INF5830 – 2017 FALL

NATURAL LANGUAGE PROCESSING

Jan Tore Lønning, Lecture 2, 29.8

Today - Classification

- Motivation
- Classification of classification
- Some simple examples
- Set-up of experiments
- Evaluation
- Naive Bayes classifier (Bernoulli)

Motivation

Classification

- Jurafsky og Martin, 3.ed. Ch. 6
  Naive Bayes Classification and Sentiment
  - slides 1-7
- NLTK book, Ch. 6

Classification

Classification

- Can be rule-based, but mostly machine learned
- Text classification is a sub-class

- Text classification examples:
  - Spam detection
  - Genre classification
  - Language classification
  - Sentiment analysis:
    - Positive-negative

- Other types of classification:
  - Word sense disambiguation
  - Sentence splitting

Machine learning

1. Supervised
   1. Classification
      1. Naive Bayes
      2. Many more
   2. Regression
2. Unsupervised
   1. Clustering
   2. …
3. Semi-supervised
4. Reinforcement learning

- Supervised:
  - Given classes
  - Given examples of correct classes
- Unsupervised:
  - Construct classes

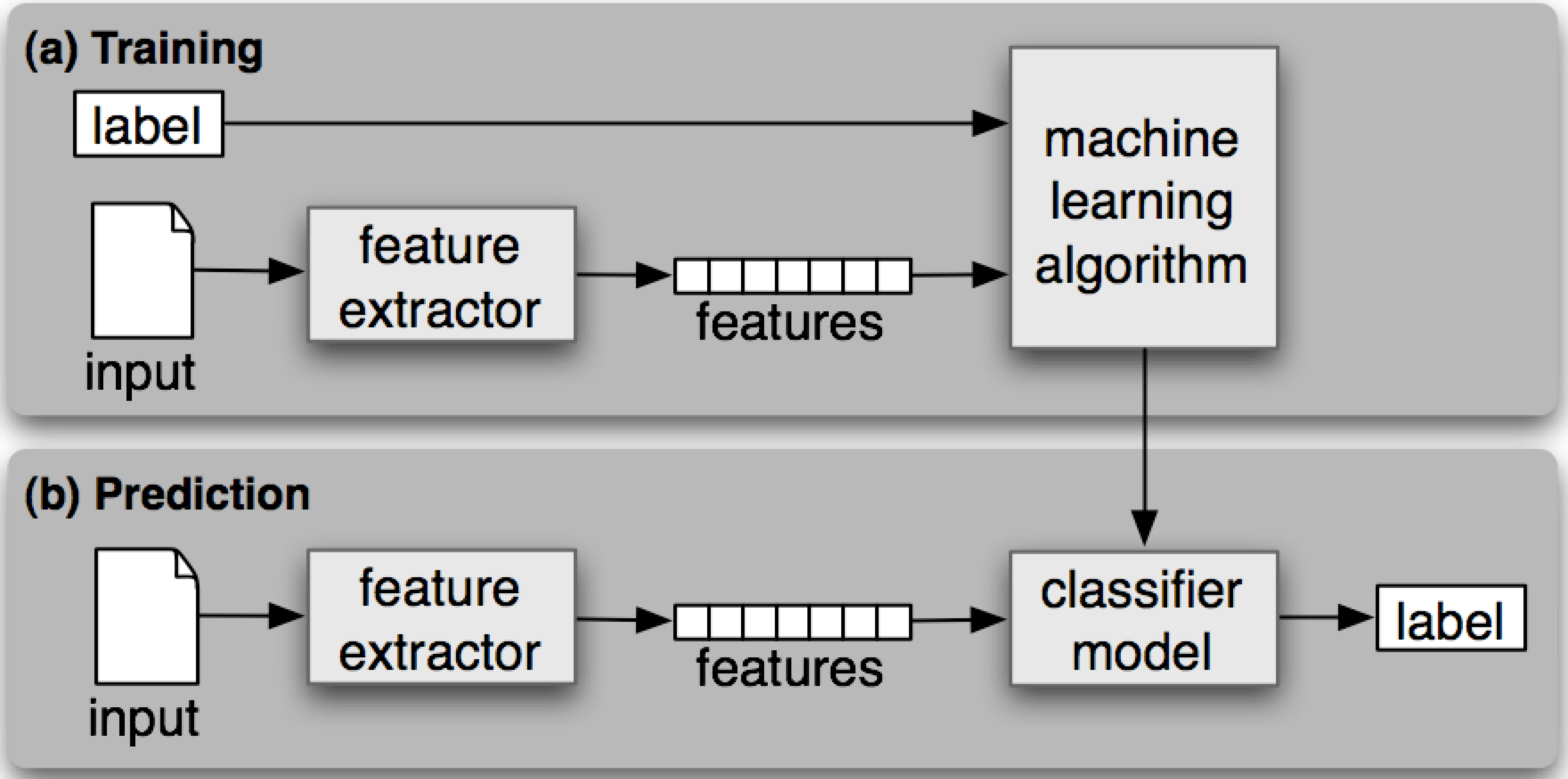
A variety of ML classifiers

- k-Nearest Neighbors
- Rocchio

INF4820

- Decision Trees
- Naive Bayes
- Maximum entropy (Logistic regression)
- Support Vector Machines
  - (INF4490)
- and more

Classification



Supervised classification

- Given
  - a well-defined set of objects, O
  - a given set of classes, $S=\{s_1, s_2, \ldots, s_k\}$
- Goal: a classifier, γ, a mapping from O to S
- For supervised training one needs a set of pairs from OxS

Task	O	S
Spam classification	E-mails	Spam, no-spam
