ML23

Lecture notes for Fall 2023 at Arkansas Tech University

Xinli Xiao

2023-05-25

Table of contents

Preface 4

1 Introduction 5

2 What is Machine Learning? 6

2.1 Types of Machine Learning Systems . . . 6

2.1.1 Tasks for Supervised Learning . . . 7

2.1.2 Classification based on complexity . . . 7

3 Basic setting for Machine learning problems 8

3.1 Input and output data structure . . . 8

3.2 Parameters and hyperparameters . . . 10

3.3 Evaluate a Machine Learning model . . . 10

3.4 Workflow in developing a machine learning application . . . 11

4 Python quick guide 12

4.1 Python Notebook . . . 12

4.2 Python fundamentals . . . 14

4.2.1 `list` and `dict` . . . 14

4.2.2 Reading files . . . 15

4.2.3 Writing files . . . 15

4.2.4 Relative paths . . . 16

4.2.5 . . . 16

4.3 Some additional topics . . . 16

4.3.1 Package management and Virtual environment . . . 16

4.3.2 Version Control . . . 17

5 Exercises 18

5.1 Python Notebook . . . 18

5.2 Basic Python . . . 18

6 k-Nearest Neighbors algorithm (k-NN) 21

6.1 k-Nearest Neighbors Algorithm (k-NN) . . . 21

6.1.1 Ideas . . . 21

6.1.2 The Algorithm . . . 22

6.1.3 Details . . . 22

6.1.4 The codes 23
6.1.5 `sklearn` packages 23
6.1.6 Normalization 24
6.2 k-NN Project 1: `iris` Classification 25
6.2.1 Explore the dataset 25
6.2.2 Apply k-NN model from `sklearn` 28
6.2.3 Using data pipeline 28
6.2.4 Visualize the Decision boundary 29
6.2.5 Choosing a `k` value 31
6.3 k-NN Project 2: Dating Classification 32
6.3.1 Background 32
6.3.2 Look at Data 32
6.3.3 Applying kNN 34
6.3.4 Choosing `k` Value 35
6.4 Exercises and Projects 36

Preface

This is the lecture notes for STAT 2304 Programming languages for Data Science Spring 2023 at ATU. If you have any comments/suggetions/concers about the notes please contact us at xxiao@atu.edu or wjia@atu.edu.

1 Introduction

In this Chapter we will discuss - What is Machine Learning? - What do typical Machine Learning problems look like? - What is the basic structure of Machine Learning models? - What is the basic work flow to use Machine Learning to solve problems? - Some supplementary materials, such as Linear Algebra and Python.

2 What is Machine Learning?

Machine Learning is the science (and art) of programming computers so they can *learn from data* {cite:p}`Ger2019`.

Here is a slightly more general definition:

```
[Machine Learning is the] field of study that gives computers the ability to learn without be

-- Arthur Samuel, 1959
```

This "without being explicitly programmed to do so" is the essential difference between Machine Learning and usual computing tasks. The usual way to make a computer do useful work is to have a human programmer write down rules — a computer program — to be followed to turn input data into appropriate answers. Machine Learning turns this around: the machine looks at the input data and the expected task outcome, and figures out what the rules should be. A Machine Learning system is *trained* rather than explicitly programmed. It's presented with many examples relevant to a task, and it finds statistical structure in these examples that eventually allows the system to come up with rules for automating the task {cite:p}`Cho2021`.

2.1 Types of Machine Learning Systems

There are many different types of Machine Learning systems that it is useful to classify them in braod categories, based on different criteria. These criteria are not exclusive, and you can combine them in any way you like.

The most popular criterion for Machine Learning classification is the amount and type of supervision they get during training. In this case there are four major types.

```
Supervised Learning
    The training set you feed to the algorithm includes the desired solutions. The machine

Unsupervised Learning
    In Unsupervised Learning, the data provided doesn't have class information or desired

Reinforcement Learning
```

```
    In Reinforcement Learning, there is a reward system to measure how well the machine pe

Semisupervised Learning
    This is actually a combination of Supervised Learning and Unsupervised Learning, that
```

2.1.1 Tasks for Supervised Learning

As mentioned above, for Supervised Learning, there are two typical types of tasks:

```
Classification
    It is the task of predicting a discrete class labels. A typical classification problem

Regression
    It is the task of predicting a continuous quantity. A typical regression problem is to
```

There are a lot of other tasks that are not directly covered by these two, but these two are the most classical Supervised Learning tasks.

```
In this course we will mainly focus on **Supervised Classification problems**.
```

2.1.2 Classification based on complexity

Along with the popularity boost of deep neural network, there comes another classificaiton: shallow learning vs. deep learning. Basically all but deep neural network belongs to shallow learning. Although deep learning can do a lot of fancy stuffs, shallow learning is still very good in many cases. When the performance of a shallow learning model is good enough comparing to that of a deep learning model, people tend to use the shallow learning since it is usually faster, easier to understand and easier to modify.

3 Basic setting for Machine learning problems

> We by default assume that we are dealing with a **Supervised** **Classification** problem.

3.1 Input and output data structure

Since we are dealing with Supervised Classification problems, the desired solutions are given. These desired solutions in Classification problems are also called *labels*. The properties that the data are used to describe are called *features*. Both features and labels are usually organized as row vectors.

Example 3.1. The example is extracted from {cite:p}`Har2012`. There are some sample data shown in the following table. We would like to use these information to classify bird species.

“‘iukpyhvswiya Bird species classification based on four features :header-rows: 1

- - Weight (g)
  - Wingspan (cm)
  - Webbed feet?
  - Back color
  - Species
- - 1000.1
  - 125.0
  - No
  - Brown
  - Buteo jamaicensis
- - 3000.7
  - 200.0
  - No
  - Gray
  - Sagittarius serpentarius
- - 3300.0
  - 220.3

  - No
  - Gray
  - Sagittarius serpentarius

- - 4100.0
  - 136.0
  - Yes
  - Black
  - Gavia immer

- - 3.0
  - 11.0
  - No
  - Green
  - Calothorax lucifer

- - 570.0
  - 75.0
  - No
  - Black
  - Campephilus principalis "‘ The first four columns are features, and the last column is the label. The first two features are numeric and can take on decimal values. The third feature is binary that can only be 1 (Yes) or 0 (No). The fourth feature is an enumeration over the color palette. You may either treat it as categorical data or numeric data, depending on how you want to build the model and what you want to get out of the data. In this example we will use it as categorical data that we only choose it from a list of colors (1 — Brown, 2 — Gray, 3 — Black, 4 — Green).

Then we are able to transform the above data into the following form:

"‘iukpyhvswiya Vectorized Bird species data :header-rows: 1

- - Features
  - Labels

- - $\begin{bmatrix}1001.1 & 125.0 & 0 & 1\end{bmatrix}$
  - 1

- - $\begin{bmatrix}3000.7 & 200.0 & 0 & 2\end{bmatrix}$
  - 2

- - $\begin{bmatrix}3300.0 & 220.3 & 0 & 2\end{bmatrix}$
  - 2

- - $\begin{bmatrix}4100.0 & 136.0 & 1 & 3\end{bmatrix}$
  - 3

- - $\begin{bmatrix}3.0 & 11.0 & 0 & 4\end{bmatrix}$

  - 4

- - $\begin{bmatrix}570.0 & 75.0 & 0 & 3\end{bmatrix}$
  - 5 “‘

Then the Supervised Learning problem is stated as follows: Given the features and the labels, we would like to find a model that can classify future data.

