K nearest neighbor

Fatemeh Mansoori

This slides are created based on the slides of the Sebastian Raschka for the introduction to machine learning course and Ali Sharifi Zarchi for the introduction to machine learning course

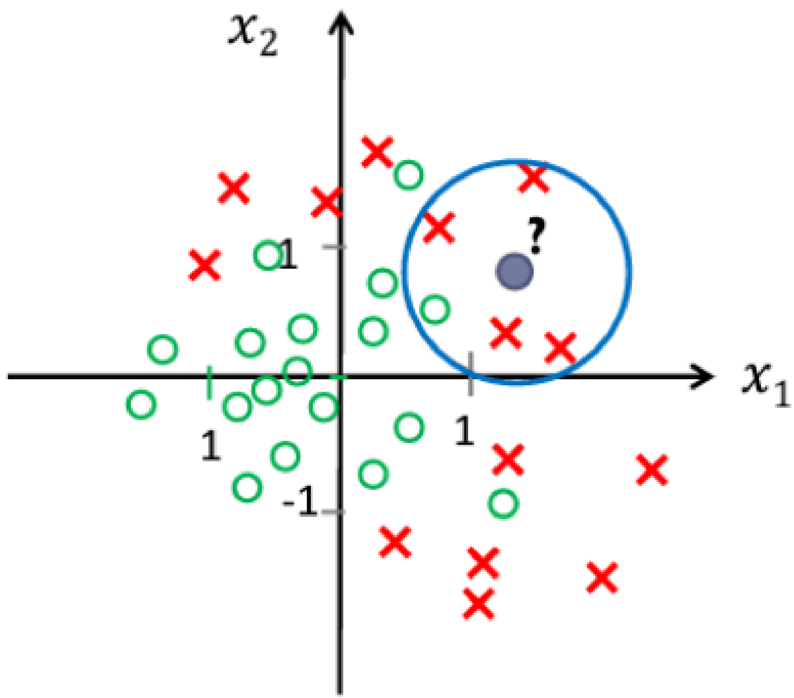
Topics

- **Intro to nearest neighbor models**
- Nearest neighbor decision boundary
- K-nearest neighbors
- Improving k-nearest neighbors: modifications and hyperparameters
- K-nearest neighbors in Python

Parametric vs. non-parametric methods

- **Parametric** methods need to **find parameters** from data and then use the inferred parameters to decide on new data points
    - Learning: finding parameters from data
    - e.g., **Linear regression**, **Logistic regression**
- **Non-parametric** methods
    - Training examples are **explicitly** used
    - **Training phase** is **not required**
    - e.g., **k-Nearest neighbors (kNN)**
- Both supervised and unsupervised learning can be categorized into parametric and non-parametric methods

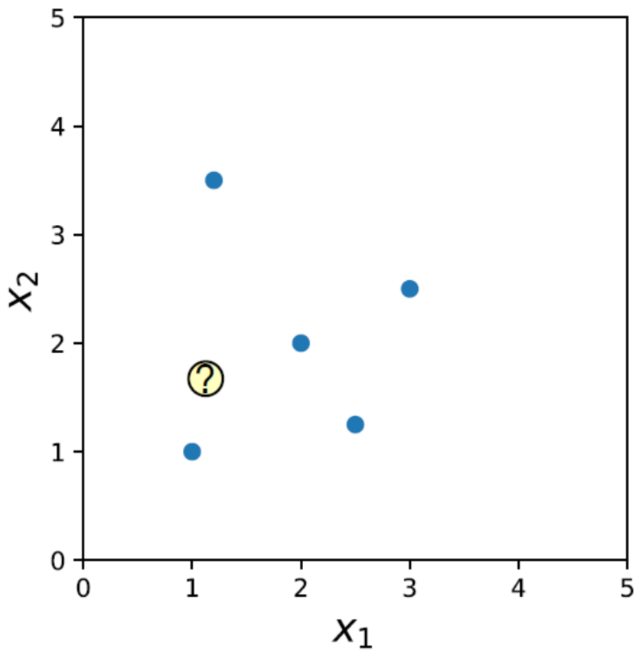
- K-NN classifier: $k \geq 1$ nearest neighbors
  - Label for x predicted by majority voting among its $k - NN$
- $k = 5$, $x = [x_1, x_2]$

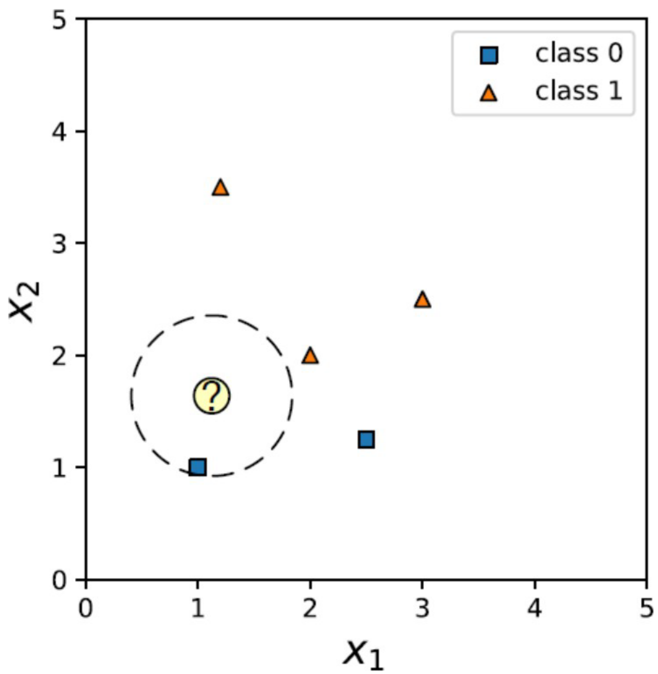


1-Nearest Neighbor

1-Nearest Neighbor

- Task: predict the target / label of a new data point





- How? Look at most "similar" data point in training set

1-Nearest Neighbor Training Step

$$\langle \mathbf{x}^{[i]}, y^{[i]} \rangle \in \mathcal{D} \quad (|\mathcal{D}| = n)$$

How do we "train" the 1-NN model?

