Text as Data

Meeting 5: Supervised Machine Learning

Petro Tolochko

Machine Learning

Machine Learning

- Supervised
  - An outcome variable is defined
  - Focus on prediction

Machine Learning

- Supervised
  - An outcome variable is defined
  - Focus on prediction
- Usupervised
  - No outcome variable defined
  - Focus on pattern recognition

Machine Learning

- **Supervised**
  - An outcome variable is defined
  - Focus on prediction
- Unsupervised
  - No outcome variable defined
  - Focus on pattern recognition

Supervised

- Objective:

Supervised

- Objective:
  - Classification of documents into preexisting categories

Supervised

- Objective:
  - Classification of documents into preexisting categories
  - Text regression (scaling)

Dictionary vs. Supervised Machine Learning

Dictionary-based Methods

Supervised Machine Learning

Dictionary vs. Supervised Machine Learning

Dictionary-based Methods $\Leftrightarrow$ **Supervised Machine Learning**

Dictionary-based Methods

- Use predefined word lists (lexicons)

Supervised Machine Learning

- Learns patterns from labeled examples

Dictionary vs. Supervised Machine Learning

Dictionary-based Methods $\Leftrightarrow$ **Supervised Machine Learning**

Dictionary-based Methods

- Use predefined word lists (lexicons)
- Meaning inferred from word presence

Supervised Machine Learning

- Learns patterns from labeled examples
- Uses features (e.g., words, embeddings)

Dictionary vs. Supervised Machine Learning

Dictionary-based Methods $\Leftrightarrow$ **Supervised Machine Learning**

Dictionary-based Methods

- Use predefined word lists (lexicons)
- Meaning inferred from word presence
- Transparent and interpretable

Supervised Machine Learning

- Learns patterns from labeled examples
- Uses features (e.g., words, embeddings)
- Often less interpretable (black box)

Dictionary vs. Supervised Machine Learning

Dictionary-based Methods $\Leftrightarrow$ **Supervised Machine Learning**

Dictionary-based Methods

- Use predefined word lists (lexicons)
- Meaning inferred from word presence
- Transparent and interpretable
- No training data required

Supervised Machine Learning

- Learns patterns from labeled examples
- Uses features (e.g., words, embeddings)
- Often less interpretable (black box)
- Requires annotated training data

Dictionary vs. Supervised Machine Learning

Dictionary-based Methods $\Leftrightarrow$ **Supervised Machine Learning**

Dictionary-based Methods

- Use predefined word lists (lexicons)
- Meaning inferred from word presence
- Transparent and interpretable
- No training data required
- Limited adaptability (depends on dictionary)

Supervised Machine Learning

- Learns patterns from labeled examples
- Uses features (e.g., words, embeddings)
- Often less interpretable (black box)
- Requires annotated training data
- Flexible and domain-adaptive

Dictionary vs. Supervised Machine Learning

Dictionary-based Methods $\Leftrightarrow$ **Supervised Machine Learning**

Dictionary-based Methods

- Use predefined word lists (lexicons)
- Meaning inferred from word presence
- Transparent and interpretable
- No training data required
- Limited adaptability (depends on dictionary)

Supervised Machine Learning

- Learns patterns from labeled examples
- Uses features (e.g., words, embeddings)
- Often less interpretable (black box)
- Requires annotated training data
- Flexible and domain-adaptive

$\rightarrow$ *Dictionary methods encode human theory; supervised ML discovers patterns from data.*

Supervised ML Pipeline

- Create a labeled dataset
- Apply a function that maps features $\rightarrow$ outcome (ML step)
- Assess performance

Classifying Documents with Supervised Learning

- **Key considerations:**

Classifying Documents with Supervised Learning

- **Key considerations:**
  - **Feature representation:** how to turn text into numbers (e.g., Bag-of-Words, TF–IDF, or embeddings)

Classifying Documents with Supervised Learning

- **Key considerations:**
  - **Feature representation:** how to turn text into numbers (e.g., Bag-of-Words, TF–IDF, or embeddings)
  - **Feature selection:** remove noisy or irrelevant features (e.g., stop words, rare terms)

Classifying Documents with Supervised Learning

- **Key considerations:**
  - **Feature representation:** how to turn text into numbers (e.g., Bag-of-Words, TF–IDF, or embeddings)
  - **Feature selection:** remove noisy or irrelevant features (e.g., stop words, rare terms)
  - **Classifier selection:** choose the learning algorithm (e.g., Naive Bayes, SVM, k-NN, Logistic Regression, or ensembles)

Classifying Documents with Supervised Learning

- **Key considerations:**
  - **Feature representation:** how to turn text into numbers (e.g., Bag-of-Words, TF–IDF, or embeddings)
  - **Feature selection:** remove noisy or irrelevant features (e.g., stop words, rare terms)
  - **Classifier selection:** choose the learning algorithm (e.g., Naive Bayes, SVM, k-NN, Logistic Regression, or ensembles)
- **Goal:** learn a function $f(X) \to Y$ that generalizes to unseen documents.

