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 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[166]:=
```
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[174]:=
```
dsTitanic = GetMachineLearningDataset["Titanic", "RowIDs" → True];
```

Here is a sample of the dataset:

In[175]:=
```
RandomSample[dsTitanic, 6]
```

Out[175]=

	id	passengerClass	passengerAge	passengerSex	passengerSurvival
1169	1169	3rd	−1	male	died
135	135	1st	−1	female	survived
113	113	1st	30	female	survived
234	234	1st	60	female	survived
806	806	3rd	−1	male	survived
1014	1014	3rd	−1	female	died

Here is the dataset summary:

In[176]:=
```
RecordsSummary[dsTitanic]
```

Out[176]=

1 id	
Min	1
1st Qu	327.75
Mean	655
Median	655
3rd Qu	982.25
Max	1309

2 passengerClass	
3rd	709
1st	323
2nd	277

3 passengerAge	
Min	-1
1st Qu	10
Median	20
Mean	23.55
3rd Qu	40
Max	80

4 passengerSex	
male	843
female	466

5 passengerSurvival	
died	809
survived	500

Here is the summary of the dataset in long form:

In[177]:=
```
smat = ToSSparseMatrix[dsTitanic];
RecordsSummary[SSparseMatrixToTriplets[smat], {"RowID", "Variable", "Value"}]
```

Out[178]=

1 RowID	
1	5
10	5
100	5
1000	5
1001	5
1002	5
(Other)	6459

2 Variable	
id	1309
passengerClass	1309
passengerSex	1309
passengerSurvival	1309
passengerAge	1253

3 Value	
male	843
died	809
3rd	709
survived	500
female	466
20	335
(Other)	2827

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

In[179]:=
```
dsTitanic = dsTitanic[Values];
```

Mushroom

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

In[180]:=
```
dsMushroom = GetMachineLearningDataset["Mushroom", "RowIDs" → True];
```

Here is a random record of the dataset:

In[181]:=
```
RandomSample[dsMushroom, 1] // First
```

Out[181]=

id	6589
cap–Shape	flat
cap–Surface	smooth
cap–Color	red
bruises?	False
odor	spicy
gill–Attachment	free
gill–Spacing	close
gill–Size	narrow
gill–Color	buff
stalk–Shape	tapering
stalk–Root	NA
stalk–Surface–Above–Ring	smooth
stalk–Surface–Below–Ring	smooth
stalk–Color–Above–Ring	pink
stalk–Color–Below–Ring	white
veil–Type	partial
veil–Color	white
ring–Number	one
ring–Type	evanescent
spore–Print–Color	white
population	several
habitat	paths
edibility	poisonous

Here is the summary of the dataset in long form:

```
In[182]:= smat = ToSSparseMatrix[dsMushroom];
RecordsSummary[SSparseMatrixToTriplets[smat], {"RowID", "Variable", "Value"}, "MaxTallies" → 12]
```

Out[183]=

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[184]:= dsMushroom = dsMushroom[Values];
```

Wine quality

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

```
In[185]:= dsWine = GetMachineLearningDataset["WineQuality", "RowIDs" → True];
```

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

```
In[186]:= dsWine = dsWine[All, Join[#, <|"wineQuality" -> If[#wineQuality ≥ 7, "high", "low"]|>] &];
```

Here is random sample of records:

In[187]:=
```
Magnify[RandomSample[dsWine, 6], 0.6]
```

Out[187]=

	id	fixedAcidity	volatileAcidity	citricAcid	residualSugar	chlorides	freeSulfurDioxide	totalSulfurDioxide	density	pH
1880	1880	7.2	0.2	0.61	16.2	0.043	14.	103.	0.9987	3.06
3089	3089	6.3	0.37	0.37	1.5	0.024	12.	76.	0.98876	2.94
1833	1833	7.7	0.44	0.24	11.2	0.031	41.	167.	0.9948	3.12
1301	1301	9.	0.245	0.38	5.9	0.045	52.	159.	0.995	2.93
3495	3495	6.7	0.24	0.36	8.4	0.042	42.	123.	0.99473	3.34
2231	2231	7.1	0.2	0.36	11.6	0.042	45.	124.	0.997	2.92

