The SHOGUN Machine Learning Toolbox

(and its python interface)

Sören Sonnenburg[1,2], Gunnar Rätsch[2],Sebastian Henschel[2],Christian Widmer[2],Jonas Behr[2],Alexander Zien[2],Fabio de Bona[2],Alexander Binder[1],Christian Gehl[1], and Vojtech Franc[3]

[1] Berlin Institute of Technology, Germany
[2] Friedrich Miescher Laboratory, Max Planck Society, Germany
[3] Center for Machine Perception, Czech Republic

Outline

Who Am I?

About Me

- 1997-2002 studied Computer Science
- 2002-2009 doing ML & Bioinformatics research at Fraunhofer Institute FIRST and Max Planck Society since 2002
- 2008 PhD: Machine Learning for Genomic Sequence Analysis
- 2009- Researcher at Berlin Institute of Technology

Open Source Involvement

- Debian Developer `http://www.debian.org`
- Machine Learning OSS `http://mloss.org`
- Machine Learning Data `http://mldata.org`
- **Main author of SHOGUN - this talk**

More about me `http://sonnenburgs.de/soeren`

What is Machine Learning and what can it do for you?

What is ML?

AIM: Learning from empirical data!

Applications

- speech and handwriting recognition
- search engines, natural language processing
- medical diagnosis, bioinformatics, chemoinformatics
- detecting credit card fraud
- computer vision, object recognition
- stock market analysis
- network security, intrusion detection
- brain-machine interfaces
- ...

What is Machine Learning and what can it do for you?

What is ML?

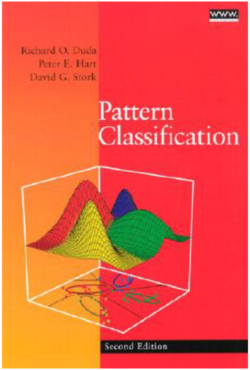
AIM: Learning from empirical data!

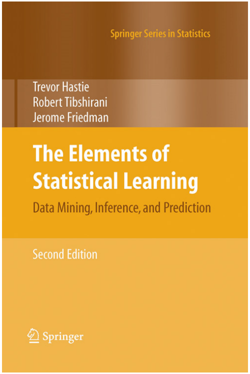
Applications

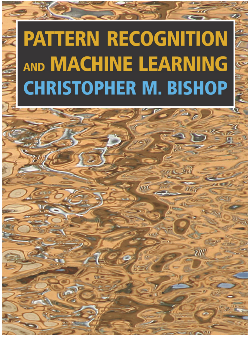
- speech and handwriting recognition
- search engines, natural language processing
- medical diagnosis, bioinformatics, chemoinformatics
- detecting credit card fraud
- computer vision, object recognition
- stock market analysis
- network security, intrusion detection
- brain-machine interfaces

...

Books on Machine Learning







... **and many more** ...

A more formal definition

Example $\mathbf{x}_i \in \mathcal{X}$, for example, text document

Label $y_i \in \mathcal{Y}$, for example, whether the document is Spam

Training Data Data consisting of examples and associated labels which are used for training the machine

Testing Data Data consisting only of examples used for generating predictions

Predictions Output of the trained machine

A more formal definition

Example $\mathbf{x}_i \in \mathcal{X}$, for example, text document

Label $y_i \in \mathcal{Y}$, for example, whether the document is Spam

Training Data Data consisting of examples and associated labels which are used for training the machine

Testing Data Data consisting only of examples used for generating predictions

Predictions Output of the trained machine

Machine Learning: Main Tasks

Supervised Learning

We have both, input and labels, for each example. The aim is to learn about the pattern between input and labels. (The input is sometimes also called example.)

Unsupervised Learning

We do not have labels for the examples, but wish to discover the underlying structure of the data.

Reinforcement Learning

How an autonomous agent that senses and acts in its environment can learn to choose optimal actions to achieve its goals.

Estimators

Basic Notion

We want to estimate the relationship between the examples $\mathbf{x}_i$ and the associated label y_i.

