Algorithmic Data Analysis

Esther Galbrun
Autumn 2022
Part I

Classification variants
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

virginica

sepal length

petal length

petal

sepal

UEF/School of Computing  ADA:Classification
A simple example

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

**Classification** aim to assign a class label to each instance
A simple example

binary there are two classes to choose from
**A simple example**

supervised labelled training instances are available
A simple example

**supervised** labelled training instances are available

**binary** there are two classes to choose from

**classification** aim to assign a class label to each instance

A typical **supervised binary classification** problem
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.
A typical supervised binary classification problem
Various classification methods are available to tackle 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

- sepal length
- petal length

pl ≥ 5.15
pl ≥ 4.85
sl ≥ 6.60
sl ≥ 4.95
Different methods

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

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

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

\[
P(\bullet | sl, sp) \leq P(\circ | sl, sp)
\]
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

A typical supervised binary classification problem

Various classification methods are available to tackle it

- $k$-NN
- decision tree
- naive Bayes
- perceptron
- SVM
A simple example

A typical supervised binary classification problem
Various classification methods are available to tackle it

Problem variants

• What if there are more than two classes?
  → Multi-class learning

• What if the two classes are not equally represented?
  → Rare-class learning

Methods

• How about combining multiple classifiers?
  → Ensemble methods
Multi-class learning
More irises

How about telling apart three species of irises?
No adaptation needed

Some methods can handle multiple classes
→ $k$ nearest neighbors ($k$-NN)
No adaptation needed

Some methods can handle multiple classes
→ classification tree
No adaptation needed

Some methods can handle multiple classes

→ Naive bayes
Adaptations needed

Other methods, like the Perceptron and SVMs are naturally designed for the binary scenario.

Method-specific adaptations to the multi-class scenario exist.

Generic, method-agnostic, meta-frameworks are helpful.

Two main strategies:

- **one-against-rest**
- **one-against-one**
One-against-rest

Create a new binary classification problem for each class: examples from that class are constitute **positive** examples, the rest are **negative** examples.
One-against-rest

Create a new binary classification problem for each class: examples from that class are constitute **positive** examples the rest are **negative** examples
One-against-rest

Create a new binary classification problem for each class: examples from that class are constitute positive examples, the rest are negative examples.

![Graph showing sepal length vs. sepal width for each class with positive and negative examples represented.]
One-against-rest

Predictions from the different problems are then combined

---

<table>
<thead>
<tr>
<th>sepal length</th>
<th>sepal width</th>
</tr>
</thead>
<tbody>
<tr>
<td>4.5</td>
<td>2.0</td>
</tr>
<tr>
<td>5.0</td>
<td>2.5</td>
</tr>
<tr>
<td>5.5</td>
<td>3.0</td>
</tr>
<tr>
<td>6.0</td>
<td>3.5</td>
</tr>
<tr>
<td>6.5</td>
<td>4.0</td>
</tr>
<tr>
<td>7.0</td>
<td>4.5</td>
</tr>
<tr>
<td>7.5</td>
<td>5.0</td>
</tr>
<tr>
<td>8.0</td>
<td>5.5</td>
</tr>
</tbody>
</table>

---

<table>
<thead>
<tr>
<th>a</th>
<th>b</th>
<th>c</th>
<th>d</th>
<th>e</th>
<th>f</th>
</tr>
</thead>
<tbody>
<tr>
<td>⭐️⭐️⭐️⭐️</td>
<td>⭐️⭐️⭐️</td>
<td>⭐️⭐️⭐️</td>
<td>⭐️⭐️⭐️</td>
<td>⭐️⭐️⭐️</td>
<td>⭐️⭐️⭐️</td>
</tr>
<tr>
<td>⭐️⭐️⭐️⭐️</td>
<td>⭐️⭐️⭐️</td>
<td>⭐️⭐️⭐️</td>
<td>⭐️⭐️⭐️</td>
<td>⭐️⭐️⭐️</td>
<td>⭐️⭐️⭐️</td>
</tr>
<tr>
<td>⭐️⭐️⭐️⭐️</td>
<td>⭐️⭐️⭐️</td>
<td>⭐️⭐️⭐️</td>
<td>⭐️⭐️⭐️</td>
<td>⭐️⭐️⭐️</td>
<td>⭐️⭐️⭐️</td>
</tr>
<tr>
<td>⭐️⭐️⭐️⭐️</td>
<td>⭐️⭐️⭐️</td>
<td>⭐️⭐️⭐️</td>
<td>⭐️⭐️⭐️</td>
<td>⭐️⭐️⭐️</td>
<td>⭐️⭐️⭐️</td>
</tr>
</tbody>
</table>
One-against-rest

