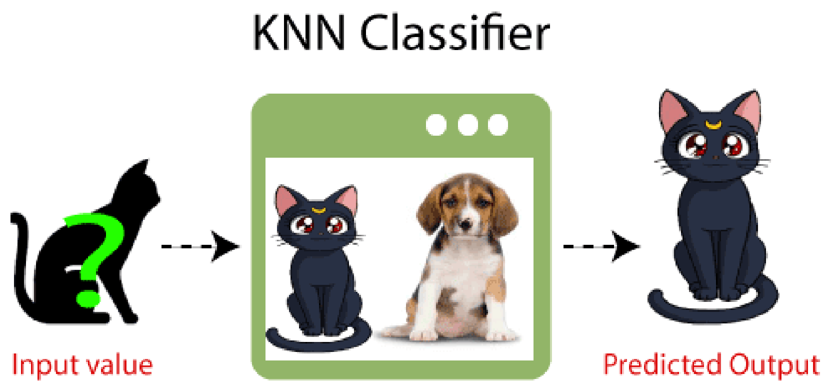
BM5702 MAKİNE ÖĞRENMESİNE GİRİŞ

Hafta 6

Doç. Dr. Murtaza CİCİOĞLU

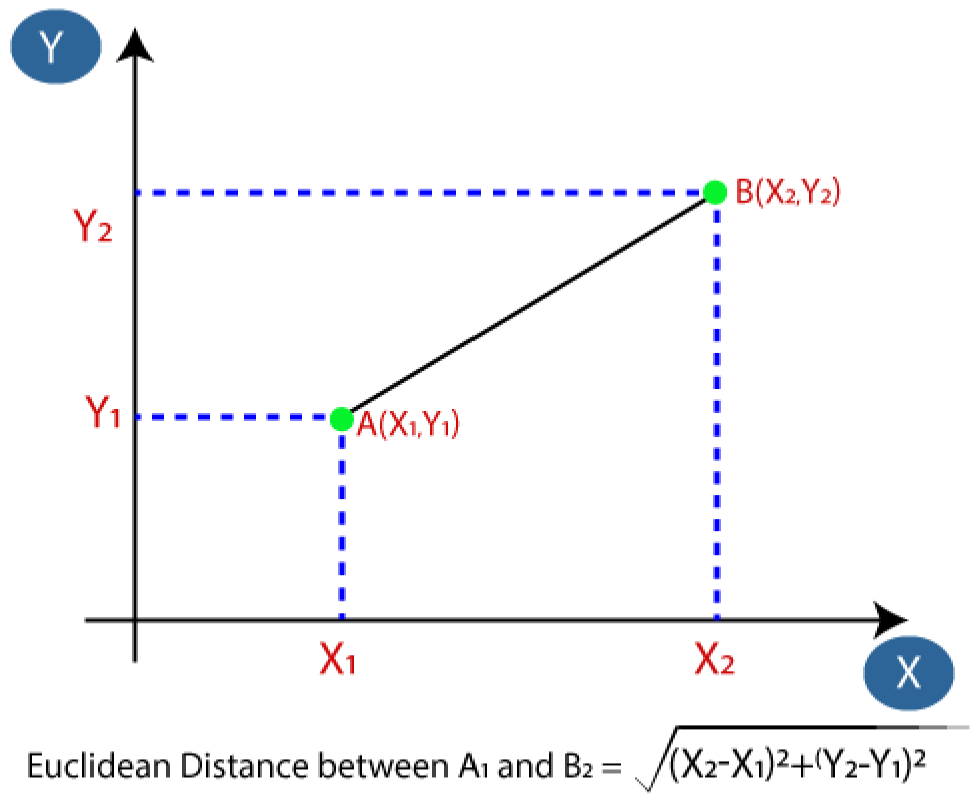
K-Nearest Neighbor

- K-Nearest Neighbour is one of the simplest Machine Learning algorithms based on Supervised Learning technique.
- K-NN is a non-parametric algorithm, which means it does not make any assumption on underlying data.
- It is also called a lazy learner algorithm because it does not learn from the training set immediately instead it stores the dataset and at the time of classification, it performs an action on the dataset.
- KNN algorithm at the training phase just stores the dataset and when it gets new data, then it classifies that data into a category that is much similar to the new data.



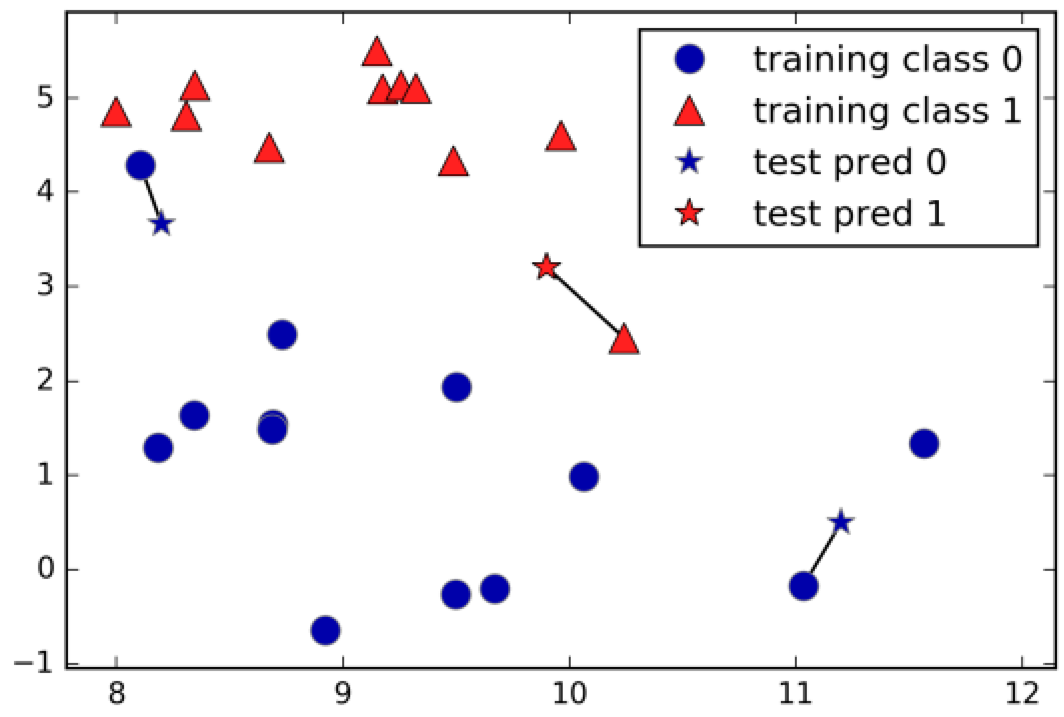
K-Nearest Neighbor

- The k-NN algorithm is arguably the simplest machine learning algorithm.
- Building the model consists only of storing the training dataset.
- To make a prediction for a new data point, the algorithm finds the closest data points in the training dataset—its “nearest neighbors.”