3.2 Parameters and hyperparameters

A model parameter is internal to the model and its value is learned from the data.

A model hyperparameter is external to the model and its value is set by people.

For example, assume that we would like to use Logistic regression to fit the data. We set the learning rate is `0.1` and the maximal iteration is `100`. After the computations are done, we get a the model

$$y = \sigma(0.8 + 0.7x).$$

The two cofficients 0.8 and 0.7 are the parameters of the model. The model `Logistic regression`, the learning rate `0.1` and the maximal iteration `100` are all hyperparametrs. If we change to a different set of hyperparameters, we may get a different model, with a different set of parameters.

The details of Logistic regression will be discussed in {numref}`Chapter %s<chapter-log-reg>`.

3.3 Evaluate a Machine Learning model

Once the model is built, how do we know that it is good or not? The naive idea is to test the model on some brand new data and check whether it is able to get the desired results. The usual way to achieve it is to split the input dataset into three pieces: *training set*, *validation set* and *test set*.

The model is initially fit on the training set, with some arbitrary selections of hyperparameters. Then hyperparameters will be changed, and new model is fitted over the training set. Which set of hyperparameters is better? We then test their performance over the validation set. We could run through a lot of different combinations of hyperparameters, and find the best performance over the validation set. After we get the best hyperparameters, the model is selcted, and we fit it over the training set to get our model to use.

To compare our model with our models, either our own model using other algorithms, or models built by others, we need some new data. We can no longer use the training set and the validation set since all data in them are used, either for training or for hyperparameters tuning. We need to use the test set to evaluate the “real performance” of our data.

To summarize:

- Training set: used to fit the model;
- Validation set: used to tune the hyperparameters;
- Test set: used to check the overall performance of the model.

The validation set is not always required. If we use cross-validation technique for hyperparameters tuning, like `sklearn.model_selection.GridSearchCV()`, we don’t need a separated validation set. In this case, we will only need the training set and the test set, and run `GridSearchCV` over the training set. The cross-validation will be discussed in {numref}`Section %s<section-cross-validation>`.

The sizes and strategies for dataset division depends on the problem and data available. It is often recommanded that more training data should be used. The typical distribution of training, validation and test is $(6:3:1)$, $(7:2:1)$ or $(8:1:1)$. Sometimes validation set is discarded and only training set and test set are used. In this case the distribution of training and test set is usually $(7:3)$, $(8:2)$ or $(9:1)$.

3.4 Workflow in developing a machine learning application

The workflow described below is from {cite:p}`Har2012`.

1. Collect data.
2. Prepare the input data.
3. Analyze the input data.
4. Train the algorithm.
5. Test the algorithm.
6. Use it.

In this course, we will mainly focus on Step 4 as well Step 5. These two steps are where the “core” algorithms lie, depending on the algorithm. We will start from the next Chapter to talk about various Machine Learning algorithms and examples.

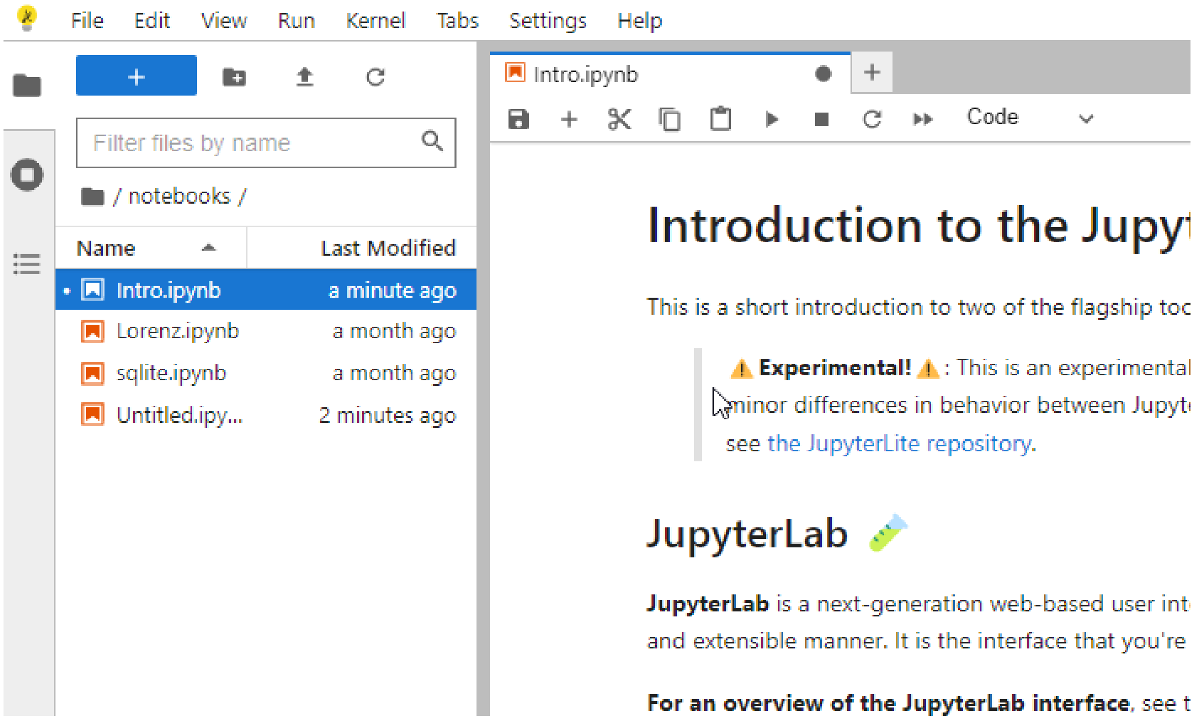
4 Python quick guide

4.1 Python Notebook

We mainly use Python Notebook (.ipynb) to write documents for this course. Currently all main stream Python IDE support Python Notebook. All of them are not entirely identical but the differences are not huge and you may choose any you like.

One of the easiest ways to use Python Notebook is through JupyterLab. The best part about it is that you don't need to worry about installation and configuration in the first place, and you can directly start to code.

Click the above link and choose JupyterLab. Then you will see the following page.



Name	Last Modified
Intro.ipynb	a minute ago
Lorenz.ipynb	a month ago
sqlite.ipynb	a month ago
Untitled.ipy...	2 minutes ago

Introduction to the Jupy

This is a short introduction to two of the flagship toc

⚠ **Experimental!** ⚠ : This is an experimental
minor differences in behavior between Jupyt
see the JupyterLite repository.

