Part III

Classification Basics
Problem
A simple example

A dataset with two classes
A simple example

**data points:** Iris flowers

**attributes:** physical properties, length of the petal and length of the sepal in cm

**class:** species, *versicolor* vs. *virginica*
A simple example

versicolor

petal

sepal

virginica

sepal length

petal length

5.0 5.5 6.0 6.5 7.0 7.5 8.0

3.0 3.5 4.0 4.5 5.0 5.5 6.0
A simple example

Class information, i.e. species, is absent for some points. Can we use the available information to predict it?
Different methods

Look at the most similar data points → $k$ nearest neighbors ($k$-NN)

majority class among $k$ nearest neighbors
Different methods

Apply a sequence of tests on attributes’ values
→ classification tree
Different methods

Look at class probabilities conditioned on attributes’ values
→ Naive bayes

\[
P(c \mid sl, sp) \propto P(c) \cdot P(sl \mid c) \cdot P(sp \mid c)
\]

\[
P(\bullet \mid sl, sp) > P(\circ \mid sl, sp)
\]

\[
P(\bullet \mid sl, sp) \leq P(\circ \mid sl, sp)
\]
Different methods

Look at the sign of a linear combination of the attributes → perceptron

\[ 0.671 \cdot \text{sl} - 1.365 \cdot \text{pl} + 2.39 < 0 \]
\[ 0.671 \cdot \text{sl} - 1.365 \cdot \text{pl} + 2.39 \geq 0 \]
Different methods

Look at the sign of a linear combination of the attributes → support vector machine (SVM)

\[ sl - 4 \cdot pl + 13.3 < 0 \]

\[ sl - 4 \cdot pl + 13.3 \geq 0 \]
Different methods ...

$k$-NN  decision tree  naive Bayes  perceptron  SVM
...but it is all about learning a decision boundary

- $k$-NN
- Decision tree
- Naive Bayes
- Perceptron
- SVM

---

JADe: Classification Basics
Some notations

The data set, denoted as $\mathcal{D}$, contains $n$ data points and $m$ attributes, i.e. it is a $n \times m$ matrix.

A data point is a $m$-dimensional vector $\mathbf{x} = \langle x_1, x_2, \ldots, x_m \rangle$

We denote $\mathbf{x}^{(j)}$ the $j^{th}$ data point of $\mathcal{D}$, i.e. the $j^{th}$ row.

Data points are sometimes called instances or examples.

Class labels are arranged into a $n$-dimensional vector $\mathbf{y} = \langle y_1, y_2, \ldots, y_n \rangle \in \mathcal{L}^n$, where $l = |\mathcal{L}|$ is the number of classes.

That is, $y_j$ is the class label associated with data point $\mathbf{x}^{(j)}$.

In binary classification, class labels take value $-1$ or $+1$ (sometimes $0$ or $1$ instead), i.e. $\mathcal{L} = \{-1, +1\}$ (respectively $\mathcal{L} = \{0, 1\}$) and the two classes might be referred to as negative and positive, respectively.
Methods
$k$ nearest neighbors

**Input:** data set $\mathcal{D}$, data point $x$

**Parameters:** distance function $d$, number of neighbors $k$

**No training**

**Prediction:** return majority class among $k$ points in $\mathcal{D}$ that minimize $d(x, x')$
\( \mathcal{K} \leftarrow \{ k \text{ points } x' \in D \text{ that minimize } d(x, x') \} \)

**return** majority class in \( \mathcal{K} \)

- \( k = 1 \)
- \( k = 3 \)
- \( k = 5 \)
- \( k = 9 \)
**$k$ nearest neighbors**

\[ \mathcal{K} \leftarrow \{k \text{ points } x' \in \mathcal{D} \text{ that minimize } d(x, x') \} \]

**return** majority class in $\mathcal{K}$

**Euclidean distance ($\ell_2$ norm)**

\[
d(x, x') = \sqrt{\sum_{i=1}^{m} (x_i - x'_i)^2}
\]

**Manhattan distance ($\ell_1$ norm)**

\[
d(x, x') = \sum_{i=1}^{m} |x_i - x'_i|
\]
**Decision tree**

**Input:** data set $\mathcal{D}$, data point $x$

**Parameters:** split evaluation measure, max depth $d$, min leaf size $l$

**Training:** construct tree $T$ by recursively finding tests that yield best splits in $\mathcal{D}$

**Prediction:** apply sequence of tests from $T$ to $x$ until reaching a leaf, return associated class
Decision tree: structure representing a succession of tests and possible classification or regression outcomes

**Leaf node** decision (class/value)

**Other node** test on an attribute’s value

\[
\begin{align*}
pl & \geq 4.85 \\
pl & \geq 5.15 \\
sl & \geq 4.95 \\
sl & \geq 6.60
\end{align*}
\]
Decision tree: Split evaluation measures

\[ X_i \geq v \]

- Yes
  - \( N_{\text{yes}} \) / \( N_{\text{yes}} \)
- No
  - \( N_{\text{no}} \) / \( N_{\text{no}} \)

\[ \text{yes} \]
\[ \text{no} \]

\[ \frac{N_{\text{yes}}}{N_{\text{yes}}} \]
\[ \frac{N_{\text{no}}}{N_{\text{no}}} \]
Decision tree: Split evaluation measures

\[ X_i \geq v \]

\[ N_{\text{yes}, \cdot} / N_{\text{yes}, \cdot} \]

\[ N_{\text{no}, \cdot} / N_{\text{no}, \cdot} \]

---

**Error rate**

\[
\sum_{b \in \{\text{yes}, \text{no}\}} \frac{1}{N} \left( N_b - \max_{c \in \{\cdot, \cdot\}} N_{b,c} \right)
\]

**Gini index**

\[
\sum_{b \in \{\text{yes}, \text{no}\}} \frac{N_b}{N} \left( 1 - \sum_{c \in \{\cdot, \cdot\}} \left( \frac{N_{b,c}}{N_b} \right)^2 \right)
\]

**Entropy**

\[
\sum_{b \in \{\text{yes}, \text{no}\}} \frac{N_b}{N} \sum_{c \in \{\cdot, \cdot\}} -\frac{N_{b,c}}{N_b} \log_2 \left( \frac{N_{b,c}}{N_b} \right)
\]
Decision tree: Split evaluation measures

\[
X_i \geq v \quad \rightarrow \quad \frac{N_{\text{yes},\cdot}}{N_{\text{yes},\cdot}}
\]

\[
\frac{N_{\text{no},\cdot}}{N_{\text{no},\cdot}}
\]

Information gain:

\[
\sum_{c \in \{\cdot, \cdot\}} - \frac{N_c}{N} \log_2 \left( \frac{N_c}{N} \right) - \sum_{b \in \{\text{yes, no}\}} \frac{N_b}{N} \sum_{c \in \{\cdot, \cdot\}} - \frac{N_{b,c}}{N_b} \log_2 \left( \frac{N_{b,c}}{N_b} \right)
\]
Decision tree: Split evaluation measures

\[ X_i \geq \nu \]

\[
\begin{array}{c|c|c}
\text{yes} & N_{\text{yes},\bullet} / N_{\text{yes},\circ} & \text{no} \\
\hline
\text{yes} & N_{\text{yes},\bullet} / N_{\text{yes},\circ} & \text{no} \\
\end{array}
\]

\[
\begin{array}{c}
\text{yes} \\
\hline
N_{\text{yes},\bullet} \\
\end{array}
\quad
\begin{array}{c}
\text{no} \\
\hline
N_{\text{no},\bullet} \\
\end{array}
\]

\[
\begin{array}{cccc}
10 & 0 & 9 & 1 \\
0 & 10 & 1 & 9 \\
7 & 3 & 5 & 5 \\
0 & 10 & 5 & 2 \\
7 & 2 & 3 & 8 \\
\end{array}
\]

\[
\begin{array}{c|c|c}
\text{ER} & 0.000 & 0.100 \\
G & 0.000 & 0.180 \\
E & 0.000 & 0.469 \\
IG & 1.000 & 0.531 \\
\end{array}
\quad
\begin{array}{c|c|c}
& N_{\text{yes},\bullet} / N_{\text{yes},\circ} & N_{\text{no},\bullet} / N_{\text{no},\circ} \\
\hline
\text{yes} & N_{\text{yes},\bullet} & N_{\text{no},\bullet} \\
\text{no} & N_{\text{no},\bullet} & N_{\text{no},\circ} \\
\end{array}
\]

\[
\begin{array}{c|c|c|c|c}
\text{ER} & 0.500 & 0.000 & 0.350 & 0.250 \\
G & 0.500 & 0.000 & 0.451 & 0.374 \\
E & 1.000 & 0.000 & 0.927 & 0.809 \\
IG & 0.000 & 1.000 & 0.073 & 0.191 \\
\end{array}
\]
Try splitting on different attributes and values
Decision tree: Training