Euclidean Distance between A_1 and $B_2 = \sqrt{(X_2-X_1)^2+(Y_2-Y_1)^2}$

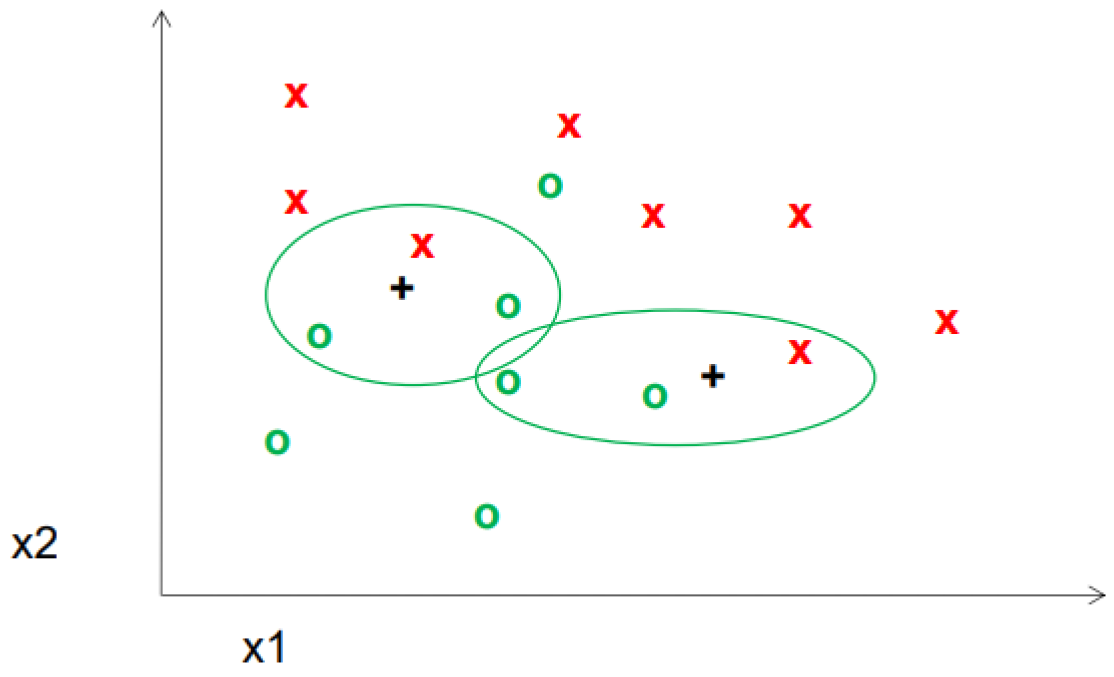
n_neighbors=1



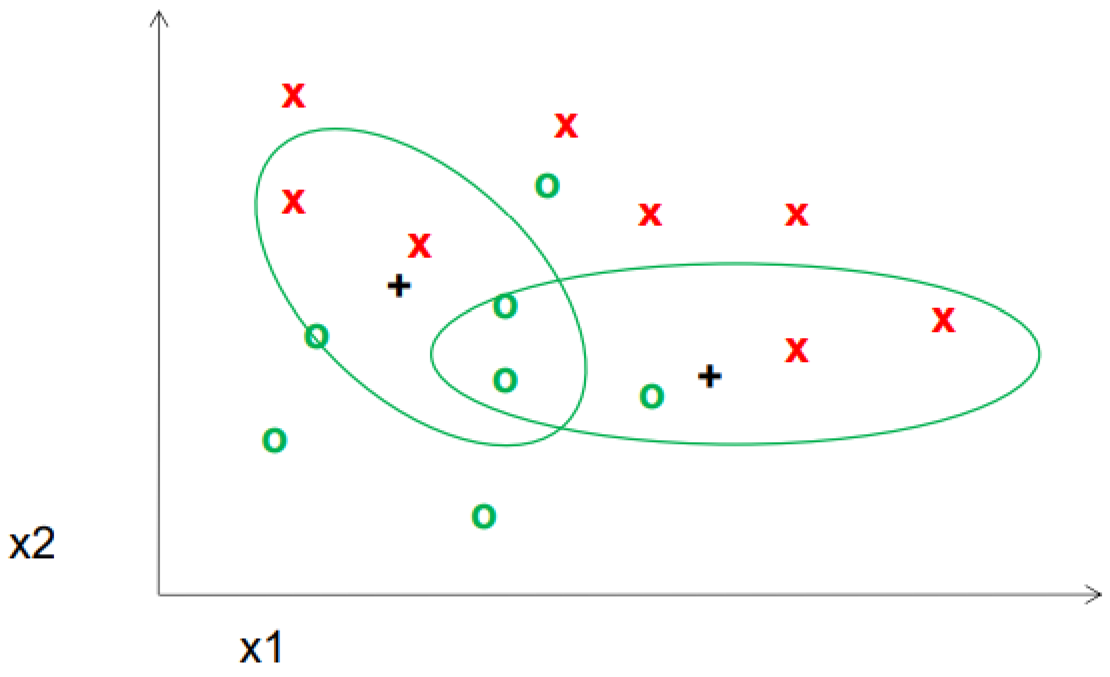
K-Nearest Neighbor

- When considering more than one neighbor, we use voting to assign a label.