Language clssification	Pieces of text	Arabian, Chinese, English, Norwegian, …
Word sense disambiguation	Occurrences of "bass"	Sense1, …, sense8

Features

- To represent the objects in O, extract a set of features

Object: person
Features:
- height
- weight
- hair color
- eye color
- …

Object: email
Features:
- length
- sender
- contained words
- language
- …

Be explicit:

- Which features
- For each feature
  - The type
    - Categorical
    - Numeric (Discrete/Continuous)
  - The value space
- Cf. First lecture

Supervised classification

- A given set of classes, $S=\{s_1, s_2, \ldots, s_k\}$
- A well defined class of objects, O

- Some features $f_1, f_2, \ldots, f_n$
- For each feature: a set of possible values $V_1, V_2, \ldots, V_n$
- The set of feature vectors: $V= V_1\times V_2\times\ldots\times V_n$
- Each object in O is represented by some member of V:
  - Written $(v_1, v_2, \ldots, v_n)$, or
  - $(f_1=v_1, f_2=v_2, \ldots, f_n=v_n)$
- A classifier, γ, can be considered a mapping from V to S

Examples

Language classifier

- C = {English, Norwegian,…}
- O is the set of strings of letters
- f_1 is last letter of o
- V_1= {a, b, c,…, å}
- f_2 is the last two letters
- V_2 is all two letter combinations
- f_3 is the length of o,
- V_3 is 1, 2, 3, 4, …

Word sense disambiguation

- C = {fish, music}
- O: all occurrences of "bass"
- $f_i = f_{wi}$: word w_i occurs in same sentence as "bass", where
  - w_1 = *fishing*, w_2 = *big*, …,
  - w_{11} = *guitar*, w_{12} = *band*
- $V_1=V_2=\ldots=V_{12}=\{1,0\}$
- Example:
  - o = (0,0,0,1,0,0,0,0,0,0,1,0)
  - o = ($f_{fishing}$=0, …, f_{guitar}=1, f_{band}=0)

