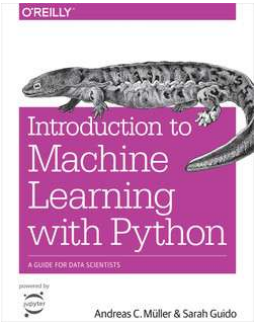
Applied Machine Learning
k-NN for Classification

BSc course Informatiekunde 2026
https://staff.fnwi.uva.nl/a.visser/education/AML

Arnoud Visser
Intelligent Robotics Lab & Computer Vision Lab
Informatics Institute
Universiteit van Amsterdam
A.Visser@uva.nl

Illustrations courtesy of Maarten Marx, Sarah Guido, Yolanda Hagar, and many others.



Section 2.3.2

Schedule

- Week 1: Python/Pandas/Numpy prerequisites + Data preprocessing
- Week 2: Regression I: KNN + Linear Regression + Metrics
- Week 3: Regression II: Polynomial regression + SVMs + MLP + under/overfitting + regularization

MidTerm exam

- Week 5: Classification I: KNN + Logistic Regression + Metrics
- Week 6: Classification II: Trees/RF + SVMs + Grid search
- Week 7: Clustering: K-means + DBSCAN + ….

Mid-Term exam

- 40% of grade
- All partial grades are averaged: pass when average > 5.5

- Difference between model-answers and grading-model!

Mid-Term exam – Programming exercise

```
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.svm import SVR
from sklearn.metrics import mean_squared_error
import matplotlib.pyplot as plt
import pandas as pd
import numpy as np
import math

# 1. Load the data
df = pd.read_csv('wineQualityReds.csv')

# 2. Drop the first column
df = df.drop(df.columns[0],axis=1)

# 3. Split the target off
X = df.drop('quality', axis=1).values
y = df['quality'].values

# 4. Train-test split
X_train, X_test, y_train, y_test = ...

# 5. Normalise
X_scaler = ...
X_train_scaled = ...
X_test_scaled = ...

y_scaler = ...
y_train_scaled =....ravel()
y_test_scaled = ....ravel()
```

Mid-Term exam – Programming exercise

```
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.svm import SVR
from sklearn.metrics import mean_squared_error
import matplotlib.pyplot as plt
import pandas as pd
import numpy as np
import math

# 1. Load the data
df = pd.read_csv('wineQualityReds.csv')

# 2. Drop the first column
df = df.drop(df.columns[0],axis=1)

# 3. Split the target off
X = df.drop('quality', axis=1).values
y = df['quality'].values

# 4. Train-test split
X_train, X_test, y_train, y_test = ...

# 5. Normalise
X_scaler = ...
X_train_scaled = ...
X_test_scaled = ...

y_scaler = ...
y_train_scaled =....ravel()
y_test_scaled = ....ravel()
```

Mid-Term exam – Programming exercise

```
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.svm import SVR
from sklearn.metrics import mean_squared_error
import matplotlib.pyplot as plt
import pandas as pd
import numpy as np
import math

# 1. Load the data
df = pd.read_csv('wineQualityReds.csv')

# 2. Drop the first column
df = df.drop(df.columns[0],axis=1)

# 3. Split the target off
X = df.drop('quality', axis=1).values
y = df['quality'].values

# 4. Train-test split
X_train, X_test, y_train, y_test = ...

# 5. Normalise
X_scaler = ...
X_train_scaled = ...
X_test_scaled = ...

y_scaler = ...
y_train_scaled =....ravel()
y_test_scaled = ....ravel()
```

Mid-Term exam – Programming exercise

```
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.svm import SVR
from sklearn.metrics import mean_squared_error
import matplotlib.pyplot as plt
import pandas as pd
import numpy as np
import math

# 1. Load the data
df = pd.read_csv('wineQualityReds.csv')

# 2. Drop the first column
df = df.drop(df.columns[0],axis=1)

# 3. Split the target off
X = df.drop('quality', axis=1).values
y = df['quality'].values

# 4. Train-test split
X_train, X_test, y_train, y_test = ...

# 5. Normalise
X_scaler = ...
X_train_scaled = ...
X_test_scaled = ...

y_scaler = ...
y_train_scaled =....ravel()
y_test_scaled = ....ravel()
```

Mid-Term exam – Programming exercise

```
# 5. Normalise
X_scaler = ...
X_train_scaled = ...
X_test_scaled = ...

y_scaler = ...
y_train_scaled =....ravel()
y_test_scaled = ....ravel()
```

```
?np.ravel()
```

Signature: np.ravel(a, order='C')
Docstring:
Return a contiguous flattened array.

A 1-D array, containing the elements of the input, is returned. A copy is
made only if needed.

Mid-Term exam – Programming exercise

```
# 5. Normalise
X_scaler = ...
X_train_scaled = ...
X_test_scaled = ...

y_scaler = ...
y_train_scaled =....ravel()
y_test_scaled = ....ravel()
```

```
?np.ravel()
```

Signature: np.ravel(a, order='C')
Docstring:
Return a contiguous flattened array.

Examples

It is equivalent to ``reshape(-1, order=order)``.

```
>>> x = np.array([[1, 2, 3], [4, 5, 6]])
>>> np.ravel(x)
array([1, 2, 3, 4, 5, 6])

>>> x.reshape(-1)
array([1, 2, 3, 4, 5, 6])
```

Mid-Term exam – Programming exercise

```
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.svm import SVR
from sklearn.metrics import mean_squared_error
import matplotlib.pyplot as plt
import pandas as pd
import numpy as np
import math

# 1. Load the data
df = pd.read_csv('wineQualityReds.csv')

# 2. Drop the first column
df = df.drop(df.columns[0],axis=1)

# 3. Split the target off
X = df.drop('quality', axis=1).values
y = df['quality'].values

# 4. Train-test split
X_train, X_test, y_train, y_test = ...

# 5. Normalise
## X_scaler = ...
## X_train_scaled = ...
## X_test_scaled = ...

## y_scaler = ...
##y_train_scaled =....ravel()
##y_test_scaled = ....ravel()
```

Mid-Term exam – Programming exercise

```
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.svm import SVR
from sklearn.metrics import mean_squared_error
import matplotlib.pyplot as plt
import pandas as pd
import numpy as np
import math

# 1. Load the data
df = pd.read_csv('wineQualityReds.csv')

# 2. Drop the first column
df = df.drop(df.columns[0],axis=1)

# 3. Split the target off
X = df.drop('quality', axis=1).values
y = df['quality'].values

# 4. Train-test split
X_train, X_test, y_train, y_test = ...

# 5. Normalise
## X_scaler = ...
## X_train_scaled = ...
## X_test_scaled = ...

## y_scaler = ...
##y_train_scaled =....ravel()
##y_test_scaled = ....ravel()
```

TEXT BLOCK

1. Load the Red Wine Dataset by using pandas read_csv-function on wineQualityReds.csv
2. Drop the first column, which contains no information, other than the sample ID.
3. Split the dataset in the features X and the target y, which is the last column ('quality')
4. Split the data into a training set (80%) and a test set (20%). Use `random_state=2009`

Mid-Term exam – Programming exercise

```
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.svm import SVR
from sklearn.metrics import mean_squared_error
import matplotlib.pyplot as plt
import pandas as pd
import numpy as np
import math

# 1. Load the data
df = pd.read_csv('wineQualityReds.csv')

# 2. Drop the first column
df = df.drop(df.columns[0],axis=1)

# 3. Split the target off
X = df.drop('quality', axis=1).values
y = df['quality'].values

# 4. Train-test split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2, random_state = 2009)

# 5. Normalise
## X_scaler = ...
## X_train_scaled = ...
## X_test_scaled = ...

## y_scaler = ...
##y_train_scaled =....ravel()
##y_test_scaled = ....ravel()
```

TEXT BLOCK

1. Load the Red Wine Dataset by using pandas read_csv-function on wineQualityReds.csv
2. Drop the first column, which contains no information, other than the sample ID.
3. Split the dataset in the features X and the target y, which is the last column ('quality')
4. Split the data into a training set (80%) and a test set (20%). Use `random_state=2009`

Mid-Term exam – Programming exercise

```
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.svm import SVR
from sklearn.metrics import mean_squared_error
import matplotlib.pyplot as plt
import pandas as pd
import numpy as np
import math

# 1. Load the data
df = pd.read_csv('wineQualityReds.csv')

# 2. Drop the first column
df = df.drop(df.columns[0],axis=1)

# 3. Split the target off
X = df.drop('quality', axis=1).values
y = df['quality'].values

# 4. Train-test split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2, random_state = 2009)

# 5. Normalise
## X_scaler = ...
## X_train_scaled = ...
## X_test_scaled = ...

## y_scaler = ...
##y_train_scaled =....ravel()
##y_test_scaled = ....ravel()
```

TEXT BLOCK

1. Load the Red Wine Dataset by using pandas read_csv-function on wineQualityReds.csv
2. Drop the first column, which contains no information, other than the sample ID.
3. Split the dataset in the features X and the target y, which is the last column ('quality')
4. Split the data into a training set (80%) and a test set (20%). Use `random_state=2009`

Mid-Term exam – Programming exercise

```
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.svm import SVR
from sklearn.metrics import mean_squared_error
import matplotlib.pyplot as plt
import pandas as pd
import numpy as np
import math

# 1. Load the data
df = pd.read_csv('wineQualityReds.csv')

# 2. Drop the first column
df = df.drop(df.columns[0],axis=1)

# 3. Split the target off
X = df.drop('quality', axis=1).values
y = df['quality'].values

# 4. Train-test split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2, random_state = 2009)

# 5. Normalise
## X_scaler = ...
## X_train_scaled = ...
## X_test_scaled = ...

## y_scaler = ...
##y_train_scaled =....ravel()
##y_test_scaled = ....ravel()
```

TEXT BLOCK

1. Load the Red Wine Dataset by using pandas read_csv-function on wineQualityReds.csv
2. Drop the first column, which contains no information, other than the sample ID.
3. Split the dataset in the features X and the target y, which is the last column ('quality')
4. Split the data into a training set (80%) and a test set (20%). Use `random_state=2009`

Mid-Term exam – Programming exercise

```
…

# 1. Load the data
df = pd.read_csv('wineQualityReds.csv')

# 2. Drop the first column
df = df.drop(df.columns[0],axis=1)

# 3. Split the target off
X = df.drop('quality', axis=1).values
y = np.asarray(df['quality'].values)
y = y.reshape(-1,1)

# 4. Train-test split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2, random_state = 2009)

# 9. Make a small table with the difference between the predicted quality and the real value (the quality according to the expert) for the first 10 training samples

for i in range(6):
    print("y_test:", y_test[i])              print("y_test:", y_test[i], "best_pred:", best_pred[i])
```

TEXT BLOCK

1. Load the Red Wine Dataset by using pandas read_csv-function on wineQualityReds.csv
2. Drop the first column, which contains no information, other than the sample ID.
3. Split the dataset in the features X and the target y, which is the last column ('quality')
4. Split the data into a training set (80%) and a test set (20%). Use `random_state=2009`

```
y_test: [5]
y_test: [5]
y_test: [5]
y_test: [5]
y_test: [7]
y_test: [6]
```

4 points

Mid-Term exam – Programming exercise

```
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.svm import SVR
from sklearn.metrics import mean_squared_error
import matplotlib.pyplot as plt
import pandas as pd
import numpy as np
import math

…

# 6. Train three Support Vector Regressor models with three different kernels
kernel_names = ["linear", "poly", "rbf"]

lowest_test_rmse = float('inf')
best_k = None
best_model = None

for k in kernel_names:
    model = ...
    model.fit(...)

    # 7. Create  the training and test prediction vectors for each of the models
    y_train_pred_scaled = ...
    y_test_pred_scaled = ...
    # 8. Inverse the scaling transform for both prediction vectors
    y_train_pred = ...
    y_test_pred = ...
```

Mid-Term exam – Programming exercise

```
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.svm import SVR
from sklearn.metrics import mean_squared_error
import matplotlib.pyplot as plt
import pandas as pd
import numpy as np
import math
```

…

6. Train three Support Vector Regressor models with three different kernels: "linear", "poly" and "rbf"
7. Create the training and test prediction vectors for each of the models

```
# 6. Train three Support Vector Regressor models with three different kernels
kernel_names = ["linear", "poly", "rbf"]

lowest_test_rmse = float('inf')
best_k = None
best_model = None

for k in kernel_names:
    model = ...
    model.fit(...)

    # 7. Create  the training and test prediction vectors for each of the models
    y_train_pred_scaled = ...
    y_test_pred_scaled = ...
    # 8. Inverse the scaling transform for both prediction vectors
    y_train_pred = ...
    y_test_pred = ...
```

Mid-Term exam – Programming exercise

```
from sklearn.svm import SVR
…
```

6. Train three Support Vector Regressor models with three different kernels: "linear", "poly" and "rbf"

```
# 6. Train three Support Vector Regressor models with three different kernels
kernel_names = ["linear", "poly", "rbf"]
…
for k in kernel_names:
    model = ...
    model.fit(...)
```

```
?sklearn.svm.SVR
```

```
Init signature:
sklearn.svm.SVR(
    *,
    kernel='rbf',
    degree=3,
    gamma='scale',
    coef0=0.0,
    tol=0.001,
    C=1.0,
    epsilon=0.1,
    shrinking=True,
    cache_size=200,
    verbose=False,
    max_iter=-1,
)
```

Mid-Term exam – Programming exercise

```
from sklearn.svm import SVR
…
```

6. Train three Support Vector Regressor models with three different kernels: "linear", "poly" and "rbf"

```
# 6. Train three Support Vector Regressor models with three different kernels
kernel_names = ["linear", "poly", "rbf"]
…
for k in kernel_names:
    model = ...
    model.fit(...)
```

```
?sklearn.svm.SVR
```

Docstring:
Epsilon-Support Vector Regression.

The free parameters in the model are C and epsilon.

The implementation is based on libsvm. The fit time complexity
is more than quadratic with the number of samples which makes it hard
to scale to datasets with more than a couple of 10000 samples. For large
datasets consider using :class:`~sklearn.svm.LinearSVR` or
:class:`~sklearn.linear_model.SGDRegressor` instead, possibly after a
:class:`~sklearn.kernel_approximation.Nystroem` transformer or
other :ref:`kernel_approximation`.

Mid-Term exam – Programming exercise

```
from sklearn.svm import SVR
…
```

6. Train three Support Vector Regressor models with three different kernels: "linear", "poly" and "rbf"

```
# 6. Train three Support Vector Regressor models with three different kernels
kernel_names = ["linear", "poly", "rbf"]
…
for k in kernel_names:
    model = ...
    model.fit(...)
```

```
?sklearn.svm.SVR
```

```
Examples
--------
>>> from sklearn.svm import SVR
>>> from sklearn.pipeline import make_pipeline
>>> from sklearn.preprocessing import StandardScaler
>>> import numpy as np
>>> n_samples, n_features = 10, 5
>>> rng = np.random.RandomState(0)
>>> y = rng.randn(n_samples)
>>> X = rng.randn(n_samples, n_features)
>>> regr = make_pipeline(StandardScaler(), SVR(C=1.0, epsilon=0.2))
>>> regr.fit(X, y)
```

Mid-Term exam – Programming exercise

```
 5. Train three Support Vector Regressor models with three different kernels
kernel_names = ["linear", "poly", "rbf"]

lowest_test_rmse = float('inf')
best_k = None
best_model = None

for k in kernel_names:
    model = SVR(kernel = k)
    model.fit(X_train, y_train)

    # 7. Create  the training and test prediction vectors for each of the models
    y_train_pred = model.predict(X_train)
    y_test_pred = model.predict(X_test)


# 9. Make a small table with the difference between the predicted quality and the real value (the quality according to the expert) for the first 10 training samples

for i in range(6):
    print("y_test:", y_test[i], " latest_pred :", y_test_pred[i]))
```

6. Train three Support Vector Regressor models with three different kernels: "linear", "poly" and "rbf"
7. Create the training and test prediction vectors for each of the models

```
y_test: 5 latest_pred: 5.182566555468947
y_test: 5 latest_pred: 5.048996882586567
y_test: 5 latest_pred: 5.719128045873678
y_test: 5 latest_pred: 5.189376835172094
y_test: 7 latest_pred: 5.357303101524947
y_test: 6 latest_pred: 5.081694933841861
```

6 points

Mid-Term exam – Programming exercise

```
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.svm import SVR
from sklearn.metrics import mean_squared_error
import matplotlib.pyplot as plt
import pandas as pd
import numpy as np
import math

# 5. Normalise
X_scaler = ...
X_train_scaled = ...
X_test_scaled = ...
```

5. Normalise the selected features and the target using StandardScaler

```
?sklearn.preprocessing.StandardScaler
```

Examples

```
--------
>>> from sklearn.preprocessing import StandardScaler
>>> data = [[0, 0], [0, 0], [1, 1], [1, 1]]
>>> scaler = StandardScaler()
>>> print(scaler.fit(data))
StandardScaler()
>>> print(scaler.mean_)
[0.5 0.5]
>>> print(scaler.transform(data))
```

Mid-Term exam – Programming exercise

```
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.svm import SVR
from sklearn.metrics import mean_squared_error
import matplotlib.pyplot as plt
import pandas as pd
import numpy as np
import math

# 5. Normalise
X_scaler = StandardScaler()
X_scaler.fit(X_train)
X_train_scaled = X_scaler.transform(X_train)
X_test_scaled = X_scaler.transform(X_test)
...
```