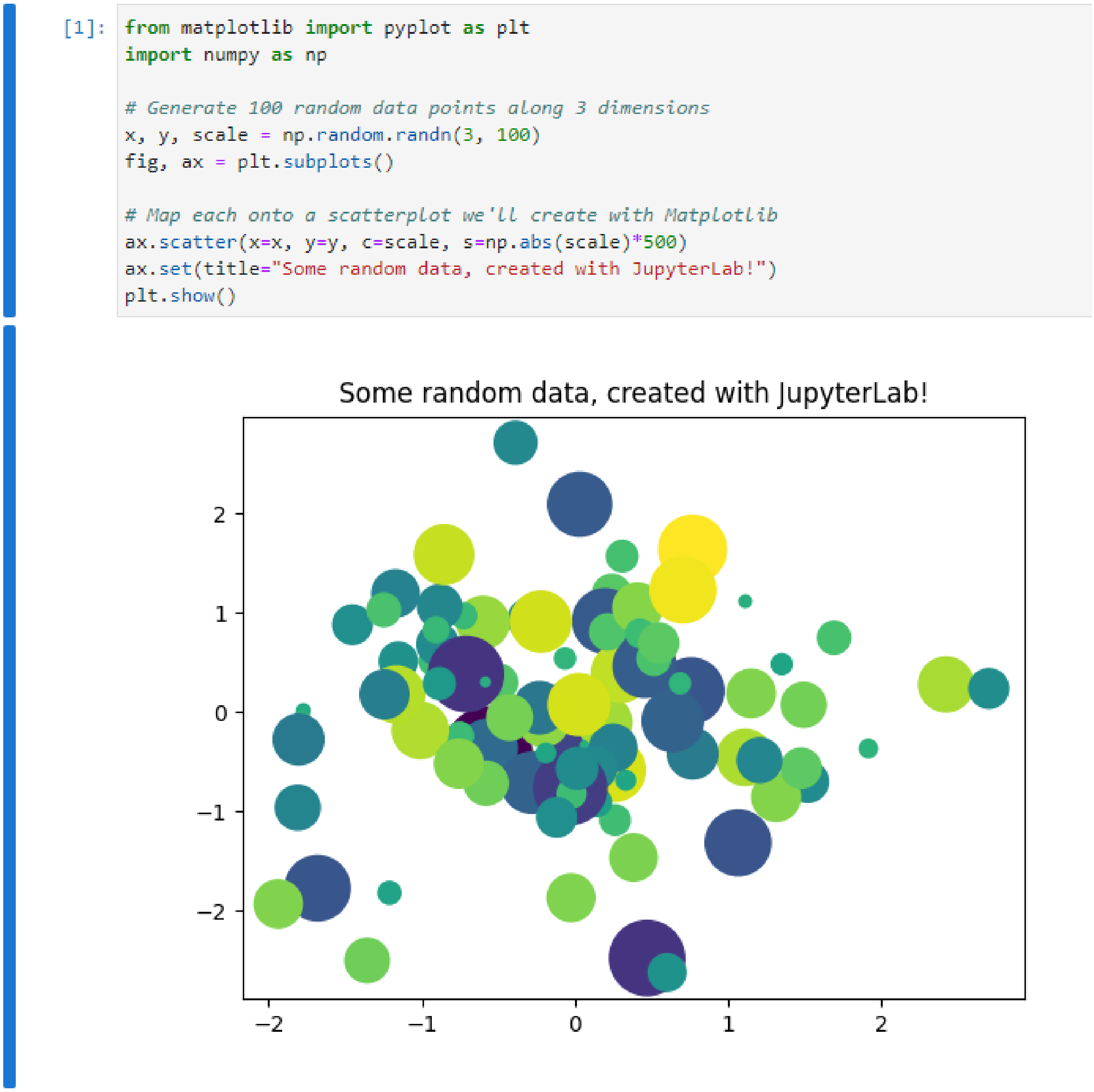
JupyterLab

JupyterLab is a next-generation web-based user int
and extensible manner. It is the interface that you're

For an overview of the JupyterLab interface, see t

The webapp you just started is called JupyterLite. This is a demo version. The full JupyterLab installation instruction can also be found from the link.

There is a small button + under the tab bar. This is the place where you click to start a new cell. You may type codes or markdown documents or raw texts in the cell according to your needs. The drag-down menu at the end of the row which is named `Code` or `Markdown` or `Raw` can help you make the switch. Markdown is a very simple light wighted language to write documents. In most cases it behaves very similar to plain texts. Codes are just regular Python codes (while some other languages are supported). You may either use the triangle button in the menu to execute the codes, or hit `shift + enter`.



```
from matplotlib import pyplot as plt
import numpy as np

# Generate 100 random data points along 3 dimensions
x, y, scale = np.random.randn(3, 100)
fig, ax = plt.subplots()

# Map each onto a scatterplot we'll create with Matplotlib
ax.scatter(x=x, y=y, c=scale, s=np.abs(scale)*500)
ax.set(title="Some random data, created with JupyterLab!")
plt.show()
```

JupyterLite contains a few popular packages. Therefore it is totally ok if you would like to play with some simple things. However since it is an online evironment, it has many limitations. Therefore it is still recommended to set up a local environment once you get familiar with Python Notebook. Please check the following links for some popular choices for notebooks and Python installations in general, either local and online.

- Jupyter Notebook / JupyterLab
- VS Code
- PyCharm
- Google Colab
- Anaconda

4.2 Python fundamentals

We will put some very basic Python commands here for you to warm up. More advanced Python knowledge will be covered during the rest of the semester. The main reference for this part is {cite:p}`Har2012`. ### Indentation Python is using indentation to denote code blocks. It is not convienent to write in the first place, but it forces you to write clean, readable code.

By the way, the `if` and `for` block are actually straightforward.

:gutter: 2

:::{grid-item-card} One! `yzoyqcxwiwyk python if jj < 3: jj = jj print("It is smaller than 3.")`

:::{grid-item-card} Two! `yzoyqcxwiwyk python if jj < 3: jj = jj print("It is smaller than 3.")` ::: ::::

:gutter: 2

:::{grid-item-card} Three! `yzoyqcxwiwyk python for i in range(3): i = i + 1 print(i)`

:::{grid-item-card} Four! `yzoyqcxwiwyk python for i in range(3): i = i + 1 print(i)` ::: :::: Please tell the differences between the above codes.

4.2.1 `list` and `dict`

Here are some very basic usage of lists of dictionaries in Python. "'yzoyqcxwiwyk python newlist = list() newlist.append(1) newlist.append('hello') print(newlist)

newlisttwo = [1, 'hello'] print(newlisttwo)

newdict = dict() newdict['one'] = 'good' newdict[1] = 'yes' print(newdict)

newdicttwo = {'one': 'good', 1: 'yes'} print(newdicttwo)

~~~
### Loop through lists
When creating `for` loops we may let Python directly loop through lists. Here is an example.
```{code-block} python
alist = ['one', 2, 'three', 4]

for item in alist:
    print(item)
~~~

4.2.2 Reading files

There are a lot of functions that can read files. The basic one is to read any files as a big string. After we get the string, we may parse it based on the structure of the data.

The above process sounds complicated. That's why we have so many different functions reading files. Usually they focus on a certain types of files (e.g. spreadsheets, images, etc..), parse the data into a particular data structure for us to use later.

I will mention a few examples.

- `csv` files and `excel` files Both of them are spreadsheets format. Usually we use `pandas.read_csv` and `pandas.read_excel` both of which are from the package `pandas` to read these two types of files.
- images Images can be treated as matrices, that each entry represents one pixel. If the image is black/white, it is represented by one matrix where each entry represents the gray value. If the image is colored, it is represented by three matrices where each entry represents one color. To use which three colors depends on the color map. `rgb` is a popular choice.

  In this course when we need to read images, we usually use `matplotlib.pyplot.imread` from the package `matplotlib` or `cv.imread` from the package `opencv`.
- `.json` files `.json` is a file format to store dictionary type of data. To read a `json` file and parse it as a dictionary, we need `json.load` from the package `json`.

4.2.3 Writing files

- `pandas.DataFrame.to_csv`
- `pandas.DataFrame.to_excel`
- `matplotlib.pyplot.imsave`
- `cv.imwrite`
- `json.dump`

4.2.4 Relative paths

In this course, when reading and writing files, please keep all the files using relative paths. That is, only write the path starting from the working directory.