3-nearest neighbor



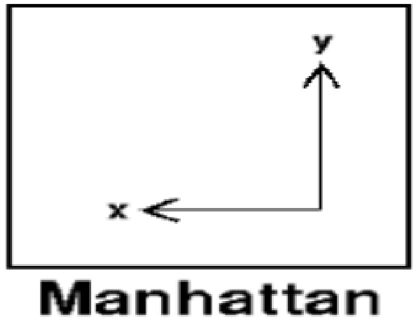
5-nearest neighbor

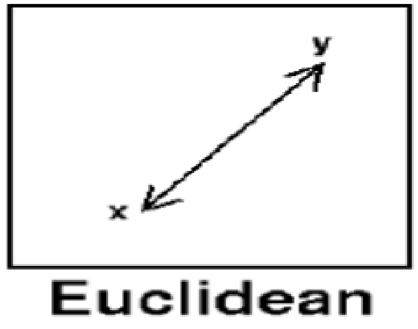


K-Nearest Neighbor

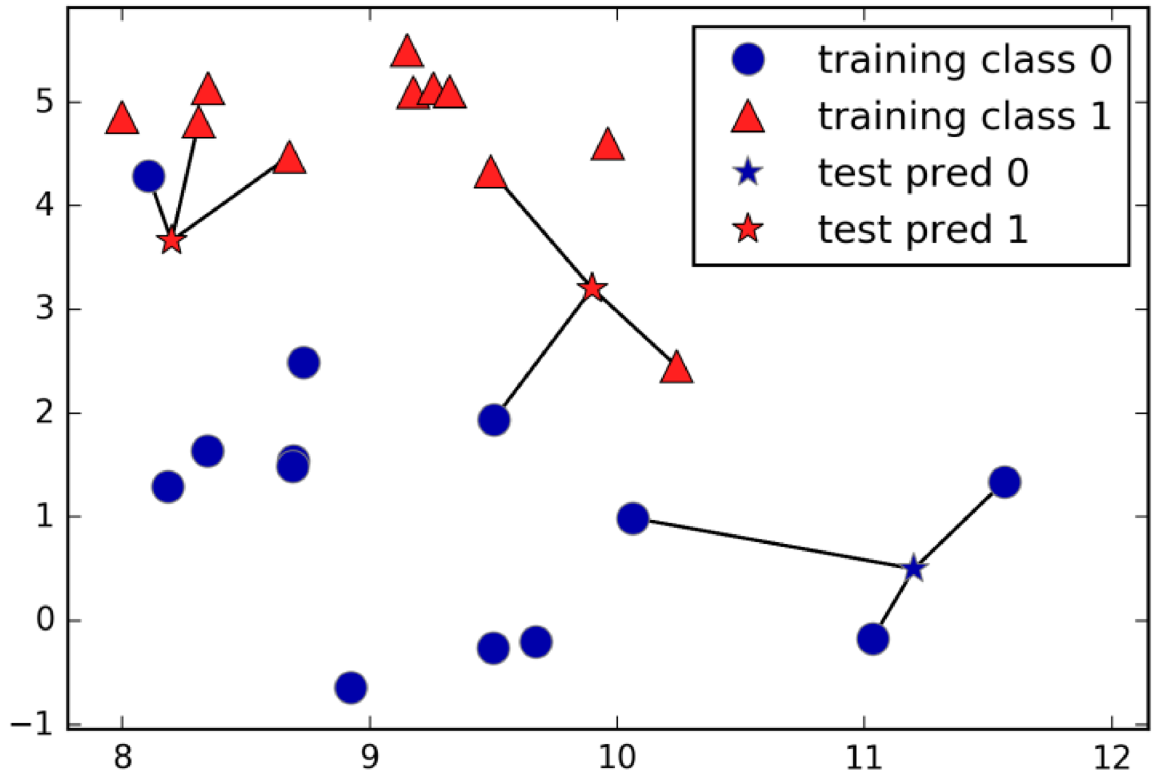
- When considering more than one neighbor, we use voting to assign a label.

- Manhattan Distance, |X1-X2| + |Y1-Y2|
- Euclidean Distance, $\sqrt{((x1-x2)^2)}+\sqrt{(y1-y2)^2}$