Predictions from the different problems are then combined.
Predictions from the different problems are then combined. Might require tie-breaking, using weighted rather than crisp votes can help.
One-against-rest

A $k$ class problem maps to $k$ binary models.
One-against-one

Create a new binary classification problem for each pair of classes, considering only examples from these two classes.
Create a new binary classification problem for each pair of classes, considering only examples from these two classes.
One-against-one

Create a new binary classification problem for each pair of classes, considering only examples from these two classes.

vs.

Graphs showing scatter plots of sepal length vs. sepal width for different classes.
One-against-one

Predictions from the different problems are then combined

![Graph](image_url)

Legend:
- a
- b
- c
- d
- e
- f

UEF/School of Computing  ADA:Classification
One-against-one

Predictions from the different problems are then combined.
Predictions from the different problems are then combined. Might require tie-breaking, using weighted rather than crisp votes can help.

---

### One-against-one

- Predictions from the different problems are then combined.
- Might require tie-breaking, using weighted rather than crisp votes can help.

---

**Graph:***

- **Sepal length** vs **Sepal width**
- Points labeled from 'a' to 'f'
- Different colors and shapes represent different classes.

---

**Table:***

<table>
<thead>
<tr>
<th></th>
<th>a</th>
<th>b</th>
<th>c</th>
<th>d</th>
<th>e</th>
<th>f</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>⭐</td>
<td>⭐</td>
<td>⭐</td>
<td>⭐</td>
<td>⭐</td>
<td>⭐</td>
</tr>
<tr>
<td>b</td>
<td>⭐</td>
<td>⭐</td>
<td>⭐</td>
<td>⭐</td>
<td>⭐</td>
<td>⭐</td>
</tr>
<tr>
<td>c</td>
<td>⭐</td>
<td>⭐</td>
<td>⭐</td>
<td>⭐</td>
<td>⭐</td>
<td>⭐</td>
</tr>
<tr>
<td>d</td>
<td>⭐</td>
<td>⭐</td>
<td>⭐</td>
<td>⭐</td>
<td>⭐</td>
<td>⭐</td>
</tr>
<tr>
<td>e</td>
<td>⭐</td>
<td>⭐</td>
<td>⭐</td>
<td>⭐</td>
<td>⭐</td>
<td>⭐</td>
</tr>
<tr>
<td>f</td>
<td>⭐</td>
<td>⭐</td>
<td>⭐</td>
<td>⭐</td>
<td>⭐</td>
<td>⭐</td>
</tr>
</tbody>
</table>
One-against-one

A $k$ class problem maps to $\binom{k}{2} = k(k - 1)/2$ binary models.
A $k$ class problem maps to $\binom{k}{2} = k(k - 1)/2$ binary models

More problems than one-against-rest, but smaller
Rare-class learning
Normal banknotes are much more common than fraudulent banknotes (343 to 37)

Under such class ratio in the test data, trivially predicting everything as normal yields 90% accuracy

False negatives have higher consequences than false positives

Need to emphasize the greater importance of the rare class
Rare-class scenario

It is important to achieve high accuracy on the rare class, at the cost of reduced accuracy on the normal class. Associate different weights to the classes and try to maximize the weighted accuracy.