Labeled Dataset

- How:
  - Human coders annotate parts of the corpus
  - Found data (e.g., self-reported profession in users' profile)
- Considerations:
  - Sampling should be representative for the corpus (e.g., Random, Stratified sample e.g., across time and source)
  - Quality of human coding matters (Assess the intercoder reliability)
  - Number of documents

Labeled Dataset

- Number of documents
  - the higher the number of categories and the lower the reliability of the coders, the higher the number of documents (Barberá et al., 2021)
- increase the sizes of manually coded validation dataset as large as possible (e.g., more than 1% of all data to be examined), assuming acceptable reliability (equal to or higher than .7) (Song et al., 2021)

Splitting the Data

- Split labeled data in training data and test data (validation data)

Splitting the Data

- Split labeled data in training data and test data (validation data)
- Training Data
  - The subset that is used to learn the model parameters

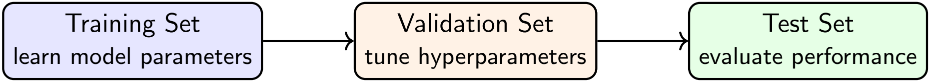
Splitting the Data

- Split labeled data in training data and test data (validation data)
- Training Data
  - The subset that is used to learn the model parameters
- Validation Data
  - The subset used to tune validate the parameters

Splitting the Data

- Split labeled data in training data and test data (validation data)
- Training Data
  - The subset that is used to learn the model parameters
- Validation Data
  - The subset used to tune validate the parameters
- Test Data
  - The subset that is used to evaluate the final performance
  - Not used for learning

Train–Validation–Test Split



Used during model development → Final evaluation only

The validation set guides learning; the test set measures generalization.

Cross-Validation

- Used to estimate model performance when data is limited.

Cross-Validation

- Used to estimate model performance when data is limited.
- Split data into k folds (subsets).

Cross-Validation

- Used to estimate model performance when data is limited.
- Split data into k folds (subsets).
- Train the model on $k-1$ folds and validate on the remaining one.

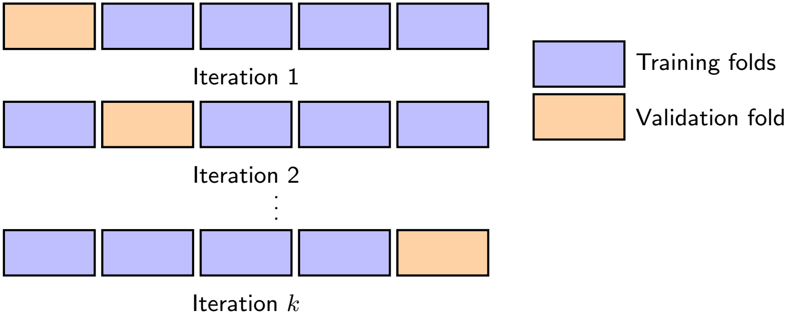
Cross-Validation

- Used to estimate model performance when data is limited.
- Split data into k folds (subsets).
- Train the model on $k - 1$ folds and validate on the remaining one.
- Repeat k times, each time with a different validation fold.

Cross-Validation

- Used to estimate model performance when data is limited.
- Split data into k folds (subsets).
- Train the model on $k-1$ folds and validate on the remaining one.
- Repeat k times, each time with a different validation fold.
- Average performance across folds $\Rightarrow$ robust estimate of generalization.

Cross-Validation



Each fold is used once for validation and $k-1$ times for training.

Document Classification

- Classifier *learns* the mapping between the features and the labels in the training set

Document Classification

- Classifier *learns* the mapping between the features and the labels in the training set
- Define a model $Y = f(X)$

Document Classification

- Classifier *learns* the mapping between the features and the labels in the training set
- Define a model $Y = f(X)$
- Apply the model to *learn* which features in X (extracted from raw text) matter to recover Y (i.e., labels from the training data)

Machine Learning 101

Machine Learning 101

- **Model:**

Machine Learning 101

- **Model:**
  - $Y = f(X; \theta)$

Machine Learning 101

- **Model:**
  - $Y = f(X;\theta)$
- **Objective function:**

Machine Learning 101

- **Model:**
  - $Y = f(X; \theta)$
- **Objective function:**
  - $\mathcal{L}(\theta; \mathcal{D})$
    - $\mathcal{L}$ – generic loss function
    - $\mathcal{D}$ – data
    - θ – parameter vector

Machine Learning 101

- **Model:**
  - $Y = f(X; \theta)$
- **Objective function:**
  - $\mathcal{L}(\theta; \mathcal{D})$
    - $\mathcal{L}$ – generic loss function
    - $\mathcal{D}$ – data
    - θ – parameter vector
    - Example: $MSE = \frac{1}{N} \sum_{i=1}^{N} (y_i - \hat{y}_i)^2$

Machine Learning 101

- **Model:**
  - $Y = f(X; \theta)$
- **Objective function:**
  - $\mathcal{L}(\theta; \mathcal{D})$
    - $\mathcal{L}$ – generic loss function
    - $\mathcal{D}$ – data
    - θ – parameter vector
    - Example: $MSE = \frac{1}{N} \sum_{i=1}^{N} (y_i - \hat{y}_i)^2$
- **Optimization step:**
  - $\hat{\theta} = \arg\min_{\theta} \mathcal{L}(\theta; \mathcal{D})$

Machine Learning 101

- **Model:**
  - $Y = f(X; \theta)$
- **Objective function:**
  - $\mathcal{L}(\theta; \mathcal{D})$
    - $\mathcal{L}$ – generic loss function
    - $\mathcal{D}$ – data
    - θ – parameter vector
    - Example: $MSE = \frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2$
- **Optimization step:**
  - $\hat{\theta} = \arg\min_{\theta} \mathcal{L}(\theta; \mathcal{D})$
  - For example: $\hat{\theta} = \arg\min_{\theta} \frac{1}{N}\sum_{i=1}^{N}(y_i - f_{\theta}(x_i))^2$