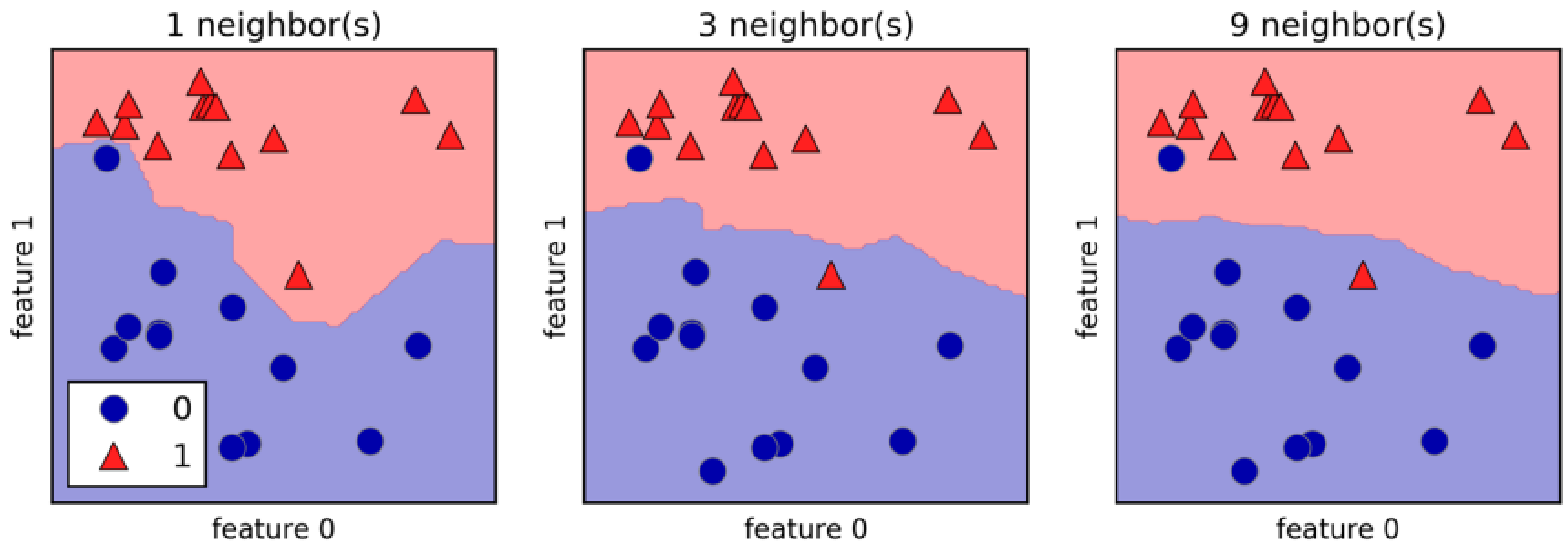


3-nearest neighbor



K-Nearest Neighbor

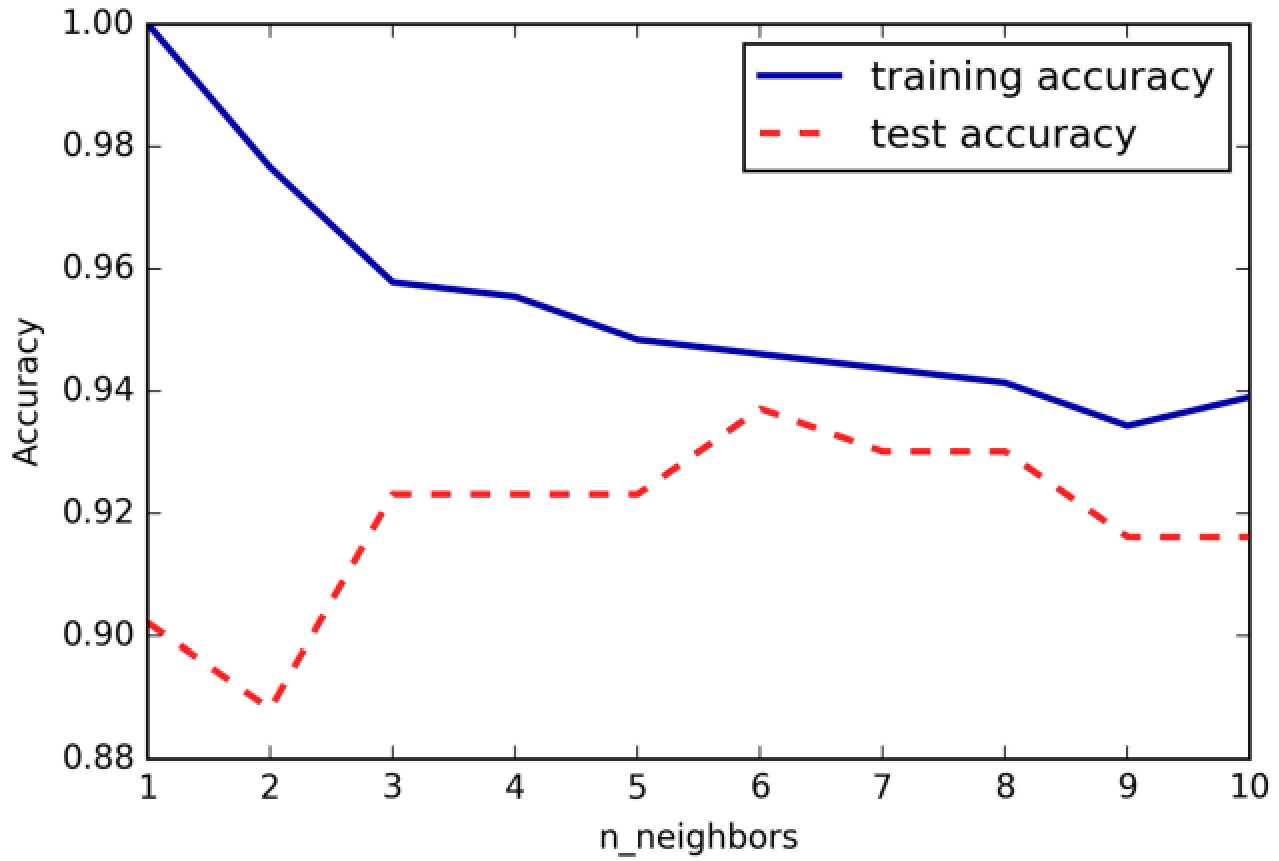
- Decision boundaries created by the nearest neighbors model for different values of n_neighbors



- A very low value for K such as K=1 or K=2, can be noisy and lead to the effects of outliers in the model.

K-Nearest Neighbor

- Comparison of training and test accuracy as a function of n_neighbors

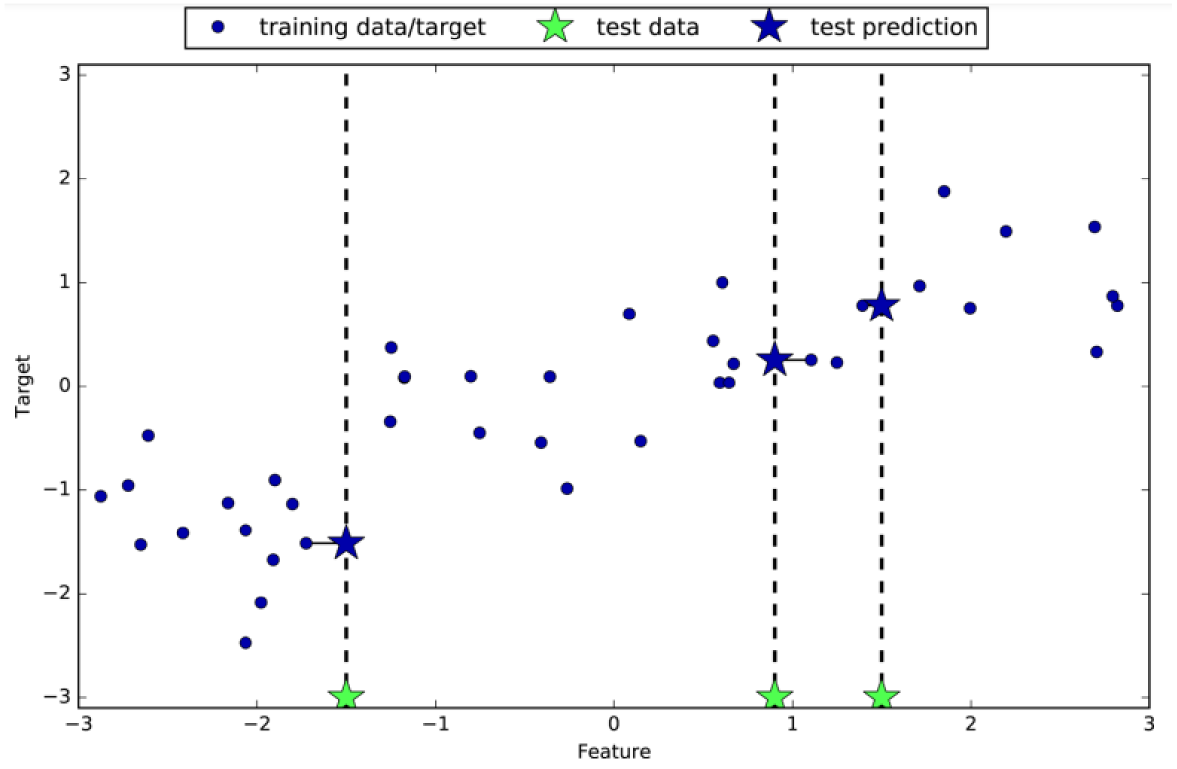


K-Nearest Neighbor

- from sklearn.model_selection import train_test_split
- from sklearn.neighbors import KNeighborsClassifier
- X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=0)
- clf = KNeighborsClassifier(n_neighbors=3)
- clf.fit(X_train, y_train)
- print("Test set predictions: {}".format(clf.predict(X_test)))
- print("Test set accuracy: {:.2f}".format(clf.score(X_test, y_test)))