Normal banknotes are much more common than fraudulent banknotes (343 to 37)
Rare-class scenario

Normal banknotes are much more common than fraudulent banknotes (343 to 37)

Two main strategies

example reweighting and example resampling
Rare-class scenario

Example reweighting

- weights are associated to training examples according to their missclassification cost
- algorithms require adaptations to handle these weights

Example resampling

- examples from rare class might be oversampled, or examples from normal class be undersampled, or a combination of both
- algorithms do not require any adaptation
Rare-class scenario

Example reweighting with $k$ nearest neighbors

Identify the $k$ nearest neighbors, assign weights according to their class when deciding majority.
Rare-class scenario with $k$ nearest neighbors

- original
- undersampling
- oversampling
- reweighting

<table>
<thead>
<tr>
<th>Method</th>
<th>acc</th>
<th>wacc</th>
</tr>
</thead>
<tbody>
<tr>
<td>original</td>
<td>0.90</td>
<td>0.64</td>
</tr>
<tr>
<td>undersampling</td>
<td>0.90</td>
<td>0.93</td>
</tr>
<tr>
<td>oversampling</td>
<td>1.00</td>
<td>1.00</td>
</tr>
<tr>
<td>reweighting</td>
<td>1.00</td>
<td>1.00</td>
</tr>
</tbody>
</table>

UEF/School of Computing  ADA:Classification
Rare-class scenario

Example reweighting with naive Bayes

Assign weights to instances when computing the classes prior probabilities
Rare-class scenario with Naive Bayes

Original undersampling oversampling reweighting

acc = 0.85
wacc = 0.61

acc = 0.85
wacc = 0.75

acc = 0.85
wacc = 0.75

acc = 0.80
wacc = 0.86
In effect, **resampling** and **reweighting** are almost equivalent. **Resampling** can be understood as sampling examples in proportion to their *weights* then treating them equally.

**Resampling** is easier to combine with other approaches.

**Undersampling** is more efficient (smaller datasets).

**Resampling** has greater randomness.

**Reweighting** is more reliable.
Ensemble methods
Different classifiers might make different predictions on the same data point due to their specific characteristics or their sensitivity to random artifact in the training data.

The aim of ensemble methods is to increase prediction accuracy by combining the results of multiple classifiers.
For $i = 1, \ldots, \ell$, train model $M^{(i)}$ on dataset $D^{(i)}$
Combine the predictions of the different models into a single robust prediction

**Data-centered ensembles** use a single algorithm on different derivative datasets

**Model-centered ensembles** use different algorithms or different parameter settings of the same algorithm on a single dataset
It is often difficult to know beforehand which classifier will work well on a particular dataset.

The training dataset is divided into two subsets $\mathcal{D}_A$ and $\mathcal{D}_B$. $\mathcal{D}_A$ is used to train different models, $\mathcal{D}_B$ is used to evaluate their performance. The best model is selected and retrained on the full dataset.

Cross-validation can be used for evaluation instead of hold-out.

The models can correspond to different algorithms or to different parameter settings of the same algorithm.
The performance of the bucket of models is only as good as the best model in the bucket for a particular dataset. Over multiple datasets, the approach is able to select the model that is best suited to each case.
Bagging

If the variance of a single prediction is $\sigma$, the variance of the average of $\ell$ independent and identically distributed (i.i.d.) such predictions is reduced to $\sigma^2/\ell$.

Derivative datasets are created using **bootstrap sampling** $D^{(i)}$ is a subset of data points sampled uniformly with replacement from $D$ to approximately the same size as $D$.

Report the majority vote among the predictions of the models as the ensemble’s prediction.

**Bagging** (a.k.a. bootstrapped aggregating) helps **reduce variance** through aggregation.

Individual models should be designed so as to reduce bias as much as possible, even at the expense of variance.
Bagging

If the variance of a single prediction is $\sigma$, the variance of the average of $\ell$ independent and identically distributed (i.i.d.) such predictions is reduced to $\sigma^2/\ell$

If the predictors have pairwise correlation of $\rho$ between them, the variance of the average prediction is $\rho \cdot \sigma^2 + (1 - \rho)\sigma^2 / \ell$

where $\rho \cdot \sigma^2$ is invariant to the number of components in the ensemble and limits the performance gains
If the variance of a single prediction is $\sigma$, the variance of the average of $\ell$ independent and identically distributed (i.i.d.) such predictions is reduced to $\sigma^2/\ell$