Try splitting on different attributes and values

- sepal length
- petal length

### Graph:
- **sl ≥ 5.05**
- **G = 0.493**

41/39
3/1
Try splitting on different attributes and values

\[ G = 0.489 \]
Decision tree: Training

Try splitting on different attributes and values

![Decision tree diagram](image)

- sepal length
- petal length

\[ sl \geq 6.15 \]

- yes
- no

\[ G = 0.392 \]

\[ 15/32 \]

\[ 29/8 \]
Try splitting on different attributes and values

G = 0.493

pl $\geq$ 3.15

43/40

1/0
Decision tree: Training

Try splitting on different attributes and values

-G = 0.111

petal length ≥ 4.85
Try splitting on different attributes and values
Select best split, divide the data accordingly and recurse on the subsets.
Considering only the ‘no’ branch, try splitting on different attributes and values.
Considering only the ‘no’ branch, try splitting on different attributes and values

\[ \text{sepal length} \geq 5.05 \]

\[ \text{petal length} \geq 4.85 \]
Considering only the ‘no’ branch, try splitting on different attributes and values

$$G = 0.088$$
Considering only the ‘no’ branch, select best split...

\[ \text{pl} \geq 4.85 \]

\[ \text{sl} \geq 4.95 \]
Node is below the minimum size, add leaf with dominant class as decision
Decision tree: Training

Considering only the current branch, try splitting on different attributes and values.
No improving split can be found, add leaf with dominant class as decision.
Considering only the ‘yes’ branch, try splitting on different attributes and values.

\[ G = 0.128 \]
Considering only the ‘yes’ branch, try splitting on different attributes and values.
Considering only the ‘yes’ branch, try splitting on different attributes and values.
Considering only the ‘yes’ branch, select best split...

**Diagram:**
- Petal length (pl) ≥ 5.15: 0/28
- Petal length (pl) ≥ 4.85: 3/10
- Sepal length (sl) ≥ 4.95: -
Node is pure, add leaf with class as decision.

![Decision Tree Diagram]

- If petal length $\geq 5.15$
  - Yes: $0/28$
- If petal length $\geq 4.85$
  - No: $3/10$

- If sepal length $\geq 4.95$
  - Blue dot: $1/28$

---

**UEF://School of Computing**  **JADE: Classification Basics**  16/66
Considering only the current branch, try splitting on different attributes and values.
Considering only the current branch, try splitting on different attributes and values.

- **Sepal length**
  - $\geq 4.85$: Yes
  - $< 4.85$: No

- **Petal length**
  - $\geq 5.15$: Yes
  - $< 5.15$: No

- **Sepal length**
  - $\geq 6.60$: Yes
  - $< 6.60$: No

$G = 0.139$
Considering only the current branch, select best split...
Maximum depth has been reached, add leaves with dominant classes as decision.
Tree is fully grown...

- Sepal length vs. Petal length
- Decision tree with conditions:
  - Petal length $\geq 5.15$: yes
  - Petal length $\geq 4.85$: no
  - Sepal length $\geq 6.60$: yes
  - Sepal length $\geq 4.95$: no
Decision tree: Prediction

Given a point to classify: \((sl = 6.0, pl = 4.8)\)
Decision tree: Prediction

Apply test and follow branch according to the outcome

```
sepal length

petal length
```

![Graph showing a decision tree with nodes for sepal length and petal length, leading to classifications based on thresholds like \( pl \geq 4.85 \), \( sl \geq 6.60 \), and \( sl \geq 4.95 \).]
Decision tree: Prediction

Apply test and follow branch according to the outcome

![Decision Tree Diagram]

- **pl ≥ 5.15**
  - yes
  - **pl ≥ 4.85**
  - **sl ≥ 6.60**
    - no
  - **sl ≥ 4.95**
Apply test and follow branch according to the outcome
Decision tree: Prediction

Assign the class associated to the leaf

Assign the class associated to the leaf
Decision tree: Prediction

The sequences of tests corresponding to the branches of the tree define decision boundaries.
The sequences of tests corresponding to the branches of the tree define decision boundaries.
**Naive Bayes**

**Input:** data set $\mathcal{D}$ with attributes $X_1, \ldots, X_m$ and class labels $Y$, data point $x$

**Training:** estimate the class probabilities $P(Y)$ and conditional probabilities $P(X_i | Y)$ from $\mathcal{D}$

**Prediction:** compute conditional probabilities $P(Y | X_1, \ldots, X_m)$ according to Bayes’ rule, return class with highest probability
Naive Bayes: Bayes’ rule

\[ P(Y | X) = \frac{P(Y) \cdot P(X | Y)}{P(X)} \]

\[ P(Y = c | X_1 = a_1, \ldots, X_m = a_m) \]

\[ = \frac{P(Y = c) \cdot P(X_1 = a_1, \ldots, X_m = a_m | Y = c)}{P(X_1 = a_1, \ldots, X_m = a_m)} \]

\[ \propto P(Y = c) \cdot \prod_{i=1}^{i=m} P(X_i = a_i | Y = c) \]
Estimate $P(Y = c)$ and $P(X_i = a_i \mid Y = c)$ from the data, for the different classes $c$, attributes $X_i$ and values $a_i$. 
Naive Bayes: Training

\[ P(Y = c) \text{ count occurrences of each class in the data} \]

\[ P(Y = c) = \frac{\#(c)}{n} \]
Naive Bayes: Training

\[ P(Y = c) \text{ count occurrences of each class in the data} \]

\[ P(\bullet) = \frac{40}{84} = 0.476 \]

\[ P(\bullet) = \frac{44}{84} = 0.524 \]
Naive Bayes: Training

\[ P(X_i = a_i \mid Y = c) \] count occurrences of each values in the data, for each class and each attribute

\[
P(X_i = a_i \mid Y = c) = \frac{\#(a_i, Y = c)}{\#(c)}
\]
\[ P(X_i = a_i \mid Y = c) \] count occurrences of each values in the data, for each class and each attribute

\[
P(X_i = a_i \mid Y = c) = \frac{\#(a_i, Y = c)}{\#(c)}
\]

! Needs much data to get reliable values

! How about rare values?
Naive Bayes: Training

\[ P(X_i = a_i \mid Y = c) \]

count occurrences of each values in the data, for each class and each attribute

\[
P(X_i = a_i \mid Y = c) = \frac{\#(a_i, Y = c)}{\#(c)}
\]

Needs much data to get reliable values

How about rare values?

→ use Laplacian smoothing

\[
P(X_i = a_i \mid Y = c) = \frac{\#(a_i, Y = c) + \alpha}{\#(c) + \kappa \cdot \alpha}
\]

where \( \kappa \) is the number of distinct values of attribute \( X_i \)

That is, denoting the domain of \( X_i \) as \( A_i \), we let \( \kappa = |A_i| \)

so that \[
\sum_{a_i \in A_i} P(X_i = a_i \mid Y = c) = 1
\]
Naive Bayes: Training

\[ P(X_i = a_i | Y = c) \] count occurrences of each values in the data, for each class and each attribute

\[ P(X_i = a_i | Y = c) = \frac{#(a_i, Y = c)}{#(c)} \]

! Needs much data to get reliable values

! How about continuous domains?
Naive Bayes: Training

\[ P(X_i = a_i | Y = c) \] count occurrences of each values in the data, for each class and each attribute

\[ P(X_i = a_i | Y = c) = \frac{\#(a_i, Y = c)}{\#(c)} \]

! Needs much data to get reliable values

! How about continuous domains?

