

Progressive machine learning examples

Anton Antonov

MathematicaForPrediction at WordPress

MathematicaForPrediction at GitHub

MathematicaVsR at GitHub

April 2018

Introduction

In this notebook (document) we are going to follow several examples of doing Progressive machine learning over several datasets.

Progressive learning is a type of Online machine learning. For more details see [Wk1]. The Progressive learning problem is defined as follows.

Problem definition:

- Assume that the data is sequentially available.
 - Meaning, at a given time only part of the data is available, and after a certain time interval new data can be obtained.
 - In view of classification, it is assumed that at a given time not all class labels are presented in the data already obtained.
 - Let us call this a *data stream*.
- Make a machine learning algorithm that updates its model continuously or sequentially in time over a given data stream.
 - Let us call such an algorithm a Progressive Learning Algorithm (PLA).

In comparison, the typical (classical) machine learning algorithms assume that representative training data is available and after training that data is no longer needed to make predictions. Progressive machine learning has more general assumptions about the data and its problem formulation is closer to how humans learn to classify objects.

Below we are shown the applications of two types of classifiers as PLA's. One is based on Tries with Frequencies (TF), [AAp2, AAp3, AA1], the other on a Sparse Matrix Recommender (SMR) framework [AAp4, AA2].

Remark: Note that both TF and SMR come from tackling Unsupervised machine learning tasks, but here they are applied in the context of Supervised machine learning.

Additional introductory notes

Progressive learning definition from Wikipedia

Here is the definition of Progressive learning from [Wk1]:

Progressive learning is an effective learning model which is demonstrated by the human learning process. It is the process of learning continuously from direct experience. Progressive learning technique (PLT) in machine learning can learn new classes/labels dynamically on the run. Though online learning can learn new samples of data that arrive sequentially, they cannot learn new classes of data being introduced to the model. The learning paradigm of progressive learning, is independent of the number of class constraints and it can learn new classes while still retaining the knowledge of previous classes. Whenever a new class (non-native to the knowledge learnt thus far) is encountered, the classifier gets remodeled automatically and the parameters are calculated in such a way that it retains the knowledge learnt thus far. This technique is suitable for real-world applications where the number of classes is often unknown and online learning from real-time data is required.

On making Progressive learning algorithms

Simple statistical procedures like mean and standard deviation finding can be made “progressive” with recursive definitions. This implies that certain outlier finding algorithms can be made “progressive”. Also, Progressive learning mean computation implies that the K-means clustering algorithm, [Wk2], can be made progressive too.

Of course, (most) of the Naive Bayes Classifier (NBC) algorithms, [Wk3], can be made progressive. The TF classification algorithm shown below is a NBC algorithm.

We can see that, in general, PLA’s can be derived from different “standard” machine learning algorithms. A more comprehensive list is given in [Wk1].

Load packages

Here we load the packages used in this notebook. (See [AAp1-AAp9].)

```

In[1]:= Import[
  "https://raw.githubusercontent.com/antononcube/MathematicaForPrediction/master/MathematicaForPredictionUtilities.
    m"]
Import[
  "https://raw.githubusercontent.com/antononcube/MathematicaForPrediction/master/DocumentTermMatrixConstruction.m"]
Import["https://raw.githubusercontent.com/antononcube/MathematicaForPrediction/master/SSparseMatrix.m"]
Import["https://raw.githubusercontent.com/antononcube/MathematicaForPrediction/master/TriesWithFrequencies.m"]
Import[
  "https://raw.githubusercontent.com/antononcube/MathematicaForPrediction/master/JavaTriesWithFrequencies.m"]
Import[
  "https://raw.githubusercontent.com/antononcube/MathematicaForPrediction/master/SparseMatrixRecommenderFramework.m
    "]
Import["https://raw.githubusercontent.com/antononcube/MathematicaForPrediction/master/ROCFunctions.m"]
Import["https://raw.githubusercontent.com/antononcube/MathematicaVsR/master/Projects/ProgressiveMachineLearning/
  Mathematica/GetMachineLearningDataset.m"]

```

Data

In this section we obtain and summarize the well known "Titanic" dataset and the "Mushroom" dataset. The data for this project has been prepared with the Mathematica (Wolfram Language) package [AAp1].

Titanic

This obtains the "Titanic" dataset (using [AAp1]):

```

In[9]:= dsTitanic = GetMachineLearningDataset["Titanic", "RowIDs" → True];

```

Here is a sample of the dataset:

In[10]:= **RandomSample[dsTitanic, 6]**

Out[10]=

	id	passengerClass	passengerAge	passengerSex	passengerSurvival
336	336	2nd	50	male	died
121	121	1st	40	male	survived
389	389	2nd	30	female	survived
671	671	3rd	30	male	died
1181	1181	3rd	-1	female	died
993	993	3rd	30	female	died

Here is the dataset summary:

In[11]:= **RecordsSummary[dsTitanic]**

Out[11]=

1 id	3 passengerAge
Min 1	Min -1
1st Qu 327.75	1st Qu 10
Mean 655	Mean 23.55
Median 655	Median 20
3rd Qu 982.25	3rd Qu 40
Max 1309	Max 80

2 passengerClass 4 passengerSex 5 passengerSurvival

1st 709, 1st 323, male 843, died 809, female 466, survived 500

Here is the summary of the dataset in long form:

```
In[12]:= smat = ToSSparseMatrix[dsTitanic];
RecordsSummary[SSparseMatrixToTriplets[smat], {"RowID", "Variable", "Value"}]
```

```

      1 RowID                                3 Value
      1      5      2 Variable                male      843
     10      5      id                      1309 died      809
Out[13]= { 100      5      passengerClass    1309 3rd      709 }
          { 1000     5      , passengerSex      1309 , survived 500 }
          { 1001     5      passengerSurvival 1309 female  466 }
          { 1002     5      passengerAge      1253 20      335 }
          { (Other) 6459                                (Other) 2827 }
```

The dataset was ingested with row IDs; here we drop them:

```
In[14]:= dsTitanic = dsTitanic[Values];
```

