A Machine Learning Analysis of Halting in the SKI Combinator Calculus

Euan Ong

Abstract

Much of machine learning is driven by the question: can we learn what we cannot compute? The learnability of the halting problem, the canonical undecidable problem (?) , to an arbitrarily high accuracy for Turing machines was proven by Lathrop (Lathrop, 1996). The SKI combinator calculus can be seen as a reduced form of the untyped lambda calculus, which is Turing-complete (Turing, 1937); hence, the SKI combinator calculus forms a universal model of computation. In this vein, we (?) analyse the growth and halting times of SKI combinator expressions, estimate the probability of an SKI combinator expression halting after a given number of steps(,?) and investigate the feasibility of a machine learning approach to predicting whether a given SKI combinator expression is likely to halt.

SK Combinators

<Write about SKI combinators and what halting means - nb, we are not trying to *solve* the *halting problem* which is undecidable. Talk about https://www.ics.uci.edu/~rickl/publications/1996-icml.pdf>
https://en.wikipedia.org/wiki/Unlambda

```
k[x_][y_] := x
```

```
In[◦]:= s[x_][y_][z_] := x[z][y[z]]
```

```
In[◦]:= s[k][s][k]
Out[◦]= k
```

Rules

```
In[•]:= SKRules = {k[x_][y_] :→ x, s[x_][y_][z_] :→ x[z][y[z]]}
Out[•]= {k[x_][y_] :→ x, s[x_][y_][z_] :→ x[z][y[z]]}

In[•]:= s[k][s][k] /. SKRules
Out[•]= k[k][s[k]]

In[•]:= k[k][s[k]] /. SKRules
Out[•]= k

In[•]:= x = s[k][s][k]
Out[•]= s[k][s][k]

In[•]:= y = x /. SKRules
Out[•]= k[k][s[k]]

In[•]:= y
Out[•]= k[k][s[k]]

In[•]:= x
Out[•]= s[k][s][k]

Out[•]= s[k][s][k] == s[k][s]

In[•]:= ClearAll[s, k, x, y, z]

In[•]:= SKRules = {k[x_][y_] :→ x, s[x_][y_][z_] :→ x[z][y[z]]};
        SKEvaluate[expr_] :=
         NestList[#1 /. SKRules &, expr, 50]

In[•]:= SKEvaluate[s[k][s][k]]
Out[•]= {s[k][s][k], k[k][s[k]], k, k, k, k, k, k, k, k, k, k, k, k, k, k, k, k, k, k, k, k,
         k, k, k, k, k, k, k, k, k, k, k, k, k, k, k, k, k, k, k, k, k, k, k, k, k, k, k, k, k}

In[•]:= {s[k][s][k]}
Out[•]= {s[k][s][k]}

In[•]:= expr = s[s[s]][s][s][s]
Out[•]= s[s[s]][s][s][s]

In[•]:= y = Characters /@ ToString /@ SKEvaluate[expr]

In[•]:= x = s[s[s[s]][s]][s[s[s[s]][s]]]
Out[•]= s[s[s[s]][s]][s[s[s[s]][s]]]
```

```
Characters[y]
```

Out[•]= {s, [, s, [, s, [, s,],], [, s,],], [, s, [, s, [, s, [, s,],], [, s,],],]}

In[•]:=
```
SKColourTable = Table[{x, y} → Blue, {x, 11}, {y, 11}]

abc = Flatten[SKColourTable, 1]
```

Rasterization - add examples

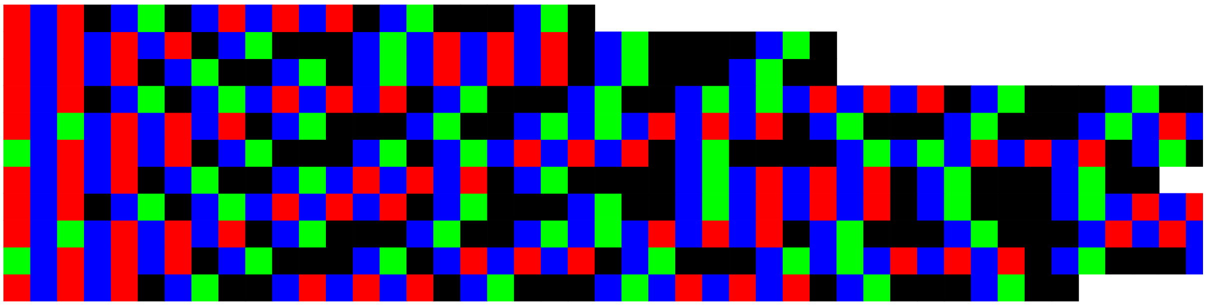
In[•]:=
```
SKRasterize[func_] := SKRasterize[func, 10];
SKRasterize[func_, n_] := Module[{SKRules, SKEvaluate, SKArray, SKGrid},
  SKRules = {k[x_][y_] :→ x, s[x_][y_][z_] :→ x[z][y[z]]};
  SKEvaluate[expr_] := NestList[#1 /. SKRules &, expr, n];
  SKArray[expr_] := Characters /@ ToString /@ SKEvaluate[expr];
  SKGrid[exp_] :=
   ArrayPlot[SKArray[exp], {ColorRules → {"s" → RGBColor[1, 0, 0],
        "k" → RGBColor[0, 1, 0], "[" → RGBColor[0, 0, 1], "]" → RGBColor[0, 0, 0]},
      PixelConstrained → True, Frame → False, ImageSize → 1000}];
  SKGrid[func]
 ]
```

In[•]:=
```
SKRules = {k[x_][y_] :→ x, s[x_][y_][z_] :→ x[z][y[z]]};
SKEvaluate[expr_, n_] := NestList[#1 /. SKRules &, expr, n];
SKEvaluate[expr_] := SKEvaluate[expr, 10];
SKArray[expr_] := SKArray[expr, 10];
SKArray[expr_, n_] := Characters /@ ToString /@ SKEvaluate[expr, n];
SKGrid[exp_] := SKGrid[exp, 10];
SKGrid[exp_, n_] :=
  ArrayPlot[SKArray[exp, n], {ColorRules → {"s" → RGBColor[1, 0, 0],
       "k" → RGBColor[0, 1, 0], "[" → RGBColor[0, 0, 1], "]" → RGBColor[0, 0, 0]},
     PixelConstrained → True, Frame → False, ImageSize → 1000}];
SKRasterize[func_] := SKRasterize[func, 10];

SKRasterize[func_, n_] := SKGrid[func, n]
```

