Classification

Herman Kamper

http://www.kamperh.com/

Classification

From regression to classification:

- Regression: Predict scalar output $y \in \mathbb{R}$ given input $\mathbf{x}$
- Classification: Predict categorical class label y given input $\mathbf{x}$

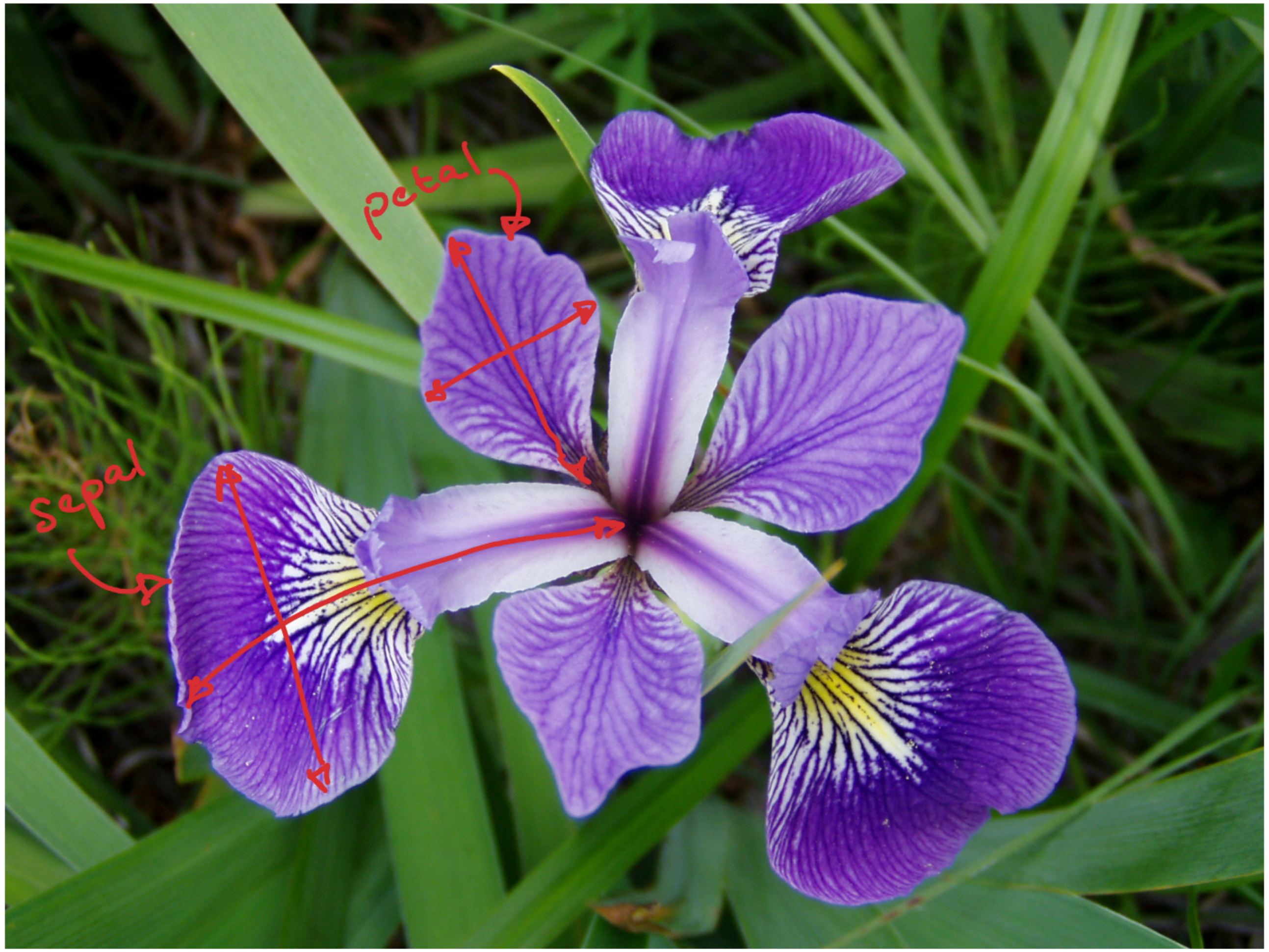
Examples:

- Disease diagnoses: Classifying whether a patient is healthy or not
- Text classification: Classifying documents according to topic
- Fault diagnoses: Is a photovoltaic system/antenna operating as expected or not?

Target output

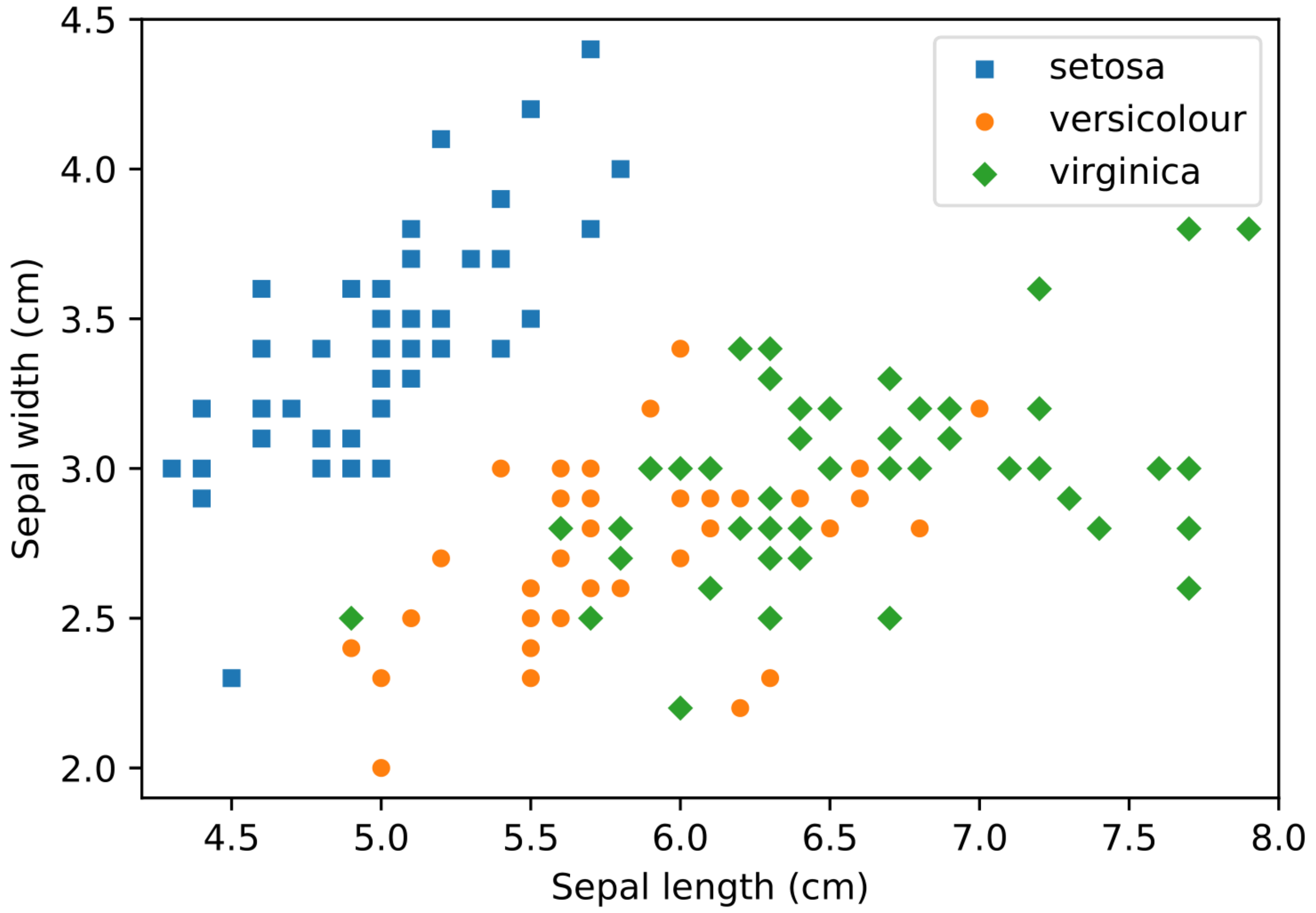
- Classification: Predict categorical class label y given input $\mathbf{x}$
- Data: In $\{(\mathbf{x}^{(n)}, y^{(n)})\}_{n=1}^{N}$, the label $y^{(n)}$ should tell us which class $\mathbf{x}^{(n)}$ belongs to
- There is a number of ways to encode y numerically
- Binary classification: $y \in \{0, 1\}$ or $y \in \{-1, 1\}$
- Classification among K classes: $y \in \{1, 2, \ldots, K\}$