Mushroom

This obtains the “Mushroom” dataset using [AAp1]:

```
In[15]:= dsMushroom = GetMachineLearningDataset["Mushroom", "RowIDs" → True];
```

Here is a random record of the dataset:

```
In[16]:= RandomSample[dsMushroom, 1] // First
```

Out[16]=

id	4763
cap-Shape	convex
cap-Surface	scaly
cap-Color	yellow
bruises?	False
odor	foul
gill-Attachment	free
gill-Spacing	close
gill-Size	broad
gill-Color	pink
stalk-Shape	enlarging
stalk-Root	bulbous
stalk-Surface-Above-Ring	silky
stalk-Surface-Below-Ring	silky
stalk-Color-Above-Ring	buff
stalk-Color-Below-Ring	buff
veil-Type	partial
veil-Color	white
ring-Number	one
ring-Type	large
spore-Print-Color	chocolate
population	solitary
habitat	woods
edibility	poisonous

Here is the summary of the dataset in long form:

```
In[17]:= smat = ToSSparseMatrix[dsMushroom];
RecordsSummary[SSparseMatrixToTriplets[smat], {"RowID", "Variable", "Value"}, "MaxTallies" → 12]
```

```
Out[18]= {
  1 RowID      2 Variable      3 Value
  1          24      bruises?      8124      white      21 402
  10         24      cap-Color      8124      smooth      12 668
  100        24      cap-Shape      8124      partial      8124
  1000       24      cap-Surface     8124      free         7914
  1001       24      edibility       8124      one          7488
  1002       24      , gill-Attachment 8124      , close       6812
  1003       24      gill-Color      8124      brown        6356
  1004       24      gill-Size       8124      broad        5612
  1005       24      gill-Spacing    8124      pink         5380
  1006       24      habitat        8124      False        4748
  1007       24      id            8124      silky        4676
  (Other)    194 712  (Other)        105 612  (Other)      103 796
}
```

The dataset was ingested with row IDs; here we drop them:

```
In[19]:= dsMushroom = dsMushroom[Values];
```

Wine quality

In this sub-section is ingested an additional dataset -- "WineQuality".

```
In[20]:= dsWine = GetMachineLearningDataset["WineQuality", "RowIDs" → True];
```

Here we modify the class label column to be categorical (and binary):

```
In[21]:= dsWine = dsWine[All, Join[#, <|"wineQuality" -> If[#wineQuality ≥ 7, "high", "low"] |>] &];
```

Here is random sample of records:

```
In[22]:= RandomSample[dsWine, 6]
```

	id	fixedAcidity	volatileAcidity	citricAcid	residualSugar	chlorides	freeSulfurDioxide	totalSulfurDioxide	density
583	583	7.3	0.3	0.22	6.4	0.056	44.	168.	0.9947
1412	1412	7.	0.2	0.34	5.7	0.035	32.	83.	0.9928
1491	1491	5.6	0.185	0.49	1.1	0.03	28.	117.	0.9918
4626	4626	8.3	0.49	0.23	6.65	0.034	6.	158.	0.99344
3611	3611	6.7	0.35	0.48	8.8	0.056	35.	167.	0.99628
676	676	6.1	0.28	0.25	6.9	0.056	44.	201.	0.9955

Here is the summary of the dataset in long form:

```
In[23]:= smat = ToSSparseMatrix[dsWine];
```

```
RecordsSummary[SSparseMatrixToTriplets[smat], {"RowID", "Variable", "Value"}, "MaxTallies" → 12]
```

1 RowID	2 Variable	3 Value
1	13 alcohol	4898 low 3838
10	13 chlorides	4898 high 1060
100	13 density	4898 0.28 558
1000	13 fixedAcidity	4898 0.3 536
1001	13 freeSulfurDioxide	4898 0.32 493
1002	13 , id	4898 , 0.26 463
1003	13 pH	4898 0.27 447
1004	13 residualSugar	4898 0.34 444
1005	13 sulphates	4898 0.24 435
1006	13 totalSulfurDioxide	4898 0.36 401
1007	13 volatileAcidity	4898 0.29 400
(Other)	63 512 (Other)	9777 (Other) 54 580

The dataset was ingested with row IDs; here we drop them:

```
In[25]:= dsWine = dsWine[Values];
```

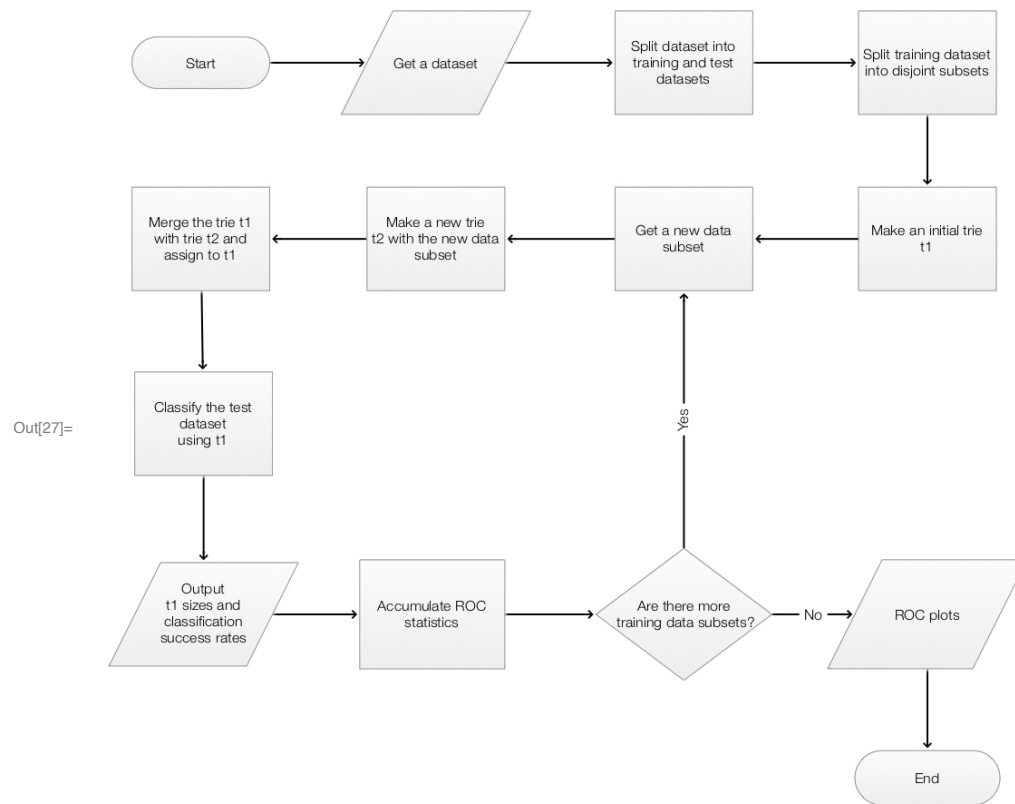
General progressive learning simulation loop

In this notebook we simulate the data streams given to Progressive learning algorithms.

