CAP 5510: Introduction to Bioinformatics
CGS 5166: Bioinformatics Tools

Giri Narasimhan

ECS 254; Phone: x3748

giri@cis.fiu.edu

www.cis.fiu.edu/~giri/teach/BioinfS13.html

Machine Learning

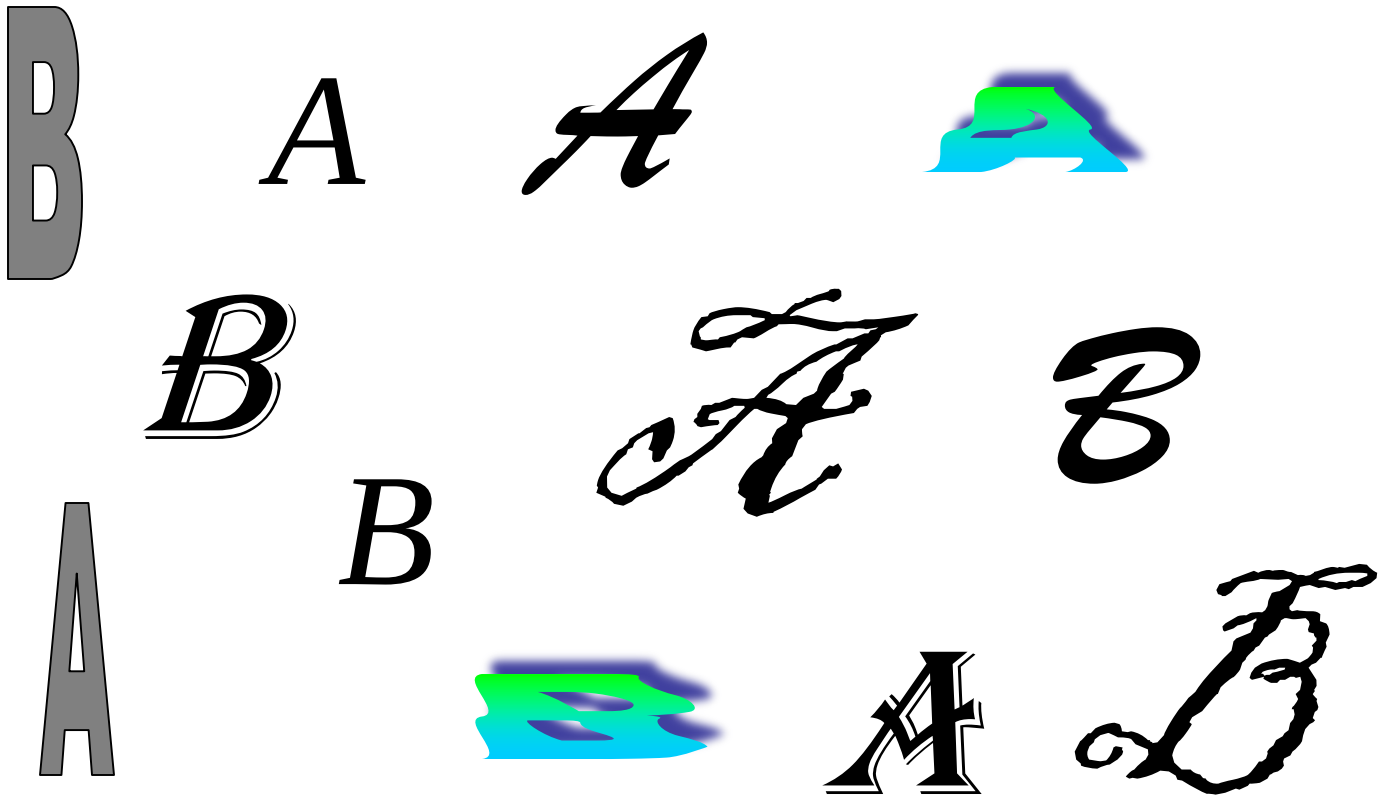
Machine Learning

- Human Endeavor
  - Data → Information → Knowledge
- Machine Learning
  - Automatically extracting information from data
- Types of Machine Learning
  - Unsupervised
    - Clustering
    - Pattern Discovery
  - Supervised
    - Learning
    - Classification

Support Vector Machines

- Supervised Statistical Learning Method for:
  - Classification
  - Regression
- Simplest Version:
  - Training: Present series of labeled examples (e.g., gene expressions of tumor vs. normal cells)
  - Prediction: Predict labels of new examples.