Formally

We want to choose an estimator

$$f : \mathcal{X} \to \mathcal{Y}.$$

Intuition

We would like a function f which correctly predicts the label y for a given example $\mathbf{x}$.

Question

How do we measure how well we are doing?

Loss Function

Basic Notion

We characterize the quality of an estimator by a loss function.

Formally

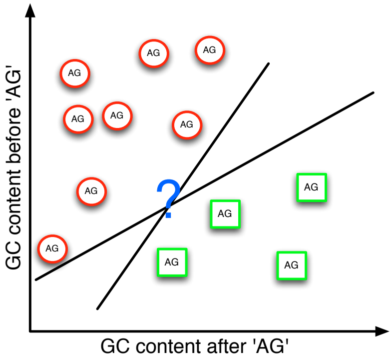
We define a loss function as

$$\ell(f(\mathbf{x}_i), y_i) : \mathcal{Y} \times \mathcal{Y} \to \mathbb{R}_+.$$

Intuition

For a given label y_i and a given prediction $f(\mathbf{x}_i)$, we want a positive value telling us how much error there is.

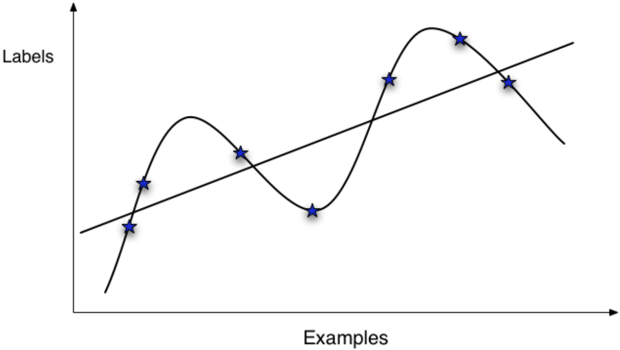
Classification



In binary classification ($\mathcal{Y} = \{-1, +1\}$), we one may use the 0/1-*loss function*:

$$\ell(f(\mathbf{x}_i), y_i) = \begin{cases} 0 & \text{if } f(\mathbf{x}_i) = y_i \\ 1 & \text{if } f(\mathbf{x}_i) \neq y_i \end{cases}$$

Regression



In regression ($\mathcal{Y} = \mathbb{R}$), one often uses the *square loss function*:

$$\ell(f(\mathbf{x}_i), y_i) = (f(\mathbf{x}_i) - y_i)^2.$$

Expected vs. Empirical Risk

Expected Risk

This is the average loss on *unseen examples*. We would like to have it as small as possible, but it is hard to compute.

Empirical Risk

We can compute the *average on training data*. We define the empirical risk to be:

$$\mathbf{R}_{emp}(f, X, Y) = \frac{1}{n}\sum_{i=1}^{n}\ell(f(\mathbf{x}_i), y_i).$$

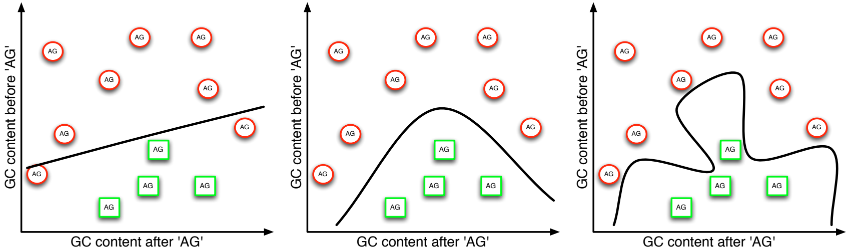
Basic Notion

Instead of minimizing the expected risk, we minimize the empirical risk. This is called empirical risk minimization.

Question

How do we know that our estimator will perform well on unseen data?

Simple vs. Complex Functions



Which function is preferable?