5. Normalise the selected features and the target using StandardScaler

Mid-Term exam – Programming exercise

```
from sklearn.preprocessing import StandardScaler
from sklearn.svm import SVR

# 5. Normalise
X_scaler = StandardScaler()
X_scaler.fit(X_train)
X_train_scaled = X_scaler.transform(X_train)
X_test_scaled = X_scaler.transform(X_test)

for k in kernel_names:
    model = SVR(kernel = k)
    model.fit(X_train_scaled, y_train)

    # 7. Create  the training and test prediction vectors for each of the models
    y_train_pred = model.predict(X_train_scaled)
    y_test_pred = model.predict(X_test_scaled)


# 10. For the  10 test samples, show the difference in prediction-value and the actual-value of the best model

for i in range(6):
    print("y_test:", y_test[i], "latest_pred:", y_test_pred[i])
```

5. Normalise the selected features and the target using StandardScaler

```
y_test: 5 latest_pred: 4.87146606516193
y_test: 5 latest_pred: 4.707594359484559
y_test: 5 latest_pred: 5.27273634277968 65
y_test: 5 latest_pred: 5.264084272409812
y_test: 7 latest_pred: 7.023631826635567
y_test: 6 latest_pred: 5.055762797414054
```

7 points

Mid-Term exam – Programming exercise

```
from sklearn.preprocessing import StandardScaler
from sklearn.svm import SVR

# 5. Normalise
…..
y_scaler = StandardScaler()
y_train_scaled = y_scaler.fit_transform(y_train.reshape(-1, 1)).ravel()
y_test_scaled = y_scaler.transform(y_test.reshape(-1, 1)).ravel()

for k in kernel_names:
    model = SVR(kernel = k)
    model.fit(X_train_scaled, y_train_scaled)

    # 7. Create  the training and test prediction vectors for each of the models
    y_train_pred_scaled = model.predict(X_train_scaled)
    y_test_pred_scaled = model.predict(X_test_scaled)
    # 8. Inverse the scaling transform for both prediction vectors
    y_train_pred = ...
    y_test_pred = ...


# 10. For the  10 test samples, show the difference in prediction-value and the actual-value of the best model

for i in range(6):
    print("y_test:", y_test[i], "latest_pred:", y_test_pred[i])
```

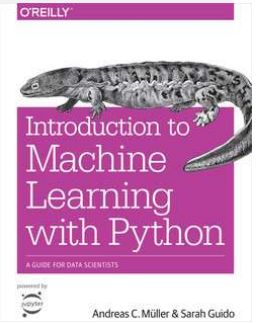
5. Normalise the selected features and the target using StandardScaler

Mid-Term exam – Programming exercise

```
from sklearn.preprocessing import StandardScaler
from sklearn.svm import SVR

# 5. Normalise
…..
y_scaler = StandardScaler()
y_train_scaled = y_scaler.fit_transform(y_train.reshape(-1, 1)).ravel()
y_test_scaled = y_scaler.transform(y_test.reshape(-1, 1)).ravel()

for k in kernel_names:
    model = SVR(kernel = k)
    model.fit(X_train_scaled, y_train_scaled)

    # 7. Create  the training and test prediction vectors for each of the models
    y_train_pred_scaled = model.predict(X_train_scaled)
    y_test_pred_scaled = model.predict(X_test_scaled)
    # 8. Inverse the scaling transform for both prediction vectors
    y_train_pred = ...
    y_test_pred = ...

# 10. For the  10 test samples, show the difference in prediction-value and the actual-value of the best mod

for i in range(6):
    print("y_test:", y_test[i], "latest_pred:", y_test_pred[i])
```

5. Normalise the selected features and the target using `StandardScaler`

then rotating back into the original space. This return to the original feature space can be done using the `inverse_transform` method. Here, we visualize the reconstruction of some faces using 10, 50, 100, 500, or 2,000 components (Figure 3-11):



page 154

Mid-Term exam – Programming exercise

```
from sklearn.preprocessing import StandardScaler
from sklearn.svm import SVR

# 5. Normalise
…..
y_scaler = StandardScaler()
y_train_scaled = y_scaler.fit_transform(y_train.reshape(-1, 1)).ravel()
y_test_scaled = y_scaler.transform(y_test.reshape(-1, 1)).ravel()

for k in kernel_names:
    model = SVR(kernel = k)
    model.fit(X_train_scaled, y_train_scaled)

    # 7. Create  the training and test prediction vectors for each of the models
    y_train_pred_scaled = model.predict(X_train_scaled)
    y_test_pred_scaled = model.predict(X_test_scaled)
    # 8. Inverse the scaling transform for both prediction vectors
    y_train_pred = ...
    y_test_pred = ...
```

5. Normalise the selected features and the target using `StandardScaler`

then rotating back into the original space. This return to the original feature space can be done using the `inverse_transform` method. Here, we visualize the reconstruction of some faces using 10, 50, 100, 500, or 2,000 components (Figure 3-11):

```
dir(sklearn.preprocessing.StandardScaler)
```

```
…
 'fit',
 'fit_transform',
 …
 'inverse_transform',
```

Mid-Term exam – Programming exercise

```
from sklearn.preprocessing import StandardScaler
from sklearn.svm import SVR

# 5. Normalise
…..
y_scaler = StandardScaler()
y_train_scaled = y_scaler.fit_transform(y_train.reshape(-1, 1)).ravel()
y_test_scaled = y_scaler.transform(y_test.reshape(-1, 1)).ravel()

for k in kernel_names:
    model = SVR(kernel = k)
    model.fit(X_train_scaled, y_train_scaled)

    # 7. Create  the training and test prediction vectors for each of the models
    y_train_pred_scaled = model.predict(X_train_scaled)
    y_test_pred_scaled = model.predict(X_test_scaled)
    # 8. Inverse the scaling transform for both prediction vectors
    y_train_pred = ...
    y_test_pred = ...
```

5. Normalise the selected features and the target using `StandardScaler`

then rotating back into the original space. This return to the original feature space can be done using the `inverse_transform` method. Here, we visualize the reconstruction of some faces using 10, 50, 100, 500, or 2,000 components (Figure 3-11):

```
?sklearn.preprocessing.StandardScaler.inverse_transform
```

Signature: sklearn.preprocessing.StandardScaler.inverse_transform(self, X, copy=**None**)
Docstring:
Scale back the data to the original representation.

page 134

Mid-Term exam – Programming exercise

```
from sklearn.preprocessing import StandardScaler
from sklearn.svm import SVR

# 5. Normalise
…..
y_scaler = StandardScaler()
y_train_scaled = y_scaler.fit_transform(y_train.reshape(-1, 1)).ravel()
y_test_scaled = y_scaler.transform(y_test.reshape(-1, 1)).ravel()

for k in kernel_names:
    model = SVR(kernel = k)
    model.fit(X_train_scaled, y_train_scaled)

    # 7. Create  the training and test prediction vectors for each of the models
    y_train_pred_scaled = model.predict(X_train_scaled)
    y_test_pred_scaled = model.predict(X_test_scaled)
    # 8. Inverse the scaling transform for both prediction vectors
    y_train_pred = y_scaler.inverse_transform(y_train_pred_scaled.reshape(-1, 1))
    y_test_pred = y_scaler.inverse_transform(y_test_pred_scaled.reshape(-1, 1))
```

5. Normalise the selected features and the target using StandardScaler

```
y_test: 5 latest_pred: 4.87146606516193
y_test: 5 latest_pred: 4.707594359484559
y_test: 5 latest_pred: 5.2727363427796865
y_test: 5 latest_pred: 5.264084272409812
y_test: 7 latest_pred: 7.023631826635567
y_test: 6 latest_pred: 5.055762797414054
```

9 points

Mid-Term exam – Programming exercise

```
from sklearn.metrics import mean_squared_error

# 9. Calculate the MSE of the training set and the test-set for all three models.
    train_rmse = ...
    test_rmse = ...

    if test_rmse < lowest_test_rmse:
        lowest_test_rmse = ...
        lowest_training_rmse = ...
        best_k = ...
        best_model = ...
        best_pred = ...

    print(f"k = {k}")
    print("Root Mean Squared Error Train:", round(train_rmse, 5))
    print("Root Mean Squared Error Test:", round(test_rmse, 5))
    print()
```

9. Calculate the MSE of the training set and the test-set for all three models.

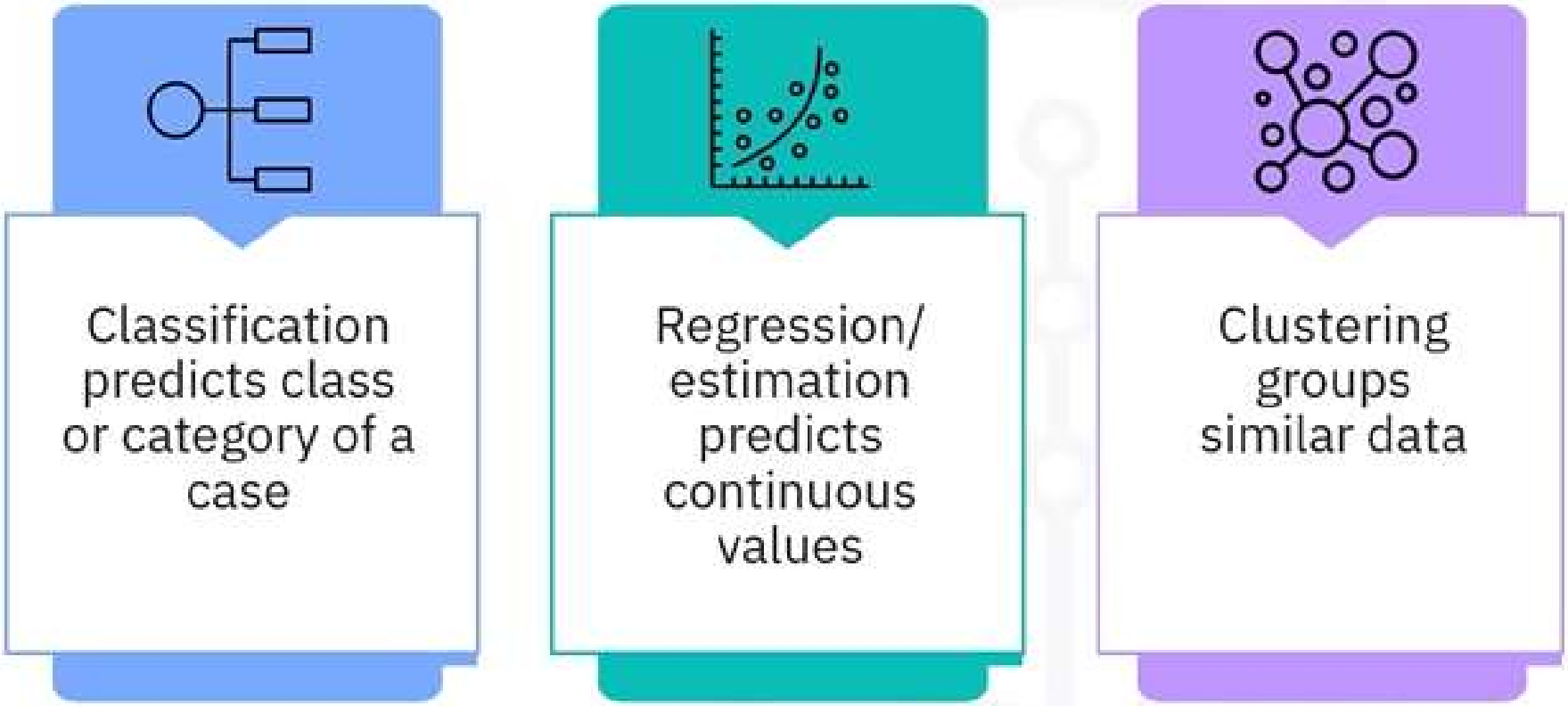
10 points

Mid-Term exam

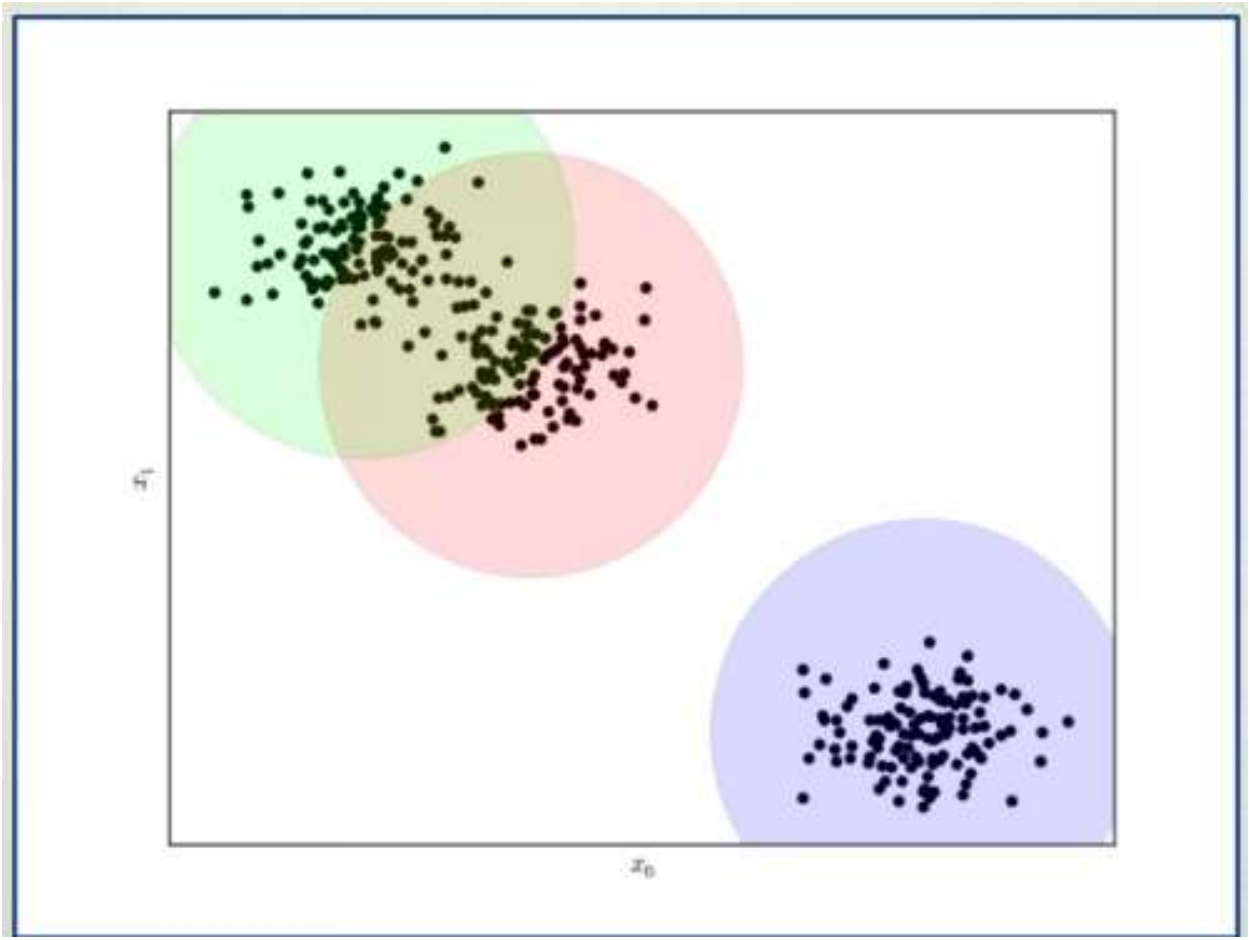
- Grade consist of both Theoretical & Programming part


```
from sklearn.* import MODEL

for m in MODEL_VERSIONS:
    model = MODEL(hyper_parameter = m)
    model.fit(X_train)

    y_train_pred = model.predict(X_train)
    y_test_pred = model.predict(X_test)
```

Introduction to Machine Learning



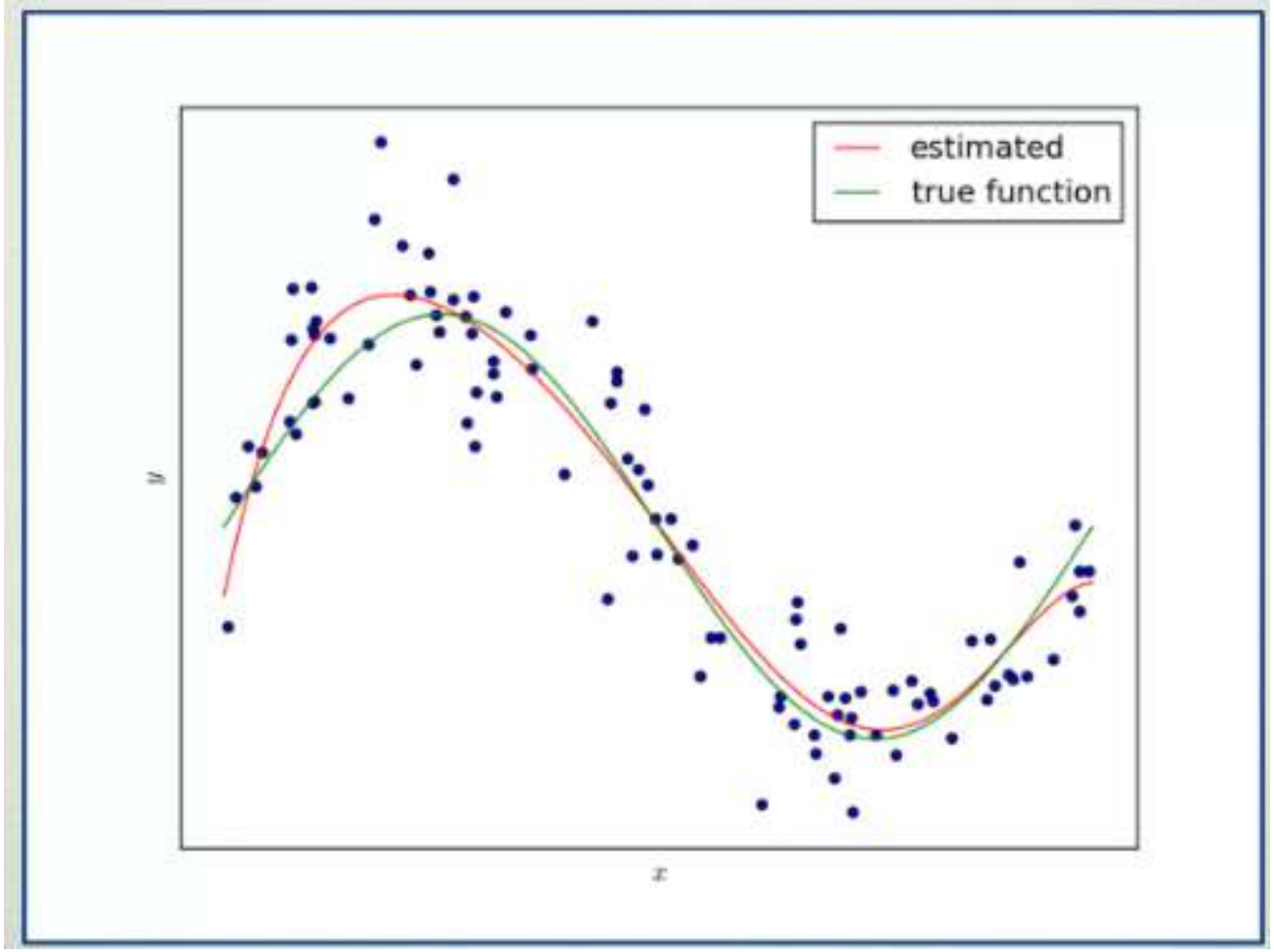
Clustering



- Given training data, determine a pattern of associated data points via their simularity of distance from each other.