To train the 1-NN model, we simply "remember" the training dataset

1-Nearest Neighbor Prediction Step

closest_point := None

closest_distance := ∞

query point

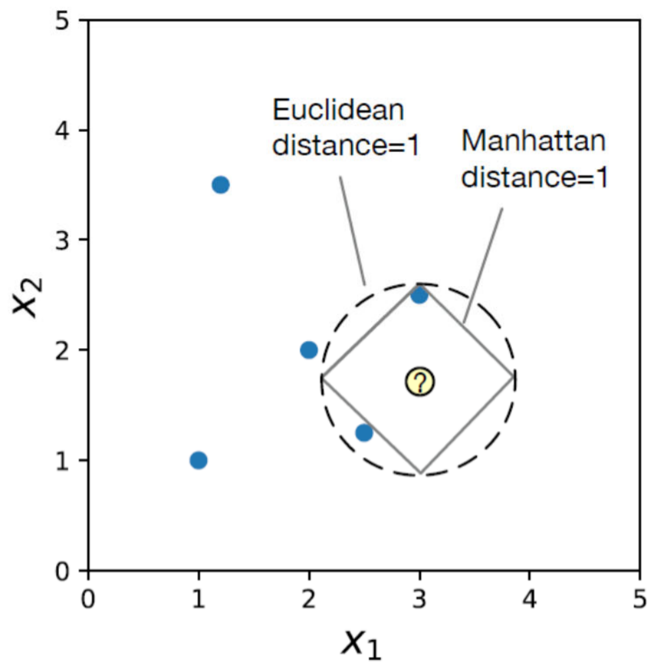
- for $i = 1, \ldots, n$:
    - current_distance := $d(\mathbf{x}^{[i]}, \mathbf{x}^{[q]})$
    - if current_distance < closest_distance:
        - closest_distance := current_distance
        - closest_point := $\mathbf{x}^{[i]}$
- return f(closest_point)

Which Point is Closest to ? ?

- Depends on the Distance Measure!
- Commonly used: Euclidean Distance

$$d(\mathbf{x}^{[a]}, \mathbf{x}^{[b]}) = \sqrt{\sum_{j=1}^{m} \left(x_j^{[a]} - x_j^{[b]}\right)^2}$$

- Other metrics : Manhatan distance
-



Distance Measure

- Euclidean distance

$$d(x, x') = \sqrt[2]{\|x - x'\|_2^2} = \sqrt[2]{(x_1 - x'_1)^2 + \cdots + (x_d - x'_d)^2}$$

- Distance learning methods for this purpose
    - Weighted Euclidean distance

$$d_w(x, x') = \sqrt[2]{w_1(x_1 - x'_1)^2 + \cdots + w_d(x_d - x'_d)^2}$$

Distance Measure

- Minkowski distance

$$d(x, x') = (\sum_{i=1}^{n} |x_i - x'_i|^p)^{\frac{1}{p}}$$

  - for $p \geq 1$ is a distance metric
  - As you can see Minkowski distance with $p = 2$ is the same as Euclidean distance
  - Minkowski distance is the same as L^p norm of $(x - x')$
- Remember L^p norm from linear algebra:

$$\|x\|_p = \sqrt[p]{(|x_1|^p + \cdots + |x_n|^p)}$$

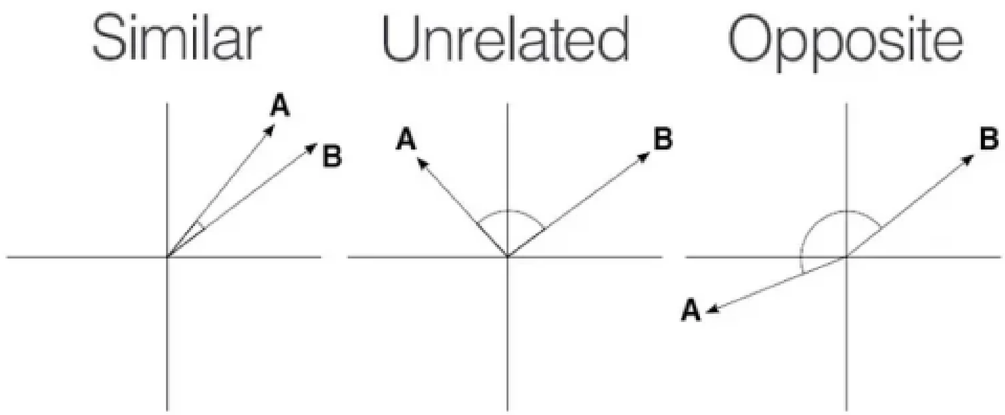
$$\text{Some famous } L^p \text{ norms} \begin{cases} \|x\|_1 & = \sum_{i=1}^{n} |x_i| \\ \|x\|_2 & = \sqrt{x_1^2 + \cdots + x_n^2} \\ \|x\|_\infty & = \max\{|x_1|, |x_2|, \ldots, |x_n|\} \end{cases}$$

Distance Measure

- Cosine distance (angle)

$$d(x, x') = 1 - \text{cosine similarity}(x, x')$$

Where, $\quad \text{cosine similarity}(x, x') = \dfrac{x.x'}{\|x\|_2 \|x'\|_2} = \dfrac{\sum_{i=1}^{d} x_i x'_i}{\sqrt{\sum_{i=1}^{d} x_i^2}\sqrt{\sum_{i=1}^{d} x_i'^2}}$