Machine Learning 101

- **Model (Machine):**
  - $Y = f(X; \theta)$
- **Objective function (Learning):**
  - $\mathcal{L}(\theta; \mathcal{D})$
    - $\mathcal{L}$ – generic loss function
    - $\mathcal{D}$ – data
    - θ – parameter vector
    - Example: $MSE = \frac{1}{N} \sum_{i=1}^{N} (y_i - \hat{y}_i)^2$
- **Optimization step (Learning):**
  - $\hat{\theta} = \arg\min_{\theta} \mathcal{L}(\theta; \mathcal{D})$
  - For example: $\hat{\theta} = \arg\min_{\theta} \frac{1}{N} \sum_{i=1}^{N} (y_i - f_{\theta}(x_i))^2$

Questions?

Naive Bayes

Bayes' Rule

$$P(A \mid B) = \frac{P(B \mid A)\, P(A)}{P(B)}$$

Bayes' Rule

$$P(A \mid B) = \frac{P(B \mid A)\, P(A)}{P(B)}$$

- $P(A)$ — prior probability of hypothesis A

Bayes' Rule

$$P(A \mid B) = \frac{P(B \mid A)\, P(A)}{P(B)}$$

- $P(A)$ — prior probability of hypothesis A
- $P(B \mid A)$ — likelihood of observing B if A is true

Bayes' Rule

$$P(A \mid B) = \frac{P(B \mid A)\, P(A)}{P(B)}$$

- $P(A)$ — prior probability of hypothesis A
- $P(B \mid A)$ — likelihood of observing B if A is true
- $P(B)$ — marginal probability of observing B

Bayes' Rule

$$P(A \mid B) = \frac{P(B \mid A)\, P(A)}{P(B)}$$

- $P(A)$ — prior probability of hypothesis A
- $P(B \mid A)$ — likelihood of observing B if A is true
- $P(B)$ — marginal probability of observing B
- $P(A \mid B)$ — posterior probability after seeing B

Bayes' Rule

$$P(A \mid B) = \frac{P(B \mid A)\, P(A)}{P(B)}$$

- $P(A)$ — prior probability of hypothesis A
- $P(B \mid A)$ — likelihood of observing B if A is true
- $P(B)$ — marginal probability of observing B
- $P(A \mid B)$ — posterior probability after seeing B

Bayesian idea: update prior beliefs with new evidence.

From Bayes' Rule to the Naive Bayes Classifier

$$P(C \mid X) = \frac{P(X \mid C)\,P(C)}{P(X)}$$

$$\Rightarrow \quad \hat{C} = \arg\max_{C} P(X \mid C)\,P(C)$$

From Bayes' Rule to the Naive Bayes Classifier

$$P(C \mid X) = \frac{P(X \mid C)\, P(C)}{P(X)}$$

$$\Rightarrow \quad \hat{C} = \arg\max_{C} P(X \mid C)\, P(C)$$

- C: class label (e.g., positive/negative sentiment)

From Bayes' Rule to the Naive Bayes Classifier

$$P(C \mid X) = \frac{P(X \mid C)\, P(C)}{P(X)}$$

$$\Rightarrow \quad \hat{C} = \arg\max_{C} P(X \mid C)\, P(C)$$

- C: class label (e.g., positive/negative sentiment)
- X: document represented as words or features

From Bayes' Rule to the Naive Bayes Classifier

$$P(C \mid X) = \frac{P(X \mid C)\, P(C)}{P(X)}$$

$$\Rightarrow \quad \hat{C} = \arg\max_{C} P(X \mid C)\, P(C)$$

- C: class label (e.g., positive/negative sentiment)
- X: document represented as words or features
- Naive assumption: features are conditionally independent given the class

$$P(X \mid C) = \prod_{i=1}^{n} P(x_i \mid C)$$

From Bayes' Rule to the Naive Bayes Classifier

$$P(C \mid X) = \frac{P(X \mid C)\, P(C)}{P(X)}$$

$$\Rightarrow \quad \hat{C} = \arg\max_{C} P(X \mid C)\, P(C)$$

- C: class label (e.g., positive/negative sentiment)
- X: document represented as words or features
- Naive assumption: features are conditionally independent given the class

$$P(X \mid C) = \prod_{i=1}^{n} P(x_i \mid C)$$

- In text classification: estimate $P(x_i \mid C)$ from word frequencies.

From Bayes' Rule to the Naive Bayes Classifier

$$P(C \mid X) = \frac{P(X \mid C)\, P(C)}{P(X)}$$

$$\Rightarrow \quad \hat{C} = \arg\max_{C} P(X \mid C)\, P(C)$$

- C: class label (e.g., positive/negative sentiment)
- X: document represented as words or features
- Naive assumption: features are conditionally independent given the class

$$P(X \mid C) = \prod_{i=1}^{n} P(x_i \mid C)$$

- In text classification: estimate $P(x_i \mid C)$ from word frequencies.

$\rightarrow$ *The classifier picks the class with the highest posterior probability.*

How Priors and Likelihoods Combine

$$\underset{\text{(Prior)}}{P(C)} \longrightarrow \underset{\text{(Likelihood)}}{P(X \mid C)} \longrightarrow \underset{\text{(Posterior)}}{P(C \mid X)}$$

Posterior = Prior × Likelihood (then normalized).

