Classification of genome data with *n*-gram models

Anton Antonov
MathematicaForPrediction at WordPress
MathematicaForPrediction at GitHub
January 2014

Introduction

In this document we consider the following problem:

Gene Sequence Classification Problem (GSCP): Given two genes, G_1 and G_2, and a (relatively short) sub-sequence S from one of them tell from which gene the sub-sequence S is part of.

One way to derive a solution for this problem is to model each gene with an n-gram model, [2], and classify the sub-sequences according to the risk ratio or odds ratio test, [3], based on the Markov chain state transition probabilities. We are going to make classification experiments for different n's of the n-gram model and use a type of Receiver Operating Characteristic (ROC) plot, [4], to judge which n is the best.

This approach can be also applied to text classification. We consider genes for conciseness and clarity.

This document demonstrates the usage of the Mathematica package for n-gram Markov chain models provided by the MathematicaForPrediction project at GitHub, see [1].

Mathematica has the function `GenomeData` that gives the DNA sequences for specified genes.

The approach

Our approach to answer GSCP is to define the algorithm named NGramClassifier(k) that takes as parameter $k \in \mathbb{N}$, $k > 2$, and then conduct a series of experiments to find the best k.

The algorithm NGramClassifier(k)

1. For each gene G_1 and G_2 create a k-gram, $(k-1)$ order Markov chain model. Denote them NGM_1 and NGM_2 respectively. NGM_i tells what is the probability the sequence of characters $c_1\, c_2 \ldots c_{k-1}$ to be followed by the character c_k in G_i. We have $c_j \in \{A, C, G, T\}$, $j \in [1, k]$, $j \in \mathbb{N}$.

1.1. Each of the models i has a Markov chain state transition matrix M_i.

2. Partition the sub-sequence S into k-grams. I.e. if $S := \{c_1, c_2, c_3, \ldots, c_m\}$ form the set

$$g_k(S) := \{\{c_1, \ldots, c_k\}, \{c_2, \ldots, c_{k+1}\}, \{c_3, \ldots, c_{k+2}\}, \ldots, \{c_{m-k+1}, \ldots, c_m\}\}. \tag{1}$$

3. To each element of $g_k(S)$ we apply NGM_1 to obtain the sequence of probabilities:

$$\mathrm{cp}_1 := \{p_1^1, p_2^1, \ldots, p_{m-k}^1\}. \tag{2}$$

4. To each element of $g_k(S)$ we apply NGM_2 to obtain the sequence of probabilities:

$$\mathrm{cp}_2 := \{p_1^2, p_2^2, \ldots, p_{m-k}^2\}. \tag{3}$$

5. We compute the estimated sequence appearance probabilities ratio test

$$\mathrm{spr}(S) := \prod_{i=1}^{m-k} \frac{p_i^1}{p_i^2}. \tag{4}$$

6. If $\mathrm{spr}(S) > 1$ we consider S to be part of the gene G_1. If $\mathrm{spr}(S) < 1$ we consider S to be part of G_2.

Remark: The numerator of (4) estimates the the probability the sequence S to appear in the gene G_1 using the Markov chain model NGM_1. Similarly, the denominator of (4) estimates the probability of S to appear in G_2 using NGM_2. The equation (4) can be seen as the risk ratio statistic, [3].

Remark: Instead of probabilities ratio in (4) we can use the odds ratio statistic, [3], defined for algorithm above with the following formulas:

$$\begin{aligned} \omega_1(S) &:= \prod_{i=1}^{m-k} p_i^1, \\ \omega_2(S) &:= \prod_{i=1}^{m-k} p_i^2, \\ \mathrm{or}(S) &:= \frac{\omega_1(S)/(1-\omega_1(S))}{\omega_2(S)/(1-\omega_2(S))}. \end{aligned} \tag{5}$$

Tuning -- finding the best k for NGramClassifier(k)

1. For each $k \in [1, \ldots, 10]$ calculate k-gram models for G_i. For these models use only the first 80% of the gene sequences. (In other words, for each G_i the training set is only 80% of G_i's length.)

2. Using the last 20% of G_1 and G_2 derive a set of test sequences T by randomly picking G_i and the sub-sequence length within the range $\{l_{\min}, l_{\max}\}$.