Simple examples from NLTK

NLTK-example 1: names

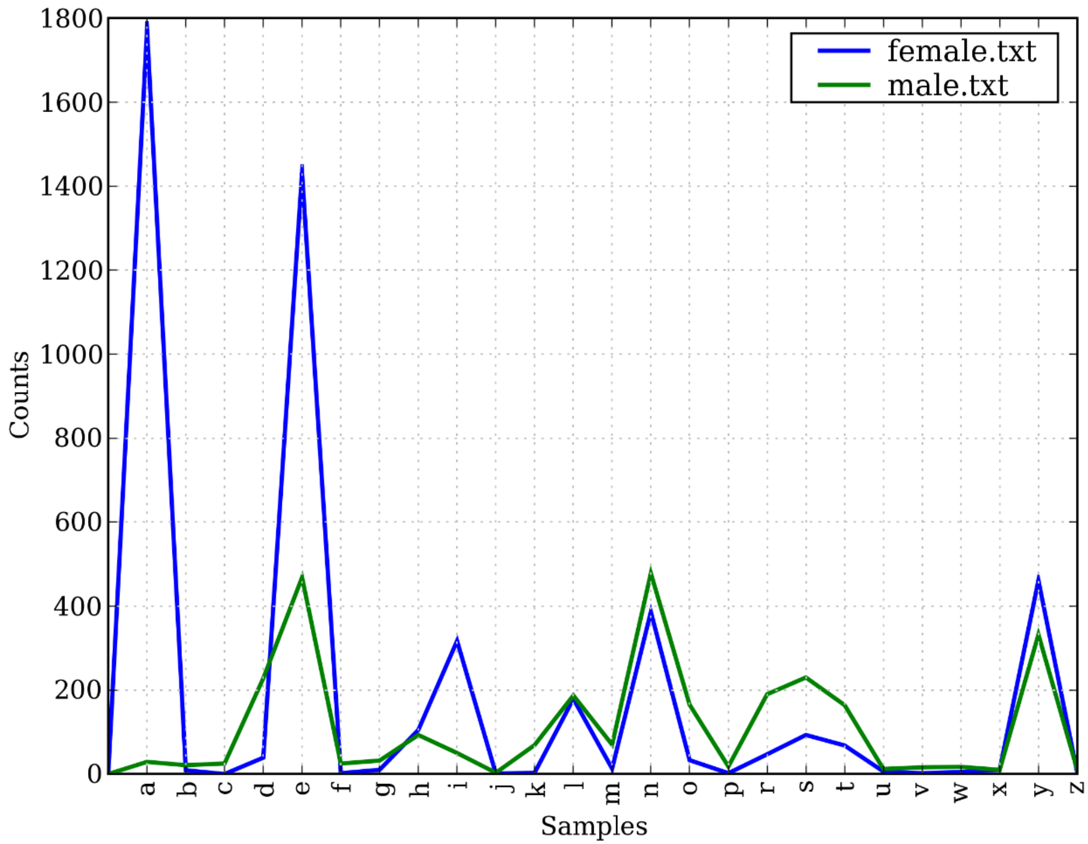

```
In [2]: def gender_features(word):
   ...:     return {'last letter': word[-1]}

In [3]: gender_features('Shrek')
Out[3]: {'last letter': 'k'}

In [4]: from nltk.corpus import names

In [5]: labeled_names =
  ([(name, 'male') for name in names.words('male.txt')] +
   [(name, 'female') for name in names.words('female.txt')])
```

NLTK: names



1800
1600
1400
1200
1000
800
600
400
200
0
Counts
a
b
c
d
e
f
g
h
i
j
k
l
m
n
o
p
r
s
t
u
v
w
x
y
z
Samples
female.txt
male.txt

NLTK-example 1, contd.


```
In [6]: import random
In [8]: random.shuffle(labeled_names)
In [9]: featuresets = [(gender_features(n), gender)
                       for (n, gender) in labeled_names]
In [10]: train_set, test_set =
                       featuresets[500:], featuresets[:500]
```

When you conduct several experiments, use the same split so you can compare the results.

Split before you extract features

NLTK-example1, contd.


```
In [11]: classifier = nltk.NaiveBayesClassifier.train(train_set)
In [12]: classifier.classify(gender_features('Neo'))
Out[12]: 'male'
In [13]: classifier.classify(gender_features('Ada'))
Out[13]: 'female'
In [31]: print(nltk.classify.accuracy(classifier, test_set))
0.79
```

Why do I get 0.79 and the book 0.75?

Example 1 ctd.

- A given set of classes, $S=\{s_1, s_2, \ldots, s_k\}$ = {'male', 'female'}
- A well defined class of objects, O = {'Ada', 'Albert', …} = all strings of letters

- Some features $f_1, f_2, \ldots, f_n$, only f_1 'last_letter'
- For each feature: a set of possible values $V_1, V_2, \ldots, V_n$
  $V_1 = \{a, b, c, \ldots, z\}$
- The set of feature vectors: $V = V_1 \times V_2 \times \ldots \times V_n$
- Each object in O is represented by some member of V:
  - Written $(v_1, v_2, \ldots, v_n)$, or (e.g. 'u')
  - $(f_1=v_1, f_2=v_2, \ldots, f_n=v_n)$ (e.g. last_letter: 'u')
- A classifier, γ, can be considered a mapping from V to S

