

Lesson outline

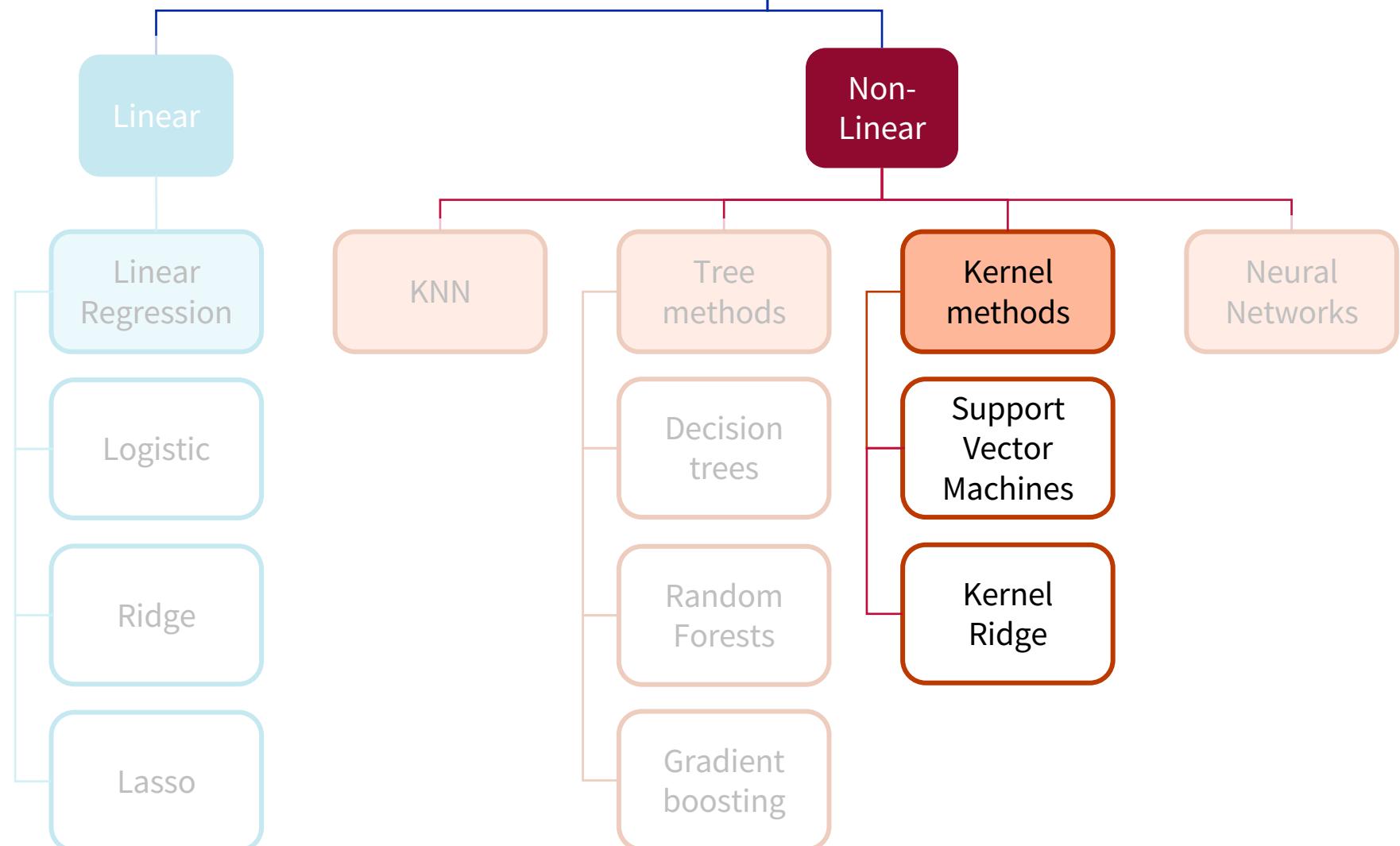
Evaluation Recap

Data leakage

Metrics

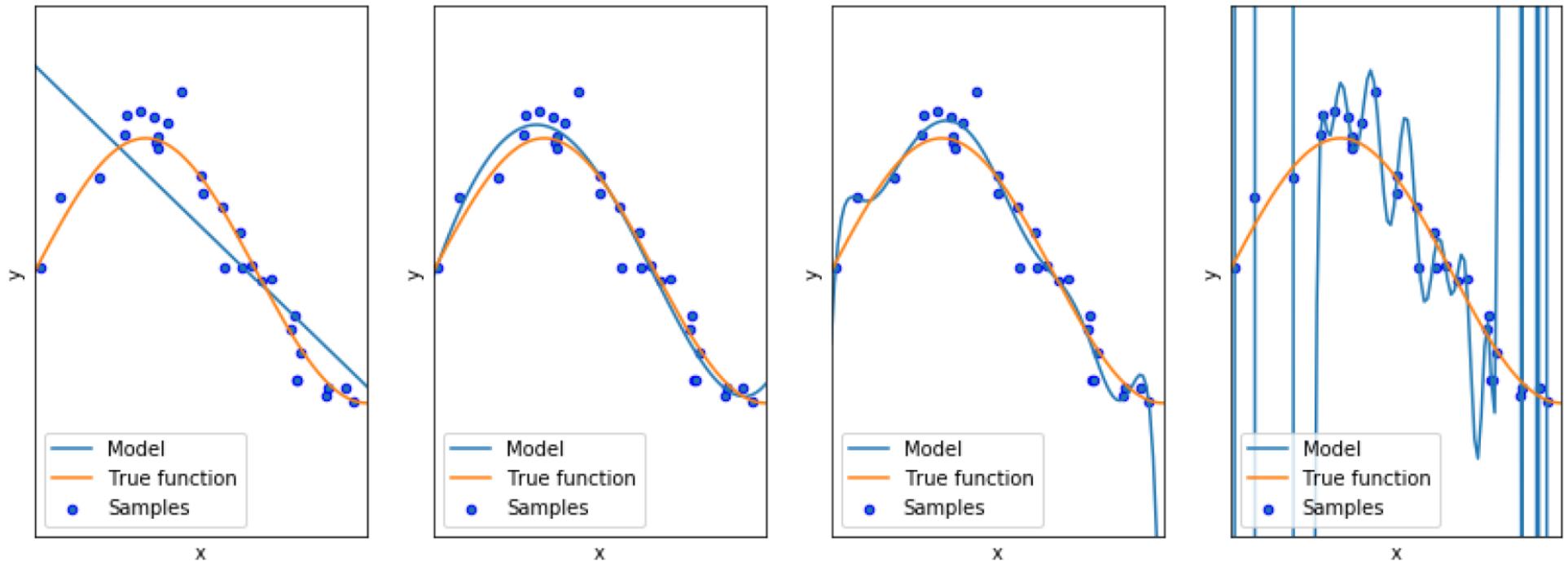
Support Vector Machines

Overview of Supervised ML algorithms



Evaluation: Recap

- Training error
- Generalisation error
- ERM
- Bias-variance tradeoff



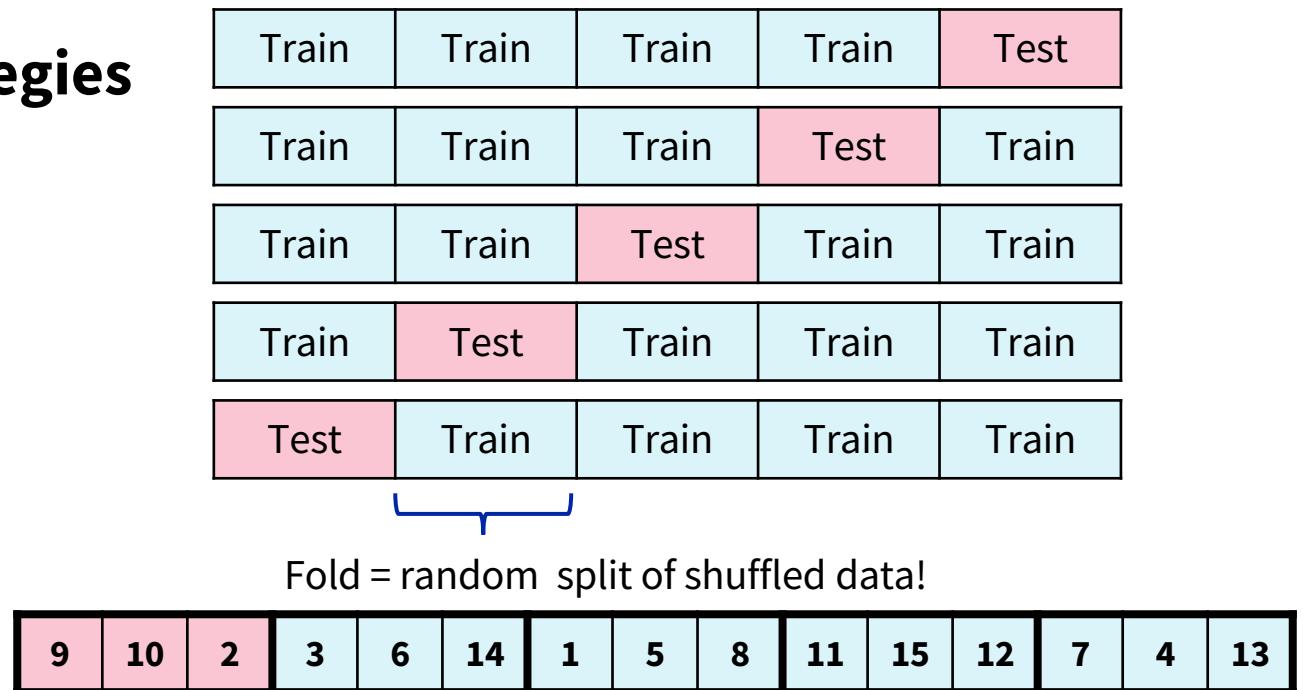
Picture from A. Schörgenhuber, *Hands-on AI I*, Lecture materials, 2023, JKU Linz.

Some remedies: Dataset Splitting Strategies

- Train/Test/Validation