We are going to use the following steps (based on Tries with Frequencies, [AA1]):

1. Get a dataset.
2. Split the dataset into training and test datasets.
3. Split the training dataset into disjoint datasets.
4. Make an initial trie t_1 .
5. Get a new training data subset.
6. Make a new trie t_2 with the new dataset.
7. Merge trie t_1 with trie t_2 .
8. Classify the test data-set using t_1 .
9. Output sizes of t_1 and classification success rates.
10. Accumulate ROC statistics.
11. Are there more training data subsets?
 - 11.1. If "Yes" go to step 5.
 - 11.2. If "No" go to step 12.
12. Display ROC plots.

The flow chart below follows the sequence of steps given above.



Data sorting

We sort the training data and the training and sample indices in order to exaggerate the Progressive learning effect.

With the data stream based on sorted data in the initial Progressive learning stages not all class labels and variable correlations would be seen.

```

In[29]:= ordInds = Ordering[Normal@dsTitanic[All, 2 ;; -2]];
         dsTitanic = dsTitanic[ordInds];

In[31]:= ordInds = Ordering[Normal@dsMushroom[All, 2 ;; -2]];
         dsMushroom = dsMushroom[ordInds];

In[33]:= ordInds = Ordering[Normal@dsWine[All, 2 ;; -2]];
         dsWine = dsMushroom[ordInds];

```

On the choice of algorithms

In this section we explain how Progressive learning classification can be done with two simple algorithms machine learning algorithms: Tries with Frequencies and Nearest Neighbors.

Why use Tries with Frequencies?

Tries with Frequencies fit Progressive Learning pretty well. The incremental growth of a trie is exactly in the sequential, data streaming manner of Progressive Learning.

The procedure in this sub-section is supported by operations of the packages [AAp2, AAp3].

Let us create an example comprised of the following two steps:

- a trie is made with a few dozen records, and
- that trie is extended with a new, single record.

For clarity in the rest of the sub-section we are going to drop the column “passengerAge” from the “Titanic” records.

```

In[35]:= trainingColumnNames = {"passengerClass", "passengerSex", "passengerSurvival"};

```

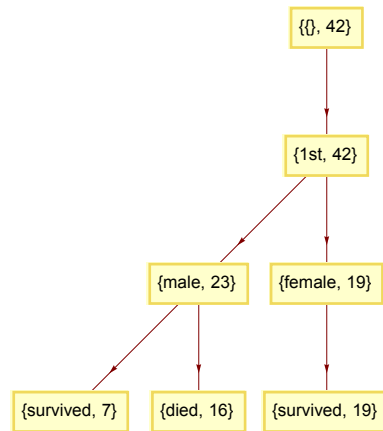
Here we create a trie with a sample of records of the “Titanic” dataset, dsTitanic:

```

In[36]:= SeedRandom[343]
rInds = RandomInteger[{1, 300}, 42];
tr = TrieCreate[Normal[dsTitanic[rInds, trainingColumnNames][Values]]];
TrieForm[tr, ImageSize -> {Automatic, 250}]

```

Out[39]=



Let us add a new record to that first trie. Here is another record:

```

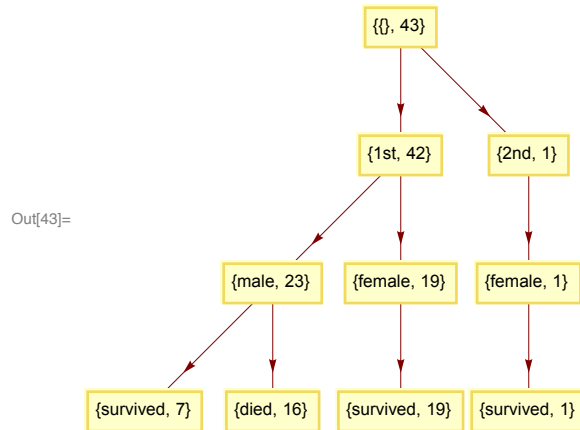
In[40]:= newInd = {RandomInteger[{301, 400}]}];
Normal[dsTitanic[newInd, trainingColumnNames][Values]]

```

Out[41]= { { 2nd, female, survived } }

And here is how the new, updated trie looks like:

```
In[42]:= tr2 = TrieMerge[tr, TrieCreate[Normal[dsTitanic[newInd, trainingColumnNames][Values]]]];
TrieForm[tr2, ImageSize -> {Automatic, 250}]
```



Remark: Note that TF can be used to find conditional probabilities of the record elements.

Here we classify a (made up, partial) record with the trie made so far:

```
In[44]:= TrieClassify[tr2, {"1st", "male"}, "Probabilities"] // N
```

```
Out[44]= < | died -> 0.695652, survived -> 0.304348 | >
```

(The function `TrieClassify` behaves like `Classify` with built-in classifiers.)

At this point it is clear that Tries with Frequencies can be directly applied to Progressive learning in a fairly straightforward way.

Why use a nearest neighbors classifier?

In this sub-section we are going to present a PLA algorithm based on Nearest Neighbours (NNs) and sparse matrix linear algebra. We are going to use extensively the package `SSparseMatrix.m`, [AAp8], that provides sparse matrices with named columns and rows and corresponding related operations.

The procedure in this sub-section is supported by methods of the Sparse Matrix Recommender objects of the package [AAp4].

