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

\[ \mathbf{w} \cdot \mathbf{x}_i + b > 0 \]

\[ \mathbf{w} \cdot \mathbf{x}_j + b < 0 \]

Separator \[ \mathbf{w} \cdot \mathbf{x} + b = 0 \]
Perceptron Algorithm (Primal)

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

$w_0 = 0$; // Weight

$b_0 = 0$; // Bias

$k = 0$; $R = \max |x_i|$

repeat

for $i = 1$ to $N$

if $y_i (w_k \cdot x_i + b_k) \leq 0$ then

$w_{k+1} = w_k + \eta y_i x_i$

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

$k = k + 1$

Until no mistakes made within loop

Return $k$, and $(w_k, b_k)$ where $k = \#$ of mistakes

Rosenblatt, 1956

$$w = \sum a_i y_i x_i$$
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 $\Phi$, 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
Given separable training set S and learning rate $\eta>0$

$\mathbf{w}_0 = \mathbf{0};$  // Weight
$b_0 = 0;$  // Bias

$k = 0; R = \max |x_i|$

repeat
  for $i = 1$ to $N$
    if $y_i (\mathbf{w}_k \cdot x_i + b_k) \leq 0$ then
      $\mathbf{w}_{k+1} = \mathbf{w}_k + \eta y_i x_i$
      $b_{k+1} = b_k + \eta y_i R^2$
      $k = k + 1$

Until no mistakes made within loop

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

Rosenblatt, 1956
Perceptron Algorithm (Dual)

Given a separable training set $S$

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

$R = \max |x_i|$

repeat
  for $i = 1$ to $N$
    if $y_i (\sum a_j y_j x_i \cdot x_j + b) \leq 0$ then
      $a_i = a_i + 1$
      $b = b + y_i R^2$
    endif
  endif
Until no mistakes made within loop
Return $(\mathbf{a}, b)$
Perceptron Algorithm (Dual)

Given a separable training set S

\( a = 0; \ b_0 = 0; \)

\( R = \max |x_i| \)

repeat
  for \( i = 1 \) to \( N \)
    if \( y_i (\sum a_j y_j \ &\Phi'(x_i, x_j) + b) \leq 0 \) then
      \( a_i = a_i + 1 \)
      \( b = b + y_i R^2 \)

Until no mistakes made within loop

Return \((a, b)\)

\( \&\Phi'(x_i, x_j) = \Phi(x_i) \cdot \Phi(x_j) \)
Different Kernel Functions

- Polynomial kernel
  \[(X, Y) = (X \cdot Y)^d\]

- Radial Basis Kernel
  \[(X, Y) = \exp \left( \frac{\|X \cdot Y\|^2}{2} \right)\]

- Sigmoid Kernel
  \[(X, Y) = \tanh (X \cdot Y)\]
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**
<table>
<thead>
<tr>
<th>Class</th>
<th>Method</th>
<th>Learned threshold</th>
<th>Optimized threshold</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>FP</td>
<td>FN</td>
<td>TP</td>
</tr>
<tr>
<td>Tricarboxylic acid</td>
<td>Radial SVM</td>
<td>8</td>
<td>8</td>
</tr>
<tr>
<td></td>
<td>Dot-product-1 SVM</td>
<td>11</td>
<td>9</td>
</tr>
<tr>
<td></td>
<td>Dot-product-2 SVM</td>
<td>5</td>
<td>10</td>
</tr>
<tr>
<td></td>
<td>Dot-product-3 SVM</td>
<td>4</td>
<td>12</td>
</tr>
<tr>
<td>Respiration</td>
<td>Parzen</td>
<td>4</td>
<td>12</td>
</tr>
<tr>
<td></td>
<td>FLFD</td>
<td>9</td>
<td>10</td>
</tr>
<tr>
<td></td>
<td>C4.5</td>
<td>7</td>
<td>17</td>
</tr>
<tr>
<td></td>
<td>MOCl</td>
<td>3</td>
<td>16</td>
</tr>
<tr>
<td>Ribosome</td>
<td>Radial SVM</td>
<td>9</td>
<td>6</td>
</tr>
<tr>
<td></td>
<td>Dot-product-1 SVM</td>
<td>21</td>
<td>10</td>
</tr>
<tr>
<td></td>
<td>Dot-product-2 SVM</td>
<td>7</td>
<td>14</td>
</tr>
<tr>
<td></td>
<td>Dot-product-3 SVM</td>
<td>3</td>
<td>15</td>
</tr>
<tr>
<td></td>
<td>Parzen</td>
<td>22</td>
<td>10</td>
</tr>
<tr>
<td></td>
<td>FLFD</td>
<td>10</td>
<td>10</td>
</tr>
<tr>
<td></td>
<td>C4.5</td>
<td>18</td>
<td>17</td>
</tr>
<tr>
<td></td>
<td>MOCl</td>
<td>12</td>
<td>26</td>
</tr>
</tbody>
</table>

