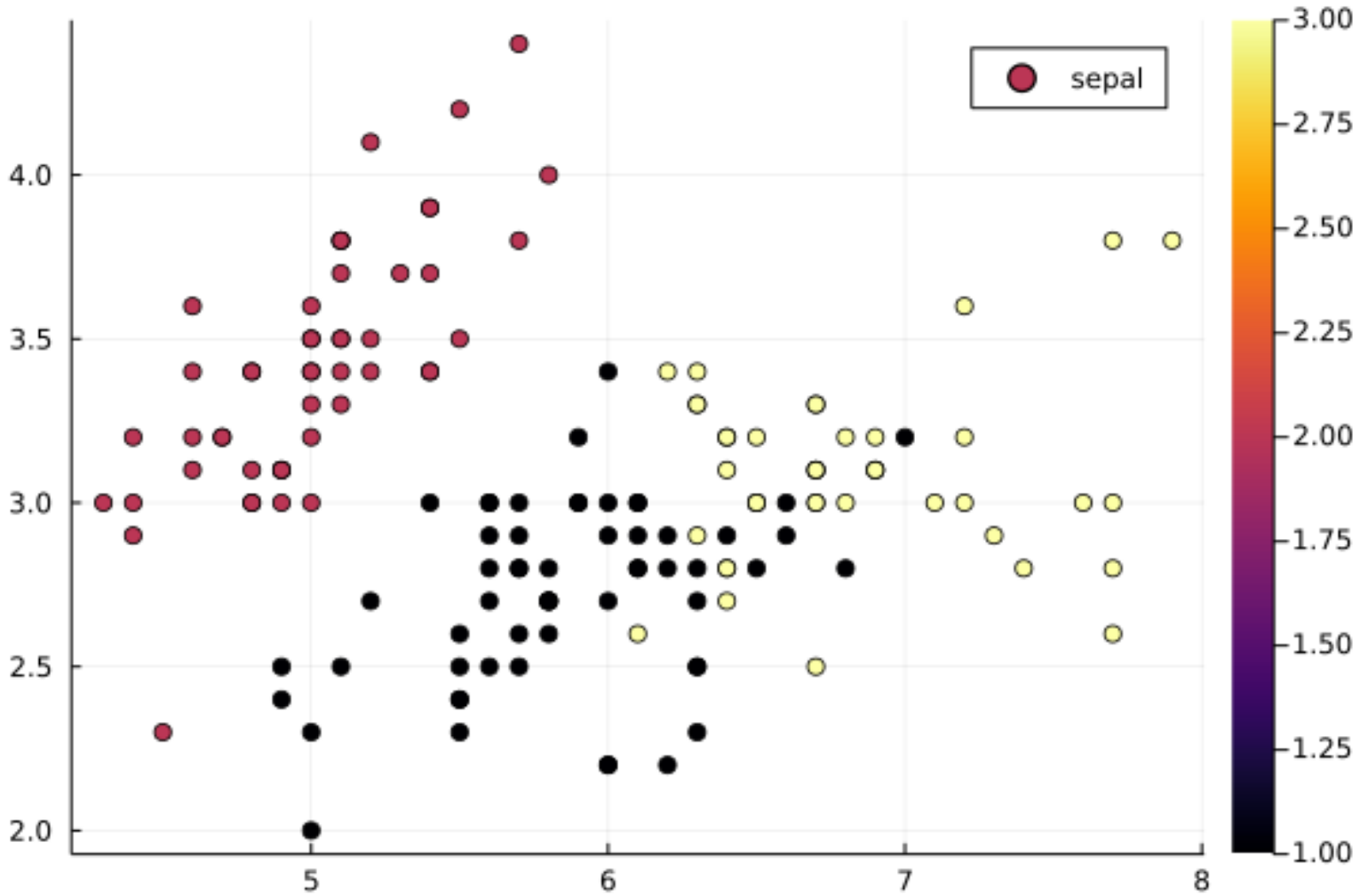
We use the `Clustering` package.

```
using Clustering
C = kmeans(F, 3)
dump(C)
```