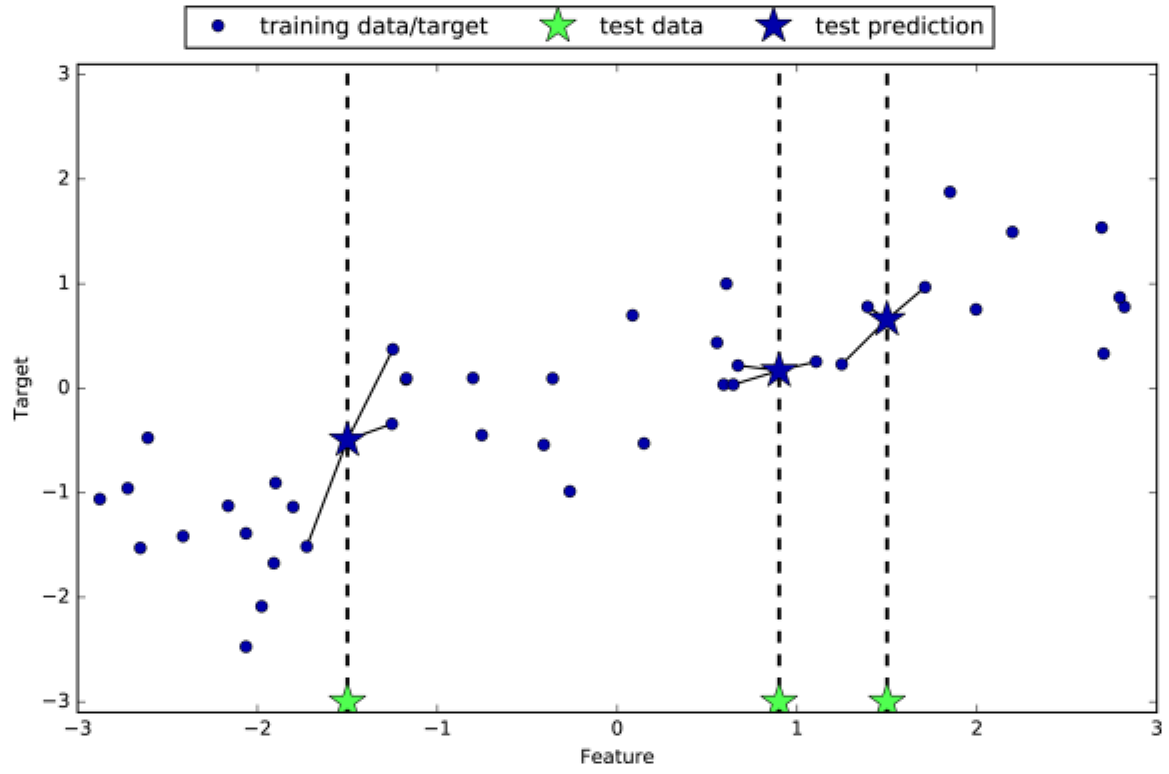
K-Nearest Neighbor

- k-neighbors regression : There is also a regression variant of the k-nearest neighbors algorithm.
- Predictions made by one-nearest-neighbor regression on the wave dataset



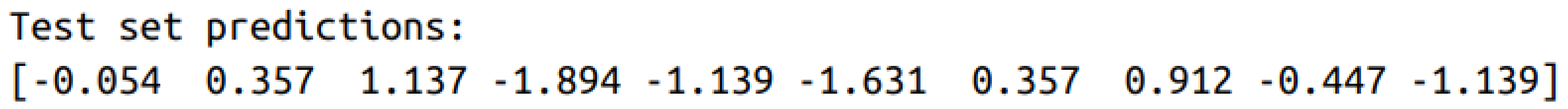
K-Nearest Neighbor

- Predictions made by three-nearest-neighbors regression on the wave dataset



K-Nearest Neighbor

- from sklearn.neighbors import KNeighborsRegressor
- X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=0)
- reg = KNeighborsRegressor(n_neighbors=3)
- reg.fit(X_train, y_train)
- print("Test set predictions:\n{}".format(reg.predict(X_test))) ???

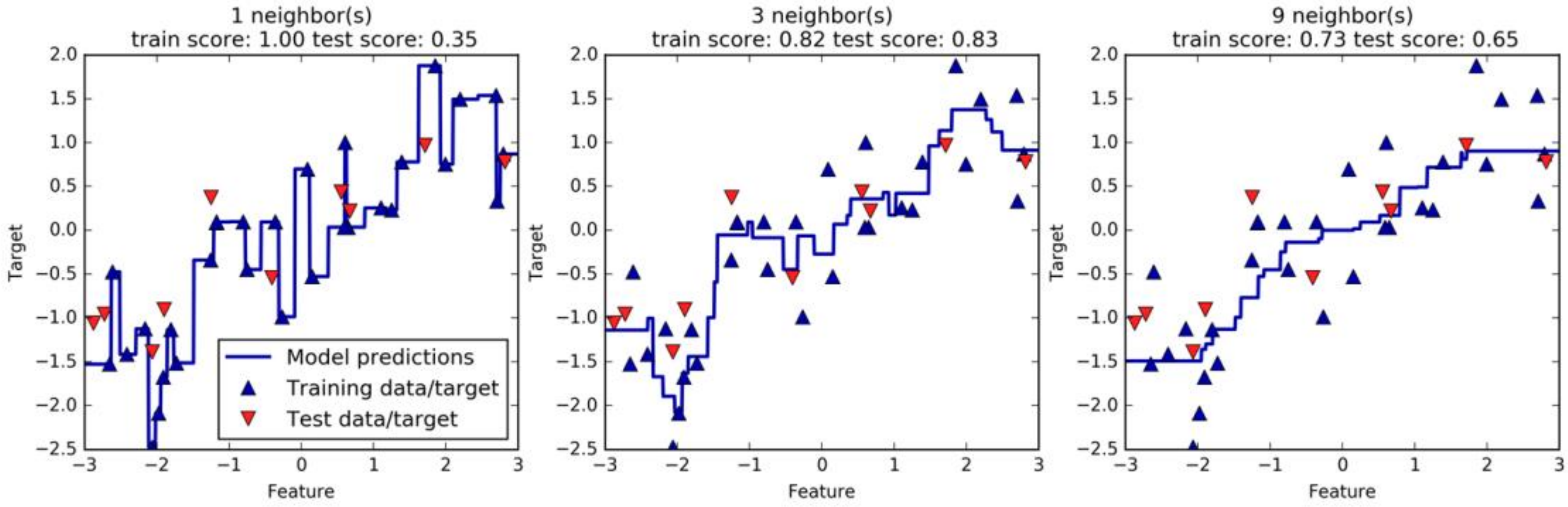


```
Test set predictions:
[-0.054  0.357  1.137 -1.894 -1.139 -1.631  0.357  0.912 -0.447 -1.139]
```