Example of angle difference for cosine similarity

Discrete Distance Measures

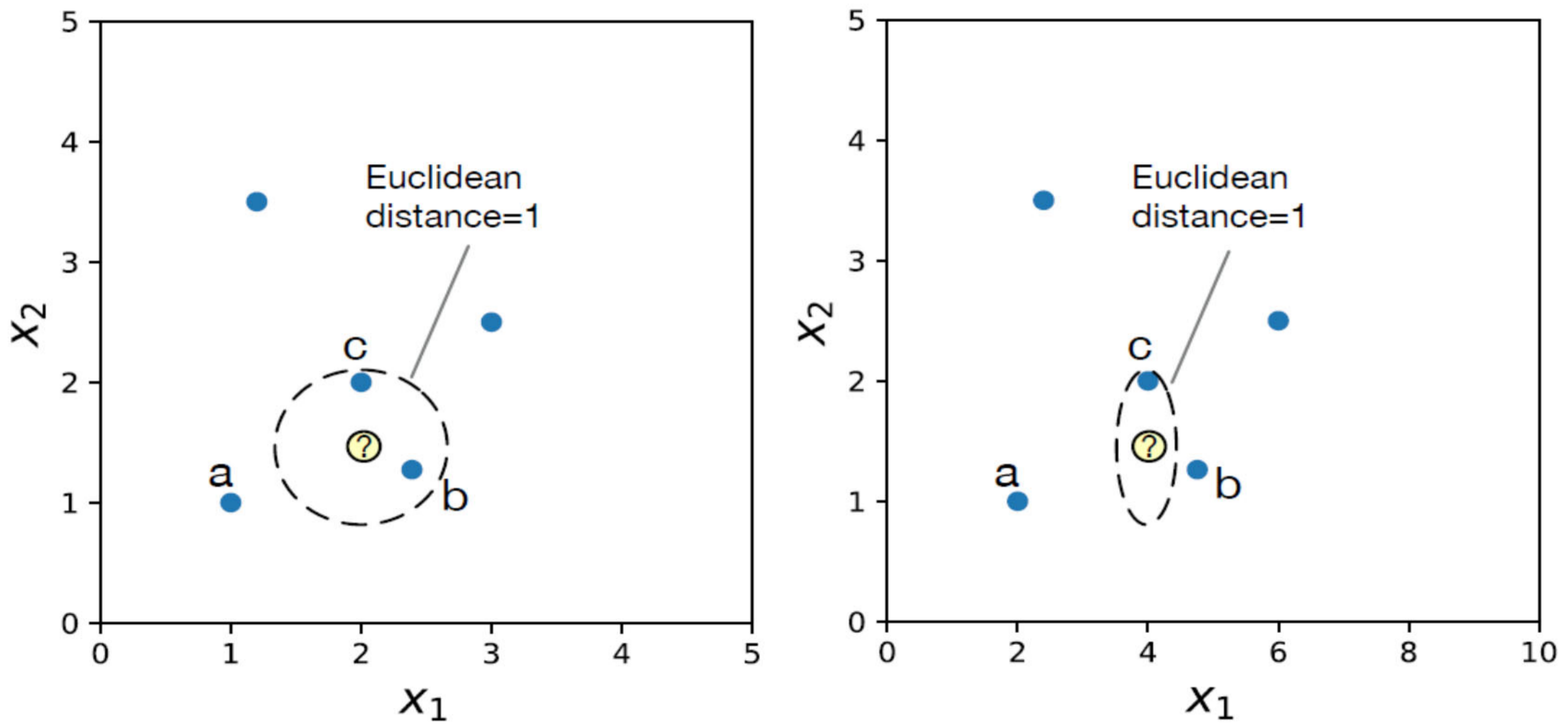
Hamming distance: $d(\mathbf{x}^{[a]}, \mathbf{x}^{[b]}) = \sum_{j=1}^{m} \left| x_j^{[a]} - x_j^{[b]} \right|$ **where** $x_j \in \{0,1\}$

Jaccard/Tanimoto similarity:

$$J(A, B) = \frac{|A \cap B|}{|A \cup B|} = \frac{|A \cap B|}{|A| + |B| - |A \cap B|}$$