If the predictors have pairwise correlation of $\rho$ between them, the variance of the average prediction is $\rho \cdot \sigma^2 + (1 - \rho)\sigma^2/\ell$

where $\rho \cdot \sigma^2$ is invariant to the number of components in the ensemble and limits the performance gains

! When using **bagging** with decision trees split choices at the top levels likely remain invariant to bootstrapped sampling → resulting decision trees are correlated → error reduction from aggregation is curtailed
A random forest is an ensemble of decision trees where randomness is added explicitly at the split selection to reduce correlation between the components.

During tree construction, each split selection is preceded by the random selection of $q$ attributes, among which the split criterion is then chosen, rather than from the entire set of $m$ attributes.
Random forests

During tree construction, each split selection is preceded by the random selection of \( q \) attributes, among which the split criterion is then chosen, rather than from the entire set of \( m \) attributes.

Parameter \( q \) regulates the amount of randomness:

- **small** \( q \) leads to more randomness, less correlations across components and more efficient tree growth.
- **large** \( q \) leads to more accurate individual components.

\[
q = \log_2(m) + 1 \]

has been shown to achieve good trade-off.

\[
q = 1 \]

(i.e. totally random trees) can achieve good accuracy in aggregation but requires a large number of components.
Random forests

During tree construction, each split selection is preceded by the random selection of $q$ attributes, among which the split criterion is then chosen, rather than from the entire set of $m$ attributes.

This approach based on *random input selection* is referred to as *Forest-RI*.

When $m$ is small this approach does not work well. Instead, generate a subset of $q$ linear combinations of attributes with random coefficients in $[-1, 1]$. This approach based on *random linear combinations* is referred to as *Forest-RC*.
Random forests

During tree construction, each split selection is preceded by the random selection of $q$ attributes, among which the split criterion is then chosen, rather than from the entire set of $m$ attributes.

Each tree is grown without pruning, on a bootstrapped sample.

Restricted split selection increases bias of individual components and leads to problems when the fraction of informative attributes is small.

Aggregation provides variance reduction.

Random forests are quite resistant to noise and outliers.
Boosting

**weak learner**  a classifier that is only slightly correlated with the ground truth, i.e. one that performs only slightly better than random guessing

**strong learner**  a classifier that is arbitrarily well correlated with the ground truth, i.e. one of arbitrarily high accuracy

*Hypothesis boosting* aims to turn a weak learner into a strong learner
Boosting

Successive models $M^{(t)}$ are built by applying the same algorithm to weighted variants $D^{(t)}$ of the dataset.

Weight associated to every training instance are adjusted so that the model will focus more on previously misclassified instances.

The prediction of the ensemble is a weighted combination of all the models’ predictions.

Many boosting algorithms have been proposed. **AdaBoost** (short for Adaptive Boosting) is most popular.
AdaBoost

$t \leftarrow 1; \quad w_i^{(t)} \leftarrow 1/n, \quad i = 1, \ldots, n$

repeat

Train model $M^{(t)}$ on $D$ weighted by $w^{(t)}$

$\epsilon_t \leftarrow$ corresponding training error rate

$\alpha_t \leftarrow \ln((1 - \epsilon_t)/\epsilon_t)/2$

$w_i^{(t+1)} \leftarrow \begin{cases} 
  w_i^{(t)} e^{-\alpha_t} & \text{if instance } i \text{ is correctly classified} \\
  w_i^{(t)} e^{\alpha_t} & \text{otherwise} 
\end{cases}$

$t \leftarrow t + 1; \quad w^{(t)} \leftarrow w^{(t)}/\sum_i w_i^{(t)}$

until $t > T$ or $\epsilon_{t-1} = 0$ or $\epsilon_{t-1} \geq 0.5$
AdaBoost

The algorithm starts with equal weights for all instances

t \leftarrow 1; \quad w_i^{(t)} \leftarrow 1/n, \ i = 1, \ldots, n

repeat

Train model M^{(t)} on D weighted by \ w^{(t)}

\epsilon_t \leftarrow \text{corresponding training error rate}

\alpha_t \leftarrow \ln\left((1 - \epsilon_t)/\epsilon_t\right)/2

w_i^{(t+1)} \leftarrow \begin{cases} 
    w_i^{(t)} e^{-\alpha_t} & \text{if instance } i \text{ is correctly classified} \\
    w_i^{(t)} e^{\alpha_t} & \text{otherwise}
\end{cases}