NLTK-eksempel 2


```
In [56]: def gender_features2(name):
   ...:     features = {}
   ...:     features["first_letter"] = name[0].lower()
   ...:     features["last_letter"] = name[-1].lower()
   ...:     for letter in 'abcdefghijklmnopqrstuvwxyz':
   ...:         features["count({})".format(letter)] = name.lower().count(letter)
   ...:         features["has({})".format(letter)] = (letter in name.lower())
   ...:     return features

In [59]: featuresets2 = [(gender_features2(n), gender) for (n, gender) in labeled_names]
In [60]: train_set2, test_set2 = featuresets2[500:], featuresets2[:500]
In [61]: classifier2 = nltk.NaiveBayesClassifier.train(train_set2)

In [62]: print(nltk.classify.accuracy(classifier2, test_set2))
0.78
```

NLTK-example 2


```
In [56]: def gender_features2(name):
   ...:     features = {}
   ...:     features["first_letter"] = name[0].lower()
   ...:     features["last_letter"] = name[-1].lower()
   ...:     for letter in 'abcdefghijklmnopqrstuvwxyz':
   ...:         features["count({})".format(letter)] = name.lower().count(letter)
   ...:         features["has({})".format(letter)] = (letter in name.lower())
   ...:     return features
```

What are the features here?

- How many?
- What are their resp. value spaces?

Comparing features

- NLTK-boook printed:
  - gender_features (gf1) yields acc 0.758
  - gender_features2 (gf2) yields acc 0.748
- Indicates
  - More features aren't always better
  - Danger that gender_features2 "is overfitting":
    - Adapt itself too much to the training set
- Web edition: gf1_acc: 0.77, gf2_acc: 0.768
- We: gf1_acc: 0.79, gf2_acc: 0.78

A more complex picture

- 10 experiments
- Do not draw hasty conclusions from small differences
- Variation
- We will later consider how statistics may tell us which differences are significant

- Accuracy:

Exp.no	gf1	gf2
1	0.760	0.756
2	0.770	0.784
3	0.782	0.774
4	0.772	0.796
5	0.744	0.744
6	0.760	0.792
7	0.776	0.754
8	0.782	0.784
9	0.774	0.774
10	0.772	0.794

NLTK-book's best shot


```
def feat_suff_1_2(word):
    return {'suffix1': word[-1], 'suffix2': word[-2:]}
```

Exp.no	gf1	gf2	feat_suff_1_2
1	0.764	0.778	0.766
2	0.760	0.748	0.772
3	0.758	0.764	0.772
4	0.772	0.786	0.800
5	0.748	0.766	0.752
6	0.742	0.792	0.768
7	0.758	0.766	0.784
8	0.752	0.788	0.774
9	0.752	0.756	0.778
10	0.744	0.778	0.776

Beware:


```
def feat_suff_1_2(word):
    return {'suffix1': word[-1],
              'suffix2': word[-2:]}
```

=/=

```
def feat_two_last(word):
    return {'suffix1': word[-1],
             'suffix2': word[-2]}
```

- Accuracy:

Exp.no	f_suff_1_2	f_two_last
1	0.792	0.786
2	0.754	0.746
3	0.792	0.780
4	0.768	0.772
5	0.786	0.784
6	0.782	0.762
7	0.798	0.792
8	0.812	0.784
9	0.794	0.770
10	0.774	0.766

Movie reviews 1

- from nltk.corpus import movie_reviews
- documents = [(list(movie_reviews.words(fileid)), category)
  for category in movie_reviews.categories()
  for fileid in movie_reviews.fileids(category)]
- random.shuffle(documents)
- all_words = nltk.FreqDist(w.lower() for w in
  movie_reviews.words())
- word_features = [w for (w,_) in all_words.most_common(2000)]
- wrong_features = list(all_words)[:2000] #Wrong (earlier version)

Movie reviews 2

- def document_features(word_features, document):
    document_words = set(document)
    features = {}
    for word in word_features:
        features['contains({})'.format(word)] =
            (word in document_words) #True or False
    return features

- featuresets = [(document_features(word_features, d), c)
    for (d,c) in documents]

- train_set, test_set = featuresets[100:], featuresets[:100]
- classifier = nltk.NaiveBayesClassifier.train(train_set)

- print(nltk.classify.accuracy(classifier, test_set))

  0.83

Movie reviews 3

Peoperties

- Two classes: ‘neg’, ‘pos’
- Features’:
  - 2000 most frequent words in corpus
- Values: True/False
  - Don't count number of occs in each corpus
  - All features (words) not in corpus gets value ‘’False’’