In[•]:=
```
SKLengths[exp_, n_] := StringLength /@ ToString /@ SKEvaluate[exp, n];
```

In[•]:=
```
SKRasterize[s[s][k][s[s[s][k]]][k]]
```

Out[•]=

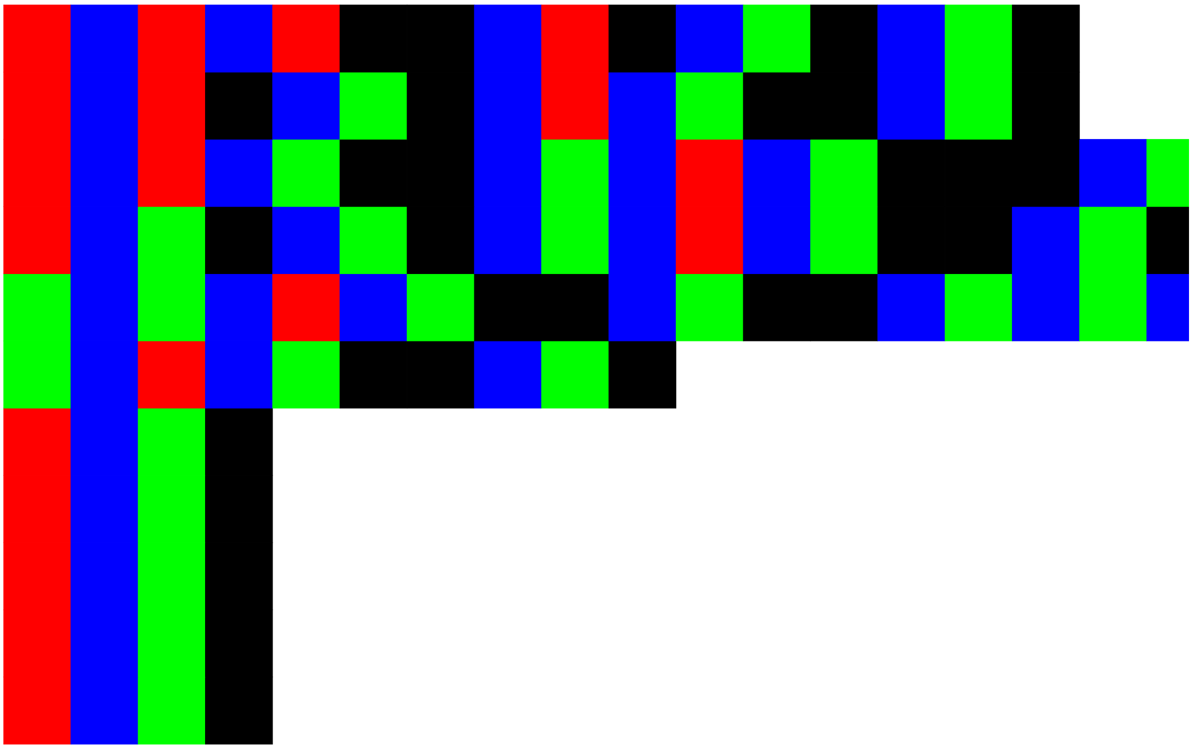


In[∘]:=
```
SKFuncs = {s[s[s]][s][k][k], s[s[s]][s][s][s], s[s[s]][s][s][s][k],
  s[s][s][s[s[s]]][s], s[s[s]][s][s][s][s], s[s[s]][s][s][s][s],
  s[s][s][s[s[s[s]]]][k], s[s][k][s[s[s]]][s][k], s[s][s][s[s[s]]][s][k],
  s[s][s][s[s]][s][s[k]], s[s[s[s][s]]][s][s][k], s[s][k][s[s[s][k]]][k]}
```

Out[∘]=
```
{s[s[s]][s][k][k], s[s[s]][s][s][s], s[s[s]][s][s][s][k],
 s[s][s][s[s[s]]][s], s[s[s]][s][s][s][s], s[s[s]][s][s][s][s],
 s[s][s][s[s[s[s]]]][k], s[s][k][s[s[s]]][s][k], s[s][s][s[s[s]]][s][k],
 s[s][s][s[s]][s][s[k]], s[s[s[s][s]]][s][s][k], s[s][k][s[s[s][k]]][k]}
```

In[∘]:=
```
SKRasterize /@ SKFuncs
```

Out[∘]= {



Random SK combinators ()

Investigating growth of SK combinators

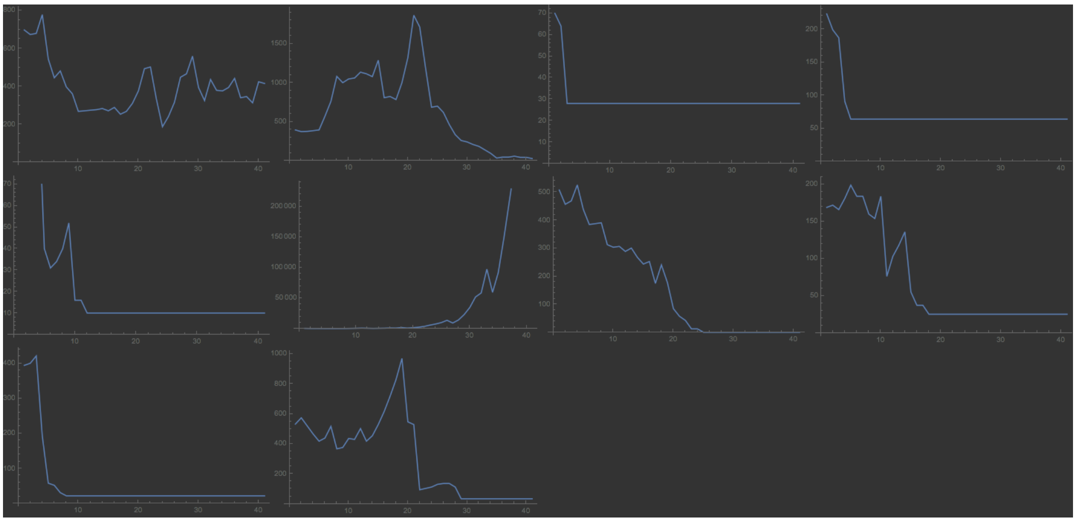
Halting Graphs

SKLengths generates a table of length of combinator

In[∘]:=
```
ListLinePlot[
 SKLengths[k[k[k[k[s[s[s]]]][s]][k[s][k]][s[k]][k]][k[s[k[k]][s]][k[k]][k]][
   s[k[k[k]]]][s[s]][s], 40]]
```