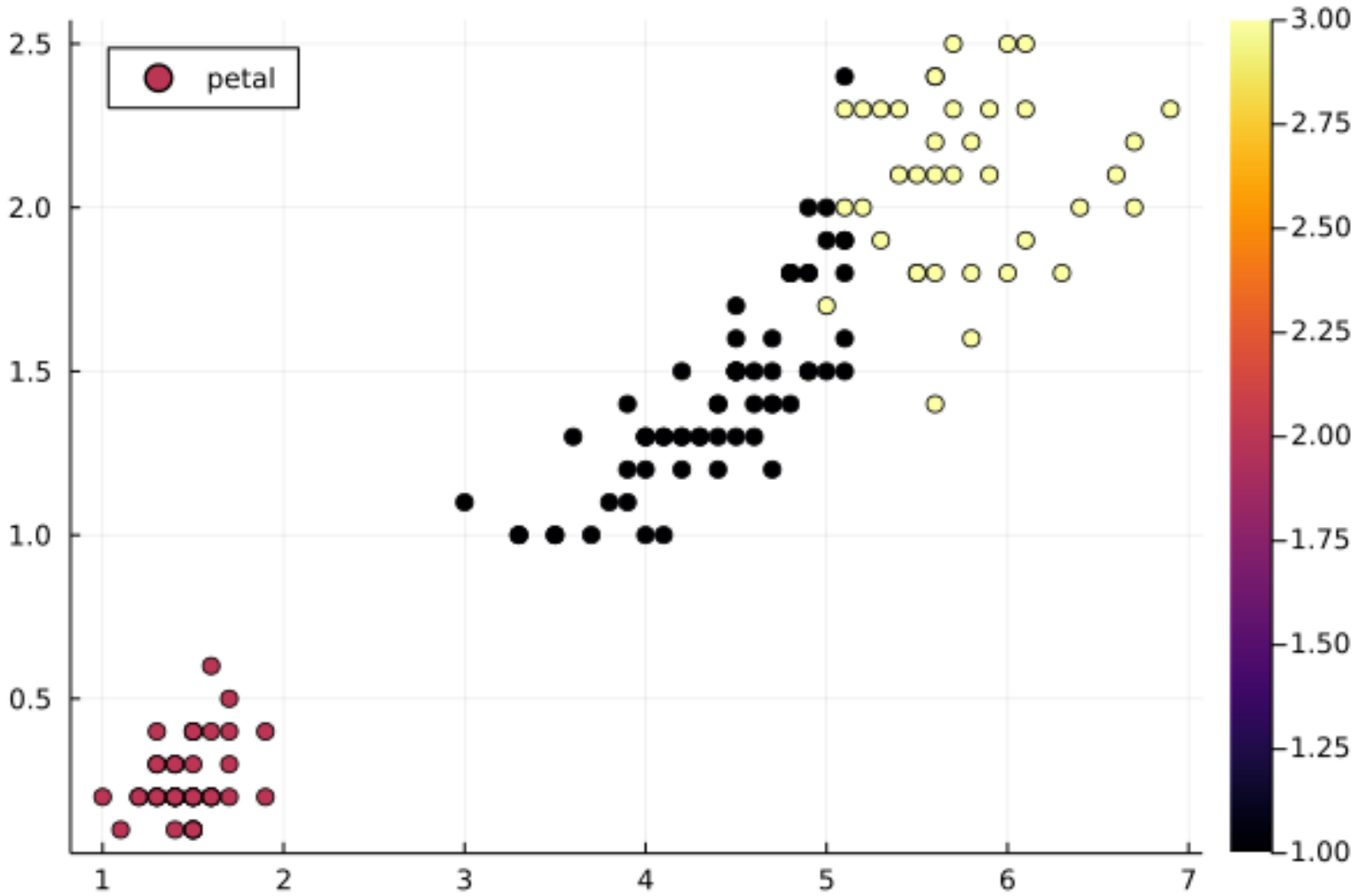
In the returned C.assignments we have 150 integers, 1, 2, or 3, corresponding to the three computed centroids in the data.