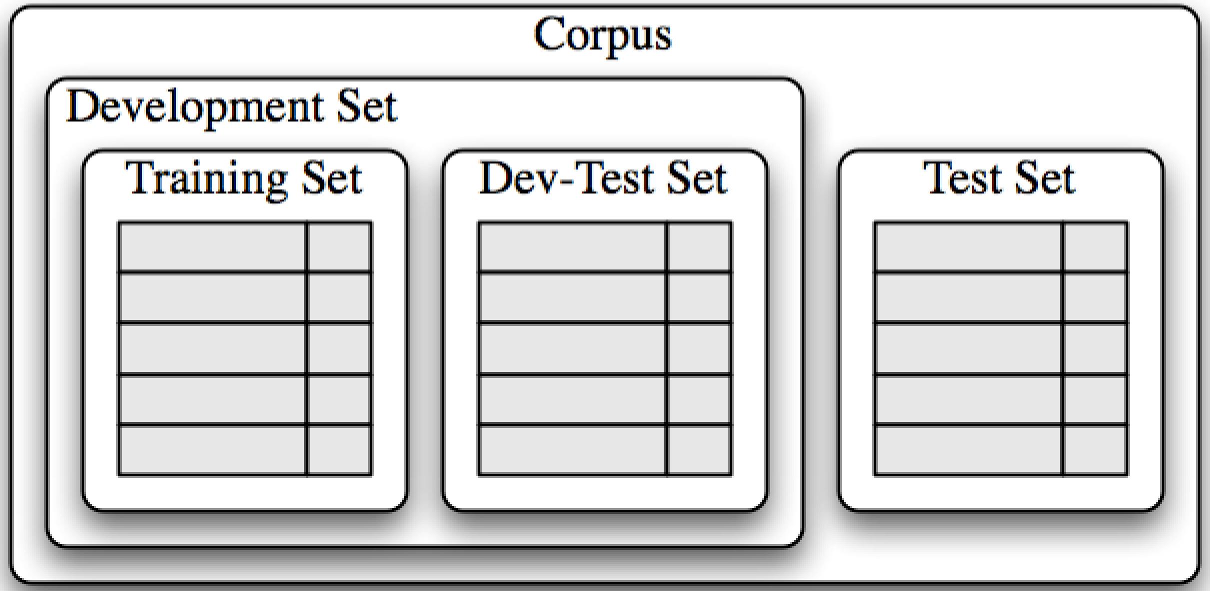
Comments

- Strictly speaking, the "most frequent" should be counted from training data only

29

Set-up for experiments

Set-up for experiments



- Before you start: split into development set and test set.
- Hide the test set
- Split development set into Training and Development-Test set
- Use training set for training a learner
- Use Dev(-Test) for repeated evaluation in the test phase
- Finally test on the test set!

Procedure

1. Train classifier on training set
2. Test it on dev-test set
3. Compare to earlier runs, is this better?
4. Error analysis: What are the mistakes (on dev-test set)
5. Make changes to the classifier
6. Repeat from 1

===================

- When you have run empty on ideas, test on test set. Stop!

Cross-validation

- Small test sets → Large variation in results
- N-fold cross-validation:
  - Split the development set into n equally sized bins
    - (e.g. n = 10)
  - Conduct n many experiments:
    - In experiment m, use part m as test set and the n-1 other parts as training set.
  - This yields n many results:
    - We can consider the mean of the results
    - We can consider the variation between the results.
      - Statistics!

