Machine Learning and Data Mining

Introduction

Albert Bifet(@abifet)

Data Science

Data Science is an interdisciplinary field focused on extracting knowledge or insights from large volumes of data.

Data Scientist

DATA SCIENTIST
Math
Statistics
Programming
Database
Domain Knowledge
Soft Skills
Communication
Visualization

Figure: `http://www.marketingdistillery.com/2014/11/29/is-data-science-a-buzzword-modern-data-scientist-defined/`

Data Science

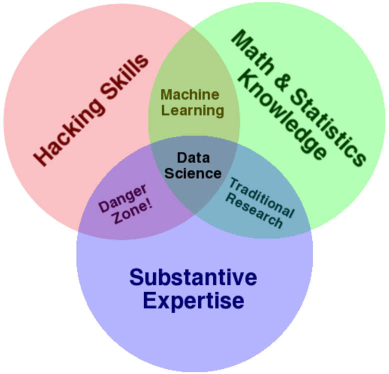


Figure: Drew Convay's Venn diagram

Classification

Definition

Given n_C different classes, a classifier algorithm builds a model that predicts for every unlabelled instance I the class C to which it belongs with accuracy.

Example

A spam filter

Example

Twitter Sentiment analysis: analyze tweets with positive or negative feelings

Classification

Data set that describes e-mail features for deciding if it is spam.

Example

Contains "Money"	Domain type	Has attach.	Time received	spam
yes	com	yes	night	yes
yes	edu	no	night	yes
no	com	yes	night	yes
no	edu	no	day	no
no	com	no	day	no
yes	cat	no	day	yes

Assume we have to classify the following new instance:

Contains "Money"	Domain type	Has attach.	Time received	spam
yes	edu	yes	day	?

k-Nearest Neighbours

k-NN Classifier

- Training: store all instances in memory
- Prediction:
    - Find the *k* nearest instances
    - Output majority class of these *k* instances

Bayes Classifiers

Naïve Bayes

- Based on Bayes Theorem:

$$P(c|d) = \frac{P(c)P(d|c)}{P(d)}$$

$$posterior = \frac{prior \times likelikood}{evidence}$$

- Estimates the probability of observing attribute *a* and the prior probability $P(c)$
- Probability of class *c* given an instance *d*:

$$P(c|d) = \frac{P(c)\prod_{a \in d} P(a|c)}{P(d)}$$

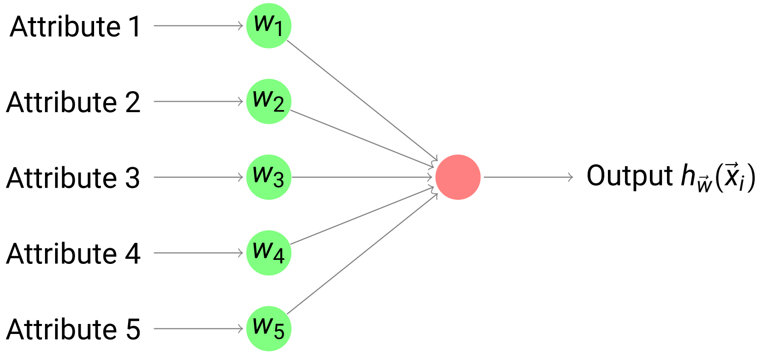
Bayes Classifiers

Multinomial Naïve Bayes

- Considers a document as a bag-of-words.
- Estimates the probability of observing word w and the prior probability $P(c)$
- Probability of class c given a test document d:

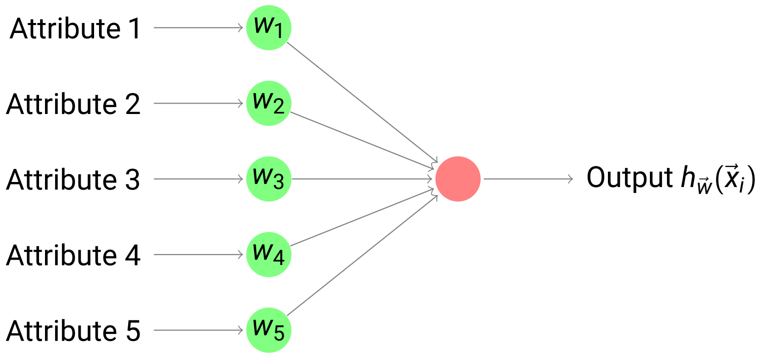
$$P(c|d) = \frac{P(c)\prod_{w \in d} P(w|c)^{n_{wd}}}{P(d)}$$

