A Gentle Introduction to Machine Learning in Natural Language Processing using R

ESSLLI '2013
Düsseldorf, Germany

http://ufal.mff.cuni.cz/mlnlpr13

Barbora Hladká
hladka@ufal.mff.cuni.cz

Martin Holub
holub@ufal.mff.cuni.cz

Charles University in Prague,
Faculty of Mathematics and Physics,
Institute of Formal and Applied Linguistics

Day 4

- 4.1 Information Theory and Feature Selection
- 4.2 SVM learning – Theory
- 4.3 SVM learning – Practice

Block 4.1
Information Theory and Feature Selection

In machine learning and statistics, **feature selection** is the process of selecting a subset of relevant features for use in model construction.

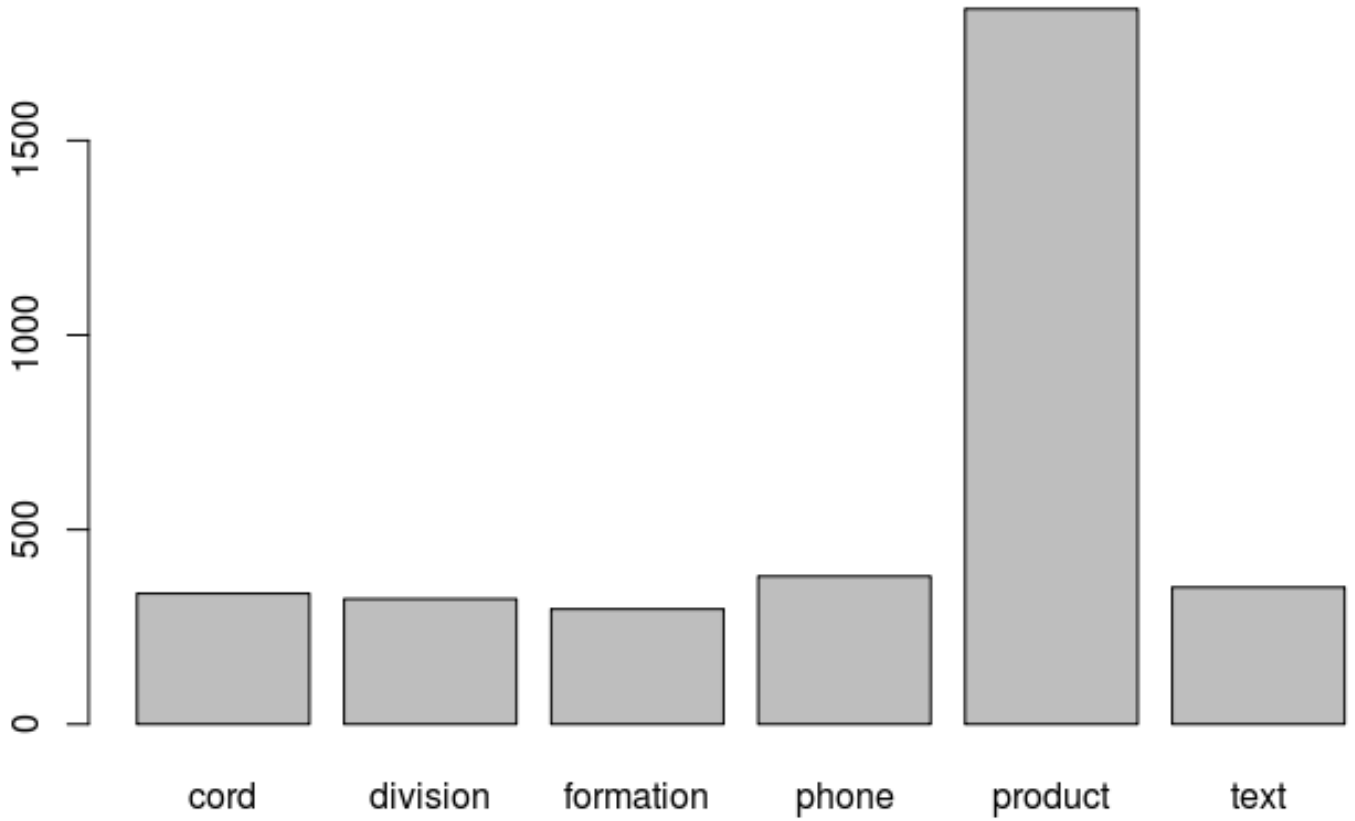
— *by Wikipedia*

Why do we need feature selection?

- to get best performance
- to keep the model compact
- to remove redundant or irrelevant features

Distribution of target class values

```
> plot(examples$SENSE)
>
```



Amount of information contained in a value?

How much information do you gain when you observe a random event?

According to the **Information Theory**, **amount of information** contained in an event is given by

$$I = \log_2 \frac{1}{p} = -\log_2 p$$

where p is probability of the event occurred.

Thus, the lower probability, the more information you get when you observe an event (e.g. a feature value). If an event is certain ($p = 100\,\%$), then the amount of information is zero.

Amount of information in SENSE values

```
### probability distribution of SENSE
> round(table(examples$SENSE)/nrow(examples), 3)

     cord  division formation     phone   product       text
    0.095     0.091     0.084     0.108     0.522     0.100
>

### amount of information contained in SENSE values
> round(-log2(table(examples$SENSE)/nrow(examples)), 3)

     cord  division formation     phone   product       text
    3.391     3.452     3.574     3.213     0.939     3.324
>
```

What is the average amount of information that you get when you observe values of the attribute SENSE?

Entropy

The average amount of information that you get when you observe random values is

$$\sum_{value} \Pr(value) \cdot \log_2 \frac{1}{\Pr(value)} \quad = \quad -\sum_{value} \Pr(value) \cdot \log_2 \Pr(value)$$

This is what information theory calls entropy.

- Entropy of a random variable X is denoted by $\mathrm{H}(X)$.
- Entropy is a measure of the uncertainty in a random variable.

The unit of entropy is bit. Entropy says how many bits on average you necessarily need to encode a value of the given random variable.

Entropy of SENSE

Entropy of SENSE is 2.107129 **bits.**

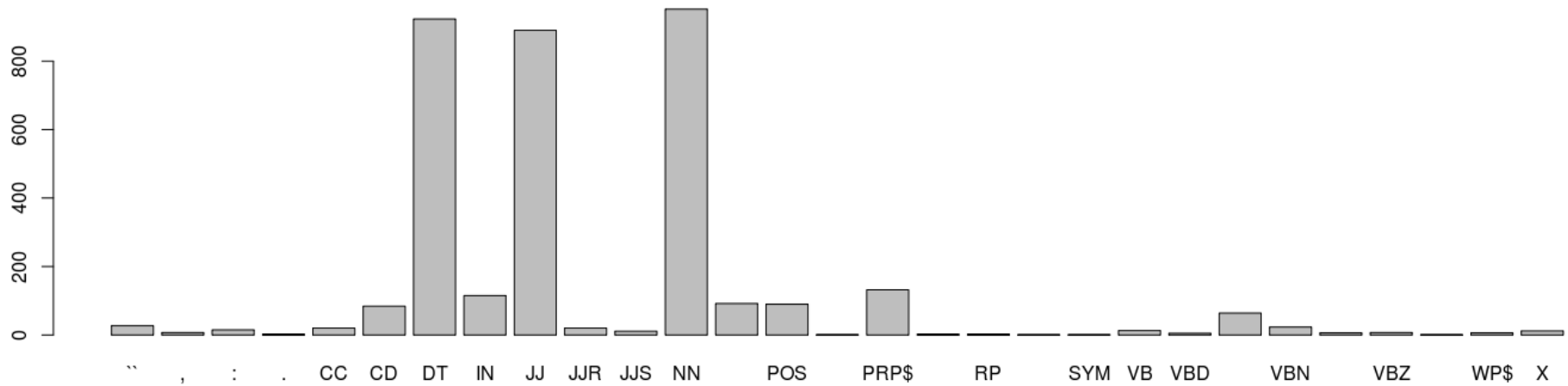
```
### probability distribution of SENSE
> p.sense <- table(examples$SENSE)/nrow(examples)
>
### entropy of SENSE
> H.sense <-  - sum( p.sense * log2(p.sense) )
> H.sense
[1] 2.107129
```

The maximum entropy value would be $\log_2(6) = 2.584963$
if and only if the distribution of the 6 senses was uniform.

```
> p.uniform <- rep(1/6, 6)
> p.uniform
[1] 0.1666667 0.1666667 0.1666667 0.1666667 0.1666667 0.1666667
>
### entropy of uniformly distributed 6 senses
> - sum( p.uniform * log2(p.uniform) )
[1] 2.584963
```

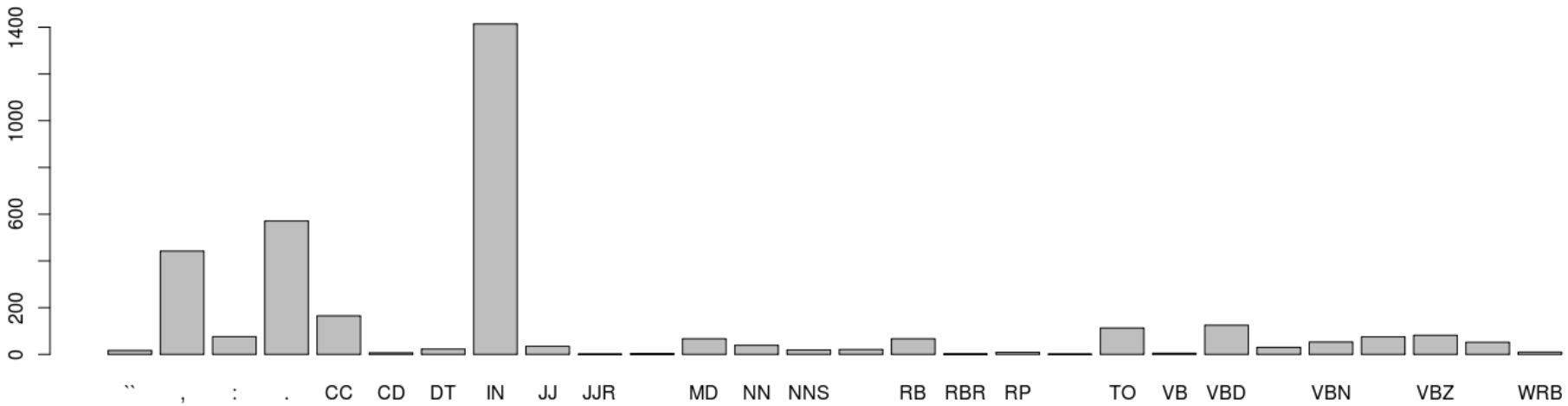
Distribution of feature values – A16

```
> levels(examples$A16)
 [1] "``"    ","     ":"     "."     "CC"    "CD"    "DT"    "IN"    "J"
[10] "JJR"   "JJS"   "NN"    "NNS"   "POS"   "PRP"   "PRP$"  "RB"    "R"
[19] "-RRB-" "SYM"   "VB"    "VBD"   "VBG"   "VBN"   "VBP"   "VBZ"   "W"
[28] "WP$"   "X"
> plot(examples$A16)
>
```