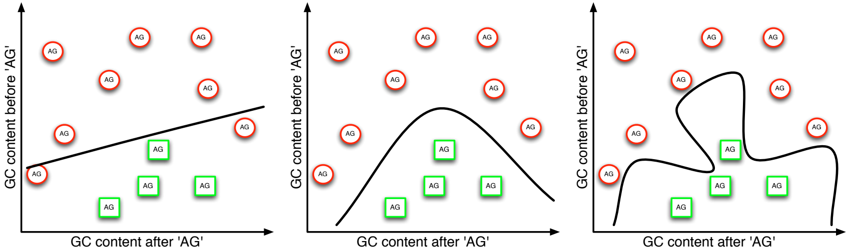
Occam's razor (a.k.a. Occam's Law of Parsimony):
(William of Occam, 14th century)

"Entities should not be multiplied beyond necessity"

("Do not make the hypothesis more complex than necessary")

[http://www.franciscans.org]

Simple vs. Complex Functions



Which function is preferable?

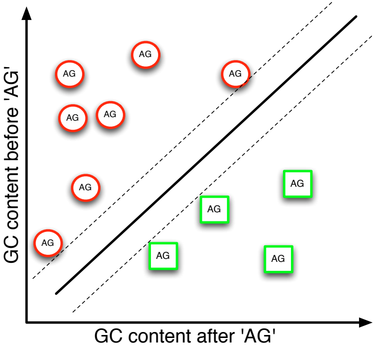
Occam's razor (a.k.a. Occam's Law of Parsimony):
(William of Occam, 14th century)

"Entities should not be multiplied beyond necessity"

("Do not make the hypothesis more complex than necessary")

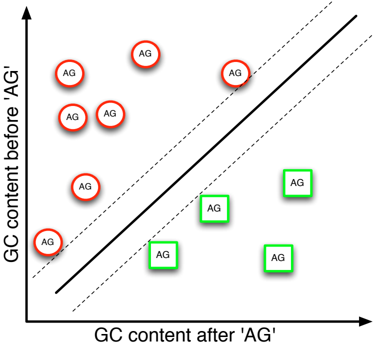
[http://www.franciscans.org]

Special Case: Complexity of Hyperplanes



What is the complexity of a hyperplane classifier?

Special Case: Complexity of Hyperplanes



What is the complexity of a hyperplane classifier?

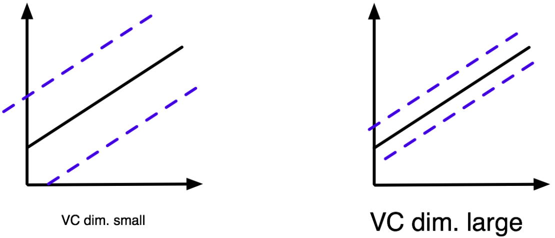
Vladimir Vapnik and Alexey Chervonenkis: Vapnik-Chervonenkis (VC) dimension

(VapChe71,Vap95)

[http://tinyurl.com/cl8jo9,http://tinyurl.com/d71mux]

Larger Margin $\Rightarrow$ Less Complex

Support Vector Machines (SVMs)



Maximum Margin $\Rightarrow$ Minimum Complexity

Minimize complexity by maximizing margin (irrespective of the dimension of the space).

Useful Idea:

Find the hyperplane that classifies all points correctly, while maximizing the margin (=SVMs).

Summary of Empirical Inference

Learn function $f : \mathcal{X} \to \mathcal{Y}$ given N labeled examples $(\mathbf{x}_i, y_i) \in \mathcal{X} \times \mathcal{Y}$.