Learning Problems



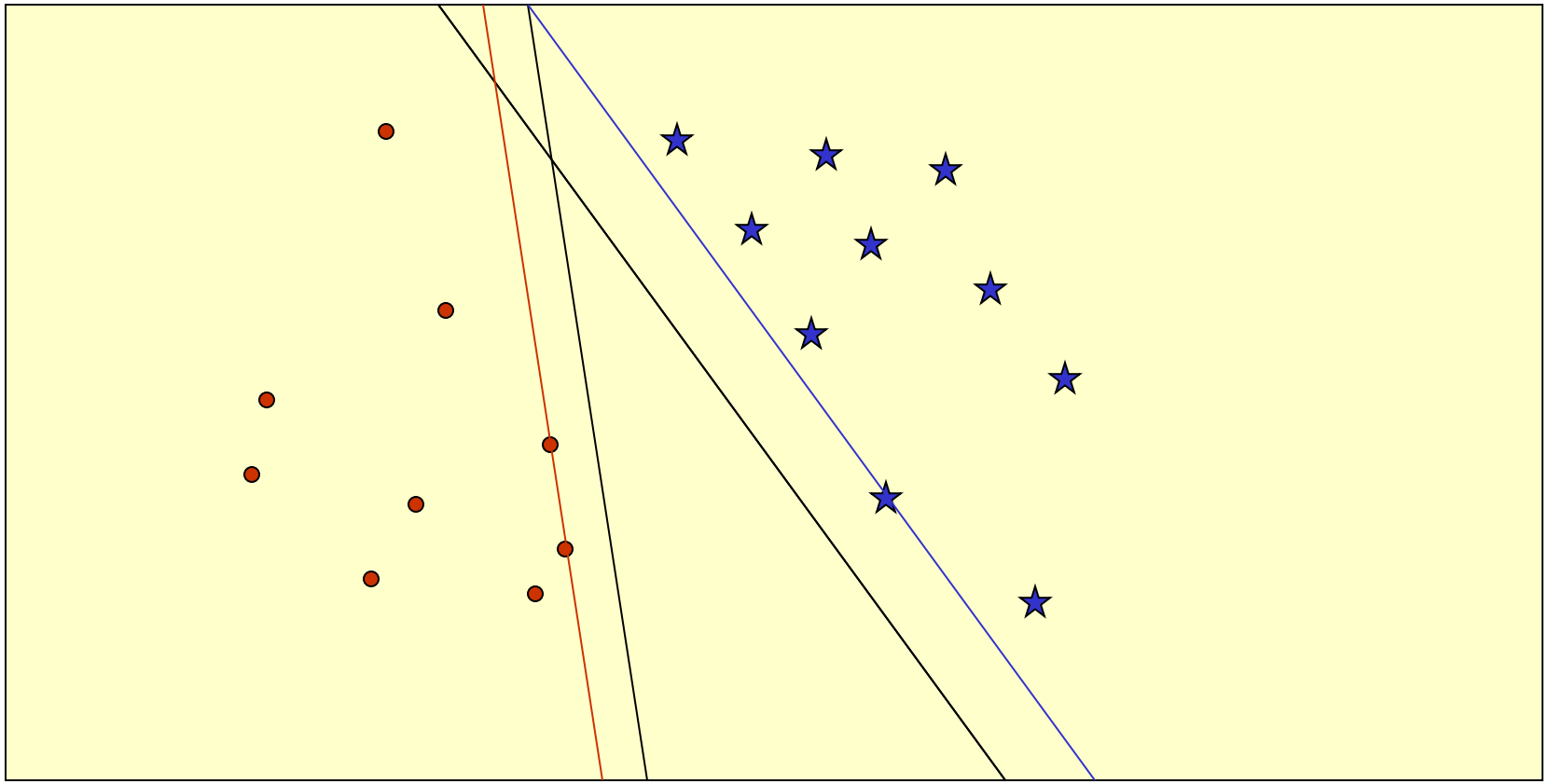
Learning Problems

- Binary Classification
- Multi-class classification
- Regression

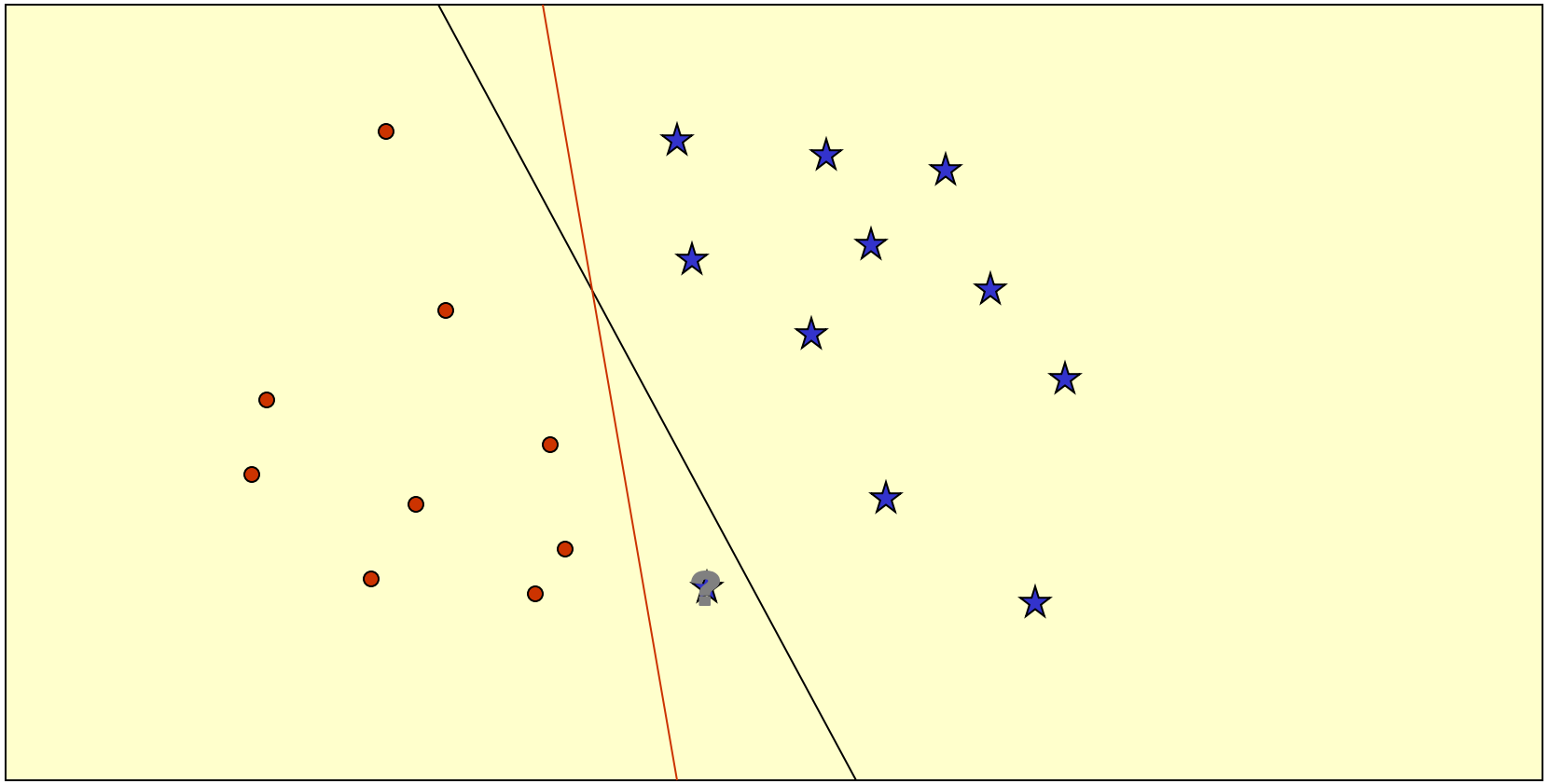
SVM – Binary Classification

- Partition feature space with a surface.
- Surface is implied by a subset of the training points (vectors) near it. These vectors are referred to as Support Vectors.
- Efficient with high-dimensional data.
- Solid statistical theory
- Subsume several other methods.