The `C.counts` are `[62, 50, 38]`, for one run.

assignments plotted versus sepal data



assignments plotted versus petal data



machine learning in Python and Julia

1. Machine Learning
   - learning from data
2. Working with scikit-learn in Python
   - about scikit-learn
   - clustering, dimension reduction, nearest neighbors
   - regression and cross validation
3. Clustering and Dimension Reduction in Julia
   - the k-means algorithm
   - principal component analysis
   - training and testing

principal component analysis

Principal Component Analysis (PCA) derives an orthogonal projection to convert observations to linearly uncorrelated variables, called *principal components*.

Given a PCA model, we transform the observations x

$$y = P^T(x - \mu)$$

into its principal components y, with a projection matrix P.

The observations can be reconstructed from principal components:

$$\widetilde{x} = Py + \mu.$$

In Julia, the principal component analysis is in `MultivariateStats`.

Steps in the Principal Component Analysis

Three steps:

1. Standardize the data, via the formula

$$z = \frac{x - \text{mean}}{\text{standard deviation}},$$

   applied to each number x in the input.

2. Setup the covariance matrix C. For each pair (z_i, z_j),

$$\text{cov}(z_i, z_j) \text{ is the } (i,j)\text{-th element of the matrix.}$$

   The matrix C is symmetric and positive definite.

3. Compute the eigenvectors and eigenvalues of C.
   Sort the eigenvectors in increasing magnitude of the eigenvalues.

Then the feature vector consist of the first p eigenvectors.

making a PCA model

The output of `M3d = fit(PCA, F; maxoutdim=3)` is

```
PCA(indim = 4, outdim = 3, principalratio = 0.99481691454981)

Pattern matrix (unstandardized loadings):
-----------------------------------------
        PC1          PC2          PC3
-----------------------------------------
1    0.743227    0.323137   -0.162808
2   -0.169099    0.359152    0.167129
3    1.76063   -0.0865096   0.0203228
4    0.737583  -0.0367692   0.153858
-----------------------------------------

Importance of components:
-----------------------------------------------------------
                                PC1         PC2          PC3
-----------------------------------------------------------
SS Loadings (Eigenvalues)   4.22484    0.242244    0.0785239
Variance explained          0.924616   0.0530156   0.0171851
Cumulative variance         0.924616   0.977632    0.994817
Proportion explained        0.929434   0.0532918   0.0172747
Cumulative proportion       0.929434   0.982725    1.0
-----------------------------------------------------------
```