In[76]:=
```
exprs = Table[RandomSKExpr[10], 10];
ImageCollage[Table[ListLinePlot[SKLengths[exprs[[n]], 40]], {n, 10}]]
```

Out[77]=



Halting Probabilities

Some halt, some do not. (haven’t seen a cyclical one yet) --> linear/exponential.
Assumptions: If length stays constant, it has halted. **Dataset:** SK combinators with depth 10
P(halts by 20|doesn’t halt by 10) = ? P(halts by 30|doesn’t halt by 20

In[∘]:=
```
exprs = Monitor[Table[RandomSKExpr[10], {n, 100}], n];
```

In[∘]:=
```
lengths = Monitor[Table[SKLengths[exprs[[n]], 40], {n, 100}], n];
```

Out[∘]=
```
$Aborted
```

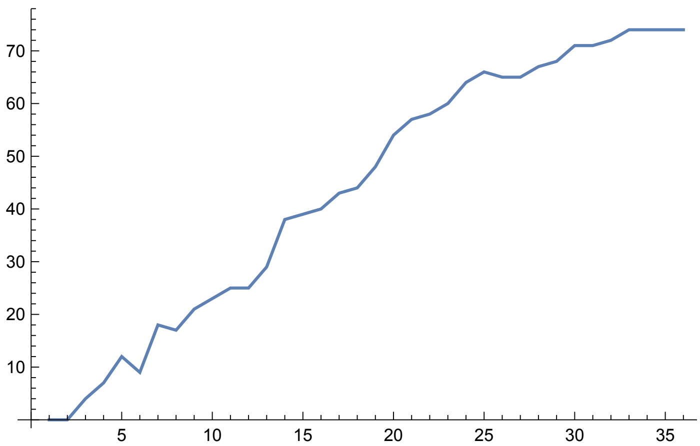
```
HaltIf[n_, list_] := SameQ[list[[n]], list[[n - 1]]]
```

In[∘]:=
```
HaltBy[n_, lens_] := Count[lens, x_ /; HaltIf[n, x] == True]
```

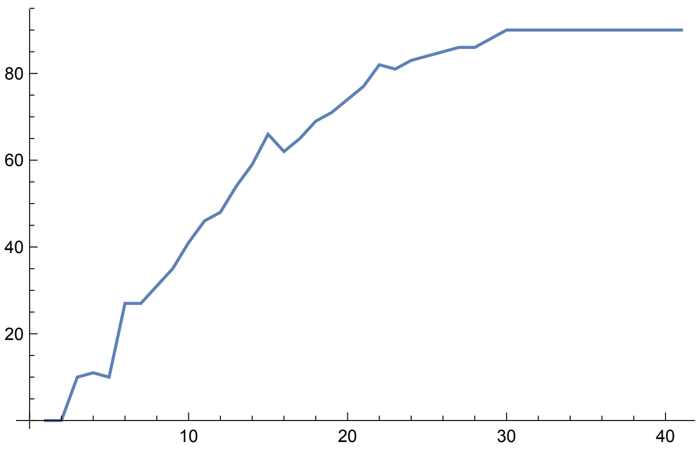
Taking only lengths:

In[∘]:=
```
ListLinePlot[Table[HaltBy[n, lengths], {n, 0, 35}]]
```

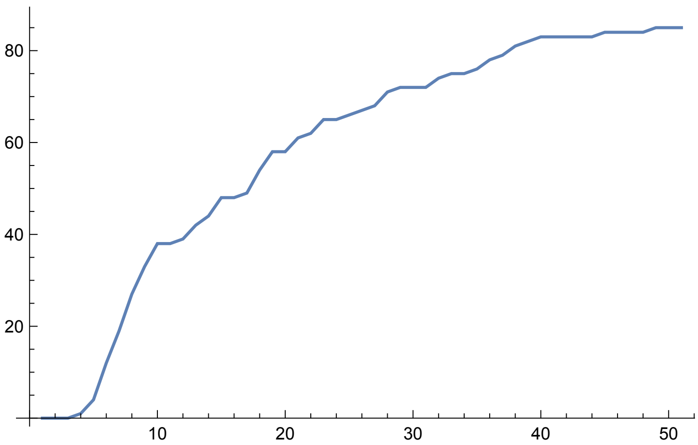
Out[∘]=



Number that have ‘constant length’ after every 5 iterations (c.f. nuclear half life?)



Number that have halted after every 5 iterations (checking for actual halting)



In[∘]:=
```
Table[HaltBy[n, lengths] - HaltBy[n - 5, lengths], {n, 5, 40, 5}]
```
Out[∘]=
```
{27, 19, 16, 15, 8, 5, 0, 0}
```

In[∘]:=
```
exprs = Monitor[Table[RandomSKExpr[10], {n, 1000}], n];
```
In[∘]:=
```
lengths = Monitor[Table[SKEvaluate[exprs[[n]], 50], {n, 1000}], n];
```
Out[∘]=
```
$Aborted
```

In[∘]:=
```
haltbytable = GenerateHaltByTable[10, 50, 1000]
```
Out[∘]=
```
{… 1 … [s] → 26, s[s[… 1 …]] → False, … 996 …, k[… 1 …] → 13,
 k[k[k[s[k[s[s[s[s[s[k[k[k[… 1 …][s[s]][s]][k[k[… 1 …][k[s]]]]]]]]][
   k[k[k[k]]]]][s]]][
   s[… 1 …]]][… 1 … [k]]][s[s[s]]]][s] → 11}
```
large output | show less | show more | show all | set size limit...

