Fourth Industrial Summer School

Module 4: ML

Supervised Learning: Classification

Friday, July 5, 2024

Outlines

✓Classification Intro.

✓ Non-parametrized methods

✓Evaluation

✓Parameterized methods

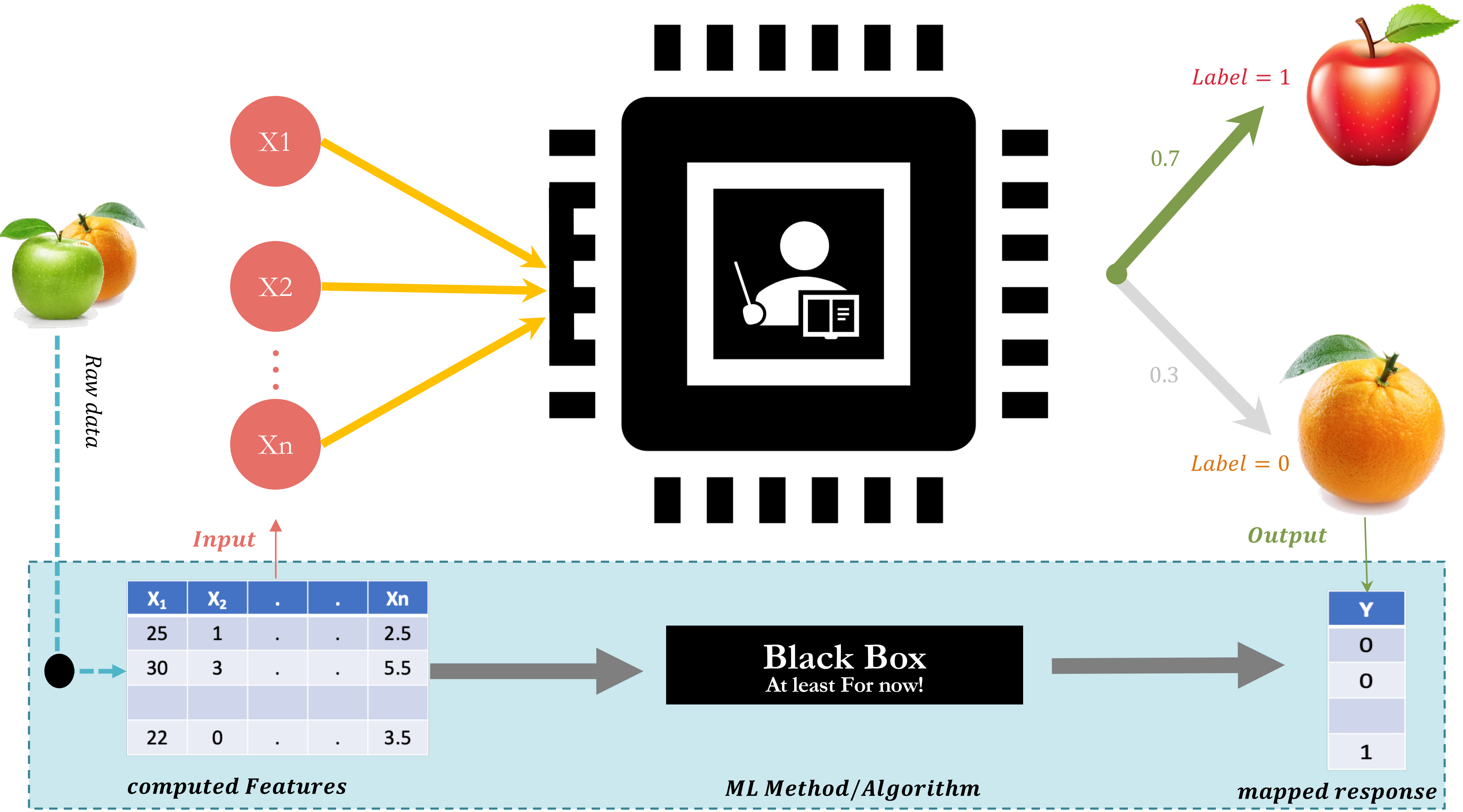
Classification

- Classification refers to a predictive modeling where a machine can distinguish samples by categories.
- The learning is to find a suitable mapping from features space (independent variables, predictors, etc.) to the categorical space (dependent variable, response, or label space).
- That means, it is <u>supervised</u> learning!
- Usually, the setting has two or more categories (i.e., classes).