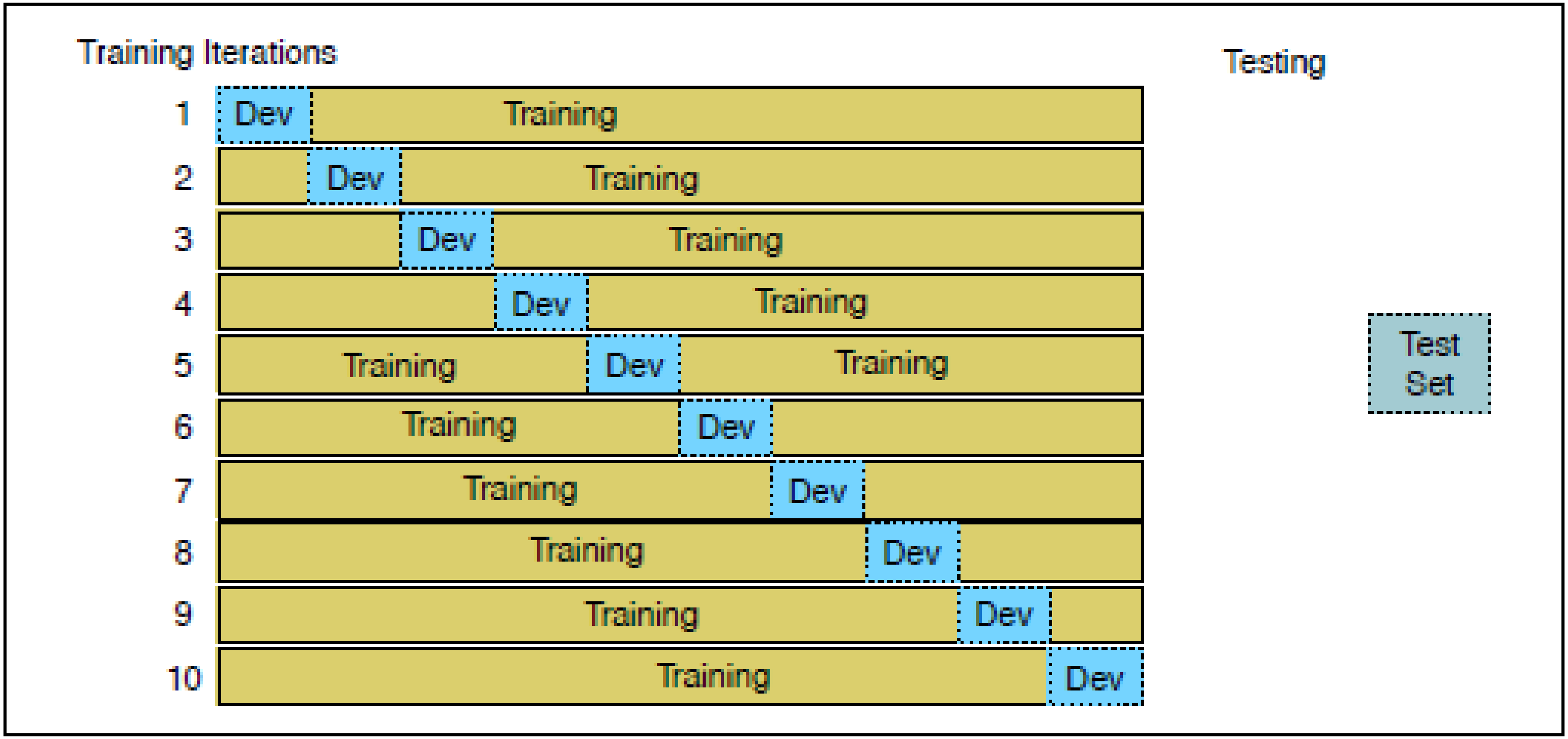


Figure 6.7 10-fold crossvalidation

- But take away a final test set first!

Evaluation

Evaluation measure: Accuracy

- What does accuracy 0.81 tell us?
- Given a test set of 500 sentences:
  - The classifier will classify 405 correctly
  - And 95 incorrectly
- A good measure given:
  - The 2 classes are equally important
  - The 2 classes are roughly equally sized
  - Example:
    - Woman/man
    - Movie reviews: pos/neg