In[∘]:=
```
DumpSave["/Users/eohomegrownapps/CODE/Assorted
    codings/Wolfram/SK-Combinators/10_50_haltbytable.mx", haltbytable];
```

In[•]:=
```
vals = BinCounts[Sort[Values[haltbytable]], {1, 51, 1}]
```

Out[•]=
```
{0, 2, 19, 33, 39, 37, 44, 42, 27, 35, 30, 37, 31,
 30, 29, 33, 30, 25, 22, 26, 25, 27, 23, 15, 18, 20, 18, 15, 14,
 5, 13, 6, 17, 9, 8, 5, 2, 4, 5, 1, 0, 4, 8, 2, 4, 3, 1, 2, 3, 0}
```

In[•]:=
```
cumulative = Table[Total[vals[[1 ;; n]]], {n, 1, Length[vals]}]
```

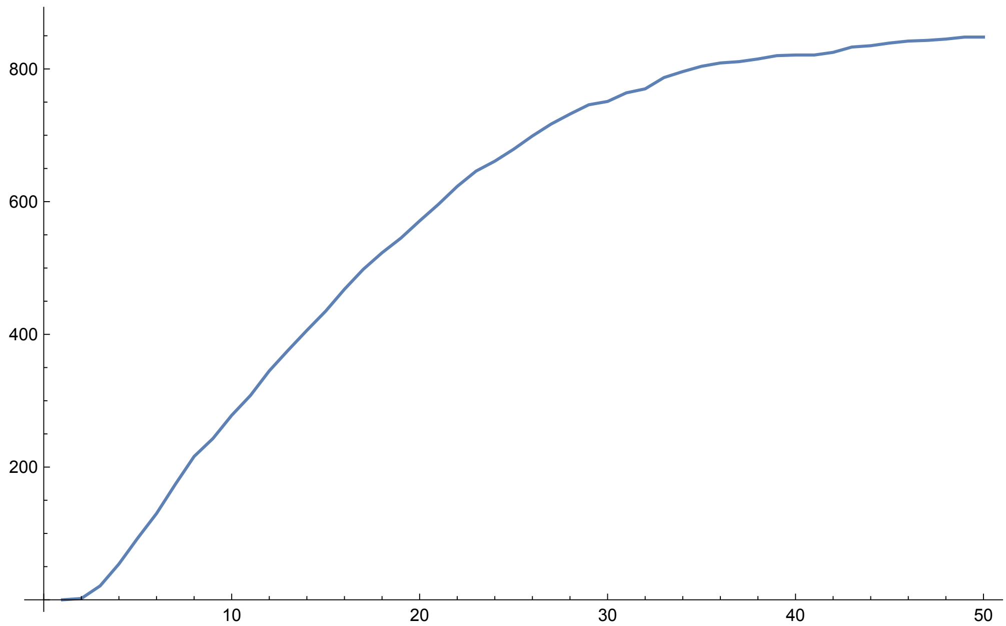
Out[•]=
```
{0, 2, 21, 54, 93, 130, 174, 216, 243, 278, 308, 345, 376, 406, 435, 468, 498, 523,
 545, 571, 596, 623, 646, 661, 679, 699, 717, 732, 746, 751, 764, 770, 787, 796,
 804, 809, 811, 815, 820, 821, 821, 825, 833, 835, 839, 842, 843, 845, 848, 848}
```

In[•]:=
```
Table[{n, cumulative[[n]]}, {n, 1, Length[cumulative]}]
```

Out[•]=
```
{{1, 0}, {2, 2}, {3, 21}, {4, 54}, {5, 93}, {6, 130}, {7, 174}, {8, 216},
 {9, 243}, {10, 278}, {11, 308}, {12, 345}, {13, 376}, {14, 406}, {15, 435},
 {16, 468}, {17, 498}, {18, 523}, {19, 545}, {20, 571}, {21, 596}, {22, 623},
 {23, 646}, {24, 661}, {25, 679}, {26, 699}, {27, 717}, {28, 732}, {29, 746},
 {30, 751}, {31, 764}, {32, 770}, {33, 787}, {34, 796}, {35, 804}, {36, 809},
 {37, 811}, {38, 815}, {39, 820}, {40, 821}, {41, 821}, {42, 825}, {43, 833},
 {44, 835}, {45, 839}, {46, 842}, {47, 843}, {48, 845}, {49, 848}, {50, 848}}
```

In[•]:=
```
ListLinePlot[Table[Total[vals[[1 ;; n]]], {n, 1, Length[vals]}]]
```

Out[•]=



In[•]:=
```
SKHalt[expr_, limit_] := Module[{evaluate},
  evaluate = SKEvaluate[expr, limit];
  HaltIf[limit, evaluate]
 ]
```

In[•]:=
```
SKHalt[k[s[s[k[s][k[k]][s]][s[s[s]]][s[k]][s]]][k[k[s[k]]]][s[s]][s], 10]
```

Out[•]= False

In[•]:=
```
{a → b, c → d}
```

Out[•]= {a → b, c → d}

```
In[•]:= f @@ {a → b, c → d}
Out[•]= f[a → b, c → d]
```

Machine Learning Analysis of SK Combinators

Generating Datasets

~1000*n random SK expressions at each of depths {n,1,10}, halted if SKHalt[40]==True.

```
Monitor[Table[x = GenerateTable[n, 40, n * 1000];
  DumpSave["/Users/eohomegrownapps/CODE/Assorted
      codings/Wolfram/SK-Combinators/" <> ToString[n] <> ".mx", x],
  {n, 1, 15}], n] (*generate all possible expressions*)
```

~5000 random SK expressions at depth 10, halted if SKHalt[40] == True.