- Cross validation: e.g. k-fold CV

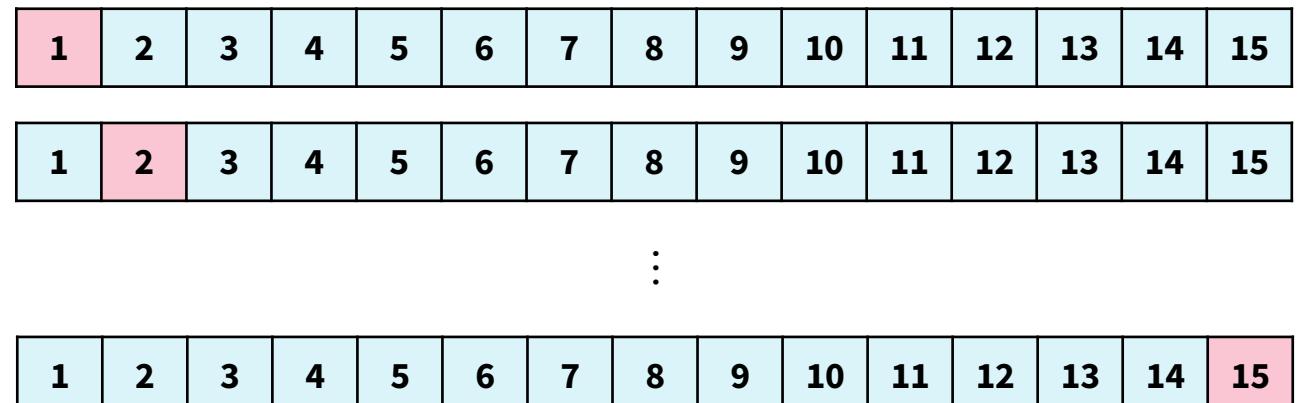
Other important CV methods:

- **Stratified k-fold**: response values close to equal in folds
- **Repeated k-fold**: repeat random splitting into k folds to average performance



Exhaustive CV methods: all possible splits explored

- **Leave one out (LOOCV)**: C_1^n (binomial coefficient)
- **Leave p out (LpOCV)**, e.g. $p=2$ as almost unbiased method (but e.g. $C_2^{15} = \binom{15}{2} = 105$ possibilities...)