Classification

- Classification problems can be categorized into
  - Binomial problems (data are for **two categories** only)
    - Emails identification: Spam / Not - Spam
    - Healthy / Illness
    - Etc.
  - Multinomial problems (data are for **multiple categories >2**)
    - Animal recognition: Cats, Dogs, Rabbets, etc.
    - Personal identification: Employees attendance
    - Etc.

General Structure for Modeling

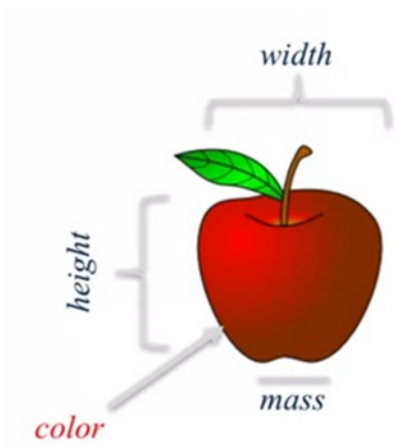


X_1	X_2	.	.	Xn
25	1	.	.	2.5
30	3	.	.	5.5
22	0	.	.	3.5

Y
0
0
1

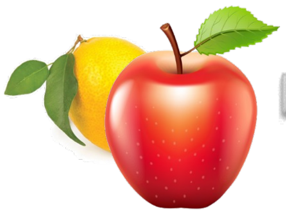
From Objects to Modeling: Fruits dataset

1. Objects characteristics (Features)



Labeling part | Feature Rep.

Fruit label	Fruit name	Fruit subtype	Mass	Width	Height	Color score
1	Apple	Cripps pink	162	7.5	7.1	0.83



Extract features

2. Learning

Modeling

Ready ML

3.

Predicted class(?)

Basic Requirements:

- Camera to capture images
- Image processing to remove noise and adjust lighting
- A robust method to get measurements (features)

The fruits dataset was created by **Dr. Iain Murray** from University of Edinburgh.

ML Modeling

- To prepare for modeling, **training**, or creating an ML model, we need to use the available data.
- As in fruit dataset, the data matrix contains **4 feature** and **1 label** columns.

```python
import pandas as pd
data = pd.read_csv('fruit_data_with_colors.csv')
data
```

Label

Features

	fruit_label	fruit_name	fruit_subtype	mass	width	height	color_score
0	1	apple	granny_smith	192	8.4	7.3	0.55
1	1	apple	granny_smith	180	8.0	6.8	0.59
2	1	apple	granny_smith	176	7.4	7.2	0.60
3	2	mandarin	mandarin	86	6.2	4.7	0.80
4	2	mandarin	mandarin	84	6.0	4.6	0.79
5	2	mandarin	mandarin	80	5.8	4.3	0.77
6	2	mandarin	mandarin	80	5.9	4.3	0.81
7	2	mandarin	mandarin	76	5.8	4.0	0.81
8	1	apple	braeburn	178	7.1	7.8	0.92
9	1	apple	braeburn	172	7.4	7.0	0.89
10	1	apple	braeburn	166	6.9	7.3	0.93
11	1	apple	braeburn	172	7.1	7.6	0.92
12	1	apple	braeburn	154	7.0	7.1	0.88
13	1	apple	golden_delicious	164	7.3	7.7	0.70
14	1	apple	golden_delicious	152	7.6	7.3	0.69
15	1	apple	golden_delicious	156	7.7	7.1	0.69