\quad t \leftarrow t + 1; \quad w^{(t)} \leftarrow w^{(t)}/\sum_i w_i^{(t)}

until \ t > T \ or \ \epsilon_{t-1} = 0 \ or \ \epsilon_{t-1} \geq 0.5
Weights can be incorporated directly to the algorithm or via sampling

\[
\begin{align*}
t &\leftarrow 1; \quad w_{i}^{(t)} \leftarrow 1/n, \ i = 1, \ldots, n \\
\text{repeat} & \\
& \text{Train model } M^{(t)} \text{ on } \mathcal{D} \text{ weighted by } w^{(t)} \\
& \epsilon_t \leftarrow \text{corresponding training error rate} \\
& \alpha_t \leftarrow \ln((1 - \epsilon_t)/\epsilon_t)/2 \\
& w_{i}^{(t+1)} \leftarrow \begin{cases} \\
& w_{i}^{(t)} e^{-\alpha_t} \text{ if instance } i \text{ is correctly classified} \\
& w_{i}^{(t)} e^{\alpha_t} \text{ otherwise} \\
\end{cases} \\
& t \leftarrow t + 1; \quad w^{(t)} \leftarrow w^{(t)}/\sum_{i} w_{i}^{(t)} \\
\text{until } & \ t > T \text{ or } \epsilon_{t-1} = 0 \text{ or } \epsilon_{t-1} \geq 0.5
\end{align*}
\]
AdaBoost

$\epsilon_t$ is the fraction of training instances misclassified by $M^{(t)}$

$t \leftarrow 1; \quad w_i^{(t)} \leftarrow 1/n, \; i = 1, \ldots, n$

repeat

Train model $M^{(t)}$ on $D$ weighted by $w^{(t)}$

$\epsilon_t \leftarrow$ corresponding training error rate

$\alpha_t \leftarrow \ln\left(\frac{(1 - \epsilon_t)/\epsilon_t}{\epsilon_t}\right)/2$

$w_i^{(t+1)} \leftarrow \begin{cases} 
  w_i^{(t)}e^{-\alpha_t} & \text{if instance } i \text{ is correctly classified} \\
  w_i^{(t)}e^{\alpha_t} & \text{otherwise}
\end{cases}$

$t \leftarrow t + 1; \quad w^{(t)} \leftarrow w^{(t)}/\sum_i w_i^{(t)}$

until $t > T$ or $\epsilon_{t-1} = 0$ or $\epsilon_{t-1} \geq 0.5$
AdaBoost

The weights of missclassified instances are increased

\[ t \leftarrow 1; \quad w_i^{(t)} \leftarrow 1/n, \quad i = 1, \ldots, n \]

repeat

Train model \( M^{(t)} \) on \( D \) weighted by \( w^{(t)} \)

\( \epsilon_t \leftarrow \) corresponding training error rate

\( \alpha_t \leftarrow \ln \left( \frac{1 - \epsilon_t}{\epsilon_t} \right) / 2 \)

\[ w_i^{(t+1)} \leftarrow \begin{cases} 
  w_i^{(t)} e^{-\alpha_t} & \text{if instance } i \text{ is correctly classified} \\
  w_i^{(t)} e^{\alpha_t} & \text{otherwise} 
\end{cases} \]

\[ t \leftarrow t + 1; \quad w^{(t)} \leftarrow w^{(t)}/\sum_i w_i^{(t)} \]

until \( t > T \) or \( \epsilon_{t-1} = 0 \) or \( \epsilon_{t-1} \geq 0.5 \)
AdaBoost

The algorithm stops if perfect accuracy is achieved ($\epsilon_t = 0$) or accuracy is worse than random guessing ($\epsilon_t = 0.5$) or maximum number of iterations $T$ has been reached.