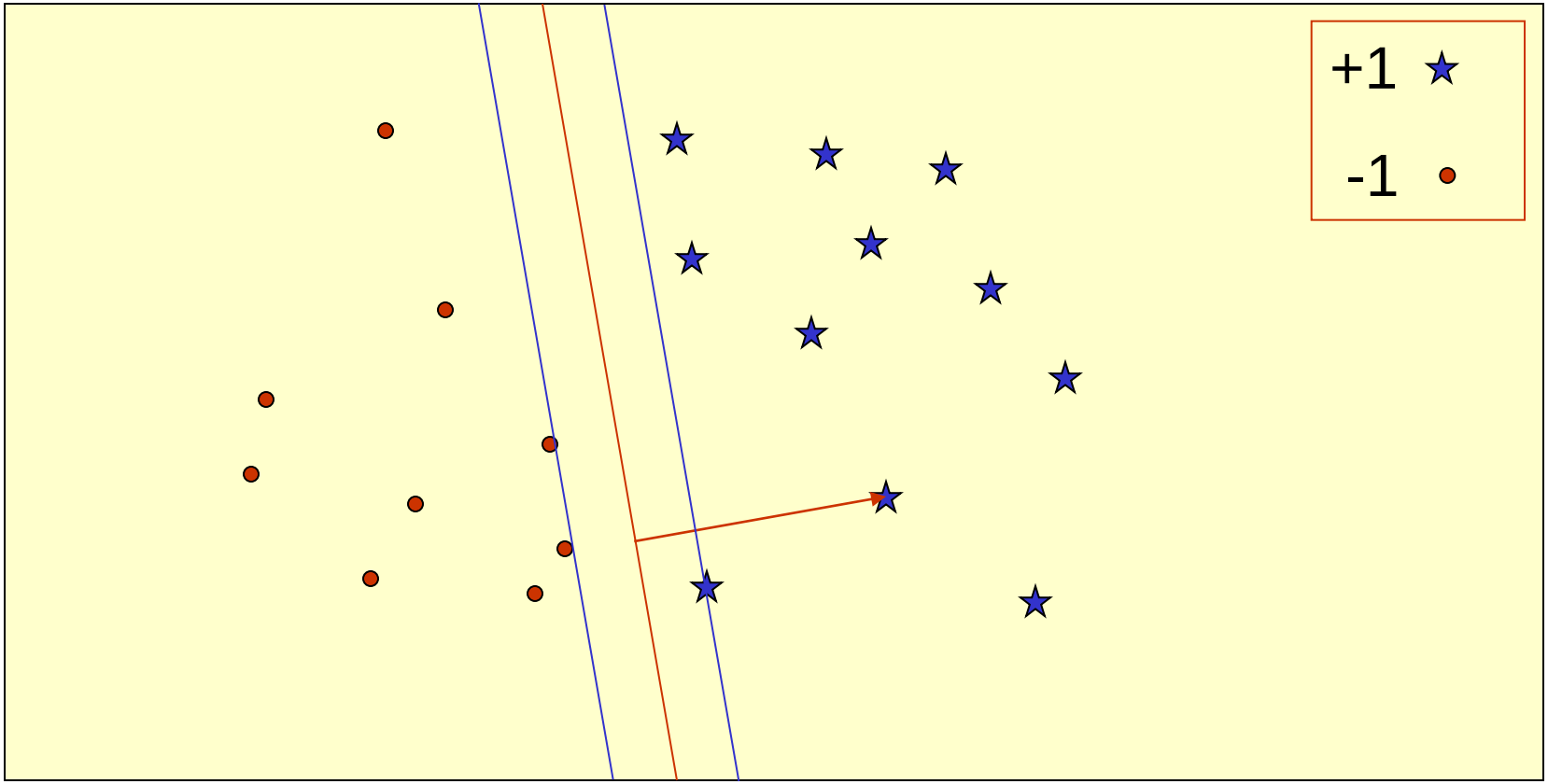
Classification of 2-D (Separable) data



Classification of (Separable) 2-D data

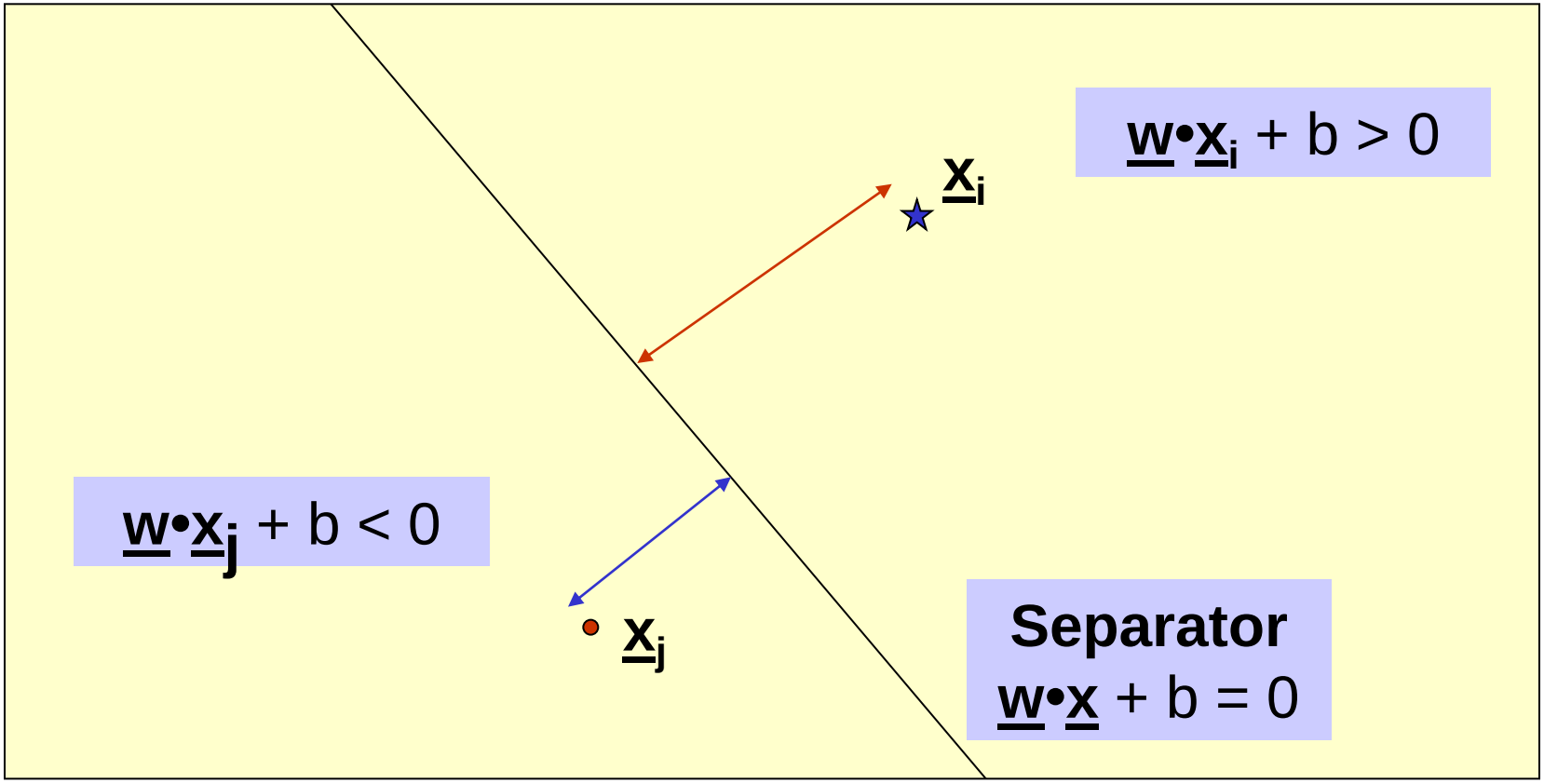


Classification of (Separable) 2-D data



- Margin of a point
- Margin of a point set

Classification using the Separator



Perceptron Algorithm (Primal)

Rosenblatt, 1956

Given separable training set S and learning rate $\eta>0$

$\underline{\mathbf{w}}_0 = \underline{0}$; // Weight

$b_0 = 0$; // Bias

$k = 0$; $R = \max |\underline{x}_i|$

$\underline{\mathbf{w}} = \Sigma\, a_i y_i \underline{\mathbf{x}}_i$

repeat

  for i = 1 to N

    if $y_i\,(\underline{\mathbf{w}}_k \bullet \underline{x}_i + b_k) \le 0$ **then**

      $\underline{\mathbf{w}}_{k+1} = \underline{\mathbf{w}}_k + \eta y_i \underline{\mathbf{x}}_i$

      $b_{k+1} = b_k + \eta y_i R^2$

      $k = k + 1$

Until no mistakes made within loop

Return k, and $(\underline{\mathbf{w}}_k, b_k)$ where k = # of mistakes

Performance for Separable Data

Theorem:

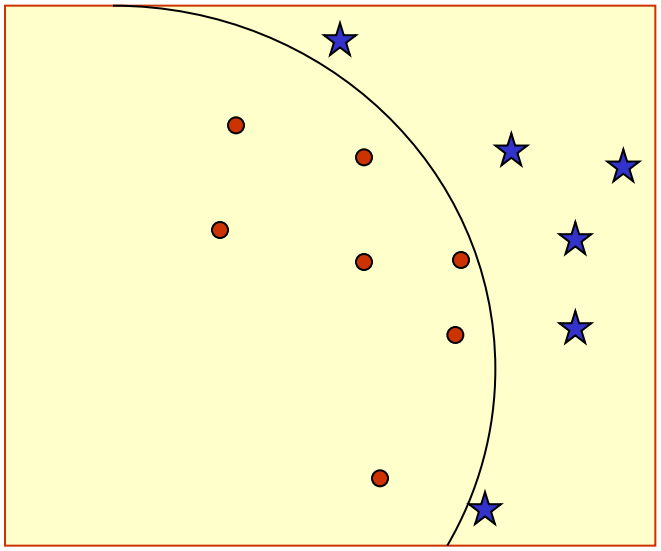
If **margin** **m** of S is positive, then

$$k \leq (2R/m)^2$$

i.e., the algorithm will always converge,

and will converge quickly.

