

ML23

Lecture notes for Fall 2023 at Arkansas Tech University

Xinli Xiao

2023-06-20

Table of contents

Preface	3
References	4
1 Introduction	5
2 k-Nearest Neighbors algorithm (k-NN)	10
2.1 k-Nearest Neighbors Algorithm (k-NN)	10
2.1.1 Ideas	10
2.1.2 The Algorithm	11
2.1.3 Details	11
2.1.4 The codes	12
2.1.5 <code>sklearn</code> packages	12
2.1.6 Normalization	13
2.2 k-NN Project 1: <code>iris</code> Classification	14
2.2.1 Explore the dataset	14
2.2.2 Apply k-NN model from <code>sklearn</code>	17
2.2.3 Using data pipeline	17
2.2.4 Visualize the Decision boundary	18
2.2.5 Choosing a <code>k</code> value	20
2.3 k-NN Project 2: Dating Classification	21
2.3.1 Background	21
2.3.2 Look at Data	21
2.3.3 Applying kNN	23
2.3.4 Choosing <code>k</code> Value	24
2.4 Exercises and Projects	25

Preface

This is the lecture notes for STAT 4803/5803 Machine Learning Fall 2023 at ATU. If you have any comments/suggetions/concers about the notes please contact us at xxiao@atu.edu.

References

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.

```
import sys

print(sys.executable)

!mamba list

import pandas as pd
```

```
C:\Users\Xinli\mambaforge\envs\base23\python.exe
# packages in environment at C:\Users\Xinli\mambaforge:
#
# Name                                Version                                Build      Channel
anyio                                  3.5.0                                  py310haa95532_0
argon2-cffi                           21.3.0                                pyhd3eb1b0_0
argon2-cffi-bindings                  21.2.0                                py310h2bbff1b_0
asttokens                             2.0.5                                  pyhd3eb1b0_0
attrs                                  22.1.0                                py310haa95532_0
babel                                  2.11.0                                py310haa95532_0
backcall                               0.2.0                                  pyhd3eb1b0_0
beautifulsoup4                         4.12.2                                py310haa95532_0
bleach                                 4.1.0                                  pyhd3eb1b0_0
boltons                                23.0.0                                py310haa95532_0
brotlipy                               0.7.0                                py310h8d17308_1005    conda-forge
bzip2                                  1.0.8                                  h8ffe710_4            conda-forge
ca-certificates                        2023.01.10                            haa95532_0
certifi                                2023.5.7                               py310haa95532_0
```

cffi	1.15.1	py310h628cb3f_3	conda-forge
charset-normalizer	3.1.0	pyhd8ed1ab_0	conda-forge
colorama	0.4.6	pyhd8ed1ab_0	conda-forge
comm	0.1.2	py310haa95532_0	
conda	23.3.1	py310haa95532_0	
conda-package-handling	2.0.2	pyh38be061_0	conda-forge
conda-package-streaming	0.7.0	pyhd8ed1ab_1	conda-forge
cryptography	40.0.1	py310h6e82f81_0	conda-forge
debugpy	1.5.1	py310hd77b12b_0	
decorator	5.1.1	pyhd3eb1b0_0	
defusedxml	0.7.1	pyhd3eb1b0_0	
entrypoints	0.4	py310haa95532_0	
executing	0.8.3	pyhd3eb1b0_0	
fmt	9.1.0	h181d51b_0	conda-forge
gettext	0.21.1	h5728263_0	conda-forge
glib	2.76.3	h12be248_0	conda-forge
glib-tools	2.76.3	h12be248_0	conda-forge
gst-plugins-base	1.22.3	h001b923_1	conda-forge
gstreamer	1.22.3	h6b5321d_1	conda-forge
icu	72.1	h63175ca_0	conda-forge
idna	3.4	pyhd8ed1ab_0	conda-forge
ipykernel	6.19.2	py310h9909e9c_0	
ipython	8.12.0	py310haa95532_0	
ipython_genutils	0.2.0	pyhd3eb1b0_1	
ipywidgets	8.0.4	py310haa95532_0	
jedi	0.18.1	py310haa95532_1	
jinja2	3.1.2	py310haa95532_0	
json5	0.9.6	pyhd3eb1b0_0	
jsonpatch	1.32	pyhd3eb1b0_0	
jsonpointer	2.1	pyhd3eb1b0_0	
jsonschema	4.17.3	py310haa95532_0	
jupyter	1.0.0	py310haa95532_8	
jupyter_client	8.1.0	py310haa95532_0	
jupyter_console	6.6.3	py310haa95532_0	
jupyter_core	5.3.0	py310haa95532_0	
jupyter_server	1.23.4	py310haa95532_0	
jupyterlab	3.5.3	py310haa95532_0	
jupyterlab_pygments	0.1.2	py_0	
jupyterlab_server	2.22.0	py310haa95532_0	
jupyterlab_widgets	3.0.5	py310haa95532_0	
krb5	1.20.1	heb0366b_0	conda-forge
libarchive	3.6.2	h27c7867_0	conda-forge
libclang	16.0.1	default_h8b4101f_0	conda-forge