Perceptron



- Data stream: $\langle \vec{x}_i, y_i \rangle$
- Classical perceptron: $h_{\vec{w}}(\vec{x}_i) = \mathrm{sgn}(\vec{w}^T \vec{x}_i)$,
- Minimize Mean-square error: $J(\vec{w}) = \frac{1}{2} \sum (y_i - h_{\vec{w}}(\vec{x}_i))^2$

Perceptron



- We use sigmoid function $h_{\vec{w}} = \sigma(\vec{w}^T\vec{x})$ where

$$\sigma(x) = 1/(1+e^{-x})$$

$$\sigma'(x) = \sigma(x)(1-\sigma(x))$$

Perceptron

- Minimize Mean-square error: $J(\vec{w}) = \frac{1}{2}\sum(y_i - h_{\vec{w}}(\vec{x}_i))^2$
- Stochastic Gradient Descent: $\vec{w} = \vec{w} - \eta \nabla J \vec{x}_i$
- Gradient of the error function:

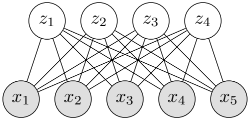
$$\nabla J = -\sum_i (y_i - h_{\vec{w}}(\vec{x}_i)) \nabla h_{\vec{w}}(\vec{x}_i)$$

$$\nabla h_{\vec{w}}(\vec{x}_i) = h_{\vec{w}}(\vec{x}_i)(1 - h_{\vec{w}}(\vec{x}_i))$$

- Weight update rule

$$\vec{w} = \vec{w} + \eta \sum_i (y_i - h_{\vec{w}}(\vec{x}_i)) h_{\vec{w}}(\vec{x}_i)(1 - h_{\vec{w}}(\vec{x}_i))\vec{x}_i$$

Restricted Boltzmann Machines (RBMs)



- Energy-based models, where

$$P(\vec{x},\vec{z}) \propto e^{-E(\vec{x},\vec{z})}.$$

- Manipulate a weight matrix W to find low-energy states and thus generate high probability $P(\vec{x},\vec{z})$, where

$$E(\vec{x},\vec{z}) = -W.$$

- RBMs can be stacked on top of each other to form so-called Deep Belief Networks (DBNs)

Classification

Example

Data set that describes e-mail features for deciding if it is spam.

Contains "Money"	**Domain type**	**Has attach.**	**Time received**	**spam**
yes	com	yes	night	yes
yes	edu	no	night	yes
no	com	yes	night	yes
no	edu	no	day	no
no	com	no	day	no
yes	cat	no	day	yes

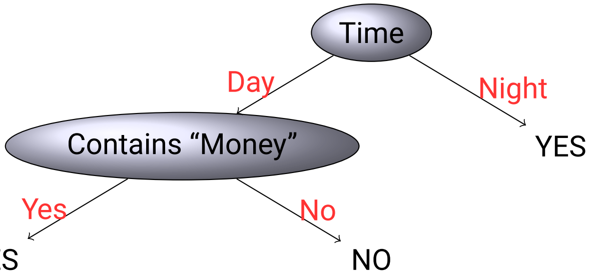
Assume we have to classify the following new instance:

Contains "Money"	**Domain type**	**Has attach.**	**Time received**	**spam**
yes	edu	yes	day	?

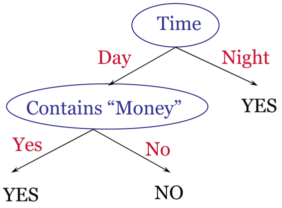
Classification

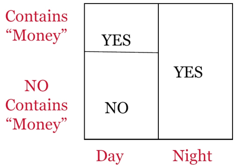
- Assume we have to classify the following new instance:

Contains “Money”	Domain type	Has attach.	Time received	spam
yes	edu	yes	day	?