→ model with Gaussian distributions

\[ P(X = v | \mu, \sigma) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(v-\mu)^2}{2\sigma^2}} \]
Naive Bayes: Training

\[ P(Y = c) \text{ count occurrences of each class in the data} \]
\[ P(X_i = a_i \mid Y = c) \text{ model with Gaussian distributions, estimating the parameters from the data} \]
Naive Bayes: Training

\[ P(Y = c) \] count occurrences of each class in the data

\[ P(X_i = a_i | Y = c) \] model with Gaussian distributions, estimating the parameters from the data
**Naive Bayes: Training**

\[ P(Y = c) \] count occurrences of each class in the data

\[ P(X_i = a_i | Y = c) \] model with Gaussian distributions, estimating the parameters from the data

![Graph showing sepal length vs. petal length with Gaussian distributions](image)

\[ P(v | \mu_{sl}, \sigma_{sl}) = \frac{1}{\sqrt{2\pi\sigma_{sl}^2}} e^{-\frac{(v - \mu_{sl})^2}{2\sigma_{sl}^2}} \]

\[ \mu_{sl} = \text{mean}(sl | \bullet) = 5.945 \]

\[ \sigma_{sl}^2 = \text{var}(sl | \bullet) = 0.285 \]
Naive Bayes: Training

\[ P(Y = c) \text{ count occurrences of each class in the data} \]

\[ P(X_i = a_i \mid Y = c) \text{ model with Gaussian distributions, estimating the parameters from the data} \]

\[
P(v \mid \mu_{pl}, \sigma_{pl}) = \frac{1}{\sqrt{2\pi\sigma_{pl}^2}} e^{-\frac{(v - \mu_{pl})^2}{2\sigma_{pl}^2}}
\]

\[
\mu_{pl} = \text{mean}(pl \mid \bullet) = 4.252
\]

\[
\sigma_{pl}^2 = \text{var}(pl \mid \bullet) = 0.219
\]
Naive Bayes: Training

\[ P(Y = c) \] count occurrences of each class in the data

\[ P(X_i = a_i | Y = c) \] model with Gaussian distributions, estimating the parameters from the data

\[ P(v | \mu_{sl}, \sigma_{sl}) \]
\[ P(v | \mu_{pl}, \sigma_{pl}) \]
Naive Bayes: Training

\[ P(Y = c) \] count occurrences of each class in the data

\[ P(X_i = a_i | Y = c) \] model with Gaussian distributions, estimating the parameters from the data

\[
P(v | \mu_{sl}, \sigma_{sl}) = \frac{1}{\sqrt{2\pi\sigma_{sl}^2}} e^{-\frac{(v-\mu_{sl})^2}{2\sigma_{sl}^2}}
\]

\[
\mu_{sl} = \text{mean}(sl | \bullet) = 6.628
\]

\[
\sigma_{sl}^2 = \text{var}(sl | \bullet) = 0.414
\]
Naive Bayes: Training

\[ P(Y = c) \] count occurrences of each class in the data

\[ P(X_i = a_i \mid Y = c) \] model with Gaussian distributions, estimating the parameters from the data

\[
P(v \mid \mu_{pl}, \sigma_{pl}) = \frac{1}{\sqrt{2\pi\sigma_{pl}^2}} e^{-\frac{(v-\mu_{pl})^2}{2\sigma_{pl}^2}}
\]

\[
\mu_{pl} = \text{mean}(pl \mid \bullet) = 5.575
\]

\[
\sigma_{pl}^2 = \text{var}(pl \mid \bullet) = 0.308
\]
Naive Bayes: Training

\[ P(Y = c) \text{ count occurrences of each class in the data} \]

\[ P(X_i = a_i | Y = c) \text{ model with Gaussian distributions, estimating the parameters from the data} \]

\[ P(v | \mu_{sl}, \sigma_{sl}) \]

\[ P(v | \mu_{pl}, \sigma_{pl}) \]
Naive Bayes: Training

\[ P(Y = c) \] count occurrences of each class in the data

\[ P(X_i = a_i \mid Y = c) \] model with Gaussian distributions, estimating the parameters from the data
Naive Bayes: Prediction

Compute the conditional probability of each class according to Bayes’ rule

\[
P(Y = c \mid X_1 = a_1, \ldots, X_m = a_m) \propto P(Y = c) \cdot \prod_{i=1}^{i=m} P(X_i = a_i \mid Y = c)
\]
Naive Bayes: Prediction

Compute the conditional probability of each class according to Bayes’ rule

\[ P(\bullet | (5.6, 3.6)) \overset{?}{\geq} P(\diamond | (5.6, 3.6)) \]
Naive Bayes: Prediction

Compute the conditional probability of each class according to Bayes’ rule

\[ P(\bullet | (5.6, 3.6)) < P(\bullet | (5.6, 3.6)) \]

0.022 < 0.978
Naive Bayes: Prediction

Compute the conditional probability of each class according to Bayes’ rule

\[ P(\bullet | (5.6, 3.6)) < P(\bullet | (5.6, 3.6)) \]

0.022 < 0.978
Naive Bayes: Prediction

Compute the conditional probability of each class according to Bayes’ rule

\[ P(\bullet | x) \] and \[ P(\bullet | x) \] can be computed for every point
Naive Bayes: Prediction

Compute the conditional probability of each class according to Bayes’ rule

\[ P(\bullet \mid x) \text{ and } P(\bullet \mid x) \text{ can be computed for every point} \]

The decision boundary is the line \[ P(\bullet \mid x) = P(\bullet \mid x) = .5 \]
Perceptron

Input: data set $D$, data point $x$

Parameters: learning rate $\eta$

Training: initialize weights vector $\mathbf{w}$ and bias $b$, iterate among points in $D$ and adjust $\mathbf{w}$ and $b$

Prediction: return class according to sign of $\mathbf{w} \cdot \mathbf{x} + b$
Perceptron: Training

Iterate among points $\mathbf{x}^{(i)}$ from $\mathcal{D}$ in a random order and adjust weights $\mathbf{w}$ and bias $b$

At step $t$,
compute current prediction
$$z_i = \text{sign}(\mathbf{w}(t) \cdot \mathbf{x}^{(i)} + b(t))$$
update
$$\mathbf{w}(t+1) = \mathbf{w}(t) + \eta(y_j - z_j)x^{(i)}$$
$$b(t+1) = b(t) + \eta(y_j - z_j)$$
Perceptron: Training

Iterate among points $\mathbf{x}^{(i)}$ from $\mathcal{D}$ in a random order and adjust weights $\mathbf{w}$ and bias $b$

At step $t$, compute current prediction

$$z_i = \text{sign}(\mathbf{w}^{(t)} \cdot \mathbf{x}^{(i)} + b^{(t)})$$

update

$$\mathbf{w}^{(t+1)} = \mathbf{w}^{(t)} + \eta (y_j - z_j) \mathbf{x}^{(i)}$$
$$b^{(t+1)} = b^{(t)} + \eta (y_j - z_j)$$
Perceptron: Training

Iterate among points $x^{(i)}$ from $\mathcal{D}$ in a random order and adjust weights $w$ and bias $b$

At step $t$, compute current prediction

$$z_i = \text{sign}(w^{(t)} \cdot x^{(i)} + b^{(t)})$$

update

$$w^{(t+1)} = w^{(t)} + \eta (y_j - z_j)x^{(i)}$$

$$b^{(t+1)} = b^{(t)} + \eta (y_j - z_j)$$

Note that if prediction is correct, $w$ and $b$ are unchanged
Perceptron: Training

Iterate among points $\mathbf{x}^{(j)}$ from $\mathcal{D}$ in a random order and adjust weights $\mathbf{w}$ and bias $b$

At step $t$, compute current prediction

$$z_i = \text{sign}(\mathbf{w}^{(t)} \cdot \mathbf{x}^{(j)} + b^{(t)})$$

update

$$\mathbf{w}^{(t+1)} = \mathbf{w}^{(t)} + \eta (y_j - z_j) \mathbf{x}^{(j)}$$

$$b^{(t+1)} = b^{(t)} + \eta (y_j - z_j)$$

Note that if prediction is correct, $\mathbf{w}$ and $b$ are unchanged
Perceptron: Training

Iterate among points $x^{(i)}$ from $\mathcal{D}$ in a random order and adjust weights $w$ and bias $b$

Might cycle several times through all points of the training data
Each such cycle is called an *epoch*
Iterate among points $x^{(i)}$ from $D$ in a random order and adjust weights $w$ and bias $b$.