libclang13	16.0.1	default_h45d3cf4_0	conda-forge
libcurl	7.88.1	h68f0423_1	conda-forge
libffi	3.4.2	h8ffe710_5	conda-forge
libglib	2.76.3	he8f3873_0	conda-forge
libiconv	1.17	h8ffe710_0	conda-forge
libjpeg-turbo	2.1.5.1	hcfcfb64_0	conda-forge
libmamba	1.4.2	h8a7d157_0	conda-forge
libmambapy	1.4.2	py310h3fe4c2e_0	conda-forge
libogg	1.3.5	h2bbff1b_1	
libpng	1.6.39	h8cc25b3_0	
libsodium	1.0.18	h62dcd97_0	
libsolv	0.7.23	h12be248_0	conda-forge
libsqlite	3.40.0	hcfcfb64_0	conda-forge
libssh2	1.10.0	h9a1e1f7_3	conda-forge
libvorbis	1.3.7	he774522_0	
libxml2	2.10.3	hc3477c8_6	conda-forge
libxslt	1.1.37	h2bbff1b_0	
libzlib	1.2.13	hcfcfb64_4	conda-forge
lxml	4.9.2	py310h2bbff1b_0	
lz4-c	1.9.4	hcfcfb64_0	conda-forge
lzo	2.10	he774522_1000	conda-forge
mamba	1.4.2	py310hd9d798f_0	conda-forge
markupsafe	2.1.1	py310h2bbff1b_0	
matplotlib-inline	0.1.6	py310haa95532_0	
menuinst	1.4.19	py310h5588dad_1	conda-forge
miniforge_console_shortcut	1.0	h57928b3_0	conda-forge
mistune	0.8.4	py310h2bbff1b_1000	
nb_conda_kernels	2.3.1	py310haa95532_0	
nbclassic	0.5.5	py310haa95532_0	
nbclient	0.5.13	py310haa95532_0	
nbconvert	6.5.4	py310haa95532_0	
nbformat	5.7.0	py310haa95532_0	
nest-asyncio	1.5.6	py310haa95532_0	
notebook	6.5.4	py310haa95532_0	
notebook-shim	0.2.2	py310haa95532_0	
openssl	3.1.0	hcfcfb64_3	conda-forge
packaging	23.0	py310haa95532_0	
pandocfilters	1.5.0	pyhd3eb1b0_0	
parso	0.8.3	pyhd3eb1b0_0	
pcre2	10.40	h17e33f8_0	conda-forge
pickleshare	0.7.5	pyhd3eb1b0_1003	
pip	23.0.1	pyhd8ed1ab_0	conda-forge
platformdirs	2.5.2	py310haa95532_0	