59 Samples

ML modeling

- The two columns **fruit_name**, and **fruit_subtype** can be removed before modeling.
- It is expected to have a clean data for modeling.

```
data2 = data.drop(['fruit_name', 'fruit_subtype'], axis=1)
```

```
data2
```

	fruit_label	mass	width	height	color_score
0	1	192	8.4	7.3	0.55
1	1	180	8.0	6.8	0.59
2	1	176	7.4	7.2	0.60
3	2	86	6.2	4.7	0.80
4	2	84	6.0	4.6	0.79
5	2	80	5.8	4.3	0.77
6	2	80	5.9	4.3	0.81
7	2	76	5.8	4.0	0.81
8	1	178	7.1	7.8	0.92
9	1	172	7.4	7.0	0.89
10	1	166	6.9	7.3	0.93
11	1	172	7.1	7.6	0.92
12	1	154	7.0	7.1	0.88
13	1	164	7.3	7.7	0.70
14	1	152	7.6	7.3	0.69
15	1	156	7.7	7.1	0.69

But when should we preprocess the data?

Data Splitting

data2

	fruit_label	mass	width	height	color_score
0	1	192	8.4	7.3	0.55
1	1	180	8.0	6.8	0.59
2	1	176	7.4	7.2	0.60
3	2	86	6.2	4.7	0.80
4	2	84	6.0	4.6	0.79

. . .

54	4	116	6.1	8.5	0.71
55	4	116	6.3	7.7	0.72
56	4	116	5.9	8.1	0.73
57	4	152	6.5	8.5	0.72
58	4	118	6.1	8.1	0.70

X_train

	mass	width	height	color_score
9	172	7.4	7.0	0.89
10	166	6.9	7.3	0.93
13	164	7.3	7.7	0.70
27	204	7.5	9.2	0.77
55	116	6.3	7.7	0.72
36	160	7.1	7.6	0.76
49	132	5.8	8.7	0.73
17	168	7.5	7.6	0.73
57	152	6.5	8.5	0.72
32	164	7.2	7.0	0.80
52	118	5.9	8.0	0.72
12	154	7.0	7.1	0.88
44	200	7.3	10.5	0.72
42	154	7.2	7.2	0.82
16	156	7.6	7.5	0.67

y_train

9	1
10	1
13	1
27	3
55	4
36	3
49	4
17	1
57	4
32	3
52	4
12	1
44	4
42	3
16	1

X_test

	mass	width	height	color_score
46	216	7.3	10.2	0.71
29	160	7.0	7.4	0.81
35	150	7.1	7.9	0.75
56	116	5.9	8.1	0.73
23	170	7.6	7.9	0.88
48	174	7.3	10.1	0.72
28	140	6.7	7.1	0.72
0	192	8.4	7.3	0.55
30	158	7.1	7.5	0.79
1	180	8.0	6.8	0.59
54	116	6.1	8.5	0.71
53	120	6.0	8.4	0.74

y_test

46	4
29	3
35	3
56	4
23	1
48	4
28	3
0	1
30	3
1	1
54	4
53	4

Holdout data: Keep this portion for **testing & reporting the model performance**