If the data is linearly separable, convergence on some solution is guaranteed.
Perceptron: Training

Iterate among points $x^{(i)}$ from $\mathcal{D}$ in a random order and adjust weights $\mathbf{w}$ and bias $b$

If the data is not linearly separable, learning will fail
Perceptron: Training

Iterate among points $x^{(i)}$ from $\mathcal{D}$ in a random order and adjust weights $w$ and bias $b$

If the data is not linearly separable, learning will fail.

The algorithm will not even approach an approximate solution.
Perceptron: Training

Iterate among points $x^{(i)}$ from $\mathcal{D}$ in a random order and adjust weights $w$ and bias $b$

If the data is not linearly separable, learning will fail

A quick fix: store best solution encountered and return it after chosen maximum number of epochs
Perceptron: Prediction

Return class according to $\text{sign}(w \cdot x + b)$

$$0.671 \cdot \text{sl} - 1.365 \cdot \text{pl} + 2.39 < 0$$

$$0.671 \cdot \text{sl} - 1.365 \cdot \text{pl} + 2.39 \geq 0$$

$$0.671 \cdot 6.0$$

$$-1.365 \cdot 4.8 + 2.39 = -0.136$$
Perceptron: Prediction

Return class according to \( \text{sign}(w \cdot x + b) \)

\[
0.671 \cdot \text{sl} - 1.365 \cdot \text{pl} + 2.39 < 0 \\
0.671 \cdot \text{sl} - 1.365 \cdot \text{pl} + 2.39 \geq 0
\]

\[
0.671 \cdot 6.0 \\
-1.365 \cdot 4.8 + 2.39 = -0.136
\]

predict
Perceptron: Prediction

Return class according to $\text{sign}(w \cdot x + b)$

\[ 0.671 \cdot \text{sl} - 1.365 \cdot \text{pl} + 2.39 < 0 \]

\[ 0.671 \cdot \text{sl} - 1.365 \cdot \text{pl} + 2.39 \geq 0 \]
Perceptron: Prediction

Return class according to $\text{sign}(w \cdot x + b)$

\[ 0.671 \cdot \text{sl} - 1.365 \cdot \text{pl} + 2.39 < 0 \]
\[ 0.671 \cdot \text{sl} - 1.365 \cdot \text{pl} + 2.39 \geq 0 \]
The Perceptron can be seen as the simplest neural network, a single-layer neural network

One *input node* for each data attribute
One *output node* computing the *activation function*
Support Vector Machine (SVM)

Input: data set $\mathcal{D}$, data point $x$

Parameters: penalty coefficient $C$

Training: solve for vector $w$ and bias $b$ that define a hyperplane separating points in $\mathcal{D}$ from the two classes with largest margin

Prediction: return class according to sign of $w \cdot x + b$
When the data is linearly separable, there might be multiple different separating hyperplanes. Which one to choose?
When the data is linearly separable, there might be multiple different separating hyperplanes. Which one to choose?

$m = 0.056$
When the data is linearly separable, there might be multiple different separating hyperplanes which one to choose?

\[ m = 0.041 \]
When the data is linearly separable, there might be multiple different separating hyperplanes. Which one to choose? Larger margin provides more stability.
Hard-margin SVM

Determining the maximum margin hyperplane

we have two hyperplanes, such that (under suitable scaling)

\[(H_+) \quad w \cdot x^{(j)} + b \geq +1 \quad \forall j, y_j = +1\]
\[(H_-) \quad w \cdot x^{(j)} + b \leq -1 \quad \forall j, y_j = -1\]

In short

\[y_j(w \cdot x^{(j)} + b) \geq 1 \quad \forall j\]
Hard-margin SVM

Determining the maximum margin hyperplane

We have two hyperplanes, such that (under suitable scaling)

\begin{align*}
(H_+) \quad w \cdot x^{(j)} + b & \geq +1 \quad \forall j, y_j = +1 \\
(H_-) \quad w \cdot x^{(j)} + b & \leq -1 \quad \forall j, y_j = -1
\end{align*}

The distance between $H_+$ and $H_-$ equals $2/\|w\|$
Hard-margin SVM

Determining the maximum margin hyperplane

the problem can be formulated as

\[
\begin{align*}
\text{minimize} & \quad \frac{1}{2} \|w\|^2 \\
\text{s.t.} & \quad y_j(w \cdot x^{(j)} + b) \geq 1 \quad \forall j
\end{align*}
\]
The problem can be formulated as

\[
\begin{align*}
\text{minimize} & \quad \frac{1}{2} \| \mathbf{w} \|^2 \\
\text{s.t.} & \quad y_j (\mathbf{w} \cdot \mathbf{x}^{(j)} + b) \geq 1 \quad \forall j
\end{align*}
\]

This is a quadratic constrained optimization problem.

It can be solved by the Lagrangian multiplier method.
Hard-margin SVM

Primal problem

\[
\min L_P = \frac{1}{2} \|w\|^2 + \sum_{j=1}^{j=n} a_j (1 - y_j (w \cdot x^{(j)} + b)) \quad \text{s.t.} \quad 0 \leq a_j \ \forall j
\]

Dual problem

\[
\max L_D = \sum_{j=1}^{j=n} a_j - \frac{1}{2} \sum_{j=1}^{j=n} \sum_{i=1}^{i=n} a_j a_i y_j y_i x^{(j)} \cdot x^{(i)}
\]

\[
\text{s.t.} \quad 0 \leq a_j \ \text{and} \ \sum_{j=1}^{j=n} a_j y_j = 0 \ \forall j
\]

variables \(a_j\) are defined in such a way that \(w = \sum_{j=1}^{j=n} a_j y_j x^{(j)}\)
Training the SVM means solving for the $a_j$ by differentiating the dual problem and setting it to zero.

Most of the $a_j$ will have value zero. Training points associated to non-zero $a_j$ are called support vectors, since they actually define the separating hyperplane.

$$w = \sum_{j=1}^{j=n} a_j y_j x^{(j)}$$

Support vectors satisfy

$$y_j (w \cdot x^{(j)} + b) = 1$$

which allows to compute $b$. 
Hard-margin SVM: Prediction

Return class according to $\text{sign}(\mathbf{w} \cdot \mathbf{x} + b)$, where

$$\mathbf{w} \cdot \mathbf{x} + b = b + \sum_{j=1}^{j=n} a_j y_j \mathbf{x}^{(j)} \cdot \mathbf{x}$$
What if the data is not linearly separable?

No hyperplane such that constraint

$$y_j(w \cdot x^{(j)} + b) \geq 1$$

would be satisfied by all points.

Some points inside the margin, or even on the wrong side of the separating hyperplane.
What if the data is not linearly separable?

Use the *hinge loss* and introduce a new variable for each point

\[ \xi_j = \max\left(0, 1 - y_j (w \cdot x^{(j)} + b)\right) \]

the value is

- **zero** if the point satisfies the margin constraint
- **proportional to the distance to the margin** otherwise
SVM: non linearly separable case

What if the data is not linearly separable?

**Hard-margin problem**

\[
\text{minimize} \quad \frac{1}{2} \| w \|^2 \\
\text{s.t.} \quad y_j (w \cdot x^{(j)} + b) \geq 1 \quad \forall j
\]

**Soft-margin problem**

\[
\text{minimize} \quad \frac{1}{2} \| w \|^2 + C \sum_{j=1}^{j=n} \xi_j \\
\text{s.t.} \quad y_j (w \cdot x^{(j)} + b) \geq 1 - \xi_j \text{ and } 0 \leq \xi_j \quad \forall j
\]
Soft-margin SVM

**Primal problem**

\[
\min L_P = \frac{1}{2} \|w\|^2 + C \sum_{j=1}^{j=n} \xi_j + \sum_{j=1}^{j=n} a_j (1 - \xi_j - y_j (w \cdot x^{(j)} + b)) - \sum_{j=1}^{j=n} \mu_j \xi_j
\]

s.t. \(0 \leq a_j\) and \(0 \leq \mu_j\) \(\forall j\)

**Dual problem**

\[
\max L_D = \sum_{j=1}^{j=n} a_j - \frac{1}{2} \sum_{j=1}^{j=n} \sum_{i=1}^{i=n} a_j a_i y_j y_i x^{(j)} \cdot x^{(i)}
\]

s.t. \(0 \leq a_j \leq C\) and \(\sum_{j=1}^{j=n} a_j y_j = 0 \forall j\)

variables \(a_j\) are defined in such a way that \(w = \sum_{j=1}^{j=n} a_j y_j x^{(j)}\)
Training the SVM means solving for the $a_j$ by differentiating the dual problem and setting it to zero.