Iris flower dataset



Source: `https://en.wikipedia.org/wiki/Iris_flower_data_set`

Iris dataset



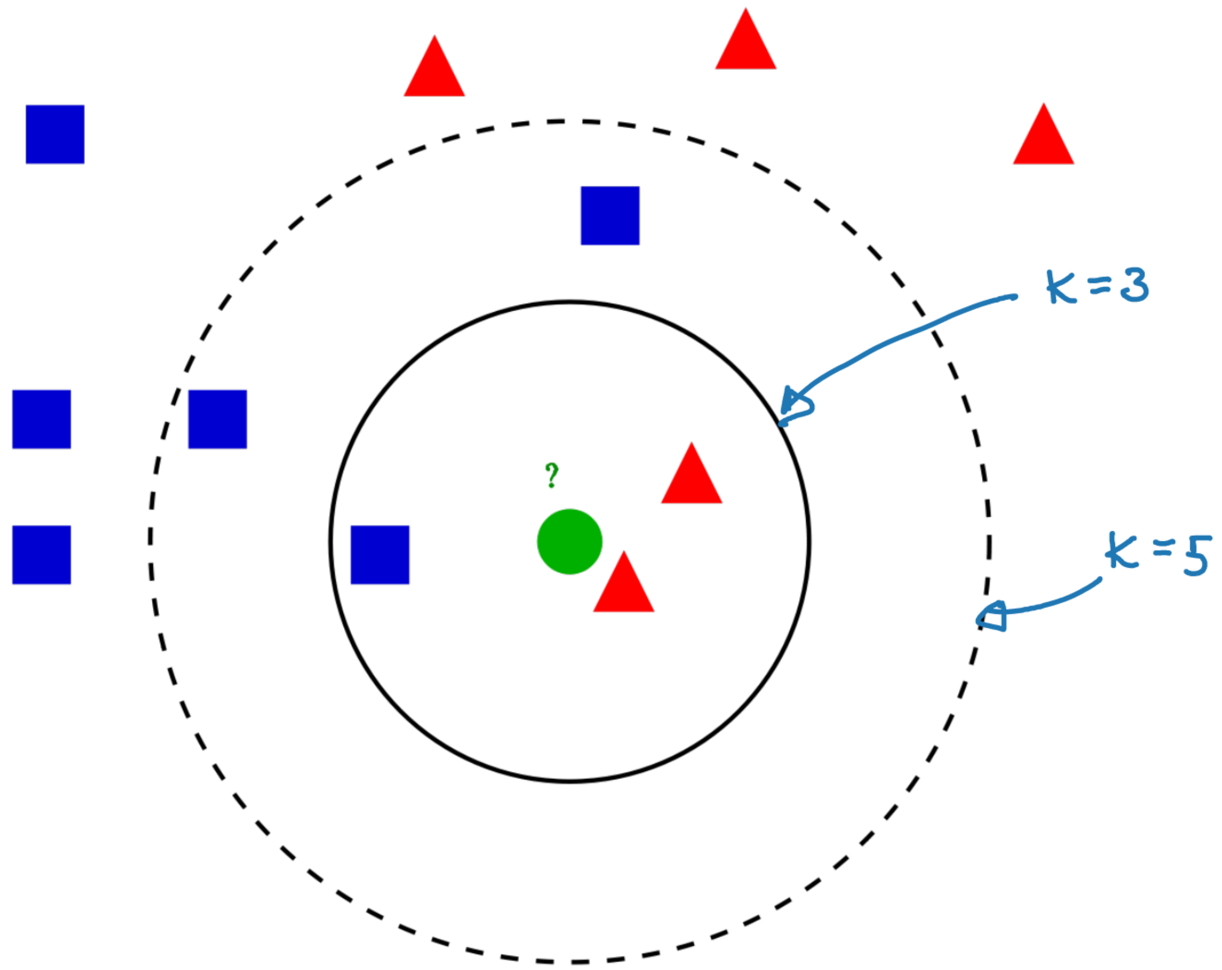
Classification

K-nearest neighbours

Herman Kamper

http://www.kamperh.com/

K-nearest neighbours (KNN)



Source: `https://commons.wikimedia.org/wiki/File:KnnClassification.svg`

K-nearest neighbours (KNN)

Algorithm:

- For a new test input $\mathbf{x}$, identify the K points in the training data closest to $\mathbf{x}$
- Predict the class of $\mathbf{x}$ as the label that occurs most often in the set $\mathcal{X}_K$ of closest points
- Can also get "soft" predictions, where the probability of $\mathbf{x}$ belonging to class k is given by:

$$P(y = k|\mathbf{x}) = \frac{1}{K} \sum_{n \in \mathcal{X}_K} \mathbb{I}(y^{(n)} = k)$$

with $\mathbb{I}$ the indicator function and $\mathcal{X}_K$ the set of indices of the nearest neighbours

$$\mathbb{I}(A) = \begin{cases} 1 & \text{if } A \text{ is true} \\ 0 & \text{if } A \text{ is false} \end{cases}$$

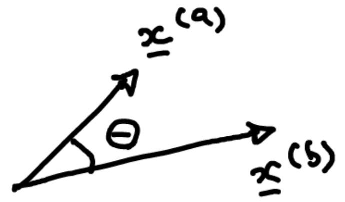
Choice of distance function:

- Euclidean:

$$d_{euclid}(\underline{x}^{(a)}, \underline{x}^{(b)}) = \sqrt{(x_1^{(a)} - x_1^{(b)})^2 + (x_2^{(a)} - x_2^{(b)})^2 + \dots + (x_D^{(a)} - x_D^{(b)})^2}$$
$$= \|\underline{x}^{(a)} - \underline{x}^{(b)}\|$$

- Cosine:

θ is angle between $\underline{x}^{(a)}$ and $\underline{x}^{(b)}$



$$d_{cos}(\underline{x}^{(a)}, \underline{x}^{(b)}) = 1 - \cos\theta$$
$$= 1 - \frac{\underline{x}^{(a)} \cdot \underline{x}^{(b)}}{\|\underline{x}^{(a)}\| \, \|\underline{x}^{(b)}\|}$$

K-nearest neighbours (KNN)

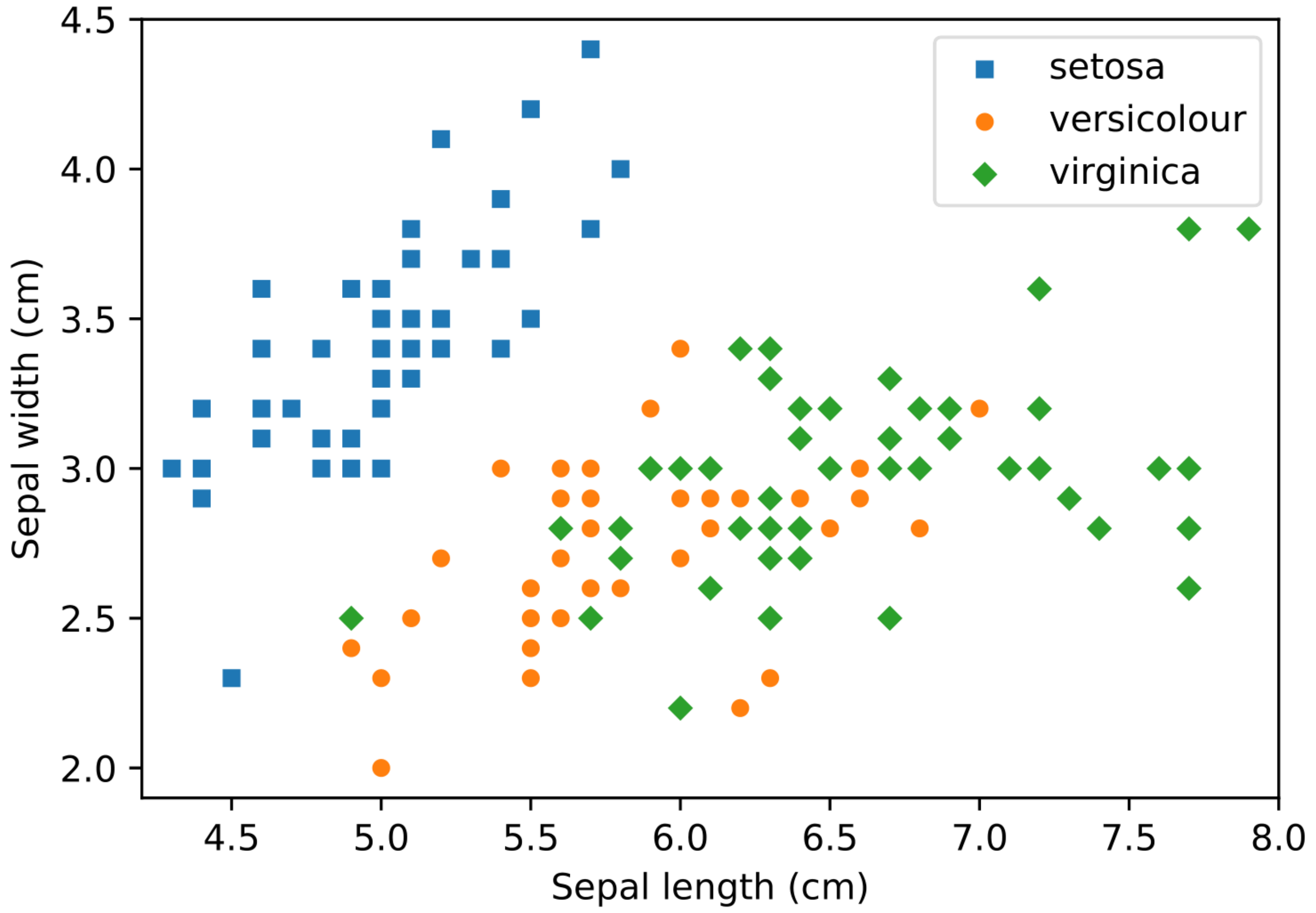
Problems with KNN:

- Computational complexity: To classify one point, need to run through entire dataset (issues when $N \gg$)
- Distance functions can be inaccurate (need to make some assumptions)
- Curse of dimensionality (issues when $D \gg$): Everything seems far away