Decision Trees





Basic induction strategy:

- $A \leftarrow$ the "best" decision attribute for next *node*
- Assign A as decision attribute for *node*
- For each value of A, create new descendant of *node*
- Sort training examples to leaf nodes
- If training examples perfectly classified, Then STOP, Else iterate over new leaf nodes

Bagging

Example

Dataset of 4 Instances : A, B, C, D

Classifier 1: B, A, C, B
Classifier 2: D, B, A, D
Classifier 3: B, A, C, B
Classifier 4: B, C, B, B
Classifier 5: D, C, A, C

Bagging builds a set of M base models, with a bootstrap sample created by drawing random samples with replacement.

Random Forests

- Bagging
- Random Trees: trees that in each node only uses a random subset of the attributes

Random Forests is one of the most popular methods in machine learning.

Boosting

The strength of Weak Learnability, Schapire 90

A boosting algorithm transforms a weak learner into a strong one

Boosting

A formal description of Boosting (Schapire)

- given a training set $(x_1, y_1), \ldots, (x_m, y_m)$
- $y_i \in \{-1, +1\}$ correct label of instance $x_i \in X$
- for $t = 1, \ldots, T$
  - construct distribution D_t
  - find weak classifier

$$h_t : X \to \{-1, +1\}$$

    with small error $\varepsilon_t = \Pr_{D_t}[h_t(x_i) \neq y_i]$ on D_t
- output final classifier

Boosting

AdaBoost

```
1: Initialize $D_1(i) = 1/m$ for all $i \in \{1, 2, ..., m\}$
2: **for** $t$ = 1,2,...$T$ **do**
3:     Call **WeakLearn**, providing it with distribution $D_t$
4:     Get back hypothesis $h_t : X \rightarrow Y$
5:     Calculate error of $h_t : \varepsilon_t = \sum_{i:h_t(x_i) \neq y_i} D_t(i)$
6:     Update distribution
       $D_t : D_{t+1}(i) = \frac{D_t(i)}{Z_t} \times \begin{cases} \varepsilon_t/(1-\varepsilon_t) & \text{if } h_t(x_i) = y_i \\ 1 & \text{otherwise} \end{cases}$
       where $Z_t$ is a normalization constant (chosen so $D_{t+1}$ is a
       probability distribution)
7: **return** $h_{fin}(x) = \arg\max_{y \in Y} \sum_{t:h_t(x)=y} -\log \varepsilon_t/(1-\varepsilon_t)$
```

Boosting

AdaBoost

```
1: Initialize $D_1(i) = 1/m$ for all $i \in \{1, 2, ..., m\}$
2: **for** $t$ = 1,2,...$T$ **do**
3:     Call **WeakLearn**, providing it with distribution $D_t$
4:     Get back hypothesis $h_t : X \to Y$
5:     Calculate error of $h_t : \varepsilon_t = \sum_{i:h_t(x_i) \neq y_i} D_t(i)$
6:     Update distribution
       $D_t : D_{t+1}(i) = \frac{D_t(i)}{Z_t} \times \begin{cases} \varepsilon_t & \text{if } h_t(x_i) = y_i \\ 1 - \varepsilon_t & \text{otherwise} \end{cases}$
       where $Z_t$ is a normalization constant (chosen so $D_{t+1}$ is a
       probability distribution)
7: **return** $h_{fin}(x) = \arg\max_{y \in Y} \sum_{t:h_t(x)=y} - \log \varepsilon_t / (1 - \varepsilon_t)$
```

Stacking

Use a classifier to combine predictions of base classifiers

Example

- Use a perceptron to do stacking
- Use decision trees as base classifiers

Clustering

Definition
Clustering is the distribution of a set of instances of examples into non-known groups according to some common relations or affinities.

Example
Market segmentation of customers

Example
Social network communities

Clustering

Definition

Given

- a set of instances I
- a number of clusters K
- an objective function $cost(I)$

a clustering algorithm computes an assignment of a cluster for each instance

$$f : I \to \{1, \dots, K\}$$

that minimizes the objective function $cost(I)$

Clustering

Definition

Given

- a set of instances I
- a number of clusters K
- an objective function $cost(C, I)$

a clustering algorithm computes a set C of instances with $|C| = K$ that minimizes the objective function

$$cost(C, I) = \sum_{x \in I} d^2(x, C)$$

where

- $d(x, c)$: distance function between x and c
- $d^2(x, C) = min_{c \in C} d^2(x, c)$: distance from x to the nearest point in C