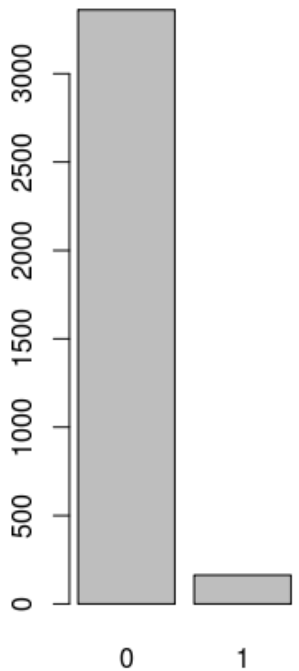
Distribution of feature values – A17

```
> levels(examples$A17)
 [1] "``"    ","     ":"     "."     "CC"    "CD"    "DT"    "IN"    "J"
[10] "JJR"   "-LRB-" "MD"    "NN"    "NNS"   "PRP"   "RB"    "RBR"   "R"
[19] "-RRB-" "TO"    "VB"    "VBD"   "VBG"   "VBN"   "VBP"   "VBZ"   "W"T"
[28] "WRB"
> plot(examples$A17)
>
```



Distribution of feature values – A4

```
> levels(examples$A4)
[1] "0" "1"
>
```



Entropy of features

Entropy of A16 is 2.78 **bits.**

```
> p.A16 <- table(examples$A16)/nrow(examples)
> H.A16 <-  - sum( p.A16 * log2(p.A16) )
> H.A16
[1] 2.777606
```

Entropy of A17 is 3.09 **bits.**

```
> p.A17 <- table(examples$A17)/nrow(examples)
> H.A17 <-  - sum( p.A17 * log2(p.A17) )
> H.A17
[1] 3.093003
```

Entropy of A4 is 0.27 **bits.**

```
> p.A4 <- table(examples$A4)/nrow(examples)
> H.A4 <-  - sum( p.A4 * log2(p.A4) )
> H.A4
[1] 0.270267
```

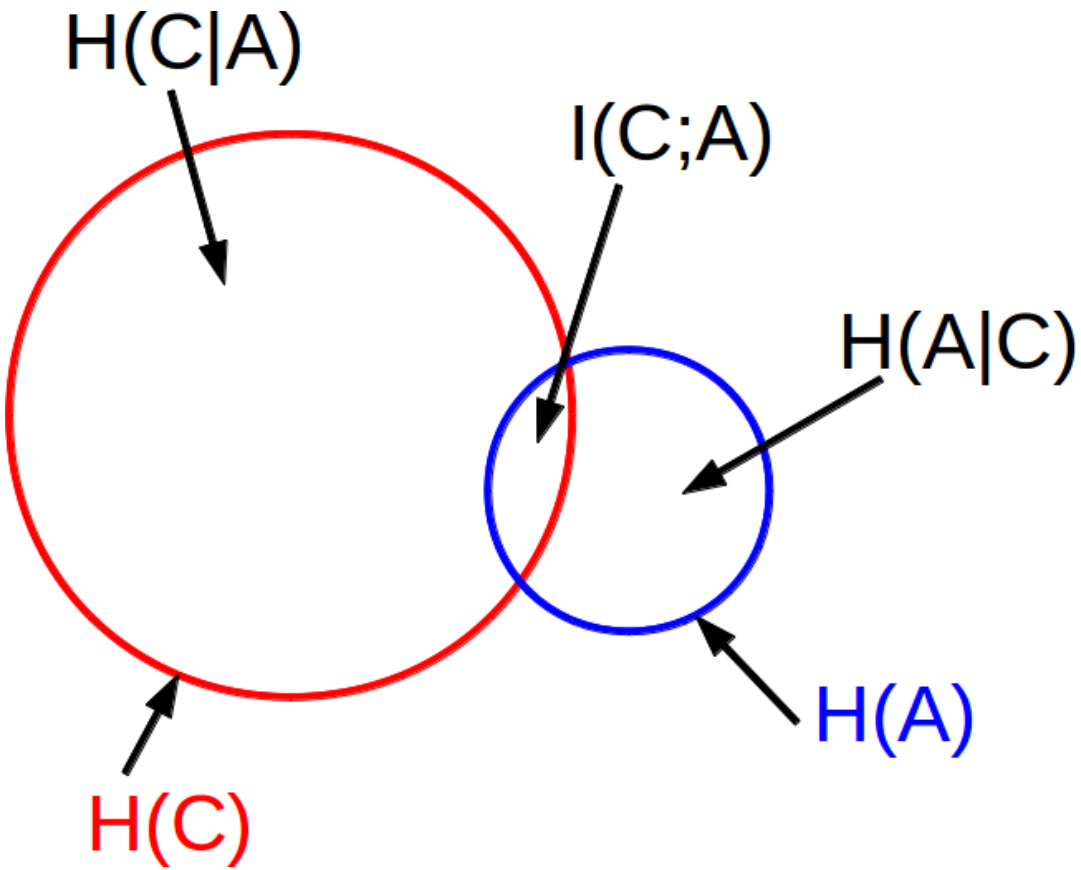
Conditional entropy H(C | A)

How much does target class entropy decrease if we have the knowledge of a feature?

The answer is conditional entropy:

$$\mathrm{H}(C \mid A) \quad = \quad - \sum_{y \in C, x \in A} \Pr(y, x) \cdot \log_2 \Pr(y \mid x)$$

Conditional entropy and mutual information



WARNING

There are NO SETS in this picture! Entropy is a quantity, only a number!

Conditional entropy and mutual information

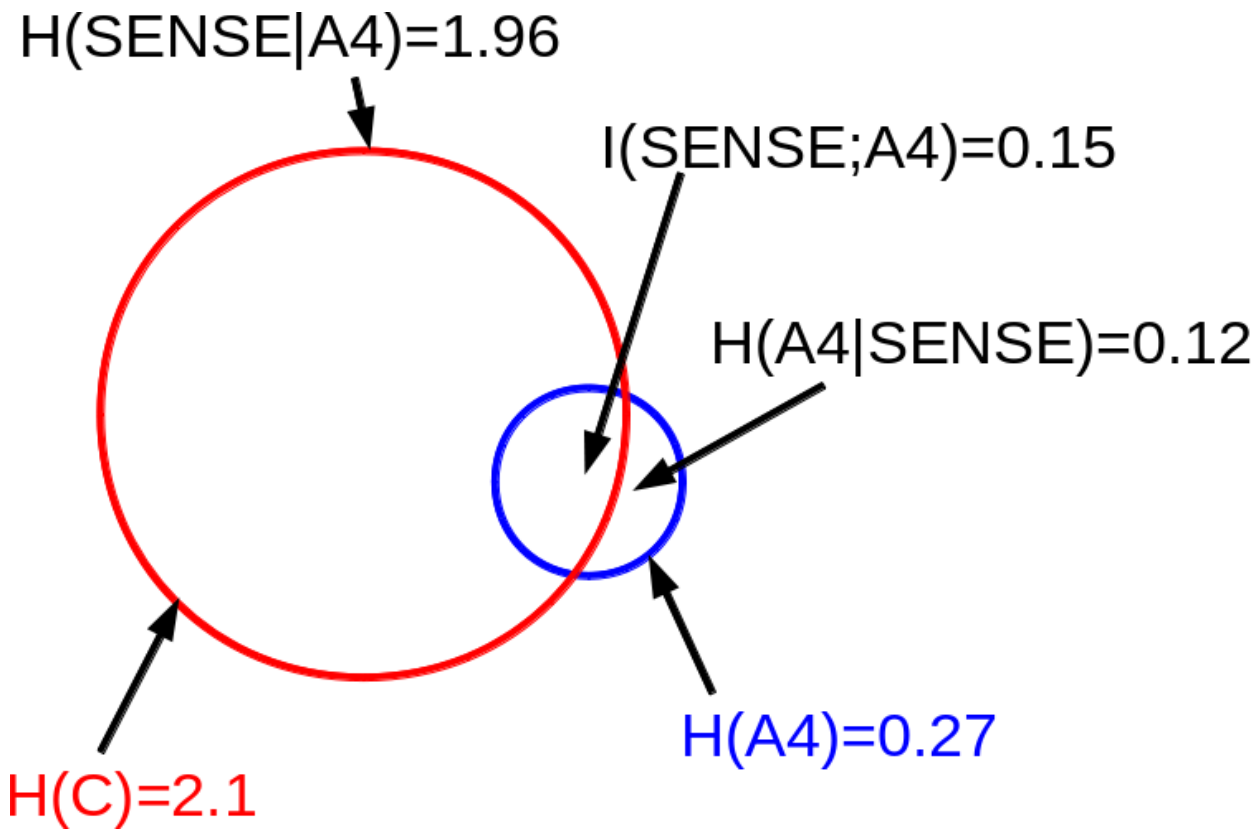
Mutual information measures the amount of information that can be obtained about one random variable by observing another.