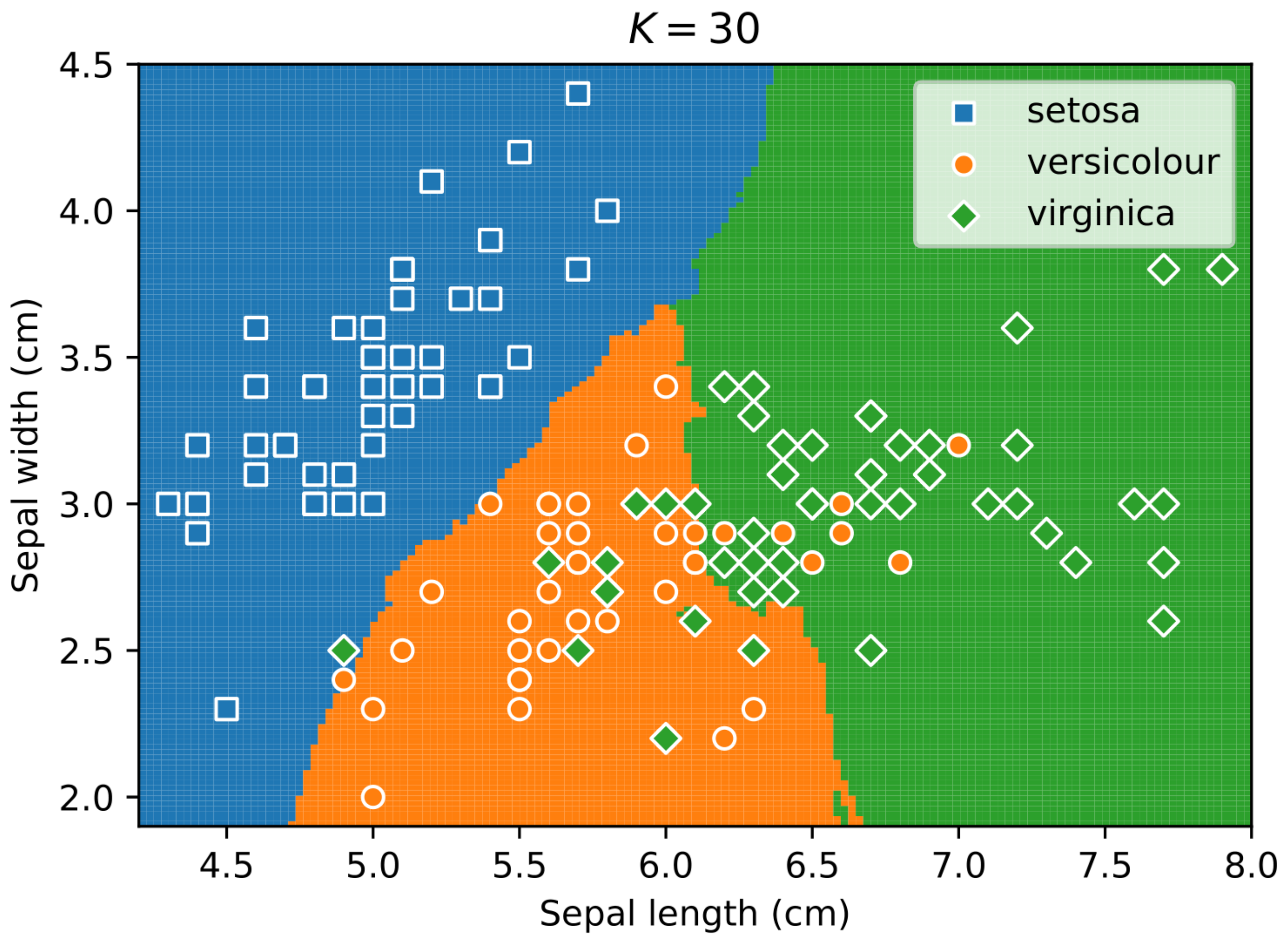
Terminology:

- KNN is a *non-parametric* classification approach
- It is an example of *memory-based* or *instance-based* learning

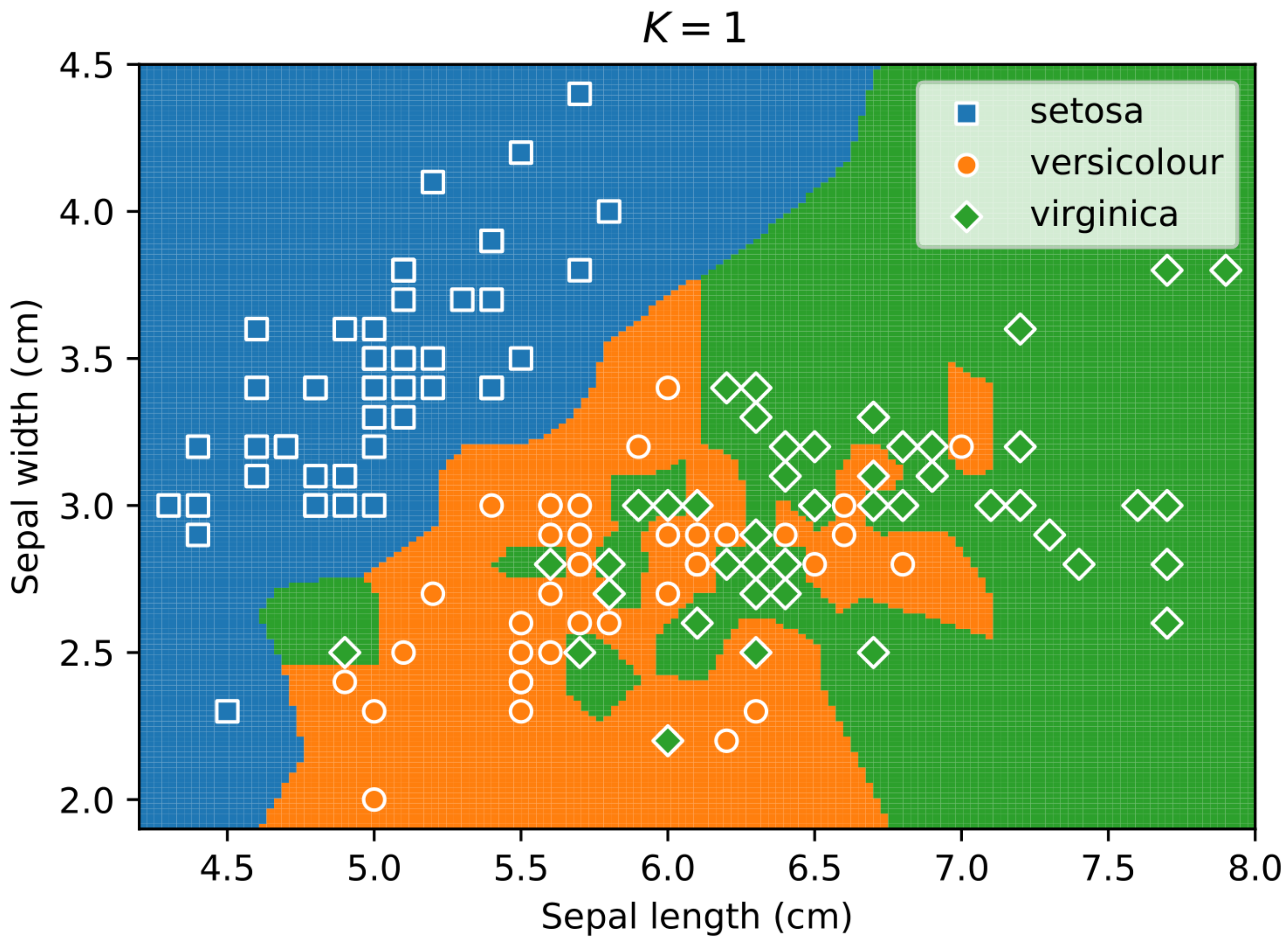
Iris dataset



K-nearest neighbours (KNN)



K-nearest neighbours (KNN)



Classification

Naive Bayes

Herman Kamper

`http://www.kamperh.com/`

The Bayes classifier

If we wanted to follow a probabilistic approach, we could use the following prediction model:

$$f(\mathbf{x};\boldsymbol{\theta}) = \arg\max_k P(y=k|\mathbf{x})$$

To use this model, we need to know $P(y=k|\mathbf{x})$. We can use Bayes' rule:

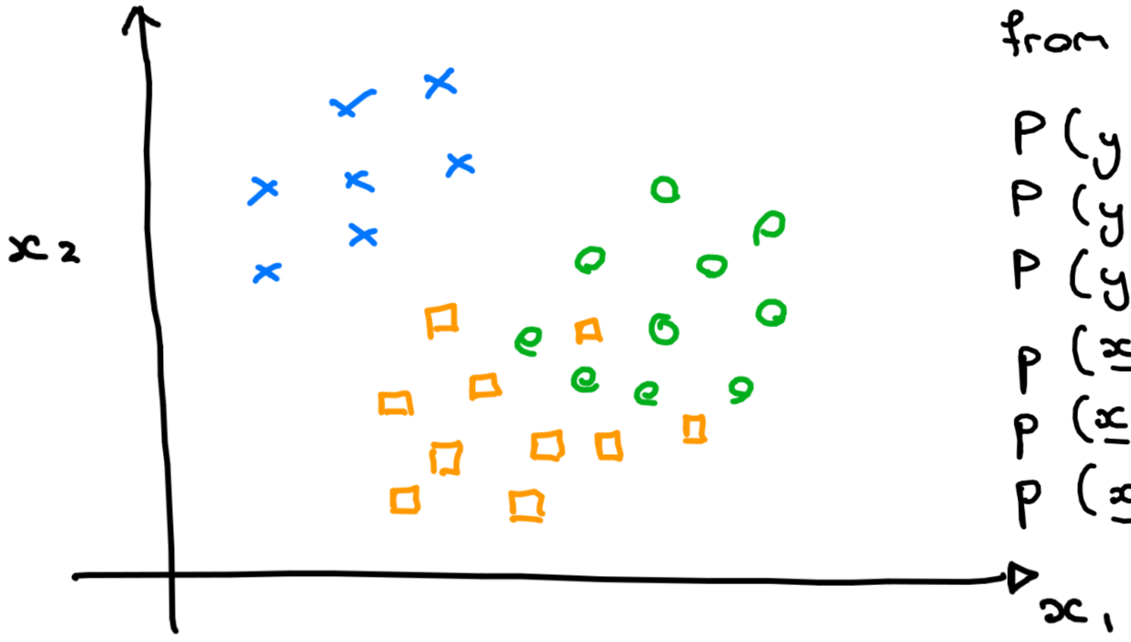
$$P(y=k|\mathbf{x}) = \frac{p(\mathbf{x}|y=k)P(y=k)}{p(\mathbf{x})}$$

Since $p(\mathbf{x})$ is the same for all k and we are only interested in the max, we can throw away the denominator:

$$P(y=k|\mathbf{x}) \propto p(\mathbf{x}|y=k)P(y=k)$$

This equation is very general. To actually use it, we need to decide on forms for $p(\mathbf{x}|y=k)$ and $P(y=k)$ and then figure out how we will learn their parameters $\boldsymbol{\theta}$ from the training data $\{(\mathbf{x}^{(n)}, y^{(n)})\}_{n=1}^N$. $P(y=k|\underline{x};\underline{\theta}) \propto p(\underline{x}|y=k;\underline{\theta}) \cdot P(y=k;\underline{\theta})$

Bayes classifier : Intuitively



$$P(y=k \mid \underline{x}) \propto p(\underline{x} \mid y=k) \cdot P(y=k)$$

Need to fit $P(y=k)$ and $p(\underline{x} \mid y=k)$ from the data. I.e., need

$$\left.\begin{array}{l} P(y=\times) \\ P(y=\circ) \\ P(y=\square) \\ p(\underline{x} \mid y=\times) \\ p(\underline{x} \mid y=\circ) \\ p(\underline{x} \mid y=\square) \end{array}\right\}$$

How would you choose these, intuitively?

Quadratic and linear discriminant analysis

For $P(y = k) = \pi_k$, a common approach is to simply count the number of training points assigned to class k:

$$\hat{\pi}_k = \frac{\sum_{n=1}^{N} \mathbb{I}(y^{(n)} = k)}{N}$$

We could decide that for each class we use

$$p(\mathbf{x}|y = k; \boldsymbol{\theta}) = \mathcal{N}(\mathbf{x}; \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)$$

and then set $\boldsymbol{\mu}_k$ and $\boldsymbol{\Sigma}_k$ to the MLE for each class. This is called *quadratic discriminant analysis* (QDA).

$$\underline{\Theta} = \{(\underbrace{\underline{\mu}_k}_{D\times 1}, \underbrace{\underline{\Sigma}_k}_{D\times D})\}_{k=1}^{K}$$

This could be problematic, though. If the dimensionality D is high and we have few training points N, there might not be enough data to estimate $\{(\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)\}_{k=1}^{K}$. Each $\boldsymbol{\Sigma}_k$ is a $D \times D$ matrix, so there can be many parameters!

We could make the assumption that all classes share the same covariance matrix $\boldsymbol{\Sigma}$ and then only fit $\{\boldsymbol{\mu}_k\}_{k=1}^{K}$, giving us more data to fit the single $\boldsymbol{\Sigma}$. This is called *linear discriminant analysis* (LDA).

The *naive Bayes* assumption goes even further!

(Gaussian) naive Bayes

In naive Bayes, we assume that each feature is independent, i.e. that each dimension of $\mathbf{x}$ is independent:

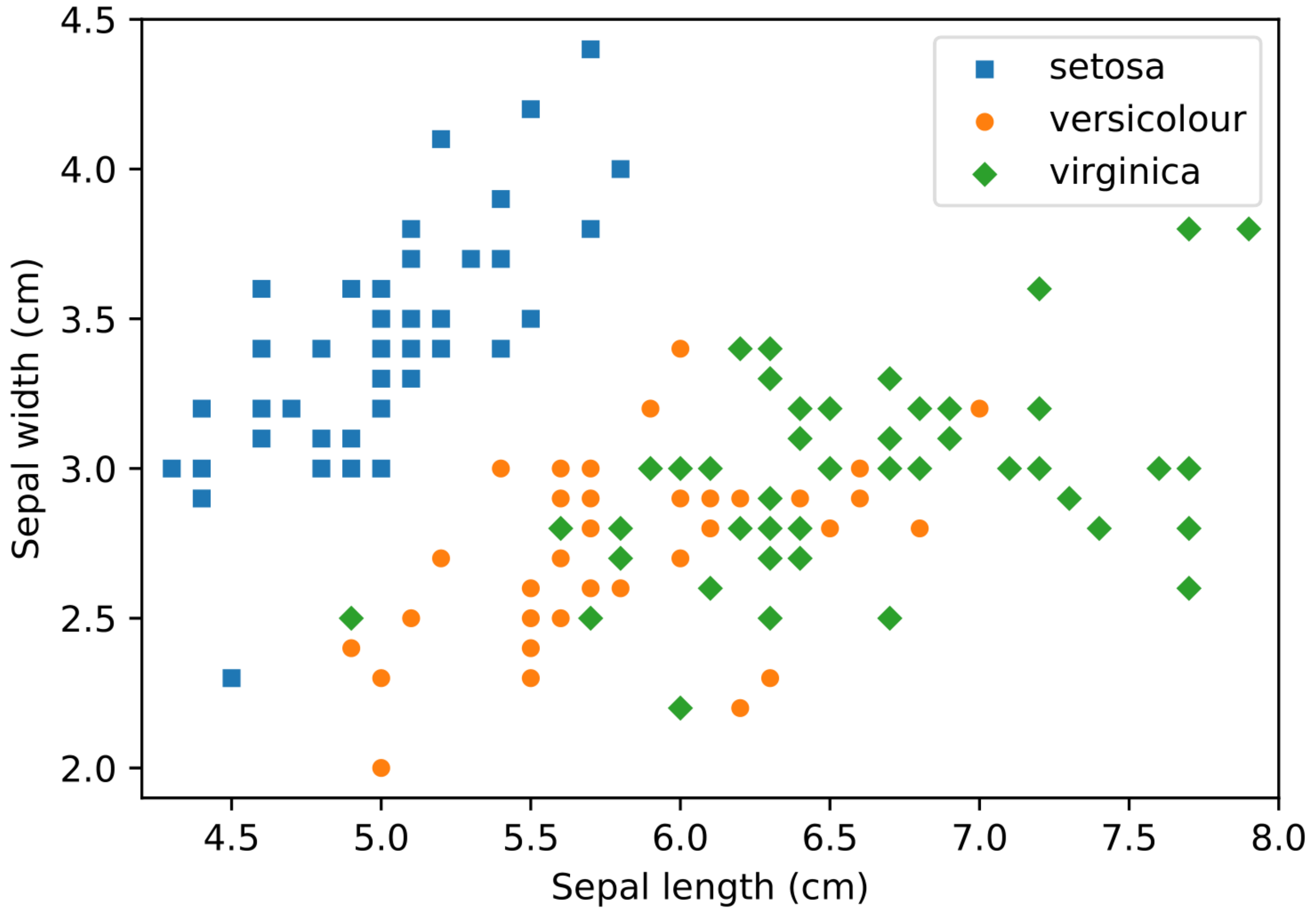
$$p(\mathbf{x}|y = k; \boldsymbol{\theta}) = \prod_{d=1}^{D} p(x_d|y = k; \boldsymbol{\theta})$$

The naive Bayes assumption can be made for any distribution, not just Gaussians. For the Gaussian case, it leads to

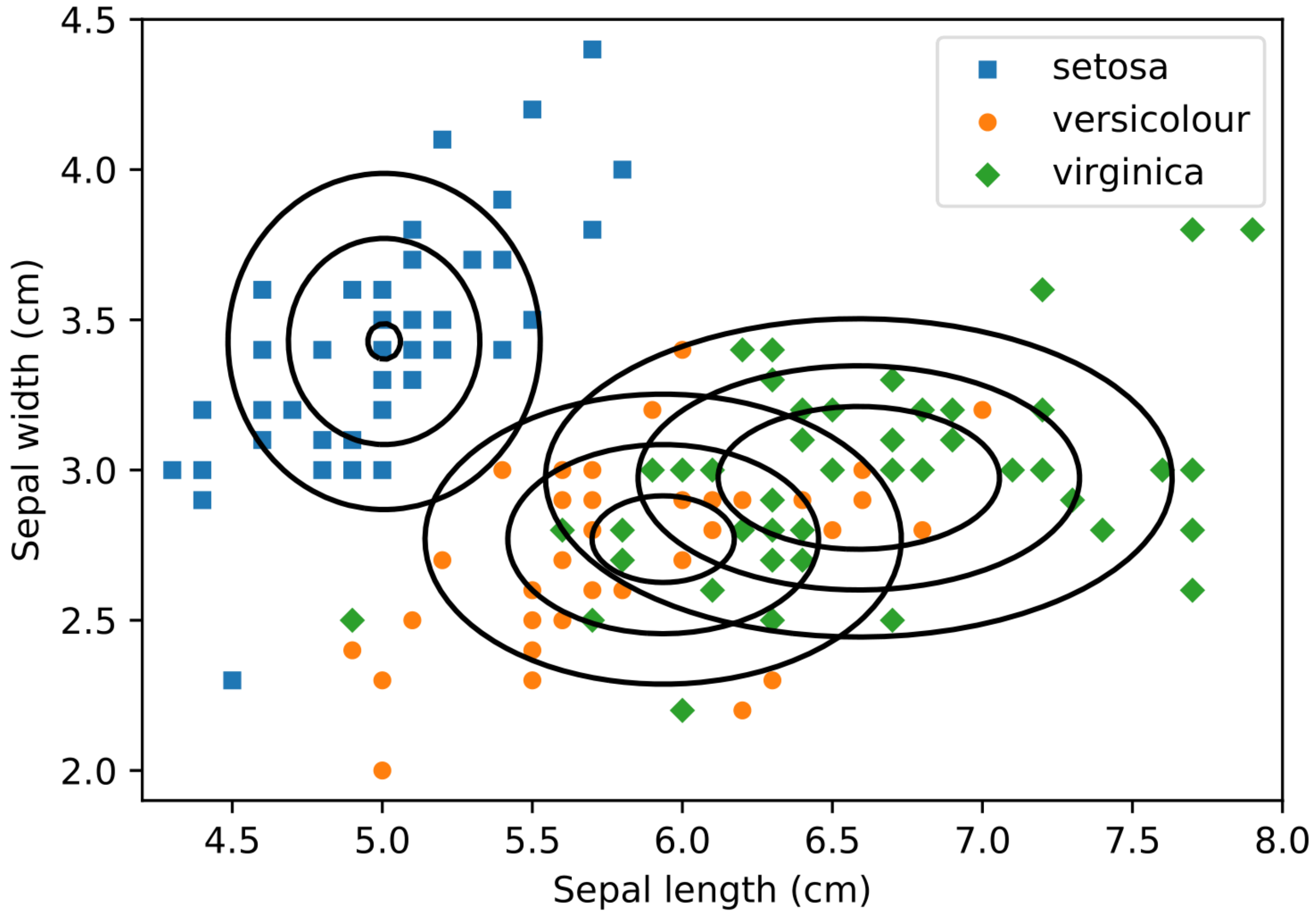
$$p(\mathbf{x}|y = k; \boldsymbol{\theta}) = \prod_{d=1}^{D} \mathcal{N}(x_d; \mu_{k,d}, \sigma^2_{k,d})$$

where the set of parameters $\boldsymbol{\theta}$ are all the means and variances. This can easily be fit using the MLE for each of the D univariate Gaussians for each of the K classes, i.e. we will have to fit $D \cdot K$ univariate Gaussians.

