Computer Science @ UKZN

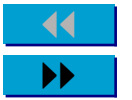
http://www.cs.ukzn.ac.za

Gene Spotting with SVMs

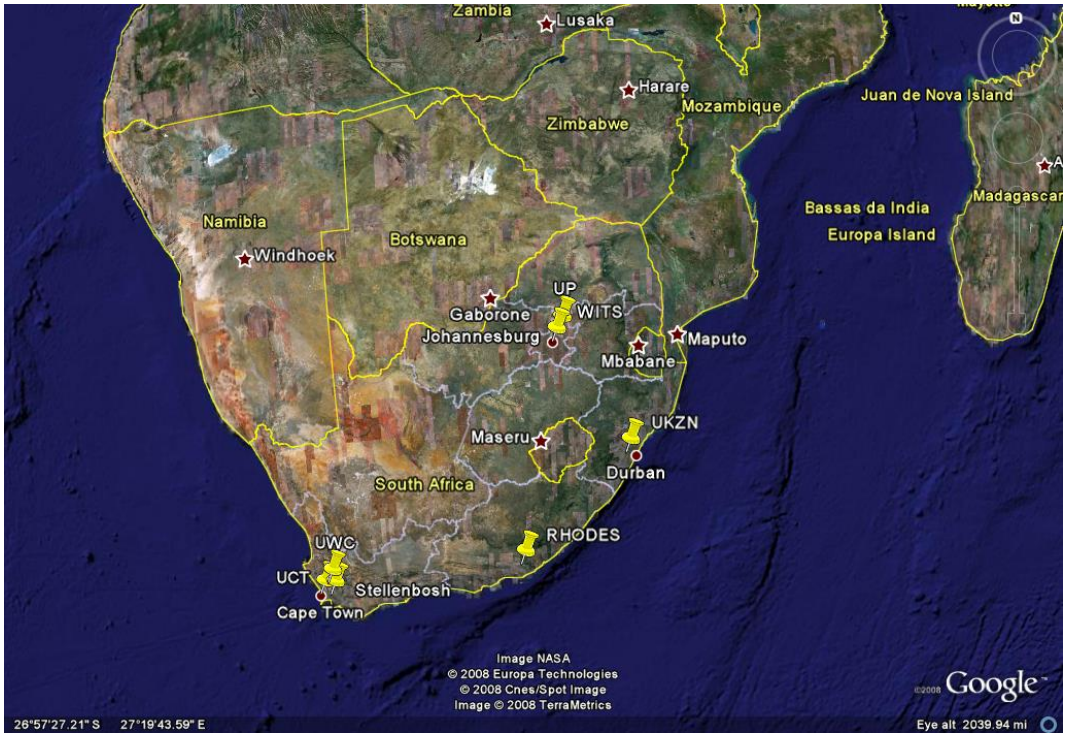
Hugh Murrell

murrellh@ukzn.ac.za

The University of KwaZulu-Natal is located in the cities of **Durban** and **Pietermaritzburg** on the eastern seaboard of **South Africa**



South African Universities



My Colleagues

Contents

- Support Vector Machines
- Bioinformatics
- Recognizing genetic material
- Boundary Detection

Support Vector Machines

Consider two *linearly separable* sets of data points, X_- and X_+, from an inner product space.

For any projection direction $\hat{w}$ find the two points x_-^w and x_+^w (one from each set) that minimizes $\langle \hat{w}, x_+^w - x_-^w \rangle$.

An optimal projection direction $\hat{\alpha}$ satisfies:

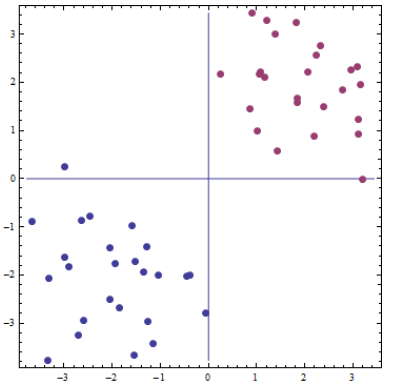
$$\langle \hat{\alpha}, x_+^\alpha - x_-^\alpha \rangle \geq \langle \hat{w}, x_+^w - x_-^w \rangle$$

Once the optimum projection direction $\hat{\alpha}$ has been found, new data points x are classified via the sign of: $\langle \hat{\alpha}, x \rangle + \beta$

where β is chosen so that $\langle \hat{\alpha}, x \rangle + \beta = 0$ for any x midway between x_-^α and x_+^α.