But

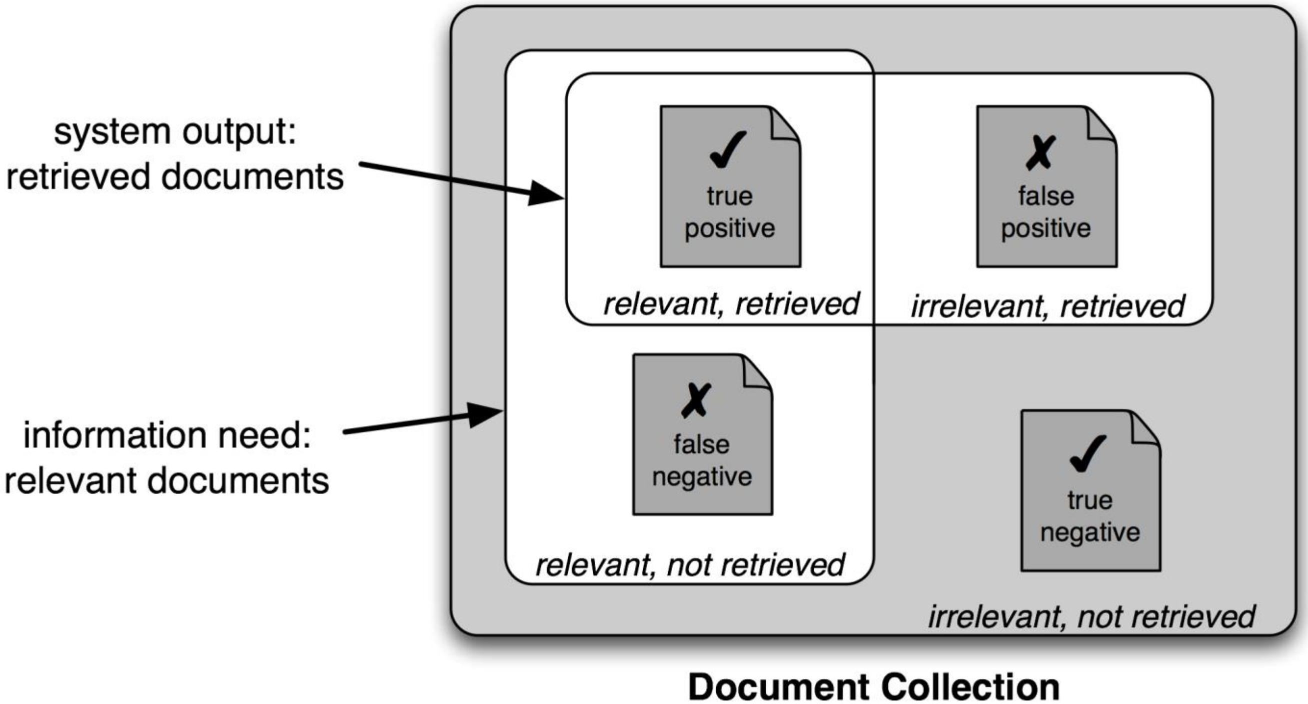
- For some tasks the classes aren't equally important
  - Worse too loose an important mail than to receive yet another spam mail
- For some tasks the different classes have different sizes.

Information retrieval (IR)

- Traditional IR, e.g. a library
  - Goal: Find all the (5) documents on a particular topic out of 100 000 documents
  - The system delivers 5 documents: all irrelevant
    - What is the accuracy?

- For these tasks, focus on
  - The relevant documents
  - The documents returned by the system
- Forget the
  - Irrelevant documents which are not returned

IR - evaluation



Confusion matrix

		gold standard labels		
		gold positive	gold negative	
system output labels	system positive	**true positive**	**false positive**	**precision** = $\frac{tp}{tp+fp}$
	system negative	**false negative**	**true negative**	
		recall = $\frac{tp}{tp+fn}$		**accuracy** = $\frac{tp+tn}{tp+fp+tn+fn}$

Figure 6.4 Contingency table

- Beware what the rows and columns are:
  - NLTKs ConfusionMatrix swaps them compared to this table

Evaluation measures

		Is in C	
		Yes	NO
Classifier	Yes	tp	fp
	No	fn	tn

- Accuracy: (tp+tn)/N
- Precision: tp/(tp+fp)
- "Recall" (gjenfinning): tp/(tp+fn)
- F-score kombinerer recall og precision

- $F_1 = \frac{2P}{P+R} \left(= \frac{1}{\frac{\frac{1}{R}+\frac{1}{P}}{2}} \right)$
- F_1 called "harmonic mean"
- General form
  - $F = \frac{1}{\alpha\frac{1}{P}+(1-\alpha)\frac{1}{R}}$
  - for some $0 < \alpha < 1$
  - α determines the weighting of P vs. R

More than 2 classes

		gold labels			
		urgent	normal	spam	
system output	urgent	8	10	1	$\text{precision}_u = \frac{8}{8+10+1}$
	normal	5	60	50	$\text{precision}_n = \frac{60}{5+60+50}$
	spam	3	30	200	$\text{precision}_s = \frac{200}{3+30+200}$
		$\text{recall}_u = \frac{8}{8+5+3}$	$\text{recall}_n = \frac{60}{10+60+30}$	$\text{recall}_s = \frac{200}{1+50+200}$	

Figure 6.5 Confusion matrix for a three-class categorization task, showing for each pair of classes (c_1, c_2), how many documents from c_1 were (in)correctly assigned to c_2

