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 \textit{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 $\beta$ is chosen so that $\langle \hat{\alpha}, x \rangle + \beta = 0$ for any $x$ midway between $x^\alpha_-$ and $x^\alpha_+$. 
A simple example in 2D

\[
\begin{align*}
\text{len} & = 50; \\
X_n & = \text{RandomReal}[\text{NormalDistribution}[-2, 1], \{\text{len}/2, 2\}]; \\
X_p & = \text{RandomReal}[\text{NormalDistribution}[2, 1], \{\text{len}/2, 2\}]; \\
X & = \text{Join}[X_n, X_p]; \\
y & = \text{Join}[\text{Table}[-1, \{\text{len}/2\}], \text{Table}[+1, \{\text{len}/2\}]]; \\
\end{align*}
\]
A Rudimentary SVM

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

\[
\text{Obj}[X_\_, y_\_, \{x1_\text{Real}, x2_\text{Real}\}] := \\
\text{Module}[\{Xn, Xp, PXn, PXp, Sn, Ln, Sp, Lp, alpha\}, \\
Xn = \text{Extract}[X, \text{Position}[y, -1]]; \\
Xp = \text{Extract}[X, \text{Position}[y, 1]]; \\
PXn = \text{Map}[\text{Inner}[\times, \{x1, x2\}, \#, \text{Plus}] \&, Xn]; \\
PXp = \text{Map}[\text{Inner}[\times, \{x1, x2\}, \#, \text{Plus}] \&, Xp]; \\
Sn = \text{Min}[PXn]; \\
Ln = \text{Max}[PXn]; \\
Sp = \text{Min}[PXp]; \\
Lp = \text{Max}[PXp]; \\
alpha = 0; \\
\text{If}[Ln < Sp, obj = (Sp - Ln), \\
\text{If}[Lp < Sn, obj = (Sn - Lp), \\
alpha = 0]];
\]

alpha
Maximum Separation

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

Note that if we choose $\beta$ and scale $\alpha$ so that

$$\langle \alpha, x_- \rangle + \beta = -1 \quad \text{and} \quad \langle \alpha, x_+ \rangle + \beta = +1$$

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:

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

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:

Maximum Separation with *MathSVM*

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

```math
<< 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 constraints are penalized. The primal problem for the nonseparable situation is formulated as:

\[
\begin{align*}
\text{minimize} & \quad \frac{1}{2} \| \alpha \|^2 + C \sum_i \xi_i \\
\text{subject to} & \quad y_i (\langle \alpha, x_i \rangle + \beta) \geq 1 - \xi_i, \\
\text{and} & \quad \xi_i \geq 0
\end{align*}
\]
Non Separable Data

```mathematica
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:

\( \{ARDNCEQGHILKMFPSWYV\} \)

\[ DNA \implies RNA \implies Protein \]
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

<table>
<thead>
<tr>
<th>First Letter</th>
<th>Second Letter</th>
<th>Third Letter</th>
</tr>
</thead>
<tbody>
<tr>
<td>T</td>
<td>TTT {Phe}</td>
<td>TGT {Cys}</td>
</tr>
<tr>
<td></td>
<td>TTC {Leu}</td>
<td>TGC {Trp}</td>
</tr>
<tr>
<td>C</td>
<td>CTT {Leu}</td>
<td>CTT {Leu}</td>
</tr>
<tr>
<td></td>
<td>CCT {Pro}</td>
<td>CCG {Gln}</td>
</tr>
<tr>
<td>A</td>
<td>ATT {Ile}</td>
<td>AGT {Ser}</td>
</tr>
<tr>
<td></td>
<td>ATA {Ile}</td>
<td>AGA {Arg}</td>
</tr>
<tr>
<td>G</td>
<td>GTT {Val}</td>
<td>GGT {Gly}</td>
</tr>
<tr>
<td></td>
<td>GTC {Val}</td>
<td>GCC {Ala}</td>
</tr>
</tbody>
</table>

Stop codons are indicated in red.
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 nucleotide string.
Evaluating Neucleotide Base Frequencies

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

```math
\text{bases} = \"ACGT\"
```

```math
\text{baseFreq}[\text{dnaStr}_\_] := \\
\text{Map}[\text{StringCount}[\text{dnaStr}, \#] \&, \text{Characters}[\text{bases}]] / \text{StringLength}[\text{dnaStr}]
```

```math
\text{baseFreq}[\text{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:

\[
zCurve[{a_, c_, g_, t_}] := 
\begin{align*}
&((a + g) - (c + t)), \\
&(a + c) - (g + t), \\
&(a + t) - (g + c)}
\end{align*}
\]

\[
zCurve[baseFreq[exonSeq[[1]]]] = \{-29/825, 19/825, -19/165}\]
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, ""}]}
Base Frequency Separation

\[ ik = \text{IdentityKernel}[#1, #2] \&; \]
\[ \text{SVMTrainAndTest}[\text{exonTrain}, \text{intronTrain}, \]
\[ \text{exonTest}, \text{intronTest}, \text{featureVector3}, ik] \]
Amino Acid Frequencies

aminoAcids = "ACDEFGHIKLMNPQRSTVWY#"

CodonRules =

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:

<table>
<thead>
<tr>
<th>Structure</th>
<th>Frame Dept</th>
<th>Dim</th>
<th>Acc</th>
</tr>
</thead>
<tbody>
<tr>
<td>Neucleotide</td>
<td>No</td>
<td>3</td>
<td>72%</td>
</tr>
<tr>
<td>Neucleotide</td>
<td>Yes</td>
<td>9</td>
<td>75%</td>
</tr>
<tr>
<td>Di-Neucleotide</td>
<td>Yes</td>
<td>24</td>
<td>84%</td>
</tr>
<tr>
<td>Amino Acid</td>
<td>Yes</td>
<td>21</td>
<td>79%</td>
</tr>
<tr>
<td>All of the above</td>
<td>Yes &amp; No</td>
<td>57</td>
<td>86%</td>
</tr>
</tbody>
</table>
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