~~~
Consider the following tasks:

1. Your working directory is `C:/Users/Xinli/projects/`.
2. Want to read a file `D:/Files/example.csv`.
3. Want to generate a file whose name is `result.csv` and put it in a subfoler named `fold

To do the tasks, don't directly run the code `pd.read_csv('D:/Files/example.csv')`. Instea

```{code-block} python
import pandas as pd

df = pd.read_csv('example.csv')
df.to_csv('foldername/result.csv')
```
Please pay attention to how the paths are written.
~~~

4.2.5 .

- class and packages.
- Get access to attributes and methods
- Chaining dots.

4.3 Some additional topics

4.3.1 Package management and Virtual environment

- `conda`
  - `conda create`
    - * `conda create --name myenv`
    - * `conda create --name myenv python=3.9`
    - * `conda create --name myenv --file spec-file.txt`
  - `conda install`
    - * `conda install -c conda-forge numpy`

  - conda activate myenv
  - conda list
    * conda list numpy
    * conda list --explicit > spec-file.txt
  - conda env list

- pip / venv
  - python -m venv newenv
  - newenv\Scripts\activate
  - pip install
  - pip freeze > requirements.txt
  - pip install -r /path/to/requirements.txt
  - deactivate

4.3.2 Version Control

- Git
  - Install
  - git config --list
  - git config --global user.name "My Name"
  - git config --global user.email "myemail@example.com"
- GitHub

5 Exercises

These exercises are from {cite:p}`Klo2021`, {cite:p}`Ger2019` and {cite:p}`Har2012`.

5.1 Python Notebook

“‘vcohkfiwff Hello World!

Please set up a Python Notebook environment and type `print('Hello World!')`.

~~~~~
```{exercise}

Please set up a Python Notebook and start a new virtual environment and type `print('Hello Wo
~~~~~

5.2 Basic Python

“‘vcohkfiwff Play with lists :label: ex1helloworld

Please complete the following tasks. - Write a `for` loop to print values from 0 to 4. - Combine two lists `['apple', 'orange']` and `['banana']` using `+`. - Sort the list `['apple', 'orange', 'banana']` using `sorted()`.

~~~~~
````{solution} ex1helloworld
:class: dropdown

```{code-block} python
for i in range(5):
    print(i)

newlist = ['apple', 'orange'] + ['banana']
~~~~~

```
sorted(['apple', 'orange', 'banana'])
```

Please be careful about the last line. `sorted()` doesn't change the original list. It create a new list. There are some Python functions which change the inputed object in-place. Please read documents on all packages you use to get the desired results.

~~~~
```{exercise} Play with list, dict and pandas.
:label: ex1list

Please complete the following tasks.
- Create a new dictionary `people` with two keys `name` and `age`. The values are all empty
- Add `Tony` to the `name` list in `people`.
- Add `Harry` to the `name` list in `people`.
- Add number 100 to the `age` list in `people`.
- Add number 10 to the `age` list in `people`.
- Find all the keys of `people` and save them into a list `namelist`.
- Convert the dictionary `people` to a Pandas DataFrame `df`.
```
````{solution} ex1list
:class: dropdown
```{code-block} python
import pandas as pd

people = {'name': list(), 'age': list()}
people['name'].append('Tony')
people['name'].append('Harry')
people['age'].append(100)
people['age'].append(10)

namelist = people.keys()

df = pd.DataFrame(people)
```
~~~~

““vcohkfiwff The dataset iris :label: ex1iris

`yzoyqcxwiwyk python from sklearn.datasets import load_iris iris = load_iris()` Please explore this dataset. - Please get the features for `iris` and save it into `X` as an numpy array. - What is the meaning of these features? - Please get the labels for `iris` and save it into `y` as an numpy array. - What is the meaning of labels?

~~~
````{solution} ex1iris
:class: dropdown
We first find that `iris` is a dictionary. Then we can look at all the keys by `iris.keys()`
```{code-block} python
X = iris['data']
print(iris['feature_names'])
y = iris['target']
print(iris['target'])
```
Since the data is already saved as numpy arrays, we don't need to do anything to change its
~~~

“‘vcohkfiwff Play with Pandas :label: ex1pandastitanic Please download the Titanic data file from {Download}here<./assests/datasets/titanic.csv>‘. Then follow the instructions to perform the required tasks.

- Use `pandas.read_csv` to read the dataset and save it as a dataframe object `df`.
- Change the values of the `Sex` column that `male` is `0` and `female` is `1`.
- Pick the columns `Pclass`, `Sex`, `Age`, `Siblings/Spouses Aboard`, `Parents/Children Aboard` and `Fare` and transform them into a 2-dimensional `numpy.ndarray`, and save it as `X`.
- Pick the column `Survived` and transform it into a 1-dimensional `numpy.ndarray` and save it as `y`.

~~~
````{solution} ex1pandastitanic
:class: dropdown

