Applied Machine Learning
Lecture 1: Introduction

UNIVERSITY OF
GOTHENBURG

CHALMERS

Richard Johansson

January 16, 2018

welcome to the course!

- machine learning is getting increasingly popular among students
  - our courses are full!
  - many thesis projects apply ML
- ...and in industry
  - many companies come to us looking for students
  - joint research projects

why the fuss?

- media exposure; some impressive recent results
- snowball effect: everyone wants to do ML
- more data available
- lower barriers to entry: ML software is becoming user-friendly
- ML is more efficient because of improvements in hardware

topics covered in the course

- the usual "zoo": a selection of machine learning models
  - what's the idea behind them?
  - how are they implemented? (at least on a high level)
  - what are the use cases?
  - how can we apply them practically?
- but hopefully also the "real-world context":
  - extended "messy" practical assignments requiring that you think of what you're doing
  - (probably) 2 invited talks from industry
  - ethical and legal issues, interpretability

overview

practical issues about the course

basic ideas in machine learning

example of a learning algorithm: decision tree learning

machine learning libraries in Python

taxonomy of machine learning methods and use cases

course webpage

- the official course webpage is the **GUL page**
- (google "DIT865 GUL")

structure of teaching

- video lectures: mainly for theory
  - please watch the videos before each exercise session!
- lecture / exercise sessions (Tuesdays and Fridays)
  - some theory and introduction to ML software
  - interactive coding
  - solving exercises in groups
  - (tentatively) two industrial guest lectures
- lab sessions: you work on your assignments
  - please go to the 13-15 or the 15-17 session

assignments

- warmup exercise: quick tour of the scikit-learn library
- four compulsory **assignments**:
  1. "mini-project" where you solve a supervised learning task
  2. implement a classification algorithm
  3. neural network design
  4. written essay on ethics in ML
- please refer to the course PM for details about grading
- we will use the **Python** programming language
  - please ask for permission if you prefer to use something else

literature

- the main course book is *A Course in Machine Learning* by Hal Daumé III: `http://ciml.info`
- and additional papers to read for some topics
- example code will be posted on the course page

written exam on March 15

- a first part about basic concepts: you need to answer most of these questions correctly to pass
- a second part that requires more insight: answer these questions for a higher grade

overview

practical issues about the course

basic ideas in machine learning

example of a learning algorithm: decision tree learning

machine learning libraries in Python

taxonomy of machine learning methods and use cases

basic ideas

- given some object, make a **prediction**
  - is this patient diabetic?
  - is the sentiment of this movie review positive?
  - does this image contain a cat?
  - what will be tomorrow's share value of this stock?
  - what are the phonemes contained in this speech signal?

basic ideas

- given some object, make a **prediction**
  - is this patient diabetic?
  - is the sentiment of this movie review positive?
  - does this image contain a cat?
  - what will be tomorrow's share value of this stock?
  - what are the phonemes contained in this speech signal?
- the goal of machine learning is to build the prediction functions by **observing data**

why machine learning?

why would we want to “learn” the function from data instead of just implementing it?

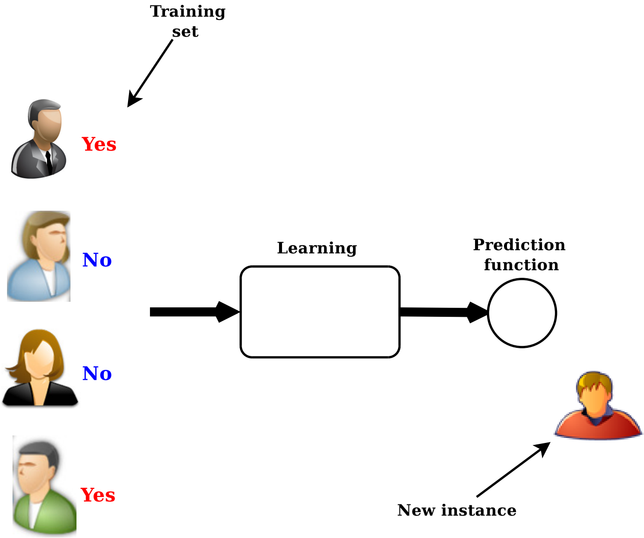
- usually **because we don’t really know** how to write down the function by hand
  - speech recognition
  - image classification
  - machine translation
  - ...
- might not be necessary for **limited** tasks where we **know**
- what is more expensive in your case? knowledge or data?