- print("Test set R^2: {:.2f}".format(reg.score(X_test, y_test)))

K-Nearest Neighbor

- Comparing predictions made by nearest neighbors regression for different values of n_neighbors



Strengths, weaknesses, and parameters

- In principle, there are two important parameters to the KNeighbors classifier: the number of neighbors and how you measure distance between data points. In practice, using a small number of neighbors like three or five often works well, but you should certainly adjust this parameter. By default, Euclidean distance is used, which works well in many settings.
- One of the strengths of k-NN is that the model is very easy to understand, and often gives reasonable performance without a lot of adjustments.
- Using this algorithm is a good baseline method to try before considering more advanced techniques.
- Building the nearest neighbors model is usually very fast, but when your training set is very large (either in number of features or in number of samples) prediction can be slow.

Strengths, weaknesses, and parameters

- When using the k-NN algorithm, it’s important to preprocess your data. This approach often does not perform well on datasets with many features (hundreds or more), and it does particularly badly with datasets where most features are 0 most of the time (so-called sparse datasets).
- So, while the nearest k-neighbors algorithm is easy to understand, it is not often used in practice, due to prediction being slow and its inability to handle many features.

Evaluation Metrics and Scoring

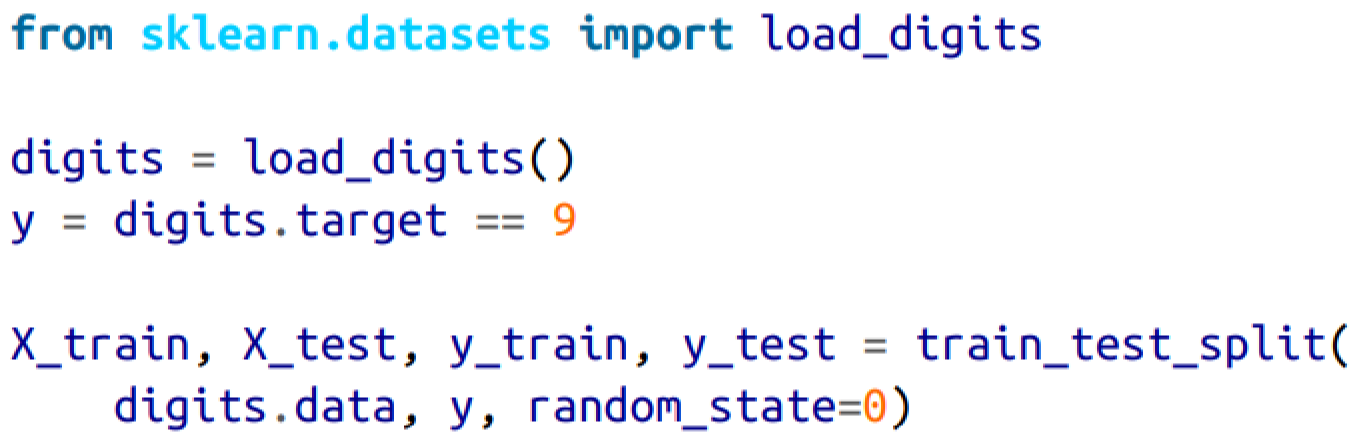
- Metrics for Binary Classification:
- positive class and a negative class,
- the positive class is the one we are looking for.
- accuracy is not a good measure of predictive performance!
- early detection of cancer: positive test (an indication of cancer) the positive class, and a negative test the negative class.
- One possible mistake is that a healthy patient will be classified as positive, leading to additional testing. This leads to some costs and an inconvenience for the patient (and possibly some mental distress).
- An incorrect positive prediction is called a false positive.

Evaluation Metrics and Scoring

- The other possible mistake is that a sick patient will be classified as negative, and will not receive further tests and treatment.
- The undiagnosed cancer might lead to serious health issues, and could even be fatal. A mistake of this kind—an incorrect negative prediction—is called a false negative.
- In statistics, a false positive is also known as type I error, and a false negative as type II error.
- In the cancer diagnosis example, it is clear that we want to avoid false negatives as much as possible, while false positives can be viewed as more of a minor nuisance.

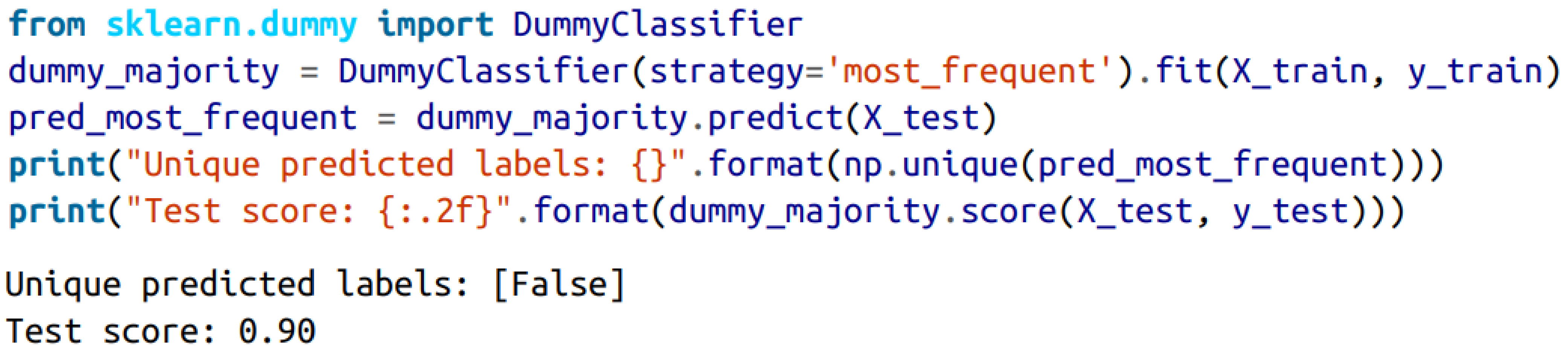
Evaluation Metrics and Scoring

- Imbalanced datasets : Types of errors play an important role when one of two classes is much more frequent than the other one.
- 99 “no click” data points, there is 1 “clicked” data point; 99% of the samples belong to the “no click” class.
- you build a classifier that is 99% accurate on the click prediction task.