k-means

- 1. Choose k initial centers $C = \{c_1, \dots, c_k\}$
- 2. while stopping criterion has not been met
    - For $i = 1, \dots, N$
        - find closest center $c_k \in C$ to each instance p_i
        - assign instance p_i to cluster C_k
    - For $k = 1, \dots, K$
        - set c_k to be the center of mass of all points in C_i

k-means++

- 1. Choose a initial center c_1
- For $k = 2, \ldots, K$
    - select $c_k = p \in I$ with probability $d^2(p, C)/cost(C, I)$
- 2. while stopping criterion has not been met
    - For $i = 1, \ldots, N$
        - find closest center $c_k \in C$ to each instance p_i
        - assign instance p_i to cluster C_k
    - For $k = 1, \ldots, K$
        - set c_k to be the center of mass of all points in C_i

Performance Measures

Internal Measures

- Sum square distance
- Dunn index $D = \frac{d_{min}}{d_{max}}$
- C-Index $C = \frac{S - S_{min}}{S_{max} - S_{min}}$

External Measures

- Rand Measure
- F Measure
- Jaccard
- Purity

Density based methods

DBSCAN

- ε-neighborhood(p): set of points that are at a distance of p less or equal to ε
- Core object: object whose ε-neighborhood has an overall weight at least μ
- A point p is *directly density-reachable* from q if
  - p is in ε-neighborhood(q)
  - q is a core object
- A point p is *density-reachable* from q if
  - there is a chain of points $p_1, \ldots, p_n$ such that p_{i+1} is directly density-reachable from p_i
- A point p is *density-connected* from q if
  - there is point o such that p and q are density-reachable from o

Density based methods

DBSCAN

- A *cluster* C of points satisfies
  - if $p \in C$ and q is density-reachable from p, then $q \in C$
  - all points $p, q \in C$ are density-connected
- A *cluster* is uniquely determined by any of its core points
- A *cluster* can be obtained
  - choosing an arbitrary core point as a seed
  - retrieve all points that are density-reachable from the seed

DBSCAN

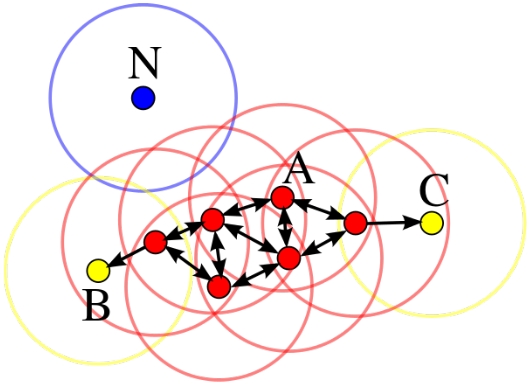


Figure: DBSCAN Point Example with μ=3

Density based methods

DBSCAN

- select an arbitrary point *p*
- retrieve all points density-reachable from *p*
- if *p* is a core point, a cluster is formed
- If p is a border point
  - no points are density-reachable from p
  - DBSCAN visits the next point of the database
- Continue the process until all of the points have been processed

Frequent Patterns

Suppose $\mathscr{D}$ is a dataset of patterns, $t \in \mathscr{D}$, and *min_sup* is a constant.

Definition
Support (t): number of patterns in $\mathscr{D}$ that are superpatterns of t.

Definition
Pattern t is *frequent* if *Support* $(t) \geq$ *min_sup*.

Frequent Subpattern Problem
Given $\mathscr{D}$ and *min_sup*, find all frequent subpatterns of patterns in $\mathscr{D}$.

Frequent Patterns

Suppose $\mathscr{D}$ is a dataset of patterns, $t \in \mathscr{D}$, and *min_sup* is a constant.

Definition
Support (t): number of patterns in $\mathscr{D}$ that are superpatterns of t.

Definition
Pattern t is *frequent* if *Support* (t) $\geq$ *min_sup*.

Frequent Subpattern Problem
Given $\mathscr{D}$ and *min_sup*, find all frequent subpatterns of patterns in $\mathscr{D}$.

Frequent Patterns

Suppose $\mathscr{D}$ is a dataset of patterns, $t \in \mathscr{D}$, and *min_sup* is a constant.

Definition
Support (t): number of patterns in $\mathscr{D}$ that are superpatterns of t.

Definition
Pattern t is *frequent* if *Support* $(t) \geq$ *min_sup*.