Non-linear Separators



Main idea: Map into feature space

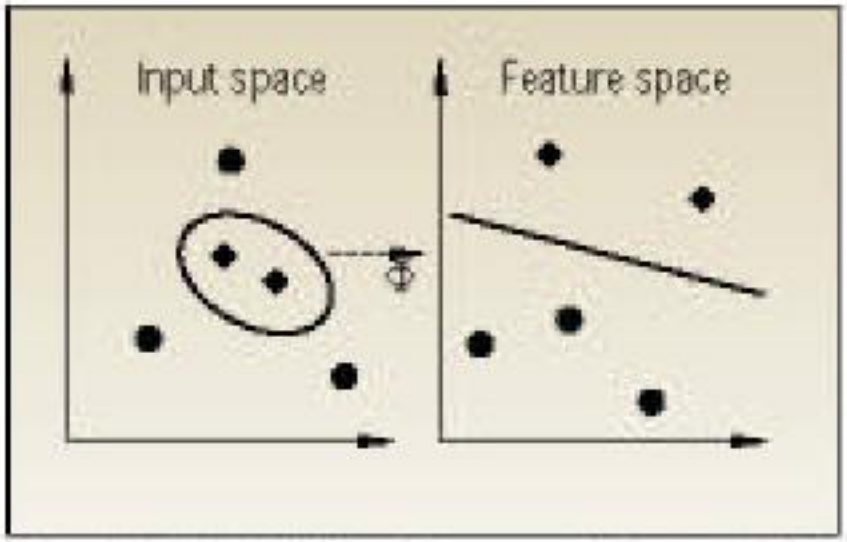
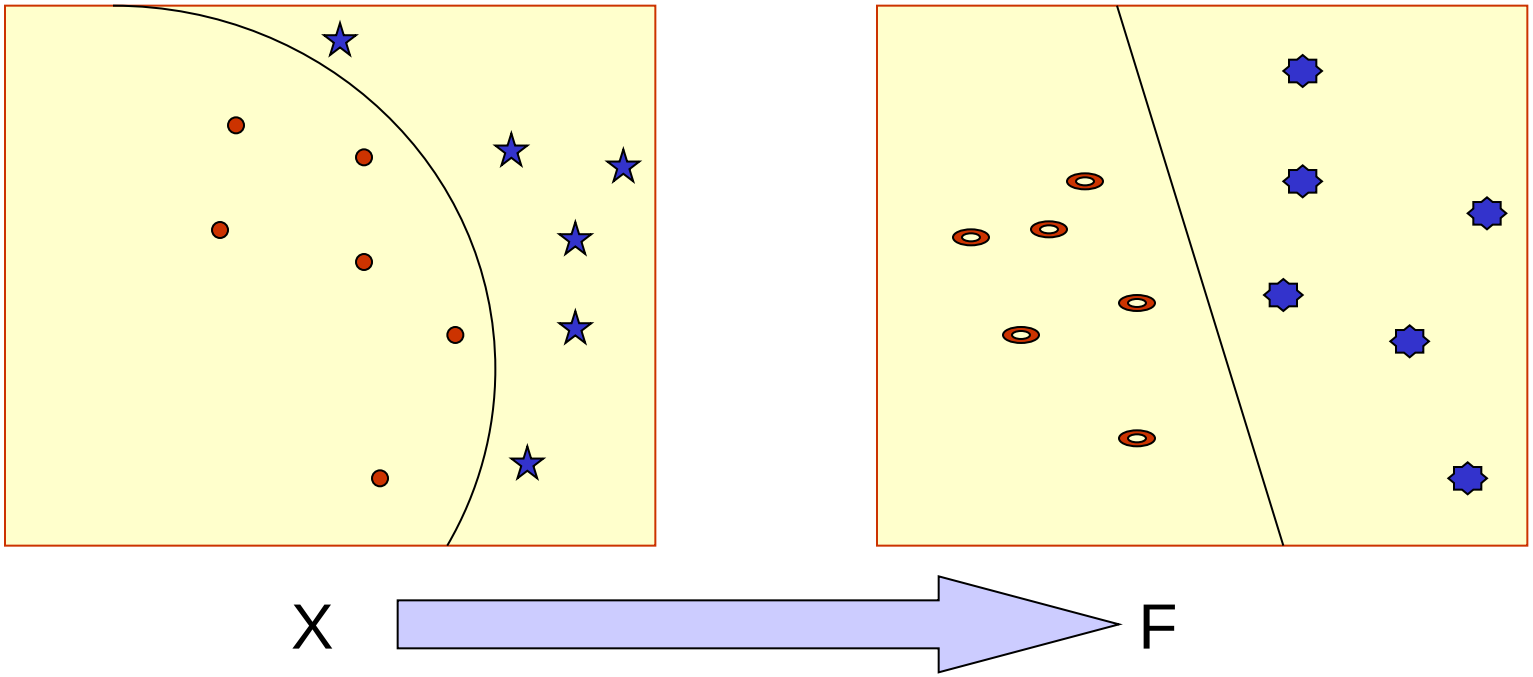


Figure 2. The idea of SV machines: map the training data nonlinearly into a higher-dimensional feature space via Φ, and construct a separating hyperplane with maximum margin there. This yields a nonlinear decision boundary in input space. By the use of a kernel function, it is possible to compute the separating hyperplane without explicitly carrying out the map into the feature space.

Non-linear Separators



Useful URLs

- http://www.support-vector.net

Perceptron Algorithm (Primal)

Rosenblatt, 1956

Given separable training set S and learning rate $\eta>0$

$\underline{\mathbf{w}}_0 = \underline{0}$; // Weight

$b_0 = 0$; // Bias

$k = 0$; $R = \max |\underline{x}_i|$

$\underline{\mathbf{w}} = \Sigma\, a_i y_i \underline{\mathbf{x}}_i$

repeat

  for i = 1 to N

    if $y_i\,(\underline{\mathbf{w}}_k \bullet \underline{x}_i + b_k) \le 0$ **then**

      $\underline{\mathbf{w}}_{k+1} = \underline{\mathbf{w}}_k + \eta y_i \underline{\mathbf{x}}_i$

      $b_{k+1} = b_k + \eta y_i R^2$

      $k = k + 1$

Until no mistakes made within loop

Return k, and $(\underline{\mathbf{w}}_k, b_k)$ where k = # of mistakes

Perceptron Algorithm (Dual)

Given a separable training set S

$\underline{\mathbf{a}} = \underline{0}$; $b_0 = 0$;

$R = \max \left|\underline{x}_i\right|$

repeat

  for i = 1 to N

  if $y_i \left(\sum a_j y_j \, \underline{x}_i \bullet \underline{x}_j + b\right) \leq 0$ **then**

    $a_i = a_i + 1$

    $b = b + y_i R^2$

  endif

Until no mistakes made within loop

Return ($\underline{\mathbf{a}}$, b)

Perceptron Algorithm (Dual)

Given a separable training set S

$\mathbf{\underline{a}} = \underline{0}$; $b_0 = 0$;

$R = \max |\underline{x}_i|$

repeat

  for i = 1 to N

    if $y_i (\sum a_j y_j \, \&(\underline{x}_i, \underline{x}_j) + b) \leq 0$ **then**

      $a_i = a_i + 1$

      $b = b + y_i R^2$

Until no mistakes made within loop

Return ($\mathbf{\underline{a}}$, b)

$\&(\underline{x}_i, \underline{x}_j) = \Phi(\underline{x}_i) \bullet \Phi(\underline{x}_j)$

Different Kernel Functions

- Polynomial kernel

$$k(X,Y) = (X \cdot Y)^d$$

- Radial Basis Kernel

$$k(X,Y) = \exp\left(\frac{-\|X - Y\|^2}{2\sigma^2}\right)$$

- Sigmoid Kernel

$$k(X,Y) = \tanh(\kappa(X \cdot Y) + \theta)$$

SVM Ingredients

- Support Vectors
- Mapping from Input Space to Feature Space
- Dot Product – Kernel function
- Weights

Generalizations

- How to deal with more than 2 classes?

  Idea: Associate weight and bias for each class.

- How to deal with non-linear separator?

  Idea: Support Vector Machines.

- How to deal with linear regression?
- How to deal with non-separable data?

Applications

- Text Categorization & Information Filtering
  - 12,902 Reuters Stories, 118 categories (91% !!)
- Image Recognition
  - Face Detection, tumor anomalies, defective parts in assembly line, etc.
- Gene Expression Analysis
- Protein Homology Detection

		Learned threshold					Optimized threshold				
Class	Method	FP	FN	TP	TN	Cost	FP	FN	TP	TN	Cost
Tricarboxylic acid	Radial SVM	8	8	9	2442	24	4	7	10	2446	18
	Dot-product-1 SVM	11	9	8	2439	29	3	6	11	2447	15
	Dot-product-2 SVM	5	10	7	2445	25	4	6	11	2446	16
	Dot-product-3 SVM	4	12	5	2446	28	4	6	11	2446	16
	Parzen	4	12	5	2446	28	0	12	5	2450	24
	FLD	9	10	7	2441	29	7	8	9	2443	23
	C4.5	7	17	0	2443	41	–	–	–	–	–
	MOC1	3	16	1	2446	35	–	–	–	–	–
Respiration	Radial SVM	9	6	24	2428	21	8	4	26	2429	16
	Dot-product-1 SVM	21	10	20	2416	41	6	9	21	2431	24
	Dot-product-2 SVM	7	14	16	2430	35	7	6	24	2430	19
	Dot-product-3 SVM	3	15	15	2434	33	7	6	24	2430	19
	Parzen	22	10	20	2415	42	7	12	18	2430	31
	FLD	10	10	20	2427	30	14	4	26	2423	22
	C4.5	18	17	13	2419	52	–	–	–	–	–
	MOC1	12	26	4	2425	64	–	–	–	–	–
Ribosome	Radial SVM	9	4	117	2337	17	6	1	120	2340	8
	Dot-product-1 SVM	13	6	115	2333	25	11	1	120	2335	13
	Dot-product-2 SVM	7	10	111	2339	27	9	1	120	2337	11
	Dot-product-3 SVM	3	18	103	2343	39	7	1	120	2339	9
	Parzen	6	8	113	2340	22	5	8	113	2341	21
	FLD	15	5	116	2331	25	8	3	118	2338	14
	C4.5	31	21	100	2315	73	–	–	–	–	–
	MOC1	26	26	95	2320	78	–	–	–	–	–

Table 2: **Comparison of error rates for various classification methods.** Classes are as described in Table 1. The methods are the radial basis function SVM, the SVMs using the scaled dot product kernel raised to the first, second and third power, Parzen windows, Fisher's linear discriminant, and the two decision tree learners, C4.5 and MOC1. The next five columns are the false positive, false negative, true positive and true negative rates summed over three cross-validation splits, followed by the cost, which is the number of false positives plus twice the number of false negatives. These five columns are repeated twice, first using the threshold learned from the training set, and then using the threshold that minimizes the cost on the test set. The threshold optimization is not possible for the decision tree methods, since they do not produce ranked results.