3. For each $k \in [1, \ldots, 10]$ calculate the classifications of the elements of T using NGramClassifier(k) with the models calculated in step 1.

4. Calculate the True Positive Rate (TPR) and False Positive Rate (FPR) for each k in step 3.

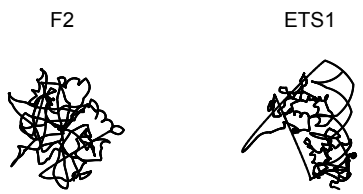
5. Plot the points with coordinates $\{\{\mathrm{TPR}_k, \mathrm{TPR}_k\}\}_{k=1}^{10}$ and select the best k.

Experiments

The genes

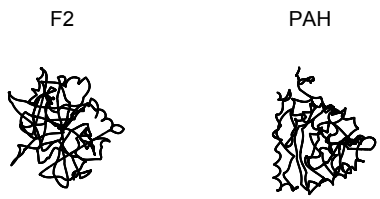
We are going to answer GSCP using the pair of genes “F2” and “ETS1”:

Out[1]=



and the pair of genes “F2” and “PAH”:

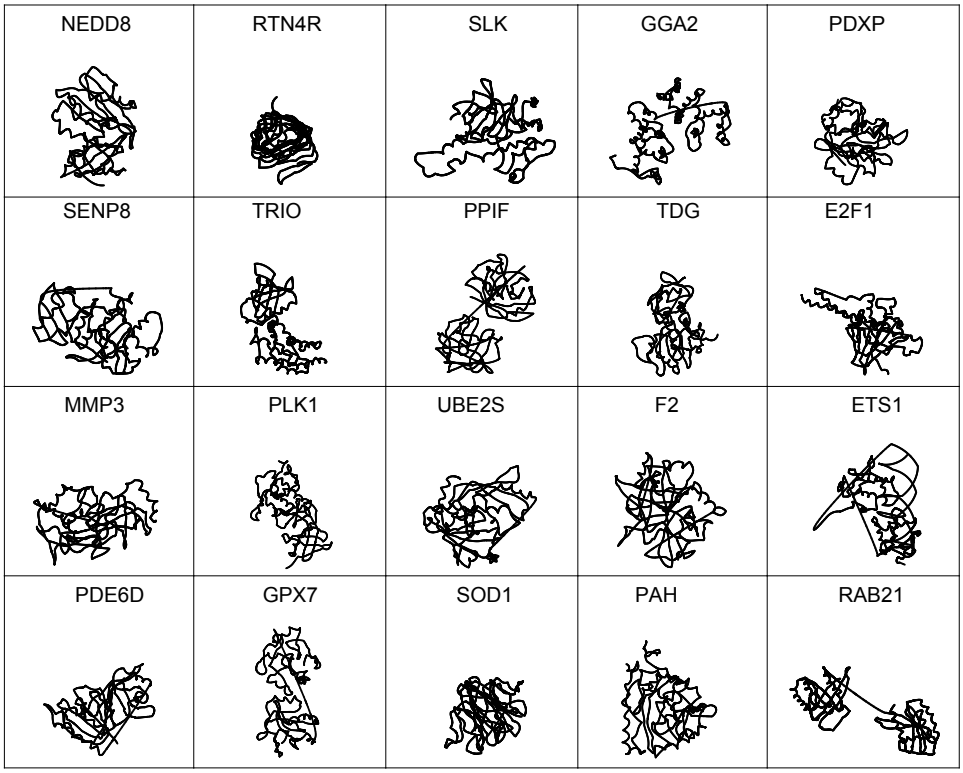
Out[3]=



The genes were picked from this human genome data table:

In[61]:=
```
GraphicsGrid[Partition[Graphics3D[{Black, Thickness[0.01], BezierCurve[
      ProteinData[#, "AtomPositions", "Residue"][[All, 2]]]}, Boxed → False,
    ImageSize → 100, PlotLabel → #, Method → {"ShrinkWrap" → True}] & /@
  {"NEDD8", "RTN4R", "SLK", "GGA2", "PDXP", "SENP8", "TRIO", "PPIF",
   "TDG", "E2F1", "MMP3", "PLK1", "UBE2S", "F2", "ETS1", "PDE6D",
   "GPX7", "SOD1", "PAH", "RAB21"}, 5], ImageSize → 500, Dividers → All]
```

Out[61]=



For more details and examples see the Mathematica function GenomeData.