Dice: $$D(A, B) = \frac{2|A \cap B|}{|A| + |B|}$$

Feature Scaling



Feature Scaling

- Min-Max Scaling

$$X' = \frac{X - X_{min}}{X_{max} - X_{min}}$$

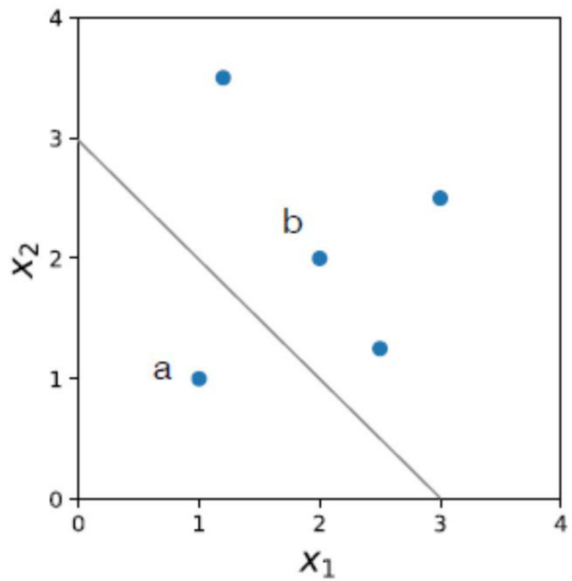
- Z-score normalization

$$X' = \frac{X - \mu}{\sigma}$$

Nearest Neighbor Decision Boundary

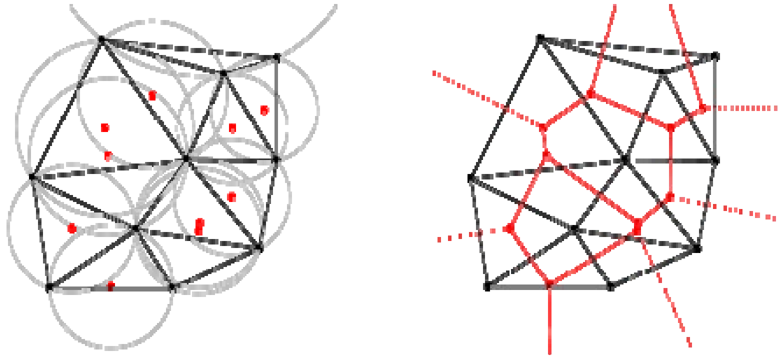
- With **kNN** we can obtain **non-linear** decision surfaces unlike the previous methods (linear and logistic regression)
- But note that this method could be prone to **outliers** or **noisy** data especially if:
  - We have **small dataset**
  - Our data is **low-dimensional**
  - We use a **small value of k** (like $k = 1$ is only determined by the nearest neighbor and could be misleading in many test cases.

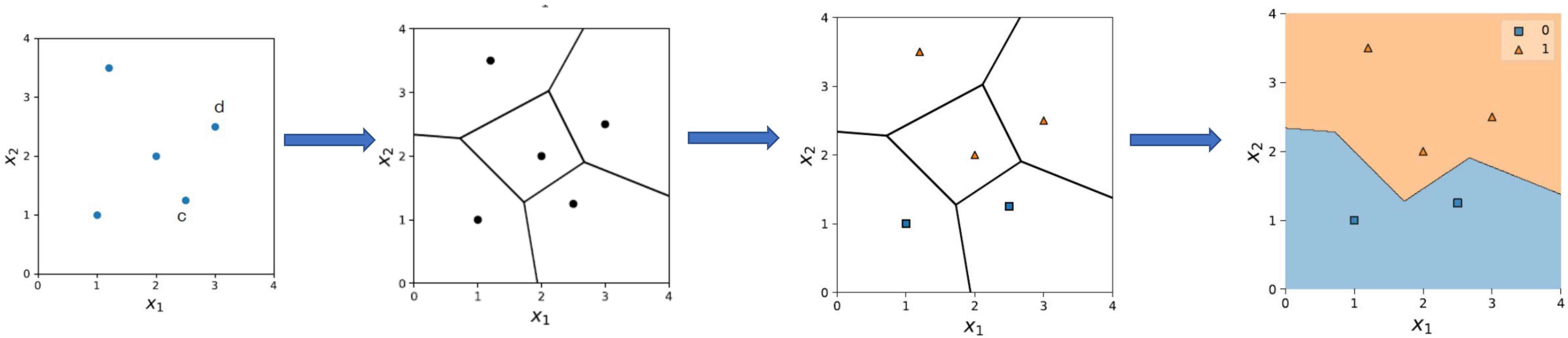
Decision Boundary Between (a) and (b)