pluggy	1.0.0	pyhd8ed1ab_5	conda-forge
ply	3.11	py310haa95532_0	
prometheus_client	0.14.1	py310haa95532_0	
prompt-toolkit	3.0.36	py310haa95532_0	
prompt_toolkit	3.0.36	hd3eb1b0_0	
psutil	5.9.0	py310h2bbff1b_0	
pure_eval	0.2.2	pyhd3eb1b0_0	
pybind11-abi	4	hd8ed1ab_3	conda-forge
pycosat	0.6.4	py310h8d17308_1	conda-forge
pycparser	2.21	pyhd8ed1ab_0	conda-forge
pygments	2.15.1	py310haa95532_1	
pyopenssl	23.1.1	pyhd8ed1ab_0	conda-forge
pyqt	5.15.7	py310h1fd54f2_3	conda-forge
pyqt5-sip	12.11.0	py310h00ffb61_3	conda-forge
pypersistent	0.18.0	py310h2bbff1b_0	
pysocks	1.7.1	pyh0701188_6	conda-forge
python	3.10.10	h4de0772_0_cpython	conda-forge
python-dateutil	2.8.2	pyhd3eb1b0_0	
python-fastjsonschema	2.16.2	py310haa95532_0	
python_abi	3.10	3_cp310	conda-forge
pytz	2022.7	py310haa95532_0	
pywin32	305	py310h2bbff1b_0	
pywinpty	2.0.10	py310h5da7b33_0	
pyzmq	25.0.2	py310hd77b12b_0	
qt-main	5.15.8	h7f2b912_9	conda-forge
qtconsole	5.4.2	py310haa95532_0	
qtpy	2.2.0	py310haa95532_0	
reproc	14.2.4	hcfcfb64_0	conda-forge
reproc-cpp	14.2.4	h63175ca_0	conda-forge
requests	2.28.2	pyhd8ed1ab_1	conda-forge
ruamel.yaml	0.17.21	py310h8d17308_3	conda-forge
ruamel.yaml.clib	0.2.7	py310h8d17308_1	conda-forge
send2trash	1.8.0	pyhd3eb1b0_1	
setuptools	65.6.3	pyhd8ed1ab_0	conda-forge
sip	6.7.9	py310h00ffb61_0	conda-forge
six	1.16.0	pyhd3eb1b0_1	
sniffio	1.2.0	py310haa95532_1	
soupsieve	2.4	py310haa95532_0	
stack_data	0.2.0	pyhd3eb1b0_0	
terminado	0.17.1	py310haa95532_0	
tinycss2	1.2.1	py310haa95532_0	
tk	8.6.12	h8ffe710_0	conda-forge
toml	0.10.2	pyhd3eb1b0_0	

tomli	2.0.1	py310haa95532_0	
toolz	0.12.0	pyhd8ed1ab_0	conda-forge
tornado	6.2	py310h2bbff1b_0	
tqdm	4.65.0	pyhd8ed1ab_1	conda-forge
traitlets	5.7.1	py310haa95532_0	
typing-extensions	4.5.0	py310haa95532_0	
typing_extensions	4.5.0	py310haa95532_0	
tzdata	2023c	h71feb2d_0	conda-forge
ucrt	10.0.22621.0	h57928b3_0	conda-forge
urllib3	1.26.15	pyhd8ed1ab_0	conda-forge
vc	14.3	hb6edc58_10	conda-forge
vc14_runtime	14.34.31931	h5081d32_13	conda-forge
vs2015_runtime	14.34.31931	h4c5c07a_10	conda-forge
wcwidth	0.2.5	pyhd3eb1b0_0	
webencodings	0.5.1	py310haa95532_1	
websocket-client	0.58.0	py310haa95532_4	
wheel	0.40.0	pyhd8ed1ab_0	conda-forge
widgetsnbextension	4.0.5	py310haa95532_0	
win_inet_pton	1.1.0	pyhd8ed1ab_6	conda-forge
winpty	0.4.3	4	
xz	5.2.6	h8d14728_0	conda-forge
yaml-cpp	0.7.0	h63175ca_2	conda-forge
zeromq	4.3.4	hd77b12b_0	
zlib	1.2.13	hcfcfb64_4	conda-forge
zstandard	0.19.0	py310h0009e47_1	conda-forge
zstd	1.5.2	h12be248_6	conda-forge

2 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.