		Learned threshold					Optimized threshold				
Class	Method	FP	FN	TP	TN	Cost	FP	FN	TP	TN	Cost
Proteasome	Radial SVM	3	7	28	2429	17	4	5	30	2428	14
	Dot-product-1 SVM	14	11	24	2418	36	2	7	28	2430	16
	Dot-product-2 SVM	4	13	22	2428	30	4	6	29	2428	16
	Dot-product-3 SVM	3	18	17	2429	39	2	7	28	2430	16
	Parzen	21	5	30	2411	31	3	9	26	2429	21
	FLD	7	12	23	2425	31	12	7	28	2420	26
	C4.5	17	10	25	2415	37	–	–	–	–	–
	MOC1	10	17	18	2422	44	–	–	–	–	–
Histone	Radial SVM	0	2	9	2456	4	0	2	9	2456	4
	Dot-product-1 SVM	0	4	7	2456	8	0	2	9	2456	4
	Dot-product-2 SVM	0	5	6	2456	10	0	2	9	2456	4
	Dot-product-3 SVM	0	8	3	2456	16	0	2	9	2456	4
	Parzen	2	3	8	2454	8	1	3	8	2455	7
	FLD	0	3	8	2456	6	2	1	10	2454	4
	C4.5	2	2	9	2454	6	–	–	–	–	–
	MOC1	2	5	6	2454	12	–	–	–	–	–
Helix-turn-helix	Radial SVM	1	16	0	2450	33	0	16	0	2451	32
	Dot-product-1 SVM	20	16	0	2431	52	0	16	0	2451	32
	Dot-product-2 SVM	4	16	0	2447	36	0	16	0	2451	32
	Dot-product-3 SVM	1	16	0	2450	33	0	16	0	2451	32
	Parzen	14	16	0	2437	46	0	16	0	2451	32
	FLD	14	16	0	2437	46	0	16	0	2451	32
	C4.5	2	16	0	2449	34	–	–	–	–	–
	MOC1	6	16	0	2445	38	–	–	–	–	–

Table 3: **Comparison of error rates for various classification methods (continued).** See caption for Table 2.

Class	Kernel	Cost for each split					Total
Tricarboxylic acid	Radial	18	21	15	22	21	97
	Dot-product-1	15	22	18	23	22	100
	Dot-product-2	16	22	17	22	22	99
	Dot-product-3	16	22	17	23	22	100
Respiration	Radial	16	18	23	20	16	93
	Dot-product-1	24	24	29	27	23	127
	Dot-product-2	19	19	26	24	23	111
	Dot-product-3	19	19	26	22	21	107
Ribosome	Radial	8	12	15	11	13	59
	Dot-product-1	13	18	14	16	16	77
	Dot-product-2	11	16	14	16	15	72
	Dot-product-3	9	15	11	15	15	65
Proteasome	Radial	14	10	9	11	11	55
	Dot-product-1	16	12	12	17	19	76
	Dot-product-2	16	13	15	17	17	78
	Dot-product-3	16	13	16	16	17	79
Histone	Radial	4	4	4	4	4	20
	Dot-product-1	4	4	4	4	4	20
	Dot-product-2	4	4	4	4	4	20
	Dot-product-3	4	4	4	4	4	20

Table 4: **Comparison of SVM performance using various kernels.** For each of the MYGD classifications, SVMs were trained using four different kernel functions on five different random three-fold splits of the data, training on two-thirds and testing on the remaining third. The first column contains the class, as described in Table 1. The second column contains the kernel function, as described in Table 2. The next five columns contain the threshold-optimized cost (i.e., the number of false positives plus twice the number of false negatives) for each of the five random three-fold splits. The final column is the total cost across all five splits.

Family	Gene	Locus	Error	Description
TCA	YPR001W	CIT3	FN	mitochondrial citrate synthase
	YOR142W	LSC1	FN	α subunit of succinyl-CoA ligase
	YNR001C	CIT1	FN	mitochondrial citrate synthase
	YLR174W	IDP2	FN	isocitrate dehydrogenase
	YIL125W	KGD1	FN	α-ketoglutarate dehydrogenase
	YDR148C	KGD2	FN	component of α-ketoglutarate dehydrogenase complex in mitochondria
	YDL066W	IDP1	FN	mitochondrial form of isocitrate dehydrogenase
	YBL015W	ACH1	FP	acetyl CoA hydrolase
Resp	YPR191W	QCR2	FN	ubiquinol cytochrome-c reductase core protein 2
	YPL271W	ATP15	FN	ATP synthase epsilon subunit
	YPL262W	FUM1	FP	fumarase
	YML120C	NDI1	FP	mitochondrial NADH ubiquinone 6 oxidoreductase
	YKL085W	MDH1	FP	mitochondrial malate dehydrogenase
	YDL067C	COX9	FN	subunit VIIa of cytochrome c oxidase
Ribo	YPL037C	EGD1	FP	β subunit of the nascent-polypeptide-associated complex (NAC)
	YLR406C	RPL31B	FN	ribosomal protein L31B (L34B) (YL28)
	YLR075W	RPL10	FP	ribosomal protein L10
	YAL003W	EFB1	FP	translation elongation factor EF-1β
Prot	YHR027C	RPN1	FN	subunit of 26S proteasome (PA700 subunit)
	YGR270W	YTA7	FN	member of CDC48/PAS1/SEC18 family of ATPases
	YGR048W	UFD1	FP	ubiquitin fusion degradation protein
	YDR069C	DOA4	FN	ubiquitin isopeptidase
	YDL020C	RPN4	FN	involved in ubiquitin degradation pathway
Hist	YOL012C	HTA3	FN	histone-related protein
	YKL049C	CSE4	FN	required for proper kinetochore function

Table 6: **Consistently misclassified genes.** The table lists all 25 genes that are consistently misclassified by SVMs trained using the MYGD classifications listed in Table 1. Two types of errors are included: a false positive (FP) occurs when the SVM includes the gene in the given class but the MYGD classification does not; a false negative (FN) occurs when the SVM does not include the gene in the given class but the MYGD classification does.

Kernel DF	Feature	FP	FN	TP	TN
dot-product 0	25	5	4	10	12
dot-product 2	25	5	2	12	12
dot-product 5	25	4	2	12	13
dot-product 10	25	4	2	12	13
dot-product 0	50	4	2	12	13
dot-product 2	50	3	2	12	14
dot-product 5	50	3	2	12	14
dot-product 10	50	3	2	12	14
dot-product 0	100	4	3	11	13
dot-product 2	100	5	3	11	12
dot-product 5	100	5	3	11	12
dot-product 10	100	5	3	11	12
dot-product 0	500	5	3	11	12
dot-product 2	500	4	3	11	13
dot-product 5	500	4	3	11	13
dot-product 10	500	4	3	11	13
dot-product 0	1000	7	3	11	10
dot-product 2	1000	5	3	11	12
dot-product 5	1000	5	3	11	12
dot-product 10	1000	5	3	11	12
dot-product 0	97802	17	0	14	0
dot-product 2	97802	9	2	12	8
dot-product 5	97802	7	3	11	10
dot-product 10	97802	5	3	11	12

Table 1: **Error rates for ovarian cancer tissue experiments.**
For each setting of the SVM consisting of a kernel and diagonal factor (DF), each tissue was classified. Column 2 is the number of features (clones) used. Reported are the number of normal tissues misclassified (FP), tumor tissues misclassified (FN), tumor tissues classified correctly (TP), and normal tissues classified correctly (TN).

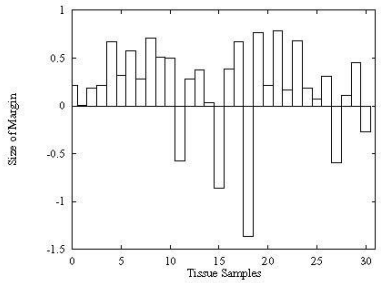


Figure 1: **SVM classification margins for ovarian tissues.** When classifying, the SVM calculates a margin which is the distance of an example from the decision boundary it has learned. In this graph, the margin for each tissue sample calculated using (10) is shown. A positive value indicates a correct classification, and a negative value indicates an incorrect classification. The most negative point corresponds to tissue N039. The second most negative point corresponds to tissue HWBC3.