A simple example in 2D

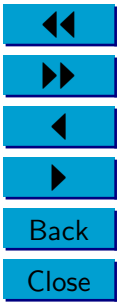
```
len = 50;
Xn = RandomReal[NormalDistribution[-2, 1], {len/2, 2}];
Xp = RandomReal[NormalDistribution[2, 1], {len/2, 2}];
X = Join[Xn, Xp];
y = Join[Table[-1, {len/2}], Table[+1, {len/2}]];
```



A Rudimentary SVM

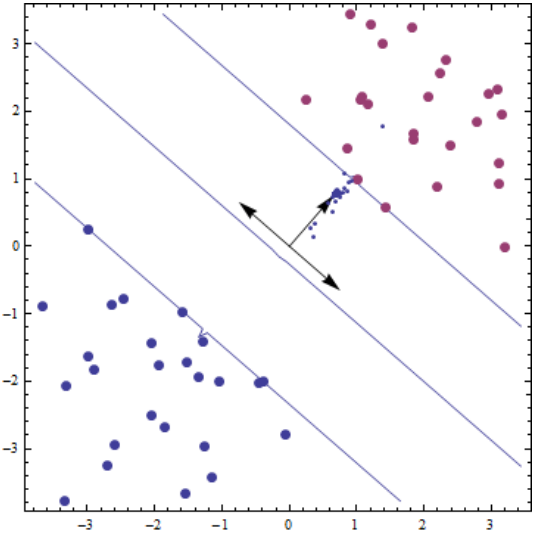
In two dimensions, we can make a rudimentary attempt at this computation by employing Mathematica's built in `NMaximum` function to optimize the following objective function.

```
Obj[X_, y_, {x1_Real, x2_Real}] :=
 Module[{Xn, Xp, PXn, PXp, Sn, Ln, Sp, Lp, alpha},
  Xn = Extract[X, Position[y, -1]];
  Xp = Extract[X, Position[y, 1]];
  PXn = Map[Inner[Times, {x1, x2}, #, Plus] &, Xn];
  PXp = Map[Inner[Times, {x1, x2}, #, Plus] &, Xp];
  Sn = Min[PXn];
  Ln = Max[PXn];
  Sp = Min[PXp];
  Lp = Max[PXp];
  alpha = 0;
  If[Ln < Sp, obj = (Sp - Ln),
   If[Lp < Sn, obj = (Sn - Lp),
    alpha = 0]];
  alpha
  ]
```

Maximum Separation

```
NMaximize[{Obj[X, y, {x1, x2}], x1^2 + x2^2 == 1}, {x1, x2}]
```



A robust implementation

Note that if we choose β and scale α so that

$$\langle \alpha, x_- \rangle + \beta = -1$$
$$\langle \alpha, x_+ \rangle + \beta = +1$$

then the width of the separating margin is given by:

$$\langle \frac{\alpha}{\|\alpha\|}, (x_+ - x_-) \rangle = \frac{2}{\|\alpha\|}$$

The Primal Problem

Thus to find the optimal separating hyperplane one solves the following optimization problem:

$$\begin{aligned} \text{minimize} \quad & \frac{1}{2}\|\alpha\|^2 \\ \text{subject to} \quad & y_i(\langle \alpha, x_i \rangle + \beta) \geq 1 \end{aligned}$$