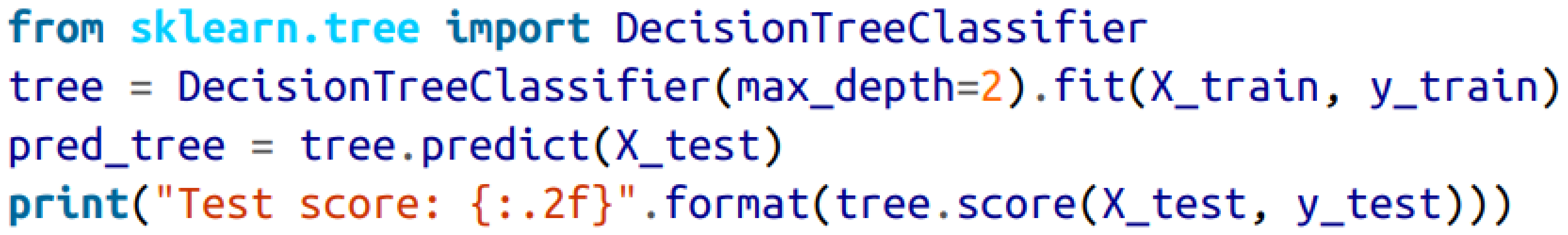
```
from sklearn.datasets import load_digits

digits = load_digits()
y = digits.target == 9

X_train, X_test, y_train, y_test = train_test_split(
    digits.data, y, random_state=0)
```

Evaluation Metrics and Scoring



```python
from sklearn.dummy import DummyClassifier
dummy_majority = DummyClassifier(strategy='most_frequent').fit(X_train, y_train)
pred_most_frequent = dummy_majority.predict(X_test)
print("Unique predicted labels: {}".format(np.unique(pred_most_frequent)))
print("Test score: {:.2f}".format(dummy_majority.score(X_test, y_test)))
```

```
Unique predicted labels: [False]
Test score: 0.90
```

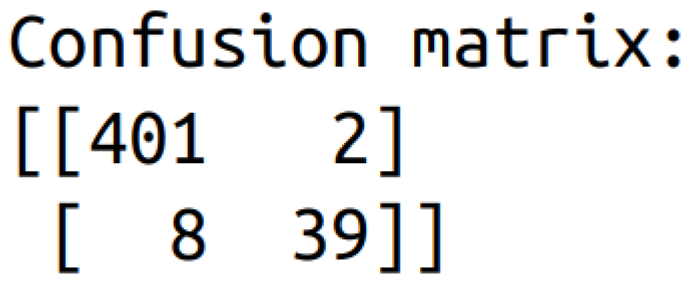


```python
from sklearn.tree import DecisionTreeClassifier
tree = DecisionTreeClassifier(max_depth=2).fit(X_train, y_train)
pred_tree = tree.predict(X_test)
print("Test score: {:.2f}".format(tree.score(X_test, y_test)))
```