Not yet done!
~~~

6 k-Nearest Neighbors algorithm (k-NN)

This algorithm is different from other algorithms covered in this course, that it doesn't really extract features from the data. However, since its idea is easy to understand, we use it as our first step towards machine learning world.

Similar to other algorithms, we will only cover the beginning part of the algorithm. All later upgrades of the algorithms are left for yourselves to learn.

References: {cite:p}`Har2012`.

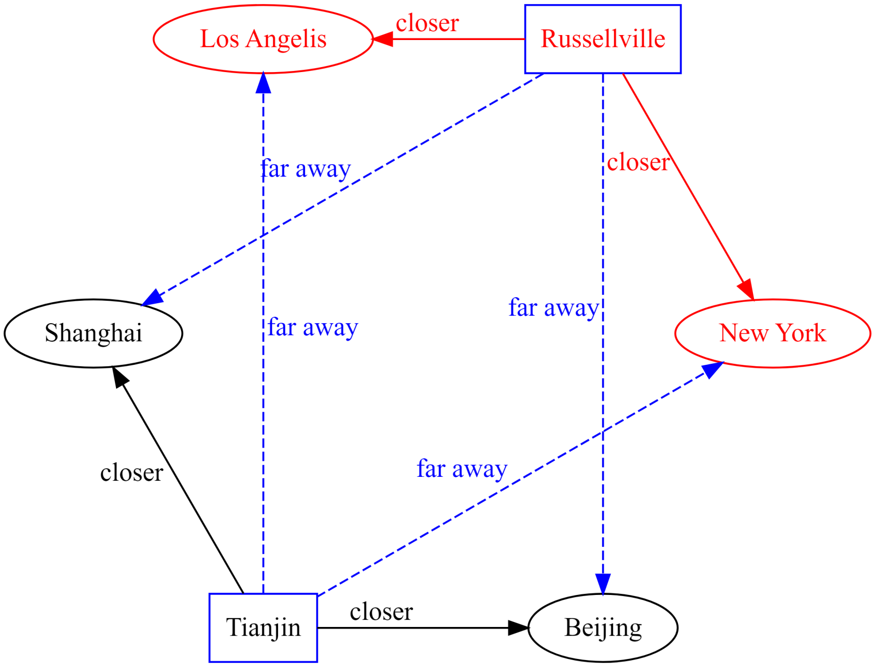
6.1 k-Nearest Neighbors Algorithm (k-NN)

6.1.1 Ideas

Assume that we have a set of labeled data $\{(X_i, y_i)\}$ where y_i denotes the label. Given a new data X, how do we determine the label of it?

k-NN algorithm starts from a very straightforward idea. We use the distances from the new data point X to the known data points to identify the label. If X is closer to y_i points, then we will label X as y_i.

Let us take cities and countries as an example. New York and Los Angeles are U.S cities, and Beijing and Shanghai are Chinese cities. Now we would like to consider Tianjin and Russellville. Do they belong to China or U.S? We calculate the distances from Tianjin (resp. Russellville) to all four known cities. Since Tianjin is closer to Beijing and Shanghai comparing to New York and Los Angeles, we classify Tianjin as a Chinese city. Similarly, since Russellville is closer to New York and Los Angeles comparing to Beijing and Shanghai, we classify it as a U.S. city.



This naive example explains the idea of k-NN. Here is a more detailed description of the algorithm.

6.1.2 The Algorithm

k-NN Classifier **Inputs** Given the training data set $\{(X_i, y_i)\}$ where $X_i = (x_i^1, x_i^2, \ldots, x_i^n)$ represents n features and y_i represents labels. Given a new data point $\tilde{X} = (\tilde{x}^1, \tilde{x}^2, \ldots, \tilde{x}^n)$.

Outputs Want to find the best label for $\tilde{X}$.

1. Compute the distance from $\tilde{X}$ to each X_i.
2. Sort all these distances from the nearest to the furthest.
3. Find the nearest k data points.
4. Determine the labels for each of these k nearest points, and compute the frenqucy of each labels.
5. The most frequent label is considered to be the label of $\tilde{X}$.

6.1.3 Details

- The distance between two data points are defined by the Euclidean distance:

$$dist\left((x_i^j)_{j=1}^n, (\tilde{x}^j)_{j=1}^n\right) = \sqrt{\sum_{j=1}^n (x_i^j - \tilde{x}^j)^2}.$$

- Using linear algebra notations:

$$dist(X_i, \tilde{X}) = \sqrt{(X_i - \tilde{X}) \cdot (X_i - \tilde{X})}.$$

- All the distances are stored in a 1-dim numpy array, and we will combine it together with another 1-dim array that store the labels of each point.

6.1.4 The codes

- `argsort`
- `get`
- `sorted`

```
def classify_kNN(inX, X, y, k):
    # create a new 2-d numpy array by copying inX for each row.
    Xmat = np.tile(np.array([inX]), (X.shape[0], 1))
    # compute the distance between each row of X and Xmat
    Dmat = np.sqrt(np.sum((Xmat - X)**2, axis=1))
    # sort by distance
    sortedlist = Dmat.argsort()
    # count the freq. of the first k items
    k = min(k, len(sortedlist))
    classCount = dict()
    for i in sortedlist[:k]:
        classCount[y[i]] = classCount.get(y[i], 0) + 1
    # find out the most freqent one
    sortedCount = sorted(classCount.items(), key=lambda x:x[1],
                         reverse=True)
    return sortedCount[0][0]
```

6.1.5 `sklearn` packages

You may also directly use the kNN function `KNeighborsClassifier` packaged in `sklearn.neighbors`. You may check the description of the function online from here.

There are many ways to modify the kNN algorithm. What we just mentioned is the simplest idea. It is correspondent to the argument `weights='uniform'`, `algorithm='brute` and metric='euclidean'. However due to the implementation details, the results we got from sklearn are still a little bit different from the results produced by our naive codes.

```
from sklearn.neighbors import KNeighborsClassifier
clf = KNeighborsClassifier(n_neighbors=10, weights='uniform', algorithm='brute',
                           metric='euclidean')
clf.fit(X_train, y_train)
y_pred = ckf.predict(X_test)
```

6.1.6 Normalization

Different features may have different scales. It might be unfair for those features that have small scales. Therefore usually it is better to rescale all the features to make them have similar scales. After examining all the data, we find the minimal value minVal and the range ranges for each column. The normalization formula is:

$$X_{norm} = \frac{X_{original} - minVal}{ranges}.$$

We could also convert the normalized number back to the original value by

$$X_{original} = X_{norm} \times ranges + minVal.$$

The sample codes are listed below.

```
def encodeNorm(X, parameters=None):
    # parameters contains minVals and ranges
    if parameters is None:
        minVals = np.min(X, axis=0)
        maxVals = np.max(X, axis=0)
        ranges = np.maximum(maxVals - minVals, np.ones(minVals.size))
        parameters = {'ranges': ranges, 'minVals': minVals}
    else:
        minVals = parameters['minVals']
        ranges = parameters['ranges']
    Nmat = np.tile(minVals, (X.shape[0], 1))
    Xnorm = (X - Nmat)/ranges
    return (Xnorm, parameters)


def decodeNorm(X, parameters):
    # parameters contains minVals and ranges
    ranges = parameters['ranges']
    minVals = parameters['minVals']
    Nmat = np.tile(minVals, (X.shape[0], 1))
    Xoriginal = X * ranges + Nmat
    return Xoriginal
```

6.2 k-NN Project 1: `iris` Classification

This data is from `sklearn.datasets`. This dataset consists of 3 different types of irises' petal / sepal length / width, stored in a 150×4 `numpy.ndarray`. We already explored the dataset briefly in the previous chapter. This time we will try to use the feature provided to predict the type of the irises. For the purpose of plotting, we will only use the first two features: `sepal length` and `sepal width`.

6.2.1 Explore the dataset

We first load the dataset.

```
from sklearn import datasets
iris = datasets.load_iris()
X = iris.data[:, :2]
y = iris.target
```

Then we would like to split the dataset into trainning data and test data. Here we are going to use `sklearn.model_selection.train_test_split` function. Besides the dataset, we should also provide the propotion of the test set comparing to the whole dataset. We will choose `test_size=0.1` here, which means that the size of the test set is 0.1 times the size of the whole dataset. `stratify=y` means that when split the dataset we want to split respects the distribution of labels in `y`.

The split will be randomly. You may set the argument `random_state` to be a certain number to control the random process. If you set a `random_state`, the result of the random process will stay the same. This is for reproducible output across multiple function calls.

After we get the training set, we should also normalize it. All our normalization should be based on the training set. When we want to use our model on some new data points, we will use the same normalization parameters to normalize the data points in interests right before we apply the model. Here since we mainly care about the test set, we could normalize the test set at this stage.

Note that in the following code, I import functions `encodeNorm` from `assests.codes.knn`. You need to modify this part based on your file structure. See here for more details.