Dataset	Features	FP	FN	SVM FP	SVM FN
Ovarian(original)	97802	4.6	4.8	5	3
Ovarian(modified)	97802	4.4	3.4	0	0
AML/ALL train	7129	0.6	2.8	0	0
AML treatment	7129	4.8	3.5	3	2
Colon	2000	3.8	3.7	3	3

Table 5: Results for the perceptron on all data sets. The results are averaged over 5 shufflings of the data as this algorithm is sensitive to the order in which it receives the data points. The first column is the dataset used and the second is number of features in the dataset. For the ovarian and colon datasets, the number of normal tissues misclassified (FP) and the number of tumor tissues misclassified (FN) is reported. For the AML/ALL training dataset, the number of AML samples misclassified (FP) and the number of ALL patients misclassified (FN) is reported. For the AML treatment dataset, the number of unsuccessfully treated patients misclassified (FP) and the number of successfully treated patients misclassified (FN) is reported. The last two columns report the best score obtained by the SVM on that dataset.

Decision Trees

BioPerl

Perl: Practical Extraction & Report Language

- Created by Larry Wall, early 90s
- Portable, "glue" language for interfacing C/Fortran code, WWW/CGI, graphics, numerical analysis and much more
- Easy to use and extensible
- OOP support, simple databases, simple data structures.
- From interpreted to compiled
- high-level features, and relieves you from manual memory management, segmentation faults, bus errors, most portability problems, etc, etc.
- Competitors: Python, Tcl, Java

Perl Features

- Bit Operations
- Pattern Matching
- Subroutines
- Packages & Modules
- Objects
- Interprocess Communication
- Threads , Process control
- Compiling

Managing a Large Project

- Devise a common data exchange format.
- Use modules that have already been developed.
- Write Perl scripts to convert to and from common data exchange format.
- Write Perl scripts to "glue" it all together.

What is Bioperl?

- Tookit of Perl modules useful for bioinformatics
- Open source; Current version: Bioperl 1.5.2
- Routines for handling biosequence and alignment data.
- Why? Human Genome Project: Same project, same data. different data formats! Different input formats. Different output formats for comparable utility programs.
  - BioPerl was useful to interchange data and meaningfully exchange results. "Perl Saved the Human Genome Project"
- Many routine tasks automated using BioPerl.
- String manipulations (string operations: substring, match, etc.; handling string data: names, annotations, comments, bibliographical references; regular expression operations)
- Modular: modules in any language

Miscellaneous

- pTk – to enable building Perl-driven GUIs for X-Window systems.
- BioJava
- BioPython
- The BioCORBA Project provides an object-oriented, language neutral, platform-independent method for describing and solving bioinformatics problems.

Perl: Examples

```
#!/usr/bin/perl -w
# Storing DNA in a variable, and printing it out

# First we store the DNA in a variable called $DNA
$DNA = 'ACGGGAGGACGGGAAAATTACTACGGCATTAGC';

# Next, we print the DNA onto the screen
print $DNA;

# Finally, we'll specifically tell the program to exit.
exit;  #perl1.pl
```

Perl: Strings

```
#!/usr/bin/perl -w
$DNA1 = 'ACGGGAGGACGGGAAAATTACTACGGCATTAGC';
$DNA2 = 'ATAGTGCCGTGAGAGTGATGTAGTA';
# Concatenate the DNA fragments
$DNA3 = "$DNA1$DNA2";
print "Concatenation 1):\n\n$DNA3\n\n";
# An alternative way using the "dot operator":
$DNA3 = $DNA1 . $DNA2;
print "Concatenation 2):\n\n$DNA3\n\n";
# transcribe from DNA to RNA; make rev comp; print;
$RNA = $DNA3; $RNA =~ s/T/U/g;
$rev = reverse $DNA3; $rev =~ tr/AGCTacgt/TCGAtgca/;
print "$RNA\n$rev\n";
exit;  #perl2.pl
```

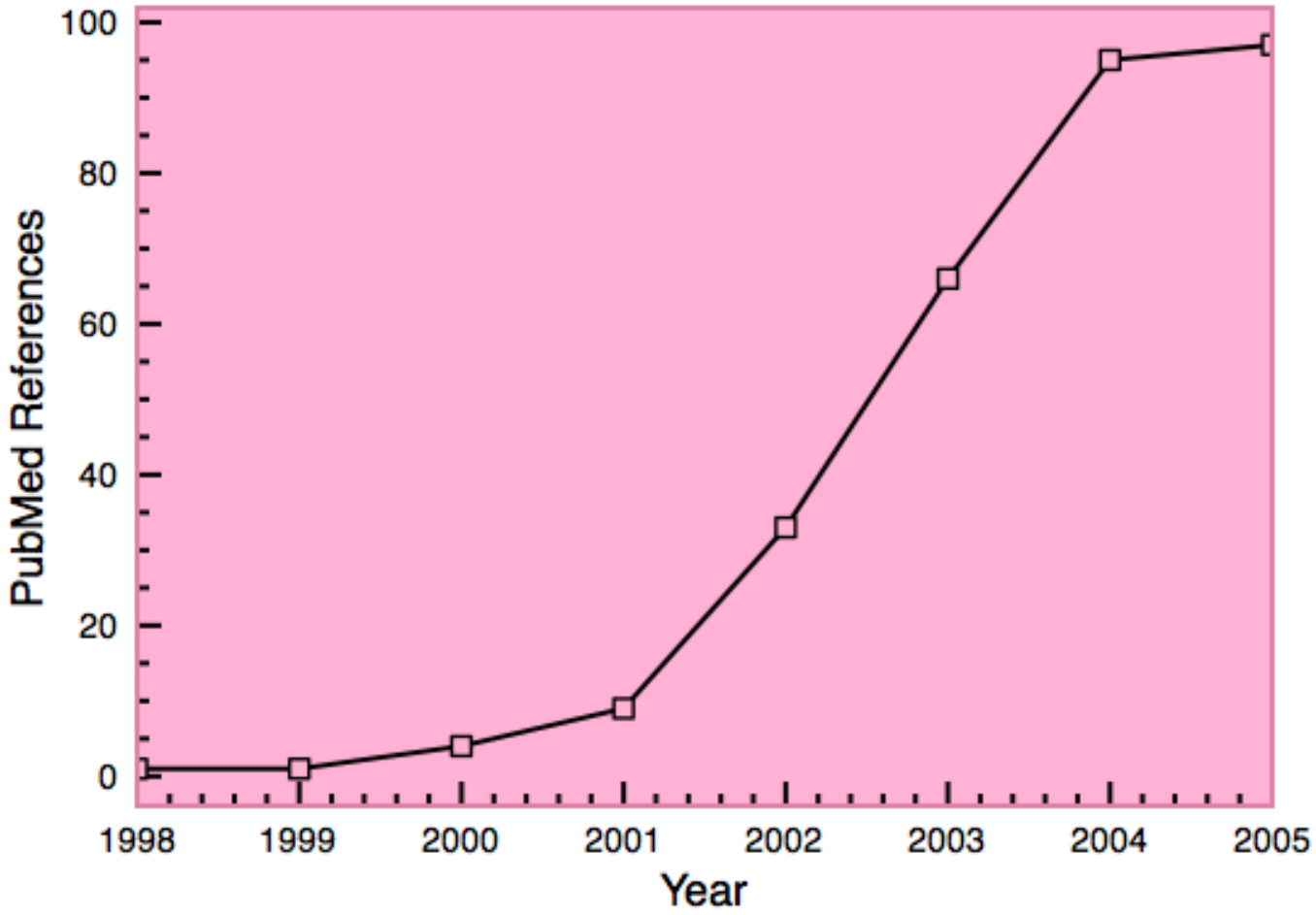
Perl: arrays

```
#!/usr/bin/perl -w
# Read filename & remove newline from string
print "Type name of protein file: ";
$protFile = <STDIN>; chomp $protFile;
# First we have to "open" the file
unless (open(PROTEINFILE, $protFile) {
   print "File $protFile does not exist"; exit;}
# Each line becomes an element of array @protein
@protein = <PROTEINFILE>;
print @protein;
# Print line #3 and number of lines
print $protein[2], "File contained ", scalar @protein, "
   lines\n";
# Close the file.
close PROTEINFILE;
exit; #perl3.pl
```

Perl: subroutines

```
#!/usr/bin/perl -w
# using command line argument
$dna1 = $ARGV[0]; $dna2 = $ARGV[1];

# Call subroutine with arguments; result in $dna
$dna = addGACAGT($dna1, $dna2);
print "Add GACAGT to $dna1 & $dna2 to get $dna\n\n";
exit;
##### addACGT: concat $dna1, $dna2, & "ACGT". #####
sub addGACAGT {
    my($dnaA, $dnaB) = @_; my($dnaC) = $dnaA.$dnaB;
    $dnaC .= 'ACGT';
    return $dnaC;
} #perl4.pl
```

How Widely is Bioperl Used?



