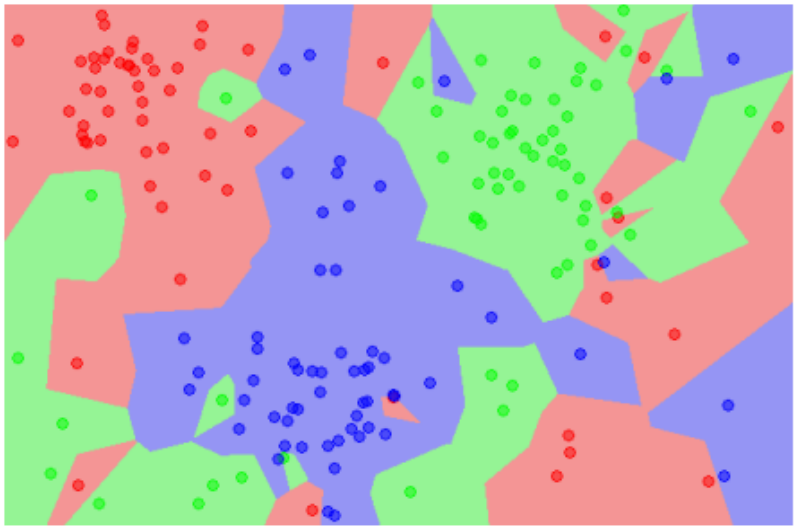
Introduction

K-nearest neighbor

- KNN is one the simplest supervised learning, it's also called **Instance-based Learning**
- KNN does not learn any pattern from data set, so that's why it's called **Lazy Learning**
- KNN will classify the new data point based on K points nearest this new point and **major voting** it



Hình 9.1: Ví dụ về 1NN. Các hình tròn là các điểm dữ liệu huấn luyện. Các hình khác màu thể hiện các lớp khác nhau. Các vùng nền thể hiện các điểm được phân loại vào lớp có màu tương ứng khi sử dụng 1NN (Nguồn: K-nearest neighbors algorithm – Wikipedia).

- Clearly see that there is green region in direction 11h between red and blue region, it's more likely to be noise, so it proves that KNN is very **sensitive to noise** especially when we decrease the K neighbors, consequently KNN is the example of Overfitting
- *Philosophy in ML*: When solving the problem in ML, Correct or Incorrect model does not exist, there is just model producing better result. So we just need model which can solve the problem, after that increase the quality of model

Math aspects

- Because KNN is lazy learning, so KNN does not need any cost function or optimization, any calculations will be done during adding new data points
- So we have 2 calculations to be defined
  - How to determine the distance among the data points: Use L2 Norm
  - With *Large-scale problem* (big data set, many features), suppose $X \in \mathbb{R}^{N \times d}$ (big *N* data points and big *d* features), KNN has to the distance between new data point $\mathbf{z} \in \mathbb{R}^{d}$ and all other data points, so here KNN needs to do too many calculations if KNN does not apply any efficient distance calculation. This is the big **disadvantage of Lazy Learning**
- How to find distance between *z* and x_i
  - Based on L2 Norm, we need $\|\mathbf{z}-\mathbf{x}_i\|_2$ but usually we ignore *root* and calculate $\|\mathbf{z}-\mathbf{x}_i\|_2^2$, but we need to modify $\|\mathbf{z}-\mathbf{x}_i\|_2^2$ a little bit

$$\|\mathbf{z}-\mathbf{x}_i\|_2^2 = (\mathbf{z}-\mathbf{x}_i)^T(\mathbf{z}-\mathbf{x}_i) = \|\mathbf{z}\|_2^2 + \|\mathbf{x}_i\|_2^2 - 2\mathbf{x}_i^T\mathbf{z}$$

  - Our purpose is just to find which number k of x_i nearest to *z*, so $\|\mathbf{z}\|_2^2$ can be ignored. When having any new data points or when we initialize the data points, we can calculate $\|\mathbf{x}_i\|_2^2$ at this time, so we don't need to care about $\|\mathbf{x}_i\|_2^2$ when calculating distance, so now we just need to calculate $\mathbf{x}_i^T\mathbf{z}$
- Find the distance between Matrix of new data points *Z* and Matrix *X*
  - We usually need to calculate the distance between each point in Matrix *Z* and every point in Matrix *X*. Basically, it's same with the principle of finding distance above, but here we can use Vectorization or Tools: *cdist* in *scipy.spatial.distance* or *pairwise_distances* in *sklearn.metrics.pairwise*

Evaluation

- Advantage
  - Don't need to learn any pattern from model

- Don’t need to assume the independence between data points
- Drawbacks
  - Overfitting (robust to noise) especially when K small
  - Complex when calculating the distance to all points and find K nearest point