Most of the $a_j$ will have value zero.

Training points associated to non-zero $a_j$ are called support vectors, since they actually define the separating hyperplane.

For every support vector

$$y_j(w \cdot x^{(j)} + b) = 1 - \xi_j$$

if $\xi_j = 0$, the point is on the margin.

otherwise, it is within the margin or even on the wrong side.
Soft-margin SVM: Training

Training the SVM means solving for the $a_j$ by differentiating the dual problem and setting it to zero.

Most of the $a_j$ will have value zero. Training points associated to non-zero $a_j$ are called support vectors, since they actually define the separating hyperplane.

Parameter $C$ allows to adjust the trade-off between width of the margin and constraint violations.
Soft-margin SVM: Training

Training the SVM means solving for the $a_j$ by differentiating the dual problem and setting it to zero.

Most of the $a_j$ will have value zero.

Training points associated to non-zero $a_j$ are called support vectors, since they actually define the separating hyperplane.

Parameter $C$ allows to adjust the trade-off between width of the margin and constraint violations.
Soft-margin SVM: Training

Training the SVM means solving for the $a_j$ by differentiating the dual problem and setting it to zero.

Most of the $a_j$ will have value zero.

Training points associated to non-zero $a_j$ are called support vectors, since they actually define the separating hyperplane.

Parameter $C$ allows to adjust the trade-off between width of the margin and constraint violations.
Soft-margin SVM: Prediction

Return class according to $\text{sign}(\mathbf{w} \cdot \mathbf{x} + b)$, where

$$\mathbf{w} \cdot \mathbf{x} + b = b + \sum_{j=1}^{j=n} a_j y_j x^{(j)} \cdot \mathbf{x}$$
Kernelized SVM

What if a linear decision boundary is not the right option?

Project the data to a different space

$$\phi(\langle x_1, x_2 \rangle) = \langle x_2 x_1, x_2 x_2 \rangle$$
Kernelized SVM

What if a linear decision boundary is not the right option?

Project the data to a different space $\phi(s(\langle x_1, x_2 \rangle)) = \langle x_2^1, x_2^2 \rangle$.
Kernelized SVM

What if a linear decision boundary is not the right option? Project the data to a different space
Kernelized SVM

What if a linear decision boundary is not the right option?
Train the SVM in the projected space

$$\varphi_s(\langle x_1, x_2 \rangle) = \langle x_1^2, x_2^2 \rangle$$
Kernelized SVM

What if a linear decision boundary is not the right option?
Train the SVM in the projected space

$$\varphi_S(\langle x_1, x_2 \rangle) = \langle x_1^2, x_2^2 \rangle$$
If the data cannot be separated in the original space

1. find a projection $\varphi$ to a space where the data can be separated

2. apply the SVM method to the transformed dataset
Kernelized SVM

If the data cannot be separated in the original space

1. find a projection $\varphi$ to a space where the data can be separated
2. apply the SVM method to the transformed dataset

Dual problem

$$\max L_D = \sum_{j=1}^{j=n} a_j - \frac{1}{2} \sum_{j=1}^{j=n} \sum_{i=1}^{i=n} a_j a_i y_j y_i \varphi(x^{(j)}) \cdot \varphi(x^{(i)})$$

s.t. $0 \leq a_j \leq C$ and $\sum_{j=1}^{j=n} a_j y_j = 0 \ \forall j$

Prediction

$$\text{sign} \left( b + \sum_{j=1}^{j=n} a_j y_j \varphi(x^{(j)}) \cdot \varphi(x^{(i)}) \right)$$
Kernelized SVM

Dual problem

$$\max L_D = \sum_{j=1}^{j=n} a_j - \frac{1}{2} \sum_{j=1}^{j=n} \sum_{i=1}^{i=n} a_j a_i y_j y_i \, \varphi(x^{(j)}) \cdot \varphi(x^{(i)})$$

s.t. $0 \leq a_j \leq C$ and $\sum_{j=1}^{j=n} a_j y_j = 0 \forall j$

Prediction

$$\text{sign} \left( b + \sum_{j=1}^{j=n} a_j y_j \, \varphi(x^{(j)}) \cdot \varphi(x^{(i)}) \right)$$

The transformed values $\varphi(x)$ appear only in dot products
The transformed values $\varphi(x)$ appear only in dot products.

If the dot product in the transformed space can be replaced by a function

$$K(x, x') = \varphi(x) \cdot \varphi(x')$$

we can avoid performing the transformation explicitly.
Kernelized SVM

Dual problem

$$\begin{align*}
\text{max } L_D &= \sum_{j=1}^{j=n} a_j - \frac{1}{2} \sum_{j=1}^{j=n} \sum_{i=1}^{i=n} a_j a_i y_j y_i \ K(x^{(j)}, x^{(i)}) \\
\text{s.t. } 0 \leq a_j \leq C \text{ and } \sum_{j=1}^{j=n} a_j y_j = 0 \ \forall j
\end{align*}$$

Prediction

$$\text{sign} \left( b + \sum_{j=1}^{j=n} a_j y_j K(x^{(j)}, x^{(i)}) \right)$$
The Kernel Trick

By replacing the dot product in the transformed space by a function

$$K(x, x') = \varphi(x) \cdot \varphi(x')$$

we can avoid performing the transformation explicitly.

The feature map $\varphi$ does not need to be explicitly defined. It is enough that $K$ be expressible as an inner product.

Mercer’s theorem gives the conditions for $K$ to be a valid kernel function.

In particular, the similarity matrix (a.k.a. Gram matrix) $S_{ij} = K(v_i, v_j)$ for a finite input space $\langle v_1, \ldots, v_l \rangle$ must be positive semi-definite (PSD).
The Kernel Trick

Improving the separability of data points typically means projecting into a high dimensional space. Computing with high dimensional vectors is costly, as the kernel function operates in the original space.

*The kernel trick provides the benefits of high-dimensionality without the costs*

Kernels are useful beyond SVMs, in other methods where dot products, i.e. similarity computations, are involved.
Common kernel functions

Polynomial kernel

\[ K(x, x') = (x \cdot x' + c)^h \]

Sigmoid kernel

\[ K(x, x') = \tanh(\kappa x \cdot x' - \delta) \]

Gaussian radial basis kernel

\[ K(x, x') = e^{-\|x - x'\|^2 / 2\sigma^2} \]
Kernelized SVM

\[ \varphi_S(\langle x_1, x_2 \rangle) = \langle x_1^2, x_2^2 \rangle \]
Consider the polynomial kernel of degree two

\[
K(x, x') = (x \cdot x')^2 = (x_1 x'_1 + x_2 x'_2)^2
= x_1^2 x'^2_1 + x_2^2 x'^2_2 + 2x_1 x'_1 x_2 x'_2
= \langle x_1^2, x_2^2, \sqrt{2}x_1 x_2 \rangle \cdot \langle x'^2_1, x'^2_2, \sqrt{2}x'_1 x'_2 \rangle
= \varphi_p(x) \cdot \varphi_p(x') \quad \text{where } \varphi_p(\langle x_1, x_2 \rangle) = \langle x_1^2, x_2^2, \sqrt{2}x_1 x_2 \rangle
\]

The terms of the feature map \( \varphi_s \) is a subset of those of \( \varphi_p \)
Kernelized SVM

Using the polynomial kernel

\[ K(x, x') = (x \cdot x')^2 \]

for the two-dimensional example dataset corresponds to projecting the points into three dimensional space

\[ \varphi_p(\langle x_1, x_2 \rangle) = \langle x_1^2, x_2^2, \sqrt{2}x_1x_2 \rangle \]

before training the SVM
Different kernels

<table>
<thead>
<tr>
<th>poly $h = 2$</th>
<th>poly $h = 4$</th>
<th>poly $h = 8$</th>
<th>rbf $\sigma = 1$</th>
<th>rbf $\sigma = .5$</th>
<th>rbf $\sigma = .1$</th>
</tr>
</thead>
</table>

UEF//School of Computing  JADe:Classification Basics
Evaluation
Model evaluation

Given a dataset, we can build a number of models, that come with different variants and parameter settings.