don't forget your domain expertise!

machine learning automatizes some tasks, but we still need our brains:

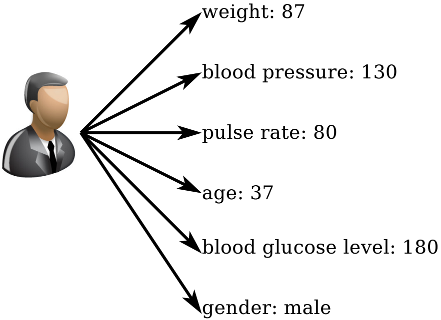
- **defining** the tasks, terminology, evaluation metrics
- **annotating** training and testing data
- having an intuition about which **features** may be useful can be crucial
  - in general, features are more important than the choice of learning algorithm
- **error analysis**
- defining **constraints** to guide the learner

learning from data



example: is the patient diabetic?

example: is the patient diabetic?



- in order to predict, we make some measurements of properties we believe will be useful
  - these are called the **features**

features: different views

- many learning algorithms operate on numerical vectors:

```
features = [ 1.5, -2, 3.8, 0, 9.12 ]
```

- more abstractly, we often represent the features as **attributes** with **values** (in Python, typically a dictionary)

```
features = { "gender":"male",
             "age":37,
             "blood_pressure":130, ... }
```

- sometimes, it's easier just to see the features as a list of e.g. words (**bag of words**)

```
features = [ "here", "are", "some", "words",
             "in", "a", "document" ]
```

basic ML methodology: evaluation

- select an **evaluation procedure** (a "metric") such as
  - **classification accuracy**: proportion correct classifications?
  - **mean squared error** often used in regression
- apply your model to a held-out **test set** and evaluate
  - the test set must be different from the training set
  - also: don't optimize on the test set; use a development set or cross-validation!

overview

practical issues about the course

basic ideas in machine learning

example of a learning algorithm: decision tree learning

machine learning libraries in Python

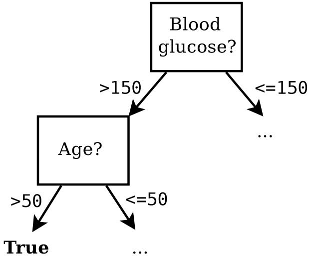
taxonomy of machine learning methods and use cases

classifiers as rule systems

- assume that we're building the prediction function by hand
- how would it look?
- probably, you would start writing rules like this:
  - IF the blood glucose level > 150, THEN
    - IF the age > 50, THEN return True
    - ELSE ...
    - ...
- a human would construct such a rule system by trial and error
- could this kind of rule system be learned **automatically**?

decision tree classifiers

- a **decision tree** is a tree where
  - the internal nodes represent how we choose based on a feature
  - the leaves represent the return value of the classifier
- like the example we had previously:
  - IF the blood glucose level > 150, THEN
    - IF the age > 50, THEN return True
    - ELSE ...
    - ...



general idea for learning a tree

- it should make few errors on the training set
- and an Occam's razor intuition: we'd like a small tree
- however, finding the smallest tree is a complex computational problem
  - it is NP-hard
- instead, we'll look at an algorithm that works top-down by selecting the "most useful feature"
- the basic approach is called the **ID3** algorithm
  - see e.g. Daumé III's book or `http://en.wikipedia.org/wiki/ID3_algorithm`

greedy decision tree learning (pseudocode)