Risk: Data leakage

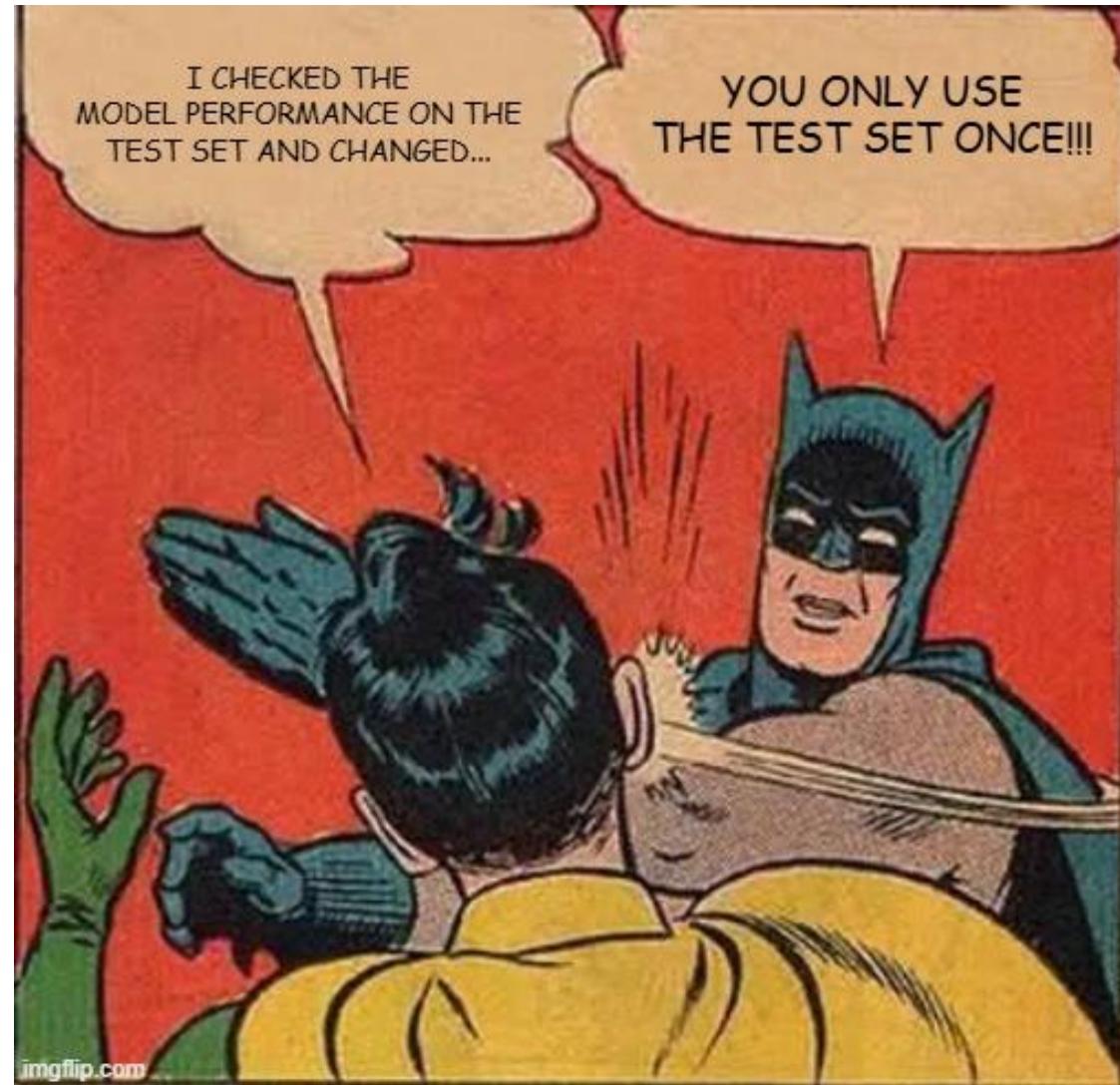
Data outside of the training data influences the model training!

Result: **optimistic performance** in validation **but poor generalisation!**

- Train-test contaminations, e.g. scaling on whole dataset, (partial) duplicated data in train and test splits
- Target leakage, e.g. model has access to features that encode the target
- Temporal leakage, e.g. random splits of time-series data
- CV leakage, e.g. reusing fitted transformers across folds
- Preprocessing leakage, e.g. transformation (PCA, normalisation, ...) done before splitting

Good practice to avoid insufficient isolation:

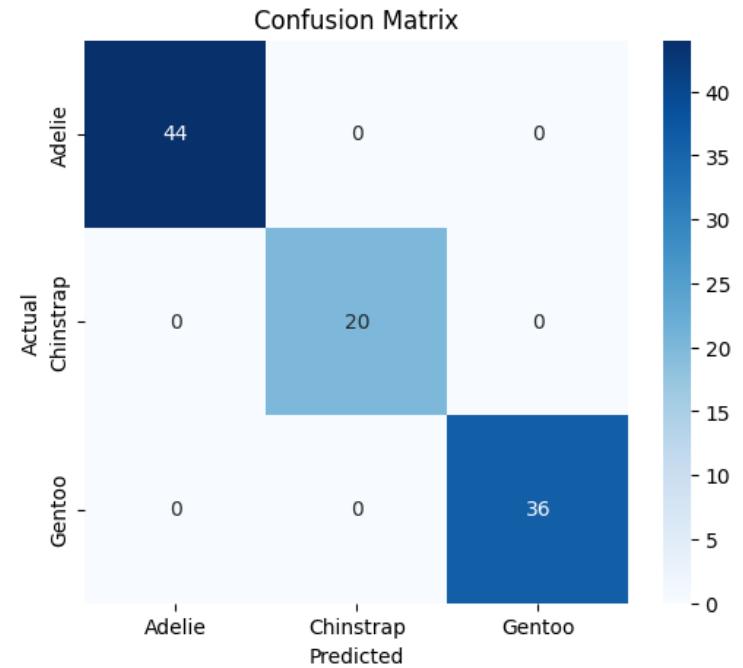
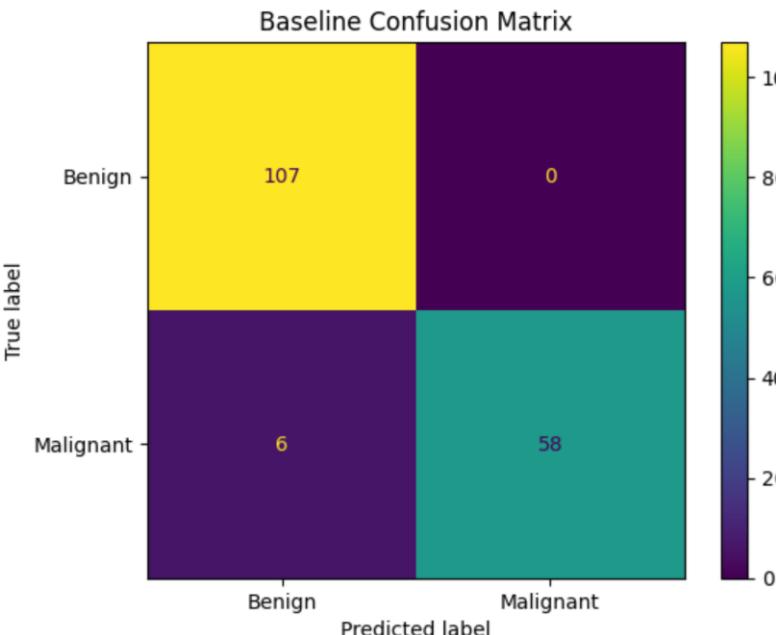
- Keep strict separation: training, validation, testing!
- Perform transformations post-splitting
- Use pipelines to avoid “manual” mistakes



Classification Metrics

Confusion matrix:

- **Diagonal:** Correct predictions, e.g. true positives (**TP**) and true negatives (**TN**)
- **Off-diagonal:** Misclassifications, e.g. False positives (**FP**) and false negatives (**FN**)



Basis for further metrics, e.g.
accuracy:

- Binary classification:

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

- Generally (N x N):

$$\text{Accuracy} = \frac{\sum_i^N M_{ii}}{\sum_i^N \sum_j^N M_{ij}}$$

```
ConfusionMatrixDisplay.from_estimator(rf_default, X_test, y_test)
plt.title("Baseline Confusion Matrix")
plt.show()
```

```
cm = confusion_matrix(y_test, y_test_pred)

plt.figure(figsize=(6,5))
sns.heatmap(cm, annot=True, fmt="d",
            xticklabels=knn.classes_,
            yticklabels=knn.classes_,
            cmap="Blues")

plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.title("Confusion Matrix")
plt.show()
```

Classification Metrics

Scores based on confusion matrix:

- **Accuracy:** How often the model is correct
- **Precision:** How many predicted positives were actually positive
- **Recall (Sensitivity, TP rate (TPR):** How many actual positives did the model catch?
- **Specificity (TNR):** How many actual negatives did the model catch?
- **F1-Score:** Harmonic mean of precision & recall

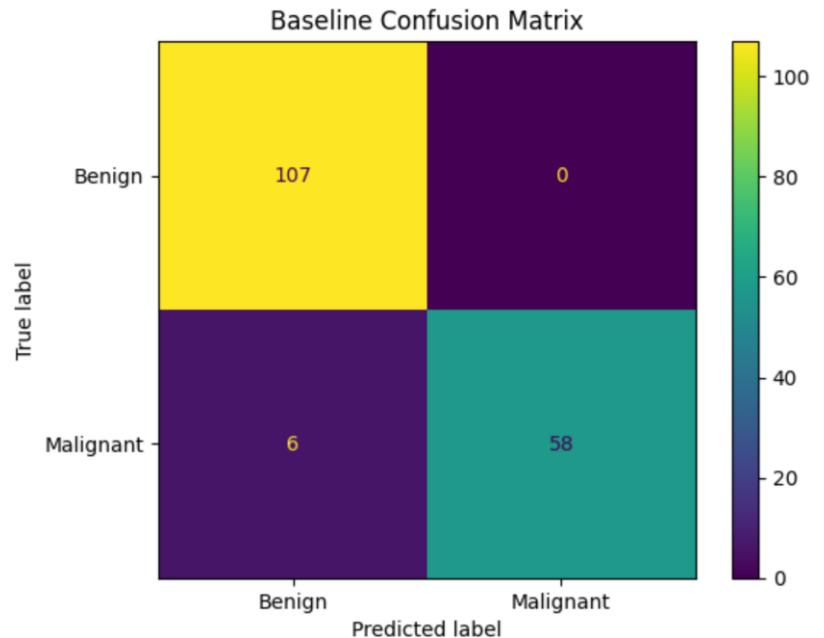
$$\frac{TP + TN}{TP + TN + FP + FN}$$

$$\frac{TP}{TP + FP}$$

$$TPR = \frac{TP}{TP + FN}$$

$$TNR = \frac{TN}{TN + FP}$$

$$2 \cdot \frac{precision \cdot recall}{precision + recall}$$



Classification Metrics: ROC & AUC

ROC (Receiver Operating Characteristics)