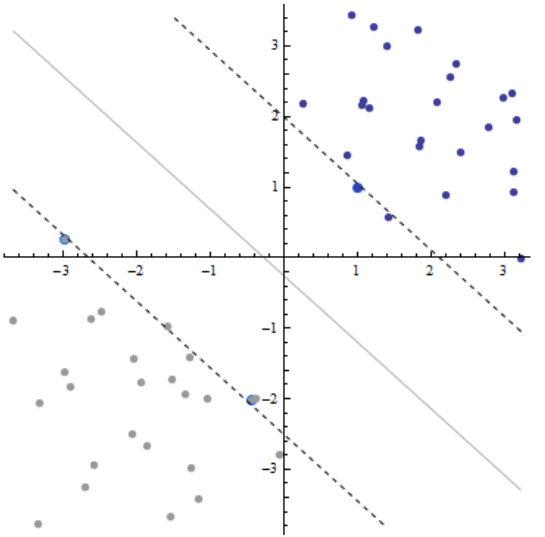
In the SVM literature this is referred to as the *primal* problem and is usually converted into a *dual* problem suitable for a quadratic programming solution:

see: Nilsson, Bjorkegren and Tegner, and the *MathSVM* package for details.

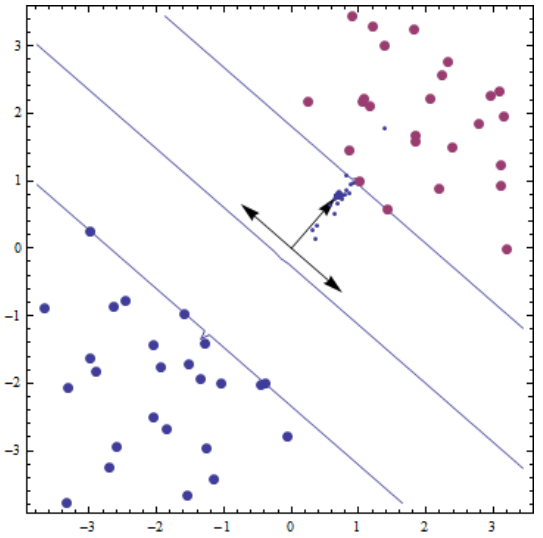
Maximum Separation with *MathSVM*

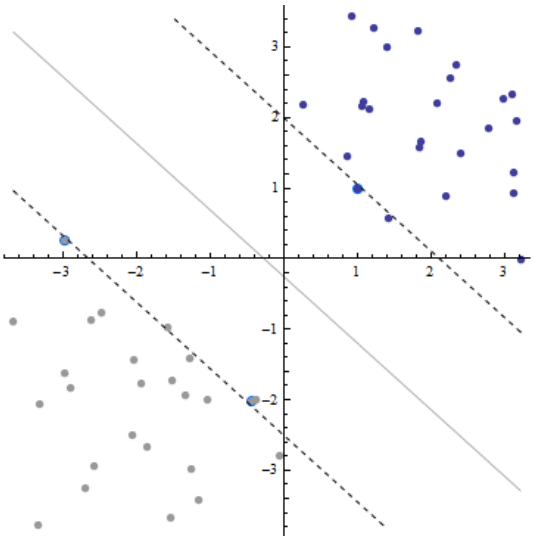
Our two dimensional example can be separated with maximum margin via the commands:

```
<< MathSVM‘
tol = 0.1;
alpha = SeparableSVM[X, y, tol];
SVMPlot[alpha,X,y]
```



Comparison:





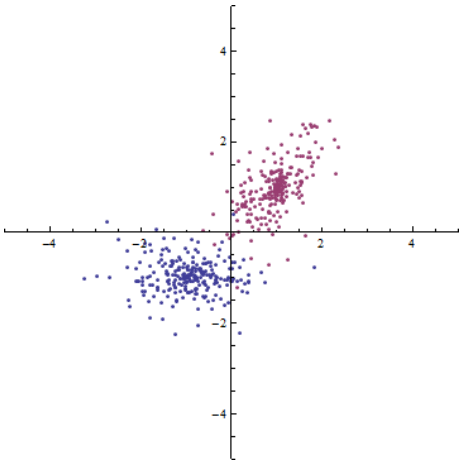
Non Separable Classes

Nilsson and colleagues extend the SVM method to classes that are **not** separable by including a parameter C that determines how hard points violating the boundary contraints are penalized. The primal problem for the nonseparable situation is formulated as:

$$\begin{aligned} \text{minimize} \quad & \frac{1}{2}\|\alpha\|^2 + C\sum_i \xi_i \\ \text{subject to} \quad & y_i(\langle \alpha, x_i \rangle + \beta) \geq 1 - \xi_i, \\ \text{and} \quad & \xi_i \geq 0 \end{aligned}$$