Naive Bayes picks the class with the highest posterior.

Naive Bayes and the Multinomial Model of Language

- The **Naive Bayes classifier** assumes conditional independence of features:

$$P(X \mid C) = \prod_{i=1}^{V} P(x_i \mid C)$$

Naive Bayes and the Multinomial Model of Language

- The **Naive Bayes classifier** assumes conditional independence of features:

$$P(X \mid C) = \prod_{i=1}^{V} P(x_i \mid C)$$

- For text, we use a **multinomial model**:

$$P(X \mid C) = \frac{N!}{x_1! \ldots x_V!} \prod_{i=1}^{V} P(w_i \mid C)^{x_i}$$

Naive Bayes and the Multinomial Model of Language

- The **Naive Bayes classifier** assumes conditional independence of features:

$$P(X \mid C) = \prod_{i=1}^{V} P(x_i \mid C)$$

- For text, we use a **multinomial model**:

$$P(X \mid C) = \frac{N!}{x_1! \dots x_V!} \prod_{i=1}^{V} P(w_i \mid C)^{x_i}$$

- Each class defines its own word distribution $P(w_i \mid C)$.

Naive Bayes and the Multinomial Model of Language

- The **Naive Bayes classifier** assumes conditional independence of features:

$$P(X \mid C) = \prod_{i=1}^{V} P(x_i \mid C)$$

- For text, we use a **multinomial model**:

$$P(X \mid C) = \frac{N!}{x_1! \dots x_V!} \prod_{i=1}^{V} P(w_i \mid C)^{x_i}$$

- Each class defines its own word distribution $P(w_i \mid C)$.
- Classification rule:

$$\hat{C} = \arg\max_{C} P(C) \prod_{i=1}^{V} P(w_i \mid C)^{x_i}$$

Naive Bayes and the Multinomial Model of Language

- The **Naive Bayes classifier** assumes conditional independence of features:

$$P(X \mid C) = \prod_{i=1}^{V} P(x_i \mid C)$$

- For text, we use a **multinomial model**:

$$P(X \mid C) = \frac{N!}{x_1! \ldots x_V!} \prod_{i=1}^{V} P(w_i \mid C)^{x_i}$$

- Each class defines its own word distribution $P(w_i \mid C)$.
- Classification rule:

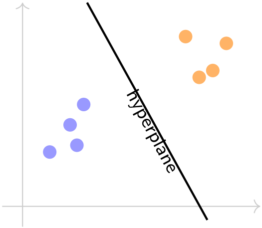
$$\hat{C} = \arg\max_{C} P(C) \prod_{i=1}^{V} P(w_i \mid C)^{x_i}$$

- Interpretation: each class is a simple *language model* — choose the one most likely to have generated the document.

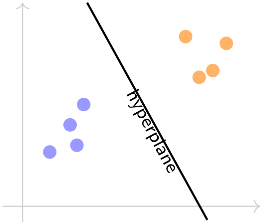
Questions?

Support Vector Machines

What is a Hyperplane?

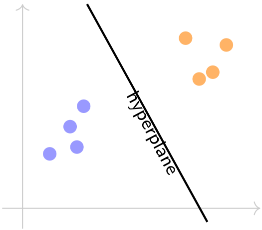


What is a Hyperplane?



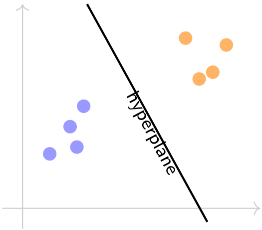
- A **hyperplane** is a flat surface that divides the feature space into two halves.

What is a Hyperplane?



- A **hyperplane** is a flat surface that divides the feature space into two halves.
- In 2D $\rightarrow$ it's a line. In 3D $\rightarrow$ a plane. In text (many features) $\rightarrow$ a high-dimensional surface.

What is a Hyperplane?

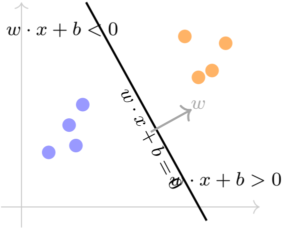


- A **hyperplane** is a flat surface that divides the feature space into two halves.
- In 2D $\rightarrow$ it's a line. In 3D $\rightarrow$ a plane. In text (many features) $\rightarrow$ a high-dimensional surface.
- Used to separate points belonging to different classes.

The Hyperplane Equation

Hyperplane: $w \cdot x + b = 0$

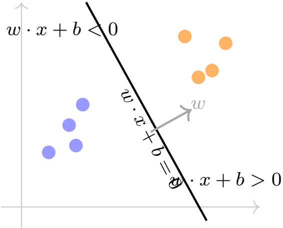
Decision rule: $\hat{y} = \text{sign}(w \cdot x + b)$



The Hyperplane Equation

Hyperplane: $w \cdot x + b = 0$

Decision rule: $\hat{y} = \text{sign}(w \cdot x + b)$

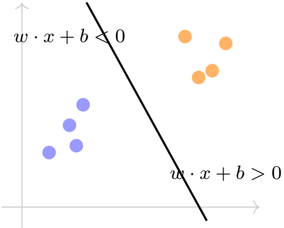


The hyperplane is the boundary where $w \cdot x + b = 0$.