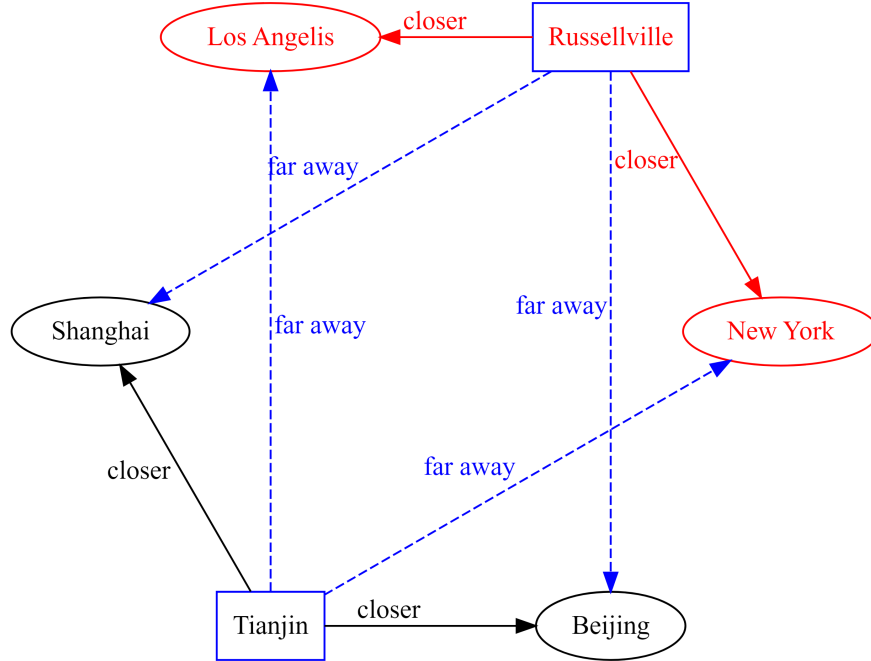
2.1 k-Nearest Neighbors Algorithm (k-NN)

2.1.1 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.

2.1.2 The Algorithm

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

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 frequency of each labels.
5. The most frequent label is considered to be the label of \tilde{X} .

2.1.3 Details

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

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

- Using linear algebra notations:

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

- 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.

2.1.4 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 frequent one
    sortedCount = sorted(classCount.items(), key=lambda x:x[1],
                          reverse=True)
    return sortedCount[0][0]
```

2.1.5 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 = clf.predict(X_test)
```

2.1.6 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

```

2.2 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`.

2.2.1 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 training 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 proportion 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.

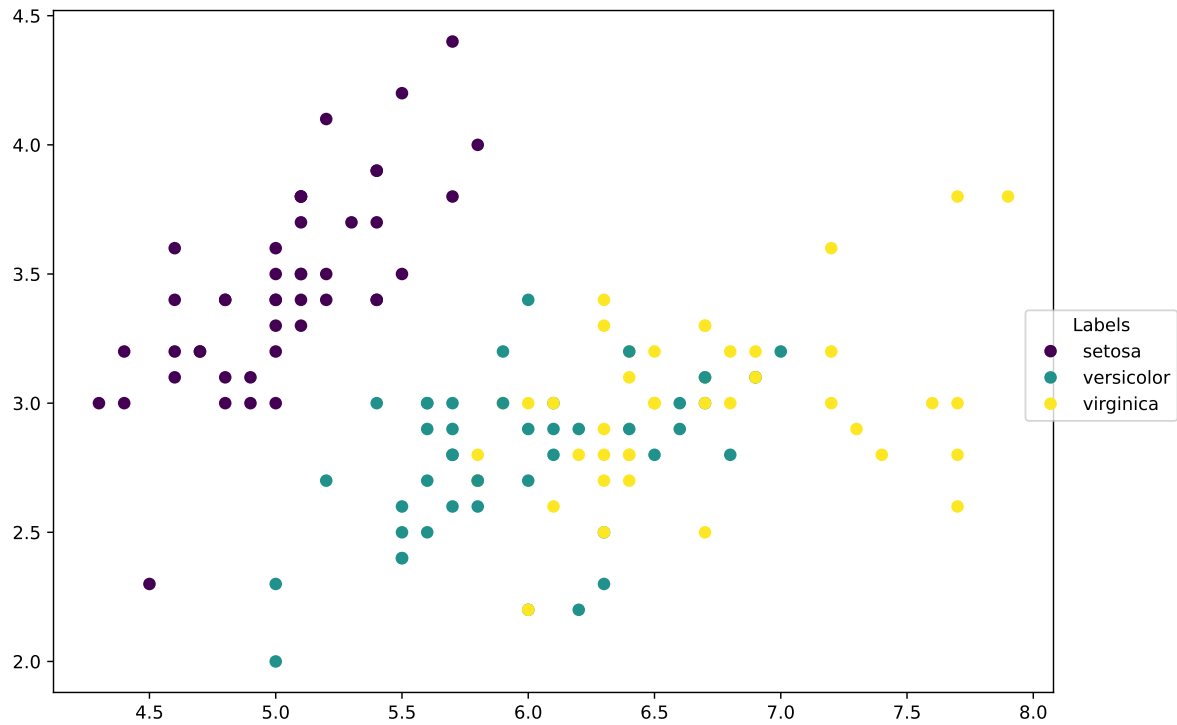
```
import matplotlib.pyplot as plt
import numpy as np