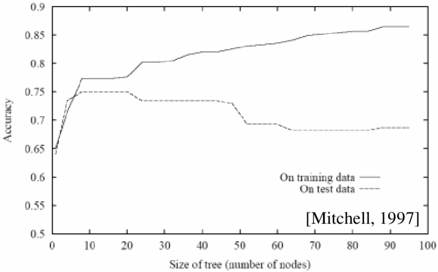
def TrainDecisionTree(T)
    if T is unambiguous
        return a leaf with the class of the examples in T
    if T has no features
        return a leaf with the majority class of T
    $F \leftarrow$ the "*most useful feature*" in T
    for each possible value f_i of F
        $T_i \leftarrow$ the subset of T where $F = f_i$
        remove F from T_i
        $\text{tree}_i \leftarrow$ TrainDecisionTree(T_i)
    return a tree node that splits on F,
        where f_i is connected to the subtree tree_i

how to select the "most useful feature"?

- there are many rules of thumb to select the most useful feature
  - idea: a feature is good if the subsets T_i are unambiguous
- in Daumé III's book, he uses a simple score to rank the features:
  - for each subset T_i, compute the frequency of its majority class
  - sum the majority class frequencies
- however, the most well-known ranking measure is the **information gain**
  - this measures the reduction of entropy (statistical uncertainty) we get by considering the feature

problems with the naive approach

- ID3 and similar decision tree learning algorithms often have troubles with large, noisy datasets
- often, performance decreases with training set size!



- can be improved by using a separate development set:
  - **prune** the tree by removing the nodes that don't seem to matter for accuracy on the development set

overview

practical issues about the course

basic ideas in machine learning

example of a learning algorithm: decision tree learning

machine learning libraries in Python

taxonomy of machine learning methods and use cases

machine learning software: a small sample

- general-purpose software, large collections of algorithms:
  - **scikit-learn**: `http://scikit-learn.org`
    - Python library – will be used in this course
  - Weka: `http://www.cs.waikato.ac.nz/ml/weka`
    - Java library with nice user interface
- special-purpose software, small collections of algorithms:
  - LibSVM/LibLinear for support vector machines
  - **Keras**, PyTorch, TensorFlow, Caffe for neural networks
  - ...
- large-scale learning in distributed architectures:
  - Spark MLLib

scikit-learn toy example: a simple training set

```
# training set: the features
X = [{'city':'Gothenburg', 'month':'July'},
     {'city':'Gothenburg', 'month':'December'},
     {'city':'Paris', 'month':'July'},
     {'city':'Paris', 'month':'December'}]

# training set: the gold-standard outputs
Y = ['rain', 'rain', 'sun', 'rain']
```

scikit-learn toy example: training a classifier

```
from sklearn.feature_extraction import DictVectorizer
from sklearn.svm import LinearSVC
from sklearn.pipeline import make_pipeline
import pickle

pipeline = make_pipeline( DictVectorizer(), LinearSVC() )

# train the classifier
pipeline.fit(X, Y)

# optionally: save the classifier to a file...
with open('weather.classifier', 'wb') as f:
    pickle.dump(pipeline, f)
```

explanation of the code: `DictVectorizer`

- internally, the features used by scikit-learn's classifiers are numbers, not strings
- a Vectorizer converts the strings into numbers – more about this in the next lecture!
- rule of thumb:
  - use a `DictVectorizer` for attribute–value features
  - use a `CountVectorizer` or `TfidfVectorizer` for bag-of-words features

explanation of the code: `LinearSVC`

- `LinearSVC` is the actual classifier we're using
  - this is called a **linear support vector machine**
  - more about this in lecture 3
- use a decision tree instead:

```
from sklearn.tree import DecisionTreeClassifier
...
pipeline = Pipeline( DictVectorizer(),
                     DecisionTreeClassifier() )
```