Here are 6 random records of the “Titanic” dataset, `dsTitanic`.

```
In[45]:= SeedRandom[143]
rInds = RandomInteger[{1, Length[dsTitanic]}, 10];
dsTitanic[TakeDrop[rInds, 6][[1]]]
```

Out[47]=

id	passengerClass	passengerAge	passengerSex	passengerSurvival
96	1st	50	female	survived
213	1st	40	male	died
298	1st	-1	female	survived
877	3rd	-1	male	died
559	2nd	20	female	survived
701	3rd	20	male	died

Assume that these records are the only records our PLA has seen so far.

Consider the following matrix made from those records. Each row corresponds of to a record and the values of the “id” column are row names. Note that the unique values of each column are “unfolded” into separate columns.

```
In[48]:= smat = ColumnBind@@Map[ToSSparseMatrix[CrossTabulate[dsTitanic[TakeDrop[rInds, 6][[1]], {"id", #}]]] &,
Rest[Normal[Keys[dsTitanic[1]]]]];
MatrixForm[
smat]
```

Out[49]//MatrixForm=

	1st	2nd	3rd	-1	20	40	50	female	male	died	survived
96	1	0	0	0	0	0	1	1	0	0	1
213	1	0	0	0	0	1	0	0	1	1	0
298	1	0	0	1	0	0	0	1	0	0	1
559	0	1	0	0	1	0	0	1	0	0	1
701	0	0	1	0	1	0	0	0	1	1	0
877	0	0	1	1	0	0	0	0	1	1	0

Assume we see a new set of records and make the corresponding matrix:

```
In[50]:= smat2 = ColumnBind @@ Map[ToSSparseMatrix[CrossTabulate[dsTitanic[TakeDrop[rInds, 6][[2]], {"id", #}]]] &,
      Rest[Normal[Keys[dsTitanic[1]]]]];
MatrixForm[
  smat2]
```

Out[51]//MatrixForm=

	1st	3rd	20	30	40	female	male	died	survived
36	1	0	0	0	1	1	0	0	1
196	1	0	1	0	0	1	0	0	1
960	0	1	0	1	0	0	1	1	0
1299	0	1	0	0	1	0	1	1	0

Let us combine by *row binding* the old matrix and the new matrix into one. For this we need to make sure they have the same column names in both matrices.

```
In[52]:= allColNames = Union[Join[ColumnNames[smat], ColumnNames[smat2]]];
smat = ImposeColumnNames[smat, allColNames];
smat2 = ImposeColumnNames[smat2, allColNames];
smat = RowBind[smat, smat2];
MatrixForm[smat]
```

Out[56]//MatrixForm=

	-1	1st	20	2nd	30	3rd	40	50	died	female	male	survived
96	0	1	0	0	0	0	0	1	0	1	0	1
213	0	1	0	0	0	0	1	0	1	0	1	0
298	1	1	0	0	0	0	0	0	0	1	0	1
559	0	0	1	1	0	0	0	0	0	1	0	1
701	0	0	1	0	0	1	0	0	1	0	1	0
877	1	0	0	0	0	1	0	0	1	0	1	0
36	0	1	0	0	0	0	1	0	0	1	0	1
196	0	1	1	0	0	0	0	0	0	1	0	1
960	0	0	0	0	1	1	0	0	1	0	1	0
1299	0	0	0	0	0	1	1	0	1	0	1	0

Here the matrix making command above is repeated for a certain single record from the dataset:

```
In[57]:= newInd = RandomSample[Complement[Range[Length[dsTitanic]], rInds], 1];
svec = ColumnBind @@
  Map[ToSSparseMatrix[CrossTabulate[dsTitanic[newInd, {"id", #}]]] &, Rest[Normal[Keys[dsTitanic[1]]]]];
MatrixForm[
  svec]
```

Out[59]//MatrixForm=

$$\left(\begin{array}{c|cccc} & 3rd & 30 & male & survived \\ \hline 614 & 1 & 1 & 1 & 1 \end{array} \right)$$

Let us drop the survival columns:

```
In[60]:= svec = svec[[All, Complement[ColumnNames[svec], {"died", "survived"}]]];
MatrixForm[svec]
```

Out[61]//MatrixForm=

$$\left(\begin{array}{c|ccc} & 30 & 3rd & male \\ \hline 614 & 1 & 1 & 1 \end{array} \right)$$

Here we combine the column names of smat and svec:

```
In[62]:= allColNames = Union[Join[ColumnNames[smat], ColumnNames[svec]]]
```

```
Out[62]= {-1, 1st, 20, 2nd, 30, 3rd, 40, 50, died, female, male, survived}
```

Note that generally not all of those column names are in smat and svec.

Here we the single row matrix, svec, is extended to have all of the columns:

```
In[63]:= svec = ImposeColumnNames[svec, allColNames];
MatrixForm[svec]
```

Out[64]//MatrixForm=

$$\left(\begin{array}{c|cccccccccccccc} & -1 & 1st & 20 & 2nd & 30 & 3rd & 40 & 50 & died & female & male & survived \\ \hline 614 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 1 & 0 \end{array} \right)$$

Here is how we find the NNs scores for the search vector svec:


```
In[65]:= res = smat.Transpose[svec];
```

```
MatrixForm[res]
```

```
Out[66]//MatrixForm=
```

	614
96	0
213	1
298	0
559	0
701	2
877	2
36	0
196	0
960	3
1299	2

Let us extract the actual records from the “Titanic” dataset.

```
In[67]:= TakeLargest[RowSumsAssociation[res], 2]
```

```
Out[67]= <| 960 → 3, 701 → 2 |>
```

```
In[68]:= dsTitanic[Select[MemberQ[ToExpression/@Keys[%], #id] &]]
```

```
Out[68]=
```

id	passengerClass	passengerAge	passengerSex	passengerSurvival
701	3rd	20	male	died
960	3rd	30	male	died

With the obtained records above we determine the class label to be assigned to the record represented with svec. (In this case “died”.)