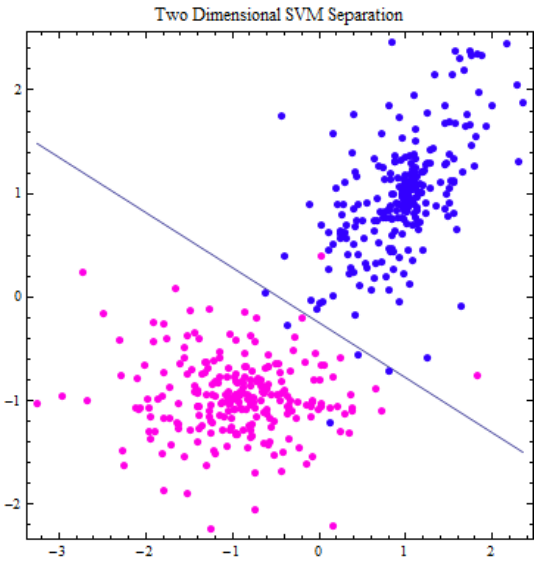
Non Separable Data

```
len = 500;
EllipsePoint[] :=
  Module[{},
    t = RandomReal[{0, Pi}];
    r = RandomReal[NormalDistribution[0, 0.5]];
    {2 r Cos[t], r Sin[t]}
  ];
Xn = Table[EllipsePoint[] - {1, 1}, {len/2}];
rm = RotationMatrix[Pi/3];
Xp = Table[rm.EllipsePoint[] + {1, 1},{len/2}];
mu = Mean[Join[Xn, Xp]];
Xn = Map[# - mu &, Xn]; Xp = Map[# - mu &, Xp];
X = Join[Xn, Xp]; y = Join[Table[-1, {len/2}],Table[+1, {len/2}]];
ListPlot[{Xn, Xp}, PlotRange -> {{-5,5}, {-5, 5}},AspectRatio -> 1]
```



SVM separation

```
alpha = NonseparableSVM[X, y, 1.0, 0.1];
beta = Bias[alpha, X, y]
ContourPlotSVM[X, y, alpha, beta]
```



Bioinformatics (A simplified version)

DNA is a string (of nucleotides) over the 4-character alphabet:

$$\{ACGT\}$$

RNA is a string (of nucleotides) over the 4-character alphabet:

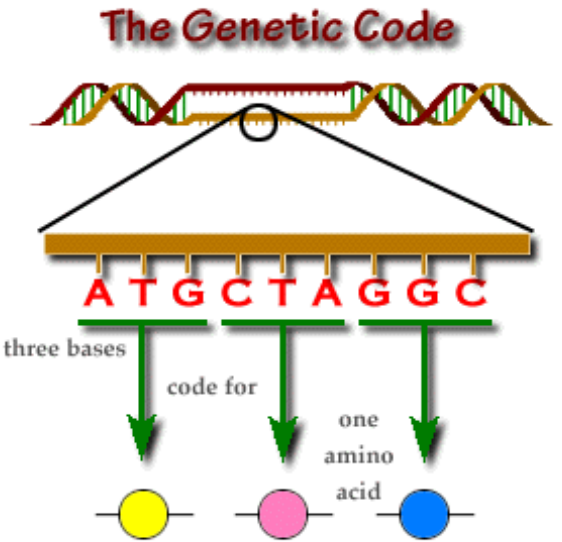
$$\{ACGU\}$$

Proteins are strings (of amino acids) over the 20-character alphabet:

$$\{ARDNCEQGHILKMFPSTWYV\}$$

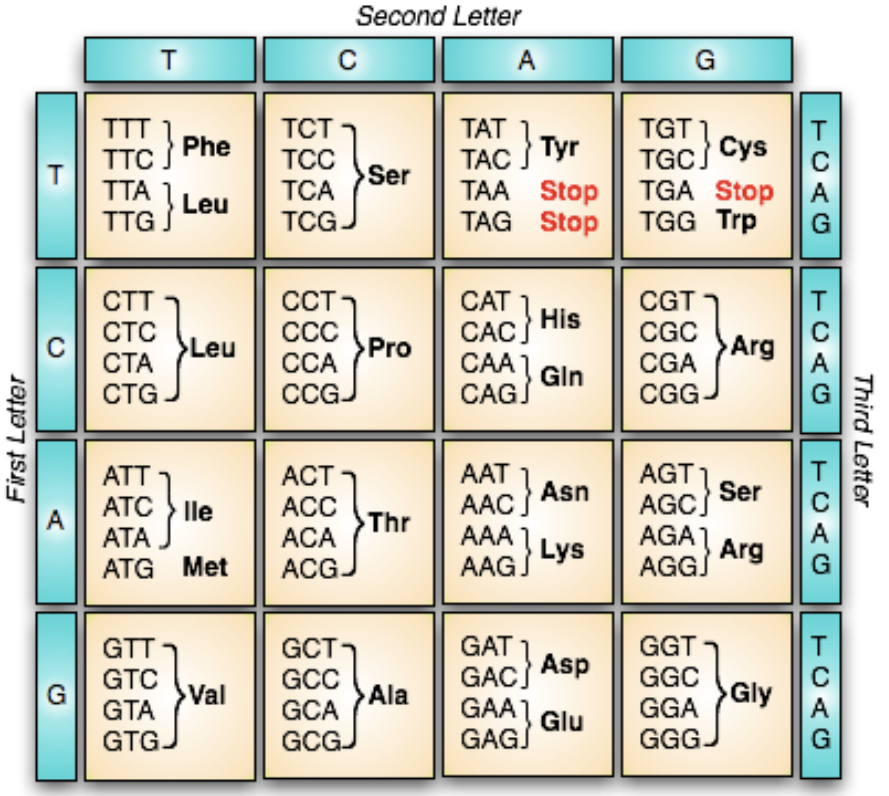
$$DNA \Longrightarrow RNA \Longrightarrow Protein$$

Central Dogma:



Substrings in the DNA known as **genes** code for Proteins, by translating nucleotide triplets to amino acids. The translation table is known as the **genetic code**.

The Genetic Code



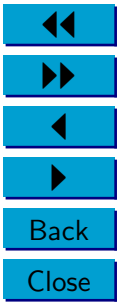
Can we use the Support Vector Machine to classify DNA segments as intron or exon?

The Berkeley Genome Project at `www.fruitfly.org` provides two flat-files that were used to test and train the human gene-finding algorithm of GENIE.

File `1754introns` contains 1754 intron DNA sequences

File `2107exons` contains 2107 exon DNA sequences.

The first line of each record contains a gene identifier and the exon/intron position within the gene. Subsequent lines contain the neucleotide string.



Evaluating Neucleotide Base Frequencies

First we try to train a SVM to classify sequences by comparing neucleotide frequencies within the string.

```
bases = "ACGT"

baseFreq[dnaStr_] :=
  Map[StringCount[dnaStr, #] &, Characters[bases]] /
    StringLength[dnaStr]

baseFreq[exonSeq[[1]]]
{12/55, 22/75, 218/825, 37/165}
```

The zCurve map

The four frequency counts are not independent, their sum is always unity.

To get independent quantities we follow use Zhang's so-called `zCurve` map from R^4 to R^3 given by:

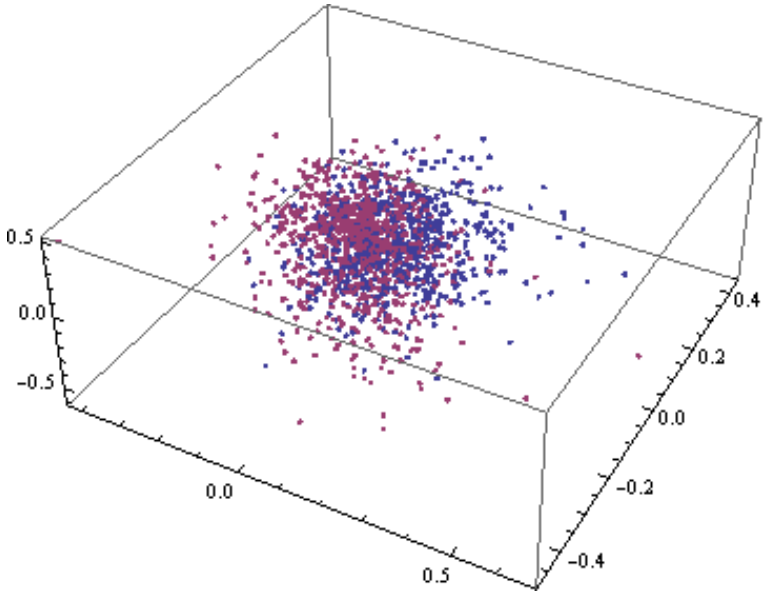
```
zCurve[{a_, c_, g_, t_}] :=
  {(a + g) - (c + t),
   (a + c) - (g + t),
   (a + t) - (g + c)}

zCurve[baseFreq[exonSeq[[1]]]]
{-29/825, 19/825, -19/165}
```