Classification Rule

$$\text{Hyperplane: } w \cdot x + b = 0$$

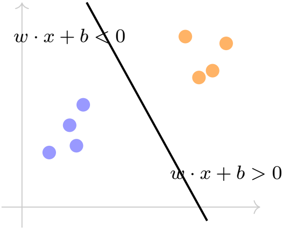
$$\text{Prediction: } \hat{y} = \begin{cases} +1, & \text{if } w \cdot x + b > 0 \\ -1, & \text{if } w \cdot x + b < 0 \end{cases}$$



Classification Rule

Hyperplane: $w \cdot x + b = 0$

$$\text{Prediction: } \hat{y} = \begin{cases} +1, & \text{if } w \cdot x + b > 0 \\ -1, & \text{if } w \cdot x + b < 0 \end{cases}$$

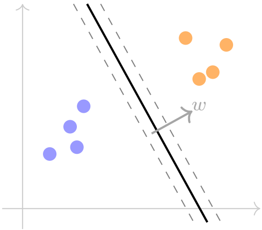


The sign of $(w \cdot x + b)$ *determines on which side of the hyperplane a point lies.*

Distance to the Hyperplane

Signed distance: $d_i = \dfrac{w \cdot x_i + b}{\|w\|}$

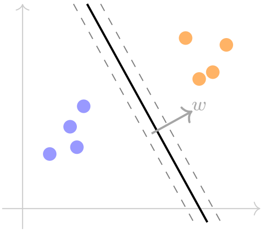
Margin: distance between supporting hyperplanes $= \dfrac{2}{\|w\|}$



Distance to the Hyperplane

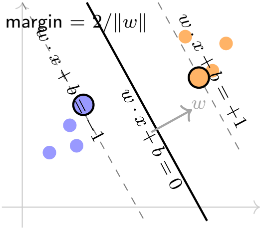
$$\text{Signed distance:} \quad d_i = \frac{w \cdot x_i + b}{\|w\|}$$

$$\text{Margin:} \quad \text{distance between supporting hyperplanes} = \frac{2}{\|w\|}$$

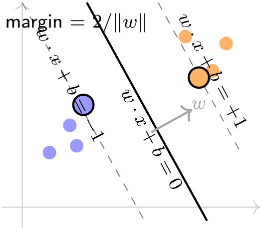


SVM chooses the hyperplane that maximizes this margin.

Support Vectors and the Margin



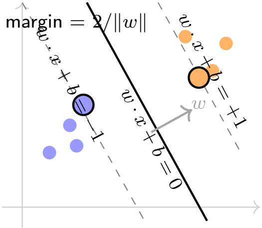
Support Vectors and the Margin



- **Support vectors** lie exactly on the margin hyperplanes:

$$w \cdot x + b = \pm 1$$

Support Vectors and the Margin

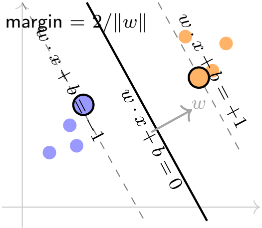


- **Support vectors** lie exactly on the margin hyperplanes:

$$w \cdot x + b = \pm 1$$

- They define the optimal separating boundary.

Support Vectors and the Margin



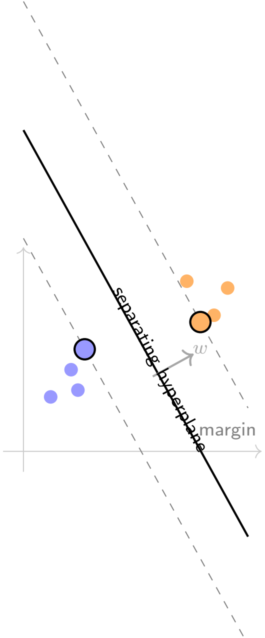
- **Support vectors** lie exactly on the margin hyperplanes:

$$w \cdot x + b = \pm 1$$

- They define the optimal separating boundary.
- SVM maximizes the distance between these two margin planes.

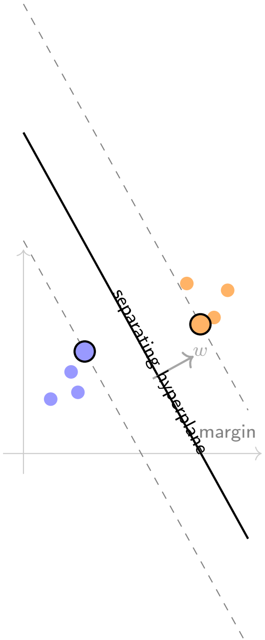
Support Vector Machines (SVMs)

- Represent each document as a feature vector (e.g., TF–IDF)



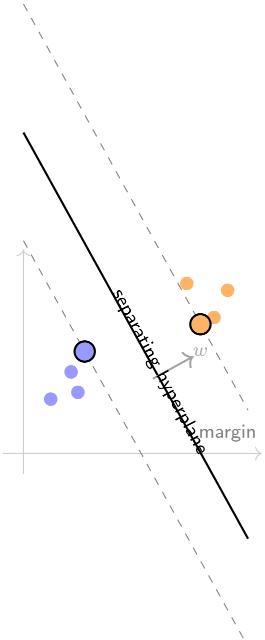
Support Vector Machines (SVMs)