Progressive learning classification by Tries with Frequencies

In order to have the code more general let use the variable ds for the selected dataset.

```
In[69]:= ds = dsTitanic;
```

Here we set the classification label column:

```
In[70]:= labelColumnName = "passengerSurvival";
```

Data separation and preparation

Here we split the data into training and test parts.

```
In[71]:= SeedRandom[1232]
{dsTraining, dsTest} = ds[Sort@#] & /@ TakeDrop[RandomSample[Range[Length[ds]]], Floor[Length[ds] .75]];
```

In general, when using a tree for classification that process might be sensitive of the order the variables, especially for data with smaller number of records and/or large number of variables.

That is why here we select a permutation. (And make sure that the selected class label variable is the last index in the permutation.)

```
In[73]:= perm = Complement[Range[1, Dimensions[ds][[2]], Flatten[Position[Normal@Keys@ds[1], "id" | labelColumnName]]];
perm = Join[perm, Flatten[Position[Normal@Keys@ds[1], labelColumnName]]]
```

```
Out[74]= {2, 3, 4, 5}
```

```
In[75]:= trTraining = Normal@dsTraining[[All, perm]] [Values];
trTest = Normal@dsTest[[All, perm]] [Values];
```

```
In[77]:= trTraining = Map[ToString, trTraining, {-1}] /. {_Missing -> "NA", x_?NumericQ -> ToString[x]};
trTest = Map[ToString, trTest, {-1}] /. {_Missing -> "NA", x_?NumericQ -> ToString[x]};
```

```
In[79]:= Dimensions[trTraining]
Dimensions[trTest]
```

```
Out[79]= {981, 4}
```

```
Out[80]= {328, 4}
```

Classification

Here are the training data splitting ranges:

```
In[81]:= splitRanges = Map[# + {1, 0} &,
  Partition[Union@Join[Range[0, 300, 100], Range[300, Length[dsTraining], 200], {Length[dsTraining]}], 2, 1]];
MatrixForm@ToSSparseMatrix[SparseArray@splitRanges, "ColumnNames" → {"Begin", "End"}]
```

Out[82]//MatrixForm=

Begin	End
1	100
101	200
201	300
301	500
501	700
701	900
901	981

```
In[83]:= (*Initial trie creation.*)
tr = TrieCreate[trTraining[[Span@@First[splitRanges], All]]];

(*For accumulation of ROC statistics.*)
aROCStats = <| |>;

(*Main loop.*)
Do[
  (*Train a new trie.*)
  tr2 = TrieCreate[trTraining[[Span@@rng, All]]];

  (*Merge the new trie with the current trie.*)
  tr = TrieMerge[tr, tr2];

  (*Convert frequencies to probabilities.*)
  ptr = TrieNodeProbabilities[tr];

  (*Classify with the trie.*)
  clRes = Map[TrieClassify[ptr, #, "Default" → "NA"] &, trTest[[All, 1 ;; Length[perm] - 1]]];
```

```

(*Compute success rates.*)
cPairs = Transpose[{Normal[dsTest[All, labelColumnName]], clRes}];
ctMat = ToSSparseMatrix@CrossTabulate[cPairs];

(*Proclaim.*)
Print[Style[StringJoin@ConstantArray["-", 100], Blue]];
Print[Style[Row[{"Iteration with range:", rng}], Blue]];
Echo[TrieNodeCounts[ptr], "Size of trie:"];
Echo[Tally[clRes], "Predicted tally:"];
Echo[Count[Equal@@@cPairs, True] / Length[cPairs] // N, "Accuracy:"];
Echo[Row[{MatrixForm[ctMat], " ", MatrixForm[
  ctMat * Transpose[SparseArray[Table[1. / RowSums[ctMat], Dimensions[ctMat][[2]]]]]], "Actual vs predicted:"];
(*Accumulate for ROC statistics.*)
clRes = TrieClassify[ptr, trTest[All, 1 ;; Length[perm] - 1], "Probabilities", "Default" -> "NA"];
clRes = Map[Join[AssociationThread[Normal[dsTest[All, labelColumnName]], 0], #] &, clRes];
clDS = If[labelColumnName == "passengerSurvival",
  Dataset[KeyDrop[clRes, "NA"]][All, <|"survived" -> #survived, "died" -> #died|> &],
  Dataset[KeyDrop[clRes, "NA"]]
];
aROCStats = Join[aROCStats, <|rng -> ROCValues[clDS, trTest[All, -1]]|>],
{rng, Rest@splitRanges}]

```

Iteration with range: {101, 200}

» Size of trie: <| total -> 41, internal -> 21, leaves -> 20 |>

» Predicted tally: {{survived, 36}, {died, 45}, {NA, 247}}

» Accuracy: 0.192073

» Actual vs predicted: $\left(\begin{array}{c|ccc} & \text{died} & \text{NA} & \text{survived} \\ \hline \text{died} & 28 & 177 & 1 \\ \text{survived} & 17 & 70 & 35 \end{array} \right), \left(\begin{array}{c|ccc} & \text{died} & \text{NA} & \text{survived} \\ \hline \text{died} & 0.135922 & 0.859223 & 0.00485437 \\ \text{survived} & 0.139344 & 0.57377 & 0.286885 \end{array} \right)$

Iteration with range: {201, 300}

» Size of trie: $\langle | \text{total} \rightarrow 80, \text{internal} \rightarrow 42, \text{leaves} \rightarrow 38 | \rangle$

» Predicted tally: $\{\{\text{survived}, 53\}, \{\text{died}, 90\}, \{\text{NA}, 185\}\}$

» Accuracy: 0.335366

» Actual vs predicted: $\left(\begin{array}{c|ccc} & \text{died} & \text{NA} & \text{survived} \\ \hline \text{died} & 58 & 147 & 1 \\ \text{survived} & 32 & 38 & 52 \end{array} \right), \left(\begin{array}{c|ccc} & \text{died} & \text{NA} & \text{survived} \\ \hline \text{died} & 0.281553 & 0.713592 & 0.00485437 \\ \text{survived} & 0.262295 & 0.311475 & 0.42623 \end{array} \right)$

Iteration with range: {301, 500}

» Size of trie: $\langle | \text{total} \rightarrow 112, \text{internal} \rightarrow 59, \text{leaves} \rightarrow 53 | \rangle$