Package load

Load the package [1]:

In[6]:=
```
Import[
 "https://raw.githubusercontent.com/antononcube/MathematicaForPrediction/master
   /NGramMarkovChains.m"]
```

Data and models

Each *n*-gram model has an array representing the *n*-gram probabilities and rules of converting gene component letters to indices. *Mathematica* has the symbol `DiscreteMarkovProcess` that has similar but different function and representation. The *n*-gram models built with [1] use high dimensional arrays instead of matrices. (Matrices are used only for the 2-gram models.)

For example,

In[7]:=
```
NGramMarkovChainModel[
 Characters[GenomeData["F2", "FullSequence"]], 2, "ColumnStochastic" → True]
```

Out[7]=
```
NGramModel[SparseArray[ Specified elements: 64 Dimensions: {4, 4, 4} ],
 Dispatch[ Length: 4 ], Dispatch[ Length: 4 ]]
```

The following commands built the *n*-gram models for each of the genes.

In[8]:=
```
geneName1 = "F2";
sGeneSeq1 = GenomeData[geneName1, "FullSequence"];
StringLength[sGeneSeq1]
```

Out[10]= 20 301

In[11]:=
```
Clear[ngm1]
AbsoluteTiming[
 Do[
  ngm1[morder] = NGramMarkovChainModel[Characters[
      StringTake[sGeneSeq1, {1, 10 000}]], morder, "ColumnStochastic" → True],
  {morder, 1, 10}]
]
```

Out[12]= {221.515, Null}

In[13]:=
```
geneName2 = "ETS1";
sGeneSeq2 = GenomeData[geneName2, "FullSequence"];
StringLength[sGeneSeq2]
```

Out[15]= 63 502

In[16]:=
```
Clear[ngm2]
AbsoluteTiming[
 Do[
  ngm2[morder] = NGramMarkovChainModel[Characters[
      StringTake[sGeneSeq2, {1, 10 000}]], morder, "ColumnStochastic" → True],
  {morder, 1, 10}]
]
```

Out[17]= {236.101, Null}

In[18]:=
```
geneName3 = "PAH";
sGeneSeq3 = GenomeData[geneName3, "FullSequence"];
StringLength[sGeneSeq3]
```

Out[20]= 79 278

In[21]:=
```
Clear[ngm3]
AbsoluteTiming[
 Do[
  ngm3[morder] = NGramMarkovChainModel[Characters[
      StringTake[sGeneSeq3, {1, 10 000}]], morder, "ColumnStochastic" → True],
  {morder, 1, 10}]
]
```

Out[22]= {230.786, Null}

Classification example

Here is an example of classification using steps 3,4, and 5 of the algorithm NGramClassifier described above.

In[23]:=
```
sampleToGuess = Characters[sGeneSeq1][[11 200 ;; 11 200 + 100]];
StringJoin@sampleToGuess
```

Out[24]= AGCAGGCATATCTAGGGGAGGAGCACCGCAGGCTGGGGGCATGGCAGGCACTAAGGCCCTGAGGTGGGAGCACTCTTG
GCTTGTCTGGGGAGCAGTAGGGA

In[25]:=
```
(* step 3 *)
cp1 =
  ngm1[2][[1]][[Sequence @@ (# /. ngm1[2][[2]])]] & /@ Partition[sampleToGuess, 2 + 1, 1];
(* step 4 *)
cp2 =
  ngm2[2][[1]][[Sequence @@ (# /. ngm2[2][[2]])]] & /@ Partition[sampleToGuess, 2 + 1, 1];
(* step 5 *)
Apply[Times, cp1 / cp2]
"sub-sequence of " <> If[% >= 1, geneName1, geneName2]
```

Out[27]= 269.471

Out[28]= sub-sequence of F2

Classification tuning runs F2 vs. ETS1

Generate test sub-sequences:

In[29]:=
```
gTests12 =
  Table[(
    {offset, slen, gseq} = Flatten@
      {RandomInteger[Floor /@ {0.8 Min[StringLength /@ {sGeneSeq1, sGeneSeq2}],
          0.95 Min[StringLength /@ {sGeneSeq1, sGeneSeq2}]}, 1],
       RandomInteger[{100, 120}, 1], RandomChoice[{1, 2}]};
    sampleToGuess = Characters[If[gseq == 1, sGeneSeq1, sGeneSeq2]][[
      offset ;; offset + slen]];
    {offset, slen, gseq, sampleToGuess}),
   {2000}];
```

Calculate classifications:

In[30]:=
```
classRes12 =
  Table[
   Map[(
      cp1 = ngm1[morder][[1]][[Sequence @@ (# /. ngm1[morder][[2]])]] & /@
        Partition[#[[-1]], morder + 1, 1];
      cp2 = ngm2[morder][[1]][[Sequence @@ (# /. ngm2[morder][[2]])]] & /@
        Partition[#[[-1]], morder + 1, 1];
      cp1 = cp1 /. {0 → 10^-8, 0. → 10^-8.};
      cp2 = cp2 /. {0 → 10^-8, 0. → 10^-8.};
      Append[Most[#], Apply[Times, cp1 / cp2]]) &,
    gTests12],
   {morder, 1, 10}];
```

Classification tuning runs F2 vs. PAH

Generate test sub-sequences:

In[31]:=
```
gTests13 =
  Table[(
    {offset, slen, gseq} = Flatten@
      {RandomInteger[Floor /@ {0.8 Min[StringLength /@ {sGeneSeq1, sGeneSeq3}],
          0.95 Min[StringLength /@ {sGeneSeq1, sGeneSeq3}]}, 1],
       RandomInteger[{100, 120}, 1], RandomChoice[{1, 2}]};
    sampleToGuess = Characters[If[gseq == 1, sGeneSeq1, sGeneSeq3]][[
      offset ;; offset + slen]];
    {offset, slen, gseq, sampleToGuess}),
   {2000}];
```

Calculate classifications:

In[32]:=
```
classRes13 =
  Table[
   Map[(
      cp1 = ngm1[morder]⟦1⟧⟦Sequence @@ (# /. ngm1[morder]⟦2⟧)⟧ & /@
        Partition[#⟦-1⟧, morder + 1, 1];
      cp3 = ngm3[morder]⟦1⟧⟦Sequence @@ (# /. ngm3[morder]⟦2⟧)⟧ & /@
        Partition[#⟦-1⟧, morder + 1, 1];
      cp1 = cp1 /. {0 → 10^-8, 0. → 10^-8.};
      cp3 = cp3 /. {0 → 10^-8, 0. → 10^-8.};
      Append[Most[#], Apply[Times, cp1 / cp3]]) &,
    gTests13],
   {morder, 1, 10}];
```

ROC rates definitions

The following definitions are of different classification success functions as described in [4].

In[33]:=
```
(* true positive rate *)
TPR[i_, classRes_] :=
 Count[classRes⟦i⟧, {_, _, 1, r_} /; r > 1] / Count[classRes⟦i⟧, {_, _, 1, _}] // N
```

In[34]:=
```
(* true negative rate *)
TNR[i_, classRes_] :=
 Count[classRes⟦i⟧, {_, _, 2, r_} /; r < 1] / Count[classRes⟦i⟧, {_, _, 2, _}] // N
```

In[35]:=
```
(* false positive rate *)
FPR[i_, classRes_] :=
 Count[classRes⟦i⟧, {_, _, 2, r_} /; r > 1] / Count[classRes⟦i⟧, {_, _, 2, _}] // N
```

In[36]:=
```
(* false negative rate *)
FNR[i_, classRes_] :=
 Count[classRes⟦i⟧, {_, _, 1, r_} /; r < 1] / Count[classRes⟦i⟧, {_, _, 1, _}] // N
```

The ROC points plot function is define to resemble the ROC curve plot in [4].