Slide idea from Benjamin Bengfort

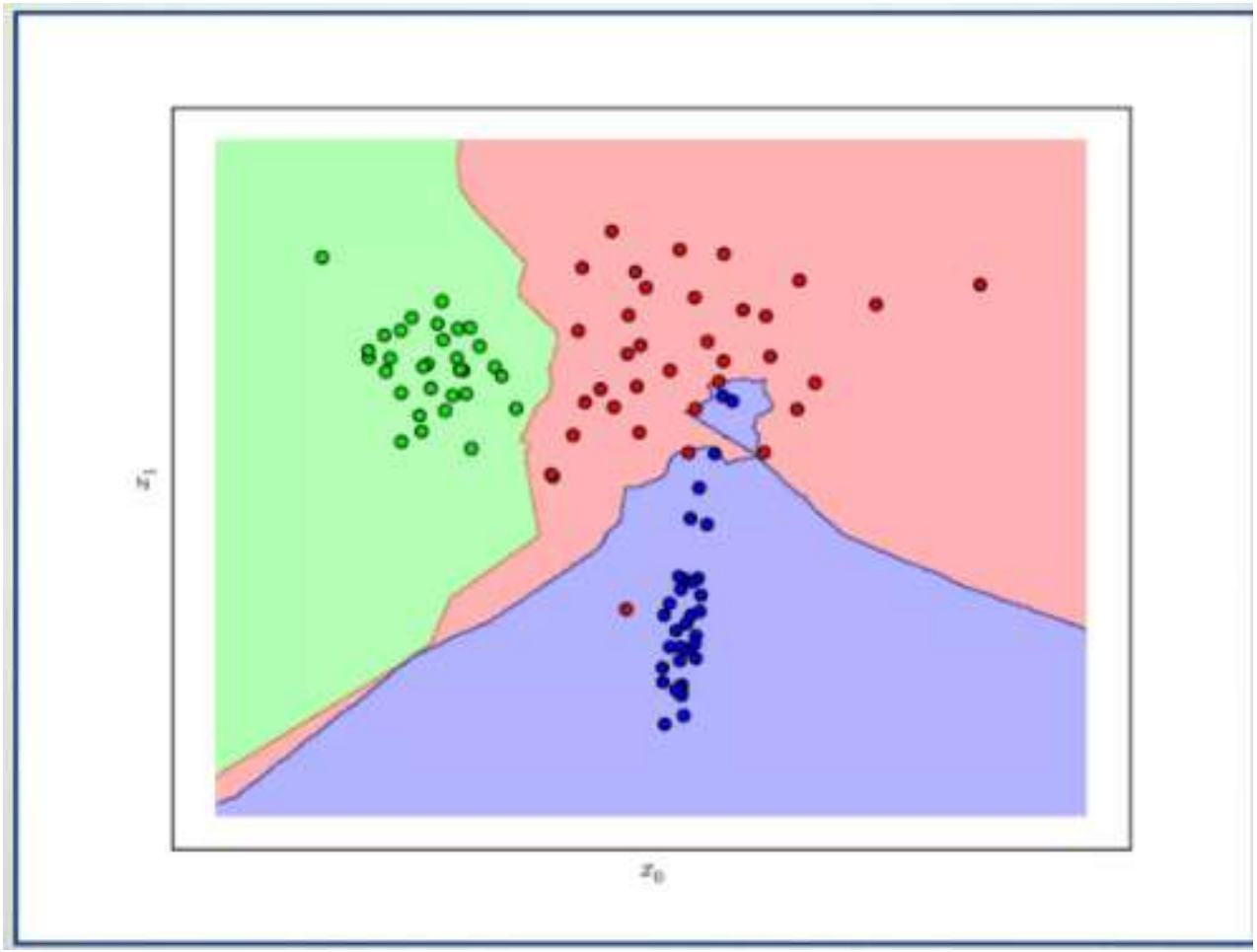
Regression



- Given training data predict the continues values in between – the function between input and output

Slide idea from Benjamin Bengfort

Classification



- Given training data, fit a function that can determine for any input, what the label is.

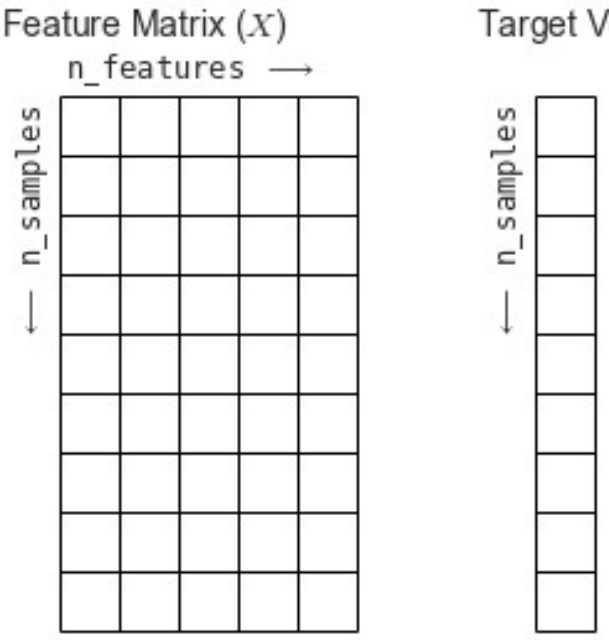
Slide idea from Benjamin Bengfort

Supervised learning

scikit learn • Model.predict(X) $\longrightarrow$ y

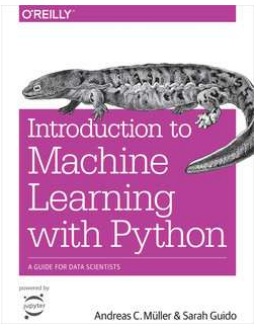
Regression

Classification



$y_i \equiv$ real number $(\mathbb{R})$

$y_i \equiv$ class $(\mathbb{N})$



Section 2.1

- Regression & Classification

Supervised learning

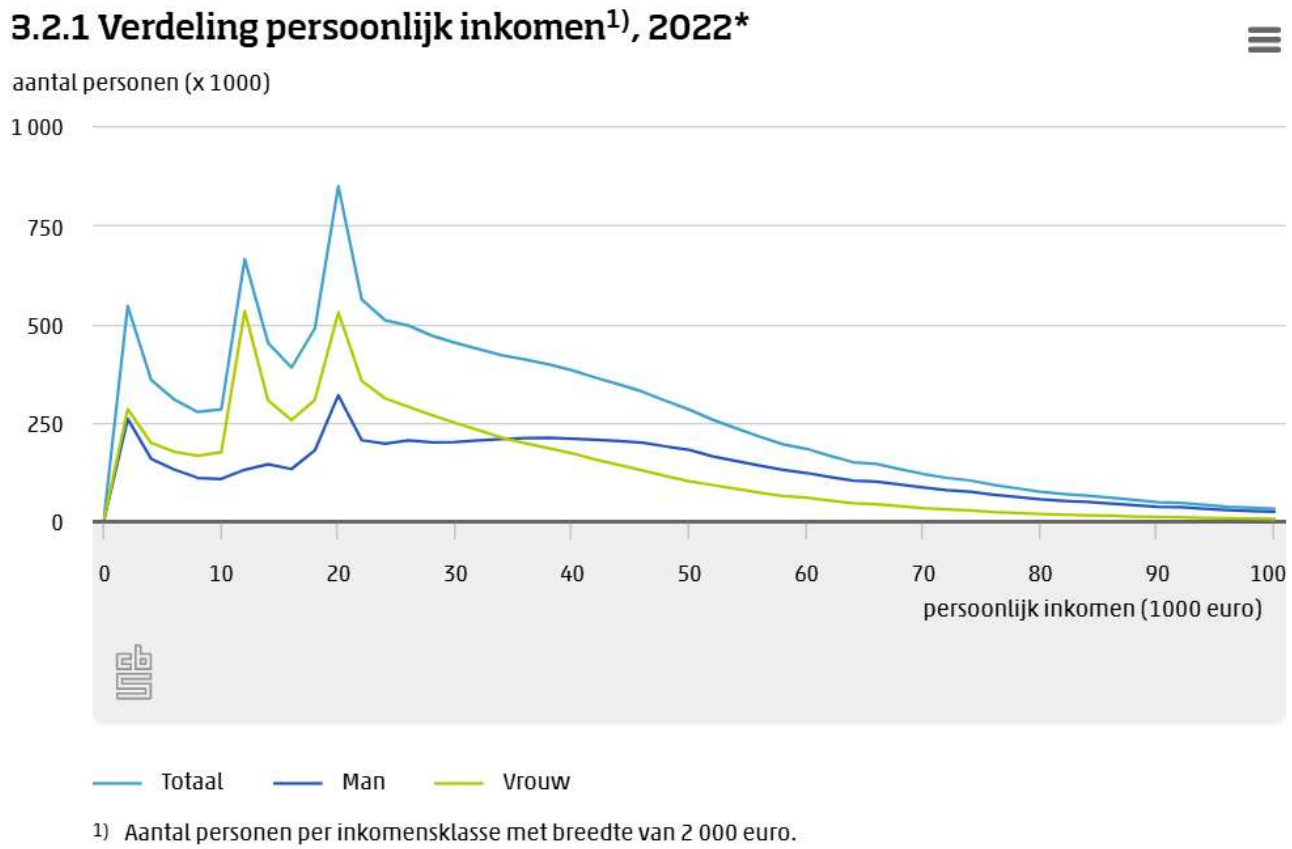
• Model.predict(X) $\longrightarrow$ y

Regression $y = f(X)$ with $y \equiv$ vector of floats

Classification $y = f(X)$ with $y \equiv$ vector of ‘integers’

Supervised learning

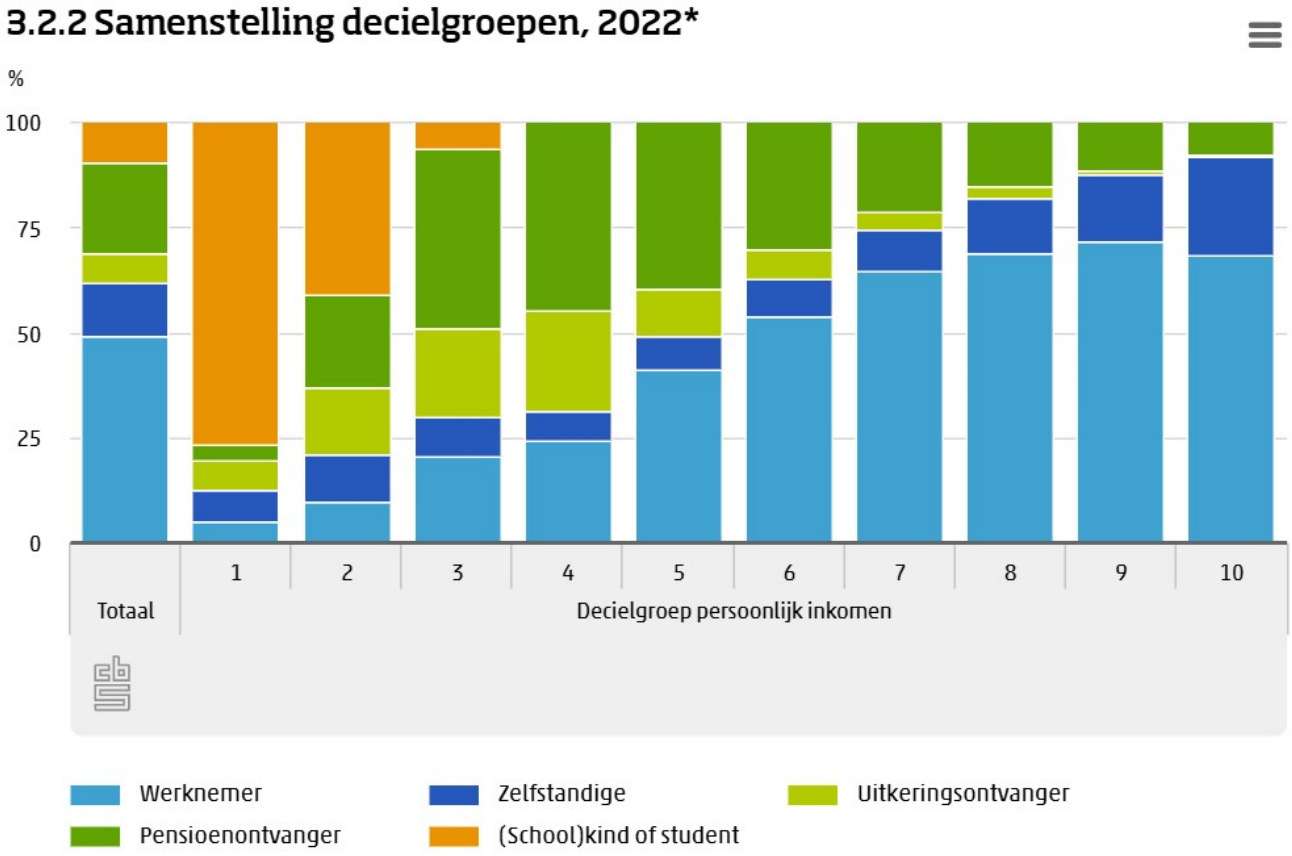
scikit learn • Model.predict(X) $\longrightarrow$ y



Regression $y = f(X)$ with $y \equiv$ vector of floats

Supervised learning

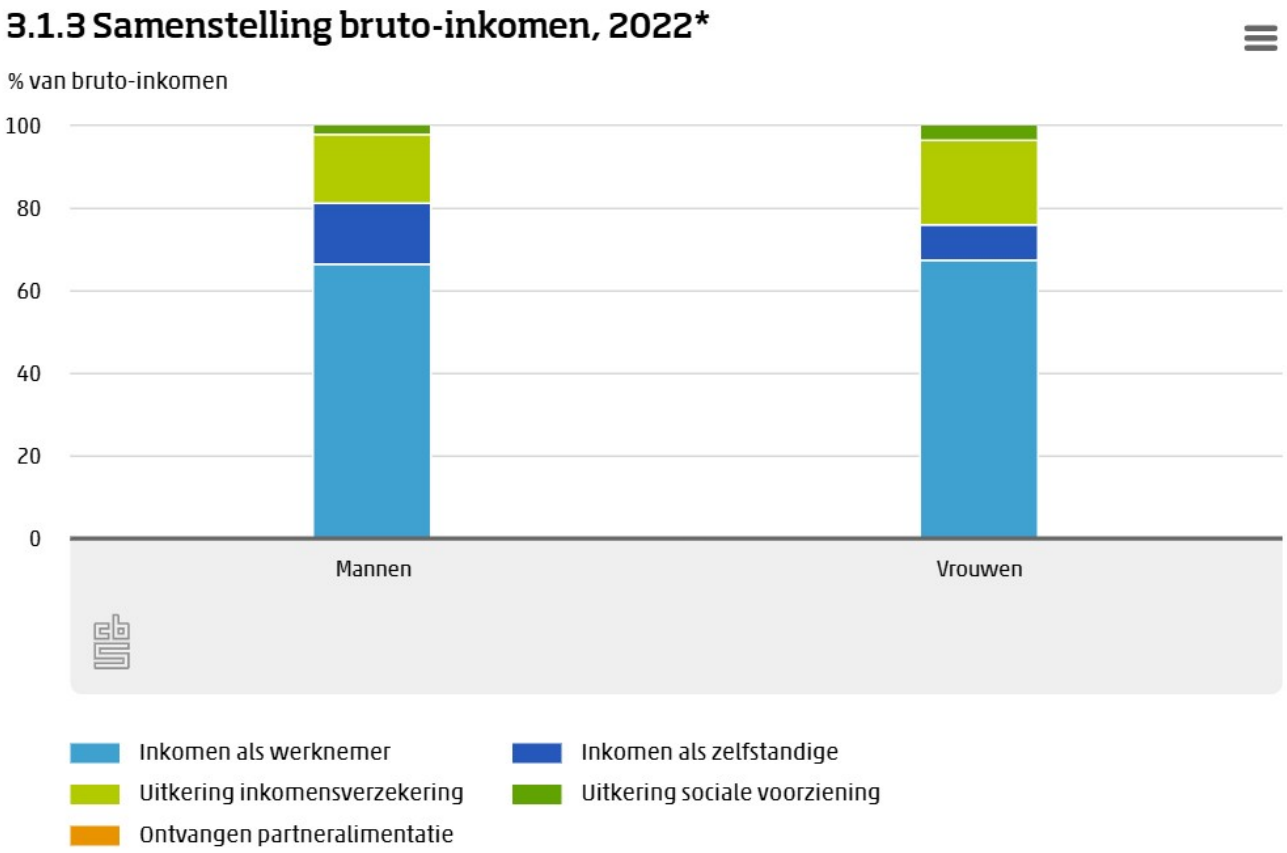
scikit learn • Model.predict(X) $\longrightarrow$ y