» Predicted tally: $\{\{\text{survived}, 252\}, \{\text{died}, 76\}\}$

» Accuracy: 0.469512

» Actual vs predicted: $\left(\begin{array}{c|cc} & \text{died} & \text{survived} \\ \hline \text{died} & 54 & 152 \\ \text{survived} & 22 & 100 \end{array} \right), \left(\begin{array}{c|cc} & \text{died} & \text{survived} \\ \hline \text{died} & 0.262136 & 0.737864 \\ \text{survived} & 0.180328 & 0.819672 \end{array} \right)$

Iteration with range: {501, 700}

» Size of trie: $\langle | \text{total} \rightarrow 133, \text{internal} \rightarrow 68, \text{leaves} \rightarrow 65 | \rangle$

» Predicted tally: $\{\{\text{survived}, 201\}, \{\text{died}, 127\}\}$

» Accuracy: 0.594512

» Actual vs predicted: $\left(\begin{array}{c|cc} & \text{died} & \text{survived} \\ \hline \text{died} & 100 & 106 \\ \text{survived} & 27 & 95 \end{array} \right), \left(\begin{array}{c|cc} & \text{died} & \text{survived} \\ \hline \text{died} & 0.485437 & 0.514563 \\ \text{survived} & 0.221311 & 0.778689 \end{array} \right)$

Iteration with range: {701, 900}

» Size of trie: $\langle | \text{total} \rightarrow 143, \text{internal} \rightarrow 72, \text{leaves} \rightarrow 71 | \rangle$

» Predicted tally: $\{\{\text{survived}, 125\}, \{\text{died}, 203\}\}$

» Accuracy: 0.716463

» Actual vs predicted: $\left(\begin{array}{c|cc} & \text{died} & \text{survived} \\ \hline \text{died} & 158 & 48 \\ \text{survived} & 45 & 77 \end{array} \right), \left(\begin{array}{c|cc} & \text{died} & \text{survived} \\ \hline \text{died} & 0.76699 & 0.23301 \\ \text{survived} & 0.368852 & 0.631148 \end{array} \right)$

Iteration with range: {901, 981}

» Size of trie: $\langle | \text{total} \rightarrow 164, \text{internal} \rightarrow 83, \text{leaves} \rightarrow 81 | \rangle$

» Predicted tally: $\{\{\text{survived}, 100\}, \{\text{died}, 228\}\}$

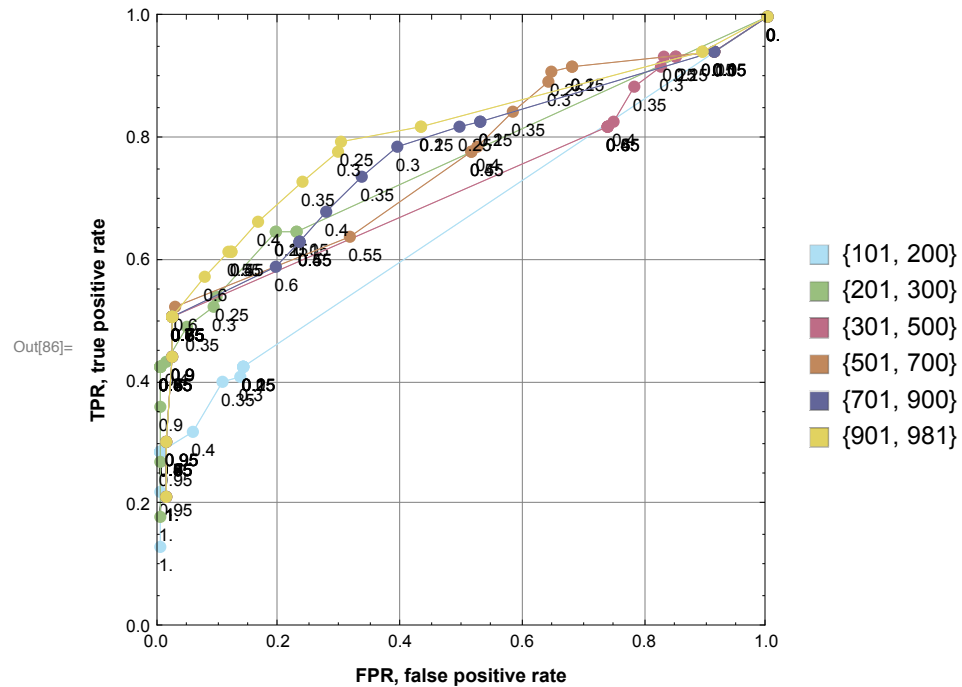
» Accuracy: 0.780488

» Actual vs predicted: $\left(\begin{array}{c|cc} & \text{died} & \text{survived} \\ \hline \text{died} & 181 & 25 \\ \text{survived} & 47 & 75 \end{array} \right), \left(\begin{array}{c|cc} & \text{died} & \text{survived} \\ \hline \text{died} & 0.878641 & 0.121359 \\ \text{survived} & 0.385246 & 0.614754 \end{array} \right)$

Plot ROC curves

Here we plot the Receiver Operating Characteristic (ROC) curves using the package [Aap7]:

```
In[86]:= ROCPlot[aROCStats, GridLines -> Automatic, PlotRange -> {{0, 1}, {0, 1}}]
```



We can see that with the Progressive learning process does improve its success rates in time.