What Can You Do with Bioperl?

- Accessing sequence data from local and remote databases
- Transforming formats of database/ file records
- Manipulating individual sequences
- Searching for similar sequences
- Creating and manipulating sequence alignments
- Searching for genes and other structures on genomic DNA
- Developing machine readable sequence annotations

Types of Perl Objects:

- Sequence objects
- Location objects
- Interface objects
- Implementation objects

BioPerl Modules

- **Bio::PreSeq**, module for reading, accessing, manipulating, analyzing single sequences.
- **Bio::UnivAln**, module for reading, parsing, writing, slicing, and manipulating multiple biosequences (sequence multisets & alignments).
- **Bio::Struct**, module for reading, writing, accessing, and manipulating 3D structures.
- Support for invoking **BLAST** & other programs.
- Download URL [http://www.bioperl.org/Core/Latest/]
- Tutorial [http://www.bioperl.org/Core/Latest/bptutorial.html]
- Course [http://www.pasteur.fr/recherche/unites/sis/formation/bioperl/index.html]

BioPerl Sequence Object

```
$seqobj->display_id(); # readable id of sequence
$seqobj->seq(); # string of sequence
$seqobj->subseq(5,10); # part of the sequence as a string
$seqobj->accession_number(); # if present, accession num
$seqobj->moltype(); # one of 'dna','rna','protein'
$seqobj->primary_id(); # unique id for sequence independent
                       # of its display_id or accession number
```

Example 1: Convert SwissProt to fasta format

```
#! /local/bin/perl -w

use strict;
use Bio::SeqIO;
my $in  = Bio::SeqIO->newFh ( -file   => '<seqs.html',
                              -format => 'swiss' );
my $out = Bio::SeqIO->newFh ( -file   => '>seqs.fasta',
                              -format => 'fasta' );

print $out $_ while <$in>;

exit; #bioperl1.pl
```

Example 2 : Load sequence from remote server

```
#!/usr/bin/perl -w
use Bio::DB::SwissProt;

$database = new Bio::DB::SwissProt;

$seq = $database->get_Seq_by_id('MALK_ECOLI');

my $out = Bio::SeqIO->newFh(-fh => STDOUT,
            -format => 'fasta');

print $out $seq;

exit;
```

```
#!/local/bin/perl -w

use Bio::DB::GenBank;

my $gb =
     new Bio::DB::GenBank(
     -retrievaltype=>'tempfile',
     -format=>'Fasta');

my ($seq) = $seq =
     $gb->get_Seq_by_id("5802612");
print $seq->id, "\n";
print $seq->desc(), "Sequence: \n";
print $seq->seq(), "\n";
exit;
```

Sequence Formats in BioPerl

```
#! /local/bin/perl -w
use strict;
use Bio::SeqIO;
my $in  = Bio::SeqIO->new ( -file   => 'seqs.html', -format => 'swiss' );
my $out = Bio::SeqIO->new ( -file   => 'seqs.fas', -format => 'fasta' );

while ($seq = $in->next_seq()) {
    $accNum = $seq->accession_number();
    print "Accession# = $accNum\n";
    $out->write_seq($seq);
}

exit; #bioperl2.pl
```

BioPerl

```
#!/usr/bin/perl –w
# define a DNA sequence object with given sequence
$seq = Bio::Seq->new('-seq'=>'actgtggcgtcaact',
    '-desc'=>'Sample Bio::Seq object',
    '-display_id' => 'somethingxxx',
    '-accession_number' => 'accnumxxx',
    '-alphabet' => 'dna' );
$gb = new Bio::DB::GenBank();

$seq = $gb->get_Seq_by_id('MUSIGHBA1'); #returns Seq object
$seq = $gb->get_Seq_by_acc('AF303112'); #returns Seq object
# this returns a SeqIO object :
$seqio = $gb->get_Stream_by_batch([ qw(J00522 AF303112)]));
exit; #bioperl3.pl
```

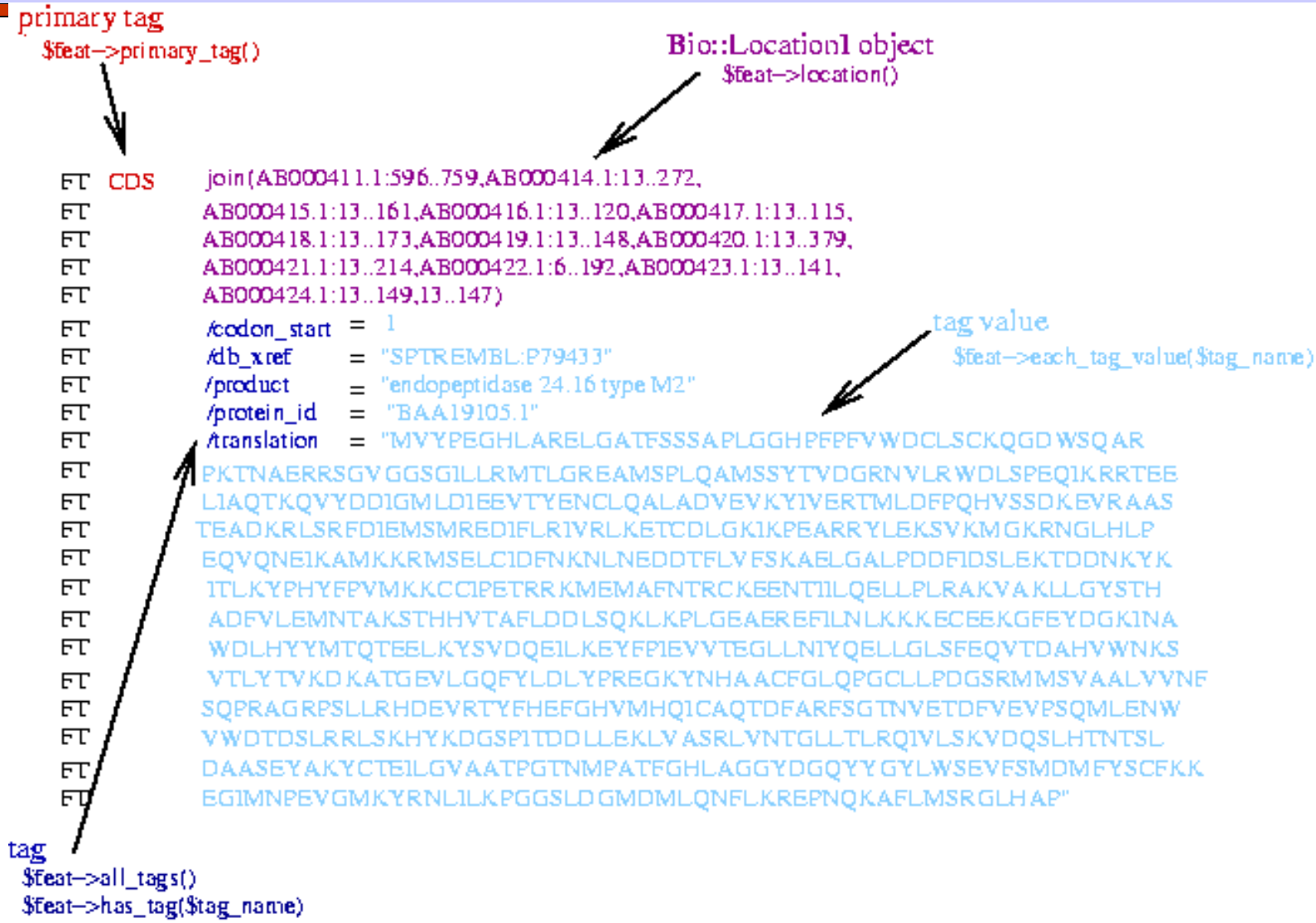
Sequence Manipulations

```
#!/local/bin/perl -w

use Bio::DB::GenBank;

$gb = new Bio::DB::GenBank();

$seq1 = $gb->get_Seq_by_acc('AF303112');
$seq2=$seq1->trunc(1,90);
$seq2 = $seq2->revcom();

print $seq2->seq(), "\n";
$seq3=$seq2->translate;
print $seq3->seq(), "\n";
exit; #bioperl4.pl
```