Three important ingredients:

- Model $f_{\boldsymbol{\theta}}$ parametrized with some parameters $\boldsymbol{\theta} \in \Theta$
- Loss function $\ell(f(\mathbf{x}), y)$ measuring the "deviation" between predictions $f(\mathbf{x})$ and the label y
- Complexity term $P[f]$ defining model classes with limited complexity

Most algorithms find $\boldsymbol{\theta}$ in $f_{\boldsymbol{\theta}}$ by minimizing:

$$\boldsymbol{\theta}^* = \operatorname*{argmin}_{\boldsymbol{\theta} \in \Theta} \left(\underbrace{\sum_{i=1}^{N} \ell(f_{\boldsymbol{\theta}}(\mathbf{x}_i), y_i)}_{\text{Emprical error}} + \overbrace{C}^{\text{Regularization parameter}} \underbrace{P[f_{\boldsymbol{\theta}}]}_{\text{Complexity term}} \right) \quad \text{for given } C$$

Summary of Empirical Inference

Learn function $f : \mathcal{X} \to \mathcal{Y}$ given N labeled examples $(\mathbf{x}_i, y_i) \in \mathcal{X} \times \mathcal{Y}$.

Three important ingredients:

- Model $f_{\boldsymbol{\theta}}$ parametrized with some parameters $\boldsymbol{\theta} \in \Theta$
- Loss function $\ell(f(\mathbf{x}), y)$ measuring the “deviation” between predictions $f(\mathbf{x})$ and the label y
- Complexity term $P[f]$ defining model classes with limited complexity

Most algorithms find $\boldsymbol{\theta}$ in $f_{\boldsymbol{\theta}}$ by minimizing:

$$\boldsymbol{\theta}^* = \operatorname*{argmin}_{\boldsymbol{\theta}\in\Theta} \left(\underbrace{\sum_{i=1}^{N} \ell(f_{\boldsymbol{\theta}}(\mathbf{x}_i), y_i)}_{\text{Emprical error}} + \overbrace{C}^{\text{Regularization parameter}} \underbrace{P[f_{\boldsymbol{\theta}}]}_{\text{Complexity term}} \right) \quad \text{for given } C$$

Summary of Empirical Inference

Learn function $f : \mathcal{X} \to \mathcal{Y}$ given N labeled examples $(\mathbf{x}_i, y_i) \in \mathcal{X} \times \mathcal{Y}$.

Three important ingredients:

- **Model $f_{\boldsymbol{\theta}}$** parametrized with some parameters $\boldsymbol{\theta} \in \Theta$
- **Loss function $\ell(f(\mathbf{x}), y)$** measuring the “deviation” between predictions $f(\mathbf{x})$ and the label y
- **Complexity term $P[f]$** defining model classes with limited complexity

Most algorithms find θ in f_θ by minimizing:

$$\boldsymbol{\theta}^* = \operatorname*{argmin}_{\boldsymbol{\theta}\in\Theta} \left(\underbrace{\sum_{i=1}^{N} \ell(f_{\boldsymbol{\theta}}(\mathbf{x}_i), y_i)}_{\text{Emprical error}} + \overbrace{C}^{\text{Regularization parameter}} \underbrace{P[f_{\boldsymbol{\theta}}]}_{\text{Complexity term}} \right) \quad \text{for given } C$$

Summary of Empirical Inference

Learn function $f : \mathcal{X} \to \mathcal{Y}$ given N labeled examples $(\mathbf{x}_i, y_i) \in \mathcal{X} \times \mathcal{Y}$.

Three important ingredients:

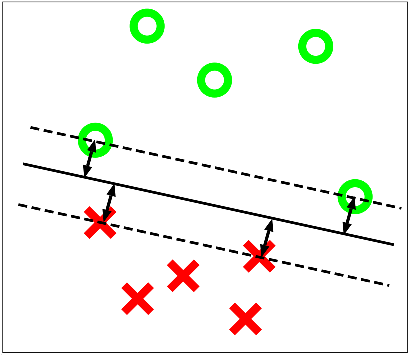
- Model $f_{\boldsymbol{\theta}}$ parametrized with some parameters $\boldsymbol{\theta} \in \Theta$
- Loss function $\ell(f(\mathbf{x}), y)$ measuring the "deviation" between predictions $f(\mathbf{x})$ and the label y
- Complexity term $P[f]$ defining model classes with limited complexity