Here is the summary of the dataset in long form:

In[188]:=
```
smat = ToSSparseMatrix[dsWine];
RecordsSummary[SSparseMatrixToTriplets[smat], {"RowID", "Variable", "Value"}, "MaxTallies" → 12]
```

Out[189]=

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[190]:=
```
dsWine = dsWine[Values];
```

General progressive learning simulation loop

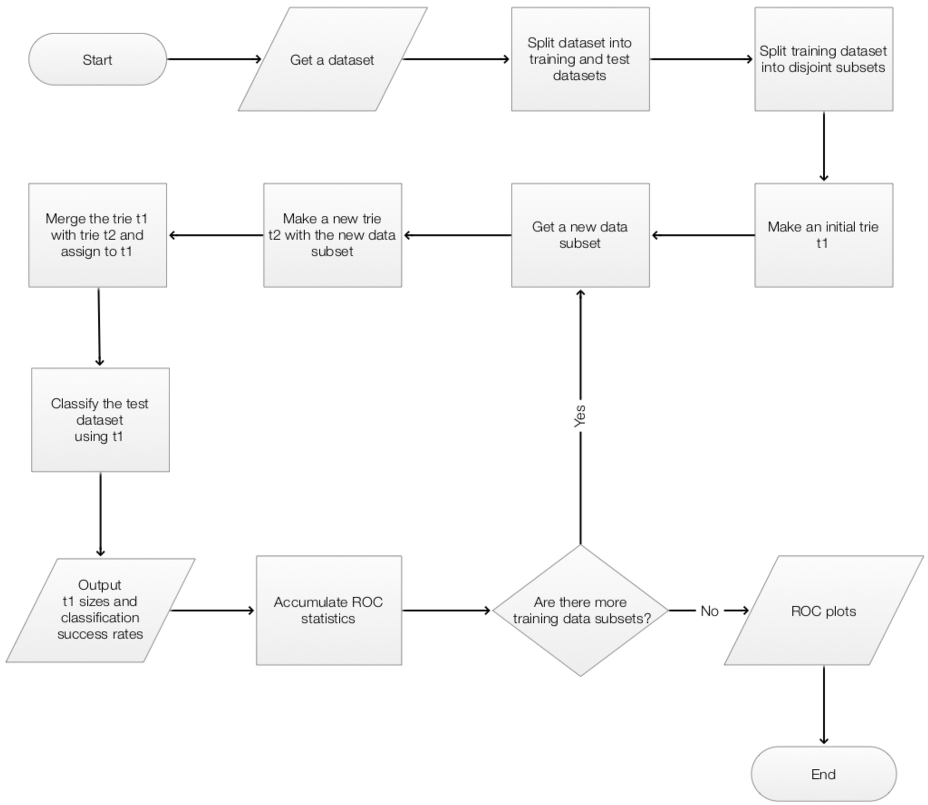
In this notebook we simulate the data streams given to Progressive learning algorithms.

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 `t1`.
5. Get a new training data subset.
6. Make a new trie `t2` with the new dataset.
7. Merge trie `t1` with trie `t2`.
8. Classify the test data-set using `t1`.
9. Output sizes of `t1` 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.

Out[192]=



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[194]:=
```
ordInds = Ordering[Normal@dsTitanic[All, 2 ;; -2]];
dsTitanic = dsTitanic[ordInds];
```

In[196]:=
```
ordInds = Ordering[Normal@dsMushroom[All, 2 ;; -2]];
dsMushroom = dsMushroom[ordInds];
```