- perceptron:

```
from sklearn.linear_model import Perceptron
...
pipeline = Pipeline( DictVectorizer(), Perceptron() )
```

explanation of the code: `Pipeline` and `fit`

- in scikit-learn, preprocessing steps and classifiers are often combined into a `Pipeline`
  - in our case, a `DictVectorizer` and a `LinearSVC`
- the whole `Pipeline` is trained by calling the method `fit`
  - which will in turn call `fit` on all the parts of the `Pipeline`

toy example: making new predictions and evaluating

```
from sklearn.metrics import accuracy_score

Xtest = [{'city':'Gothenburg', 'month':'June'},
         {'city':'Gothenburg', 'month':'November'},
         {'city':'Paris', 'month':'June'},
         {'city':'Paris', 'month':'November'}]

Ytest = ['rain', 'rain', 'sun', 'rain']

# classify all the test instances
guesses = pipeline.predict(Xtest)

# compute the classification accuracy
print(accuracy_score(Ytest, guesses))
```

a note on efficiency

- Python is a nice language for programmers but not always the most efficient
- in scikit-learn, many functions are implemented in faster languages (e.g. C) and use specialized math libraries
- so in many cases, it is much faster to call the library once than many times:

```
import time
t0 = time.time()
guesses1 = pipeline.predict(Xtest)
t1 = time.time()
guesses2 = []
for x in Xtest:
    guess = pipeline.predict(x)
    guesses2.append(guess)
t2 = time.time()

print(t1-t0)
print(t2-t1)
```

- result: 0.29 sec and 45 sec

some other practical functions

- making a training/test split:

```
from sklearn.cross_validation import train_test_split

train_files, dev_files = train_test_split(td_files,
                                           train_size=0.8,
                                           random_state=0)
```

- evaluation, e.g. accuracy, precision, recall, F-score:

```
from sklearn.metrics import f1_score

print(f1_score(Y_eval, Y_out))
```

- cross-validation over the training set:

```
from sklearn.cross_validation import cross_validate

cv_results = cross_validate(pipeline, X, Y)
```

overview

practical issues about the course

basic ideas in machine learning

example of a learning algorithm: decision tree learning

machine learning libraries in Python

taxonomy of machine learning methods and use cases

how can we classify machine learning methods?

- **output**: what are we predicting?
- **supervision**: what type of data? how do we use it?
- **representation**: how do we describe our model?
- **induction**: how are models selected?

types of machine learning problems: what are we predicting?

- **classification**: learning to output a category label
  - spam/non-spam; positive/negative; ...
- **regression**: learning to guess a number
  - value of a share; number of stars in a review; ...
- **structured prediction**: learning to build some structure
  - speech segmentation; machine translation; ...
- **ranking**: learn to order a set of items
  - search engines
- **reinforcement learning**: learning to act in an environment
  - dialogue systems; playing games; autonomous vehicles; ...

types of supervision (1): supervised

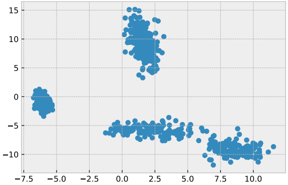
- in **supervised** learning, we have a **labeled** training set consists of **input–output** pairs
- our goal is to learn to imitate this labeling

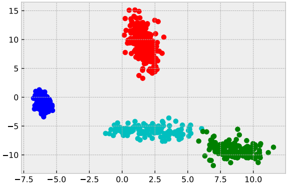
age	workclass	fnlwgt	education	education-num	marital-status	occupation	relationship	race	sex	capital-gain	capital-loss	hours-per-week	native-country	target
39	State-gov	77516	Bachelors	13	Never-married	Adm-clerical	Not-in-family	White	Male	2174	0	40	United-States	<=50K
50	Self-emp-not-inc	83311	Bachelors	13	Married-civ-spouse	Exec-managerial	Husband	White	Male	0	0	13	United-States	<=50K
38	Private	215646	HS-grad	9	Divorced	Handlers-cleaners	Not-in-family	White	Male	0	0	40	United-States	<=50K
53	Private	234721	11th	7	Married-civ-spouse	Handlers-cleaners	Husband	Black	Male	0	0	40	United-States	<=50K
28	Private	338409	Bachelors	13	Married-civ-spouse	Prof-specialty	Wife	Black	Female	0	0	40	Cuba	<=50K