Generating SVM Training Data

```
featureVector3[str_] := zCurve[baseFreq[str]]
exonTrainF3 = Map[featureVector3, exonTrain];
intronTrainF3 = Map[featureVector3,intronTrain];
ListPointPlot3D[{exonTrainF3, intronTrainF3}]
```



A Script to Train and Test a SVM

```
SVMTrainAndTest[exonTrain_, intronTrain_, exonTest_, intronTest_,
 featureVector_, kf_] :=
  Module[{x, y, alpha, b, positives, negatives, cp, cn,
    sensitivity, specificity, accuracy, label},
  x = N[Join[Map[featureVector, exonTrain],Map[featureVector, intronTrain]]];
  y = Join[Table[1, {Length[exonTrain]}],Table[-1, {Length[intronTrain]}]];
  alpha = NonseparableSVM[x, y, 1.0, 0.1, KernelFunction -> kf];
  b = Bias[alpha, x, y, KernelFunction -> kf];
  positives = Map[featureVector, exonTest];
  negatives = Map[featureVector, intronTest];
  cp = Map[SVMClassify[kf, x, alpha, y, b, #] &, positives];
  cn = Map[SVMClassify[kf, x, alpha, y, b, #] &, negatives];
  sensitivity = N[Length[Select[cp, # > 0 &]]/Length[cp], 2];
  specificity = N[Length[Select[cn, # < 0 &]]/Length[cn], 2];
  accuracy = N[(Length[Select[cp, # > 0 &]] +
    Length[Select[cn, # < 0 &]])/(Length[cp] + Length[cn]), 2];
  label = StringJoin["sens = ", ToString[sensitivity], " spec =",
    ToString[specificity], " acc =", ToString[accuracy]];
  ListPlot[{Reverse[Sort[cp]], Sort[cn]},PlotLabel ->
    StringJoin["Support Vector Machine \n KF = ", ToString[kf],
      "\n FV dim = ", ToString[Length[First[x]]]], Frame -> True,
    FrameLabel -> {label, ""}]]
```

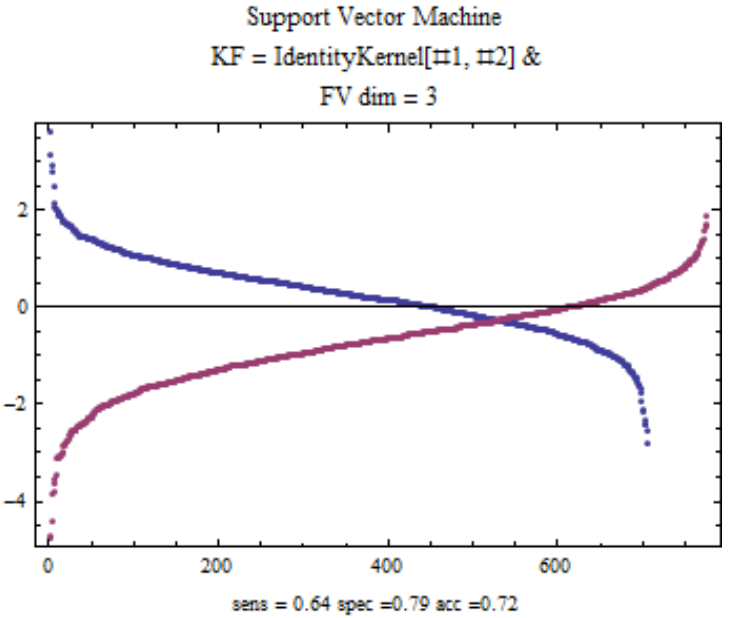




Back
Close

Base Frequency Separation

```
ik = IdentityKernel[#1, #2] &;
SVMTrainAndTest[exonTrain, intronTrain,
  exonTest, intronTest, featureVector3, ik]
```