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 MOCl. 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 their output is a range of thresholds.
<table>
<thead>
<tr>
<th>Class</th>
<th>Kernel</th>
<th>Cost for each split</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tricarboxylic</td>
<td>Radial</td>
<td>18 21 15 22 21</td>
<td>97</td>
</tr>
<tr>
<td>acid</td>
<td>Dot-product-1</td>
<td>15 22 18 23 22</td>
<td>100</td>
</tr>
<tr>
<td></td>
<td>Dot-product-2</td>
<td>16 22 17 22 22</td>
<td>99</td>
</tr>
<tr>
<td></td>
<td>Dot-product-3</td>
<td>16 22 17 23 22</td>
<td>100</td>
</tr>
<tr>
<td></td>
<td>Radial</td>
<td>16 18 23 20 16</td>
<td>93</td>
</tr>
<tr>
<td>Respiration</td>
<td>Dot-product-1</td>
<td>24 24 29 27 23</td>
<td>127</td>
</tr>
<tr>
<td></td>
<td>Dot-product-2</td>
<td>19 19 26 24 23</td>
<td>111</td>
</tr>
<tr>
<td></td>
<td>Dot-product-3</td>
<td>19 19 26 22 21</td>
<td>107</td>
</tr>
<tr>
<td>Ribosome</td>
<td>Radial</td>
<td>8 12 15 11 13</td>
<td>59</td>
</tr>
<tr>
<td></td>
<td>Dot-product-1</td>
<td>13 18 14 16 16</td>
<td>77</td>
</tr>
<tr>
<td></td>
<td>Dot-product-2</td>
<td>11 16 14 16 15</td>
<td>72</td>
</tr>
<tr>
<td></td>
<td>Dot-product-3</td>
<td>13 16 15 15 15</td>
<td>65</td>
</tr>
<tr>
<td>Proteasome</td>
<td>Radial</td>
<td>14 10 9 11 11</td>
<td>55</td>
</tr>
<tr>
<td></td>
<td>Dot-product-1</td>
<td>16 12 12 17 19</td>
<td>76</td>
</tr>
<tr>
<td></td>
<td>Dot-product-2</td>
<td>16 13 15 17 17</td>
<td>78</td>
</tr>
<tr>
<td></td>
<td>Dot-product-3</td>
<td>16 13 16 16 17</td>
<td>79</td>
</tr>
<tr>
<td>Histone</td>
<td>Radial</td>
<td>4 4 4 4 4</td>
<td>20</td>
</tr>
<tr>
<td></td>
<td>Dot-product-1</td>
<td>4 4 4 4 4</td>
<td>20</td>
</tr>
<tr>
<td></td>
<td>Dot-product-2</td>
<td>4 4 4 4 4</td>
<td>20</td>
</tr>
<tr>
<td></td>
<td>Dot-product-3</td>
<td>4 4 4 4 4</td>
<td>20</td>
</tr>
</tbody>
</table>

Table 4: Comparison of SVM performance using various kernels. For each of the MYGE 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.