We need to quantify the accuracy of models, in order to:

- measure the effectiveness and tune the parameters of a particular model.
- compare, select, combine various models.
Evaluation measures

To evaluate the performance of a model we compare the known labels of the instances, representing the *ground truth*, to the predicted labels.

Let $y$ and $z$ denote respectively the true and predicted labels.

If there are $l$ distinct classes, there are $l \cdot l$ distinct possible outcomes for a given instance:

- The instance might belong to $c_1$ and be predicted as $c_1$.
- The instance might belong to $c_1$ and be predicted as $c_2$.
- \vdots
- The instance might belong to $c_l$ and be predicted as $c_1$.
- \vdots
- The instance might belong to $c_l$ and be predicted as $c_l$. 
Let $y$ and $z$ denote respectively the true and predicted labels. The outcome of the classification of a set of instances across $l$ classes can be summarized in a $l \times l$ contingency matrix.

<table>
<thead>
<tr>
<th>Prediction</th>
<th>Ground truth</th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>$z = c_1$</td>
<td>$(z = c_1, y = c_1)$</td>
<td>$(z = c_1, y = c_i)$</td>
<td>$(z = c_1, y = c_l)$</td>
<td></td>
</tr>
<tr>
<td>$z = c_i$</td>
<td>$(z = c_i, y = c_1)$</td>
<td>$(z = c_i, y = c_i)$</td>
<td>$(z = c_i, y = c_l)$</td>
<td></td>
</tr>
<tr>
<td>$z = c_l$</td>
<td>$(z = c_l, y = c_1)$</td>
<td>$(z = c_l, y = c_i)$</td>
<td>$(z = c_l, y = c_l)$</td>
<td></td>
</tr>
</tbody>
</table>
Evaluation measures

Let $y$ and $z$ denote respectively the true and predicted labels. The outcome of the classification of a set of instances across $l$ classes can be summarized in a $l \times l$ contingency matrix. The accuracy is the fraction of correctly classified instances.
Evaluation measures

Binary classification is a special case with specific terminology.

Two classes: *positive* and *negative*

There are four possible outcomes for a given instance.

The $2 \times 2$ contingency matrix is called *confusion matrix*.

<table>
<thead>
<tr>
<th>Prediction</th>
<th>Ground truth</th>
</tr>
</thead>
<tbody>
<tr>
<td>$z = 0$</td>
<td>$y = 0$</td>
</tr>
<tr>
<td>True negative</td>
<td>False negative</td>
</tr>
</tbody>
</table>
Binary classification is a special case with specific terminology.

Two classes: *positive* and *negative*

There are four possible outcomes for a given instance:

- True negative
- False negative
- False positive
- True positive

The $2 \times 2$ contingency matrix is called *confusion matrix*.

<table>
<thead>
<tr>
<th></th>
<th>$y = 0$</th>
<th>$y = 1$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$z = 0$</td>
<td>True negative</td>
<td>False negative</td>
</tr>
<tr>
<td>$z = 1$</td>
<td>False positive</td>
<td>True positive</td>
</tr>
</tbody>
</table>

**False positive:** type I error (a.k.a. false discovery, false alarm)

**False negative:** type II error (a.k.a. miss)
A binary classifier is trained on a portion of data (training data).

For example, consider a linear SVM.

Solve for vector $w$ and bias $b$ defining a separating hyperplane.
A binary classifier is trained on a portion of data (training data).

For example, consider a linear SVM.

Solve for vector $\mathbf{w}$ and bias $b$ defining a separating hyperplane.
Evaluation measures

A binary classifier is trained on a portion of data (training data), and applied on another portion for which the ground-truth is also known but hidden from the classifier (test data).

For example, consider a linear SVM.

For instance $x$, predict class according to sign of $w \cdot x + b$. 
Evaluation measures

A binary classifier is trained on a portion of data (\textit{training data}), and applied on another portion for which the ground-truth is also known but hidden from the classifier (\textit{test data})

For example, consider a linear SVM

For instance $\mathbf{x}$, predict class according to sign of $\mathbf{w} \cdot \mathbf{x} + b$
Evaluation measures

The outcome of the binary classification of a collection of $N$ instances can be summarized in a confusion matrix

$$
\begin{bmatrix}
N_{TN} & N_{FN} \\
N_{FP} & N_{TP}
\end{bmatrix}
$$
The outcome of the binary classification of a collection of \( N \) instances can be summarized in a confusion matrix:

\[
\begin{bmatrix}
N_{TN} & N_{FN} \\
N_{FP} & N_{TP}
\end{bmatrix}
\]

<table>
<thead>
<tr>
<th></th>
<th>( y = 0 )</th>
<th>( y = 1 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( z = 0 )</td>
<td>True negative</td>
<td>False negative</td>
</tr>
<tr>
<td>( z = 1 )</td>
<td>False positive</td>
<td>True positive</td>
</tr>
</tbody>
</table>
### Evaluation measures

$$\begin{bmatrix}
N_{TN} & N_{FN} \\
N_{FP} & N_{TP}
\end{bmatrix}$$

<table>
<thead>
<tr>
<th></th>
<th>$y = 0$</th>
<th>$y = 1$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$z = 0$</td>
<td>True negative</td>
<td>False negative</td>
</tr>
<tr>
<td>$z = 1$</td>
<td>False positive</td>
<td>True positive</td>
</tr>
</tbody>
</table>

Various measures can be computed from this matrix:

- **precision** \( \frac{N_{TP}}{N_{TP} + N_{FP}} \) (a.k.a. positive predictive value)
- **recall** \( \frac{N_{TP}}{N_{TP} + N_{FN}} \) (a.k.a. sensitivity, true positive rate)
- **specificity** \( \frac{N_{TN}}{N_{TN} + N_{FP}} \) (a.k.a. selectivity, true negative rate)
- **false positive rate** \( \frac{N_{FP}}{N_{FP} + N_{TN}} \)
- **false negative rate** \( \frac{N_{FN}}{N_{FN} + N_{TP}} \)
Evaluation measures

\[
\begin{bmatrix}
N_{TN} & N_{FN} \\
N_{FP} & N_{TP}
\end{bmatrix}
\]

<table>
<thead>
<tr>
<th>( y = 0 )</th>
<th>( y = 1 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( z = 0 )</td>
<td>True negative</td>
</tr>
<tr>
<td>( z = 1 )</td>
<td>False positive</td>
</tr>
</tbody>
</table>

Various measures can be computed from this matrix:

- **Precision**: \( \frac{N_{TP}}{N_{TP} + N_{FP}} \) (a.k.a. positive predictive value)
- **Recall**: \( \frac{N_{TP}}{N_{TP} + N_{FN}} \) (a.k.a. sensitivity, true positive rate)
- **F1 Score**: harmonic mean of recall and precision
  \[
  2 \cdot \frac{\text{precision} \cdot \text{recall}}{\text{precision} + \text{recall}} = \frac{2N_{TP}}{2N_{TP} + N_{FP} + N_{FN}}
  \]
Evaluation measures

$$\begin{bmatrix} N_{TN} & N_{FN} \\ N_{FP} & N_{TP} \end{bmatrix}$$

<table>
<thead>
<tr>
<th></th>
<th>$y = 0$</th>
<th>$y = 1$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$z = 0$</td>
<td>True negative</td>
<td>False negative</td>
</tr>
<tr>
<td>$z = 1$</td>
<td>False positive</td>
<td>True positive</td>
</tr>
</tbody>
</table>

Various measures can be computed from this matrix

**accuracy** fraction of instances in which the predicted label matches the ground truth

$$\frac{N_{TP} + N_{TN}}{N}$$
**Evaluation measures**

**accuracy** fraction of instances in which the predicted label matches the ground truth

\[
\frac{\sum_{c \in C} \#(y = c, z = c)}{\sum_{c \in C} \#(y = c)}
\]

In some cases, not all classes are equally important: misclassification in one class incurs a higher cost than misclassification in the other class reflected by weight \( w_c \) assigned to each class

**weighted accuracy** (a.k.a. cost-sensitive accuracy)

\[
\frac{\sum_{c \in C} w_c \cdot \#(y = c, z = c)}{\sum_{c \in C} w_c \cdot \#(y = c)}
\]
Evaluation measures

The outcome of the binary classification of a collection of \( N \) instances can be summarized in a confusion matrix. Various measures can be computed from this matrix:

\[
\begin{bmatrix}
N_{TN} & N_{FN} \\
N_{FP} & N_{TP}
\end{bmatrix}
\]

- \( \text{acc} = 0.812 \)
- \( \text{TPR} = 0.750 \)
- \( \text{FPR} = 0.125 \)
- \( \ldots \)
Evaluation measures

Instead of crisp class assignments we might consider a numerical score reflecting the confidence of the classifier. Class probabilities, distance from the decision boundary, etc.