Mutual information is a symmetrical quantity.

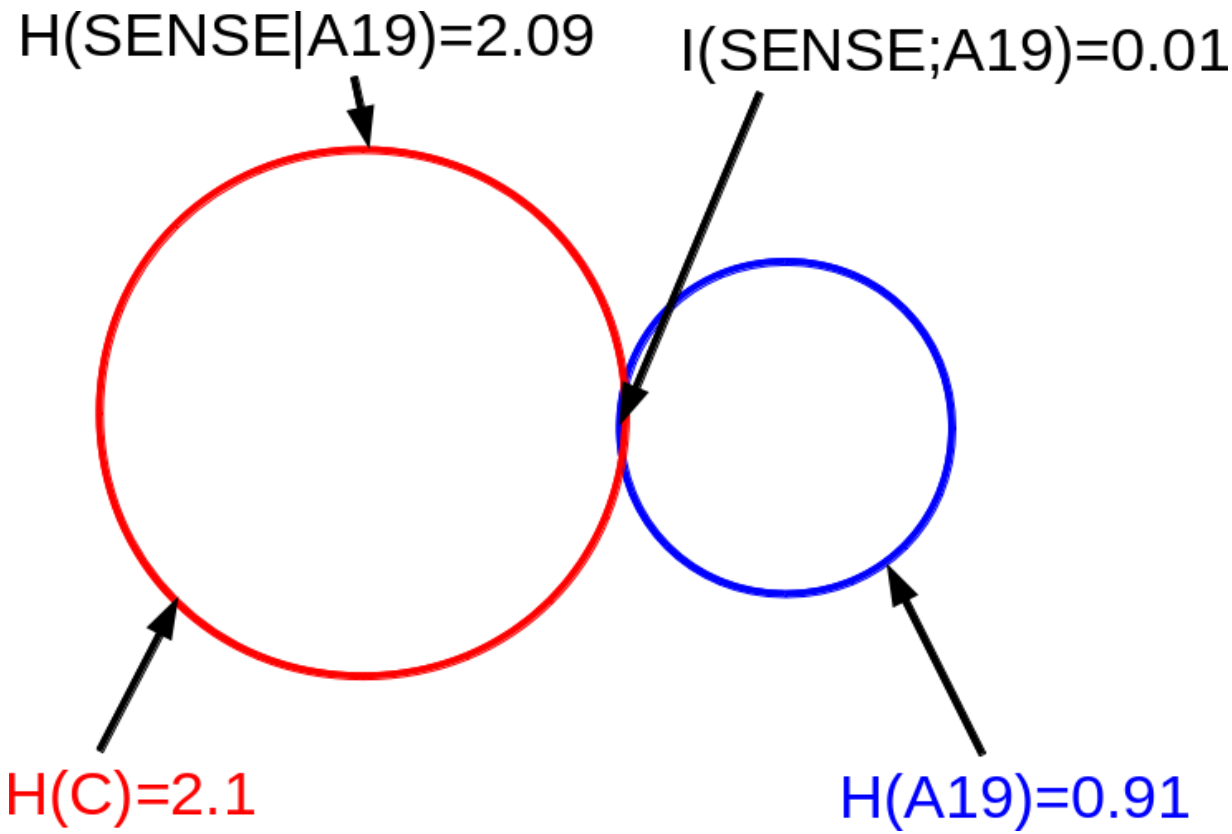
$$\mathrm{H}(C) - \mathrm{H}(C \mid A) \quad = \quad \mathrm{I}(C; A) \quad = \quad \mathrm{H}(A) - \mathrm{H}(A \mid C)$$

Another name for mutual information is **information gain**.

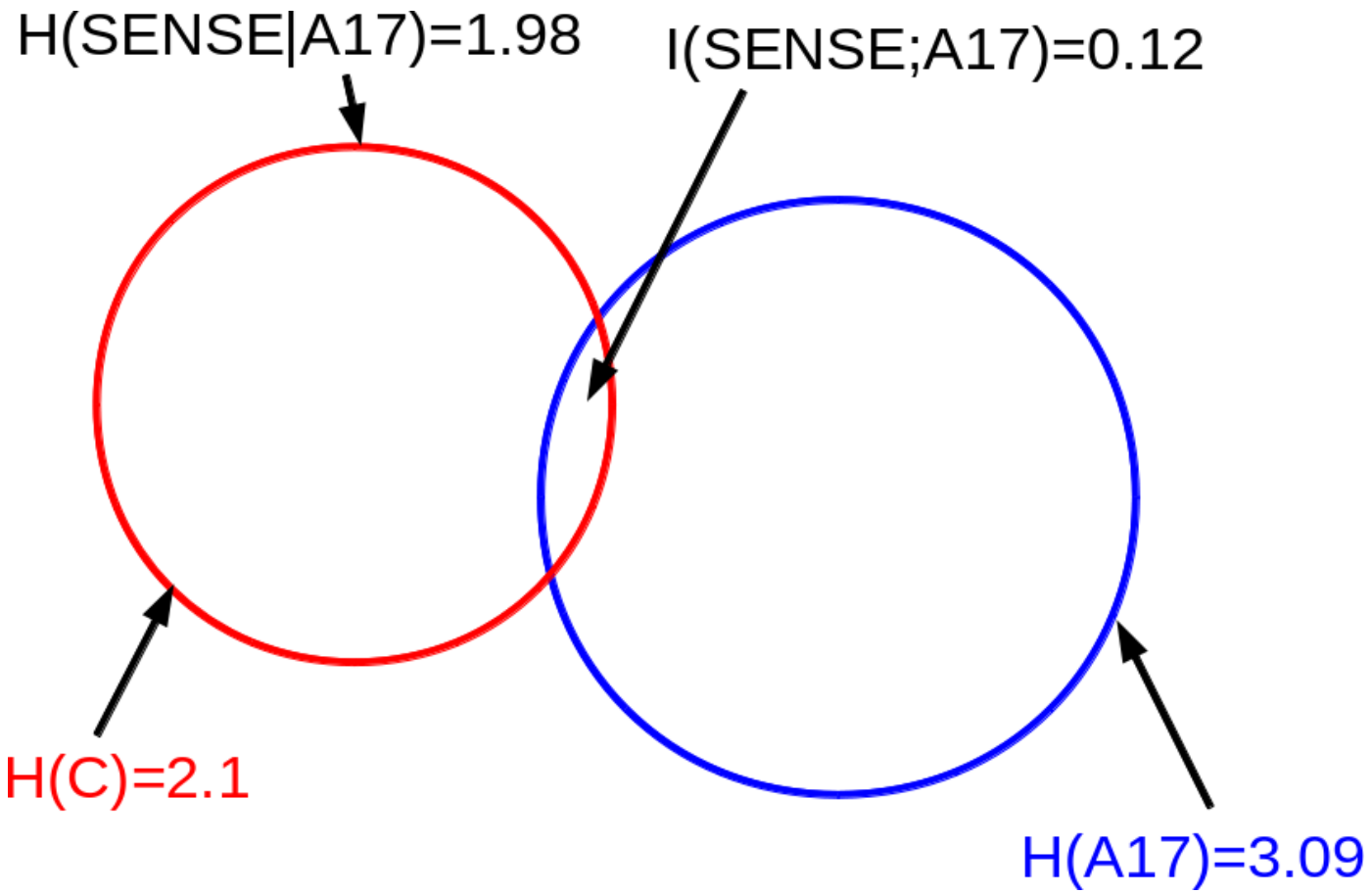
Conditional entropy – feature A4



Conditional entropy – feature A19



Conditional entropy – feature A17



Computing conditional entropy in R

You can use contingency table

```
##### computing H( C | A4 )

> p.A4 <-
    table(examples$A4)/nrow(examples)

> p.joint.sense.A4 <-
    as.vector(table(examples$A4, examples$SENSE))/nrow(examples)

> p.cond.sense.given.A4 <-
    p.joint.sense.A4 / rep(p.A4, 6)

> H.cond.sense.given.A4 <-
    - sum( p.joint.sense.A4[p.joint.sense.A4 > 0]
           * log2( p.cond.sense.given.A4[p.cond.sense.given.A4 > 0] ) )

> H.cond.sense.given.A4
[1] 1.95903
>
```

Explanation of the code

The formula for conditional entropy

$$\mathrm{H}(\texttt{SENSE} \,|\, \texttt{A4}) \quad = \quad - \sum_{y \in \texttt{SENSE}, x \in \texttt{A4}} \Pr(y, x) \cdot \log_2 \frac{\Pr(y, x)}{\Pr(x)}$$

Vectors in the code

- `p.joint.sense.A4` stands for $\Pr(y, x)$
- `p.cond.sense.given.A4` stands for $\dfrac{\Pr(y, x)}{\Pr(x)}$
- then `H.cond.sense.given.A4` computes the sum

User-defined functions in R

Structure of a user-defined function

```
myfunction <- function(arg1, arg2, ... ){
   ... statements ...
   return(object)
}
```

Objects in a function are local to the function.

Example – a function to calculate entropy

```
> entropy <- function(x){
+   p <- table(x) / NROW(x)
+   return( -sum(p * log2(p)) )
+ }
>

# invoking the function
> entropy(examples$SENSE)
[1] 2.107129
```

Example of user-defined functions – `entropy.R`

Usually we store our own functions in a separate file – look at `entropy.R`

Example – using `entropy.R` to calculate mutual information

```
> source("entropy.R")
>

### conditional entropy
> entropy.cond(examples$SENSE, examples$A4)
[1] 1.959030
> entropy.cond(examples$A4, examples$SENSE)
[1] 0.1221685
>

### mutual information I(SENSE;A4) = I(A4;SENSE)
> entropy(examples$SENSE) - entropy.cond(examples$SENSE, examples$A4)
[1] 0.1480985
> entropy(examples$A4) - entropy.cond(examples$A4, examples$SENSE)
[1] 0.1480985
>
```

Summary

Information theory provides a measure for comparing how features contribute to the knowledge about target class.