- Represent each document as a feature vector (e.g., TF–IDF)
- SVM finds the hyperplane that best separates the classes



Support Vector Machines (SVMs)

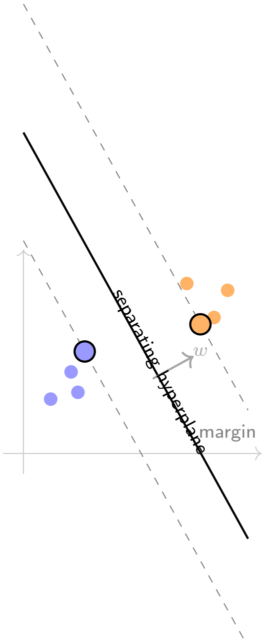
- Represent each document as a feature vector (e.g., TF–IDF)
- SVM finds the hyperplane that best separates the classes
- **Goal:** maximize the margin between the closest points of different classes



Support Vector Machines (SVMs)

- Represent each document as a feature vector (e.g., TF–IDF)
- SVM finds the hyperplane that best separates the classes
- **Goal:** maximize the margin between the closest points of different classes
- Decision rule:

$$\hat{y} = \mathsf{sign}(w \cdot x + b)$$

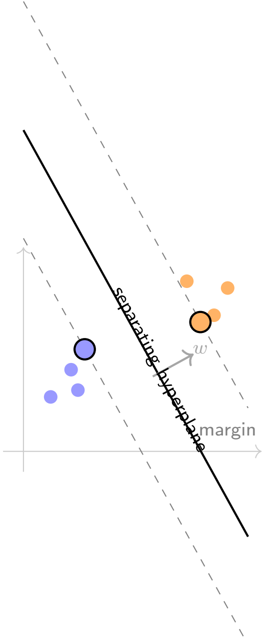


Support Vector Machines (SVMs)

- Represent each document as a feature vector (e.g., TF–IDF)
- SVM finds the hyperplane that best separates the classes
- **Goal:** maximize the margin between the closest points of different classes
- Decision rule:

$$\hat{y} = \text{sign}(w \cdot x + b)$$

- Works well for high-dimensional sparse data like text

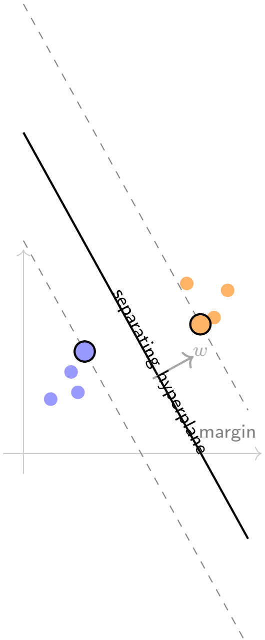


Support Vector Machines (SVMs)

- Represent each document as a feature vector (e.g., TF–IDF)
- SVM finds the hyperplane that best separates the classes
- **Goal:** maximize the margin between the closest points of different classes
- Decision rule:

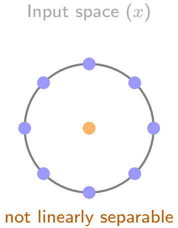
$$\hat{y} = \text{sign}(w \cdot x + b)$$

- Works well for high-dimensional sparse data like text



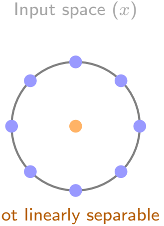
When Linear Boundaries Fail

- Linear classifiers (like SVMs) can't separate all datasets.



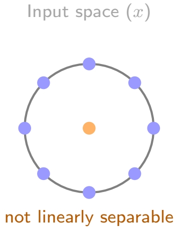
When Linear Boundaries Fail

- Linear classifiers (like SVMs) can't separate all datasets.
- Example: circular data: no straight line can split inner and outer classes.



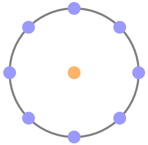
When Linear Boundaries Fail

- Linear classifiers (like SVMs) can't separate all datasets.
- Example: circular data: no straight line can split inner and outer classes.
- Need to think beyond linear boundaries.



When Linear Boundaries Fail

- Linear classifiers (like SVMs) can't separate all datasets.
- Example: circular data: no straight line can split inner and outer classes.
- Need to think beyond linear boundaries.



No linear hyperplane in 2D can separate these points.

The Kernel Trick

- Instead of forcing a linear boundary in the original space, we can **transform** the data into a higher-dimensional space.

The Kernel Trick

- Instead of forcing a linear boundary in the original space, we can **transform** the data into a higher-dimensional space.
- In that new space, the data may become **linearly separable**.

The Kernel Trick

- Instead of forcing a linear boundary in the original space, we can **transform** the data into a higher-dimensional space.
- In that new space, the data may become **linearly separable**.
- Formally, a mapping:

$$\phi : \mathbb{R}^n \to \mathbb{R}^m, \quad m > n$$

The Kernel Trick

- Instead of forcing a linear boundary in the original space, we can **transform** the data into a higher-dimensional space.
- In that new space, the data may become **linearly separable**.
- Formally, a mapping:

$$\phi : \mathbb{R}^n \to \mathbb{R}^m, \quad m > n$$

- Rather than computing $\phi(x)$ directly, SVMs use **kernel functions** to compute inner products:

$$K(x_i, x_j) = \langle \phi(x_i), \phi(x_j) \rangle$$

The Kernel Trick

- Instead of forcing a linear boundary in the original space, we can **transform** the data into a higher-dimensional space.
- In that new space, the data may become **linearly separable**.
- Formally, a mapping:

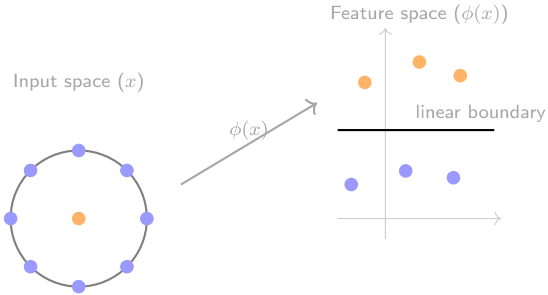
$$\phi : \mathbb{R}^n \to \mathbb{R}^m, \quad m > n$$

- Rather than computing $\phi(x)$ directly, SVMs use **kernel functions** to compute inner products:

$$K(x_i, x_j) = \langle \phi(x_i), \phi(x_j) \rangle$$

- This is called the **kernel trick**, we get all the benefits of high-dimensional geometry without ever leaving the original feature space.

The Kernel Trick

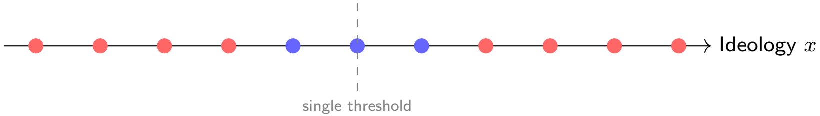


Common Kernel Functions

Kernel	Formula	Intuition
Linear	$K(x_i, x_j) = x_i \cdot x_j$	Measures direct similarity (no transformation).
Polynomial	$K(x_i, x_j) = (x_i \cdot x_j + c)^d$	Captures feature interactions (e.g., word pairs).
RBF / Gaussian	$K(x_i, x_j) = e^{-\gamma \|x_i - x_j\|^2}$	Distance-based similarity; very flexible and smooth.
Sigmoid	$K(x_i, x_j) = \tanh(\alpha\, x_i \cdot x_j + c)$	Behaves like a neural network activation.

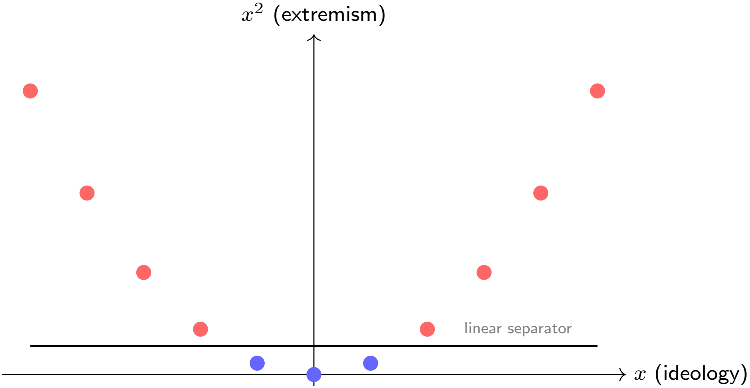
Kernels measure similarity in an implicit feature space —
they let SVMs handle nonlinear patterns without explicit transformation.

Ideology in 1D: Not Linearly Separable



Extremes (red) lie on both sides of the center — one cut on a line can't separate them from moderates (blue).

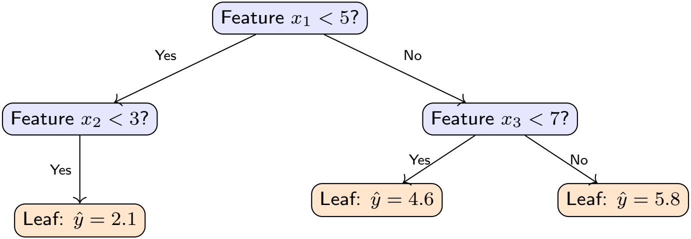
Nonlinear Map $\phi(x) = (x,\, x^2)$: Now Linearly Separable



After $\phi(x)$, a horizontal linear boundary separates extremes (red) from moderates (blue).

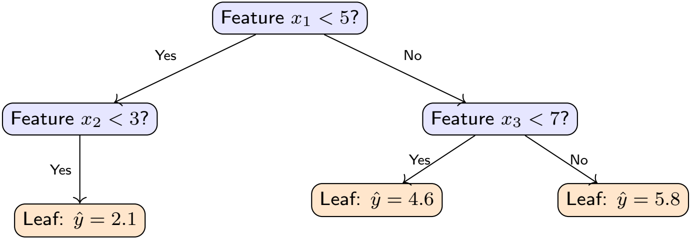
Decision Trees

- Trees split the feature space into regions where target values are similar.



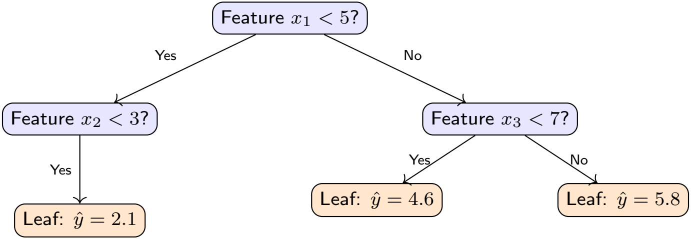
Decision Trees

- Trees split the feature space into regions where target values are similar.
- Each internal node splits data by a feature threshold (e.g., $x_j < t$).