Progressive learning classification by nearest neighbors

In order to have the code more general let use the variable `ds` for the selected dataset.

```
In[87]:= ds = dsTitanic;
```

Here we assign a class label column name and a computation parameter (should we re-weight the matrix terms or not):

```
In[88]:= labelColumnName = "passengerSurvival";
useTFIDFQ = True;
```

Data separation and preparation

```
In[90]:= SeedRandom[1232]
         {dsTraining, dsTest} = Sort[ds][Sort@#] & /@ TakeDrop[RandomSample[Range[Length[ds]]], Floor[Length[ds] .75]];

In[92]:= testRecords = Map[ToString /@ Values[KeyDrop[#, {"id", labelColumnName}]] &, Normal[dsTest]];
```

Classification

Here are the training data splitting ranges:

```
In[93]:= splitRanges = Map[# + {1, 0} &,
        Partition[Union@Join[Range[0, 300, 100], Range[300, Length[dsTraining], 200], {Length[dsTraining]}], 2, 1]];
        MatrixForm@ToSSparseMatrix[SparseArray@splitRanges, "ColumnNames" → {"Begin", "End"}]
```

Out[94]//MatrixForm=

Begin	End
1	100
101	200
201	300
301	500
501	700
701	900
901	981

```
In[95]:= (*First recommender object.*)
mainSMR = MakeItemRecommender["titanic", dsTraining[splitRanges[[1]], "id"];

(*For accumulation of ROC statistics.*)
aROCStats = <| |>;
aNNsROCStats = <| |>;

(*Main loop.*)
Do[
    (*Make a recommender object with the new data.*)
```



```

tempSMR = MakeItemRecommender["temp", dsTraining[Span@@rng], "id"];

(*Merge the new data recommender with the current one.*)
mainSMR["RowBind"] [tempSMR];

(*Reweight matrix terms if specified.*)
If[TrueQ[useTFIDFQ],
  mainSMR["M"] = WeightTerms[mainSMR["M"]]
];

(*Assign classifier parameters.*)
mainSMR["classificationParameters"] = <|"tagType" → labelColumnName,
  "nTopNNs" → 12, "voting" → False, "dropZeroScoreLabels" → True|>;

(*Classify with the current recommender.*)
clRes = ItemRecommenderClassify[mainSMR, #, "Scores"] & /@ testRecords;

(*Compute the success rates.*)
cPairs = Transpose[{Normal[dsTest[All, labelColumnName]], Map[First@*Keys, clRes]}];
ctMat = ToSSparseMatrix@CrossTabulate[cPairs];
ctMat = ImposeColumnNames[ctMat, RowNames[ctMat]];

(*Proclaim.*)
Print[Style[StringJoin@ConstantArray["-", 100], Blue]];
Print[Style[Row[{"Iteration with range:", rng}], Blue]];
Echo[Tally[First@*Keys /@ clRes], "Predicted tally:"];
Echo[Dimensions[mainSMR["M"]], "Dimensions[mainSMR[\"M\"]:]"];
Echo[Count[Equal@@@ cPairs, True] / Length[cPairs] // N, "Accuracy:"];
Echo[Row[{MatrixForm[ctMat], " ", MatrixForm[
  ctMat * Transpose[SparseArray[Table[1. / RowSums[ctMat], Dimensions[ctMat][[2]]]]]], "Actual vs predicted:"];

```

```


(*Accumulate for ROC statistics.*)
(*clRes=ItemRecommenderClassify[mainSMR,#,"TopProbabilities"→2]&/@testRecords;*)
clRes = ItemRecommenderClassify[mainSMR, #, "Scores"] & /@testRecords;
If[TrueQ[labelColumnName == "passengerSurvival"],
  clDS = Dataset[KeyDrop[clRes, "NA"]][All, <|"survived" → #survived, "died" → #died|> &],
  clDS = Dataset[clRes]
];
aROCStats = Join[aROCStats, <|rng -> ROCValues[clDS, Normal[dsTest[All, labelColumnName]]]|>,
{rng, Rest@splitRanges}]

```

 **ItemRecommender:** "MakeProfileVector": Some of the specified tags are not known in the ItemRecommender object.

 **ItemRecommender:** "MakeProfileVector": Some of the specified tags are not known in the ItemRecommender object.

 **ItemRecommender:** "MakeProfileVector": Some of the specified tags are not known in the ItemRecommender object.

 **General:** Further output of ItemRecommender::notag will be suppressed during this calculation.

Iteration with range:{101, 200}

» Predicted tally: {{survived, 147}, {died, 181}}

» Dimensions[mainSMR["M"]]: {102, 13}

» Accuracy: 0.631098

» Actual vs predicted: $\left(\begin{array}{c|cc} & \text{died} & \text{survived} \\ \hline \text{died} & 129 & 69 \\ \text{survived} & 52 & 78 \end{array} \right), \left(\begin{array}{c|cc} & \text{died} & \text{survived} \\ \hline \text{died} & 0.651515 & 0.348485 \\ \text{survived} & 0.4 & 0.6 \end{array} \right)$

Iteration with range:{201, 300}

» Predicted tally: {{survived, 161}, {died, 167}}

» Dimensions[mainSMR["M"]]: {202, 15}

» Accuracy: 0.728659

» Actual vs predicted: $\left(\begin{array}{c|cc} & \text{died} & \text{survived} \\ \hline \text{died} & 138 & 60 \\ \text{survived} & 29 & 101 \end{array} \right), \left(\begin{array}{c|cc} & \text{died} & \text{survived} \\ \hline \text{died} & 0.69697 & 0.30303 \\ \text{survived} & 0.223077 & 0.776923 \end{array} \right)$

Iteration with range:{301, 500}

» Predicted tally: {{survived, 116}, {died, 212}}

» Dimensions[mainSMR["M"]]: {402, 16}

» Accuracy: 0.737805

» Actual vs predicted: $\left(\begin{array}{c|cc} & \text{died} & \text{survived} \\ \hline \text{died} & 162 & 36 \\ \text{survived} & 50 & 80 \end{array} \right), \left(\begin{array}{c|cc} & \text{died} & \text{survived} \\ \hline \text{died} & 0.818182 & 0.181818 \\ \text{survived} & 0.384615 & 0.615385 \end{array} \right)$

Iteration with range:{501, 700}

» Predicted tally: {{survived, 105}, {died, 223}}

» Dimensions[mainSMR["M"]]: {602, 16}

» Accuracy: 0.734756

» Actual vs predicted: $\left(\begin{array}{c|cc} & \text{died} & \text{survived} \\ \hline \text{died} & 167 & 31 \\ \text{survived} & 56 & 74 \end{array} \right), \left(\begin{array}{c|cc} & \text{died} & \text{survived} \\ \hline \text{died} & 0.843434 & 0.156566 \\ \text{survived} & 0.430769 & 0.569231 \end{array} \right)$