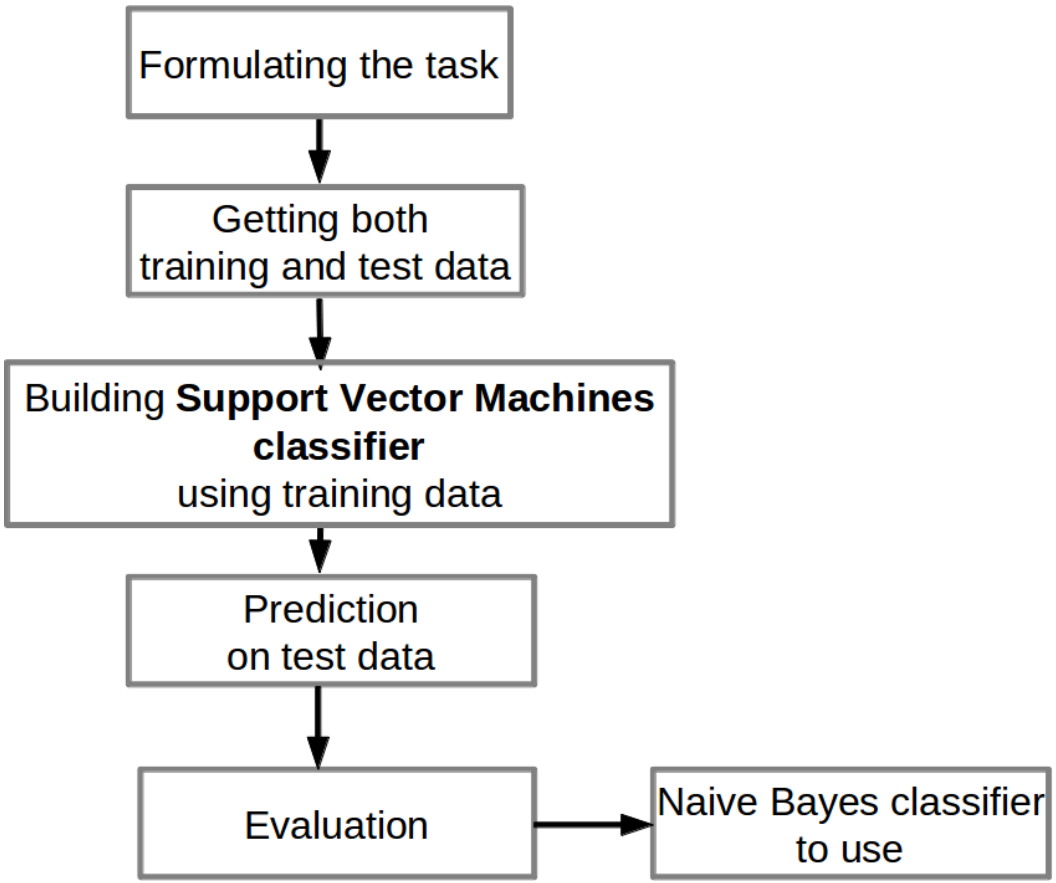
The lower conditional entropy $\mathrm{H}(C \mid A)$, the better chance that A is a useful feature.

However, since features typically interact, conditional entropy $\mathrm{H}(C \mid A)$ should NOT be the only criterion when you do feature selection. You need experiments to see if a feature with high information gain really helps.

Note
Also, decision tree learning algorithm makes use of entropy when it computes purity of training subsets.

Block 4.2
Support Vector Machines – Theory

Machine learning process - five basic steps



Assumption

Assume a binary classification task with target class values `+1`, `-1` ("blue", "red", resp.).

Decision boundary

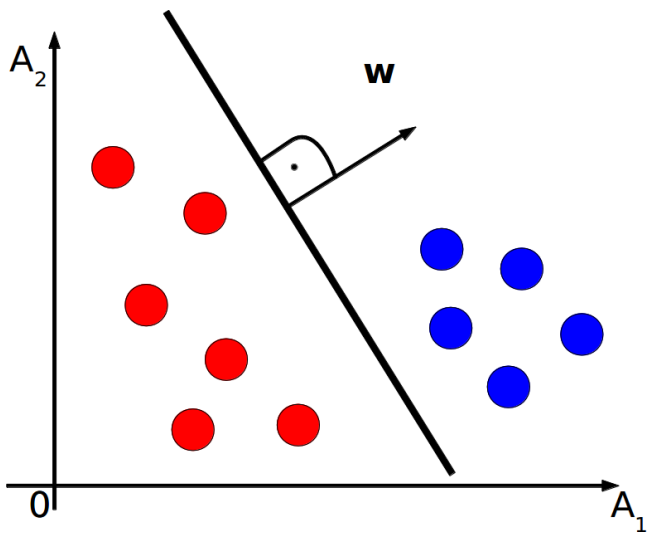
partitions a feature space into two sets, one for each class. Decision boundary takes a form of function $f(\mathbf{x})$.

Assumption 2

Assume a **linear** decision boundary, called **hyperplane**, of the form:

$$f(\mathbf{x}) = \mathbf{w}\mathbf{x} + b = \sum_{i=1}^{m} w_i x_i + b$$

where direction of w is perpendicular to the hyperplane and b determines position of the hyperplane with respect to the origin

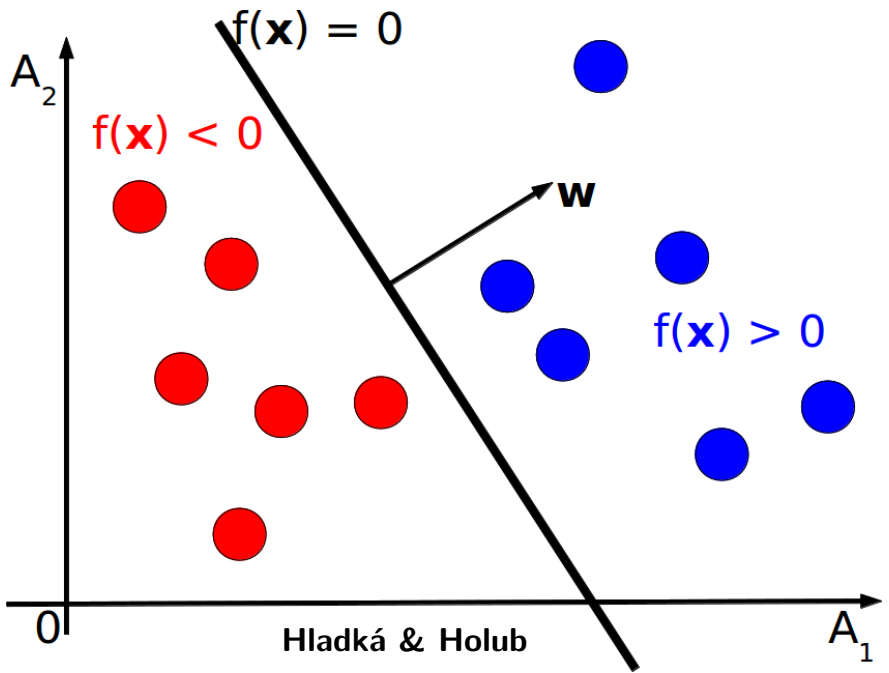


Hyperplane

A condition for instance $\mathbf{x}$ to be on the hyperplane is

$$f(\mathbf{x}) = 0, \text{ i.e. } \mathbf{wx} + b = 0$$

$f(\mathbf{x}) = 0$ defines hyperplane separating examples assigned to +1 from examples assigned to −1.

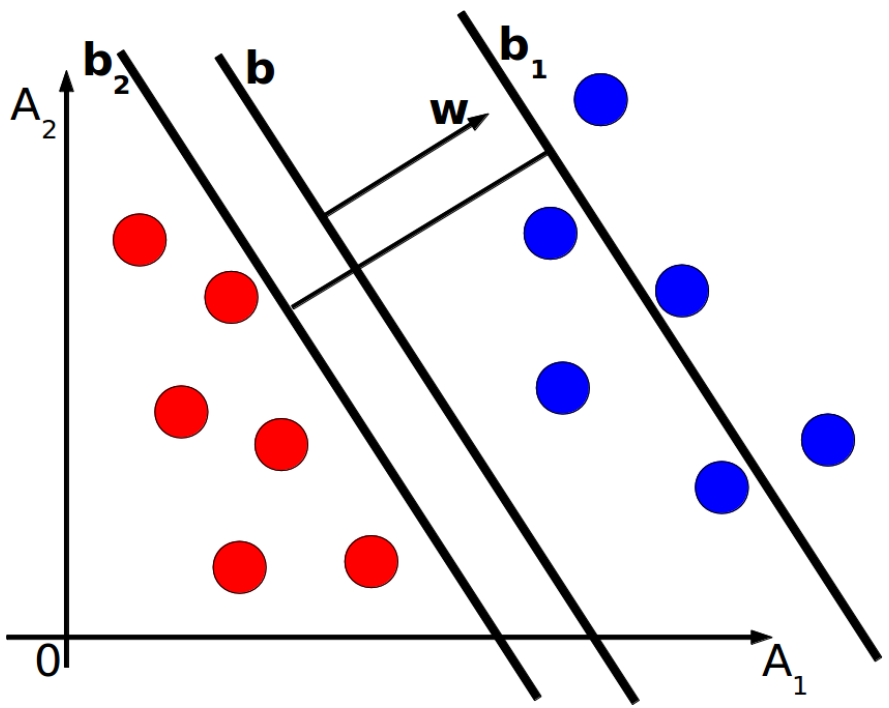


Hyperplane

What happens if the coffecient b changes?

The hyperplane moves along the direction of **w**. We get **parallel hyperplanes**.