Classification $y = f(X)$ with $y \equiv$ vector of 'integers'

Supervised learning

scikit learn • Model.predict(X) $\longrightarrow$ y



Classification $y = f(X)$ with $y \equiv$ vector of <u>classes</u>

no ordering, just index

Supervised learning

scikit learn • Model.predict(X) $\longrightarrow$ y

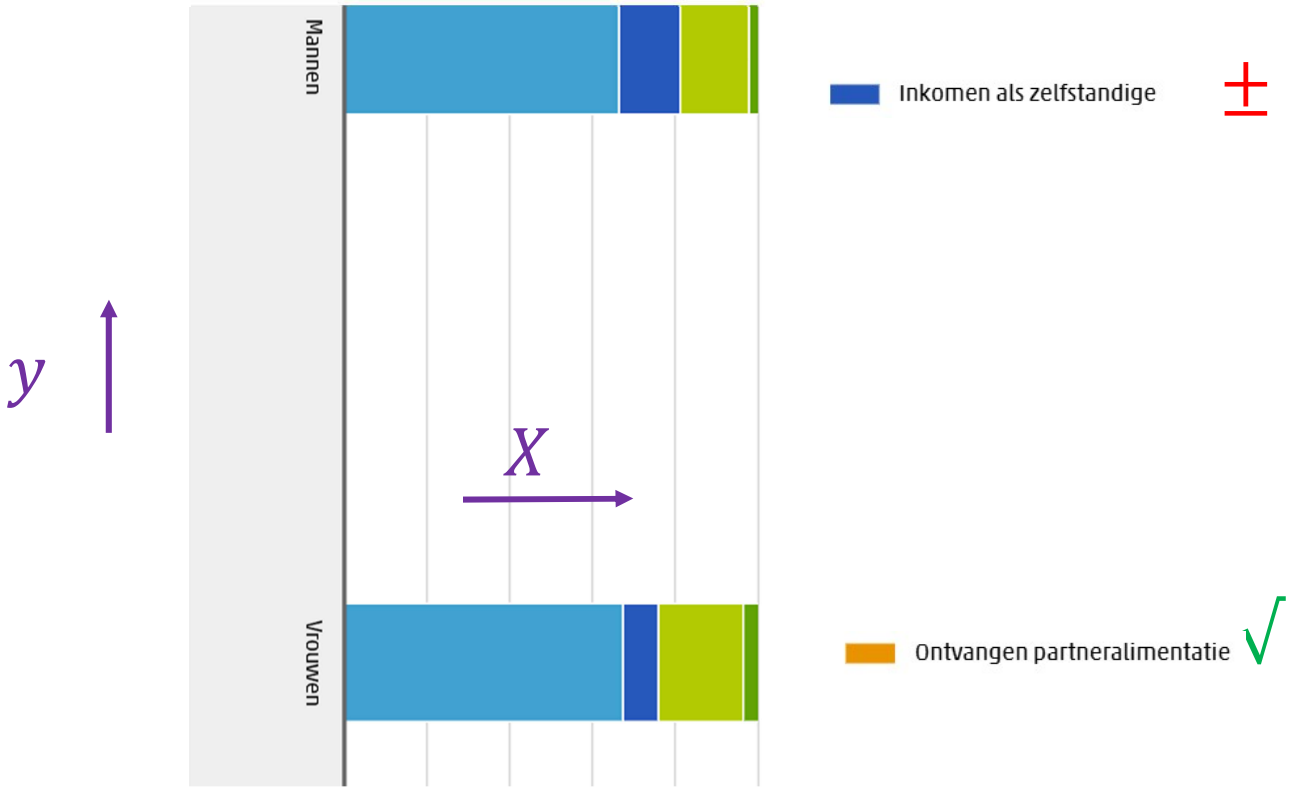
Classification $y = f(X)$ with $y \equiv$ vector of classes

no ordering, just index

Supervised learning

scikit learn

• Model.predict(X) $\longrightarrow$ y



Classification $y = f(X)$ with $y \equiv$ vector of classes

no ordering, just index

Classification

- k-Neighbours Classifier
- Logistic Regression
- Linear Support Vector Classifier
- Naive Bayes Classifier
- Decision Trees Classifier
- Random Forest Classifier
- Kernelized Support Vector Classifier
- Multi Layer Perceptron Classifier

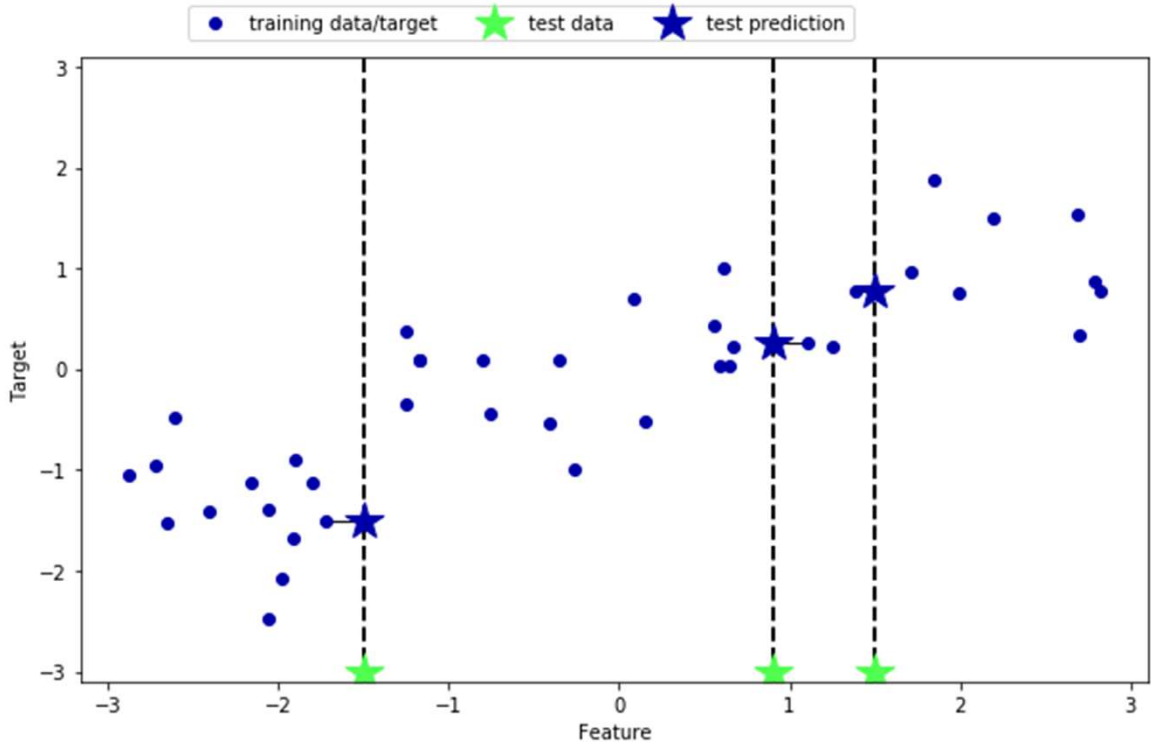
Classification

- k-Neighbours Classifier
- Logistic Regression
- Linear Support Vector Classifier
- Naive Bayes Classifier
- Decision Trees Classifier
- Random Forest Classifier
- Kernelized Support Vector Classifier
- Multi Layer Perceptron Classifier

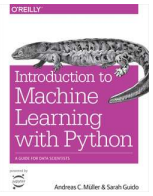
Previously

k-NN for Regression

- 1-neighbor example



Andreas C. Müller, Sarah Guido, Introduction to Machine Learning with Python, O'Reilly Media, October 2016

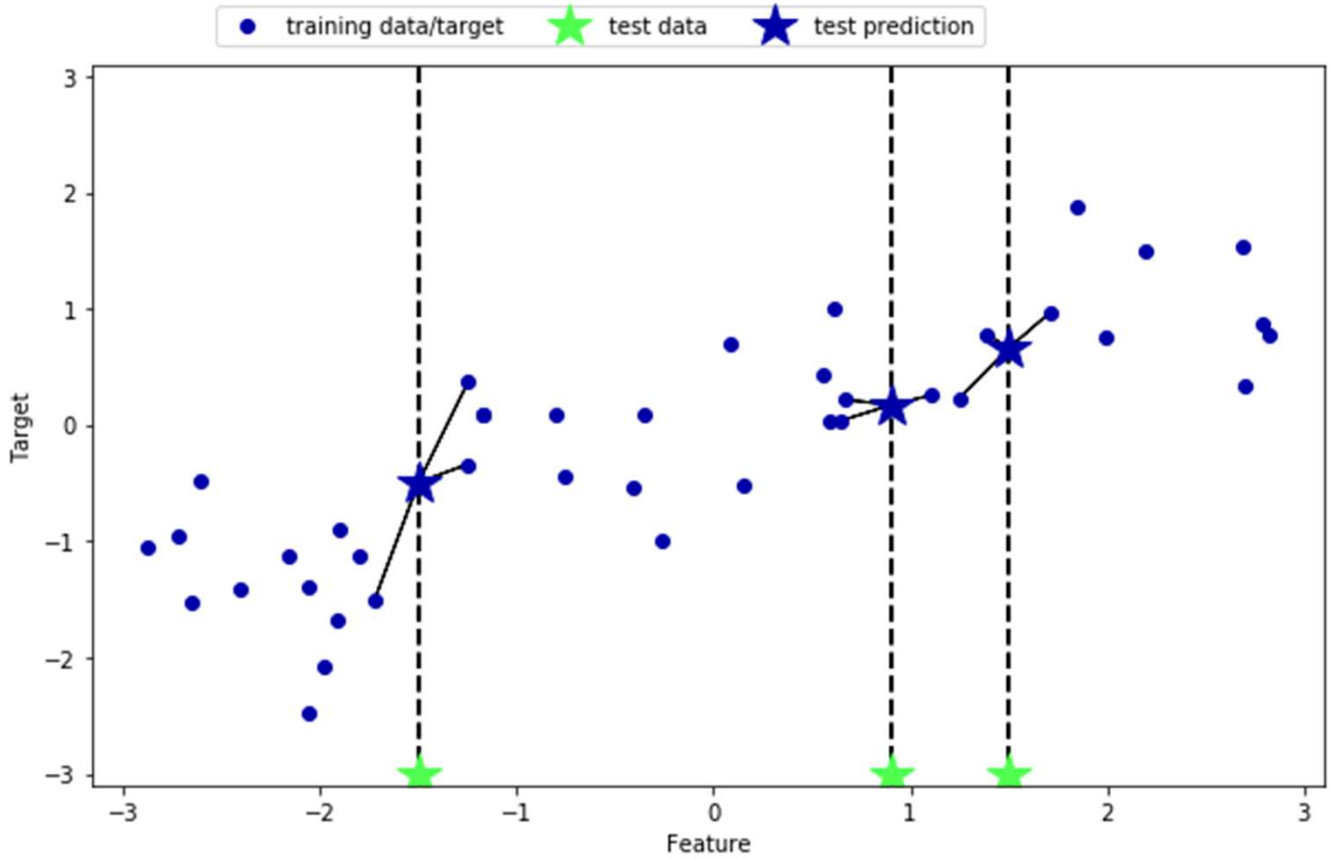


Page 42-46

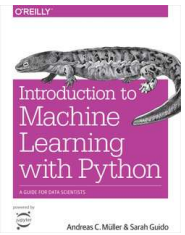
Previously

k-NN for Regression

- 3-neighbor example



Andreas C. Müller, Sarah Guido, Introduction to Machine Learning with Python, O'Reilly Media, October 2016

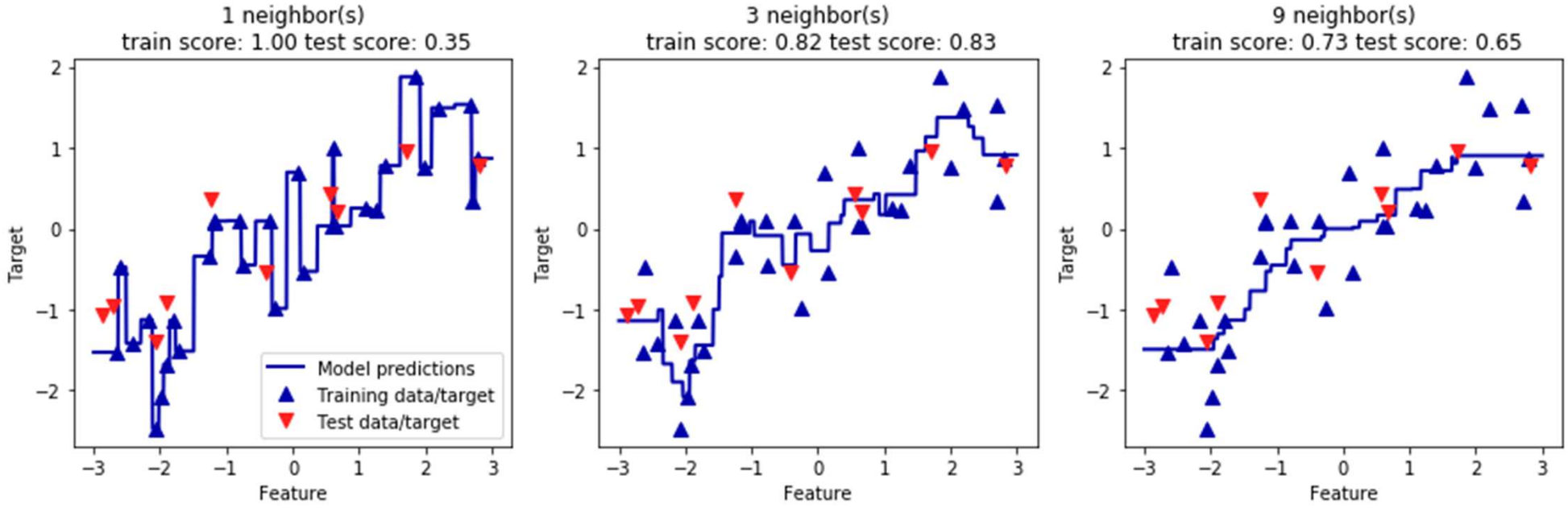


Page 42-46

Previously

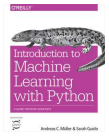
k-NN for Regression

- Fitting a ‘line’



30 training points → averaging over 9 neighbors
→ three independent values for a line

Andreas C. Müller, Sarah Guido, Introduction to Machine Learning with Python, O’Reilly Media, October 2016