~5000 random SK expressions at depth 8, halted if SKHalt[40] == True.

```
In[•]:= x = GenerateTable[8, 40, 5000];
DumpSave["/Users/eohomegrownapps/CODE/Assorted
    codings/Wolfram/SK-Combinators/8_40.mx", x];
```

```
n
```

```
In[•]:= x = GenerateTable[8, 40, 5000];
DumpSave["/Users/eohomegrownapps/CODE/Assorted
    codings/Wolfram/SK-Combinators/8_40_test.mx", x];
```

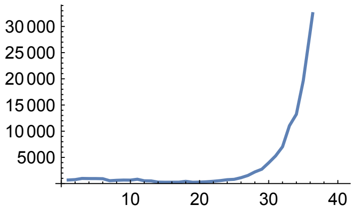
Training Attempt #1: 1000 random SK expressions, depth 10, halted if SKHalt[40]==True. NoHalt dataset same length as Halt dataset. Using raw string. Best classifier so far.

```
lengths = GenerateTable[10, 40, 1000]

NoHalt = Select[lengths, #[[2]] == False &]
Halt = Select[lengths, #[[2]] == True &]
HaltTrain = RandomSample[Halt, Length[NoHalt]]
TrainingData = Join[HaltTrain, NoHalt]
TrainingData2 = ConvertSKTableToString[TrainingData]

HaltClassifier1 = Classify[TrainingData2]
```

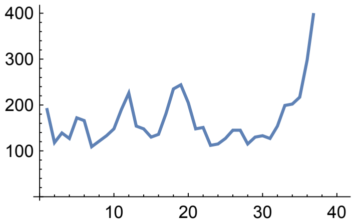
```
In[•]:= HaltClassifier1["s[s[s]][s][s][s][k]"]
Out[•]= True
```

Classifier test:

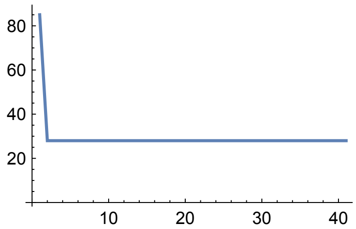
```
Table[TestClassifier[HaltClassifier1, TrainingData2], 10]
```

Out[•]= {False, 

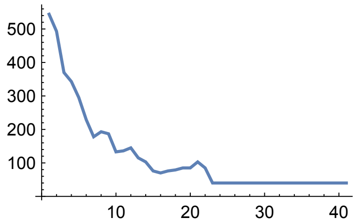
}

Out[•]= {False, 

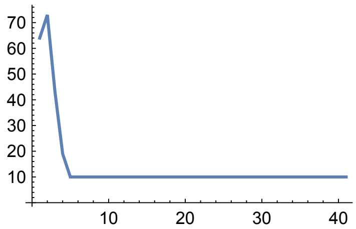
, False}

Out[•]= {True, 

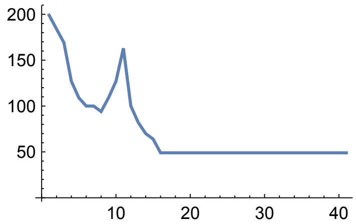
, False}

Out[•]= {True, 

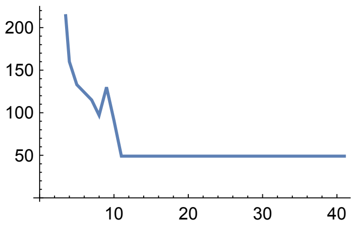
}

Out[•]= {True, 

}

Out[•]= {True, 

}

Out[•]= {True,  }


```
In[•]:= SKGrid[k[
   s[k[k[k[k[s[s[k[k[k[k][s[s[k[k[s[s][k[k[k][k[k[s[k]]]]][k[k]][s]]]]][k[s]]]][
                          k[s[k[k][k[s[k[s[s][s[s]]]][s[s[k]]]][k]]][s[s[k[k]]]][
                            k[k]]]][s[k[s[s[k][s[k][k]][s[k]]]][
                             k[s[k]][k]]][s[k]][k]]][s[k[s]][k]][
                      k[k[k]]]][k]]]]]][k[s[s[k[s[s]]]]][s]]][k[s]]]]
```

Out[•]= 

```
In[•]:= GenerateTable[depth_, iterations_, number_] := Module[{exprs},
   exprs = Monitor[Table[RandomSKExpr[depth], {n, number}], n];
   lengths =
    Monitor[Table[exprs[[n]] → SKHalt[exprs[[n]], iterations], {n, number}], n];
   Return[lengths]
  ]

Monitor[Flatten[Table[GenerateTable[n, 40, 200], {n, 1, 10}]], n]
```

```
Out[•]= {k[s] → True, s[k] → True, ⋯ 1997 ⋯ ,
  s[k[k[s[k[k[s[k[s[s[s[k][k[k[k[s[k[s[k[s[k[s][s[s[k]][k[s]][s]]][s[s][s]][
                                            k[k]]]]][k[k[s][k[s]][s]][s[s[k]][k]][s[k]][
                                         k]]]]]]][s[s[k[k[s[k[k[s[k]][s[k[k]]]]]]][
                                      s[k[k[s][s]]][k]][s[k[s]][s]]]]]]][
                               s[k][k[s]][s]][s[s[s]]][s[k]][k]]]]]][
                      s[k][k[s[s]][k]][s[k]][s]]]][k[k]]] → True}
```

large output | show less | show more | show all | set size limit...

Training Attempt #2: 200 random SK expressions at each of depths 1-10, halted if SKHalt[40]==True. NoHalt dataset not same length as Halt dataset. Using raw string. Bad performance. <citation needed>

```
LargeTrainingData = GenerateTableDepthRange[1, 10, 40, 200]
```

```
In[•]:= Length[LargeTrainingData]
Out[•]= 1563
```

```
LargeTrainingData2 = ConvertSKTableToString[LargeTrainingData]
```

```
In[•]:= LargeClassify = Classify[LargeTrainingData2]

Out[•]= ClassifierFunction[ Input type: Text
                            Classes: False, True ]

In[•]:= LargeClassify["s[s][s][s[s[s]]][s][s][s][s][s]"] (* halts *)

Out[•]= False
```

Training Attempt #3: 200 random SK expressions at each of depths 1 - 10, halted if SKHalt[40] == True. NoHalt dataset same length as Halt dataset. Using raw string. Bad performance. <citation needed>

```
NoHaltLarge2 = GetSKHalt[LargeTrainingData2, False]

In[•]:= Length[NoHaltLarge2]

Out[•]= 99

HaltLarge2 = GetSKHalt[LargeTrainingData2, True]

In[•]:= Length[HaltLarge2]

Out[•]= 1464
```

Many of these (200 from each depth) halt. (# not halting increases with depth)

```
HaltTrainLarge2 = RandomSample[HaltLarge2, Length[NoHaltLarge2]];

In[•]:= TrainLarge2Sample = Join[HaltTrainLarge2, NoHaltLarge2];

In[•]:= Large2Classify = Classify[TrainLarge2Sample]

Out[•]= ClassifierFunction[ Input type: Text
                            Classes: False, True ]

In[•]:= Large2Classify["s[s[s[k[s[s[k[s[k][s]]][k]]][k[k]][k]]]][s[s]][s]"]

Out[•]= True
```

Training Attempt #4: 200 random SK expressions at each of depths 1 - 10, halted if SKHalt[40] == True. NoHalt dataset same length as Halt dataset. Using raw string. Bad performance. <citation needed>

Training

```
In[•]:= exprs = Monitor[ParallelTable[RandomSKExpr[10], {n, 5000}], n];

SubKernels`LocalKernels`LaunchLocal: Could not provide a subkernel license.

Out[•]= $Aborted
```

```
lengths =
  Monitor[ParallelTable[exprs[[n]] → SKHalt[exprs[[n]], 40], {n, 5000}], n];
```

••• ParallelTable: No parallel kernels available; proceeding with sequential evaluation.