For example, consider a linear SVM.

Use $\mathbf{w} \cdot \mathbf{x} + b$ as score for instance $\mathbf{x}$. 

![Graph showing sepal length vs. petal length with linear SVM decision boundary and data points indicating class separation.](image-url)
Evaluation measures

Varying the score above which an instance is assigned to the positive class means moving the decision boundary.

For instance $\mathbf{x}$, predict class according to the outcome of test $\mathbf{w} \cdot \mathbf{x} + b \geq \theta$

with $\theta = -9$, $\theta = -3$, $\theta = 5$, or $\theta = 9$, for example.
Evaluation measures

Let's look at what happens if we modify the level of confidence required to assign an instance to the positive class.

From almost certain that it does not belong to the negative class ($\theta \to -\infty$) to almost certain that it belongs to the positive class ($\theta \to +\infty$) and cases in between.
Consider a linear SVM, use $\mathbf{w} \cdot \mathbf{x} + b$ as score for instance $\mathbf{x}$
Collect the test instances with their scores
Consider a linear SVM, use $w \cdot x + b$ as score for instance $x$. Sort the test instances by increasing values of the score.
Evaluation

As thresholds we can use mid-point values between successive distinct score values among the test instances.

![Graph showing relationship between petal length and sepal length]
Set the threshold to the minimum value
Record the corresponding FPR and TPR

\[
\begin{align*}
\text{FPR} &= 1.000 \\
\text{TPR} &= 1.000 \\
\text{acc} &= 0.500
\end{align*}
\]
Evaluation

Set the threshold to the minimum value, raise it progressively
Record the corresponding FPR and TPR

FPR = 0.875
TPR = 1.000
acc = 0.562
Evaluation

Set the threshold to the minimum value, raise it progressively
Record the corresponding FPR and TPR

acc = 0.625
TPR = 1.000
FPR = 0.750
Evaluation

Set the threshold to the minimum value, raise it progressively. Record the corresponding FPR and TPR.

\[ FPR = 0.625 \]
\[ TPR = 1.000 \]
\[ acc = 0.688 \]
Set the threshold to the minimum value, raise it progressively
Record the corresponding FPR and TPR

\[
\text{acc} = 0.750 \\
\text{TPR} = 1.000 \\
\text{FPR} = 0.500
\]
Evaluation

Set the threshold to the minimum value, raise it progressively.
Record the corresponding FPR and TPR.

\[ \text{acc} = 0.688 \]
\[ \text{TPR} = 0.875 \]
\[ \text{FPR} = 0.500 \]
Set the threshold to the minimum value, raise it progressively
Record the corresponding FPR and TPR

\[
\text{acc} = 0.812 \\
\text{TPR} = 0.875 \\
\text{FPR} = 0.250
\]
Set the threshold to the minimum value, raise it progressively.
Record the corresponding FPR and TPR.

acc = 0.875
TPR = 0.875
FPR = 0.125
Set the threshold to the minimum value, raise it progressively.
Record the corresponding FPR and TPR.

\[ \text{acc} = 0.812 \]
\[ \text{TPR} = 0.750 \]
\[ \text{FPR} = 0.125 \]
Set the threshold to the minimum value, raise it progressively.
Record the corresponding FPR and TPR.

$\text{acc} = 0.875$
$\text{TPR} = 0.750$
$\text{FPR} = 0.000$
Evaluation

Set the threshold to the minimum value, raise it progressively.
Record the corresponding FPR and TPR.

\[ \text{acc} = 0.812 \]
\[ \text{TPR} = 0.625 \]
\[ \text{FPR} = 0.000 \]
Set the threshold to the minimum value, raise it progressively
Record the corresponding FPR and TPR

\[
\text{acc} = 0.750 \\
\text{TPR} = 0.500 \\
\text{FPR} = 0.000
\]
Evaluation

Set the threshold to the minimum value, raise it progressively. Record the corresponding FPR and TPR.

\[ \text{acc} = 0.688 \]
\[ \text{TPR} = 0.375 \]
\[ \text{FPR} = 0.000 \]
FPR and TPR depend on where the ranking is split between classes, not the specific threshold value.
Hence, successive FPR and TPR can be computed directly from the ranked instances.
The curve FPR vs. TPR is commonly referred to as receiver operating characteristic (ROC) curve.
The curve FPR vs. TPR is commonly referred to as **(ROC) curve**. The **area under the curve (AUC)** summarizes the ROC curve in a single number.

**AUC = 0.922**
The goal is not to best mimic the labels of the training data. For evaluation, we need labelled data points not seen during training.

*Compromise:* the more labelled data for training the better, but some labelled examples need to be held out for evaluation.
Divide the labelled data into two disjoint sets

**training data** used to train the model
  typically ca. 2/3 – 3/4 of data

**test data** used to evaluate the model
  typically ca. 1/3 – 1/4 of data
Hold-out

Divide the labelled data into two disjoint sets

**training data** used to train the model
  typically ca. 2/3 – 3/4 of data

**test data** used to evaluate the model
  typically ca. 1/3 – 1/4 of data

! only a fraction of data used for training
! error estimates are pessimistic
Hold-out

Divide the labelled data into two disjoint sets

**training data** used to train the model
typically ca. 2/3 – 3/4 of data

**test data** used to evaluate the model
typically ca. 1/3 – 1/4 of data

! only a fraction of data used for training
! error estimates are pessimistic

Repeat this process over several different hold-out samples

• improve the error estimate
• measure variance and compute statistical confidence intervals on the error
Cross-validation

Divide the labelled data into $\ell$ disjoint sets of equal size $\ell/n$
use one set as test data, remaining $\ell - 1$ as training data
repeat with each set as test data

In each of $\ell$ rounds, the training data has size $n(\ell - 1)/\ell$
This is called $\ell$-fold cross-validation
e.g. 10-fold cross-validation is common
Cross-validation

Divide the labelled data into $\ell$ disjoint sets of equal size $\ell/n$
use one set as test data, remaining $\ell - 1$ as training data
repeat with each set as test data

In each of $\ell$ rounds, the training data has size $n(\ell - 1)/\ell$
This is called $\ell$-fold cross-validation

For larger values of $\ell$,
the training data comprises more examples
  $\rightarrow$ better error estimation (still pessimistic)
more rounds are needed
  $\rightarrow$ higher computational cost
Cross-validation

Divide the labelled data into $\ell$ disjoint sets of equal size $\ell/n$
use one set as test data, remaining $\ell - 1$ as training data
repeat with each set as test data

In each of $\ell$ rounds, the training data has size $n(\ell - 1)/\ell$
This is called $\ell$-fold cross-validation

Extreme case: $\ell = n$
i.e. $\ell$ rounds with $n - 1$ training examples and 1 test example
This is called leave-one-out cross-validation
Sample training data of size $n$ from the labelled data uniformly with replacement

The training data has the same size as the original data but some original data examples might be duplicated while others might be missing.