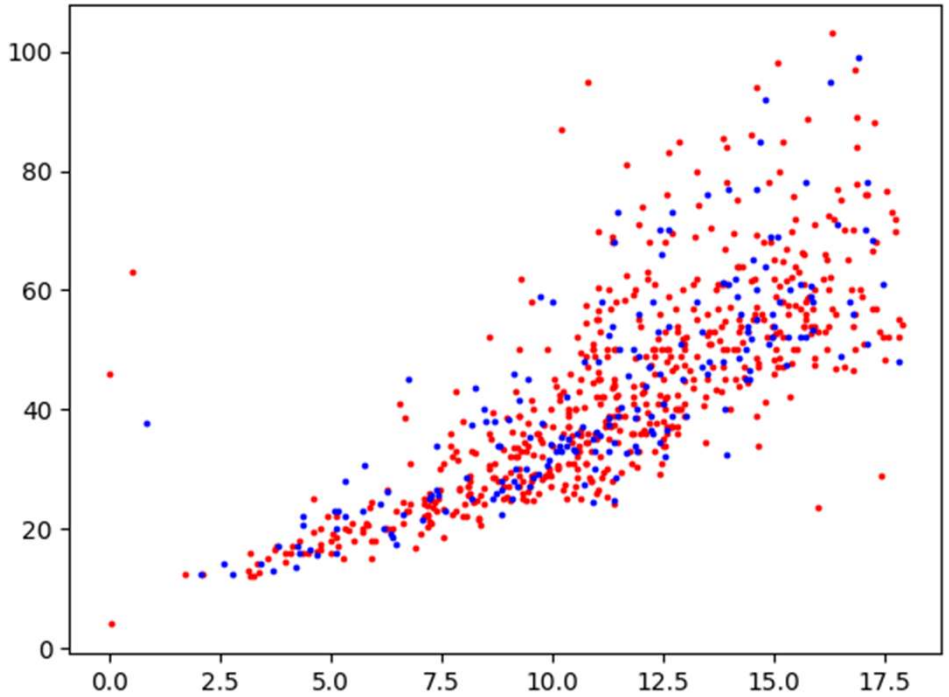
Page 42-46

Simple model – version 1-NN

$$weight = neighbor\_weight()$$

Model check

regensburg_pediatric_appendicitis

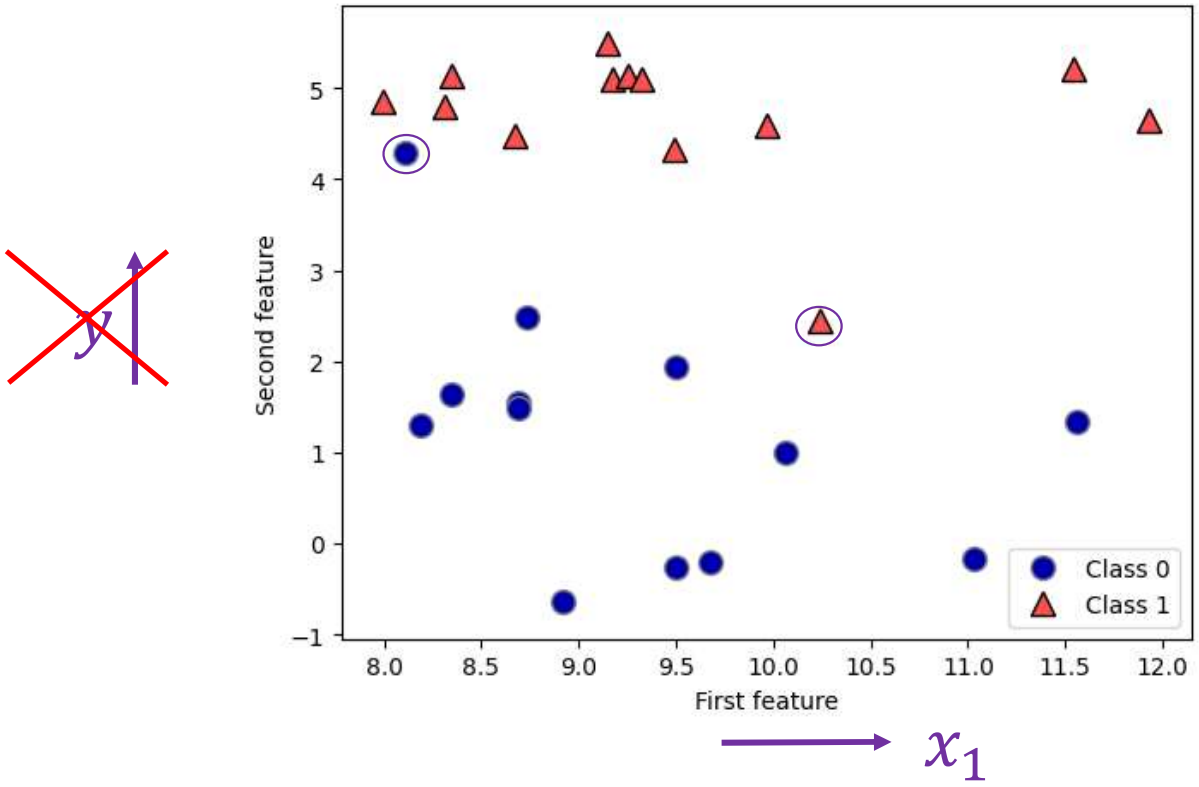


- 775 datapoints → 584 training values & 195 test values

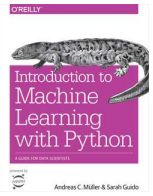
Back to Textbook

k-NN for Classification

- Artificial forge-dataset
- (2 clusters of datapoints, with two outliers)



Andreas C. Müller, Sarah Guido, Introduction to Machine Learning with Python, O'Reilly Media, October 2016

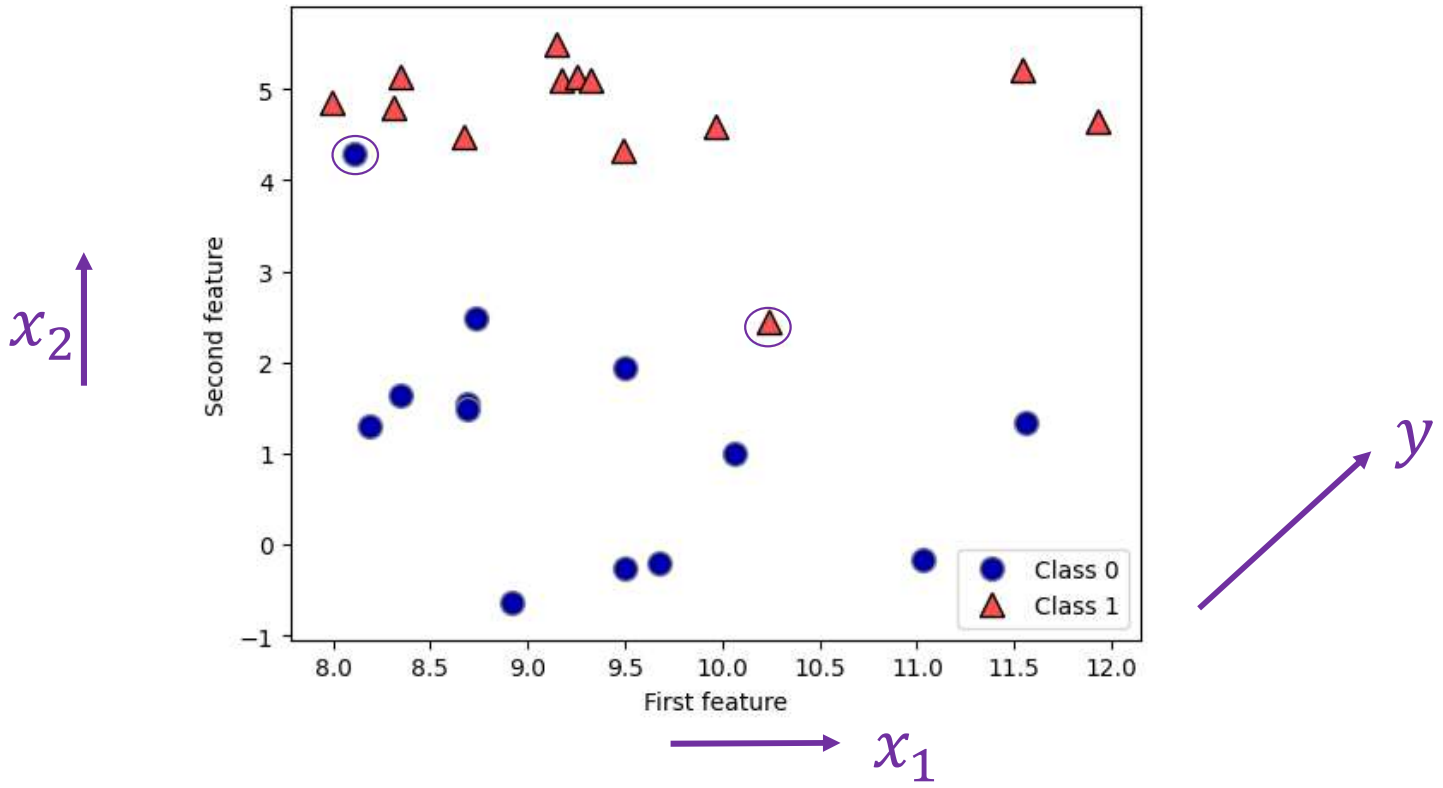


Page 33

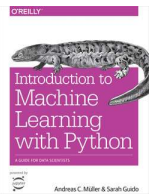
Back to Textbook

k-NN for Classification

- Artificial forge-dataset
- (2 clusters of datapoints, with two outliers)



Andreas C. Müller, Sarah Guido, Introduction to Machine Learning with Python, O'Reilly Media, October 2016

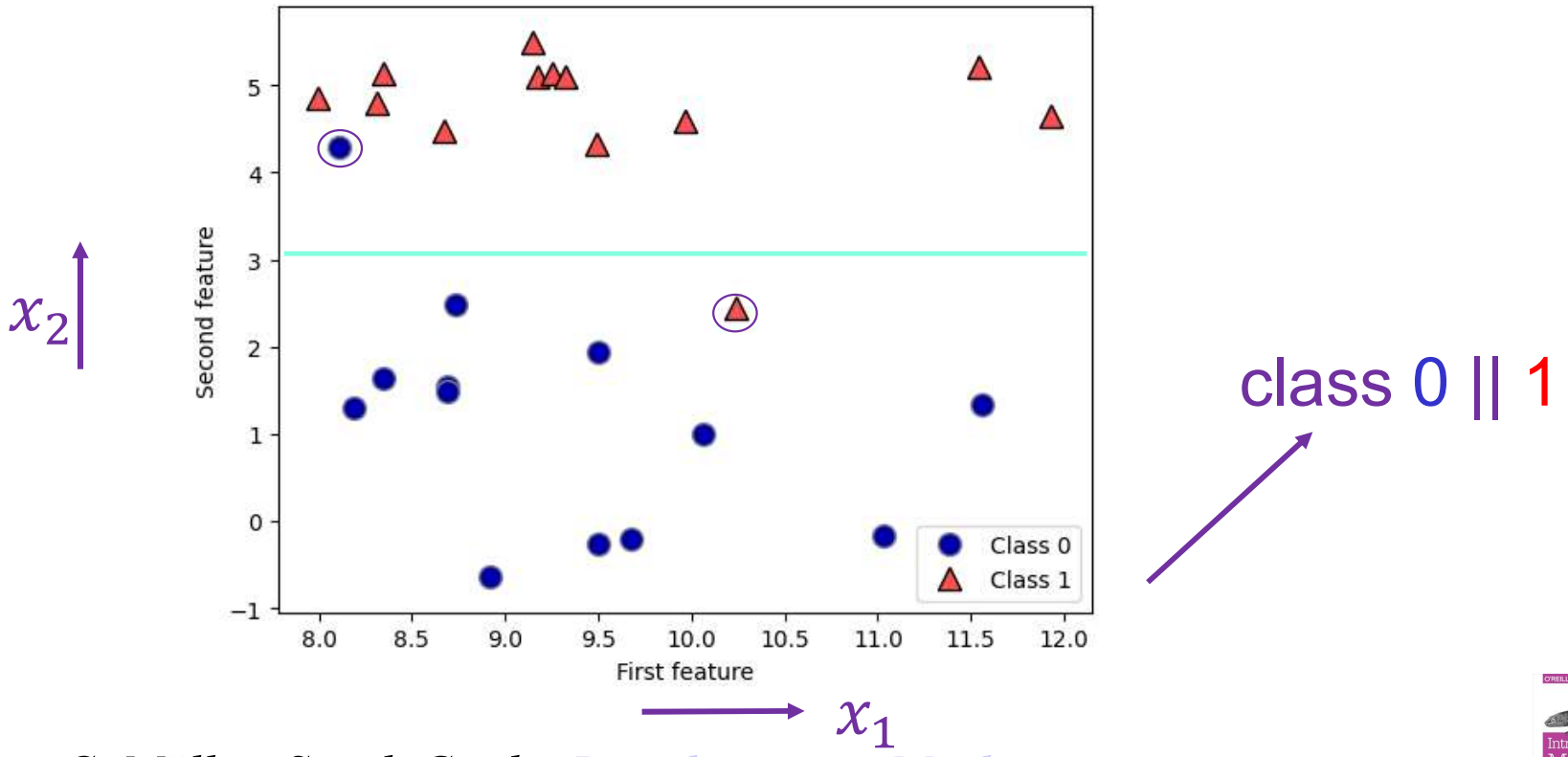


Page 33

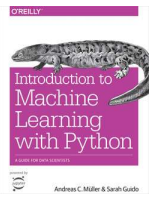
Back to Textbook

k-NN for Classification

- Artificial forge-dataset
- (2 clusters of datapoints, with two outliers)



Andreas C. Müller, Sarah Guido, Introduction to Machine Learning with Python, O'Reilly Media, October 2016

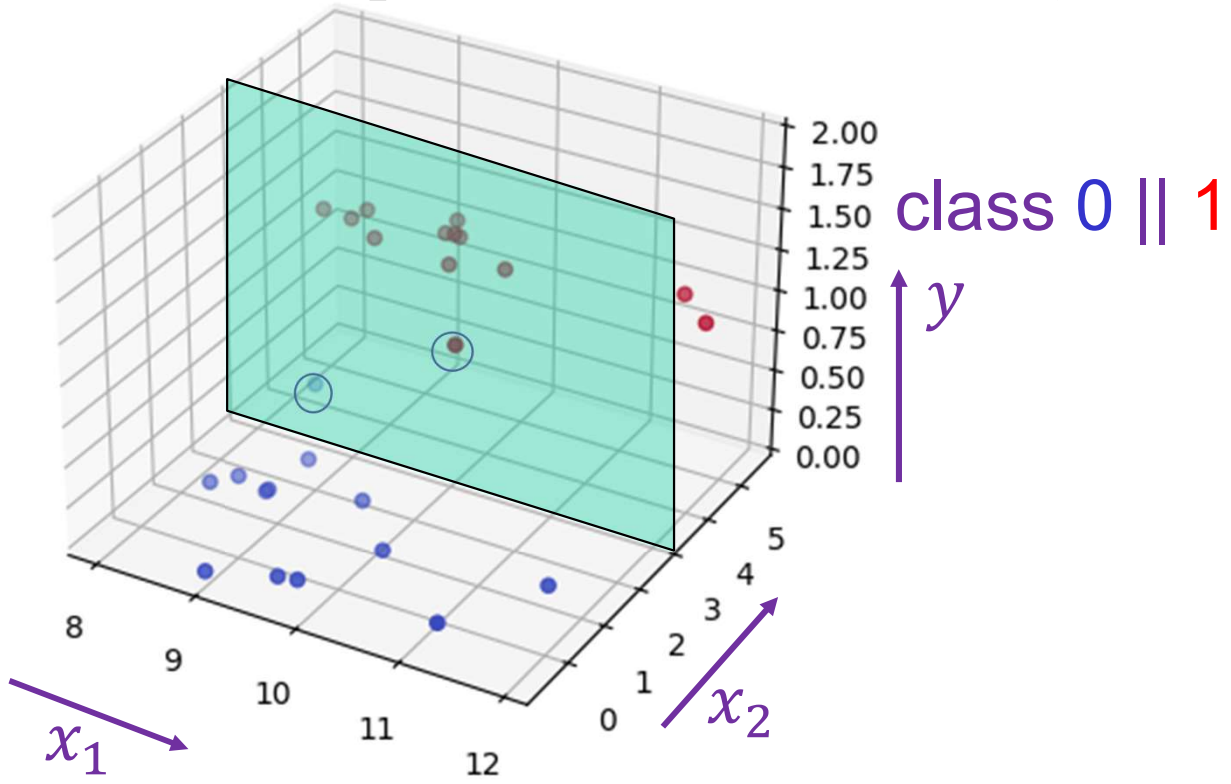


Page 33

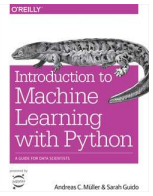
Back to Textbook

k-NN for Classification

- Artificial forge-dataset
- (2 clusters of datapoints, with two outliers)



Andreas C. Müller, Sarah Guido, Introduction to Machine Learning with Python, O'Reilly Media, October 2016

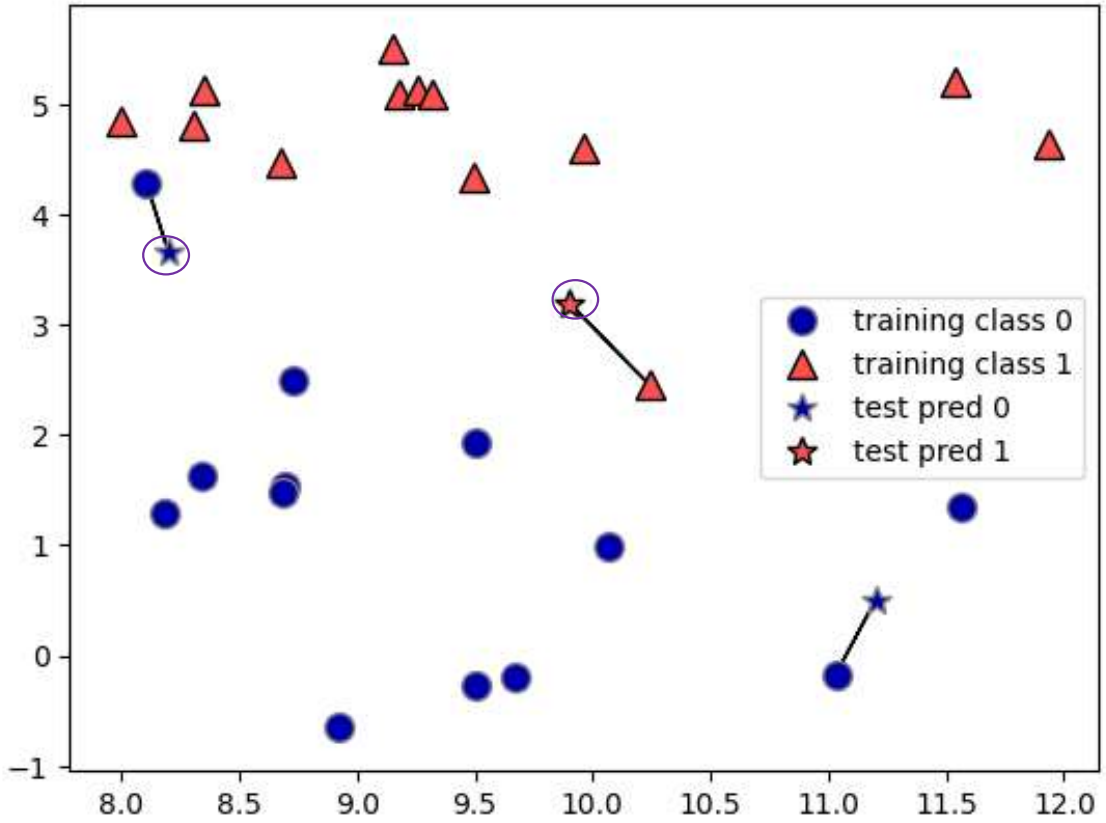


Page 33

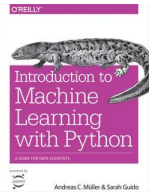
Back to Textbook

k-NN for Classification

- One-nearest-neighbor



Andreas C. Müller, Sarah Guido, Introduction to Machine Learning with Python, O'Reilly Media, October 2016