$t \leftarrow 1; \quad w_i^{(t)} \leftarrow 1/n, \ i = 1, \ldots, n$

repeat

Train model $M^{(t)}$ on $D$ weighted by $w^{(t)}$

$\epsilon_t \leftarrow$ corresponding training error rate

$\alpha_t \leftarrow \ln((1 - \epsilon_t)/\epsilon_t)/2$

$w_i^{(t+1)} \leftarrow \begin{cases} w_i^{(t)} e^{-\alpha_t} & \text{if instance } i \text{ is correctly classified} \\ w_i^{(t)} e^{\alpha_t} & \text{otherwise} \end{cases}$

$t \leftarrow t + 1; \quad w^{(t)} \leftarrow w^{(t)} / \sum_i w_i^{(t)}$

until $t > T$ or $\epsilon_{t-1} = 0$ or $\epsilon_{t-1} \geq 0.5$
The label for given test instance $x$ is predicted according to

$$\text{sign} \left( \sum_t \alpha_t f_{M_t}(x) \right)$$

i.e. aggregates the weighted predictions of all the models.

In some versions of the algorithm, weights are reset to $1/n$ whenever $\epsilon_t \geq 0.5$.

In other versions, $\epsilon_t$ is allowed to increase beyond 0.5 but the predictions of the corresponding models are effectively inverted by applying negative weights.
**Boosting** primarily focuses on **reducing bias**
It aims to combine many weak learners into a strong learner
The approach should be used with simple models having high bias but low variance

When re-weighting is done via sampling, it can also help reduce variance

The approach is vulnerable to noise
It assumes that error is caused by bias, in the presence of noise it will overtrain on low-quality portions of the data

Typically superior to bagging when noise is not excessive
The training dataset is divided into two subsets $\mathcal{D}_A$ and $\mathcal{D}_B$. $\mathcal{D}_A$ is used to train $\ell$ models, the ensemble components $\mathcal{D}_B$ is used to train a second-level classifier that combines the predictions of the ensemble components.
Stacking

$\mathcal{D}_B$ is mapped to a $\ell$-dimensional space where each dimension represents the predictions of one ensemble component.

<table>
<thead>
<tr>
<th>Original feature space</th>
<th>Transformed feature space</th>
</tr>
</thead>
<tbody>
<tr>
<td>Training data</td>
<td></td>
</tr>
<tr>
<td>$\mathcal{D}_B, n \times m$ matrix</td>
<td>$\mathcal{D}_B', n \times \ell$ matrix</td>
</tr>
<tr>
<td>Training instance</td>
<td></td>
</tr>
<tr>
<td>$(x_i, y_i)$</td>
<td>$(\langle f_{M_1}(x_i), \ldots, f_{M_\ell}(x_i) \rangle, y_i)$</td>
</tr>
</tbody>
</table>

A second-level classifier is trained on the transformed training data $\mathcal{D}_B'$, learning to predict class labels from the predictions of the ensemble components.
The ensemble components can be obtained in various ways, e.g. using \( \ell \) bootstrapped samples \( \mathcal{D}_A \) (bagging), \( \ell \) rounds of boosting on \( \mathcal{D}_A \), a bucket of \( \ell \) models trained on \( \mathcal{D}_A \), etc.

Class probabilities can be used as features instead of the predictions from the ensemble components. Original attributes are often retained in the transformed data.
Stacking can be combined with $m$-fold cross-validation

A new representation is obtained for each instance of the training data, where the features are obtained from the $\ell$ first-level classifiers trained on the $(m - 1)$ folds that do not contain that instance.

The second-level classifier is trained on this dataset representing \textit{all} training instances.

The first-level classifiers are re-trained on the full training data.
By learning from the errors of the ensemble components stacking allows to reduce both bias and variance.

The power of stacking comes from the flexible learning approach of the combiner.

Many other ensemble methods can be seen as special cases using less flexible, data-independent, combination procedures such as voting.