Most algorithms find $\boldsymbol{\theta}$ in $f_{\boldsymbol{\theta}}$ by minimizing:

$$\boldsymbol{\theta}^* = \underset{\boldsymbol{\theta}\in\Theta}{\operatorname{argmin}} \left(\underbrace{\sum_{i=1}^{N} \ell(f_{\boldsymbol{\theta}}(\mathbf{x}_i), y_i)}_{\text{Emprical error}} + \overbrace{C}^{\text{Regularization parameter}} \underbrace{P[f_{\boldsymbol{\theta}}]}_{\text{Complexity term}} \right) \quad \text{for given } C$$

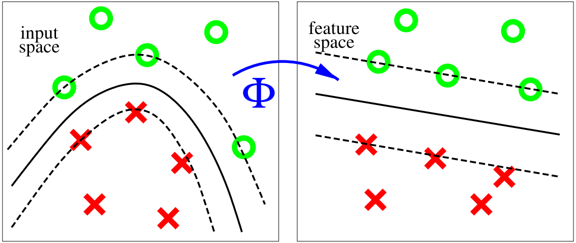
Support Vector Machine (SVMs)

- Given: Points $\mathbf{x}_i \in \mathcal{X}$ $(i = 1, \ldots, N)$ with labels $y_i \in \{-1, +1\}$
- Task: Find hyperplane that maximizes margin



Decision function $f(\mathbf{x}) = \mathbf{w} \cdot \mathbf{x} + b$

SVM with Kernels



- SVM decision function in kernel feature space:

$$f(\mathbf{x}) = \sum_{i=1}^{N} y_i \alpha_i \underbrace{\Phi(\mathbf{x}) \cdot \Phi(\mathbf{x}_i)}_{=k(\mathbf{x},\mathbf{x}_i)} + b \qquad (1)$$

- Training: Find parameters α
- Corresponds to solving quadratic optimization problem (QP)

Measuring Performance in Practice

What to do in practice

Split the data into training and validation sets; use error on validation set as estimate of the expected error

A. Cross-validation

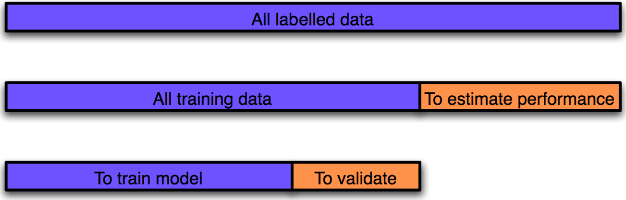
Split data into c disjoint parts; use each subset as validation set and rest as training set

B. Random splits

Randomly split data set into two parts, for example, 80% of data for training and 20% for validation; Repeat this many times

See, for instance, (DudHarSto01) for more details.

Model Selection



Do not train on the "test set"!

- Use subset of data for training
- From subset, further split to select model.

Model selection = Find best parameters

- Regularization parameter C.
- Other parameters (kernel,...)

SHOGUN Machine Learning Toolbox Overview I

History

1999	Initiated by S. Sonnenburg and G. Rätsch (SHOGUN)
2006	First public release (June)
2008	used in 3rd party code (PyVMPA)
Now	Several other contributors mostly from Berlin, Tübingen
	Debian, Ubuntu, MacPorts packaged, $>$ 1000 installations

Unified (large-scale) learning for various feature types and settings

Machine Learning Methods Overview

- Regression (Kernel Ridge Regression, SVR)
- Distributions (Hidden Markov models...)
- Performance Measures
- Clustering
- **Classification (focus: Support Vector Machines)**

SHOGUN Machine Learning Toolbox Overview I

History

1999	Initiated by S. Sonnenburg and G. Rätsch (SHOGUN)
2006	First public release (June)
2008	used in 3rd party code (PyVMPA)
Now	Several other contributors mostly from Berlin, Tübingen
	Debian, Ubuntu, MacPorts packaged, $>$ 1000 installations