BioPerl: Structure

```
use Bio::Structure::IO;
$in = Bio::Structure::IO->new(-file => "inputfilename" , '-format' =>
    'pdb');
$out = Bio::Structure::IO->new(-file => ">outputfilename" , '-format' =>
    'pdb');
# note: we quote -format to keep older perl's from complaining.
while ( my $struc = $in->next_structure() ) {
    $out->write_structure($struc);
    print "Structure ",$struc->id," number of models: ",
          scalar $struc->model,"\n";
}
```

Sequence Features



BioPerl: Seq and SeqIO

```
use Bio::SeqIO;
$seqin = Bio::SeqIO->new(-format =>'EMBL', -file=>'f1');
$seqout= Bio::SeqIO->new(-format =>'Fasta',-file=>'>f1.fa');
while ((my $seqobj = $seqin->next_seq())) {
    print "Seq: ", $seqobj->display_id, ", Start of seq ",
          substr($seqobj->seq,1,10),"\n";
    if ( $seqobj->moltype eq 'dna') {
          $rev = $seqobj->revcom;
          $id = $seqobj->display_id();
          $id = "$id.rev";
          $rev->display_id($id);
          $seqout->write_seq($rev); } #end if
    foreach $feat ( $seqobj->top_SeqFeatures() ) {
          if( $feat->primary_tag eq 'exon' ) {
            print STDOUT "Location ",$feat->start,":",
            $feat->end," GFF[",$feat->gff_string,"]\n";}
    } # end foreach
} # end while
exit; #bioperl6.pl
```

Example 3: Read alignment file using AlignIO class

```
#!/usr/bin/perl -w
use strict;
use Bio::AlignIO;
my $in = new Bio::AlignIO(-file => '<data/infile.aln',
                          -format => 'clustalw');

# returns an alignI (alignment interface)
my $aln = $in->next_aln();
print "same length of all sequences: ",
($aln->is_flush()) ? "yes" : "no", "\n";
print "alignment length: ", $aln->length, "\n";
printf "identity: %.2f %%\n", $aln->percentage_identity();
printf "identity of conserved columns: %.2f %%\n",

$aln->overall_percentage_identity();
```

Example 4: Standalone BLAST

```
#!/usr/bin/perl -w
use strict;
use Bio::SeqIO;
use Bio::Tools::Run::StandAloneBlast;
my $seq_in = Bio::SeqIO -> new(-file => '<data/prot1.fasta',
                               -format => 'fasta');
my $query = $seq_in -> next_seq();
my $factory = Bio::Tools::Run::StandAloneBlast -> new('program' => 'blastp',
                                                      'database' => 'swissprot',
                                                      _READMETHOD => 'Blast');
my $blast_report = $factory->blastall($query);
my $result = $blast_report->next_result();
while (my $hit = $result->next_hit()){
  print "\thit name: ", $hit->name(), "Significance: ", $hit->significance(), "\n";
  while (my $hsp = $hit->next_hsp()){
    print "E: ", $hsp->evalue(), "frac_identical: ", $hsp->frac_identical(), "\n";
  }

}
exit;
```

CpG Islands

- Regions in DNA sequences with increased occurrences of substring "*CG*"
- Rare: typically *C* gets methylated and then mutated into a T.
- Often around promoter or "start" regions of genes
- Few hundred to a few thousand bases long

Problem 1:

- Input: Small sequence S
- Output: Is S from a CpG island?
  - Build Markov models: M+ and M —
  - Then compare

Markov Models

+	A	C	G	T
A	0.180	0.274	0.426	0.120
C	0.171	0.368	0.274	0.188
G	0.161	0.339	0.375	0.125
T	0.079	0.355	0.384	0.182

–	A	C	G	T
A	0.300	0.205	0.285	0.210
C	0.322	0.298	0.078	0.302
G	0.248	0.246	0.298	0.208
T	0.177	0.239	0.292	0.292

How to distinguish?

- Compute

$$S(x) = \log\left(\frac{P(x \mid M+)}{P(x \mid M-)}\right) = \sum_{i=1}^{L} \log\left(\frac{p_{x(i-1)xi}}{m_{x(i-1)xi}}\right) = \sum_{i=1}^{L} r_{x(i-1)xi}$$

r=p/m	A	C	G	T
A	-0.740	0.419	0.580	-0.803
C	-0.913	0.302	1.812	-0.685
G	-0.624	0.461	0.331	-0.730
T	-1.169	0.573	0.393	-0.679

Score(GCAC)
= .461-.913+.419
< 0.
GCAC not from CpG island.

Score(GCTC)
= .461-.685+.573
> 0.
GCTC from CpG island.

Problem 1:

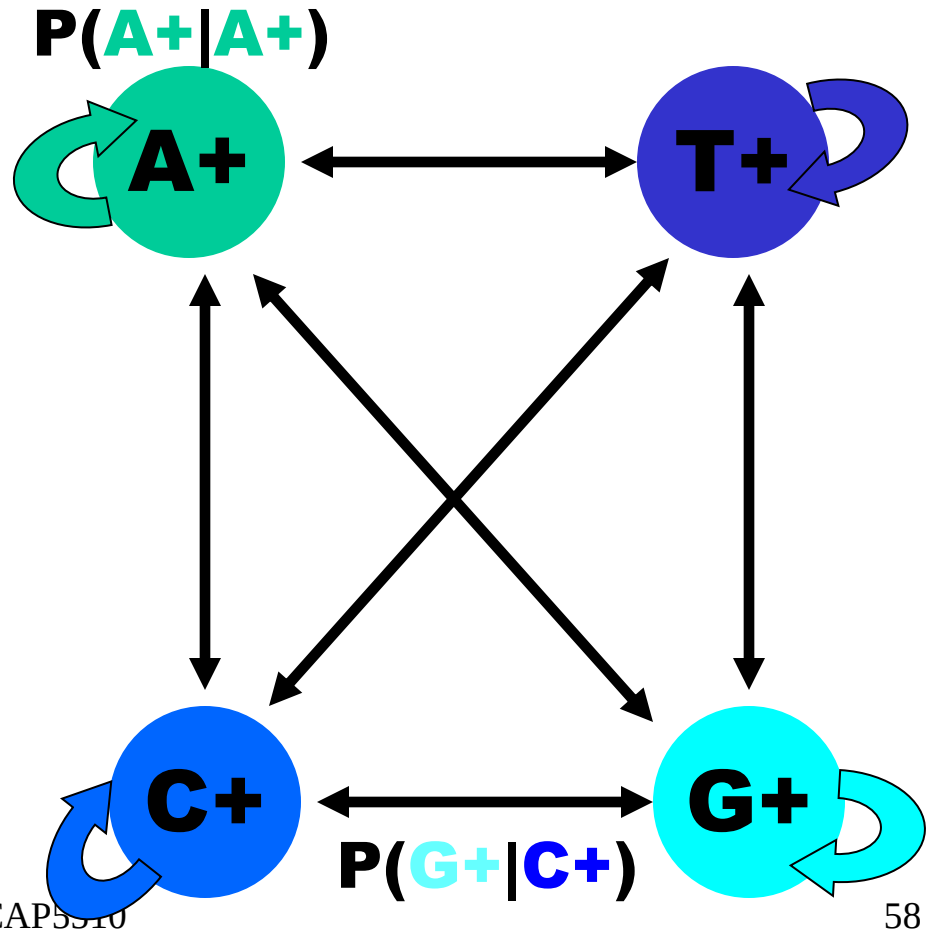
- Input: Small sequence S
- Output: Is S from a CpG island?
  - Build Markov Models: M+ & M-
  - Then compare

Problem 2:

- Input: Long sequence S
- Output: Identify the CpG islands in S.
  - Markov models are inadequate.
  - Need Hidden Markov Models.

Markov Models

+	A	C	G	T
A	0.180	0.274	0.426	0.120
C	0.171	0.368	0.274	0.188
G	0.161	0.339	0.375	0.125
T	0.079	0.355	0.384	0.182



CpG Island + in an ocean of –
First order **Hidden** Markov Model

MM=16, HMM= 64 transition probabilities (adjacent bp)