Data Splitting

- Scikit learn helps us splitting the data using the following function

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(data2.iloc[:,1:],
                                                    data2.iloc[:,0],
                                                    test_size=0.2,
                                                    random_state=20 )
```

- **Parameter: test_size** is responsible for making the division of the data, in the above example, **test_size = 0.2**

- with **test_size = 0.2,** the data will be divided into **80% training** and **20% testing**.

Remember:

- Your data may not be ready for **modeling, yet.** We just kept a portion of it for **testing stage**.
- Now it is time to perform some data ***preprocessing*** such as encoding, imputation, normalization, etc.
- The parameters or settings used in the previous step must be **retained** and repeated on testing data.
  - For example, if we perform normalization using **standard scaler**, then we should use the **mean** and **std** of the training dataset to normalize the testing data.
- The above is very important to avoid ***data leakage*** and building more robust model.

Possible tasks that lead to data leakage

- **During data preparation**, you need to pay attention to data leakage because it might happen! For example,
- **During the EDA**, you noticed that your data has different scales (i.e., range of values), or you're considering dataset with missing values, or having some outliers, etc. the question is

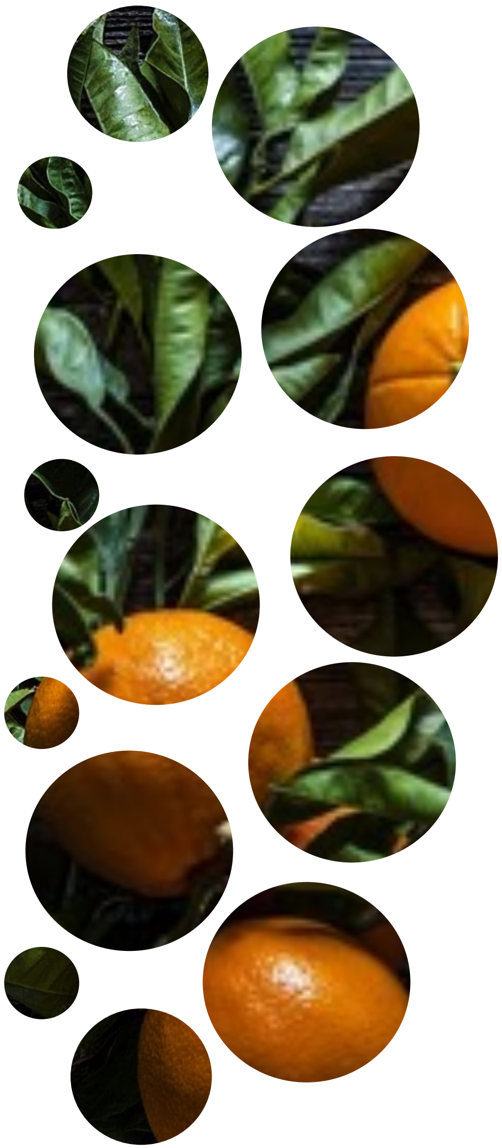
What are you going to do?

- Similarly, during cross validation!
- Rule of thumb: holdout some raw data (10 to 20%) for final model testing.

Data leakage issue

- **Robustness** is a cornerstone that we're trying to build ML models to achieve.
- **A data leakage is happening** when critical information are shared between the training and testing data, somehow!
- Data leakage leads to overly optimistic or even completely invalid ML model
- Data leakage may destroy model robustness/generalization, so that *we cannot deploy it to avoid unexpected consequences*
- The ML framework should only rely on training data to create the model, and the training parameters should be carried on to the testing stage.

Source: O'Neil, C., & Schutt, R. (2013). *Doing data science: Straight talk from the frontline*. " O'Reilly Media, Inc.".



ML workflow Demo

1. Get some images using google search (fruits)
2. Build an ML to recognize images