```
from sklearn.model_selection import train_test_split
from assests.codes.knn import encodeNorm
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.1, random_state=1, s

X_train_norm, parameters = encodeNorm(X_train)
X_test_norm, _ = encodeNorm(X_test, parameters=parameters)
```

Before we start to play with k-NN, let us look at the data first. Since we only choose two features, it is able to plot these data points on a 2D plane, with different colors representing different classes.

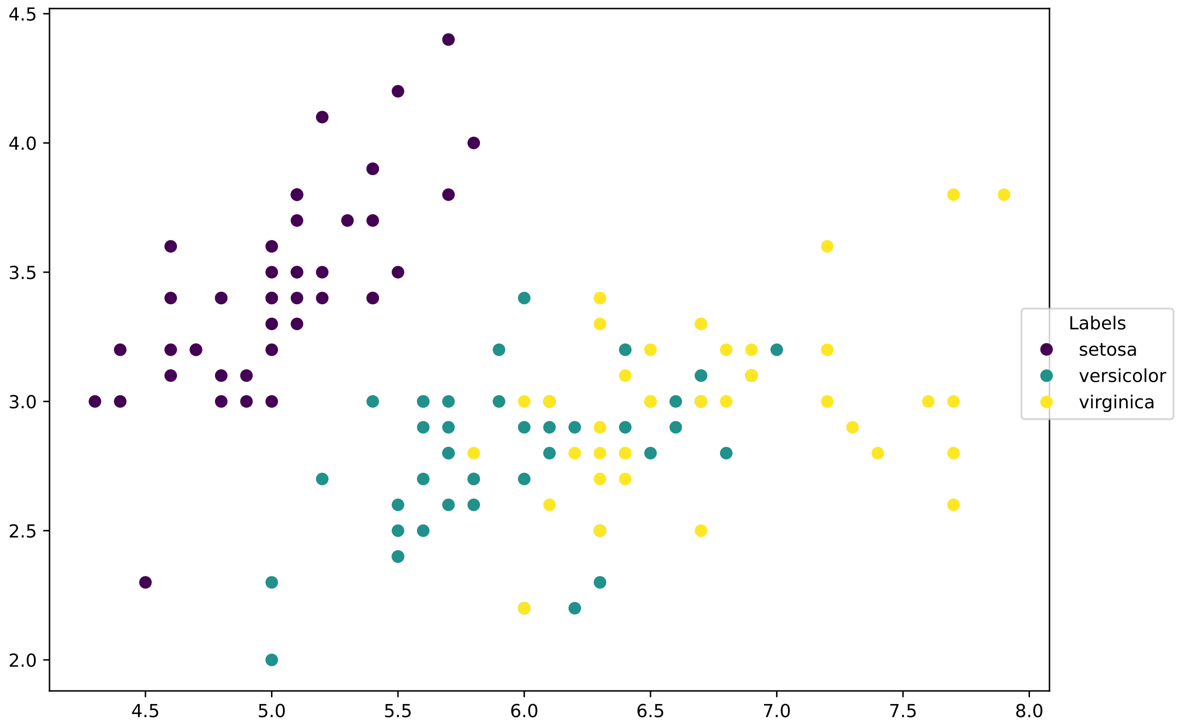
```
import matplotlib.pyplot as plt
import numpy as np

# Plot the scatter plot.
fig = plt.figure(figsize=(10,7))
ax = fig.add_subplot(111)
scatter = ax.scatter(X_train[:, 0], X_train[:, 1], c=y_train)

# Generate legends.
labels = ['setosa', 'versicolor', 'virginica']
fig.legend(handles=scatter.legend_elements()[0], labels=labels,
           loc="right", title="Labels")

# plt.show()
```

```
<matplotlib.legend.Legend at 0x2df4c55f790>
```



(section:applyourknn)= ### Apply our k-NN model

Now let us apply k-NN to this dataset. We first use our codes. Here I use `from assests.codes.knn` to import our functions since I put all our functions in `./assests/codes/knn.py`. Then the poential code is

```
y_pred = classify_kNN(X_test, X_train, y_train, k=10)
```

Note that the above code is actually wrong. The issue ist that our function `classify_kNN` can only classify one row of data. To classify many rows, we need to use a `for` loop.

```
from assests.codes.knn import classify_kNN

n_neighbors = 10
y_pred = list()
for row in X_test_norm:
    row_pred = classify_kNN(row, X_train_norm, y_train, k=n_neighbors)
    y_pred.append(row_pred)
y_pred = np.array(y_pred)
```

We could use list comprehension to simply the above codes.

```
from assests.codes.knn import classify_kNN

n_neighbors = 10
y_pred = np.array([classify_kNN(row, X_train_norm, y_train, k=n_neighbors)
                   for row in X_test_norm])
```

This `y_pred` is the result we got for the test set. We may compare it with the real answer `y_test`, and calcuate the accuracy.

```
acc = np.mean(y_pred == y_test)
print(acc)
```

```
0.7333333333333333
```

6.2.2 Apply k-NN model from `sklearn`

Now we would like to use `sklearn` to reproduce this result. Since our data is prepared, what we need to do is directly call the functions.

```
from sklearn.neighbors import KNeighborsClassifier
n_neighbors = 10
clf = KNeighborsClassifier(n_neighbors, weights="uniform", metric="euclidean",
                           algorithm='brute')
clf.fit(X_train_norm, y_train)
y_pred_sk = clf.predict(X_test_norm)

acc = np.mean(y_pred_sk == y_test)
print(acc)
```

```
0.7333333333333333
```

6.2.3 Using data pipeline

We may organize the above process in a neater way. After we get a data, the usual process is to apply several transforms to the data before we really get to the model part. Using terminolgies from `sklearn`, the former are called *transforms*, and the latter is called an *estimator*. In this example, we have exactly one tranform which is the normalization. The estimator here we use is the k-NN classifier.

sklearn provides a standard way to write these codes, which is called pipeline. We may chain the transforms and estimators in a sequence and let the data go through the pipeline. In this example, the pipeline contains two steps: 1. The normalization transform sklearn.preprocessing.MinMaxScaler. When we directly apply it the parameters ranges and minVals and will be recorded automatically, and we don't need to worry about it when we want to use the same parameters to normalize other data. 2. The k-NN classifier sklearn.neighbors.KNeighborsClassifier. This is the same one as we use previously.

The code is as follows. It is a straightforward code. Note that the () after the class in each step of steps is very important. The codes cannot run if you miss it.

After we setup the pipeline, we may use it as other estimators since it is an estimator. Here we may also use the accuracy function provided by sklearn to perform the computation. It is essentially the same as our acc computation.

```
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import MinMaxScaler
from sklearn.metrics import accuracy_score

steps = [('scaler', MinMaxScaler()),
         ('knn', KNeighborsClassifier(n_neighbors, weights="uniform",
                                      metric="euclidean", algorithm='brute'))]
pipe = Pipeline(steps=steps)
pipe.fit(X_train, y_train)
y_pipe = pipe.predict(X_test)
print(accuracy_score(y_pipe, y_test))
```