Frequent Subpattern Problem
Given $\mathscr{D}$ and *min_sup*, find all frequent subpatterns of patterns in $\mathscr{D}$.

Frequent Patterns

Suppose $\mathscr{D}$ is a dataset of patterns, $t \in \mathscr{D}$, and min_sup is a constant.

Definition

Support (t): number of patterns in $\mathscr{D}$ that are superpatterns of t.

Definition

Pattern t is *frequent* if $Support\ (t) \geq min\_sup$.

Frequent Subpattern Problem

Given $\mathscr{D}$ and min_sup, find all frequent subpatterns of patterns in $\mathscr{D}$.

Pattern Mining

Dataset Example

Document	Patterns
d1	abce
d2	cde
d3	abce
d4	acde
d5	abcde
d6	bcd

Itemset Mining

d1	abce
d2	cde
d3	abce
d4	acde
d5	abcde
d6	bcd

Support	**Frequent**
d1,d2,d3,d4,d5,d6	c
d1,d2,d3,d4,d5	e,ce
d1,d3,d4,d5	a,ac,ae,ace
d1,d3,d5,d6	b,bc
d2,d4,d5,d6	d,cd
d1,d3,d5	ab,abc,abe be,bce,abce
d2,d4,d5	de,cde

minimal support = 3

Itemset Mining

d1	abce
d2	cde
d3	abce
d4	acde
d5	abcde
d6	bcd

Support	Frequent
6	c
5	e,ce
4	a,ac,ae,ace
4	b,bc
4	d,cd
3	ab,abc,abe be,bce,abce
3	de,cde

Itemset Mining

d1	abce
d2	cde
d3	abce
d4	acde
d5	abcde
d6	bcd

Support	**Frequent**	**Gen**	**Closed**
6	c	c	c
5	e,ce	e	ce
4	a,ac,ae,ace	a	ace
4	b,bc	b	bc
4	d,cd	d	cd
3	ab,abc,abe	ab	
	be,bce,abce	be	abce
3	de,cde	de	cde

Itemset Mining

d1	abce
d2	cde
d3	abce
d4	acde
d5	abcde
d6	bcd

Support	Frequent	Gen	Closed	Max
6	c	c	c	
5	e,ce	e	ce	
4	a,ac,ae,ace	a	ace	
4	b,bc	b	bc	
4	d,cd	d	cd	
3	ab,abc,abe	ab		
	be,bce,abce	be	abce	abce
3	de,cde	de	cde	cde

Itemset Mining

d1	abce
d2	cde
d3	abce
d4	acde
d5	abcde
d6	bcd

Support	**Frequent**	**Gen**	**Closed**	**Max**
6	c	c	c	
5	e,ce	e	ce	
4	a,ac,ae,ace	a	ace	
4	b,bc	b	bc	
4	d,cd	d	cd	
3	ab,abc,abe be,bce,abce	ab be	abce	abce
3	de,cde	de	cde	cde

Itemset Mining

d1	abce
d2	cde
d3	abce
d4	acde
d5	abcde
d6	bcd

$e \longrightarrow ce$

Support	Frequent	Gen	Closed	Max
6	c	c	c	
5	e,ce	e	ce	
4	a,ac,ae,ace	a	ace	
4	b,bc	b	bc	
4	d,cd	d	cd	
3	ab,abc,abe	ab		
	be,bce,abce	be	abce	abce
3	de,cde	de	cde	cde

Itemset Mining

d1	abce
d2	cde
d3	abce
d4	acde
d5	abcde
d6	bcd

Support	**Frequent**	**Gen**	**Closed**	**Max**
6	c	c	c	
5	e,ce	e	ce	
4	a,ac,ae,ace	a	ace	
4	b,bc	b	bc	
4	d,cd	d	cd	
3	ab,abc,abe be,bce,abce	ab be	abce	abce
3	de,cde	de	cde	cde

Itemset Mining

d1	abce
d2	cde
d3	abce
d4	acde
d5	abcde
d6	bcd

Support	Frequent	Gen	Closed	Max
6	c	c	c	
5	e,ce	e	ce	
4	a,ac,ae,ace	a	ace	
4	b,bc	b	bc	
4	d,cd	d	cd	
3	ab,abc,abe	ab		
	be,bce,abce	be	abce	abce
3	de,cde	de	cde	cde