<table>
<thead>
<tr>
<th>Family</th>
<th>Gene</th>
<th>Locus</th>
<th>Error</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TCA</td>
<td>YPR001W</td>
<td>CIT3</td>
<td>FN</td>
<td>mitochondrial citrate synthase</td>
</tr>
<tr>
<td></td>
<td>YOR142W</td>
<td>LSC1</td>
<td>FN</td>
<td>α subunit of succinyl-CoA ligase</td>
</tr>
<tr>
<td></td>
<td>YNR001C</td>
<td>CIT1</td>
<td>FN</td>
<td>mitochondrial citrate synthase</td>
</tr>
<tr>
<td></td>
<td>YLR174W</td>
<td>IDP2</td>
<td>FN</td>
<td>isocitrate dehydrogenase</td>
</tr>
<tr>
<td></td>
<td>YIL125W</td>
<td>KGD1</td>
<td>FN</td>
<td>α-ketoglutarate dehydrogenase</td>
</tr>
<tr>
<td></td>
<td>YDR148C</td>
<td>KGD2</td>
<td>FN</td>
<td>component of α-ketoglutarate dehydrogenase complex in mitochondria</td>
</tr>
<tr>
<td></td>
<td>YDL065W</td>
<td>IDP1</td>
<td>FN</td>
<td>mitochondrial form of isocitrate dehydrogenase</td>
</tr>
<tr>
<td></td>
<td>YBL015W</td>
<td>ACH1</td>
<td>FP</td>
<td>acetyl CoA hydrolase</td>
</tr>
<tr>
<td>Resp</td>
<td>YPR191W</td>
<td>QCR2</td>
<td>FN</td>
<td>ubiquinol cytochrome-c reductase core protein 2</td>
</tr>
<tr>
<td></td>
<td>YPL271W</td>
<td>ATP15</td>
<td>FN</td>
<td>ATP synthase epsilon subunit</td>
</tr>
<tr>
<td></td>
<td>YPL262W</td>
<td>FUM1</td>
<td>FP</td>
<td>fumarase</td>
</tr>
<tr>
<td></td>
<td>YML120C</td>
<td>NDH1</td>
<td>FP</td>
<td>mitochondrial NADH ubiquinone 6 oxidoreductase</td>
</tr>
<tr>
<td></td>
<td>YKL085W</td>
<td>MDH1</td>
<td>FP</td>
<td>mitochondrial malate dehydrogenase</td>
</tr>
<tr>
<td></td>
<td>YDL067C</td>
<td>COX9</td>
<td>FN</td>
<td>subunit VIIa of cytochrome c oxidase</td>
</tr>
<tr>
<td>Ribo</td>
<td>YPL037C</td>
<td>EGD1</td>
<td>FP</td>
<td>β subunit of the nascent-polypeptide-associated complex (NAC)</td>
</tr>
<tr>
<td></td>
<td>YLR406C</td>
<td>RPL31B</td>
<td>FN</td>
<td>ribosomal protein L31B (L34B) (YL28)</td>
</tr>
<tr>
<td></td>
<td>YLR075W</td>
<td>RPL10</td>
<td>FP</td>
<td>ribosomal protein L10</td>
</tr>
<tr>
<td></td>
<td>YAL033W</td>
<td>EF1α</td>
<td>FP</td>
<td>translation elongation factor EF-1β</td>
</tr>
<tr>
<td>Prot</td>
<td>YHR027C</td>
<td>RPN1</td>
<td>FN</td>
<td>subunit of 26S proteasome (PA700 subunit)</td>
</tr>
<tr>
<td></td>
<td>YGR270W</td>
<td>YTA7</td>
<td>FN</td>
<td>member of CDC48/PA5/SEC18 family of ATPases</td>
</tr>
<tr>
<td></td>
<td>YGR048W</td>
<td>UFD1</td>
<td>FP</td>
<td>ubiquitin fusion degradation protein</td>
</tr>
<tr>
<td></td>
<td>YDR069C</td>
<td>DOA4</td>
<td>FN</td>
<td>ubiquitin isopeptidase</td>
</tr>
<tr>
<td></td>
<td>YDL020C</td>
<td>RPN4</td>
<td>FN</td>
<td>involved in ubiquitin degradation pathway</td>
</tr>
<tr>
<td>Hist</td>
<td>YOL012C</td>
<td>HTA3</td>
<td>FN</td>
<td>histone-related protein</td>
</tr>
<tr>
<td></td>
<td>YKL049C</td>
<td>CSE4</td>
<td>FN</td>
<td>required for proper kinetochore function</td>
</tr>
</tbody>
</table>

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.
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.