- Accuracy: $\frac{8+60+200}{8+10+1+5+60+50+3+30+200} = \frac{268}{367}$
- Precision, recall and f-score can be calculated for each class against the rest

Naive Bayes

Naive Bayes: Decision

- Given an object
  - $\langle f_1 = v_1, f_2 = v_2, \ldots, f_n = v_n \rangle$
- Consider
  - $P(s_m \mid \langle f_1 = v_1, f_2 = v_2, \ldots, f_n = v_n \rangle)$ for each class sm
- Choose the class with the largest value, in symbols

$$\arg\max_{s_m \in S} P(s_m \mid \langle f_1 = v_1, f_2 = v_2, \ldots, f_n = v_n \rangle)$$

- i.e. choose the class for which the observations are most likely

Naive Bayes: Model

- Bayes formula
  - $P\left(s_m \mid \left\langle f_1 = v_1, f_2 = v_2, \ldots, f_n = v_n \right\rangle\right) = \dfrac{P\left(\left\langle f_1 = v_1, f_2 = v_2, \ldots, f_n = v_n \right\rangle \mid s_m\right) P(s_m)}{P\left(\left\langle f_1 = v_1, f_2 = v_2, \ldots, f_n = v_n \right\rangle\right)}$
- Sparse data, we may not even have seen
  - $\left\langle f_1 = v_1, f_2 = v_2, \ldots, f_n = v_n \right\rangle$
- We assume (wrongly) independence
  - $P\left(\left\langle f_1 = v_1, f_2 = v_2, \ldots, f_n = v_n \right\rangle \mid s_m\right) \approx \prod_{i=1}^{n} P\left(f_i = v_i \mid s_m\right)$
- Putting together
  - $\underset{s_m \in S}{\arg\max}\, P\left(s_m \mid \left\langle f_1 = v_1, f_2 = v_2, \ldots, f_n = v_n \right\rangle\right) \approx \underset{s_m \in S}{\arg\max}\, P(s_m) \prod_{i=1}^{n} P\left(f_i = v_i \mid s_m\right)$

Naive Bayes: Calculation

- $$\arg\max_{s_m \in S} P\left(s_m \mid \left\langle f_1 = v_1, f_2 = v_2, \ldots, f_n = v_n \right\rangle\right) \approx \arg\max_{s_m \in S} P(s_m) \prod_{i=1}^{n} P\left(f_i = v_i \mid s_m\right)$$

- For calculations
  - avoid underflow, use logarithms
  - $$\arg\max_{s_m \in S} P(s_m) \prod_{i=1}^{n} P\left(f_i = v_i \mid s_m\right) =$$
    $$\arg\max_{s_m \in S} \left(\log \left(P(s_m) \prod_{i=1}^{n} P\left(f_i = v_i \mid s_m\right) \right) \right)$$
    $$= \arg\max_{s_m \in S} \left(\log(P(s_m)) + \sum_{i=1}^{n} \log(P\left(f_i = v_i \mid s_m\right)) \right)$$

Naive Bayes, Training 1

- Maximum Likelihood
- $$\hat{P}(s_m) = \frac{C(s_m, o)}{C(o)}$$
  - where $C(s_m, o)$ are the number of occurrences of objects o in class s_m
- Observe what we are doing in statistical terms:
  - We want to estimate the true probability $P(s_m)$ from a set of observations
  - This is similar to estimating properties (parameters) of a population from a sample.

Naive Bayes (Bernoulli): Training 2

- Maximum Likelihood
- $$\hat{P}(f_i = v_i \mid s_m) = \frac{C(f_i = v_i, s_m)}{C(s_m)}$$
  - where $C(f_i=v_i, s_m)$ is the number of occurrences of objects o
    - where the object o belongs to class s_m
    - and the feature f_i takes the value v_i
  - $C(s_m)$ is the number of occurrences belonging to class s_m

The two models

- Bernoulli
  - the standard form of NB
  - NLTK book, Sec. 6.1, 6.2, 6.5
  - Jurafsky and Martin, 2.ed, sec. 20.2, WSD
- Multinomial model
  - For text classification
  - Related to n-gram models
  - Jurafsky and Martin, 3.ed, sec. 7.1, Sentiment analysis
- Both
  - Manning, Raghavan, Schütze, *Introduction to Information Retrieval*, Sec. 13.0-13.3