Unified (large-scale) learning for various feature types and settings

Machine Learning Methods Overview

- Regression (Kernel Ridge Regression, SVR)
- Distributions (Hidden Markov models. . .)
- Performance Measures
- Clustering
- Classification (focus: Support Vector Machines)

SHOGUN Machine Learning Toolbox Overview I

History

1999	Initiated by S. Sonnenburg and G. Rätsch (SHOGUN)
2006	First public release (June)
2008	used in 3rd party code (PyVMPA)
Now	Several other contributors mostly from Berlin, Tübingen
	Debian, Ubuntu, MacPorts packaged, $>$ 1000 installations

Unified (large-scale) learning for various feature types and settings

Machine Learning Methods Overview

- Regression (Kernel Ridge Regression, SVR)
- Distributions (Hidden Markov models. . .)
- Performance Measures
- Clustering
- **Classification (focus: Support Vector Machines)**

SHOGUN Machine Learning Toolbox - Overview II

Focus: Large-Scale Learning with...

- 15 implementations of Support Vector Machines solvers
- 35 kernels (Focus on string-kernels for Bioinformatics)
- Multiple Kernel Learning
- Linear Methods

Implementation and Interfaces

- Implemented in C++ ($> 130,000$ lines of code)
- **Interfaces:** libshogun, python, octave, R, matlab, cmdline
- **Over 600 examples**
- Doxygen documentation
- Inline python documentation (e.g. `help(GaussianKernel)`)
- **Testsuite** ensuring that obvious bugs do not slip through

Installing SHOGUN

Steps to Install SHOGUN

- LINUX

  ```
  $ sudo apt-get install shogun-python-modular
  ```

- MacOSX

  ```
  $ sudo ports install shogun
  ```

- If that does not work or wrong OS
  download source code, untar and read INSTALL or follow
  `http://www.shogun-toolbox.org/doc/installation.html`

DON'T

Do not use the legacy static python interface (just called python).

Simple code example: SVM Training

A) Generate Toy Data

```
from numpy import concatenate as con
from numpy import ones,mean,sign
from numpy.random import randn

num=1000; dist=1; width=2.1; C=1.0

traindata_real=con((randn(2,num)-dist,
                    randn(2,num)+dist), axis=1)
testdata_real=con((randn(2,num)-dist,
                   randn(2,num)+dist), axis=1)
trainlab=con((-ones(num), ones(num)))
testlab=con((-ones(num), ones(num)))
```

Simple code example: SVM Training

B) Train and Apply SVM with SHOGUN

```
from shogun.Features import Labels,RealFeatures
from shogun.Kernel import GaussianKernel
from shogun.Classifier import LibSVM

feats_train=RealFeatures(traindata_real)
kernel=GaussianKernel(feats_train, feats_train, width)
labels=Labels(trainlab)
svm=LibSVM(C, kernel, labels)
svm.train()

out=svm.classify(RealFeatures(testdata_real)).get_labels()
testerr=mean(sign(out)!=testlab)
print testerr
```

Efficient Native Feature Representations

Input Features

- Dense Vectors/Matrices (SimpleFeatures)
  - `uint8_t`
  - ⋮
  - float64_t
- Sparse Vectors/Matrices (SparseFeatures)
  - `uint8_t`
  - ⋮
  - float64_t
- Variable Length Vectors/Matrices (StringFeatures)
  - `uint8_t`
  - ⋮
  - float64_t

$\Rightarrow$ **loading and saving as hdf5, ascii, binary all of numpy**

Interfaces (legacy static interface)

Interface Types

- Static Interfaces (single object of each type only)
- **NEW** Modular Interfaces (really object oriented, SWIG based)

Support for all Feature Types

- Dense, Sparse, Strings
- Possible by defining generic get/set functions, e.g.