Distane between two parallel hyperplanes $f_1(\mathbf{x}) = \mathbf{wx} + b_1$ and $f_2(\mathbf{x}) = \mathbf{wx} + b_2$ is $|b_1 - b_2|/||\mathbf{w}||$.



Linear separability

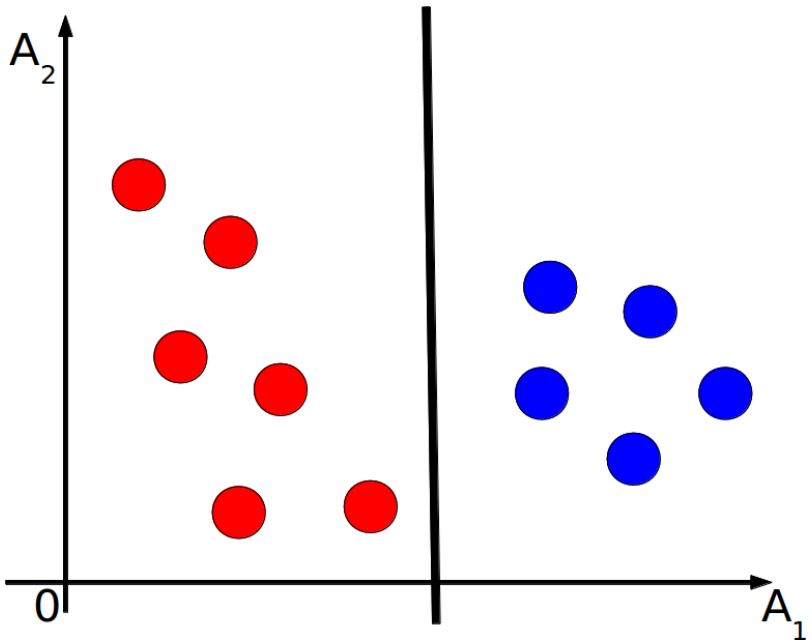
Assume data set $\{\langle \mathbf{x}_i, y_i \rangle\}$ and hyperplane $f(\mathbf{x})$. For each $\mathbf{x}_i$, $f(\mathbf{x}_i)$ is either positive of negative number.

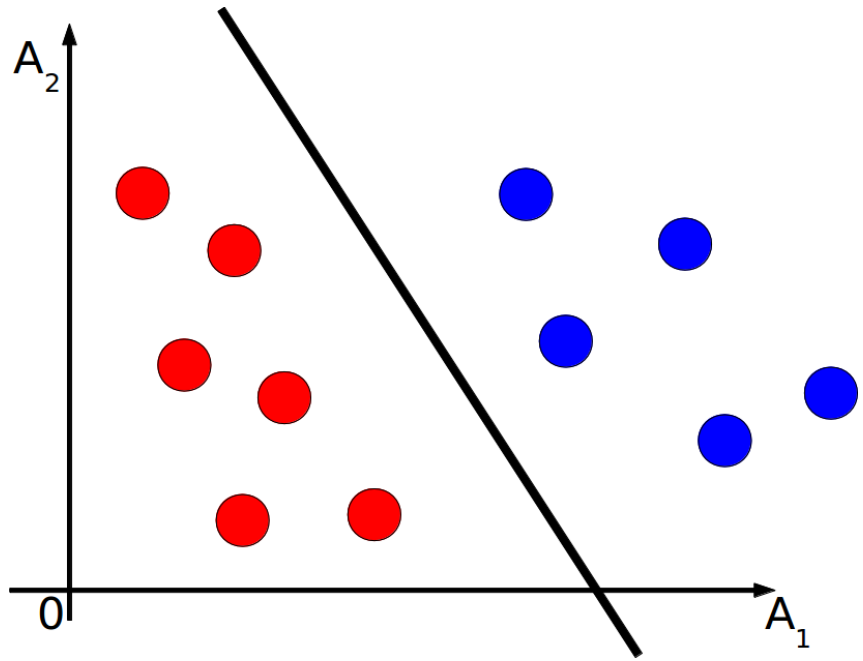
Data set is **linearly separable** if there exists a hyperplane $f(\mathbf{x}) = 0$ so that all examples from the data set are classified correctly, i.e.

$$y_i f(\mathbf{x}_i) = y_i(\mathbf{w}\mathbf{x}_i + b) > 0, \forall i = 1, ..., n; y_i \in \{-1, +1\}$$