Out[•]= $Aborted

In[•]:= LaunchKernels[]

SubKernels`LocalKernels`LaunchLocal: Could not provide a subkernel license.

SubKernels`LocalKernels`LaunchLocal: Could not provide a subkernel license.

SubKernels`LocalKernels`LaunchLocal: 2 of 2 kernels failed to launch.

Out[•]= {}

```
In[•]:= gridexprs = Monitor[ParallelTable[SKGrid[exprs[[n]]], {n, 1, Length[exprs]}], n];
```

••• ParallelTable: No parallel kernels available; proceeding with sequential evaluation.

In[•]:= exprs[[1]]

```
Out[•]= k[s[
  k[k[k[k[s[s[k][k[s[s[s[s[k[k[k[k[s[s[k[s[k[s[s[s[k[k[s]]]]][k[s[s[k[k[s]]]]][
                                   k]]]]]][s[k[k[k]][s[k]]]][
                              s[k[k[k[k[k]][s]]]]][k[k[k[k]]]][s[s]]]][
                          s[k[s[k[s[s]]]]]][s[s[s][k[k][k]][k[k]]]][
                        k[s[s][s[k]]]]][k[s]]]]]][s[s[k[s[k][s]][k[s]][k[]]][
                    s[k]][k]][k[k[s[k[k]][k]][k[k]]]]]]]]][
             s[k[k[s][k[k[s[k[k[k[s][s]][k[s]][k]]]]]][s[k[s[s]]]][
                k[k[s[s]]]]]]]]]]]
```

```
In[•]:= Count[Values[lengths], False]
Out[•]= 903
```

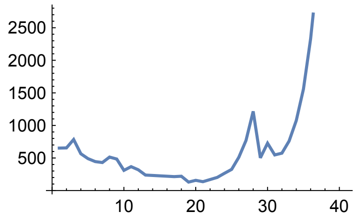
```
In[•]:= NoHalt = Select[lengths, #[[2]] == False &];
        Halt = Select[lengths, #[[2]] == True &];
        HaltTrain = RandomSample[Halt, Length[NoHalt]];
        TrainingData = Join[HaltTrain, NoHalt];
        TrainingData /. Rule → List;
        TrainingData2 =
          Table[ToString[TrainingData[[n]][[1]]] → TrainingData[[n]][[2]],
           {n, 1, Length[TrainingData]}];
```

```
In[•]:= HaltClassifier2 = Classify[TrainingData2]
```

Out[•]= ClassifierFunction[Input type: Text Classes: False, True]

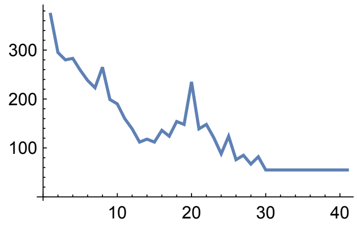
Testing

In[•]:= TestClassifier[HaltClassifier2]

Out[•]= {False, 

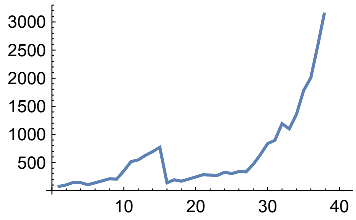
, False}

In[•]:= TestClassifier[HaltClassifier2]

Out[•]= {True, 

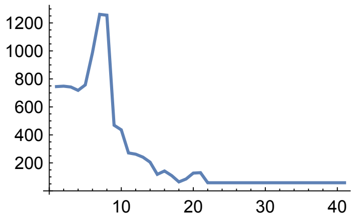
, False}

In[•]:= TestClassifier[HaltClassifier2]

Out[•]= {False, 

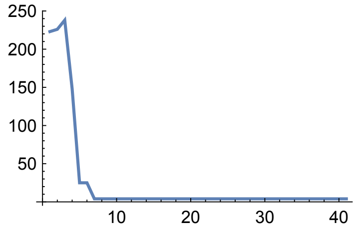
, False}

In[•]:= TestClassifier[HaltClassifier2]

Out[•]= {True, 

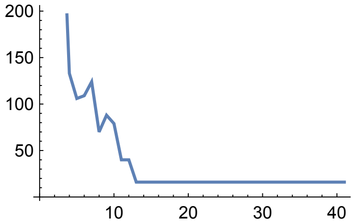
, False}

In[•]:= TestClassifier[HaltClassifier2]