```
0.7333333333333333
```

6.2.4 Visualize the Decision boundary

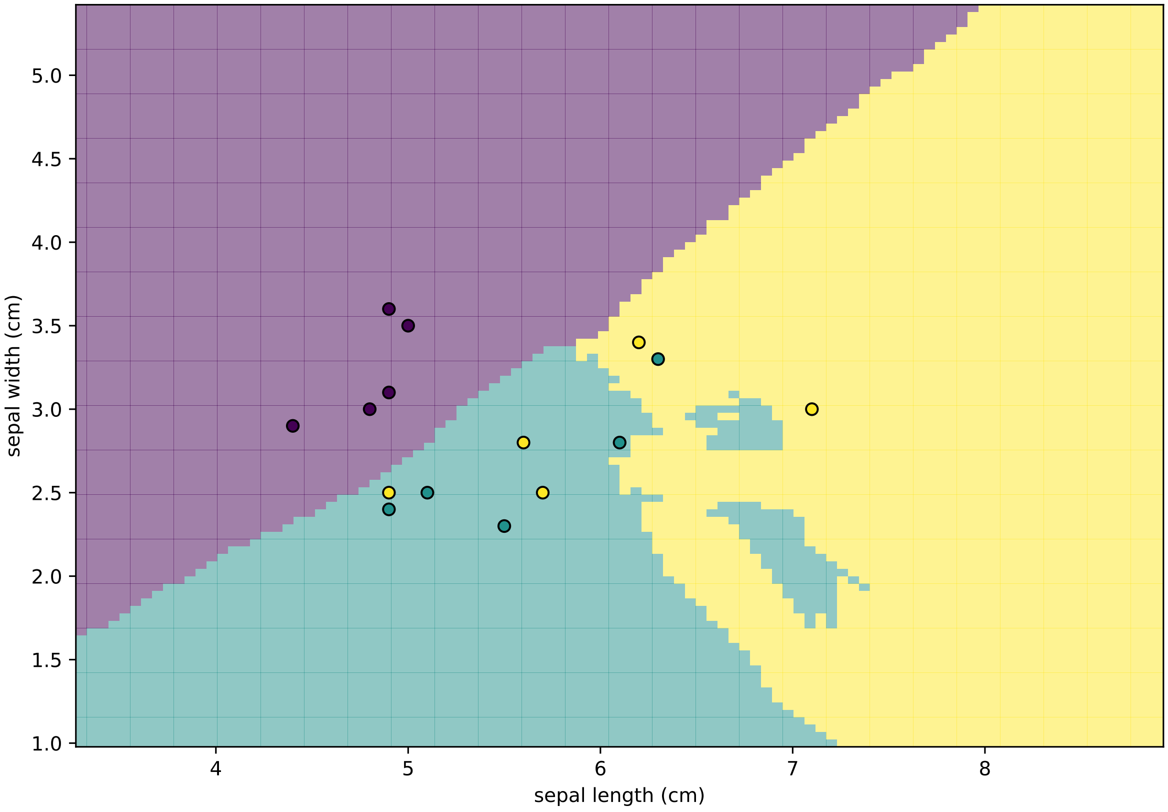
Using the classifier we get above, we are able to classify every points on the plane. This enables us to draw the following plot, which is called the Decision boundary. It helps us to visualize the relations between features and the classes.

We use DecisionBoundaryDisplay from sklearn.inspection to plot the decision boundary. The function requires us to have a fitted classifier. We may use the classifier pipe we got above. Note that this classifier should have some build-in structures that our classify_kNN function doesn't have. We may rewrite our codes to make it work, but this goes out of the scope of this section. This is supposed to be Python programming exercise. We will talk about it in the future if we have enough time.

We first plot the dicision boundary using `DecisionBoundaryDisplay.from_estimator`. Then we plot the points from `X_test`. From the plot it is very clear which points are misclassified.

```
from sklearn.inspection import DecisionBoundaryDisplay

disp = DecisionBoundaryDisplay.from_estimator(
            pipe,
            X_train,
            response_method="predict",
            plot_method="pcolormesh",
            xlabel=iris.feature_names[0],
            ylabel=iris.feature_names[1],
            alpha=0.5)
disp.ax_.scatter(X_test[:, 0], X_test[:, 1], c=y_test, edgecolor="k")
disp.figure_.set_size_inches((10,7))
```



(section-cross-validation)= ### k-Fold Cross-Validation

Previously we perform a random split and test our model in this case. What would happen if we fit our model on another split? We might get a different accuracy score. So in order to evaluate the performance of our model, it is natual to consider several different split and compute the accuracy socre for each case, and combine all these socres together to generate an index to indicate whehter our model is good or bad. This naive idea is called *k-Fold Cross-Validation*.

The algorithm is described as follows. We first randomly split the dataset into `k` groups. We use one of them as the test set, and the rest together forming the training set, and use this setting to get an accuracy score. We did this for each group to be chosen as the test set. Then the final score is the mean.

`sklearn` provides a function `sklearn.model_selection.cross_val_score` to perform the above computation. The usage is straightforward, as follows.

```python
from sklearn.model_selection import cross_val_score
cv_scores = cross_val_score(pipe, X, y, cv=5)
print(cv_scores)
print(np.mean(cv_scores))
```

```
[0.66666667 0.8        0.63333333 0.8        0.7       ]
0.7200000000000001
```

6.2.5 Choosing a `k` value

In the previous example we choose `k` to be `10` as an example. To choose a `k` value we usually run some test by trying different `k` and choose the one with the best performance. In this case, best performance means the highest cross-validation score.

`sklearn.model_selection.GridSearchCV` provides a way to do this directly. We only need to setup the esitimator, the metric (which is the cross-validation score in this case), and the hyperparameters to be searched through, and `GridSearchCV` will run the search automatically.

We let `k` go from `1` to `100`. The code is as follows.

Note that `parameters` is where we set the search space. It is a dictionary. The key is the name of the estimator plus double `_` and then plus the name of the parameter.

```python
from sklearn.model_selection import GridSearchCV
n_list = list(range(1, 101))
parameters = dict(knn__n_neighbors=n_list)
clf = GridSearchCV(pipe, parameters)
```

```
clf.fit(X, y)
print(clf.best_estimator_.get_params()["knn__n_neighbors"])
```

```
35
```

After we fit the data, the `best_estimator_.get_params()` can be printed. It tells us that it is best to use `31` neibhours for our model. We can directly use the best estimator by calling `clf.best_estimator_`.

```
cv_scores = cross_val_score(clf.best_estimator_, X, y, cv=5)
print(np.mean(cv_scores))
```

```
0.82
```

The cross-validation score using `k=31` is calculated. This serves as a benchmark score and we may come back to dataset using other methods and compare the scores.

6.3 k-NN Project 2: Dating Classification

The data can be downloaded from {Download}`here<./assests/datasets/datingTestSet2.txt>`.

6.3.1 Background

Helen dated several people and rated them using a three-point scale: 3 is best and 1 is worst. She also collected data from all her dates and recorded them in the file attached. These data contains 3 features:

- Number of frequent flyer miles earned per year
- Percentage of time spent playing video games
- Liters of ice cream consumed per week

We would like to predict her ratings of new dates when we are given the three features.

The data contains four columns, while the first column refers to `Mileage`, the second `Gamingtime`, the third `Icecream` and the fourth `Rating`.

6.3.2 Look at Data

We first load the data and store it into a DataFrame.