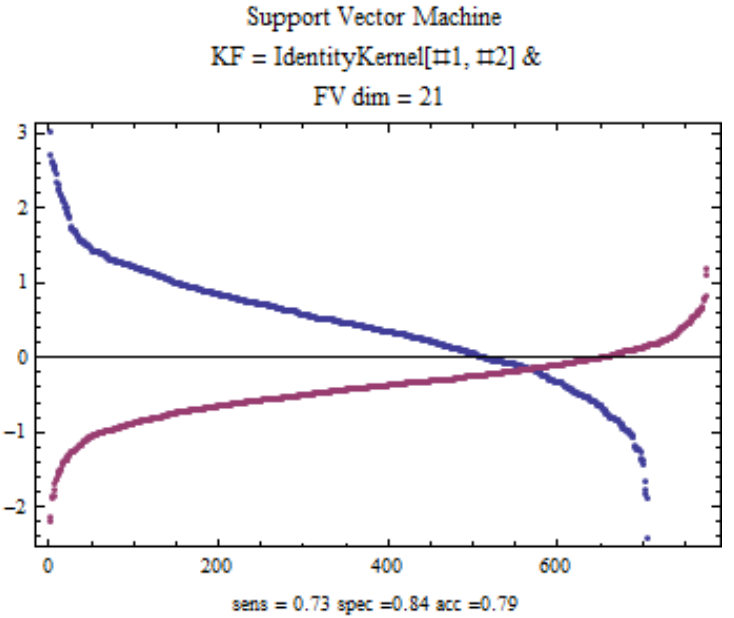


Amino Acid Frequencies


```
aminoAcids = "ACDEFGHIKLMNPQRSTVWY#"

CodonRules =
{"tca" -> "S", "tcc" -> "S", "tcg" -> "S", "tct" -> "S",
 "ttc" -> "F", "ttt" -> "F", "tta" -> "L", "ttg" -> "L",
 "tac" -> "Y", "tat" -> "Y", "taa" -> "#", "tag" -> "#",
 "tgc" -> "C", "tgt" -> "C", "tga" -> "#", "tgg" -> "W",
 "cta" -> "L", "ctc" -> "L", "ctg" -> "L", "ctt" -> "L",
 "cca" -> "P", "ccc" -> "P", "ccg" -> "P", "cct" -> "P",
 "cac" -> "H", "cat" -> "H", "caa" -> "Q", "cag" -> "Q",
 "cga" -> "R", "cgc" -> "R", "cgg" -> "R", "cgt" -> "R",
 "ata" -> "I", "att" -> "I", "atc" -> "I", "atg" -> "M",
 "aca" -> "T", "acc" -> "T", "acg" -> "T", "act" -> "T",
 "aac" -> "N", "aat" -> "N", "aaa" -> "K", "aag" -> "K",
 "agc" -> "S", "agt" -> "S", "aga" -> "R", "agg" -> "R",
 "gta" -> "V", "gtc" -> "V", "gtg" -> "V", "gtt" -> "V",
 "gca" -> "A", "gcc" -> "A", "gcg" -> "A", "gct" -> "A",
 "gac" -> "D", "gat" -> "D", "gaa" -> "E", "gag" -> "E",
 "gga" -> "G", "ggc" -> "G", "ggg" -> "G", "ggt" -> "G"};

 dnaTranslate[dna_, readingFrame_] := StringJoin[
   Partition[Drop[Characters[dna], readingFrame - 1],3] /. {x_, y_, z_} :>
     StringReplace[ StringJoin[x, y, z] /. CodonRules, _ ~~ _ ~~ _ -> "?"]]

featureVector21[dnaStr_] := Map[StringCount[dnaTranslate[dnaStr, 1], #] &,
 Characters[aminoAcids]] / (StringLength[dnaStr]/3)

featureVector21[exonSeq[[1]]]
{3/55, 4/275, 2/55, 1/25, 16/275, 19/275, 7/275, 14/275, 12/275, 6/55,
 8/275, 3/55, 19/275, 1/25, 14/275, 26/275, 2/55, 21/275, 4/275, 9/275, 0}
```