Linear separability

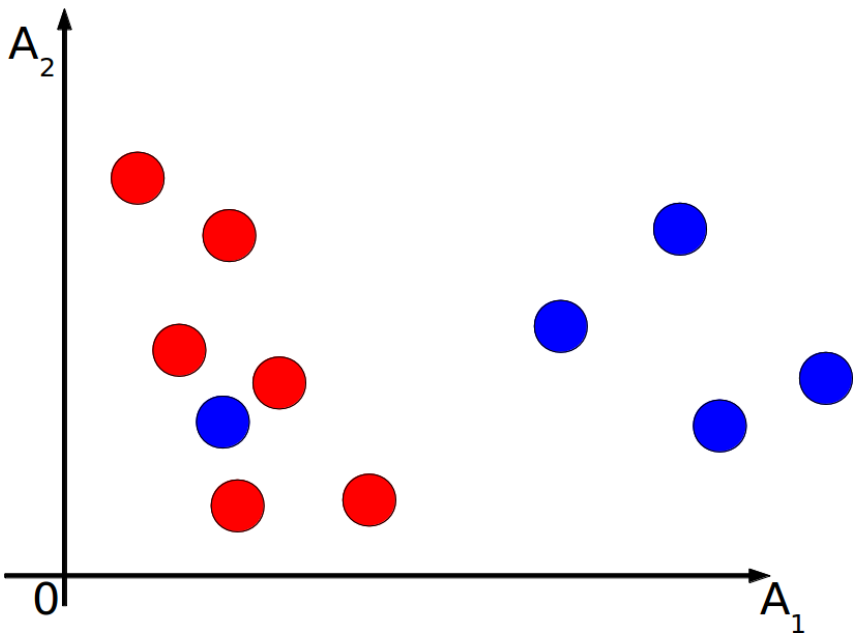
Examples

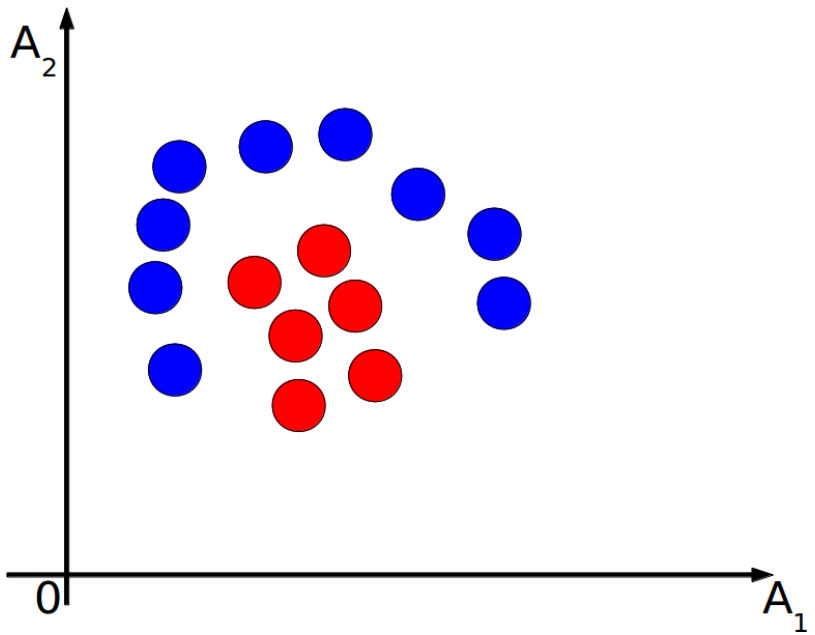




Not linear separability

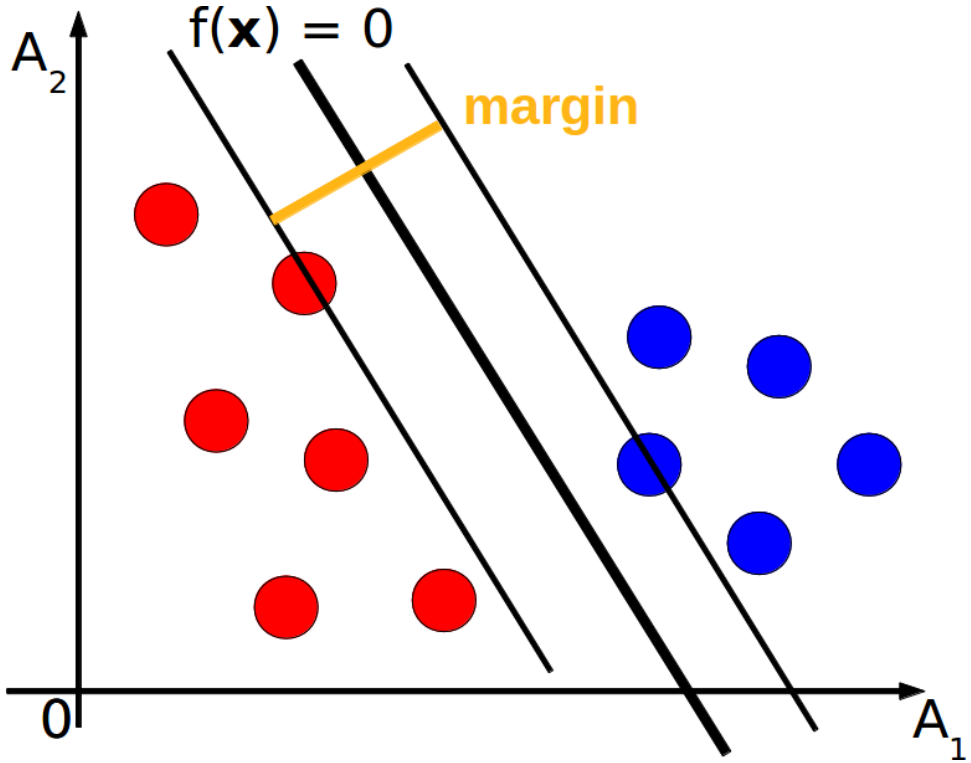
Examples





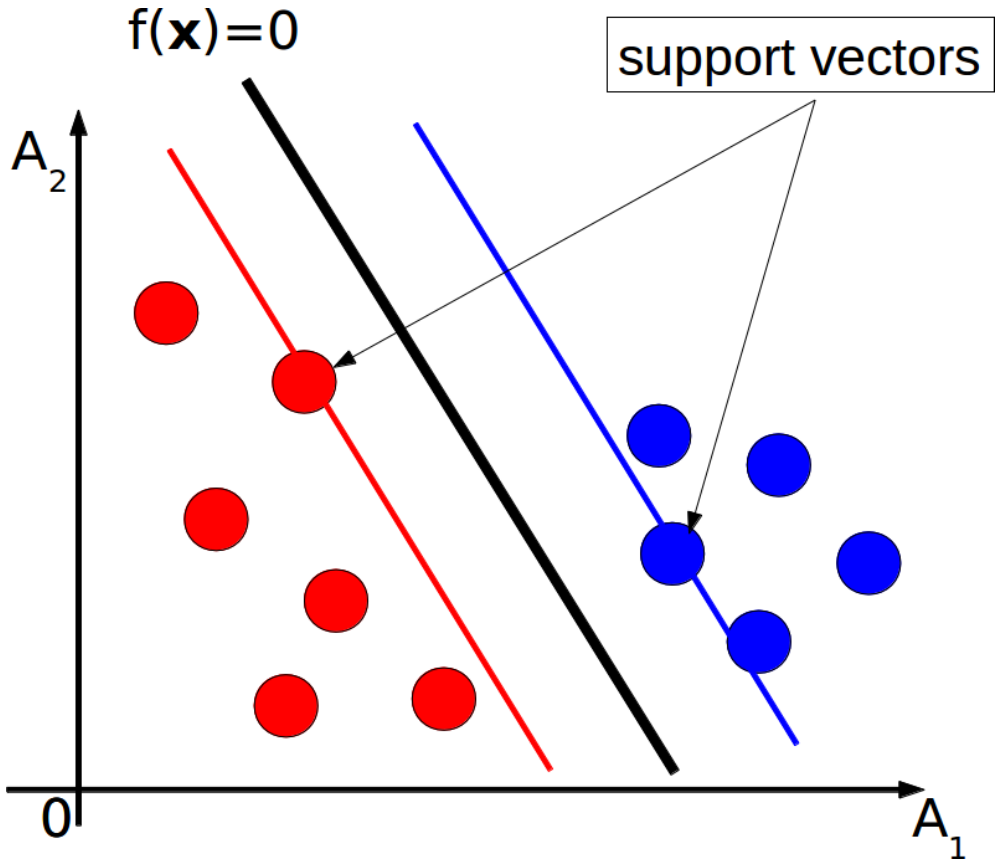
Margin of separating hyperplane

is a minimal distance between classes and separating hyperplane.



Support vectors

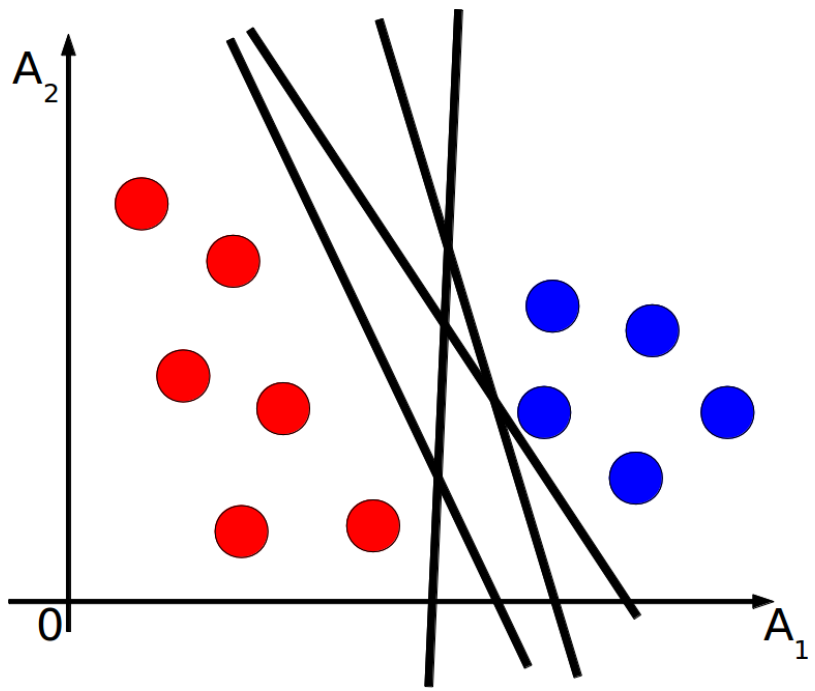
Separating hyperplane $f(\mathbf{x}) = 0$ is specified by a subset of training data. These examples are referred to as the **support vectors**.



Large margin SVM

Training examples are linearly separable

Infinitely many separating hyperplanes.

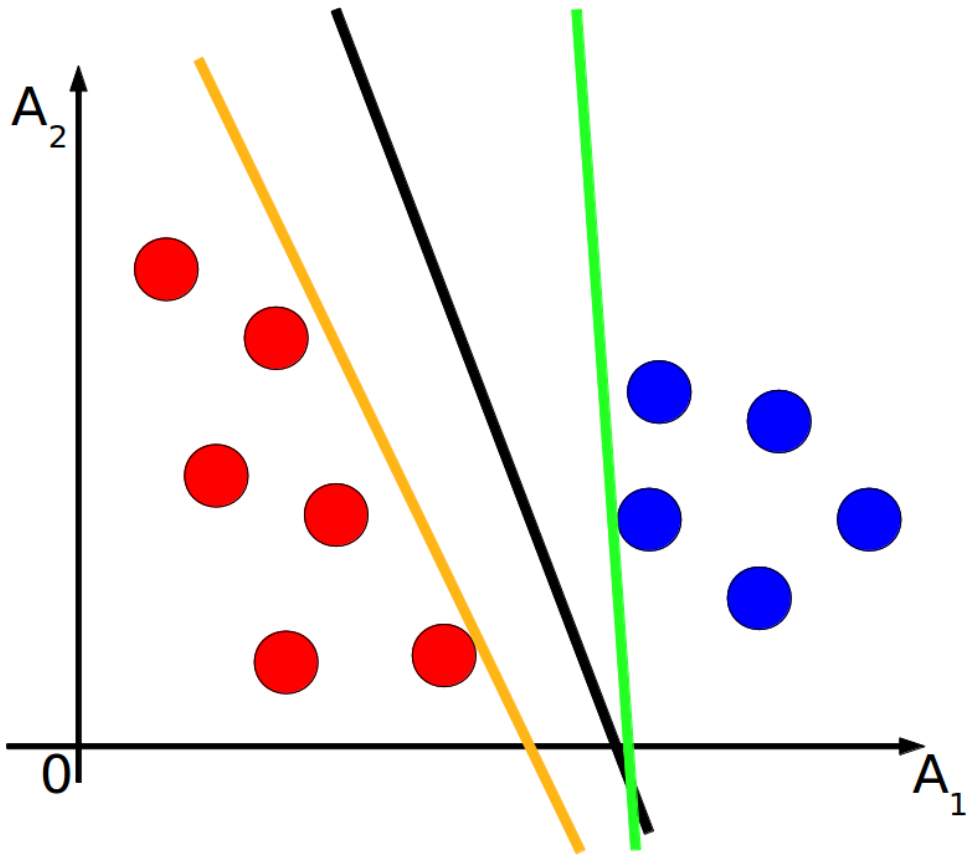


Which one is the best? The one with **maximum margin**

Large margin SVM

Maximum margin

Intuitively: Looking for a hyperplane that is maximally far away from any training example.

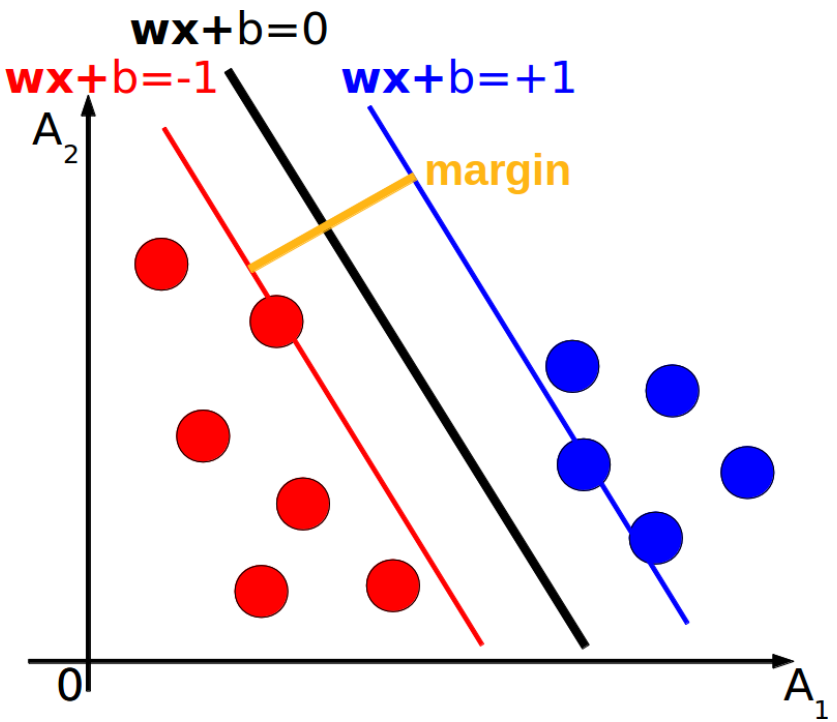


Large margin SVM

There are two supporting hyperplanes:

- $\mathbf{wx} + b = +1$, i.e. $\mathbf{wx} + (b - 1) = 0$
- $\mathbf{wx} + b = -1$ i.e. $\mathbf{wx} + (b + 1) = 0$

Distance between them, i.e. margin, is $|(b - 1) + (b + 1)|/||\mathbf{w}|| = 2/||\mathbf{w}||$.