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).

![SVM classification margins for ovarian tissues.](image)

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 N026. The second most negative point corresponds to tissue HVTBC3.
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
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.
#!/usr/bin/perl -w

# Storing DNA in a variable, and printing it out

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

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

# Finally, we'll specifically tell the program to exit.
extit;   #perl1.pl
#!/usr/bin/perl -w
$DNA1 = 'ACGGGAGGACGCGGAAAATTACTACGGCATTAGC';
$DNA2 = 'ATAGTGCCGTGAGAGTGATGTAGTA';
# Concatenate the DNA fragments
$DNA3 = "$DNA1$DNA2";
print "Concatenation 1):

$DNA3

"
# An alternative way using the "dot operator":
$DNA3 = $DNA1 . $DNA2;
print "Concatenation 2):

$DNA3

"
# 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

$rev

";
exit;  #perl2.pl
#!/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
#!/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

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

```perl
#!/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

```perl
#!/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;
```

```perl
#!/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;
```
#! /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
#!/usr/bin/perl -w

# define a DNA sequence object with given sequence
$seq = Bio::Seq->new('-seq'=>'actgtggcggtcaact',
    '-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
#!/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
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
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

```perl
#!/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 = $result->next_hit()){
    print "$hit 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

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>C</th>
<th>G</th>
<th>T</th>
</tr>
</thead>
<tbody>
<tr>
<td>+</td>
<td>0.180</td>
<td>0.274</td>
<td>0.426</td>
<td>0.120</td>
</tr>
<tr>
<td></td>
<td>0.171</td>
<td>0.368</td>
<td>0.274</td>
<td>0.188</td>
</tr>
<tr>
<td></td>
<td>0.161</td>
<td>0.339</td>
<td>0.375</td>
<td>0.125</td>
</tr>
<tr>
<td></td>
<td>0.079</td>
<td>0.355</td>
<td>0.384</td>
<td>0.182</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>C</th>
<th>G</th>
<th>T</th>
</tr>
</thead>
<tbody>
<tr>
<td>-</td>
<td>0.300</td>
<td>0.205</td>
<td>0.285</td>
<td>0.210</td>
</tr>
<tr>
<td></td>
<td>0.322</td>
<td>0.298</td>
<td>0.078</td>
<td>0.302</td>
</tr>
<tr>
<td></td>
<td>0.248</td>
<td>0.246</td>
<td>0.298</td>
<td>0.208</td>
</tr>
<tr>
<td></td>
<td>0.177</td>
<td>0.239</td>
<td>0.292</td>
<td>0.292</td>
</tr>
</tbody>
</table>
How to distinguish?

- **Compute**

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

<table>
<thead>
<tr>
<th>(r=p/m)</th>
<th>A</th>
<th>C</th>
<th>G</th>
<th>T</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>-0.740</td>
<td>0.419</td>
<td>0.580</td>
<td>-0.803</td>
</tr>
<tr>
<td>C</td>
<td>-0.913</td>
<td>0.302</td>
<td>1.812</td>
<td>-0.685</td>
</tr>
<tr>
<td>G</td>
<td>-0.624</td>
<td>0.461</td>
<td>0.331</td>
<td>-0.730</td>
</tr>
<tr>
<td>T</td>
<td>-1.169</td>
<td>0.573</td>
<td>0.393</td>
<td>-0.679</td>
</tr>
</tbody>
</table>

**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

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>C</th>
<th>G</th>
<th>T</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>0.180</td>
<td>0.274</td>
<td>0.426</td>
<td>0.120</td>
</tr>
<tr>
<td>C</td>
<td>0.171</td>
<td>0.368</td>
<td>0.274</td>
<td>0.188</td>
</tr>
<tr>
<td>G</td>
<td>0.161</td>
<td>0.339</td>
<td>0.375</td>
<td>0.125</td>
</tr>
<tr>
<td>T</td>
<td>0.079</td>
<td>0.355</td>
<td>0.384</td>
<td>0.182</td>
</tr>
</tbody>
</table>
CpG Island + in an ocean of –
First order Hidden Markov Model

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