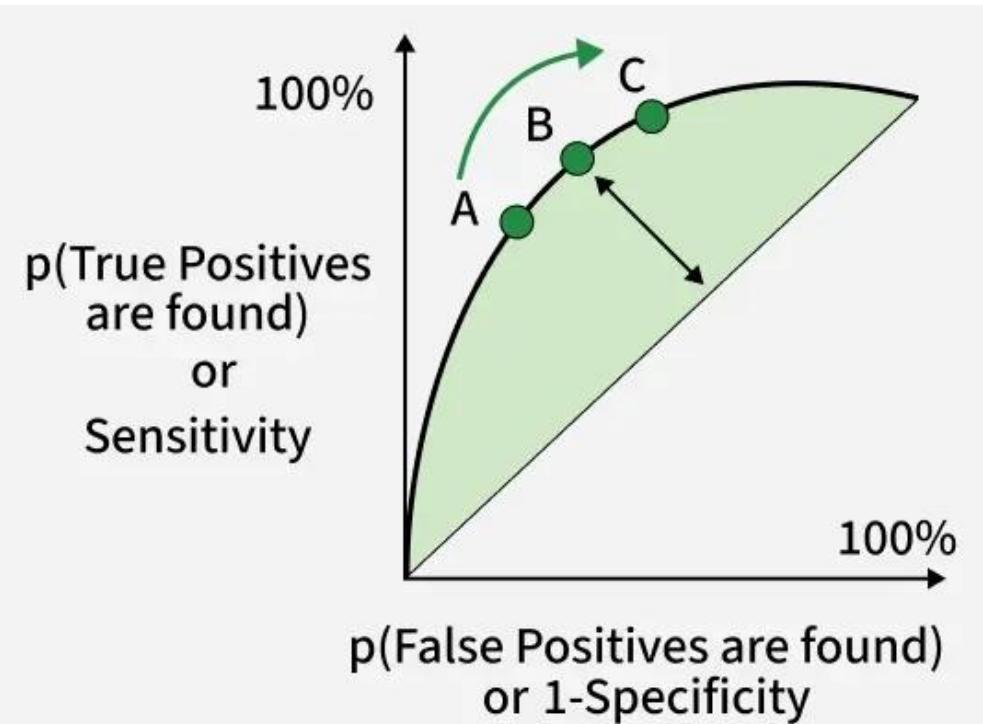
- ROC curve = plot of Recall (TPR) vs. FP rate (FPR = 1-Specificity (TNR))
- Essentially plots the trade-off **Sensitivity vs. Specificity** of a **Binary Classifier**
- **The higher the curve the better.**
- **Each point represent threshold** (primary output of model = score, not class!)

ROC-AUC: area under the ROC curve

- 1 = entire graph, indicates perfect prediction
- 0.5 = diagonal = random guessing (50/50)
- Measures ranking **quality (independent of threshold)**, not classification accuracy!

$$TPR = \frac{TP}{TP + FN}$$

$$FPR = \frac{FP}{FP + TN} = 1 - TNR$$



<https://www.geeksforgeeks.org/machine-learning/auc-roc-curve/>

Regression Metrics

Distinction according to how errors are penalised, interpretable they are, sensitive they are to outliers.

— MAE (mean absolute error, L1)

- High interpretability (same unit)
- Linear penalty for errors (More robust to outliers)

$$MAE = \frac{1}{n} \cdot \sum_{i=1}^n |y_i - \hat{y}_i|$$

— MSE (mean squared error, L2)

- Low interpretability (squared unit)
- high penalty for outliers (less robust)

$$MSE = \frac{1}{n} \cdot \sum_{i=1}^n (y_i - \hat{y}_i)^2$$

— RMSE (root MSE)

- Still high penalty for outliers (not more robust)
- But more interpretable (same unit)

$$RMSE = \sqrt{MSE}$$

— R²

- Scale-free (no unit)
- Error penalty indirectly high (via MSE)
- Not an error magnitude!
- Can be misleading!

$$R^2 = 1 - \frac{MSE}{Variance(y)}$$

ChatGPT's selection recommendation:

Use MAE when:

- You want robustness
- Errors should scale linearly
- Heavy-tailed noise expected

Use MSE/RMSE when:

- Large errors are unacceptable
- Gaussian noise assumption reasonable
- You want smooth optimisation

Use R² when:

- Comparing models on same dataset
- Communicating variance explained

Regression metrics

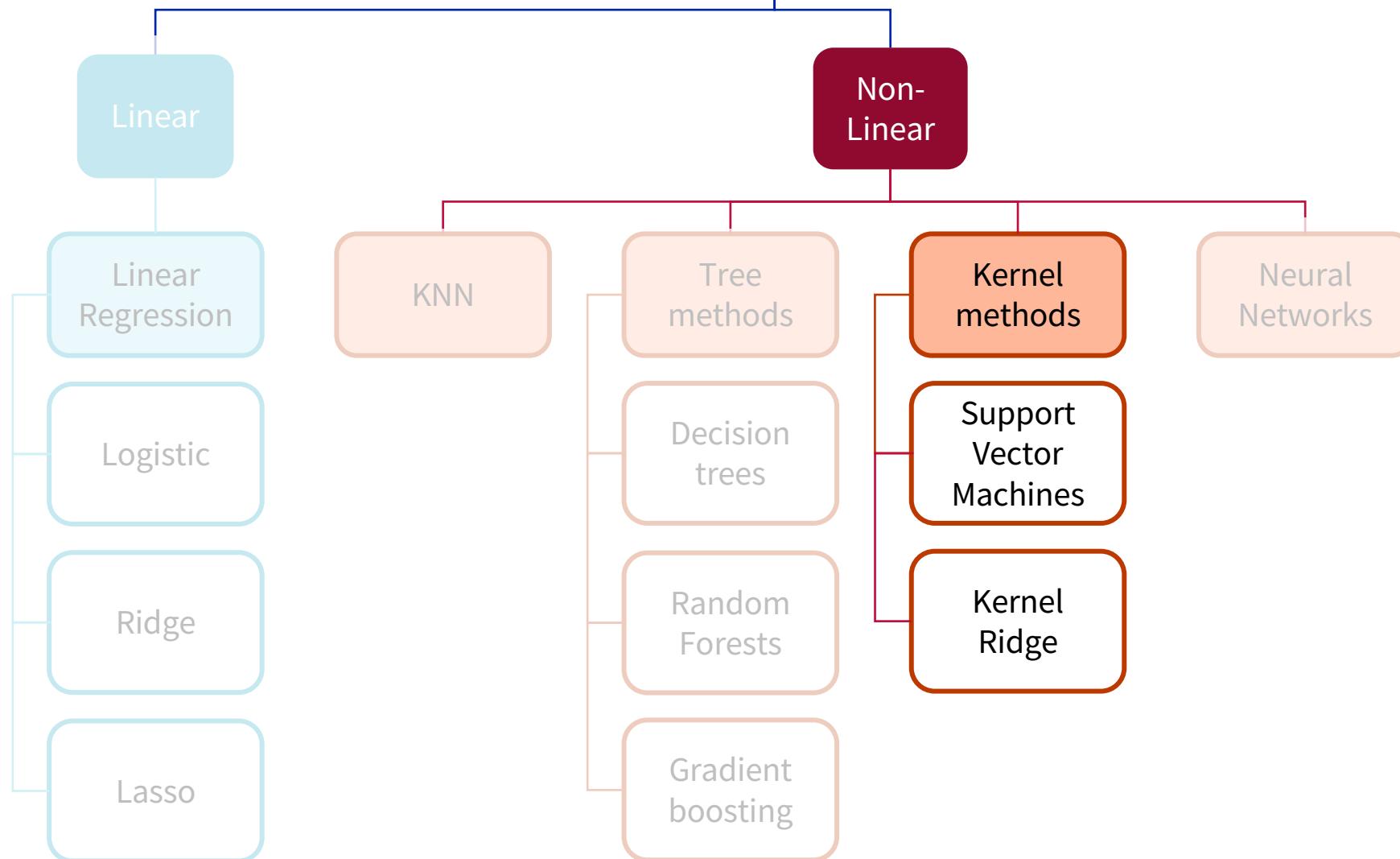
Consider the following minimal example:

[10, 10, 10, 10, 0] vs. predicted: [10, 10, 10, 10, 100]

- Calculate MAE, MSE, RMSE
- Discuss how these would affect training the model, i.e. what would the model try to do?
- Would you rather have one larger outlier and 4 perfect values, or 5 erroneous values?
- How is the noise distributed? Is MSE a valid metric in this case?
- Imagine a context. Is a 100-unit mistake catastrophic? Would you rather have
- When can you “ignore” a large error?

Next on the ML menu:

Overview of Supervised ML algorithms

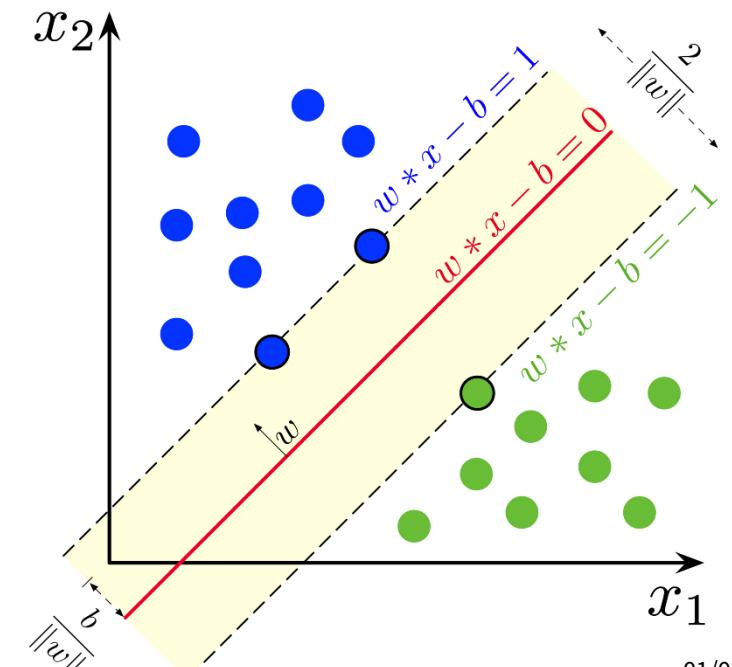
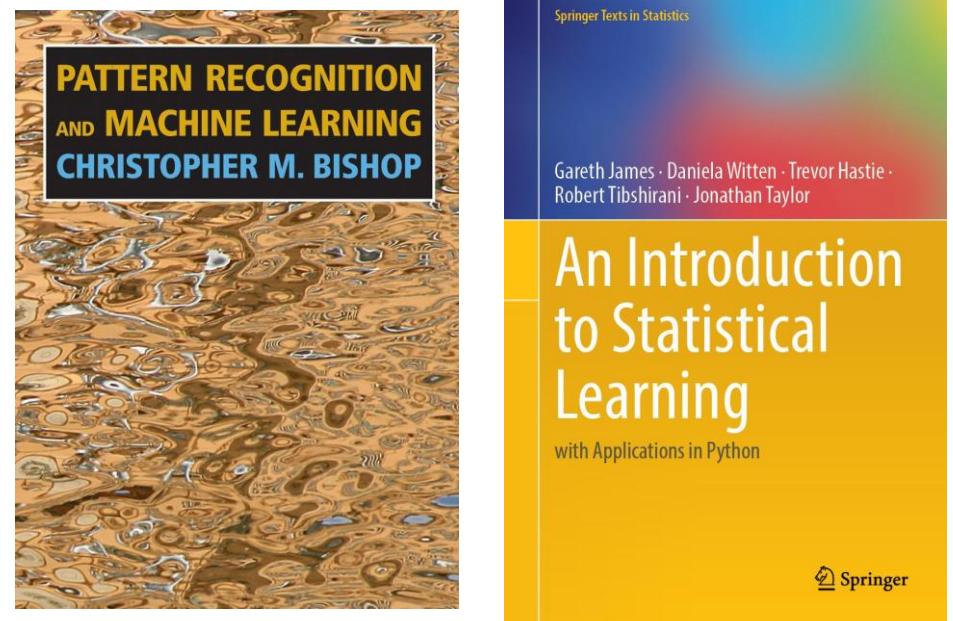


Support Vector Machines (SVM)

Supervised ML algorithm for classification (**SVC**) and regression (**SVR**)

SVC - main idea:

- Find the best possible boundary (**hyperplane**) that separates data points of different classes
- Best boundary is the one that leaves the **largest margin** (i.e. distance) between itself and the closest data points.
- Those closest points = **support vectors**, “supporting” the position of the boundary
- SVMs work well with small-medium datasets, handle high dimensionality very well
- SVMs were dominant before deep learning, e.g. for image classification, semantic parsing (text), ...