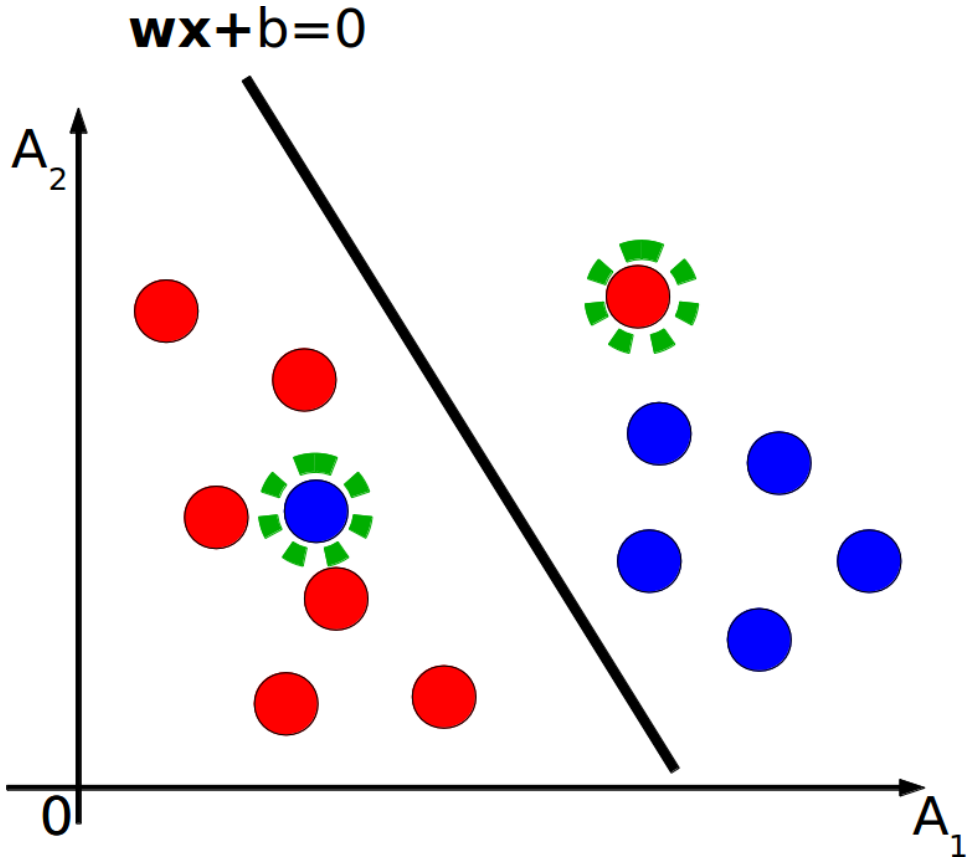


Large margin SVM

Since we want to maximize margin (i.e. $2/||\mathbf{w}||$), we need to minimize $||\mathbf{w}||$ on condition that all training examples are correctly classified.

Soft margin SVM

Training examples are not linearly separable

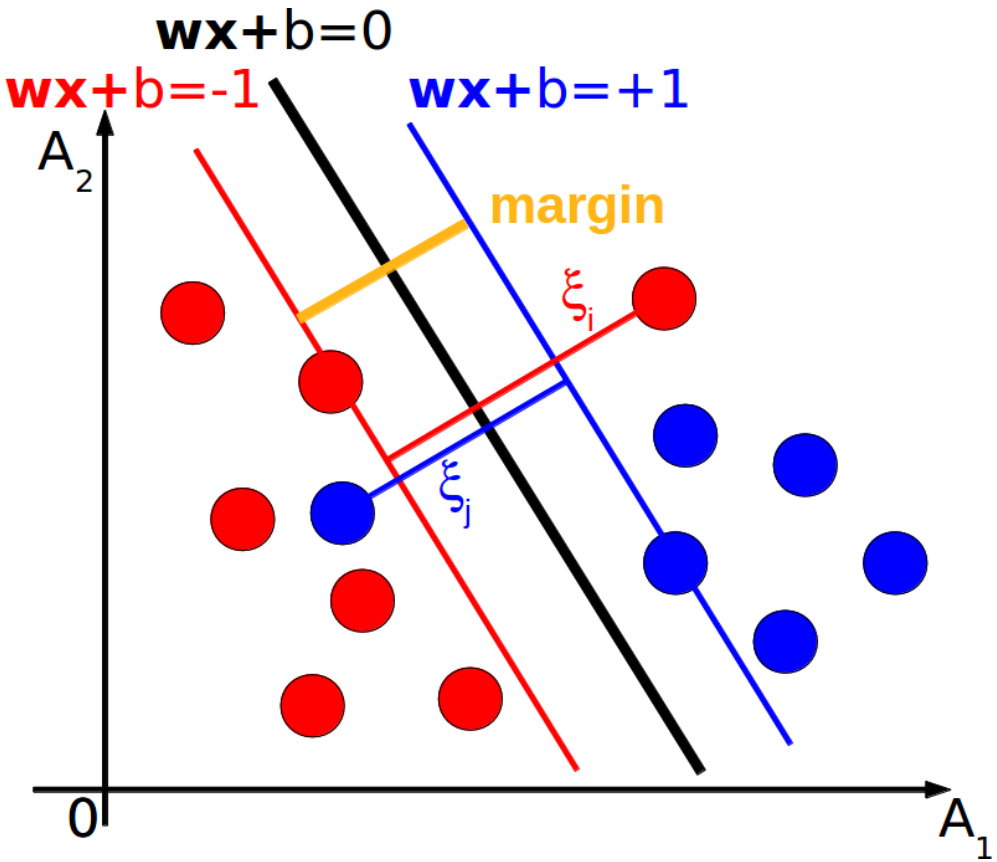


We want to handle such situation with hyperplanes as well.

Soft margin SVM

Infinitely many separating hyperplanes.

Use **slack variables** $\xi = \langle \xi_1, ..., \xi_m \rangle$ where ξ_i is the distance between training example $\mathbf{x}_i$ and its supporting hyperplane, $\xi_i \geq 0$.



Soft margin SVM

We have to control the trade-off between maximizing the margin and minimizing the number of misclassified instances.

I.e., we penalize the training error by *cost*: a larger cost tends to good generalization, a lower cost tends to overfitting.

Support vector machines learning

Two types of parameters in machine learning – Examples

ML algorithm	**learning parameters**	**hypothesis parameters**
DT	`minsplit` (minimum number of instances in the associated training subset in order for a decision to be attempted), ...	decisions
NB	–	probabilities
SVM	`cost`, `kernel`, `gamma`, ...	$\mathbf{w}, b$

Block 4.3
Support vector machines learning – Practice

- **Task**
  Decide whether the given word pair forms a semantic collocation
- **Objects**
  Word pairs (identified by two lemmas L1 and L2)
- **Target class**
  $\texttt{Class} = \{\texttt{YES}, \texttt{NO}\}$
- **Features**
  Numerical features A_1, A_9

Getting classified data

First, get examples into R

```
## Read the file with examples
> examples <- read.table("col.development.csv", header=T)
## Review the data
> str(examples)
'data.frame': 9232 obs. of  18 variables:
 $ Class: int  0 0 0 0 0 1 0 0 0 1 ...
 $ L1   : Factor w/ 2071 levels "absolutní","absolvent",..: ...
 $ L2   : Factor w/ 2456 levels "ABC-1","Abcházie",..: 148 ....
 $ A    : int  9 14 13 16 7 6 16 7 23 6 ...
 $ B    : int  6 345 232 3861 7438 15 174 454 582 1 ...
...
 $ A1   : num  9.93 9.47 7.68 4.58 5.12 ...
...
 $ A9   : num  0.835 0.421 0.547 0.289 0.208 ...
 $ A10  : num  0.0653 0.043 0.0537 0.0371 0.0352 ...
```

Splitting classified data into training and test data

Second, split them into the training and test sets

```
## Get the number of input examples
> num.examples <- nrow(examples)

## Set the number of training examples = 90% of examples
> num.train <- round(0.9 * num.examples)

## Set the number of test examples = 10% of examples
> num.test <- num.examples - num.train

## Check the numbers
> num.examples
[1] 9232
> num.train
[1] 8309
> num.test
[1] 923
```

Splitting classified data into training and test data

```
## Randomly split examples into training and test data

## Use set.seed() to be able to reconstruct the experiment
## with the SAME training and test sets

> set.seed(123)
> s <- sample(num.examples)
```

Splitting classified data into training and test data

```
### Get the training set
## First, generate indices of training examples
> indices.train <- s[1:num.train]

## Second, get the training examples
> train <- examples[indices.train,]

### Get the test set (see "pink" indeces)
> indices.test <- s[(num.train+1):num.examples]
> test <- examples[indices.test,]

## Check the results
> str(train); str(test)
```

Learning from training data

Load the package `e1071`

```
## Use the "rpart" package
## ! Run install.packages("e1071"), ***if not installed***.

# Check if the package is installed
> library()

## Load the package
> library(e1071)

# to get help info
> help(e1071)
```

`svm` documentation

```
>?svm()
svm          package:e1071          R Documentation

Support Vector Machines

Description:

     'svm' is used to train a support vector machine.
...
```

Learning from training data

`svm` documentation

```
svm(formula, data= , type= , cost= , kernel= , degree= ,
gamma= , ...  )
```

- `formula` is `y` $\sim$ `model` where
  - `y` is a target class
  - $\sim$ stands for 'is modeled as'
  - `model` is a combination of features (model by statisticians).
- `data` specifies the training set
- `type="C"` for classification

Learning from training data

- `cost` controls the training error with respect to margin
- `kernel` specifies a form of decision boundary
  - `kernel` ='linear': hyperplane
  - `kernel` ='polynomial' other type of kernel (more complex than hyperplane)
  - `kernel` ='rbf': other type of kernel (more complex than hyperplane)
- `degree` sets a degree to the polynomial kernel
- `gamma` is the parameter for the kernel
- ...