Amino Acid Separation

```
SVMTrainAndTest[exonTrain, intronTrain,
  exonTest, intronTest, featureVector21, ik]
```



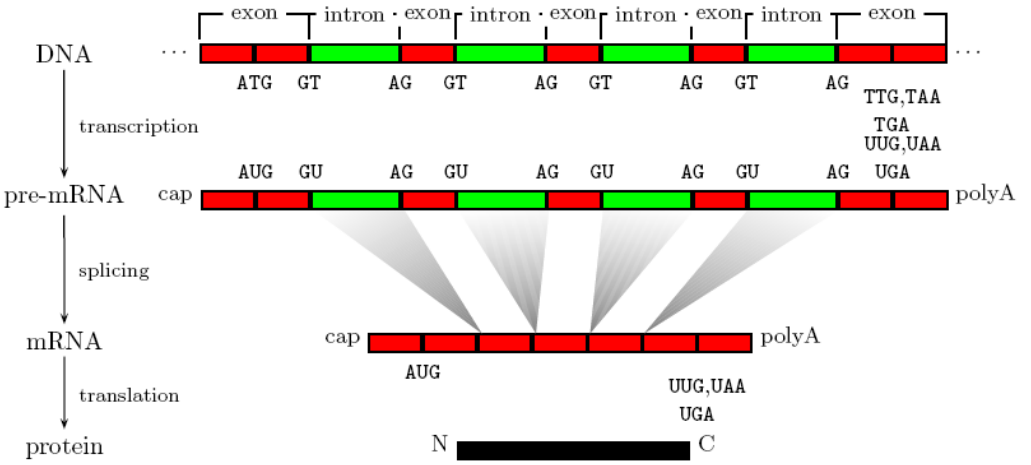
Alternative Feature Vectors

In the paper experiments with various feature vector formulations are conducted. The results are shown in the table below:

Structure	Frame Dept	Dim	Acc
Neucleotide	No	3	72%
Neucleotide	Yes	9	75%
Di-Neucleotide	Yes	24	84%
Amino Acid	Yes	21	79%
All of the above	Yes & No	57	86%

Splice Site Detection

Splice sites are locations within a gene that mark exon-intron boundaries.



Splice sites are either of type donor (if they mark an exon to intron boundary) or of type acceptor (if they mark an intron to exon boundary).

Donor sites occur at a GT pair whereas acceptor sites occur at an AG pair. Not all occurrences of these pairs enforce a splice. In fact very few of them do. For a splice to occur certain motifs must be present up and down stream.

Towards a gene predictor

In a forthcoming paper we have implemented ideas from Ratsch and Sonnenburg that recognize donor and acceptor sites with an accuracy of 92%. What remains to be seen is:

How accurate will a gene predictor be using a

- **content recognizer that is 85% accurate**

together with a

- **splice-site detector that is 92% accurate.**




Selected References

- R. Nilsson, J. Bjorkegren and J. Tegner, *A Flexible Implementation for Support Vector Machines*, The *Mathematica* Journal, 10(1), 2006, pp 114-127.
- B. Higgins, *Using Mathematica to Teach Bioinformatic Skills*, *Mathematica* Developer Conference, Champaign, 2003, http://library.wolfram.com/infocenter/Conferences/4895/
- CT. Zhang, J. Wang and R. Zhang, *Using a Euclid distance discriminant method to find protein coding genes in the yeast genome*, Computers and Chemistry, 26, 2002, pp. 195206.
- G. Ratsch and S Sonnenburg, *Accurate Splice Site Detection*, Kernel Methods in Computational Biology, ed: Scholkoph, Tsuda and Vert, MIT press, 2004.
- H. Murrell, K. Hashimoto and D. Takatori, *Fisher Discrimination with Kernels*, submitted to The *Mathematica* Journal.