In[37]:=
```
ROCPointsGraph[tbl : {{_Integer, _?NumberQ, _?NumberQ} ..}, ratesType_String,
   geneName1_String, geneName2_String, opts : OptionsPattern[]] :=
  Graphics[{Gray, Dashed, Line[{{0, 0}, {1, 1}}], PointSize[0.02],
    Darker[Red], Tooltip[Point[Rest[#]], First[#]] & /@ tbl,
    Darker[Blue], Text[First[#], Rest[#], {-2, -2}] & /@ tbl},
   PlotRange → {{0, 1}, {0, 1}}, Frame → True, Axes → False,
   AspectRatio → 1, FrameLabel → Map[Style[#, FontSize → 16] &,
     {"false " <> ratesType <> " rate", "true " <> ratesType <> " rate"}],
   GridLines → {FindDivisions[{0, 1}, 10], FindDivisions[{0, 1}, 10]},
   GridLinesStyle → Directive[Gray, Dashed], PlotLabel →
    Style["ROC for genome classification\n by n-gram models\n " <> geneName1 <>
      " vs. " <> geneName2, FontFamily → "Times", FontSize → 16], opts];
```

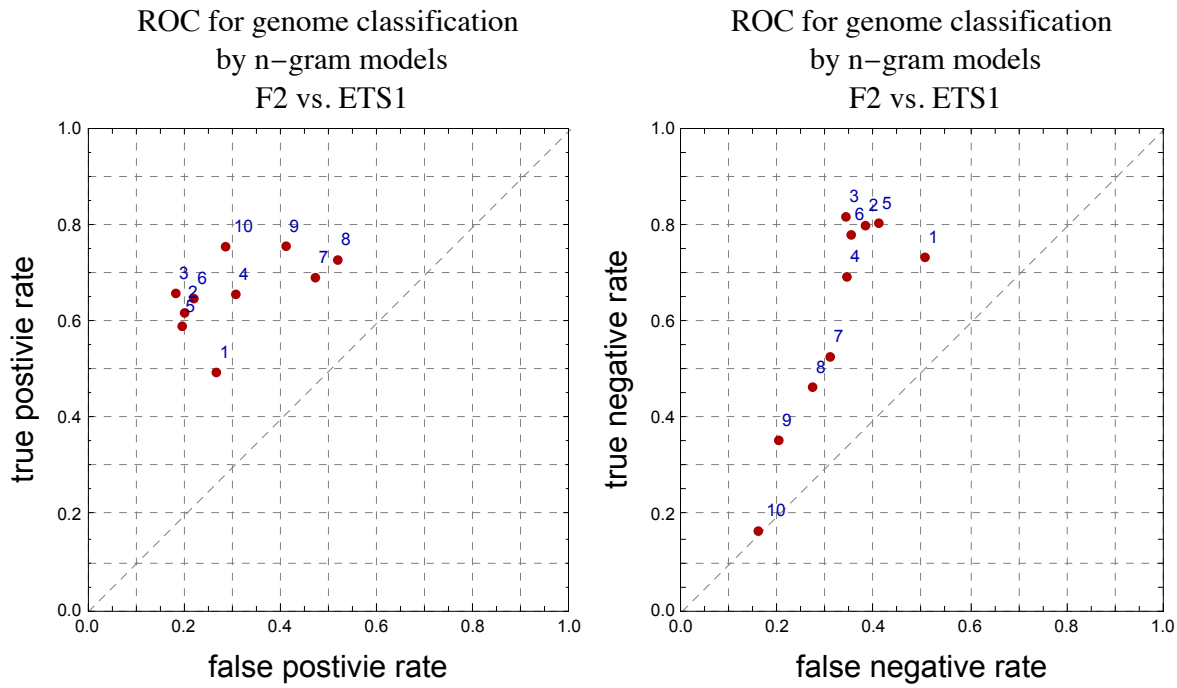
ROC for F2 vs. ETS1

Compute and plot the “standard” ROC with positive rates and its dual with negative rates.