Iris dataset

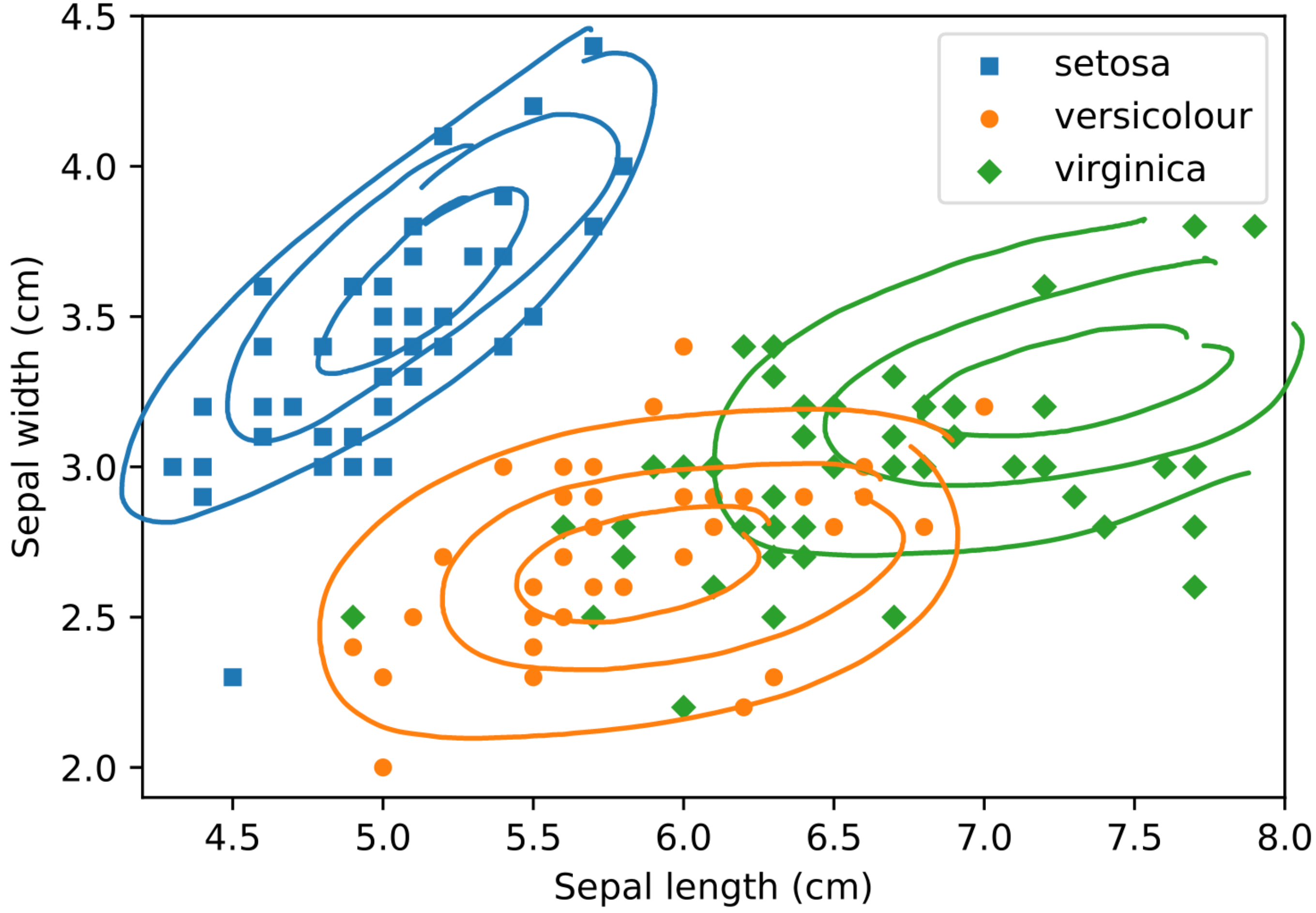


Gaussian Naive Bayes



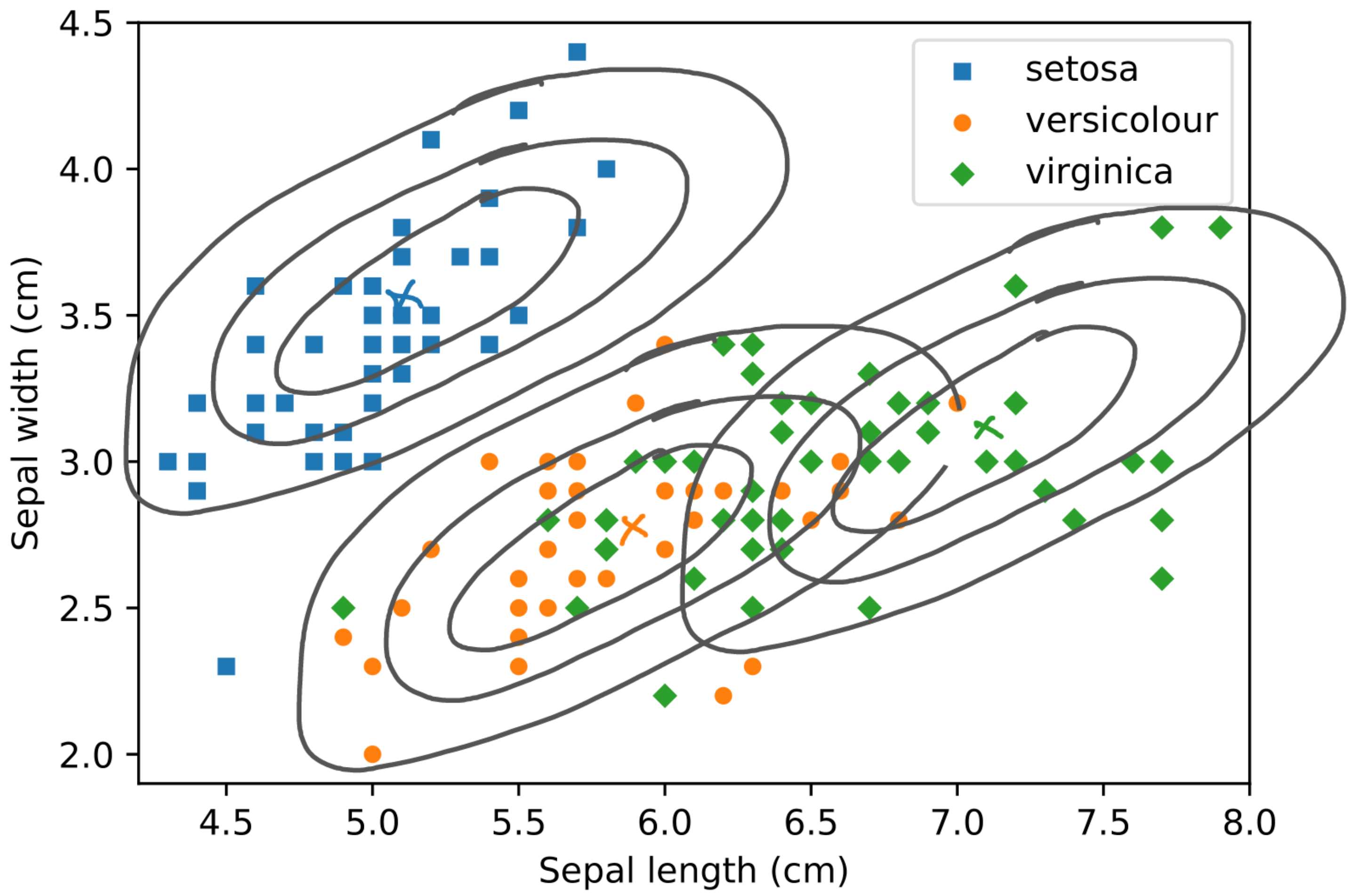
Iris dataset

QDA

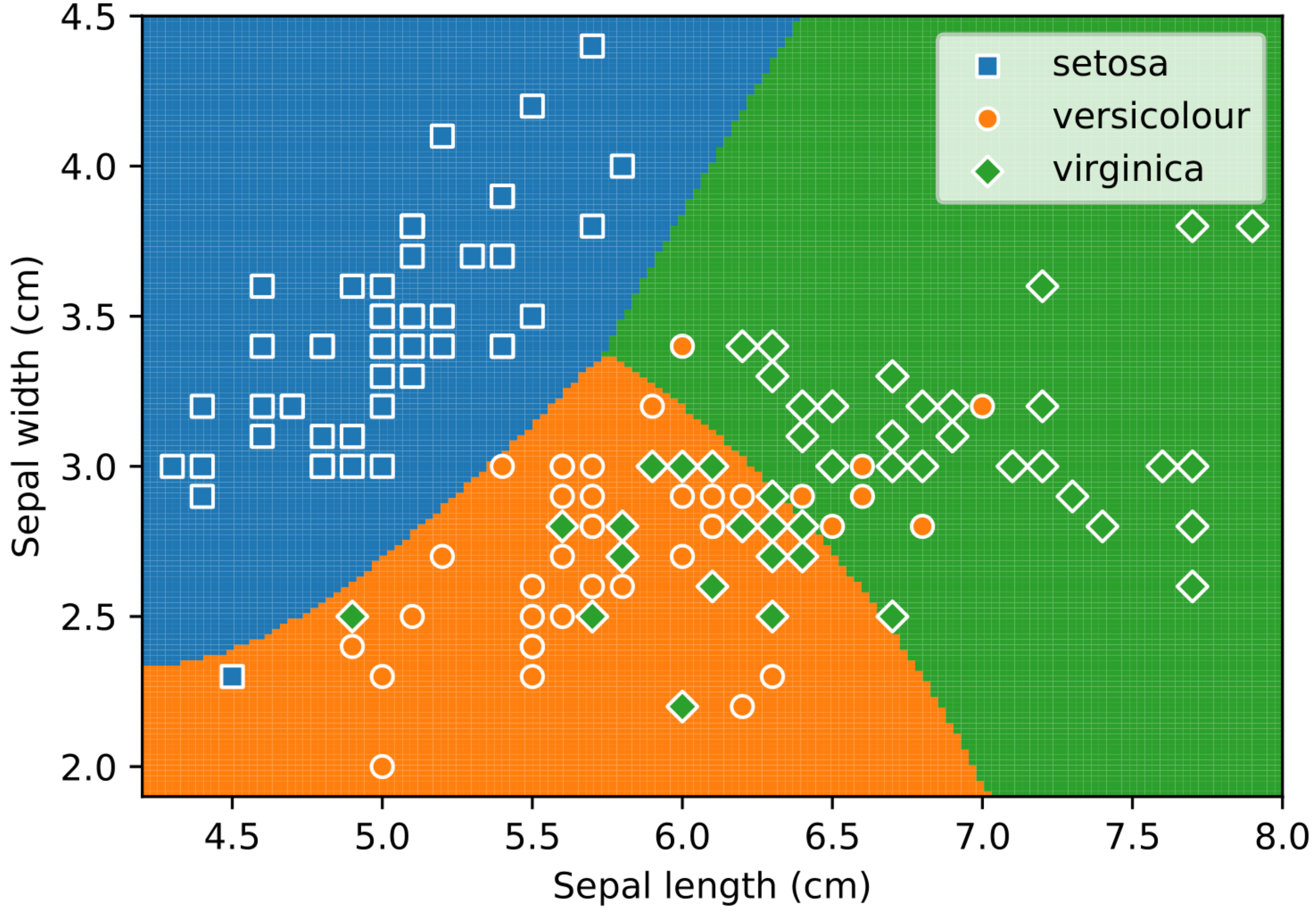


Iris dataset

LDA



Gaussian Naive Bayes



Classification

Generative vs discriminative

Herman Kamper

http://www.kamperh.com/

Generative and discriminative models

Generative models:

- Bayes classifier: $P(y = k|\mathbf{x}) \propto p(\mathbf{x}|y = k)P(y = k)$
- Choose forms for $p(\mathbf{x}|y = k)$ and $P(y = k)$ and learn from data
- Referred to as *generative*, since we can generate data: first sample class from $P(y)$ and then sample data from $p(\mathbf{x}|y =$ sampled class$)$