Picture by Larhmam - Own work, CC BY-SA 4.0, <https://commons.wikimedia.org/w/index.php?curid=73710028>

Support Vector Classifier (SVC): Linear kernel

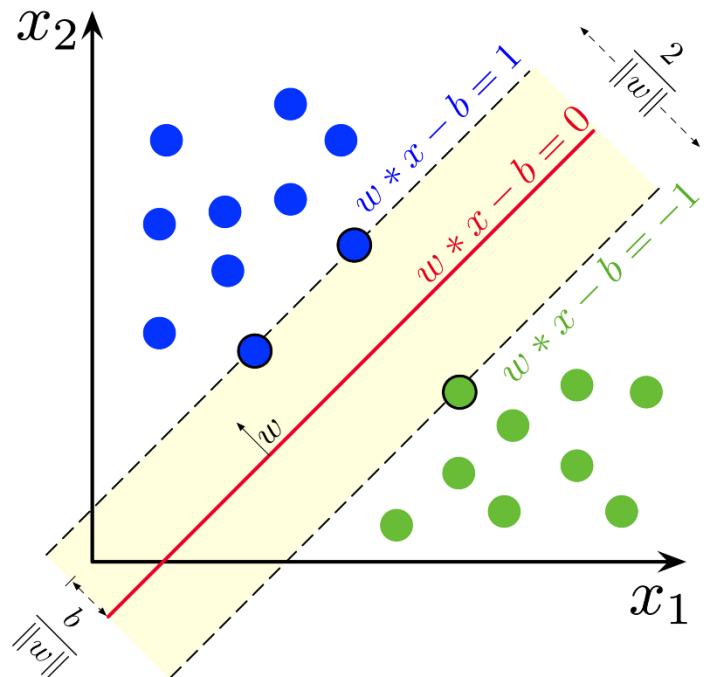
Maximum margin classifier

Hyperplane described as $(w^T x_i - b) = 0$

...where w is a column vector transposed to deliver a scalar as dot product with x_i

...and the **bias** b describes the offset of the hyperplane with regards to the origin ($\frac{b}{\|w\|}$)

Note: bias is often defined with the opposite sign so that $(w^T x_i + b) = 0$



The SVM function $f(x_i) = \text{sgn}(w^T x_i - b)$ delivers 1 and -1 (**sign function**)

... thus the margin is: $\frac{2}{\|w\|}$

Perfectly separable = Hard margin SVM

To maximise the margin means to minimise $\|w\|$, or rather $\frac{1}{2} \|w\|^2$, subject to $y_i(w^T x_i - b) \geq 1$

Picture by Larhmam - Own work, CC BY-SA 4.0, <https://commons.wikimedia.org/w/index.php?curid=73710028>

Support Vector Classifier (SVC): Linear kernel

Maximum margin classifier

More relevant: Soft-margin SVM

Hinge loss function:

$$\max(0, 1 - y_i(w^T x_i - b))$$

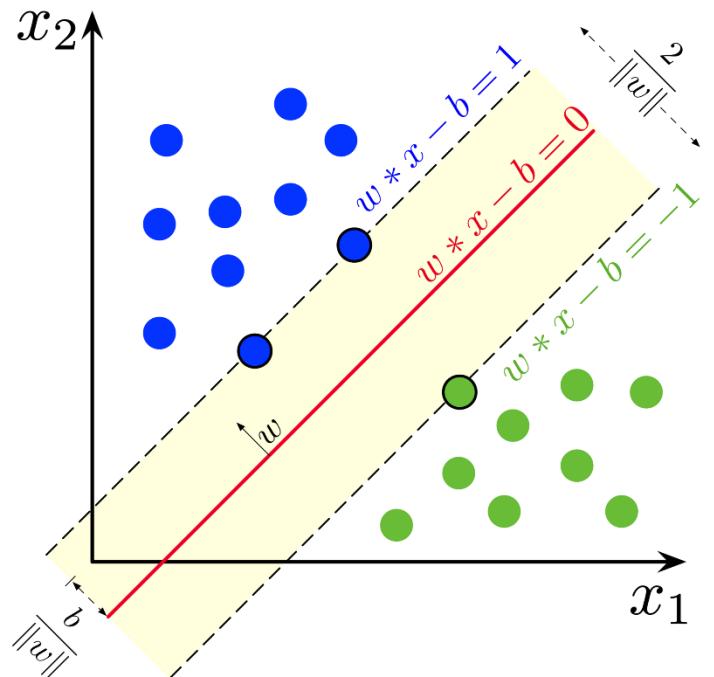
Hence minimise:

$$\|w\|^2 + C \frac{1}{n} \sum_{i=1}^n \max(0, 1 - y_i(w^T x_i - b)) = \|w\|^2 + C \sum_{i=1}^n \zeta_i$$

...with ζ_i accounting for the hinge loss (**slack variables**)

...and **regularisation parameter C** as trade-off between margin size and penalty for points within

But: what if the data is not linearly separable?



Picture by Larhmam - Own work, CC BY-SA 4.0, <https://commons.wikimedia.org/w/index.php?curid=73710028>

Support Vector Classifier (SVC): The kernel trick

Instead of non-linear separation: transform linear SVM into non-linear model!

Map data to higher dimensionality!

$$x \rightarrow \phi(x)$$

Linear model becomes:

$$f(x) = \text{sgn}(w^T \phi(x) - b)$$

But: Not practical to calculate $\phi(x)$ explicitly
(combinatorial growth of feature space up to infinite-dimensional!)