# 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 calculate the accuracy.

```

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

```

0.7333333333333333

2.2.2 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

2.2.3 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 terminologies from `sklearn`, the former are called *transforms*, and the latter is called an *estimator*. In this example, we have exactly one transform 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

2.2.4 Visualize the Decision boundary

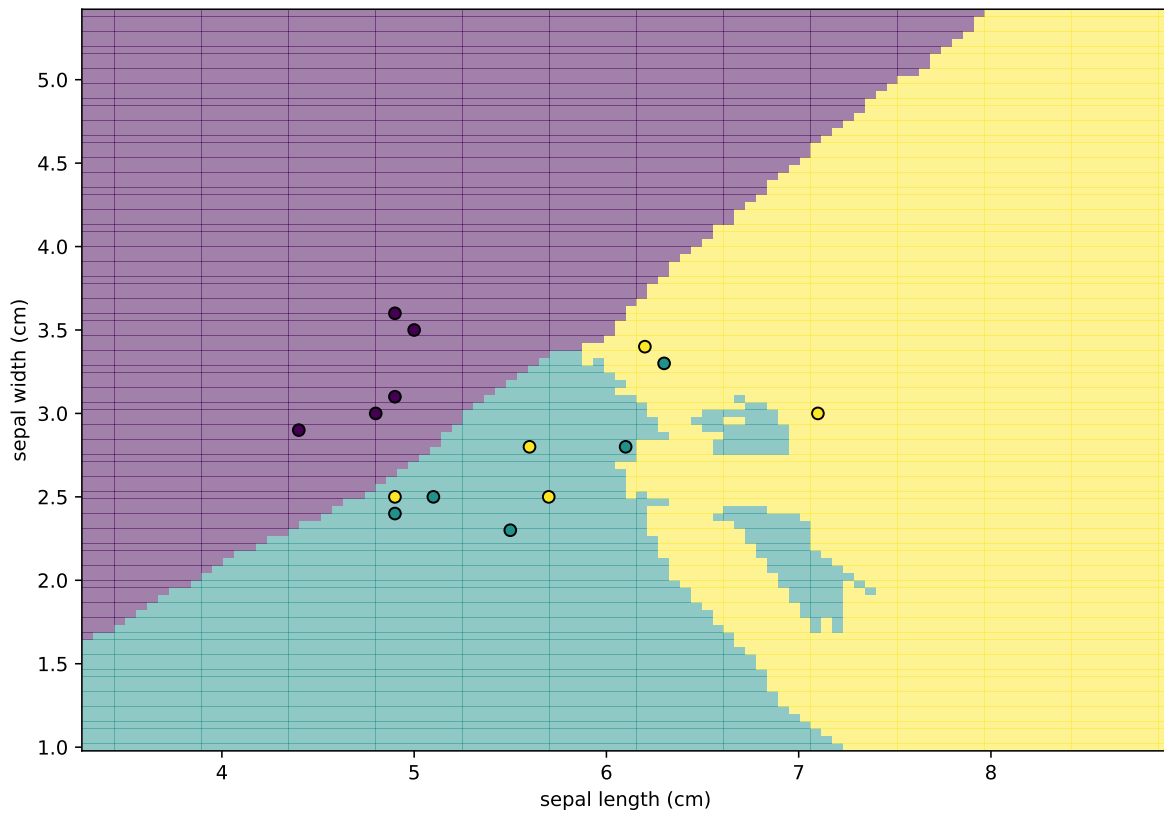
Using the classifier we get above, we are able to classify every points on the plane. This enables us to draw the following plot, which is called the Decision boundary. It helps us to visualize the relations between features and the classes.

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 decision 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 natural to consider several different splits and compute the accuracy score for each case, and combine all these scores together to generate an index to indicate whether 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.

```
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          ]
0.72000000000000001
```

2.2.5 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 estimator, 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.

```
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 neighbours 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.

2.3 k-NN Project 2: Dating Classification

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

2.3.1 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`.

2.3.2 Look at Data

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

```
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.