Decision Boundary of 1-NN

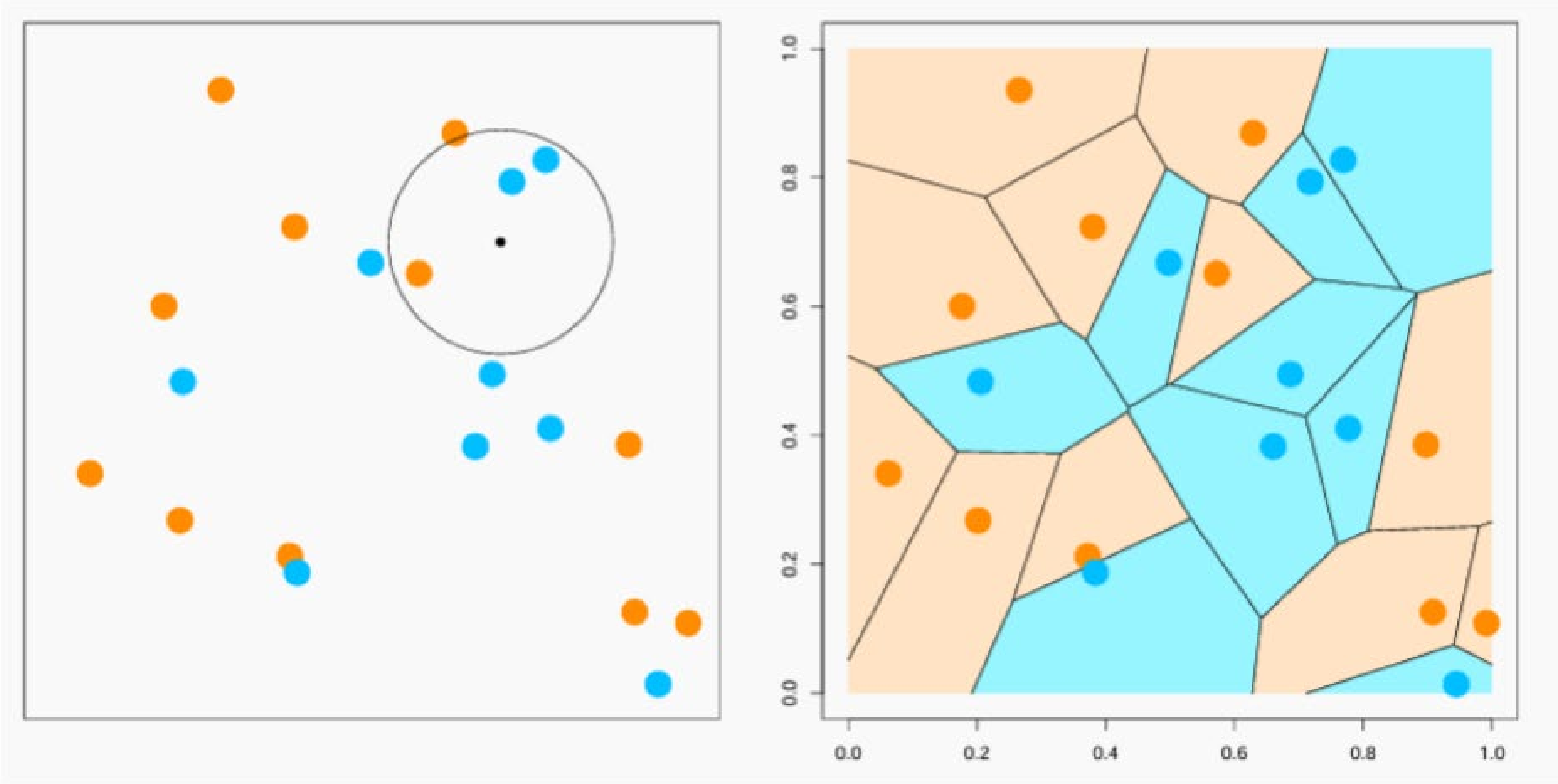
Using Delaunay triangulation





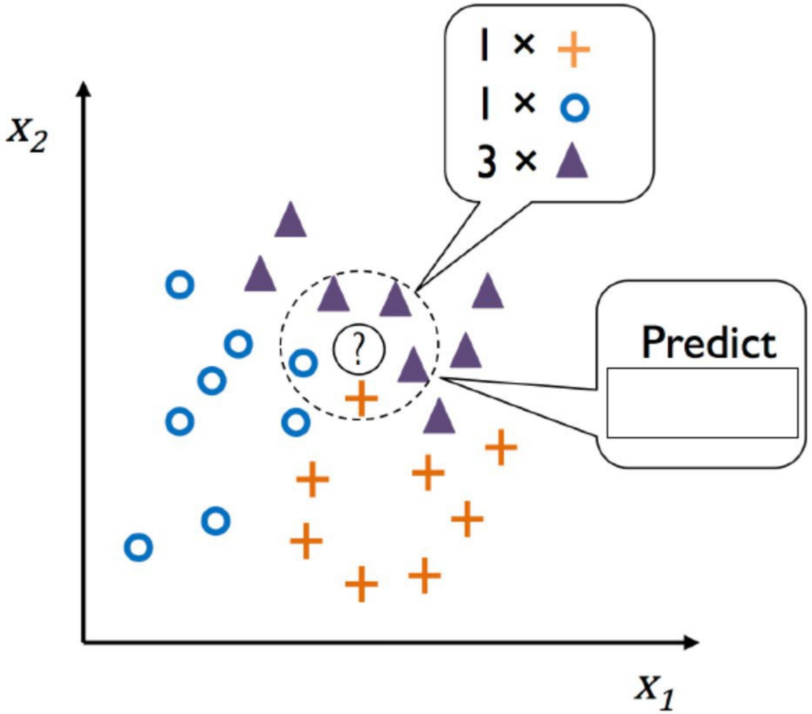
Voronoi tessellation

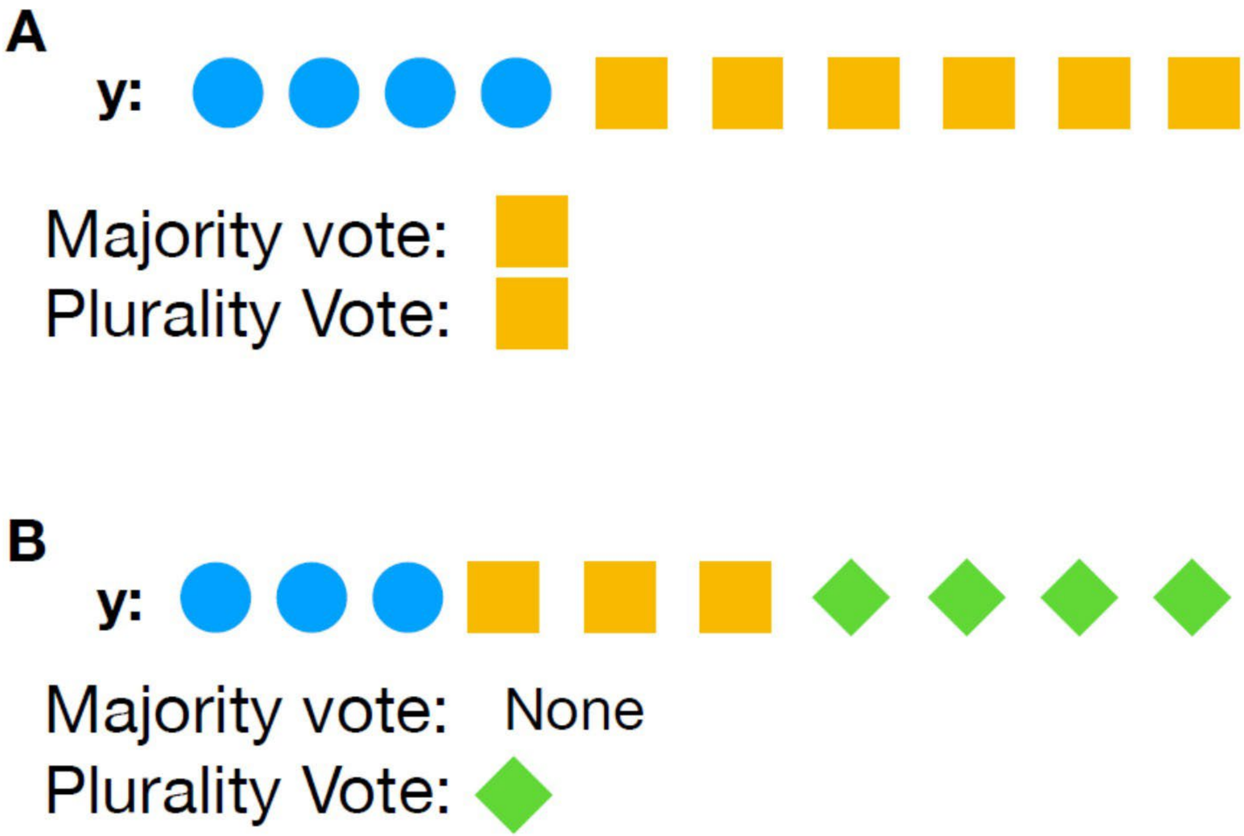
- 1NN plot is a Voronoi tessellation



K nearest neighbour

k-Nearest Neighbors




A
y:
Majority vote:
Plurality Vote:
B
y:
Majority vote: None
Plurality Vote:

kNN for Classification

$$\mathscr{D}_k = \{\langle \mathbf{x}^{[1]}, f(\mathbf{x}^{[1]})\rangle, \ldots, \langle \mathbf{x}^{[k]}, f(\mathbf{x}^{[k]})\rangle\} \quad \mathscr{D}_k \subseteq \mathscr{D}$$

$$h(\mathbf{x}^{[q]}) = arg \max_{y \in \{1,\ldots,t\}} \sum_{i=1}^{k} \delta(y, f(\mathbf{x}^{[i]}))$$

$$\delta(a, b) = \begin{cases} 1, & \textbf{if } a = b, \\ 0, & \textbf{if } a \neq b. \end{cases}$$