3. Test the created model with new images and check results

[Teachable Machines]

Nonparametric Methods

K-Nearest Neighbor

Nonparametric Methods

- A **nonparametric** model has no bounded set of parameters.
- Number of parameters are growing by ***increasing training samples***
- Such family of methods is called **instance-based learning**, or **memory-based learning** methods
- The simplest instance-based learning method is a **table lookup,** where new instance can be classified using a simple search to similar template examples from the table

Source: Artificial Intelligence: A Modern Approach, page 737

K-Nearest Neighbor (*k*NN) Classifier

- *k*NN is a popular and simple ML algorithm
- The *k*NN algorithm assumes no structure about the data
- It can be used for ***regression*** or ***classification*** problems
- It is heavily depending on
  - **The *training dataset representation*, and**
  - **The *hyperparameters* set for the model**
- So, quality training data is major requirement for *k*NN to enable the machine to model and make predictions for the unseen data.

How it works?

- In particular, *kNN* has three steps:

- Given a training set ***X_train*** with labels ***y_train*** and given a new instance ***x_test*** to be classified:

  1. Find the most similar ***k*** instances to ***x_test*** that are in ***X_train***
  2. Get the labels ***y_train*** of those similar instances to ***x_test***
  3. Predict the label for ***x_test*** by combining those identified labels

Combining labels is usually done using **majority vote**

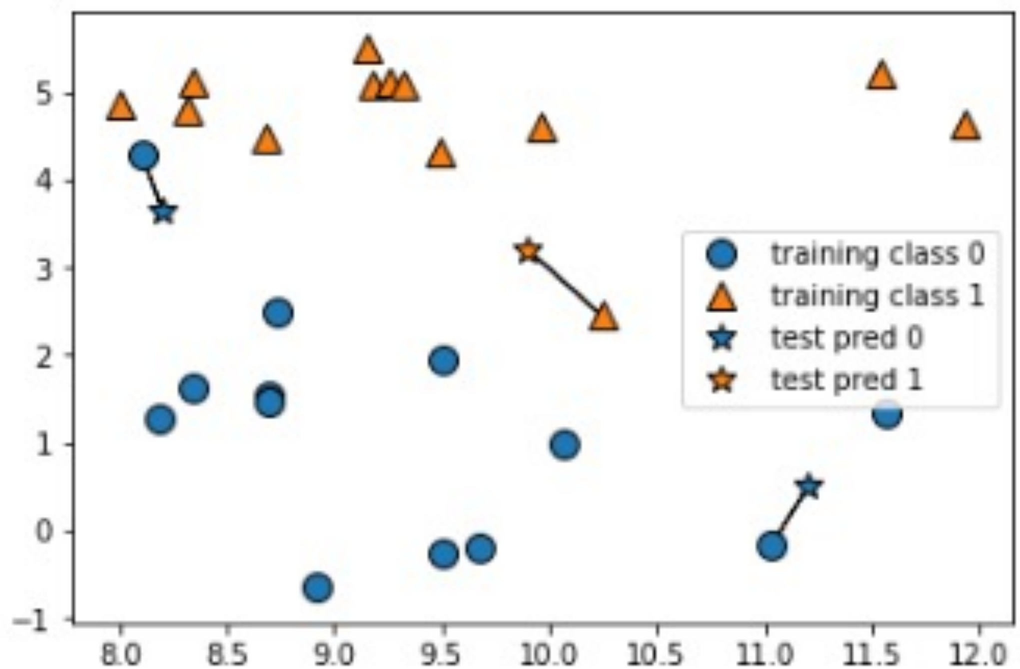
kNN algorithm hyperparameters

- A distance metric
  - Typically, Euclidean (Makowski with p = 2)
- Number of 'nearest' neighbors to look at
  - e.g. $k = 5$ neighbors
- Optional weighting function on the neighbor points
  - Ignored (uniform) , or distance

Illustration

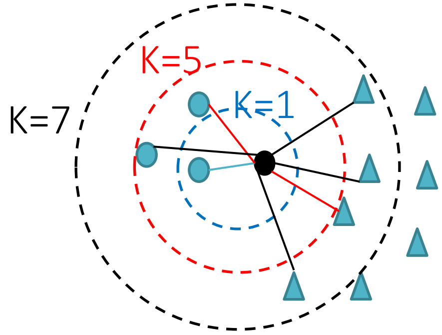
- There are two classes in this example (circles and triangles)
- The testing samples are plotted as stars.
- In case of k = *1*, for each **test** sample *kNN* finds the ***nearest training*** *sample and* assign its label to the **test** sample.

What if we increase K ?



K Effect on *kNN*

- Parameter ***K*** could be neither too large nor too small!
- If ***K*** is too large, the test samples tend to be classified as the most popular class in the training set rather than the most similar one.
- If K is too small, outliers or noisy data may affect the classification heavily.



Scikit Learn (Model)