transforming the data for a plot

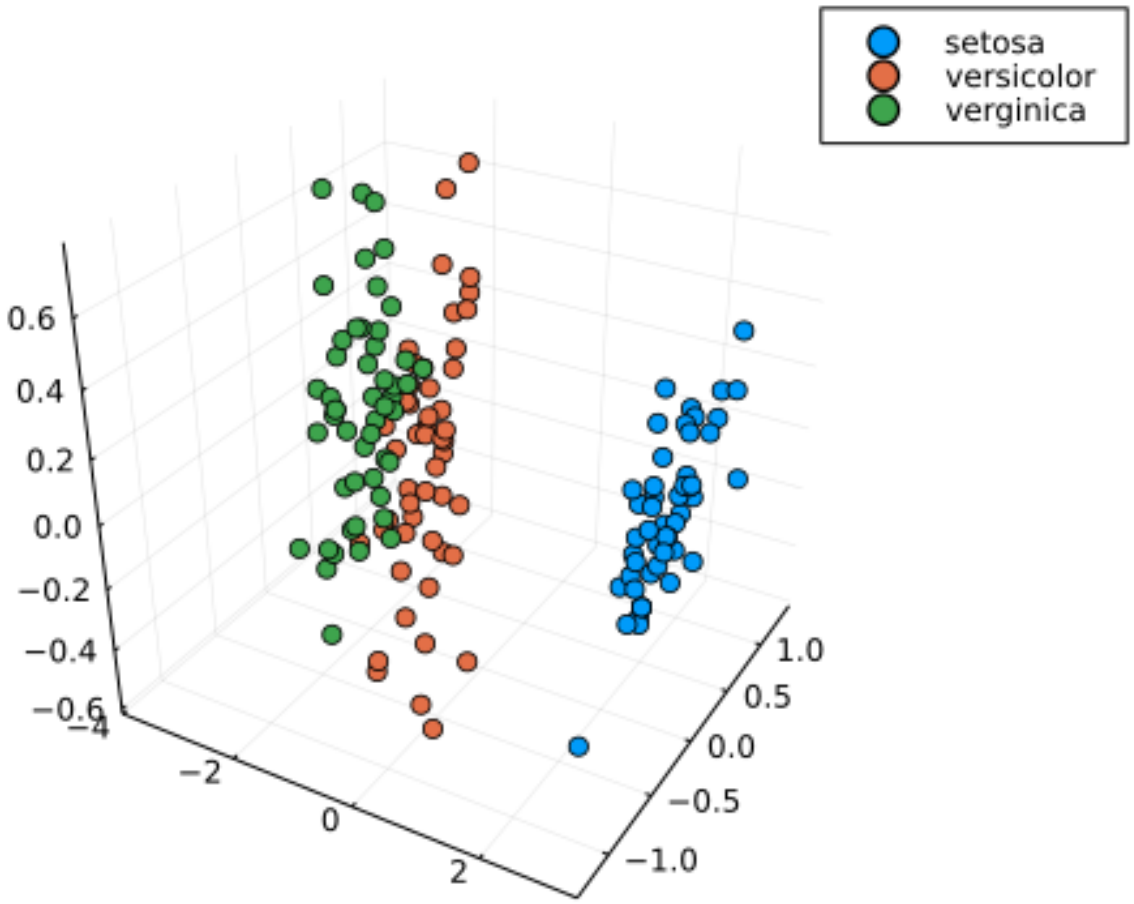
The PCA model `M3d` projects the 4-by-150 matrix `F` onto a three dimensional matrix of points:

```
F3d = transform(M3d, F)
```

The data was sorted according to the plant class, so we plot the groups of 50 as follows:

```
scatter(F3d[1,1:50], F3d[2,1:50], F3d[3,1:50],
        label="setosa")
scatter!(F3d[1,51:100], F3d[2,51:100],
         F3d[3,101:150], label="versicolor")
scatter!(F3d[1,101:150], F3d[2,101:150],
         F3d[3,101:150], label="verginica")
```

the data reduced to three dimensions



machine learning in Python and Julia

1. Machine Learning
   - learning from data
2. Working with scikit-learn in Python
   - about scikit-learn
   - clustering, dimension reduction, nearest neighbors
   - regression and cross validation
3. Clustering and Dimension Reduction in Julia
   - the k-means algorithm
   - principal component analysis
   - training and testing

training and testing

Learning in a supervised setting is a 2-step process:

1. Use one part of the data set to train the model.
2. Test the model on the other part of the data set.

Applied to the Iris plants data:

1. Train a model with the odd indexed features.
2. Test the model on the even indexed features.

With principal component analysis:

1. Train the PCA model using the odd indexed features.
2. Transform the even indexed features and reconstruct.

exercises: recognizing handwritten digits

Another well known data set in machine learning is the set of hand written digits.

1. In Python, do

   ```
   from sklearn import datasets
   digits = datasets.load_digits()
   ```

   Consult the documentation of scikit-learn to classify the data.

2. In Julia, do

   ```
   using MLDatasets: MNIST
   dataset = MNIST(:train)
   ```

   Apply machine learning tools in Julia to classify the data.