Itemset Mining

d1	abce
d2	cde
d3	abce
d4	acde
d5	abcde
d6	bcd

$a \rightarrow ace$

Support	Frequent	Gen	Closed	Max
6	c	c	c	
5	e,ce	e	ce	
4	a,ac,ae,ace	a	ace	
4	b,bc	b	bc	
4	d,cd	d	cd	
3	ab,abc,abe	ab		
	be,bce,abce	be	abce	abce
3	de,cde	de	cde	cde

Itemset Mining

d1	abce
d2	cde
d3	abce
d4	acde
d5	abcde
d6	bcd

Support	**Frequent**	**Gen**	**Closed**	**Max**
6	c	c	c	
5	e,ce	e	ce	
4	a,ac,ae,ace	a	ace	
4	b,bc	b	bc	
4	d,cd	d	cd	
3	ab,abc,abe	ab		
	be,bce,abce	be	abce	abce
3	de,cde	de	cde	cde

Closed Patterns

Usually, there are too many frequent patterns. We can compute a smaller set, while keeping the same information.

Example

A set of 1000 items, has $2^{1000} \approx 10^{301}$ subsets, that is more than the number of atoms in the universe $\approx 10^{79}$

Closed Patterns

A priori property

If t' is a subpattern of t, then $Support\ (t') \geq Support\ (t)$.

Definition

A frequent pattern t is *closed* if none of its proper superpatterns has the same support as it has.

Frequent subpatterns and their supports can be generated from closed patterns.

Maximal Patterns

Definition
A frequent pattern *t* is *maximal* if none of its proper superpatterns is frequent.

Frequent subpatterns can be generated from maximal patterns, but not with their support.

All maximal patterns are closed, but not all closed patterns are maximal.

Non streaming frequent itemset miners

Representation:

- Horizontal layout

T1: a, b, c
T2: b, c, e
T3: b, d, e

- Vertical layout

a: 1 0 0
b: 1 1 1
c: 1 1 0

Search:

- Breadth-first (levelwise): Apriori
- Depth-first: Eclat, FP-Growth

The Apriori Algorithm

```
APRIORI ALGORITHM

1  Initialize the item set size $k = 1$
2  Start with single element sets
3  Prune the non-frequent ones
4  **while** there are frequent item sets
5      **do** create candidates with one item more
6         Prune the non-frequent ones
7         Increment the item set size $k = k + 1$

8  Output: the frequent item sets
```

The Eclat Algorithm

Depth-First Search

- divide-and-conquer scheme : the problem is processed by splitting it into smaller subproblems, which are then processed recursively
  - **conditional database for the prefix a**
    - transactions that contain a
  - **conditional database for item sets without a**
    - transactions that not contain a
- Vertical representation
- Support counting is done by intersecting lists of transaction identifiers

The FP-Growth Algorithm

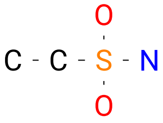
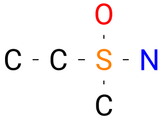
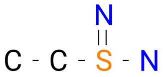
Depth-First Search

- divide-and-conquer scheme : the problem is processed by splitting it into smaller subproblems, which are then processed recursively
  - **conditional database for the prefix a**
    - transactions that contain a
  - **conditional database for item sets without a**
    - transactions that not contain a
- Vertical and Horizontal representation : FP-Tree
  - prefix tree with links between nodes that correspond to the same item
- Support counting is done using FP-Tree

Mining Graph Data

Problem

Given a data set $\mathscr{D}$ of graphs, find frequent graphs.

Transaction Id	Graph
1	
2	
3	

The gSpan Algorithm

```
GSPAN($g, D, min\_sup, S$)

    Input: A graph $g$, a graph dataset $D$, $min\_sup$.
    Output: The frequent graph set $S$.

 1  if $g \neq min(g)$
 2      then return $S$
 3  insert $g$ into $S$
 4  update support counter structure
 5  $C \leftarrow \emptyset$
 6  for each $g'$ that can be right-most
          extended from $g$ in one step
 7        do if support($g$) $\geq min\_sup$
 8              then insert $g'$ into $C$
 9  for each $g'$ in $C$
10        do $S \leftarrow$ GSPAN($g', D, min\_sup, S$)
11  return $S$
```