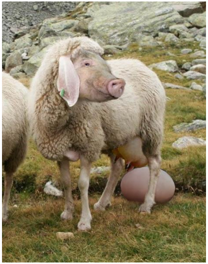
```
void set_int(int32_t scalar);
void set_real(float64_t scalar);
void set_bool(bool scalar);
void set_vector(float64_t* vector, int32_t len);
void set_matrix(float64_t* m, int32_t rws, int32_t cls);
...
```

⇒ set/get functions for access from python, R, octave, matlab

The Eierlegendewollmilchsau$^{\text{TM}}$ Interface

Embed Interface A from Interface B

- possible to run python code from octave
- possible to run octave code from python
- possible to run r code from python
- ...



Demo: Use matplotlib to plot functions from octave.

Modular Python SWIG based Interface

⇒ typemaps for numpy,scipy.sparse,files,lists of strings defined

Wrapping a C++ Object with swig (Kernel)

```
%{
#include <shogun/kernel/GaussianKernel.h>
%}
%rename(GaussianKernel) CGaussianKernel;
%include <shogun/kernel/GaussianKernel.h>
```

Wrapping a C++ Object with swig (Classifier)

```
%{
#include <shogun/classifier/svm/LibSVM.h>
%}
%rename(LibSVM) CLibSVM;
%include <shogun/classifier/svm/LibSVM.h>
```

Unique Features of SHOGUN I

Input Features

- possible to stack together features of arbitrary types (sparse, dense, string) via CombinedFeatures and DotFeatures
- chains of "preprocessors" (e.g. substracting the mean) can be attached to each feature object (on-the-fly pre-processing)

Kernels

- working with custom pre-computed kernels.
- possible to stack together kernels via CombinedKernel (weighted linear combination of a number of sub-kernels, not necessarily working on the same domain)
- kernel weighting can be learned using MKL
- Methods (e.g., SVMs) can be trained using unified interface

Unique Features of SHOGUN II

Large Scale

- multiprocessor parallelization (training with up to 10 million examples and kernels)
- implements COFFIN framework (dynamic feature / example generation; training on 200,000,000 dimensions and 50,000,000 examples)

Community Integration

- Documentation available, many many examples
- There is a Debian Package, MacOSX
- Mailing-List, open SVN repository

. . . and many more. . .

Application

Genomic Signals

- Transcription Start (Sonnenburg et al., 2006)
- Acceptor Splice Site (Sonnenburg et al., 2007)
- Donor Splice Site (Sonnenburg et al., 2007)
- Alternative Splicing (Rätsch et al., 2005)
- Transsplicing (Schweikert et al., 2009)
- Translation Initiation (Sonnenburg et al., 2008)

Genefinding

- Splice form recognition - mSplicer (Rätsch et al. 2008)
- Genefinding - mGene (Schweikert et al., 2009)

Demo I

Support Vector Classification

- Task: separate 2 clouds of gaussian distributed points in 2D

Simple code example: SVM Training

```
lab = Labels(labels)
train = RealFeatures(features)
gk = GaussianKernel(train, train, width)
svm = LibSVM(10.0, gk, lab)
svm.train()
```

Demo II

Support Vector Regression

- Task: learn a sine function

Simple code example: Support Vector Regression

```
lab = Labels(labels)
train = RealFeatures(features)
gk = GaussianKernel(train, train, width)
svm = LibSVR(10.0, gk, lab)
svm.train()
```

Demo III

Hidden Markov Model

- Task: 3 loaded dice are drawn 1000 times, find out when which dice was drawn

Clustering

- Task: find clustering of 3 clouds of gaussian distributed points in 2D

Summary

SHOGUN Machine Learning Toolbox

- Unified framework, for various interfaces
- Applicable to huge datasets (>**50 million** examples)
- Algorithms: HMM, LDA, LPM, Perceptron, SVM, SVR + many kernels, ...

Documentation, Examples, Source Code

- Implementation `http://www.shogun-toolbox.org`
- Documentation `http://www.shogun-toolbox.org/doc`
- More machine learning software `http://mloss.org`
- Machine Learning Data `http://mldata.org`

We need your help:

- Documentation, Examples, Testing, Extensions