The probability that a point is not included in the sample is $(1 – 1/n)^n$, which tends towards $1/e$ as $n$ increases.
Sample training data of size $n$ from the labelled data uniformly with replacement

Use the full original labelled data as test data

The large overlap between training and test data means the error estimate is highly optimistic

Repeating this process over several different bootstrap samples allows to compute the mean and variance of the error estimate
Generate $\ell$ bootstrap samples and use each such sample to train one classifier.

For labelled example $x$, evaluate $\text{acc}(x)$ the performance on $x$ of the classifiers trained on samples that do not contain $x$.

Average $\text{acc}(x)$ over all labelled examples.
A step of parameter tuning and model selection, called validation, might be needed.

Validation and test should not be carried out on the same set, since knowledge of the test set has been implicitly used while building the model.

A portion of the data is used to train different models and select one.

The performance of the selected model should be evaluated on a distinct, so far unseen, portion of the data.
Comparing models

The statistical robustness of models is important differences in accuracy between models might be due to random variations

Assume we have obtained $\ell$ estimates of the accuracy of two models $M_A$ and $M_B$ on different randomly sampled subsets of data (e.g. through repeated hold-out or bootstrap procedures) $\{\text{acc}(M_A, 1), \ldots, \text{acc}(M_A, \ell)\}$ and $\{\text{acc}(M_B, 1), \ldots, \text{acc}(M_B, \ell)\}$
Comparing models

Assume we have obtained $\ell$ estimates of the accuracy of two models $M_A$ and $M_B$ on different randomly sampled subsets of data (e.g. through repeated hold-out or bootstrap procedures) 

$\{\text{acc}(M_A, 1), \ldots, \text{acc}(M_A, \ell)\}$ and $\{\text{acc}(M_B, 1), \ldots, \text{acc}(M_B, \ell)\}$

Let $\delta_i$ be the difference in accuracy in round $i$, i.e.

$$\delta_i = \text{acc}(M_A, i) - \text{acc}(M_B, i)$$

the average difference in accuracy is $\Delta = \sum_{i=1}^{\ell} \delta_i / \ell$

and the standard deviation of the difference in accuracy is

$$\sigma = \sqrt{\frac{\sum_{i=1}^{\ell} (\delta_i - \Delta)^2}{\ell - 1}}$$
Comparing models

We assume that $\delta_i$ are sampled from a normal distribution with estimated mean and standard deviation $\Delta$ and $\sigma$, respectively. According to the central limit theorem, the standard deviation of the estimated mean accuracy difference $\Delta$ is $\sigma/\sqrt{\ell}$.

The number of standard deviations by which $\Delta$ is different from the break-even value of 0 is $\sqrt{\ell} |\Delta - 0| / \sigma$.

For sufficiently large number of rounds $\ell$, the probability that one model is truly better than the other can be quantified using the standard normal distribution.
Comparing models

For sufficiently large number of rounds $\ell$, the probability that one model is truly better than the other can be quantified using the standard normal distribution.

It is generally too computationally expensive to run sufficiently many rounds to robustly estimate $\sigma$ so the Student’s $t$-distribution with $\ell - 1$ degrees of freedom is used instead of the normal distribution.
Diagnostic

Classification can be seen as the problem of learning a function between the data attributes and the class label

\[ y = g(x) + \epsilon \]

\( g \) represents the true, unknown, relationship between data attributes and class label
\( \epsilon \) represents the intrinsic error in the data, the noise, cannot be modelled
Classification can be seen as the problem of learning a function between the data attributes and the class label

\[ y = g(x) + \epsilon \]

\( g \) represents the true, unknown, relationship between data attributes and class label, \textit{even the form of } \( g \) \textit{is unknown} \\
\( \epsilon \) represents the intrinsic error in the data, the noise, cannot be modelled

Classification algorithms construct models while relying on modeling assumptions about the form of the relationship

\[ z = f_D(x) \]
Classification algorithms construct models while relying on modeling assumptions about the form of the relationship

\[ z = f_D(x) \]

The function might be defined algorithmically or in closed form. The parameters of the function are estimated from the data:

- choice of approach: *Family*
- choice of setup: *Species*
- estimation from \( \mathcal{D} \): *Individual*
Classification algorithms construct models while relying on modeling assumptions about the form of the relationship

\[ z = f_D(x) \]

The function might be defined algorithmically or in closed form. The parameters of the function are estimated from the data.

choice of approach  \( Family \)  decision tree
choice of setup  \( Species \)  max depth, max leaf size
estimation from \( D \)  \( Individual \)  intermediate tests, decisions

\( f_D(x) \) is defined algorithmically
Classification algorithms construct models while relying on modeling assumptions about the form of the relationship

$$z = f_D(x)$$

The function might be defined algorithmically or in closed form. The parameters of the function are estimated from the data.

- choice of approach: Family
- choice of setup: Species
- estimation from $\mathcal{D}$: Individual
- $a_1, \ldots, a_n, b$

$$f_D(x) = \text{sign} \left( b + \sum_{j=1}^{n} a_j y_j x^{(j)} \cdot x \right)$$
Classification algorithms construct models while relying on modeling assumptions about the form of the relationship

\[ z = f_\mathcal{D}(x) \]

The function might be defined algorithmically or in closed form. The parameters of the function are estimated from the data.

**choice of approach**  
*Family*  
kernelized SVM

**choice of setup**  
*Species*  
kernel function \( K \)

**estimation from \( \mathcal{D} \)**  
*Individual*  
\( a_1, \ldots, a_n, b \)

\[
f_\mathcal{D}(x) = \text{sign} \left( b + \sum_{j=1}^{j=n} a_j y_j K(x^{(j)}, x^{(i)}) \right)
\]
Diagnostic

Classification algorithms rely on modeling assumptions estimate the parameters from the data
Classification algorithms rely on modeling assumptions estimate the parameters from the data

Assumptions may not reflect the true form of the relationship

Oversimplifying assumptions do not allow to capture the underlying structure of the data → Underfitting
Classification algorithms rely on modeling assumptions to estimate the parameters from the data. Even with correct modeling assumptions, the true parameters cannot be estimated exactly from the training data. With increasingly complex models, i.e., more parameters, the model might fit too closely to the training data and thus overfit the data. This means the model captures the structure of the data but also noise, and it does not generalize well to unseen data, leading to overfitting.
Imagine we had a very large dataset and could repeat the whole model training many times, we could estimate the expected prediction $E_D[f_D(x)]$

Because of differences between the assumed model and the true model, $g(x)$ and $E_D[f_D(x)]$ would differ

$\rightarrow$ Bias

For a fixed test instance $x$, the value $f_D(x)$ would vary for different instantiations of the training data $D$

$\rightarrow$ Variance
Bias and variance
The expected mean squared error of the prediction for test data points \( \{(x_1, y_1), \ldots, (x_l, y_l)\} \) can be written as

\[
E_D[\text{MSE}] = \frac{1}{l} \sum_{i=1}^{l} (g(x_i) - E_D[f_D(x_i)])^2 + E_D[(f_D(x_i) - E_D[f_D(x_i)])^2] + (y_i - g(x_i))^2
\]

bias + variance + noise
Diagnostic

UNDERFITTING

OVERFITTING

error

bias

validation

variance

training

model complexity
High bias model—Underfitting

Quick convergence to high error on training and validation sets
More training data brings little improvement
Increase model complexity to allow better fit
High variance model–Overfitting

Performance gap between training and validation sets
More training data can bring improvement
Limit model complexity by using e.g. regularization, pruning, early-stopping
## Bias vs. Variance

<table>
<thead>
<tr>
<th>Higher Bias</th>
<th>Lower Bias</th>
</tr>
</thead>
<tbody>
<tr>
<td>Lower Variance</td>
<td>Higher Variance</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Lower Model Complexity</th>
<th>Higher Model Complexity</th>
</tr>
</thead>
<tbody>
<tr>
<td>Shallow Decision Tree</td>
<td>Deep Decision Tree</td>
</tr>
<tr>
<td>$k$-NN with many neighbors</td>
<td>$k$-NN with few neighbors</td>
</tr>
<tr>
<td>Linear SVM</td>
<td>Kernel SVM</td>
</tr>
<tr>
<td>SVM RBF kernel large $\sigma$</td>
<td>SVM RBF kernel small $\sigma$</td>
</tr>
</tbody>
</table>