Test score: 0.92

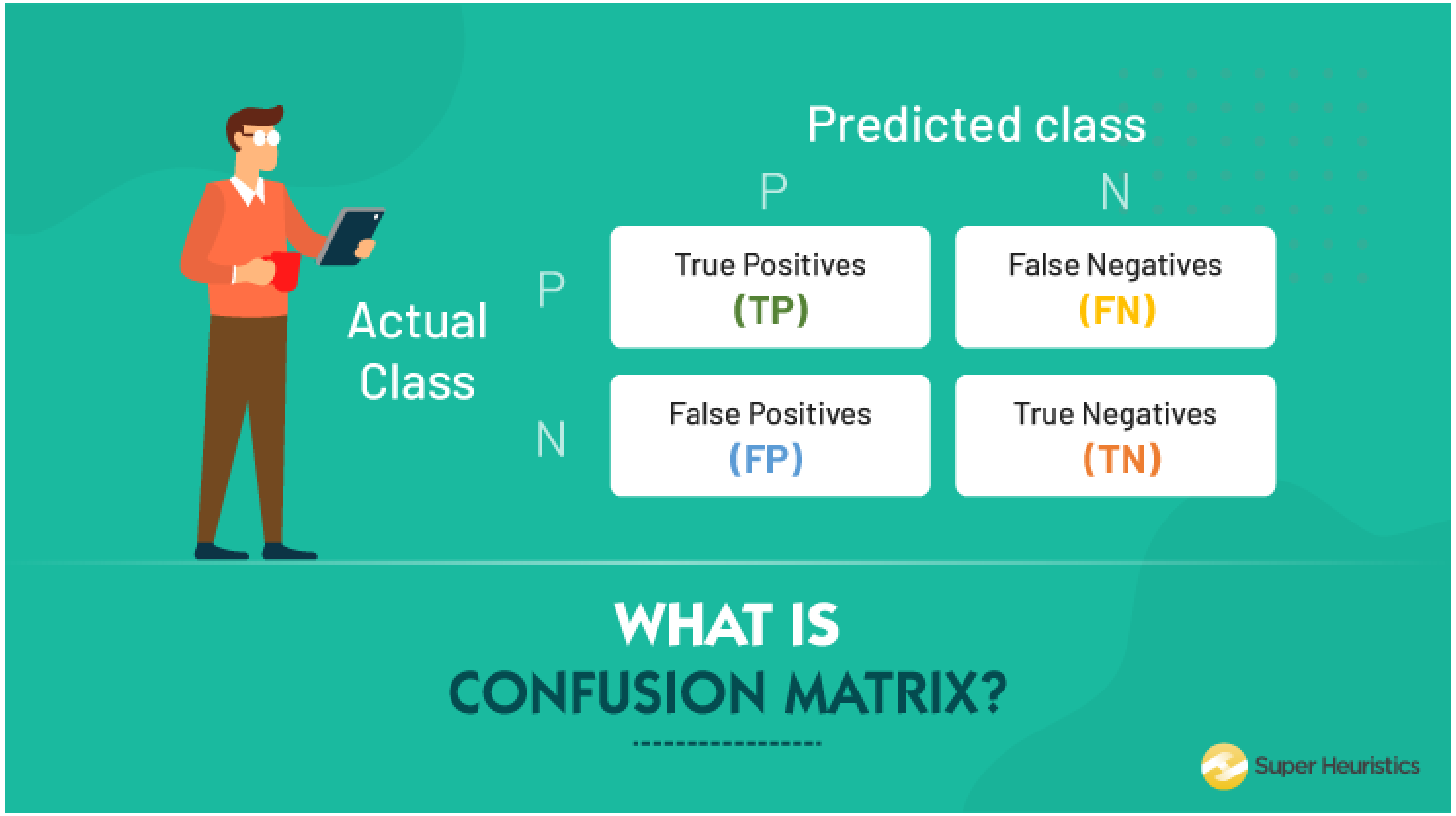
Evaluation Metrics and Scoring

- Confusion matrices:
- from sklearn.metrics import confusion_matrix
- confusion = confusion_matrix(y_test, pred_logreg)
- print("Confusion matrix:\n{}".format(confusion))



```
Confusion matrix:
[[401   2]
 [  8  39]]
```

Evaluation Metrics and Scoring: Confusion matrix



$$\text{Recall} = \frac{TP}{TP+FN}$$

sensitivity, hit rate, or true positive rate (TPR)

$$\text{Accuracy} = \frac{TP+TN}{TP+TN+FP+FN}$$

$$\text{Precision} = \frac{TP}{TP+FP}$$

$$F = 2 \cdot \frac{\text{precision} \cdot \text{recall}}{\text{precision} + \text{recall}}$$

Evaluation Metrics and Scoring: Confusion matrix

- Precision is used as a performance metric when the goal is to limit the number of false positives.
- As an example, imagine a model for predicting whether a new drug will be effective in treating a disease in clinical trials. Clinical trials are notoriously expensive, and a pharmaceutical company will only want to run an experiment if it is very sure that the drug will actually work. Therefore, it is important that the model does not produce many false positives—in other words, that it has a high precision.
- Precision is also known as positive predictive value (PPV).

$$\text{Precision} = \frac{\text{TP}}{\text{TP+FP}}$$

Evaluation Metrics and Scoring: Confusion matrix

- Recall is used as performance metric when we need to identify all positive samples; that is, when it is important to avoid false negatives.
- The cancer diagnosis example from earlier in this chapter is a good example for this: it is important to find all people that are sick, possibly including healthy patients in the prediction.
- Other names for recall are sensitivity, hit rate, or true positive rate (TPR).

$$\text{Recall} = \frac{\text{TP}}{\text{TP}+\text{FN}}$$

Evaluation Metrics and Scoring: Confusion matrix

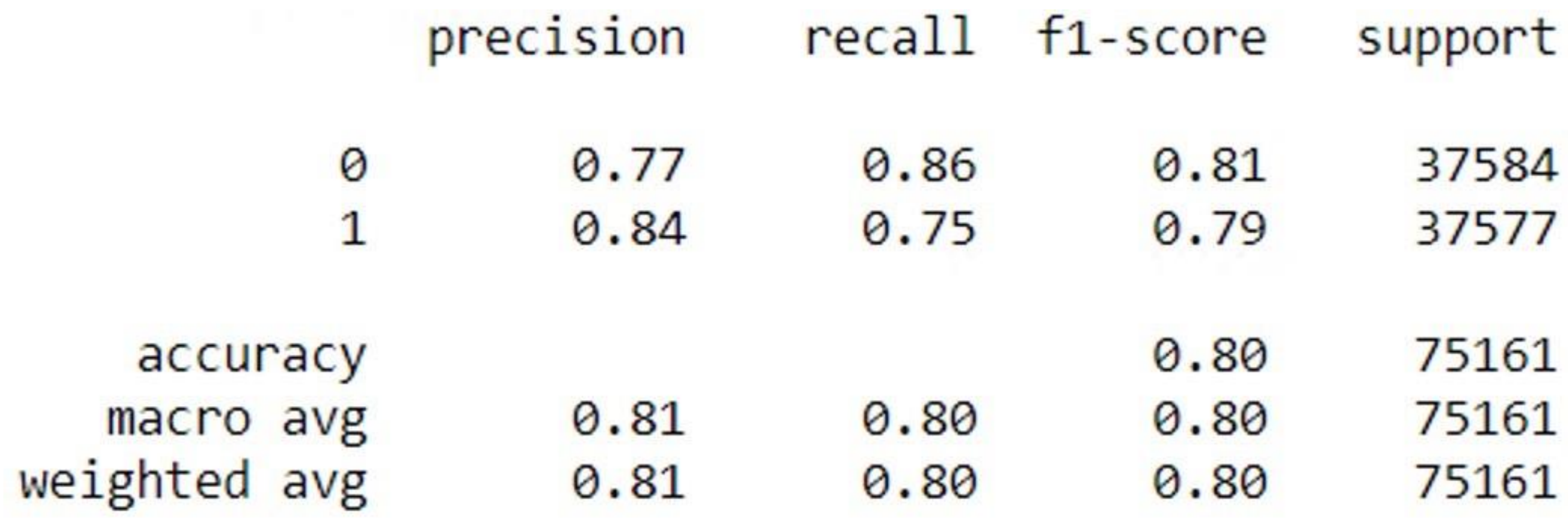
- So, while precision and recall are very important measures, looking at only one of them will not provide you with the full picture.
- One way to summarize them is the f-score or f-measure, which is with the harmonic mean of precision and recall:

$$F = 2 \cdot \frac{\text{precision} \cdot \text{recall}}{\text{precision} + \text{recall}}$$

- it can be a better measure than accuracy on imbalanced binary classifi- cation datasets.
- A disadvantage of the f-score, however, is that it is harder to interpret and explain than accuracy.

Evaluation Metrics and Scoring: Confusion matrix

- from sklearn.metrics import classification_report
  print(classification_report(y_test, y_pred, target_names=["0", "1"]))



```
              precision    recall  f1-score   support

           0       0.77      0.86      0.81     37584
           1       0.84      0.75      0.79     37577

    accuracy                           0.80     75161
   macro avg       0.81      0.80      0.80     75161
weighted avg       0.81      0.80      0.80     75161
```