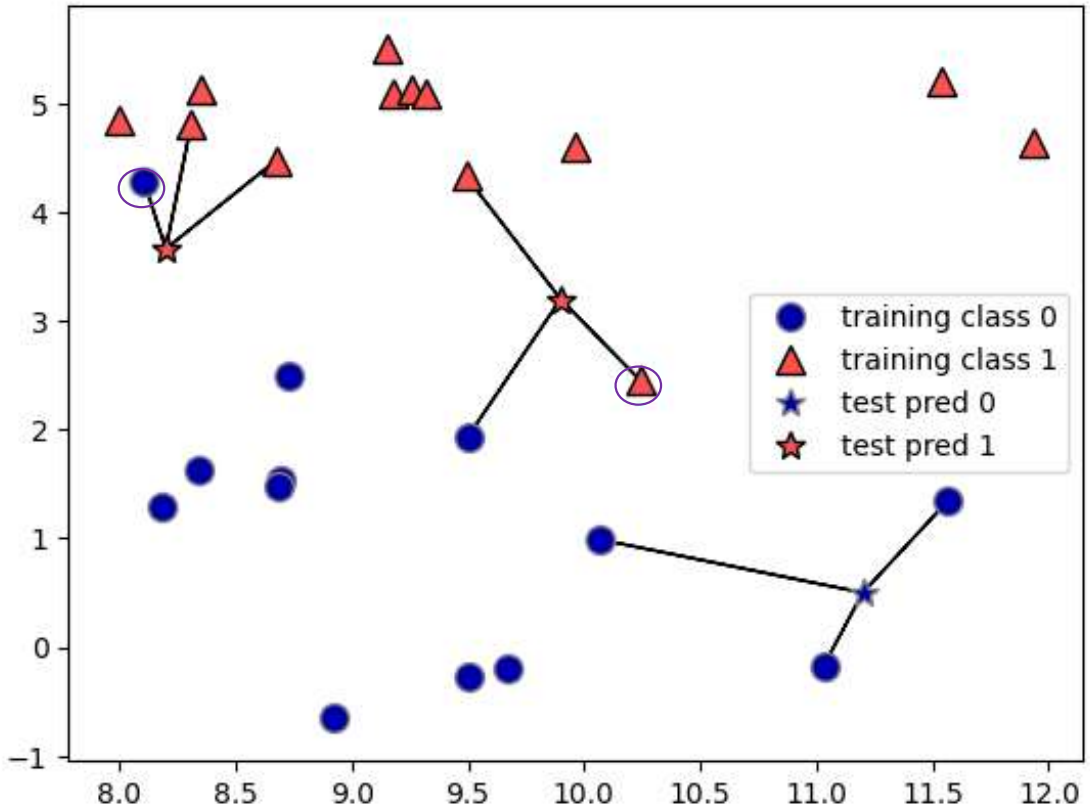


Section 2.3.2

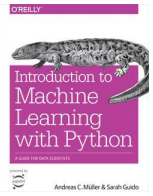
Back to Textbook

k-NN for Classification

- Three-nearest-neighbors



Andreas C. Müller, Sarah Guido, Introduction to Machine Learning with Python, O'Reilly Media, October 2016

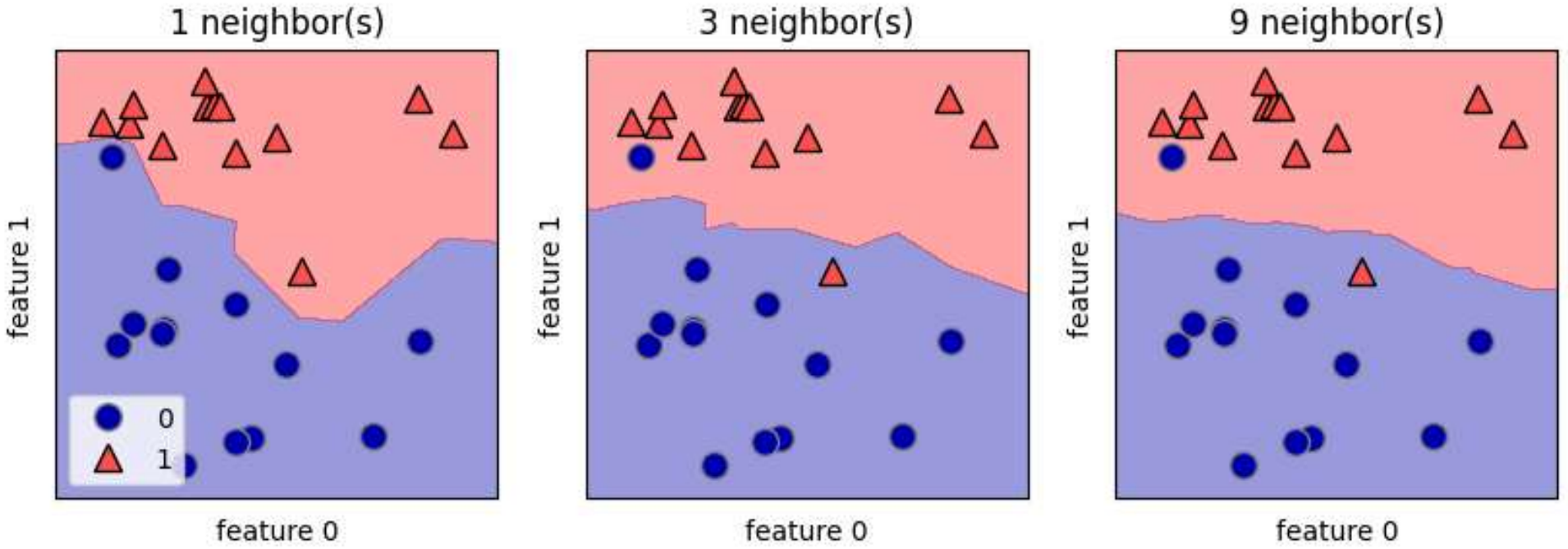


Page 38

Back to Textbook

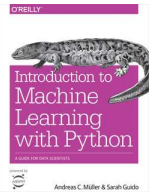
k-NN for Classification

- Decision-boundaries



Plot for all datapoints (no training_test_split)

Andreas C. Müller, Sarah Guido, Introduction to Machine Learning with Python, O'Reilly Media, October 2016

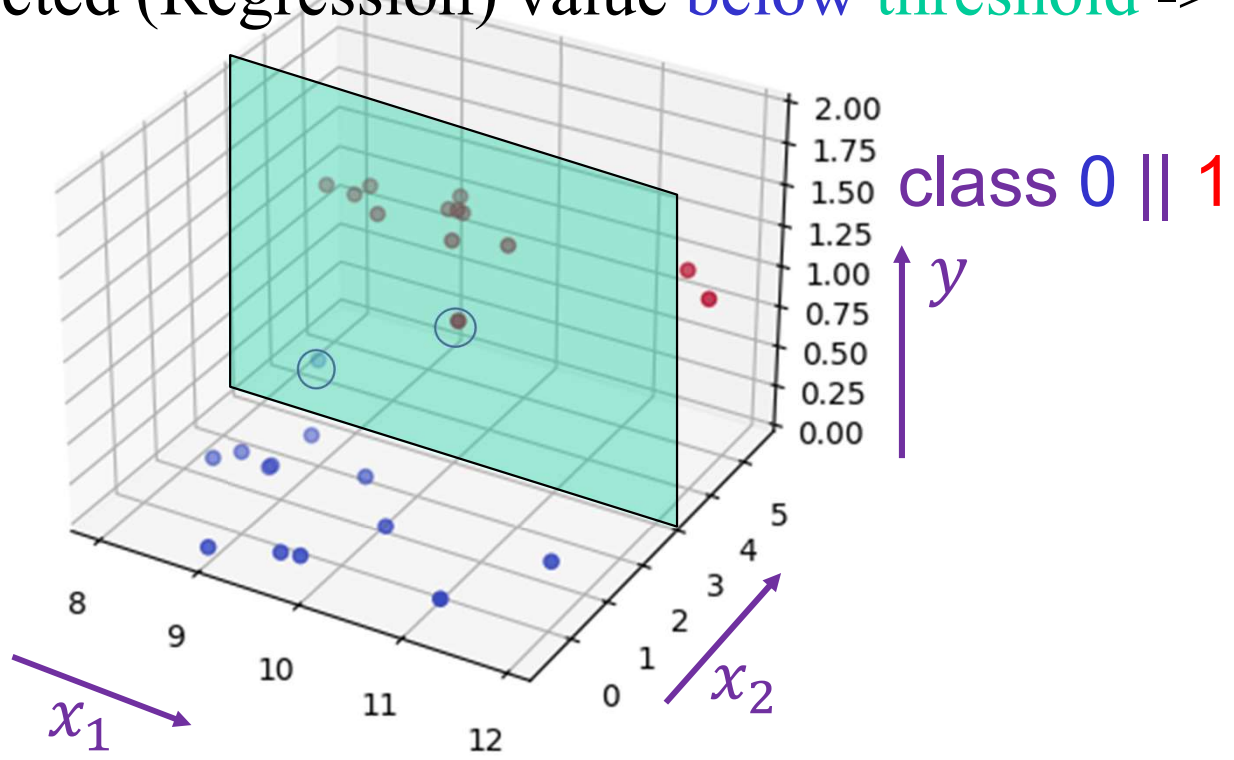


Page 40

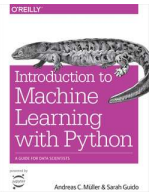
Back to Textbook

Classification as binary decision

- Predicted (Regression) value above threshold -> class 1
- Predicted (Regression) value below threshold -> class 0



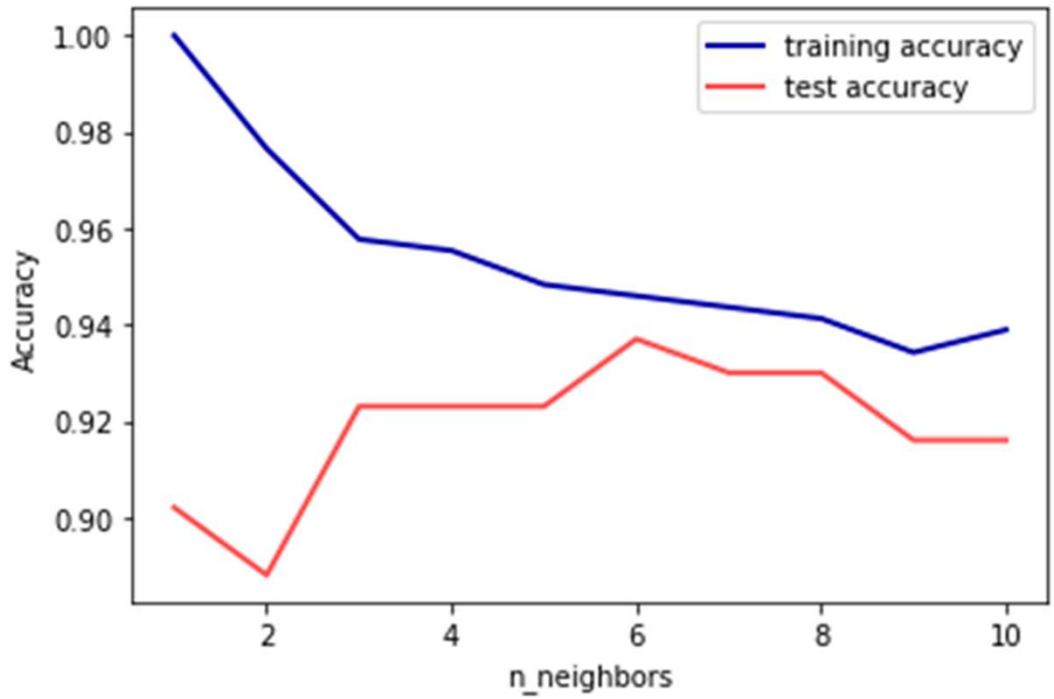
Andreas C. Müller, Sarah Guido, Introduction to Machine Learning with Python, O'Reilly Media, October 2016



Back to Textbook

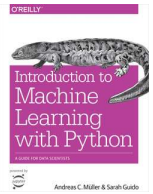
k-NN for Classification

- Hyperparameter tuning of # neighbors



Tuned on a larger and more realistic dataset (breast_cancer)

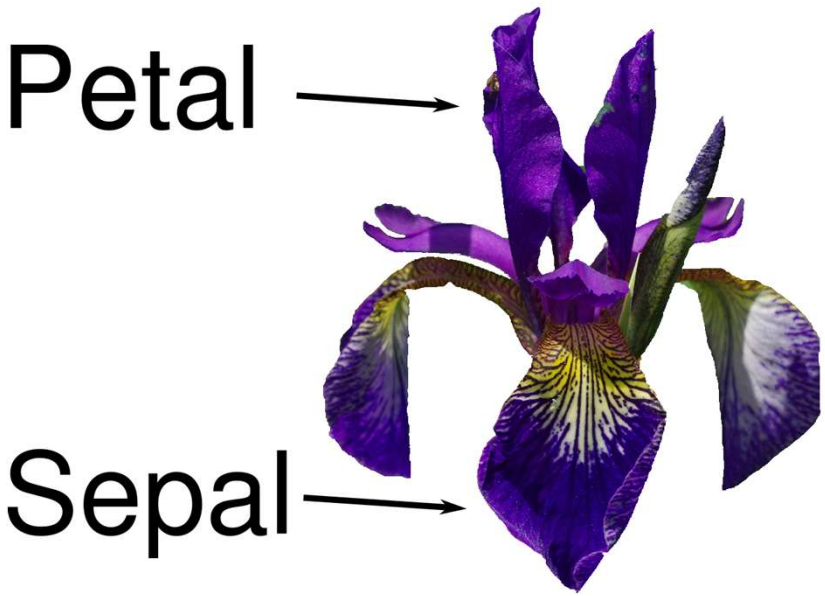
Andreas C. Müller, Sarah Guido, Introduction to Machine Learning with Python, O'Reilly Media, October 2016



Page 42

Realistic dataset

- Iris dataset



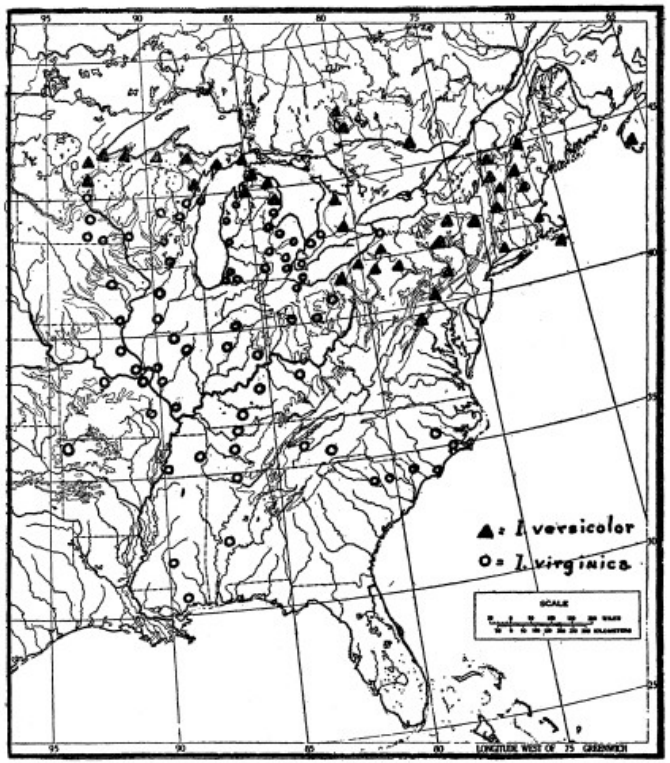
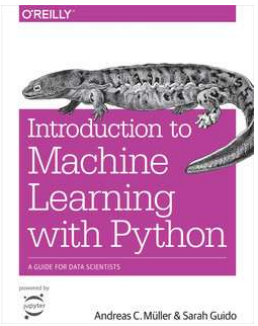


Fig. 1. Localities at which *I. versicolor* and *I. virginica* have been studied.



Anderson, E. (1928) The problem of species in the northern blue flags, Iris versicolor L. and Iris virginica L. Annals of the Missouri Botanical Garden, 15(3), 241–332.