```python
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

df = pd.read_csv('./assests/datasets/datingTestSet2.txt', sep='\t', header=None)
df.head()
```

```
C:\Users\Xinli\AppData\Roaming\Python\Python310\site-packages\IPython\core\formatters.py:343
  return method()
```

	0	1	2	3
0	40920	8.326976	0.953952	3
1	14488	7.153469	1.673904	2
2	26052	1.441871	0.805124	1
3	75136	13.147394	0.428964	1
4	38344	1.669788	0.134296	1

To make it easier to read, we would like to change the name of the columns.

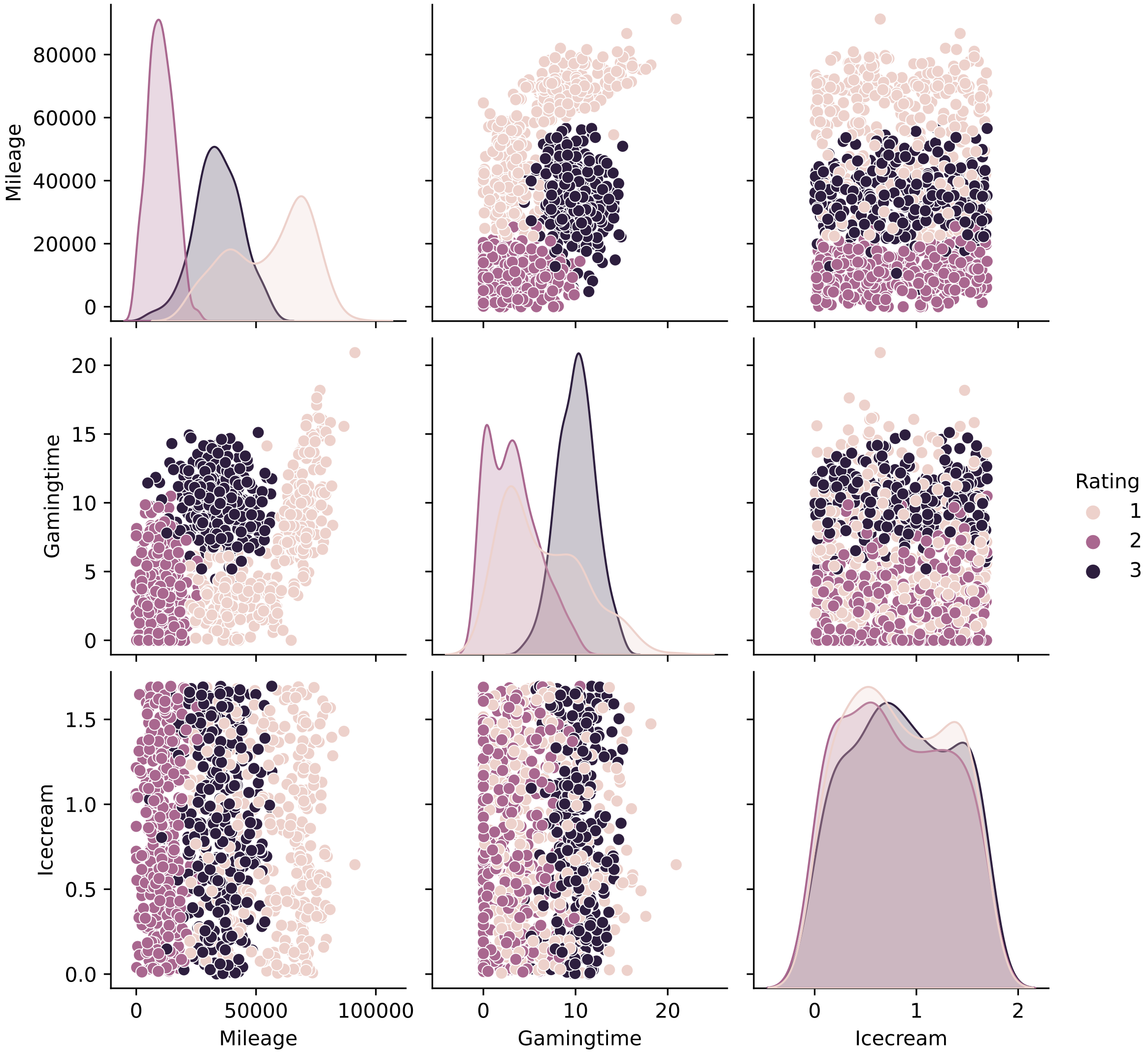
```python
df = df.rename(columns={0: "Mileage", 1: "Gamingtime", 2: 'Icecream', 3: 'Rating'})
df.head()
```

```
C:\Users\Xinli\AppData\Roaming\Python\Python310\site-packages\IPython\core\formatters.py:343
  return method()
```

	Mileage	Gamingtime	Icecream	Rating
0	40920	8.326976	0.953952	3
1	14488	7.153469	1.673904	2
2	26052	1.441871	0.805124	1
3	75136	13.147394	0.428964	1
4	38344	1.669788	0.134296	1

Since now we have more than 2 features, it is not suitable to directly draw scatter plots. We use `seaborn.pairplot` to look at the pairplot. From the below plots, before we apply any tricks, it seems that `Milegae` and `Gamingtime` are better than `Icecream` to classify the data points.

```python
import seaborn as sns
sns.pairplot(data=df, hue='Rating')
```



6.3.3 Applying kNN

Similar to the previous example, we will apply both methods for comparisons.

```
from sklearn.model_selection import train_test_split
from assests.codes.knn import encodeNorm
X = np.array(df[['Mileage', 'Gamingtime', 'Icecream']])
y = np.array(df['Rating'])

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.1, random_state=40,
X_train_norm, parameters = encodeNorm(X_train)
X_test_norm, _ = encodeNorm(X_test, parameters=parameters)
```

```
# Using our codes.
from assests.codes.knn import classify_kNN

n_neighbors = 10
y_pred = np.array([classify_kNN(row, X_train_norm, y_train, k=n_neighbors)
                   for row in X_test_norm])

acc = np.mean(y_pred == y_test)
print(acc)
```

```
0.93
```

```
# Using sklearn.
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import MinMaxScaler
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score

steps = [('scaler', MinMaxScaler()),
         ('knn', KNeighborsClassifier(n_neighbors, weights="uniform",
                                      metric="euclidean", algorithm='brute'))]
pipe = Pipeline(steps=steps)
pipe.fit(X_train, y_train)
y_pipe = pipe.predict(X_test)
print(accuracy_score(y_pipe, y_test))
```

```
0.93
```

6.3.4 Choosing `k` Value

Similar to the previous section, we can run tests on `k` value to choose one to be used in our model using `GridSearchCV`.

```
from sklearn.model_selection import GridSearchCV, cross_val_score
n_list = list(range(1, 101))
parameters = dict(knn__n_neighbors=n_list)
```

```
clf = GridSearchCV(pipe, parameters)
clf.fit(X, y)
print(clf.best_estimator_.get_params()["knn__n_neighbors"])
```

```
4
```

From this result, in this case the best `k` is `4`. The corresponding cross-validation score is computed below.

```
cv_scores = cross_val_score(clf.best_estimator_, X, y, cv=5)
print(np.mean(cv_scores))
```

```
0.952
```

6.4 Exercises and Projects

““vcohkfiwff Handwritten example :label: ex2handwritten Consider the 1-dimensional data set shown below.

“‘iukpyhvswiya Dataset :header-rows: 1

- - x
  - 1.5
  - 2.5
  - 3.5
  - 4.5
  - 5.0
  - 5.5
  - 5.75
  - 6.5
  - 7.5
  - 10.5

- - y
  - $+$
  - $+$
  - $-$
  - $-$
  - $-$
  - $+$

− +
− −
− +
− +

```
Please use the data to compute the class of $x=5.5$ according to $k=1$, $3$, $6$ and $9
```

```
zpsmgalyvc ex2handwritten :class: dropdown Not yet done!
```

```
:label: ex2titanic
Please download the titanic dataset from {Download}`here<./assests/datasets/titanic.csv>`.

Please analyze the dataset and build a k-NN model to predict whether someone is survived o
```

““zpsmgalyvc ex2titanic :class: dropdown

Not yet done! ““