	F1	F2	F3	F4	F5	F6	F7	F8	F9	RMSD
0	13558.30	4305.35	0.31754	162.1730	1.872791e+06	215.3590	4287.87	102	27.0302	17.284
1	6191.96	1623.16	0.26213	53.3894	8.034467e+05	87.2024	3328.91	39	38.5468	6.021
2	7725.98	1726.28	0.22343	67.2887	1.075648e+06	81.7913	2981.04	29	38.8119	9.275
3	8424.58	2368.25	0.28111	67.8325	1.210472e+06	109.4390	3248.22	70	39.0651	15.851
4	7460.84	1736.94	0.23280	52.4123	1.021020e+06	94.5234	2814.42	41	39.9147	7.962

types of supervision (2): **unsupervised**

- in **unsupervised** learning, we are given a set of "unorganized" data
- our goal is to discover some structure in the data





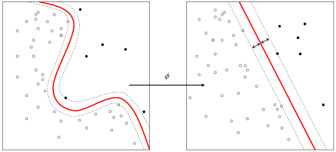
types of supervision (3): variations...

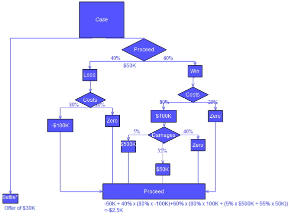
- **semisupervised** learning:
  - a small set of labeled examples plus a larger unlabeled set
- **active** learning:
  - the learning algorithm can ask for additional labeling of targeted examples
- **multitask** learning:
  - learning from closely related tasks

representation of the prediction function

we may represent our prediction function in different ways:

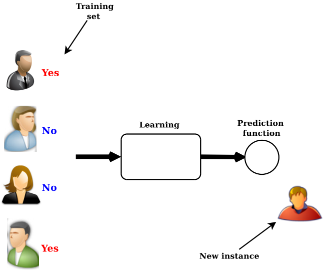
- **numerical** models:
  - weight vectors, probability tables
  - networked models
- **rule-based** models:
  - decision trees
  - rules expressed using logic





what goes on when we "learn"?

- the learning algorithm observes the examples in the training set
- it tries to find common patterns that explain the data: it **generalizes** so that we can make predictions for new examples



- how this is done depends on what algorithm we are using

principles of induction: how do we select "good" models?

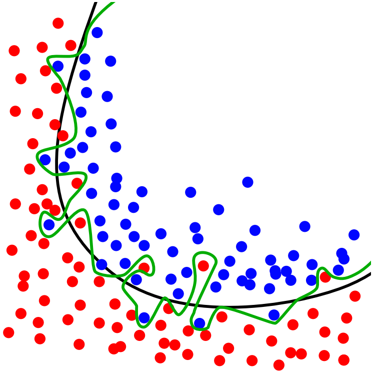
- **hypothesis space**: the set of all possible outputs of a learning algorithm
  - for decision tree learners: The set of possible trees
  - for linear separators: the set of all lines in the plane / hyperplanes in a vector space
- "learning" = searching the hypothesis space
- how do we know what hypothesis to look for?

a fundamental tradeoff in machine learning

- **goodness of fit**: the learned classifier should be able to capture the information in the training set
  - e.g. correctly classify the examples in the training data
- **regularization**: the classifier should be simple

why would we prefer "simple" hypotheses?

“overfitting” and “underfitting”: the bias–variance tradeoff



[Source: Wikipedia]

up next

- Thursday: lab session for the noncompulsory exercise
- topic of Friday's discussion: linear classifiers and regressors
- please prepare by watching the videos