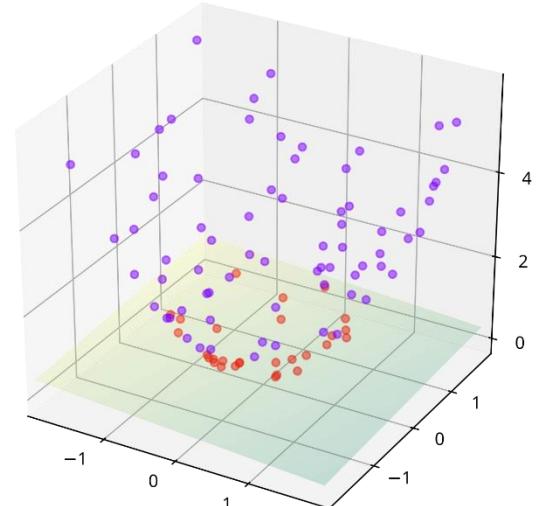
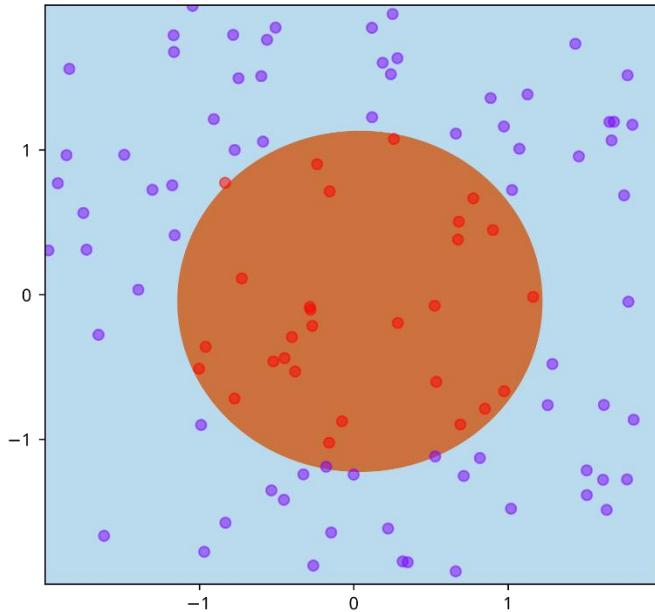


Image by Shiyu Ji - Own work, CC BY-SA 4.0, <https://commons.wikimedia.org/w/index.php?curid=60458994>

Support Vector Classifier (SVC): The kernel trick

SVM solution: Optimal vector of linear combination of training data

$$w = \sum_{i=1}^n \alpha_i x_i y_i$$

In the **dual form**, i.e. comparing x_i and x_j , when we want to minimize $\|w\|^2$:

$$\|w\|^2 = \left(\sum_{i=1}^n \alpha_i x_i y_i \right)^T \left(\sum_{j=1}^n \alpha_j x_j y_j \right) = \sum_{i=1}^n \sum_{j=1}^n \alpha_i \alpha_j x_i x_j y_i y_j$$

Which **only depends on dot product** (between pair of samples): $x_i^T x_j$

...which is a scalar and can be replaced with a **kernel**: $K(x_i, x_j)$

Support Vector Classifier (SVC): The kernel trick

The **kernel** $K(x_i, x_j)$ corresponds to a similarity function and delivers a scalar!

Instead of mapping $x \rightarrow \phi(x)$ and computing explicitly all $f(x) = \text{sgn}(w^T \phi(x))$

In the dual form, we compute:

$$K(x_i, x_j) = \phi(x_i)^T \phi(x_j)$$

...dot product = scalar

...no need to calculate $\phi(x)$!

...very efficient in huge feature spaces!

...hence, once trained, the classifier becomes, e.g.: $f(x) = \text{sgn}\left(\sum_{i=1}^n \alpha_i y_i K(x_i, x_j) - b\right)$

The kernel trick

1) Consider the following minimal dataset:

$$x_1 = -2, y_1 = -1 \text{ (class A)}$$

$$x_2 = -1, y_2 = -1 \text{ (class B)}$$

$$x_3 = 1, y_3 = 1 \text{ (class B)}$$

$$x_4 = 2, y_4 = 1 \text{ (class A)}$$

- Plot the data and see if this can be separated in 1D
 - Map data using $\phi(x) = (x, x^2)$
 - Plot transformed data
 - What kernel corresponds to this mapping (assume two points x and z)?
- 2) Imagine the following kernel $K(x, z) = (1 + x^2 z^2)$. What would be the corresponding feature map (how many dimensions)?

Common Kernels

- Linear kernel (special case): $K(x_i, x_j) = x_i^T x_j$
- Polynomial kernel): $K(x_i, x_j) = (x_i^T x_j + c)^d$
- Radial Basis Function (RBF, Gaussian) kernel:
 - $K(x_i, x_j) = \exp(-\gamma \cdot \|x_i - x_j\|^2)$
 - Most popular for non-linear data
 - Draws smooth, flexible decision boundaries
- Sigmoid kernel: $K(x_i, x_j) = \tanh(\beta x_i^T x_j + \theta)$

Try: <https://greitemann.dev/svm-demo>

SVC for Breast Cancer set

- 1) Run a Classification using a SVC with linear kernel on the Wisconsin Breast Cancer dataset (*SVC_breastcancer.ipynb*)
- 2) Compare different kernels (linear, RBF) and other models on a reduced dataset (more difficult scenario)

In both tasks,

- Use Pipelines
- Perform a GridSearchCV
- Calculate a confusion matrix

Outlook

Regression models based on the kernel trick:

- Support Vector Regressor (SVR)
- Kernel Ridge regression (KRR)

Both use the same kernel trick as SVC, but

- Minimise squared loss (KRR) or ϵ -insensitive loss (SVR)
- encourage smooth, nonlinear function fitting for continuous data
- no margin concept (except in SVR where a “tube” is used)

Will return to them when we talk about linear regression.