```python
# load  KNeighborsClassifier package
from sklearn.neighbors import KNeighborsClassifier
# make an instance of the model
knn = KNeighborsClassifier(n_neighbors = 5,
                           weights = 'uniform', #distance,
                           p = 2, #for distance,
                           metric = 'minkowski', #
                           )
# train the model
knn.fit(Xb_tr, yb_tr)
# predict labels of new data
y_pred = knn.predict(Xb_ts)
# To validate and computethe accuracy on test set
knn.score(Xb_ts,yb_ts)
```

Ploting the *kNN* decision Space

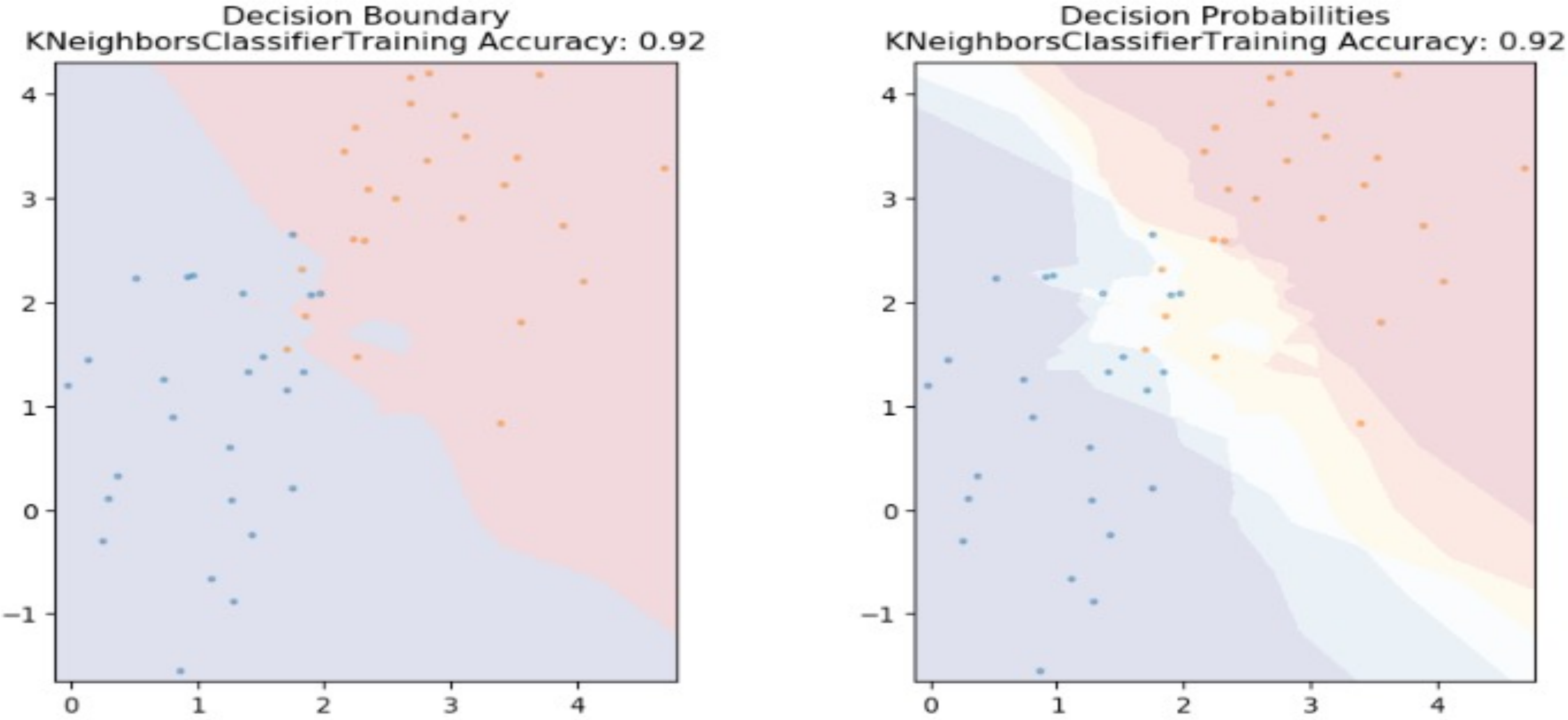
- The code of the plotting function is ready for you in the notebook.
- You need to run the Jupyter cell that contains the code before using the function to plot decision spaces.