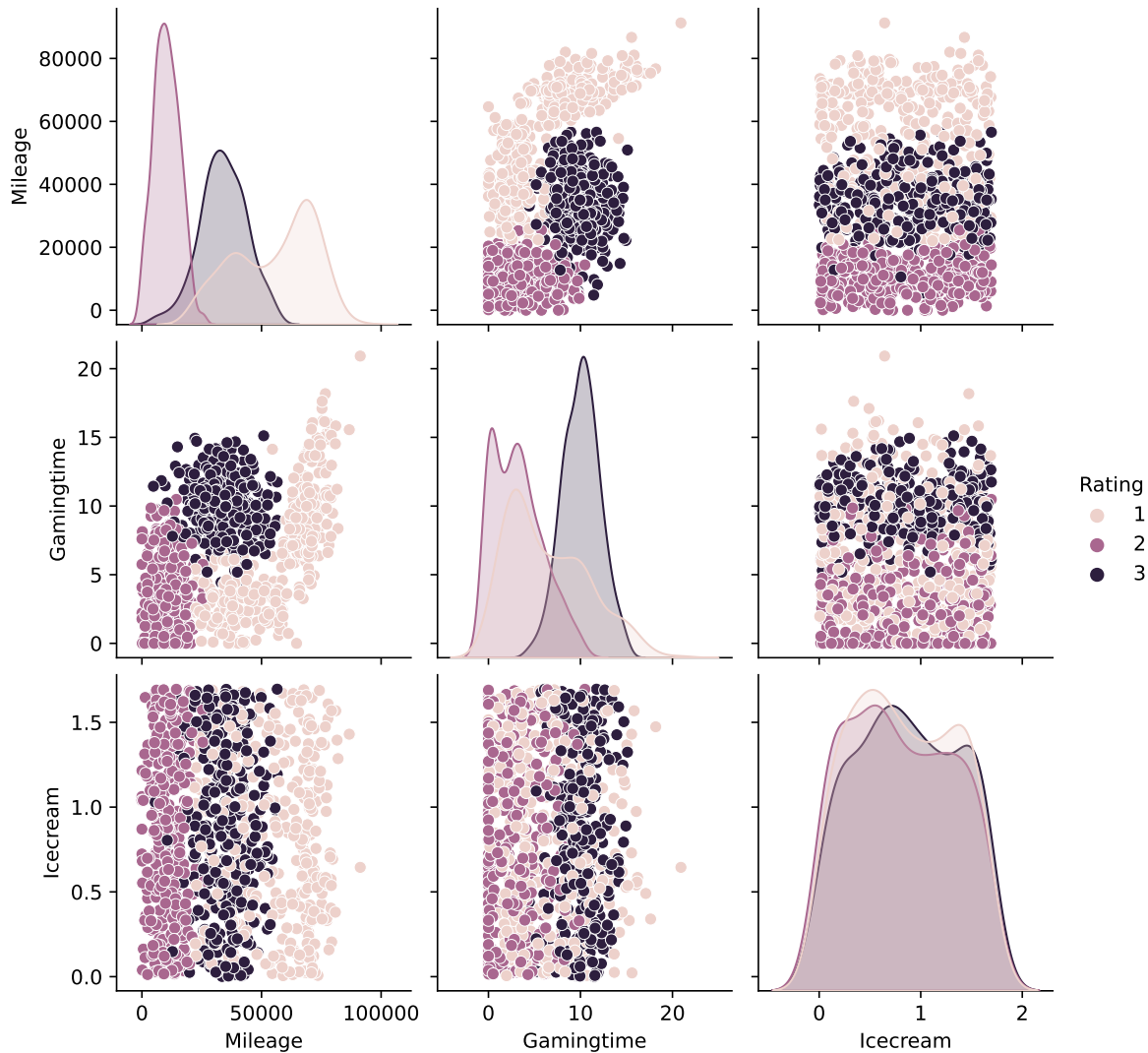
```
df = df.rename(columns={0: "Mileage", 1: "Gamingtime", 2: 'Icecream', 3: 'Rating'})
df.head()
```

```
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.

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



2.3.3 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

2.3.4 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

2.4 Exercises and Projects

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

““icugoaodoxdojo 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

oujhtxzd 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

“oujhtxzd ex2titanic :class: dropdown

Not yet done! ““