Out[•]= {False, 

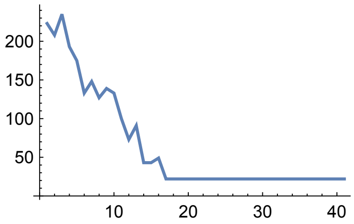
, False}

In[•]:= `TestClassifier[HaltClassifier2]`

Out[•]= {True, 

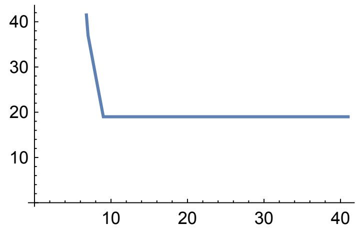
, False}

In[•]:= `TestClassifier[HaltClassifier2]`

Out[•]= {True, 

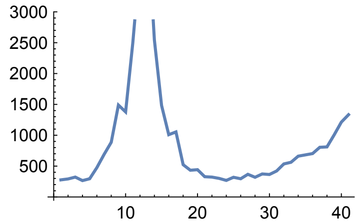
, False}

In[•]:= `TestClassifier[HaltClassifier2]`

Out[•]= {True, 

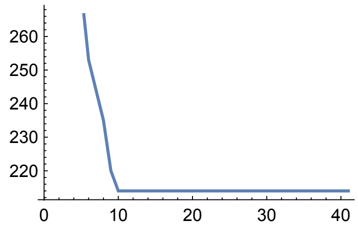
, False}

In[•]:= `TestClassifier[HaltClassifier2]`

Out[•]= {True, 

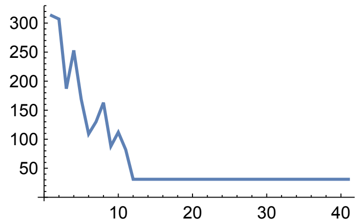
, False}

In[•]:= `TestClassifier[HaltClassifier2]`

Out[•]= {True, 

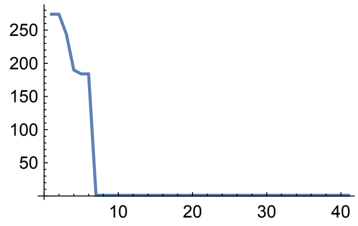
, False}

In[•]:= `TestClassifier[HaltClassifier2]`

Out[•]= {True, 

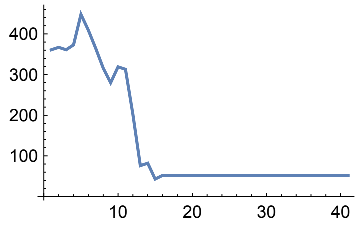
, False}

In[∘]:= `TestClassifier[HaltClassifier2]`

Out[∘]= {True, 

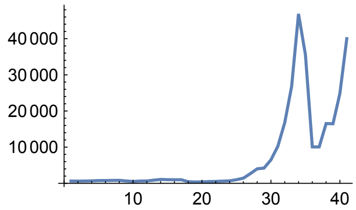
, False}

In[∘]:= `TestClassifier[HaltClassifier2]`

Out[∘]= {True, 

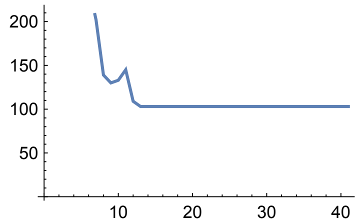
, False}

In[∘]:= `TestClassifier[HaltClassifier2]`

Out[∘]= {False, 

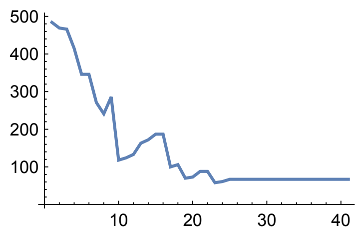
, False}

In[∘]:= `TestClassifier[HaltClassifier2]`

Out[∘]= {True, 

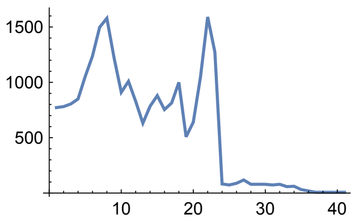
, False}

In[∘]:= `TestClassifier[HaltClassifier2]`

Out[∘]= {False, 

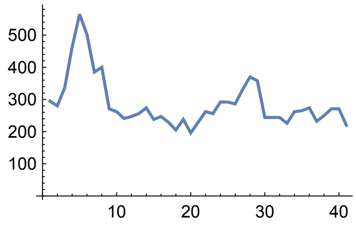
, False}

In[∘]:= `TestClassifier[HaltClassifier2]`

Out[∘]= {False, 

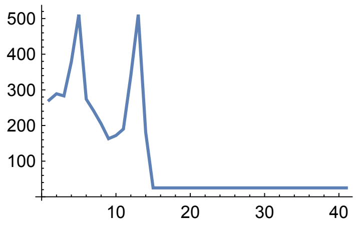
, False}

In[•]:=
```
TestClassifier[HaltClassifier2]
```

Out[•]= {False, 

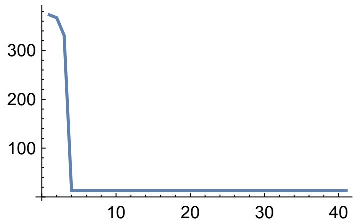
, False}

In[•]:=
```
TestClassifier[HaltClassifier2]
```

Out[•]= {True, 

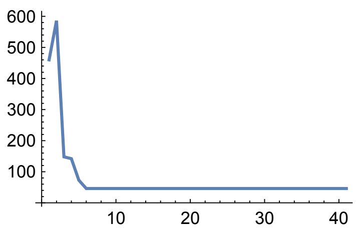
, False}

In[•]:=
```
TestClassifier[HaltClassifier2]
```

Out[•]= {True, 

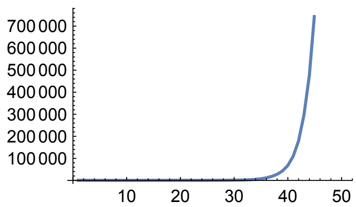
, False}

In[•]:=
```
TestClassifier[HaltClassifier2]
```

Out[•]= {True, 

, False}

In[•]:=
```
TestClassifier[HaltClassifier2]
```

Out[•]= {True, 

, False}

Training Attempt #5: 5000 random SK expressions, depth 10, halted if SKHalt[40]==True. NoHalt dataset same length as Halt dataset. Using raw string. (Same as #1, but larger dataset) Worse than 1 (slightly).