$$h(\mathbf{x}^{[t]}) = \textbf{mode}\left(\left\{f(\mathbf{x}^{[1]}), \ldots, f(\mathbf{x}^{[k]})\right\}\right)$$

*k*NN for Regression

$$\mathscr{D}_k = \{\langle \mathbf{x}^{[1]}, f(\mathbf{x}^{[1]}) \rangle, \ldots, \langle \mathbf{x}^{[k]}, f(\mathbf{x}^{[k]}) \rangle\} \qquad \mathscr{D}_k \subseteq \mathscr{D}$$

$$h(\mathbf{x}^{[t]}) = \frac{1}{k} \sum_{i=1}^{k} f(\mathbf{x}^{[i]})$$

Distance-weighted *k*NN

$$h(\mathbf{x}^{[t]}) = arg \max_{j \in \{1,...,p\}} \sum_{i=1}^{k} w^{[i]} \delta(j, f(\mathbf{x}^{[i]}))$$

$$w^{[i]} = \frac{1}{d(\mathbf{x}^{[i]}, \mathbf{x}^{[t]})^2}$$

Nearest Neighbor Search

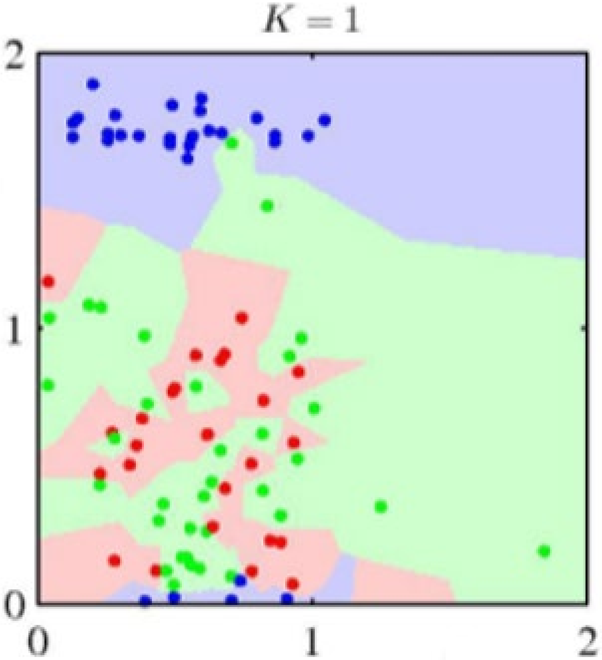
$\mathcal{D}_k := \{\}$

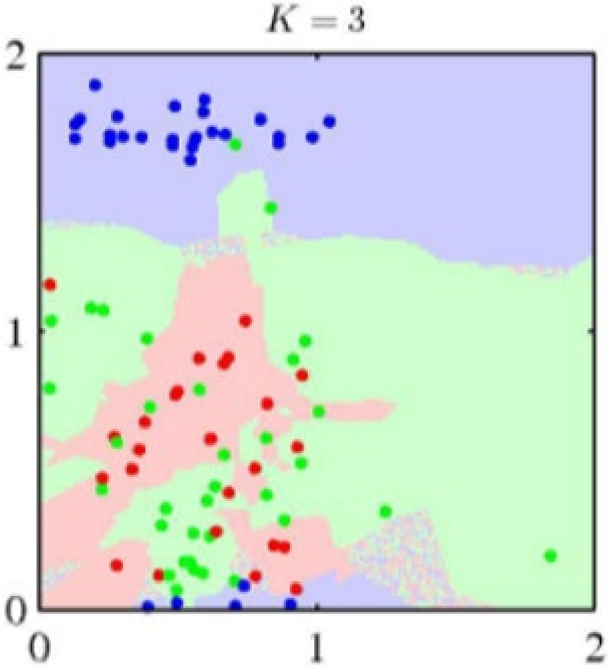
while $|\mathcal{D}_k| <$ k:

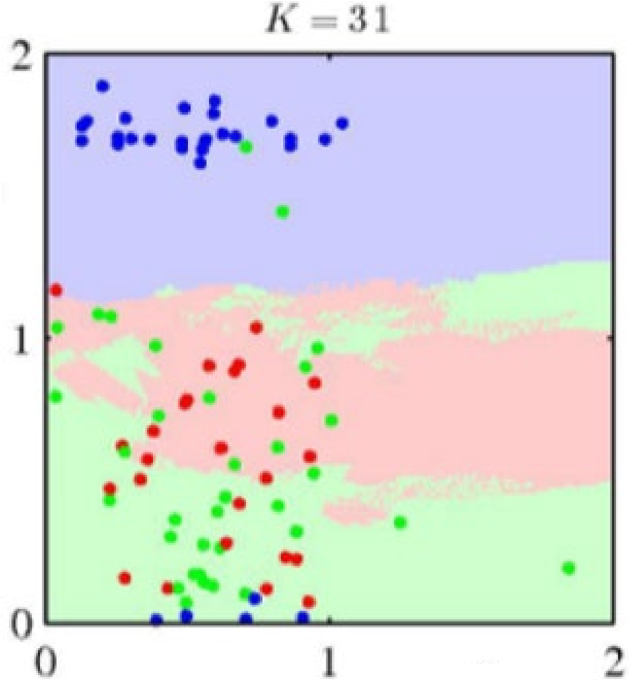
- $\texttt{closest\_distance} := \infty$
- for $i = 1, ..., n, \quad \forall i \notin \mathcal{D}_k$:
  - $\texttt{current\_distance} := d(\mathbf{x}^{[i]}, \mathbf{x}^{[q]})$
  - if $\texttt{current\_distance} < \texttt{closest\_distance}$:
    - $\texttt{closest\_distance} := \texttt{current\_distance}$
    - $\texttt{closest\_point} := \mathbf{x}^{[i]}$
- add $\texttt{closest\_point}$ to $\mathcal{D}_k$