```
[99]
# check decision space (Training)
plot_decisison (Xtr,y_tr, knn )
```

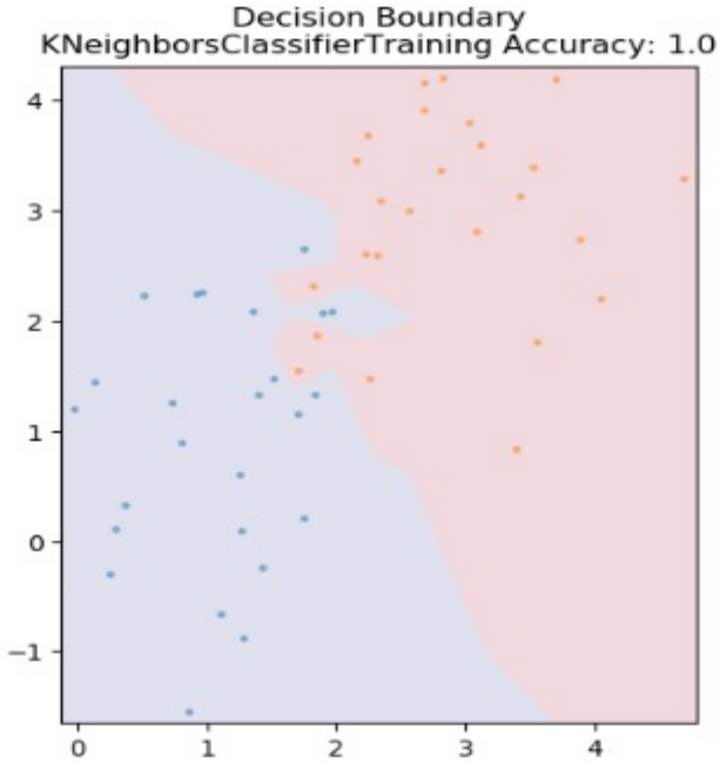
```
# check decision space (Tesing)
plot_decisison (Xts,y_ts, knn )
```

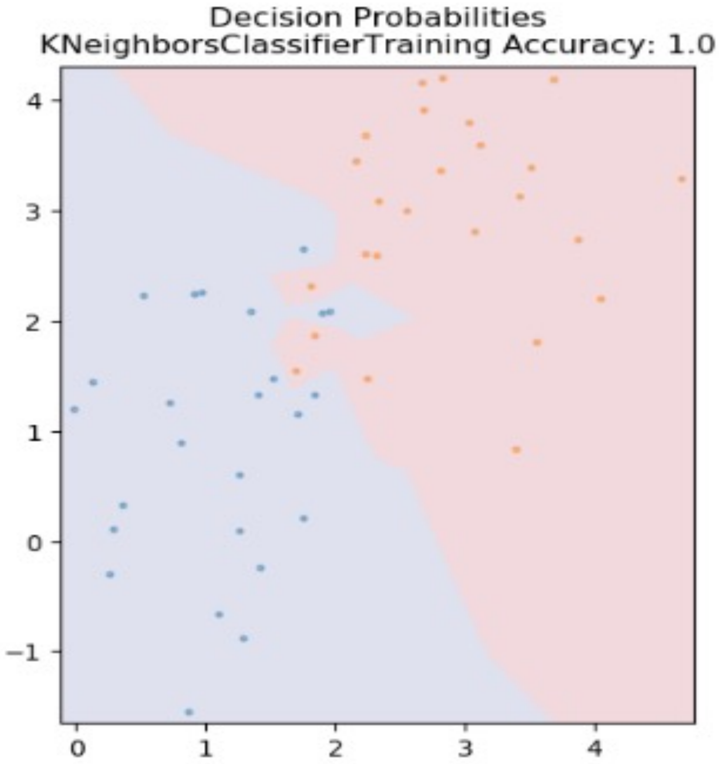
kNN decision space

- $k = 5$,



$k = 1$





- Even though it is a simple approach, ***k*NN** model can often produce surprisingly high performance.

Pros and Cons

- No assumptions about data — useful, for example, for nonlinear data
- Simple algorithm — to explain and understand/interpret
- High accuracy (relatively) — but requires more memory

- High memory requirements
- kNN struggles in case of high number of variables (Curse of dimensionality)
- Sensitive to irrelevant features, imbalanced data, and outliers

Exercises (1 - 2)

- KNearest Neighbor Classifier (Binary Classification)
- KNearest Neighbor Classifier (Multi-Classification)