In[38]:=
```
tblP = Table[{i, FPR[i, classRes12], TPR[i, classRes12]}, {i, 1, 10}];
```

In[39]:=
```
tblN = Table[{i, FNR[i, classRes12], TNR[i, classRes12]}, {i, 1, 10}];
```

Out[40]=



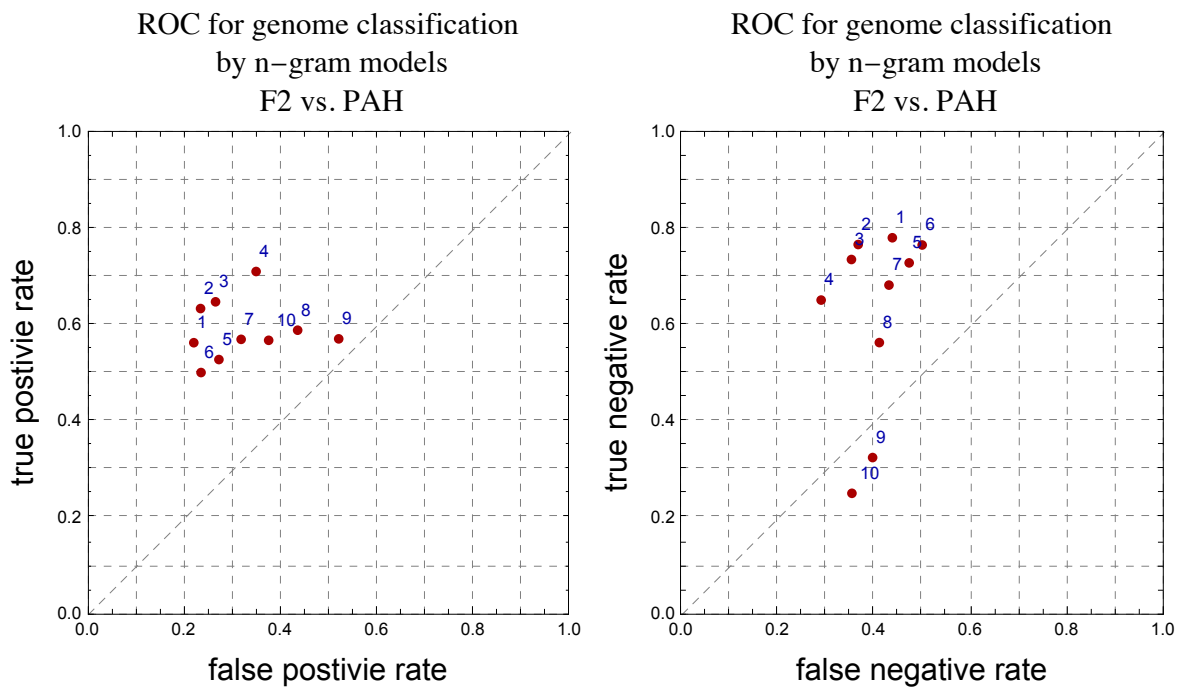
ROC for F2 vs. PAH

Compute and plot the “standard” ROC with positive rates and its dual with negative rates.

In[42]:=
```
tblP = Table[{i, FPR[i, classRes13], TPR[i, classRes13]}, {i, 1, 10}];
```

In[43]:=
```
tblN = Table[{i, FNR[i, classRes13], TNR[i, classRes13]}, {i, 1, 10}];
```

Out[44]=



Conclusions

From the ROC plots in the previous section we can see that the best classification results for GSCP with the genes “F2” and “ETS1” are obtained with Markov chain order 3 or with 4-grams. The 3-grams are almost as good.

A natural extension of the experiments described is to repeat them for other pairs of genes and across different lengths of sub-sequences. In this way we can derive more general conclusions for the best *n*-gram length in the algorithm NGramClassifier.

Repeating the experiments for the genes “F2” and “PAH” showed that using 3-grams gives best results for GSCP.

References

[1] Anton Antonov, N-gram Markov chains *Mathematica* package, source code at GitHub, https://github.com/antononcube/MathematicaForPrediction, package NGramMarkovChains.m, (2014).

[2] Wikipedia entry, *n*-gram, http://en.wikipedia.org/wiki/N-gram .

[3] Wikipedia entry, Odds ratio, http://en.wikipedia.org/wiki/Odds_ratio .

[4] Wikipedia entry, Receiver operating characteristic, http://en.wikipedia.org/wiki/Receiver_operating_characteristic .