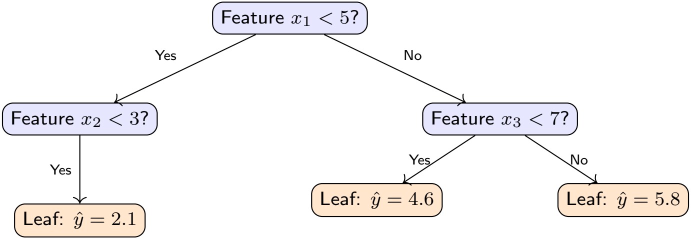
Decision Trees

- Trees split the feature space into regions where target values are similar.
- Each internal node splits data by a feature threshold (e.g., $x_j < t$).
- Each leaf predicts an outcome (class label or average value).



Decision Trees

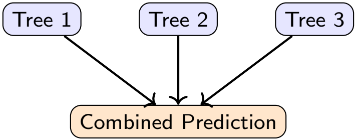
- Trees split the feature space into regions where target values are similar.
- Each internal node splits data by a feature threshold (e.g., $x_j < t$).
- Each leaf predicts an outcome (class label or average value).
- The goal is to minimize a loss function over all splits (e.g., MSE or Gini impurity).



Questions?

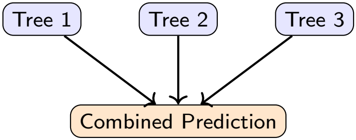
From Trees to Ensembles

- A single tree is a weak learner — high variance and easy to overfit.



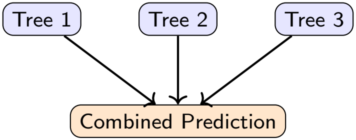
From Trees to Ensembles

- A single tree is a weak learner — high variance and easy to overfit.
- **Bagging:** average many independent trees (e.g., Random Forests).



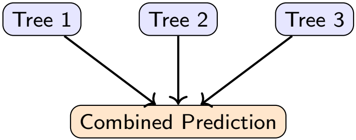
From Trees to Ensembles

- A single tree is a weak learner — high variance and easy to overfit.
- **Bagging:** average many independent trees (e.g., Random Forests).
- **Boosting:** build trees sequentially; each focuses on correcting errors of the previous ones.



From Trees to Ensembles

- A single tree is a weak learner — high variance and easy to overfit.
- **Bagging:** average many independent trees (e.g., Random Forests).
- **Boosting:** build trees sequentially; each focuses on correcting errors of the previous ones.
- Ensemble prediction combines all trees' outputs for a stronger model.



Gradient Boosting: Learning from Residuals

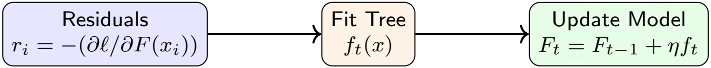
- We want to minimize a loss function $\mathcal{L}(y, F(x))$.

Residuals $r_i = -(\partial\ell/\partial F(x_i))$ $\longrightarrow$ Fit Tree $f_t(x)$ $\longrightarrow$ Update Model $F_t = F_{t-1} + \eta f_t$

Each new tree points in the direction that most reduces the loss — gradient descent in function space.

Gradient Boosting: Learning from Residuals

- We want to minimize a loss function $\mathcal{L}(y, F(x))$.
- Each new tree predicts the negative gradient (residuals) of the loss.

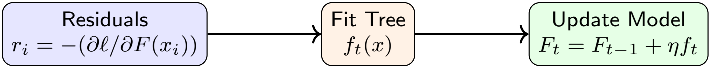


Each new tree points in the direction that most reduces the loss — gradient descent in function space.

Gradient Boosting: Learning from Residuals

- We want to minimize a loss function $\mathcal{L}(y, F(x))$.
- Each new tree predicts the negative gradient (residuals) of the loss.
- Trees are added sequentially:

$$F_t(x) = F_{t-1}(x) + \eta f_t(x)$$



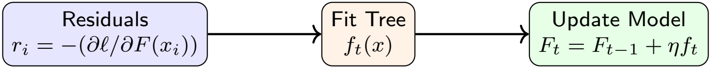
Each new tree points in the direction that most reduces the loss — gradient descent in function space.

Gradient Boosting: Learning from Residuals

- We want to minimize a loss function $\mathcal{L}(y, F(x))$.
- Each new tree predicts the negative gradient (residuals) of the loss.
- Trees are added sequentially:

$$F_t(x) = F_{t-1}(x) + \eta f_t(x)$$

- The learning rate η controls how much each tree contributes.



Each new tree points in the direction that most reduces the loss — gradient descent in function space.

Gradient Boosting: Learning from Residuals

- We want to minimize a loss function $\mathcal{L}(y, F(x))$.
- Each new tree predicts the negative gradient (residuals) of the loss.
- Trees are added sequentially:

$$F_t(x) = F_{t-1}(x) + \eta f_t(x)$$

- The learning rate η controls how much each tree contributes.
- After T iterations, the final model is the sum of all trees:

$$F_T(x) = \sum_{t=1}^{T} \eta f_t(x)$$

Residuals $r_i = -(\partial \ell / \partial F(x_i))$ → Fit Tree $f_t(x)$ → Update Model $F_t = F_{t-1} + \eta f_t$

Each new tree points in the direction that most reduces the loss — gradient descent in function space.