Hyperparameter

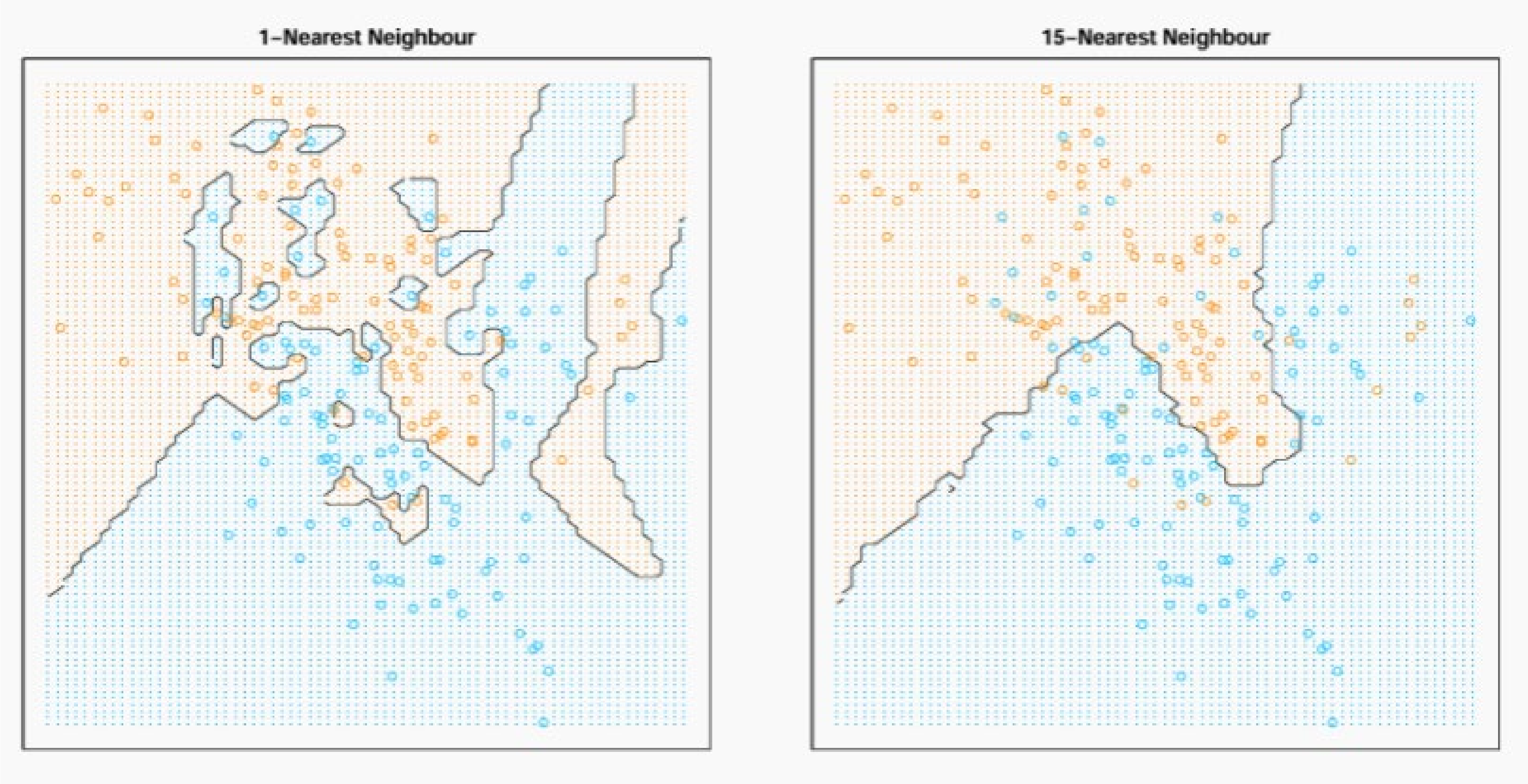
Effect of k







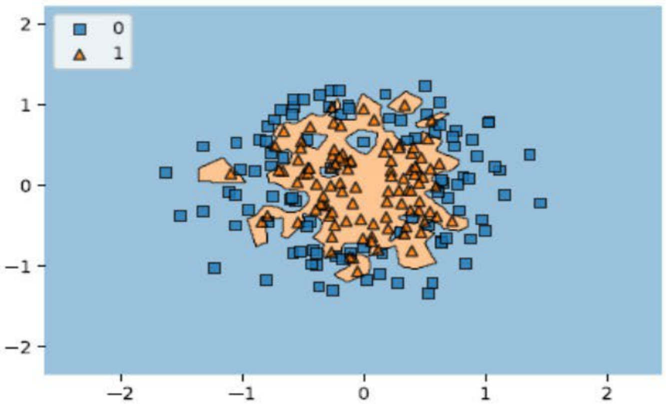
compare k = 1 with k = 15



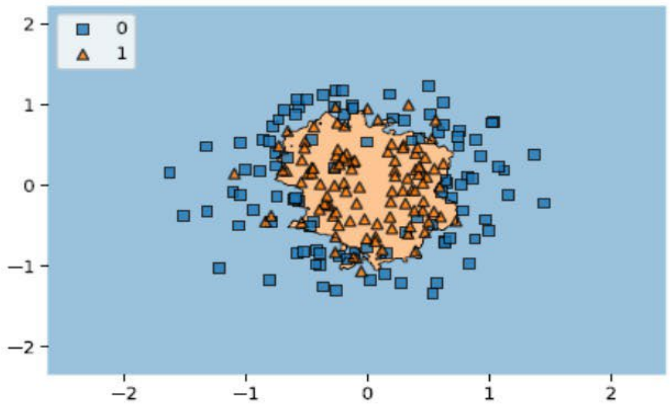
Value of k

$$k \in \{1,3,7\}$$

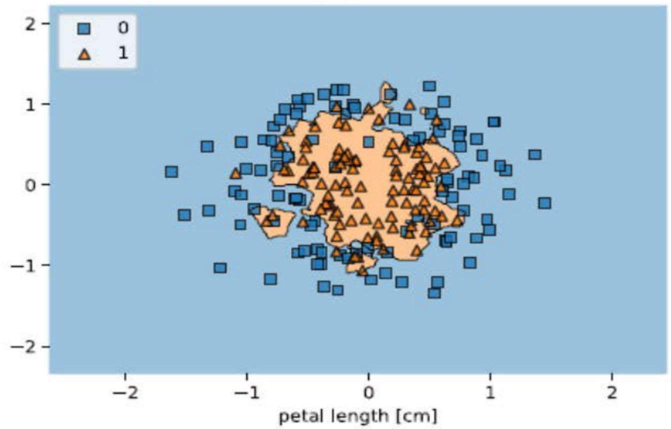
$k = \_$



$k = \_$



$k = \_$



How to choose the value of K

- **Cross-Validation**
  - try different values of K and evaluate the performance of the model using metrics like accuracy, precision, recall, or F1 score
- **Odd vs. Even K**
  - It is generally recommended to use an odd value for K to avoid ties in voting
- **Rule of Thumb**
  - A common rule of thumb is to set K to the square root of the total number of samples in your dataset. This is a good starting point but may not always be the optimal choice.

How to choose the value of K

- **Domain Knowledge**
  - Depending on the characteristics of your dataset and problem domain, you may have insights that can guide you in choosing an appropriate value of K. For example, if you know that the decision boundaries are complex, you may want to choose a smaller value of K.

```
[ ] from sklearn.datasets import load_iris
    from sklearn.model_selection import train_test_split
    from sklearn.neighbors import KNeighborsClassifier
```

```
[ ] iris = load_iris()
    X, y = iris.data[:, 2:], iris.target
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3,

                                                         shuffle=True)
```

```
[ ] knn_model = KNeighborsClassifier(n_neighbors=3)
    knn_model.fit(X_train, y_train)

    y_pred = knn_model.predict(X_test)
```

```
[ ] num_correct_predictions = (y_pred == y_test).sum()
    accuracy = (num_correct_predictions / y_test.shape[0]) * 100

    # print('Test set accuracy: %.2f%%' % accuracy)

    print(f'Test set accuracy: {accuracy:.2f}%')
```

```
Test set accuracy: 95.56%
```

```python
mean_scores = []
for k in range (1,11) :
  knn_model = KNeighborsClassifier(n_neighbors = k)
  knn_model.fit(X_train, y_train)

  scores = cross_val_score(knn_model, X_train, y_train, cv = 5)
  mean_scores.append(scores.mean())

ck = np.argmax(mean_scores)
print(ck)
knn_model = KNeighborsClassifier(n_neighbors = ck+1)
knn_model.fit(X_train, y_train)

y_pred = knn_model.predict(X_test)

num_correct_predictions = (y_pred == y_test).sum()
accuracy = (num_correct_predictions / y_test.shape[0]) * 100

# print('Test set accuracy: %.2f%%' % accuracy)

print(f'Test set accuracy: {accuracy:.2f}%')
```

```
4
Test set accuracy: 97.78%
```