In[•]:=
```
lengths = x;
```

In[ ]:=
```
NoHalt = Select[lengths, #[[2]] == False &];
Halt = Select[lengths, #[[2]] == True &];
Length[NoHalt]
Length[Halt]
```

Out[ ]= 862

Out[ ]= 4138

In[ ]:=
```
HaltTrain = RandomSample[Halt, Length[NoHalt]];
TrainingData = Join[HaltTrain, NoHalt];
TrainingData2 = ConvertSKTableToString[TrainingData];
Length[TrainingData2]
```

Out[ ]= 1724

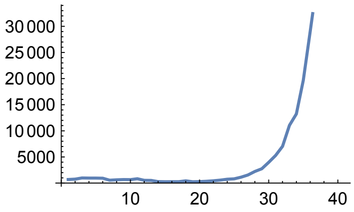
In[ ]:=
```
HaltClassifier1 = Classify[TrainingData2]
```

Out[ ]=
```
ClassifierFunction[ Input type: Text
                    Classes: False, True ]
```

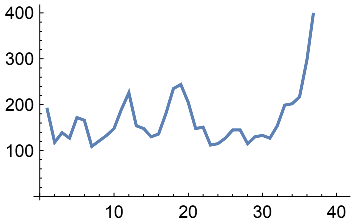
In[ ]:=
```
Table[TestClassifier[HaltClassifier1, TrainingData2], 10]
```

Classifier test:

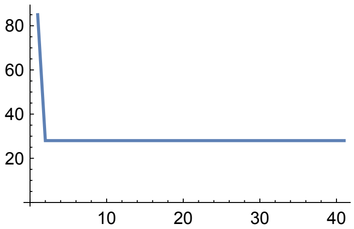
```
Table[TestClassifier[HaltClassifier1, TrainingData2], 10]
```

Out[•]= {False, 

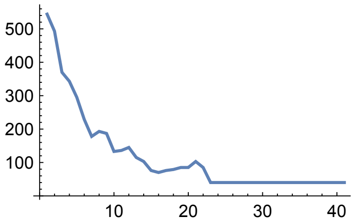
}

Out[•]= {False, 

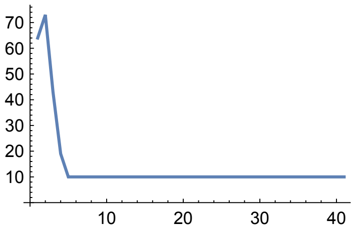
, False}

Out[•]= {True, 

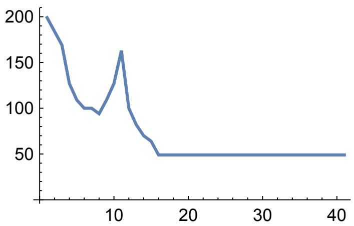
, False}

Out[•]= {True, 

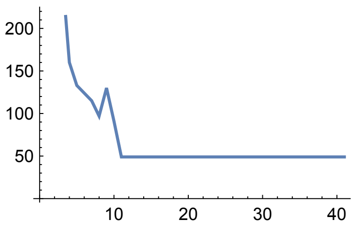
}

Out[•]= {True, 

}

Out[•]= {True, 

}

Out[•]= {True, 

}

In[∘]:=
```
SKGrid[k[
  s[k[k[k[k[s[s[k[k[k[k][s[s[k[k[s[s][k[k[k][k[k[s[k]]]]][k[k]][s]]]]][k[s]]]][
                   k[s[k[k][k[s[k[s[s][s[s]]]][s[s[k]]]][k]]][s[s[k[k]]]][
                     k[k]]]][s[k[s[s[k][s[k][k]][s[k]]]][
                      k[s[k]][k]]][s[k]][k]]][s[k[s]][k]][
               k[k[k]]]][k]]]]]][k[s[s[k[s[s]]]]][s]]][k[s]]]]
```

Out[∘]=

In[∘]:=
```
GenerateTable[depth_, iterations_, number_] := Module[{exprs},
  exprs = Monitor[Table[RandomSKExpr[depth], {n, number}], n];
  lengths =
   Monitor[Table[exprs[[n]] → SKHalt[exprs[[n]], iterations], {n, number}], n];
  Return[lengths]
 ]

Monitor[Flatten[Table[GenerateTable[n, 40, 200], {n, 1, 10}]], n]
```

Out[∘]=
```
{k[s] → True, s[k] → True, … 1997 …,
 s[k[k[s[k[k[s[k[s[s[s[k][k[k[k[s[k[s[k[s[k[s][s[s[k]][k[s]][s]]][s[s][s]][
                                      k[k]]]]][k[k[s][k[s]][s]][s[s[k]][k]][s[k]][
                                   k]]]]]]][s[s[k[k[s[k[k[s[k]][s[k[k]]]]]]][
                                s[k[k[s][s]]][k]][s[k[s]][s]]]]]]][
                         s[k][k[s]][s]][s[s[s]]][s[k]][k]]]]]][
                 s[k][k[s[s]][k]][s[k]][s]]]][k[k]]] → True}
```

large output | show less | show more | show all | set size limit...

ML Advice - from Matteo Salvarazza

ML Advice

How to represent data?

- Sequence of ‘sequences’

Can you find a mapping between one of the sequences and an integer?

--> base4 encoding (this will be unique)

Problem: input size is unbounded.

Solution: Generate training set, use base4 encoding, look at maximum

--> or ?strings or something?

Advantages - it captures subtleties of combinators

Alternatively, use base4 and padding - then they become ‘images’ (matrices).

In this case, still use RNN - a sequence of n-dimensional vectors where n is the longest element in training set.

--> or trees?

Advantages - purest method of representing combinators.

- or just initial SKcombinator (‘sequence’)

How to creat

Type of dataset? (50:50 halt:no halt or actual distribution?)
Training set *must* be balanced, even if real world not balanced.

Ratio of data within dataset? (distribution of examples belonging to specific class)
Usually unimportant - just experiment. Generate a *balanced training set* and an *unbalanced training set*

What model to use?
Recurrent neural net.
base4 format - sequence classification problem.
Usual entry-level problem - sentiment analysis. Take this architecture and experiment.
Look at tutorials about sentiment analysis (simple - this problem is much harder)

Ensure {no --> very few} combinators halt within the given training set, otherwise problem is trivial.
(e.g. size 10 vector - [[9]]!=[[10]] - experiment)

First thing to try: do the initial base 4 encoding, generate (some large n) training sets, find vocabulary size and check for presence of duplicates. If super sparse (large vocabulary, most tokens only appear once), this is bad
--> try sequence encoding, with padding method. (Problem - a lot of padding. This is also bad. Experiment with different initial evolution lengths)
(alternatively, try RNN with just initial state - will solve all of the above problems, but probably won’t work. 0th thing to try - training example just a sequence of {chars/base4 numbers})

Neural Net Attempt #1: Recurrent Neural Network, Raw String. SKCombinators_RNN_Raw_String.nb

Unsuccessful - no better than coin flipping. (Markov method earlier is better)

Neural Net Attempt #2 - Preprocessing: Find vocabulary.