Realistic dataset

- Iris dataset

```
from sklearn.datasets import load_iris
iris_dataset = load_iris()

print("Feature names:\n", iris_dataset['feature_names'])
```

```
['sepal length (cm)', 'sepal width (cm)',
'petal length (cm)', 'petal width (cm)']
```

Table I

Iris setosa				*Iris versicolor*				*Iris virginica*			
Sepal length	Sepal width	Petal length	Petal width	Sepal length	Sepal width	Petal length	Petal width	Sepal length	Sepal width	Petal length	Petal width
5·1	3·5	1·4	0·2	7·0	3·2	4·7	1·4	6·3	3·3	6·0	2·5
4·9	3·0	1·4	0·2	6·4	3·2	4·5	1·5	5·8	2·7	5·1	1·9
4·7	3·2	1·3	0·2	6·9	3·1	4·9	1·5	7·1	3·0	5·9	2·1
4·6	3·1	1·5	0·2	5·5	2·3	4·0	1·3	6·3	2·9	5·6	1·8
5·0	3·6	1·4	0·2	6·5	2·8	4·6	1·5	6·5	3·0	5·8	2·2
5·4	3·9	1·7	0·4	5·7	2·8	4·5	1·3	7·6	3·0	6·6	2·1
4·6	3·4	1·4	0·3	6·3	3·3	4·7	1·6	4·9	2·5	4·5	1·7
5·0	3·4	1·5	0·2	4·9	2·4	3·3	1·0	7·3	2·9	6·3	1·8
4·4	2·9	1·4	0·2	6·6	2·9	4·6	1·3	6·7	2·5	5·8	1·8
4·9	3·1	1·5	0·1	5·2	2·7	3·9	1·4	7·2	3·6	6·1	2·5
5·4	3·7	1·5	0·2	5·0	2·0	3·5	1·0	6·5	3·2	5·1	2·0
4·8	3·4	1·6	0·2	5·9	3·0	4·2	1·5	6·4	2·7	5·3	1·9
4·8	3·0	1·4	0·1	6·0	2·2	4·0	1·0	6·8	3·0	5·5	2·1
4·3	3·0	1·1	0·1	6·1	2·9	4·7	1·4	5·7	2·5	5·0	2·0
5·8	4·0	1·2	0·2	5·6	2·9	3·6	1·3	5·8	2·8	5·1	2·4
5·7	4·4	1·5	0·4	6·7	3·1	4·4	1·4	6·4	3·2	5·3	2·3
5·4	3·9	1·3	0·4	5·6	3·0	4·5	1·5	6·5	3·0	5·5	1·8
5·1	3·5	1·4	0·3	5·8	2·7	4·1	1·0	7·7	3·8	6·7	2·2
5·7	3·8	1·7	0·3	6·2	2·2	4·5	1·5	7·7	2·6	6·9	2·3
5·1	3·8	1·5	0·3	5·6	2·5	3·9	1·1	6·0	2·2	5·0	1·5
5·4	3·4	1·7	0·2	5·9	3·2	4·8	1·8	6·9	3·2	5·7	2·3
5·1	3·7	1·5	0·4	6·1	2·8	4·0	1·3	5·6	2·8	4·9	2·0
4·6	3·6	1·0	0·2	6·3	2·5	4·9	1·5	7·7	2·8	6·7	2·0
5·1	3·3	1·7	0·5	6·1	2·8	4·7	1·2	6·3	2·7	4·9	1·8
4·8	3·4	1·9	0·2	6·4	2·9	4·3	1·3	6·7	3·3	5·7	2·1
5·0	3·0	1·6	0·2	6·6	3·0	4·4	1·4	7·2	3·2	6·0	1·8
5·0	3·4	1·6	0·4	6·8	2·8	4·8	1·4	6·2	2·8	4·8	1·8
5·2	3·5	1·5	0·2	6·7	3·0	5·0	1·7	6·1	3·0	4·9	1·8
5·2	3·4	1·4	0·2	6·0	2·9	4·5	1·5	6·4	2·8	5·6	2·1
4·7	3·2	1·6	0·2	5·7	2·6	3·5	1·0	7·2	3·0	5·8	1·6
4·8	3·1	1·6	0·2	5·5	2·4	3·8	1·1	7·4	2·8	6·1	1·9
5·4	3·4	1·5	0·4	5·5	2·4	3·7	1·0	7·9	3·8	6·4	2·0
5·2	4·1	1·5	0·1	5·8	2·7	3·9	1·2	6·4	2·8	5·6	2·2
5·5	4·2	1·4	0·2	6·0	2·7	5·1	1·6	6·3	2·8	5·1	1·5
4·9	3·1	1·5	0·2	5·4	3·0	4·5	1·5	6·1	2·6	5·6	1·4
5·0	3·2	1·2	0·2	6·0	3·4	4·5	1·6	7·7	3·0	6·1	2·3
5·5	3·5	1·3	0·2	6·7	3·1	4·7	1·5	6·3	3·4	5·6	2·4
4·9	3·6	1·4	0·1	6·3	2·3	4·4	1·3	6·4	3·1	5·5	1·8
4·4	3·0	1·3	0·2	5·6	3·0	4·1	1·3	6·0	3·0	4·8	1·8
5·1	3·4	1·5	0·2	5·5	2·5	4·0	1·3	6·9	3·1	5·4	2·1
5·0	3·5	1·3	0·3	5·5	2·6	4·4	1·2	6·7	3·1	5·6	2·4
4·5	2·3	1·3	0·3	6·1	3·0	4·6	1·4	6·9	3·1	5·1	2·3
4·4	3·2	1·3	0·2	5·8	2·6	4·0	1·2	5·8	2·7	5·1	1·9
5·0	3·5	1·6	0·6	5·0	2·3	3·3	1·0	6·8	3·2	5·9	2·3
5·1	3·8	1·9	0·4	5·6	2·7	4·2	1·3	6·7	3·3	5·7	2·5
4·8	3·0	1·4	0·3	5·7	3·0	4·2	1·2	6·7	3·0	5·2	2·3
5·1	3·8	1·6	0·2	5·7	2·9	4·2	1·3	6·3	2·5	5·0	1·9
4·6	3·2	1·4	0·2	6·2	2·9	4·3	1·3	6·5	3·0	5·2	2·0
5·3	3·7	1·5	0·2	5·1	2·5	3·0	1·1	6·2	3·4	5·4	2·3
5·0	3·3	1·4	0·2	5·7	2·8	4·1	1·3	5·9	3·0	5·1	1·8

Fisher, R.A. "The use of multiple measurements in taxonomic problems",
Annual Eugenics, 7, Part II, 179-188 (1936).

Realistic dataset

- Iris dataset

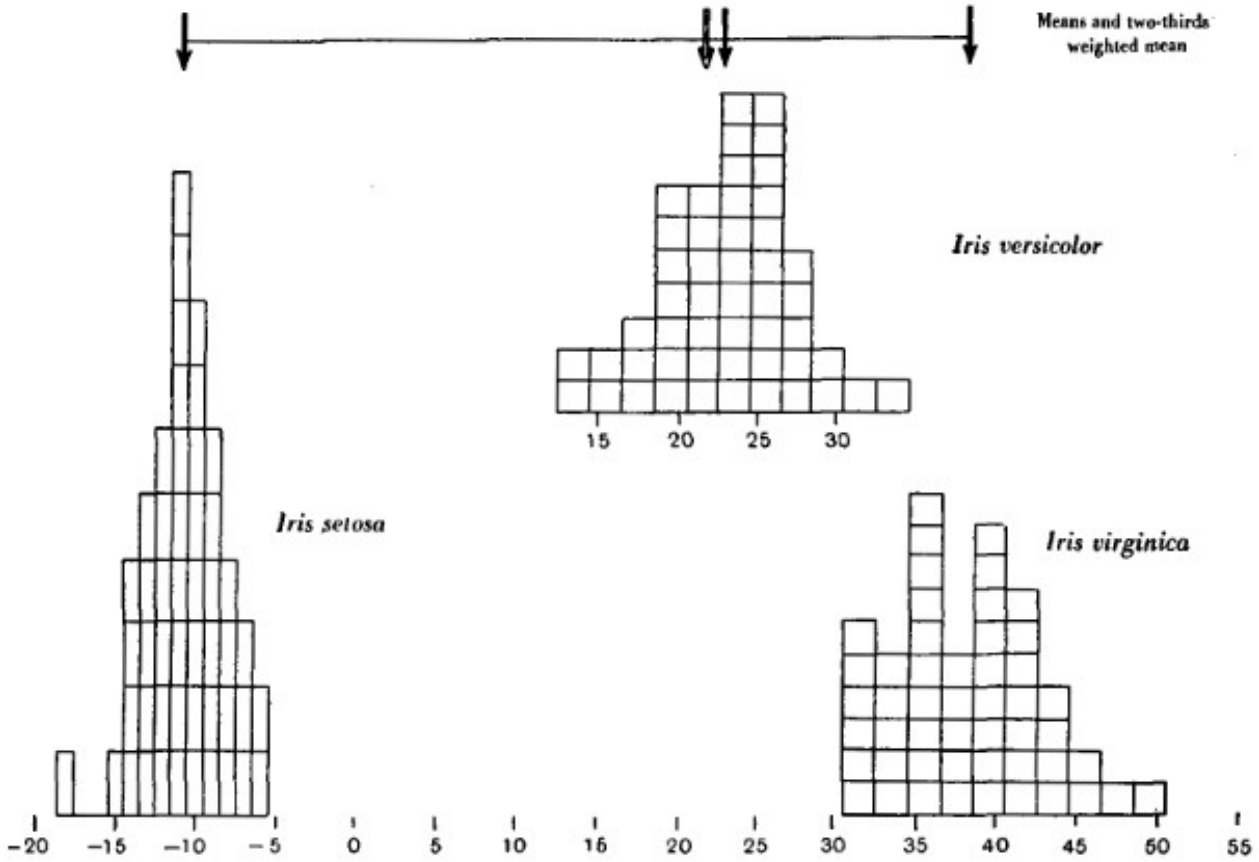
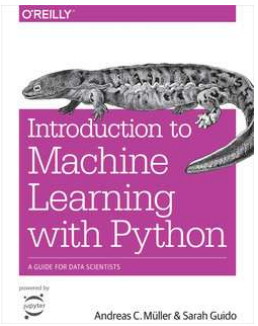


Fig. 1. Frequency histograms of the discriminating linear function, for three species of *Iris*.

Fisher, R.A. "The use of multiple measurements in taxonomic problems", Annual Eugenics, 7, Part II, 179-188 (1936).



Realistic dataset

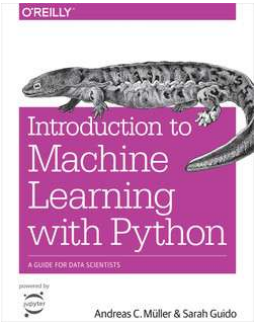
- Iris dataset



Iris Setosa, Versicolor and Virginica

```
from sklearn.datasets import load_iris
iris_dataset = load_iris()

print("Target names:", iris_dataset['target_names'])
```

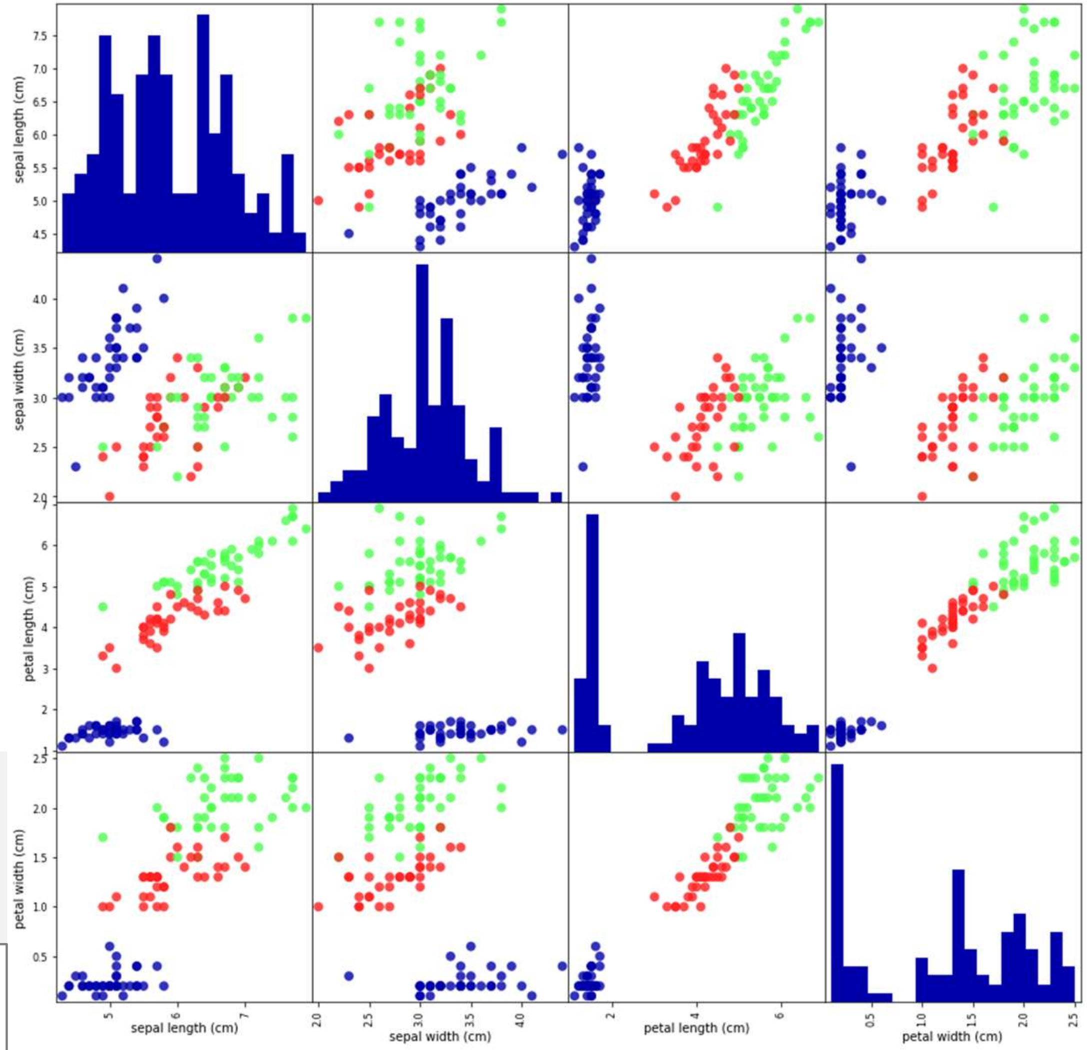
```
Target names: ['setosa' 'versicolor' 'virginica']
```



Antony Unwin, Kim Kleinman, The Iris Data Set: In Search of the Source of Virginica, Significance, Volume 18, Issue 6, December 2021, Pages 26–29

Realistic dataset

- Iris dataset



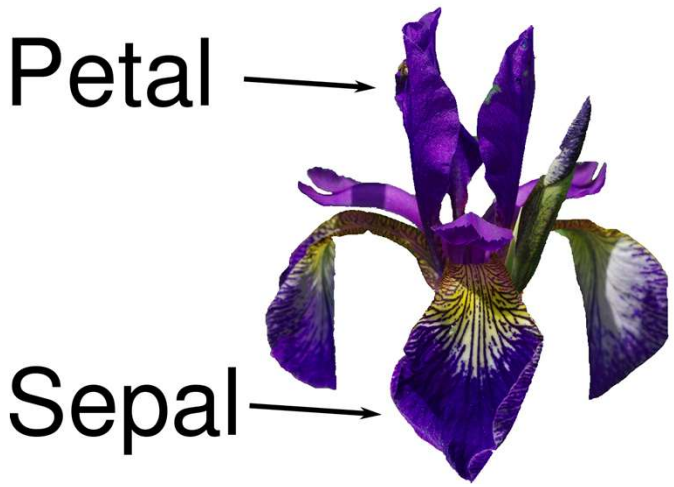
Iris Setosa, Versicolor and Virginica

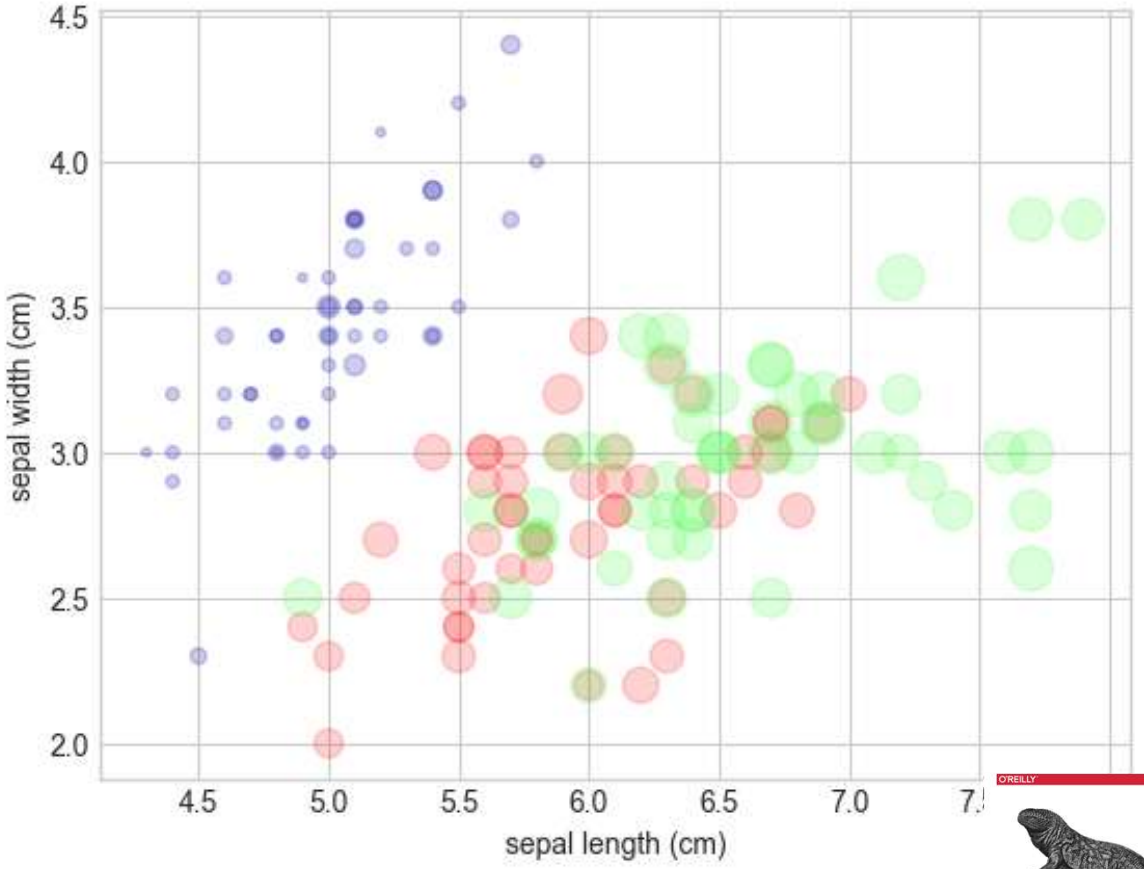
```
from sklearn.datasets import load_iris
iris_dataset = load_iris()

print("Target:\n", iris_dataset['target'])
```

```
Target:
[0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0  0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2
2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
2 2 2 2 2 2 2 2]
```