Iteration with range:{701, 900}

» Predicted tally: {{survived, 90}, {died, 238}}

» Dimensions[mainSMR["M"]]: {802, 16}

» Accuracy: 0.737805

» Actual vs predicted: $\left(\begin{array}{c|cc} & \text{died} & \text{survived} \\ \hline \text{died} & 175 & 23 \\ \text{survived} & 63 & 67 \end{array} \right), \left(\begin{array}{c|cc} & \text{died} & \text{survived} \\ \hline \text{died} & 0.883838 & 0.116162 \\ \text{survived} & 0.484615 & 0.515385 \end{array} \right)$

Iteration with range:{901, 981}

» Predicted tally: {{survived, 90}, {died, 238}}

» Dimensions[mainSMR["M"]]: {883, 16}

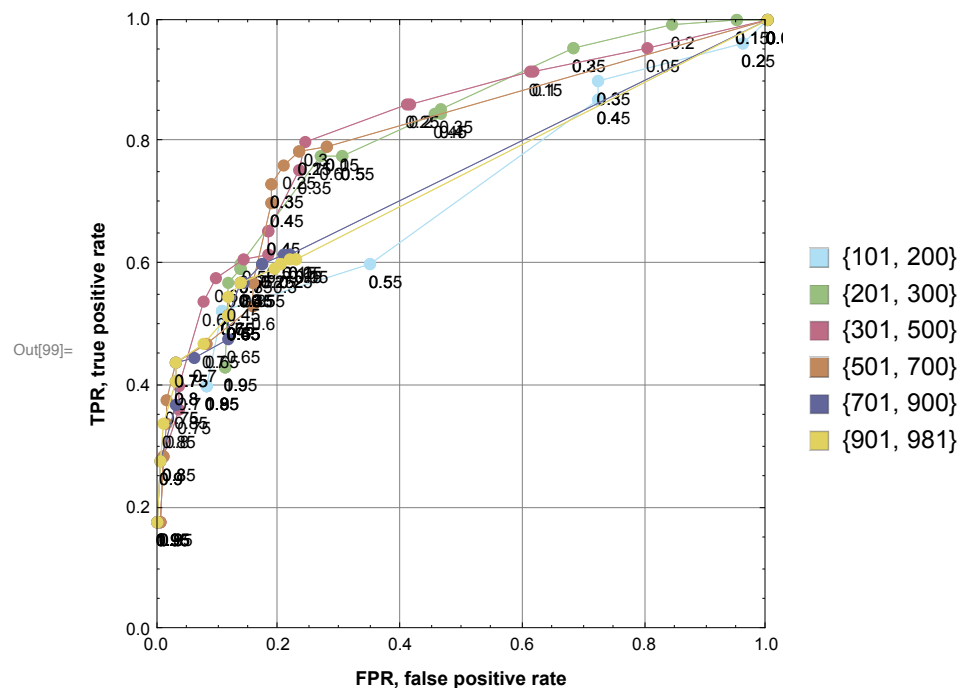
» Accuracy: 0.737805

» Actual vs predicted: $\left(\begin{array}{c|cc} & \text{died} & \text{survived} \\ \hline \text{died} & 175 & 23 \\ \text{survived} & 63 & 67 \end{array} \right), \left(\begin{array}{c|cc} & \text{died} & \text{survived} \\ \hline \text{died} & 0.883838 & 0.116162 \\ \text{survived} & 0.484615 & 0.515385 \end{array} \right)$

ROC curves

Here are ROC plots for the threshold of selection.

In[99]:= ROCPlot[aROCStats, "PlotJoined" → True, GridLines → Automatic, PlotRange → {{0, 1}, {0, 1}}]



References

Packages

- [AAp1] Anton Antonov, Obtain and transform Mathematica machine learning data-sets, `GetMachineLearningDataset.m`, (2018), *MathematicaVsR* at GitHub.
- [AAp2] Anton Antonov, Tries with frequencies Mathematica package, `TriesWithFrequencies.m`, (2013), *MathematicaForPrediction* at GitHub.
- [AAp3] Anton Antonov, Java tries with frequencies Mathematica package, `JavaTriesWithFrequencies.m`, (2017), *MathematicaForPrediction* at GitHub.
- [AAp4] Anton Antonov, Sparse matrix recommender framework in Mathematica, `SparseMatrixRecommenderFramework.m`, (2014), *MathematicaForPrediction* at GitHub.
- [AAp5] Anton Antonov, Variable importance determination by classifiers implementation in Mathematica, `VariableImportanceByClassifiers.m`, (2015), *MathematicaForPrediction* at GitHub.
- [AAp6] Anton Antonov, Receiver operating characteristic functions Mathematica package, `ROCFunctions.m`, (2016), *MathematicaForPrediction* at GitHub.
- [AAp7] Anton Antonov, MathematicaForPrediction utilities *Mathematica* package, `MathematicaForPredictionUtilities.m`, (2014), *MathematicaForPrediction* at GitHub.
- [AAp8] Anton Antonov, `SSparseMatrix` Mathematica package, `SSparseMatrix.m`, (2018), *MathematicaForPrediction* at GitHub.
- [AAp9] Anton Antonov, Implementation of document-term matrix construction and re-weighting functions in Mathematica, `DocumentTermMatrixConstruction.m`, (2013), *MathematicaForPrediction* at GitHub.

Articles

- [Wk1] Wikipedia entry, Online machine learning.
URL: https://en.wikipedia.org/wiki/Online_machine_learning.
- [Wk2] Wikipedia entry, K-means clustering.

URL: https://en.wikipedia.org/wiki/K-means_clustering .

[Wk3] Wikipedia entry, Naive Bayes classifier.

URL: https://en.wikipedia.org/wiki/Naive_Bayes_classifier .

[AA1] Anton Antonov, “Tries with frequencies in Java”, (2017), MathematicaForPrediction at WordPress.

URL: <https://mathematicaforprediction.wordpress.com/2017/01/31/tries-with-frequencies-in-java/> .

[AA2] Anton Antonov, “A Fast and Agile Item-Item Recommender: Design and Implementation”, (2011), Wolfram Technology Conference 2011.

URL: <http://library.wolfram.com/infocenter/Conferences/7964/> .