- But often we aren't actually interested in generating data: we want to classify!
- And might be tricky to model $p(\mathbf{x}|y = k)$ for each class

Discriminative models:

- Just model $P(y = k|\mathbf{x})$ directly!
- Use training data $\{(\mathbf{x}^{(n)}, y^{(n)})\}_{n=1}^{N}$ to directly fit probability we are interested in

Towards logistic regression

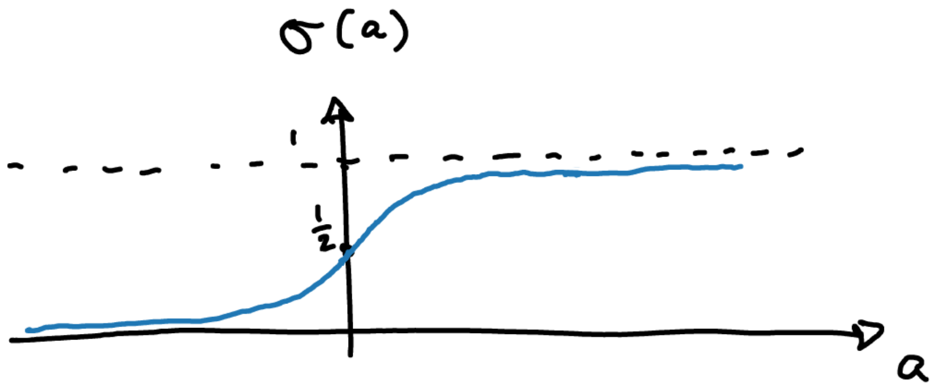
For binary classification, i.e. $y \in \{0, 1\}$, we could for instance use:

$$f(\mathbf{x}; \mathbf{w}) = \sigma(\mathbf{w}^\top \mathbf{x}) = \frac{1}{1 + e^{-\mathbf{w}^\top \mathbf{x}}}$$

to model $P(y = k | \mathbf{x})$.

signoid function

Sigmoid function:



$$\sigma(a) = \frac{1}{1 + e^{-a}}$$