Default values:

`kernel` = 'radial', `cost` = 1, `gamma` = 1/k, where k is the number of features used in learning

Learning from training data

Manual setting of SVM parameters: `kernel = 'linear'`

```
> M1 <- svm(Class ~ A1+A9, type='C', data=train, kernel='linear')

## Display the model
> M1

Call:
svm(formula = Class ~ A1 + A9, data = train,
              type = "C", kernel = "linear")

Parameters:
   SVM-Type:  C-classification
 SVM-Kernel:  linear
       cost:  1
      gamma:  0.5

Number of Support Vectors:  2734
```

Learning from training data

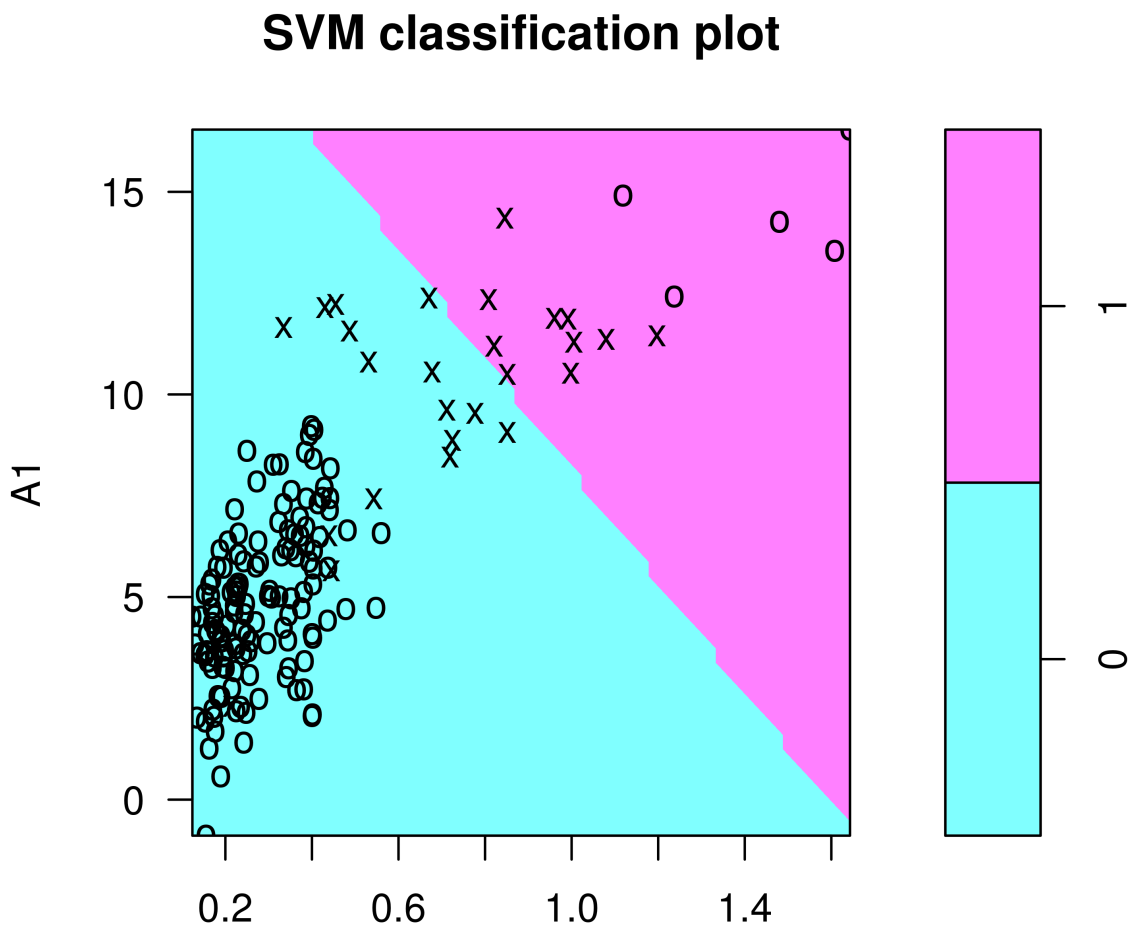
Manual setting of SVM parameters: `kernel = 'linear'`

```
## Display the boundary and first 200 training examples
> plot(M1, train[1:200,c(1,8,16)])
>
```

Learning from training data

Display the boundary and first 200 training examples

True classification TRUE – x, FALSE – o



Learning from training data

Manual setting of SVM parameters: `kernel = 'linear'`

```
> P1.test <- predict(M1, test, type="class")
> P1.train <- predict(M1, train, type="class")
```

Percentage of the correctly predicted senses on the **test set** = 84.07%
Percentage of the correctly predicted senses on the **train set** = 85.46%

Learning from training data

Manual setting of SVM parameters: `kernel = 'polynomial'`

```
> M2 <- svm(Class ~ A1+A9, type='C', data=train,
                  kernel='polynomial', degree = 2)
> P2.test <- predict(M2, test, type="class")
> P2.train <- predict(M2, train, type="class")
```

Percentage of the correctly predicted senses on the **test set** = 80.17%
Percentage of the correctly predicted senses on the **train set** = 83.31%

Learning from training data

Manual setting of SVM parameters: `kernel = 'polynomial'`

```
> M3 <- svm(Class ~ A1+A9, type='C', data=train,
                  kernel='polynomial', degree = 3)
> P3.test <- predict(M3, test, type="class")
> P3.train <- predict(M3, train, type="class")
```

Percentage of the correctly predicted senses on the **test set** = 81.26%
Percentage of the correctly predicted senses on the **train set** = 83.91%

Learning from training data

Manual setting of SVM parameters: `kernel = 'polynomial'`

```
> M4 <- svm(Class ~ A1+A9, type='C', data=train,
                  kernel='polynomial', degree = 4)
> P4.test <- predict(M4, test, type="class")
> P4.train <- predict(M4, train, type="class")
```

Percentage of the correctly predicted senses on the **test set** = 79.52%
Percentage of the correctly predicted senses on the **train set** = 82.83%

Learning from training data

Manual setting of SVM parameters: `kernel = 'radial'`

```
> M5 <- svm(Class ~ A1+A9, type='C', data=train, kernel='radial')
> P5.test <- predict(M5, test, type="class")
> P5.train <- predict(M5, train, type="class")
```

Percentage of the correctly predicted senses on the **test set** = 84.51%
Percentage of the correctly predicted senses on the **train set** = 85.46%

Learning from training data

Manual setting of SVM parameters:

`kernel = 'radial', cost = 10`

```
> M6 <- svm(Class ~ A1+A9, type='C', data=train,
                        kernel='radial', cost=10)
> P6.test <- predict(M6, test, type="class")
> P6.train <- predict(M6, train, type="class")
```

Percentage of the correctly predicted senses on the **test set** = 84.72%
Percentage of the correctly predicted senses on the **train set** = 85.5%

Learning from training data

Manual setting of SVM parameters:

`kernel = 'radial', cost = 100`

```
> M7 <- svm(Class ~ A1+A9, type='C', data=train,
                        kernel='radial', cost=100)
> P7.test <- predict(M7, test, type="class")
> P7.train <- predict(M7, train, type="class")
```

Percentage of the correctly predicted senses on the **test set** = 85.16%
Percentage of the correctly predicted senses on the **train set** = 85.73%

Learning from training data

Grid search is an exhaustive searching through a manually specified values of learning parameters.

Example

Let 10, 100, 1000 be manually specified values for `cost` and 0.1, 0.2, 0.5, 1.0 be manually specified values for `gamma`. Grid search then trains with each pair (`cost`, `gamma`), i.e. on 12 pairs.

Learning from training data

Grid search in R using the function `tune.svm`

```
## Use 1,000 training examples (just for illustration)
## Set values for cost to 1, 10, 100
## Set values for gamma to 0.01, 0.1, 1

> grid <- tune.svm(Class ~ A1+A9, data=train[1:1000,],
      gamma=10^seq(-2,0,by=1), cost=10^seq(0,2,by=1))
> summary(grid)

Parameter tuning of ‘svm’:
...
- best parameters:
 gamma cost
     1   10
...
> best.gamma <- grid$best.parameters[[1]]
> best.cost <- grid$best.parameters[[2]]
```

Learning from training data

Grid search in R using the function `tune.svm`

```
> M8 <- svm(Class ~ A1+A9, type='C', data=train[1:1000,],
      kernel='radial', cost=best.cost, gamma=best.gamma)
> P8.test <- predict(M8, test, type="class")
> P8.train <- predict(M8, train, type="class")
```

Percentage of the correctly predicted senses on the test set = 83.42

Percentage of the correctly predicted senses on the train set = 84.98

Homework 4.1

1. Download the `col.development.csv` data set
2. Load it into R
3. Use the same training and test sets you generated when solving Homework 3.1, train and test, resp.
4. Use the same feature set you worked with when solving Homework 3.1
5. Build SVM classifiers with the following settings
   - `data`=train[1:200,], `kernel`='linear', `cost` = 10
   - `data`=train[1:200,], `kernel`='linear', `cost` = 100
   - get optimal `cost` value using `tune.svm`(`Class` ~ ..., `data`=train[1:200,], `cost`=10^ seq(0,2,by=0.5), `gamma`=10^ seq(-2,0,by=0.5)) -> `best.cost`, `best.gamma`
   - `data`=train[1:200,], `kernel`='linear', `cost` = best.cost, `gamma` = best.gamma
6. List the percentage of correctly classified test and training examples for each trained classfier, i.e. you will list 6 (3 x 2) numbers.