Realistic dataset

- Iris dataset






```
from matplotlib.colors import ListedColormap
cm3 = ListedColormap(['#0000aa', '#ff2020', '#50ff50'])

features = iris.data.T

plt.scatter(features[0], features[1], alpha=0.2,
            s=100*features[3], c=iris.target, cmap=cm3)
plt.xlabel(iris.feature_names[0])
plt.ylabel(iris.feature_names[1]);
```

Iris Setosa, Versicolor and Virginica

Realistic dataset

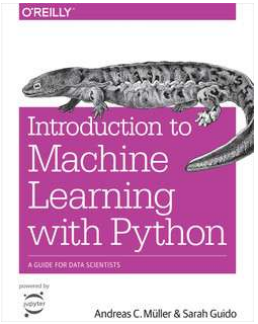
- Iris dataset



Iris Setosa, Versicolor and Virginica

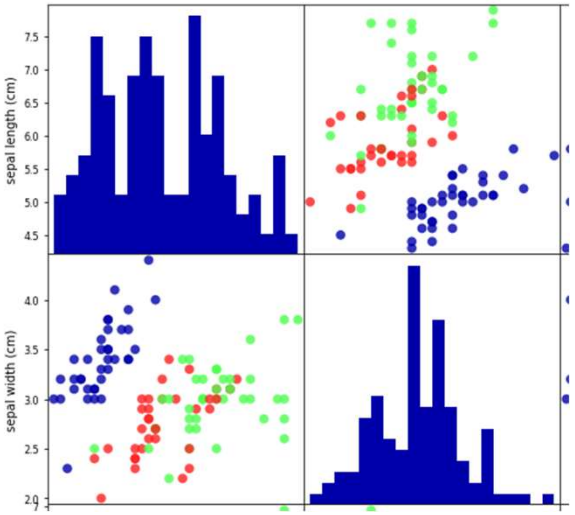
```
X_train, X_test, y_train, y_test = train_test_split(
    iris_dataset['data'], iris_dataset['target'], random_state=0)

knn = KNeighborsClassifier(n_neighbors=1)
knn.fit(X_train, y_train)

print("Test set score: {:.2f}".format(knn.score(X_test, y_test)))
```

```
Test set score: 0.97
```



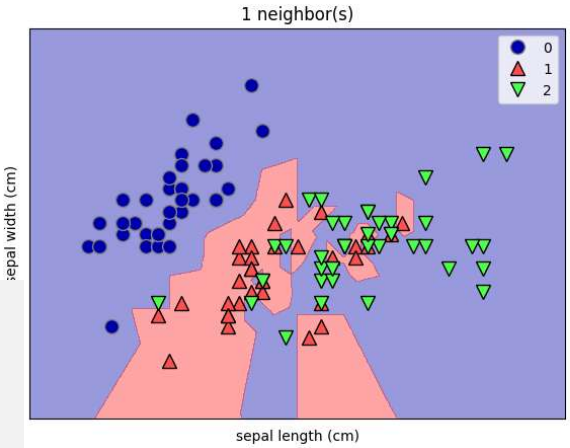
Realistic dataset

- Iris dataset

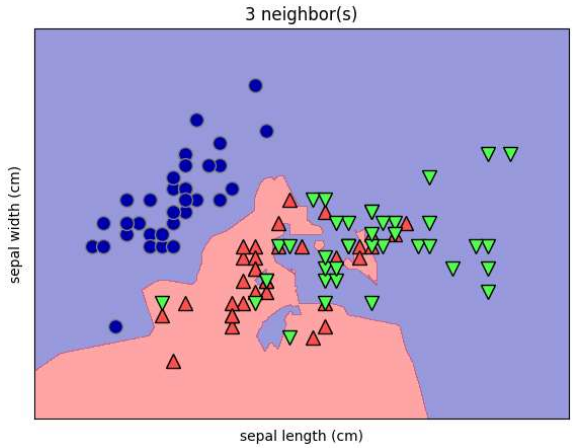


Iris Setosa, Versicolor and Virginica

```
knn = KNeighborsClassifier(n_neighbors=n_neighbors)
knn.fit(X_train[:,[0,1]], y_train)

accuracy = knn.score(X_test[:,[0,1]], y_test)))

print("Test set score: {:.2f}".format(accuracy))
```



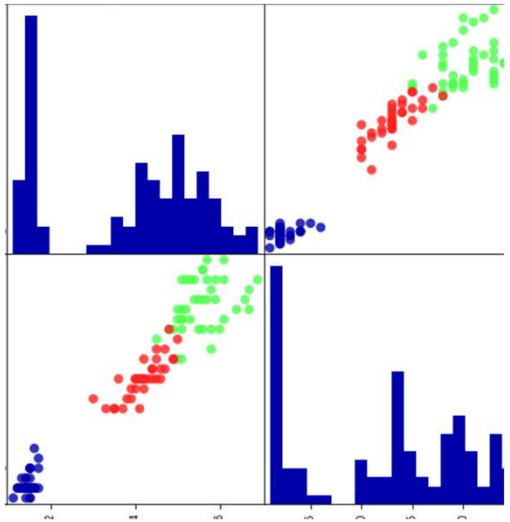
Test set score: 0.74



Test set score: 0.68

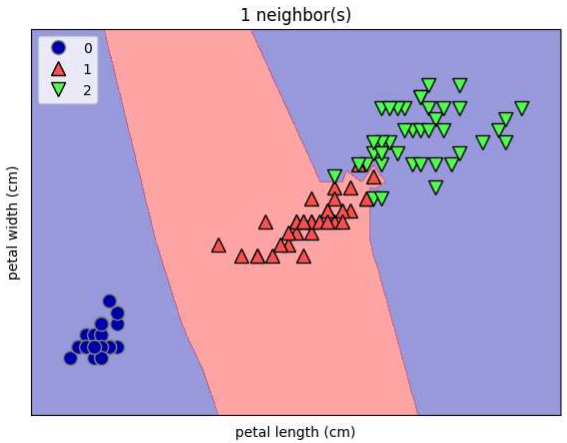
Realistic dataset

- Iris dataset

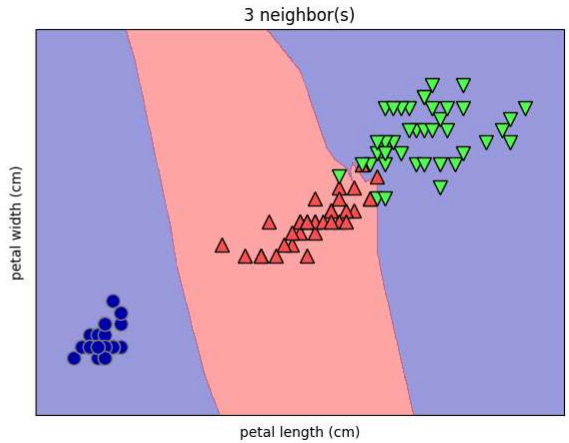


Iris Setosa, Versicolor and Virginica

```
knn = KNeighborsClassifier(n_neighbors=n_neighbors)
knn.fit(X_train[:,[2,3]], y_train)

accuracy = knn.score(X_test[:,[2,3]], y_test)))

print("Test set score: {:.2f}".format(accuracy))
```



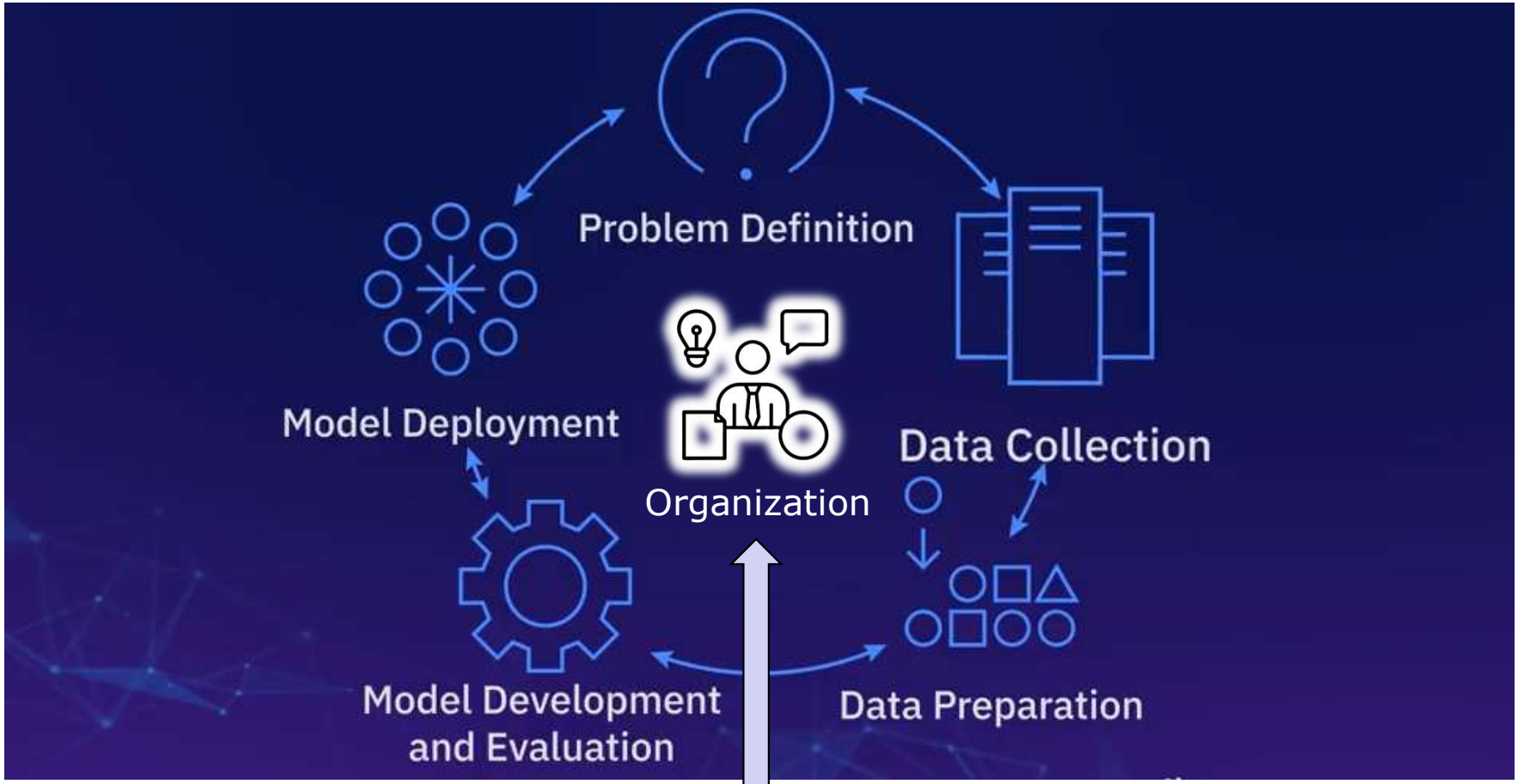
Test set score: 0.97



Test set score: 0.97

Introduction to Machine Learning

Building a predictive model is a circular process.

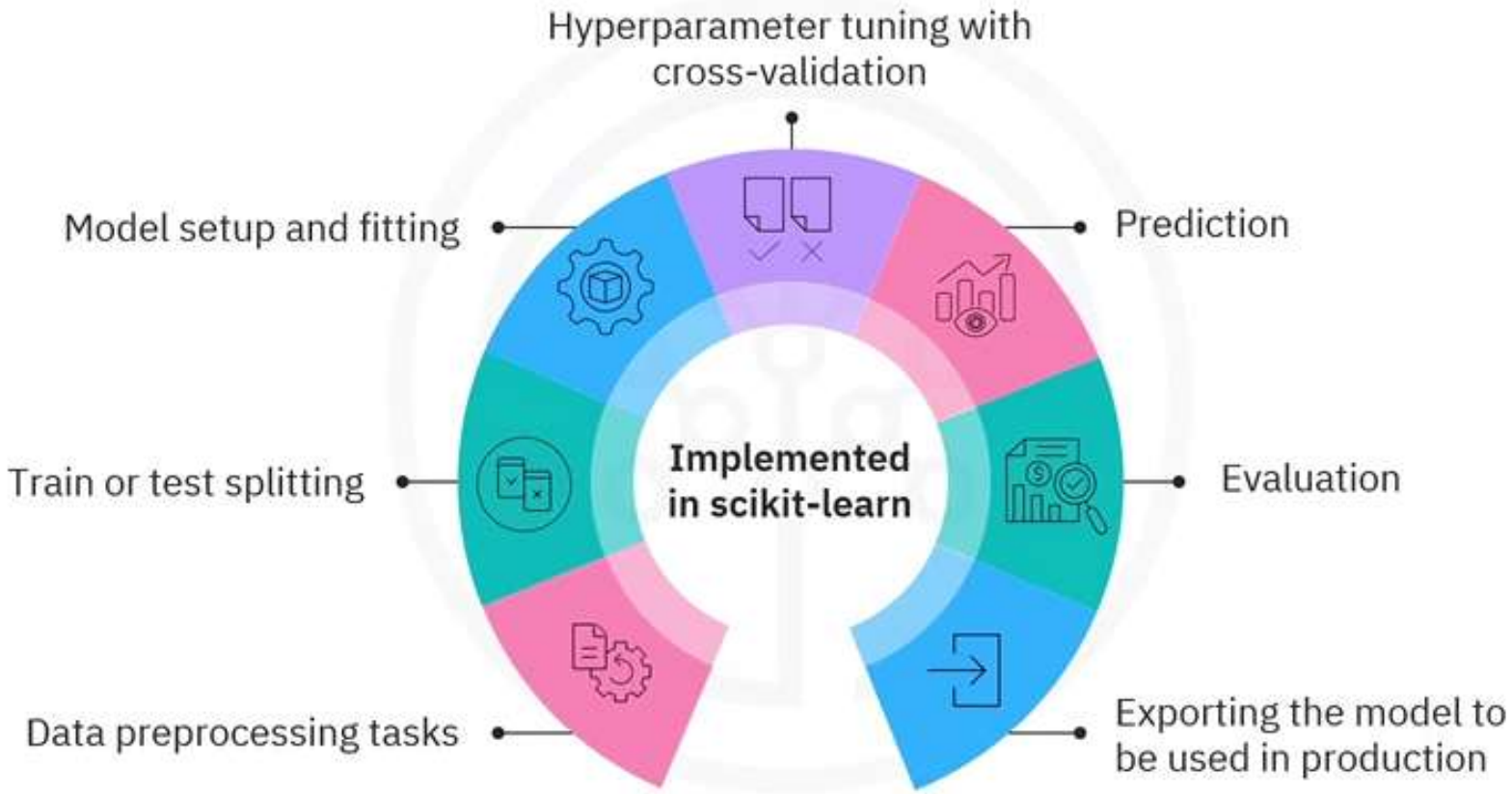


Expertise

- Depends on the organization
- Interpret the information in the right way

Introduction to Machine Learning

Building a predictive model is a circular process.



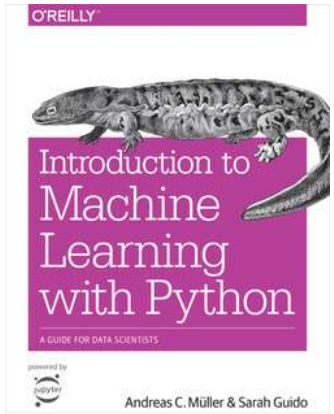
Joseph Santarcangelo & Joss Grossman, Machine Learning with Python, IBM Skills network, 2025

Introduction to Machine Learning

- 2.3.2 k-Nearest Neighbors

- Strengths:
  - Easy to understand
  - Main tuning parameter (# neighbours)
- Weakness:
  - Slow
  - Scale dependent (pre-processing required)
  - Bad for sparse-datasets

Andreas C. Müller, Sarah Guido, Introduction to Machine Learning with Python, O'Reilly Media, October 2016



Conclusion

Learning outcomes of this course covered today

- Many Regression algorithms have a Classification counterpart