In[198]:=
```
ordInds = Ordering[Normal@dsWine[All, 2 ;; -2]];
dsWine = dsWine[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.

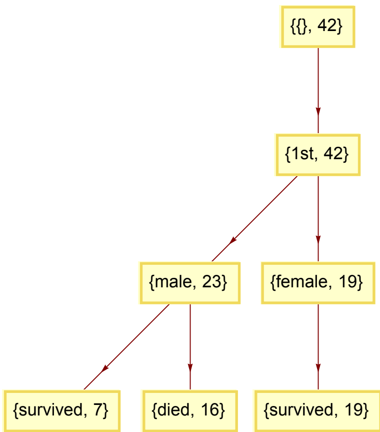
For clarity in the rest of the sub-section we are going to drop the column “passengerAge” from the “Titanic” records.

In[200]:=
```
trainingColumnNames = {"passengerClass", "passengerSex", "passengerSurvival"};
```

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

In[201]:=
```
SeedRandom[343]
rInds = RandomInteger[{1, 300}, 42];
tr = TrieCreate[Normal[dsTitanic[rInds, trainingColumnNames][Values]]];
TrieForm[tr, ImageSize → {Automatic, 250}]
```

Out[204]=



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

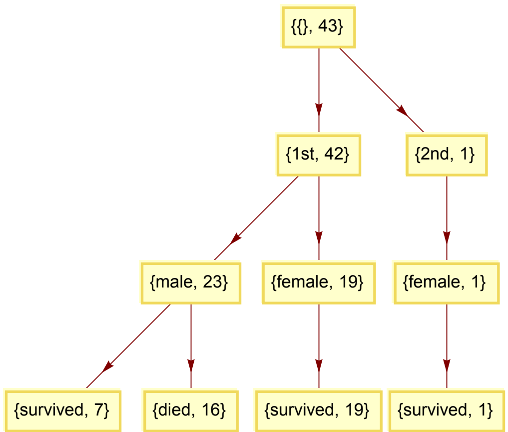
In[205]:=
```
newInd = {RandomInteger[{301, 400}]};
Normal[dsTitanic[newInd, trainingColumnNames][Values]]
```

Out[206]=
```
{{2nd, female, survived}}
```

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

In[207]:=
```
tr2 = TrieMerge[tr, TrieCreate[Normal[dsTitanic[newInd, trainingColumnNames][Values]]]];
TrieForm[tr2, ImageSize → {Automatic, 250}]
```

Out[208]=



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[209]:=
```
TrieClassify[tr2, {"1st", "male"}, "Probabilities"] // N
```

Out[209]=
```
<|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[210]:=
```
SeedRandom[143]
rInds = RandomInteger[{1, Length[dsTitanic]}, 10];
dsTitanic[TakeDrop[rInds, 6]〚1〛]
```

Out[212]=

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[213]:=
```
smat = ColumnBind @@ Map[ToSSparseMatrix[CrossTabulate[dsTitanic[TakeDrop[rInds, 6]〚1〛, {"id", #}]]] &,
    Rest[Normal[Keys[dsTitanic[1]]]]];
MatrixForm[
 smat]
```

Out[214]//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[215]:=
```
smat2 = ColumnBind @@ Map[ToSSparseMatrix[CrossTabulate[dsTitanic[TakeDrop[rInds, 6]⟦2⟧, {"id", #}]]] &,
    Rest[Normal[Keys[dsTitanic[1]]]]];
MatrixForm[
 smat2]
```

Out[216]//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[217]:=
```
allColNames = Union[Join[ColumnNames[smat], ColumnNames[smat2]]];
smat = ImposeColumnNames[smat, allColNames];
smat2 = ImposeColumnNames[smat2, allColNames];
smat = RowBind[smat, smat2];
MatrixForm[smat]
```

Out[221]//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[222]:=
```
newInd = RandomSample[Complement[Range[Length[dsTitanic]], rInds], 1];
svec = ColumnBind @@
   Map[ToSSparseMatrix[CrossTabulate[dsTitanic[newInd, {"id", #}]]] &, Rest[Normal[Keys[dsTitanic[1]]]]];
MatrixForm[
 svec]
```

Out[224]//MatrixForm=

	3rd	30	male	survived
614	1	1	1	1

Let us drop the survival columns:

In[225]:=
```
svec = svec[[All, Complement[ColumnNames[svec], {"died", "survived"}]]];
MatrixForm[svec]
```

Out[226]//MatrixForm=

	30	3rd	male
614	1	1	1

Here we combine the column names of `smat` and `svec`:

In[227]:=
```
allColNames = Union[Join[ColumnNames[smat], ColumnNames[svec]]]
```

Out[227]= {-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[228]:=
```
svec = ImposeColumnNames[svec, allColNames];
MatrixForm[svec]
```

Out[229]//MatrixForm=

	-1	1st	20	2nd	30	3rd	40	50	died	female	male	survived
614	0	0	0	0	1	1	0	0	0	0	1	0

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

In[230]:=
```
res = smat.Transpose[svec];
MatrixForm[res]
```

Out[231]//MatrixForm=

$$\left(\begin{array}{c|c} & 614 \\ \hline 96 & 0 \\ 213 & 1 \\ 298 & 0 \\ 559 & 0 \\ 701 & 2 \\ 877 & 2 \\ 36 & 0 \\ 196 & 0 \\ 960 & 3 \\ 1299 & 2 \end{array}\right)$$

Let us extract the actual records from the "Titanic" dataset.

In[232]:=
```
TakeLargest[RowSumsAssociation[res], 2]
```

Out[232]= <|960 → 3, 701 → 2|>

In[233]:=
```
dsTitanic[Select[MemberQ[ToExpression /@ Keys[%], #id] &]]
```

Out[233]=

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[234]:=
```
ds = dsTitanic;
```

Here we set the classification label column:

In[235]:=
```
labelColumnName = "passengerSurvival";
```

Here we determine the class label to focus on:

In[236]:=
```
focusLabel = SortBy[Normal[Tally[ds[All, labelColumnName]]], #[[2]] &][[1, 1]]
```

Out[236]=
```
survived
```

Data separation and preparation

Here we split the data into training and test parts.

In[237]:=
```
SeedRandom[1232]
{dsTraining, dsTest} = ds[Sort@#] & /@ TakeDrop[RandomSample[Range[Length[ds]]], Floor[Length[ds] .75]];
```

In general, when using a trie 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[239]:=
```
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[240]=
```
{2, 3, 4, 5}
```

In[241]:=
```
trTraining = Normal@dsTraining[[All, perm]][Values];
trTest = Normal@dsTest[[All, perm]][Values];
```

In[243]:=
```
trTraining = Map[ToString, trTraining, {-1}] /. {_Missing → "NA", x_?NumericQ :→ ToString[x]};
trTest = Map[ToString, trTest, {-1}] /. {_Missing → "NA", x_?NumericQ :→ ToString[x]};
```

In[245]:=
```
Dimensions[trTraining]
Dimensions[trTest]
```

Out[245]=
```
{981, 4}
```

Out[246]=
```
{328, 4}
```

Classification

Here are the training data splitting ranges:

In[247]:=
```
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[248]//MatrixForm=

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

In[249]:=
```
(*Inital 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 = Dataset[KeyDrop[clRes, "NA"]];
aROCStats = Join[aROCStats, <|rng → ROCValues[clDS, trTest⟦All, -1⟧, "ClassLabel" -> focusLabel]|>]
,
{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:

	died	NA	survived
died	28	177	1
survived	17	70	35

,

	died	NA	survived
died	0.135922	0.859223	0.00485437
survived	0.139344	0.57377	0.286885

Iteration with range:{201, 300}

» Size of trie: ⟨|total → 80, internal → 42, leaves → 38|⟩

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

» Accuracy: 0.335366

» Actual vs predicted:

	died	NA	survived
died	58	147	1
survived	32	38	52

,

	died	NA	survived
died	0.281553	0.713592	0.00485437
survived	0.262295	0.311475	0.42623

Iteration with range:{301, 500}

» Size of trie: ⟨|total → 112, internal → 59, leaves → 53|⟩

» Predicted tally: {{survived, 252}, {died, 76}}

» Accuracy: 0.469512

» Actual vs predicted:

	died	survived
died	54	152
survived	22	100

,

	died	survived
died	0.262136	0.737864
survived	0.180328	0.819672

Iteration with range:{501, 700}

» Size of trie: ⟨|total → 133, internal → 68, leaves → 65|⟩

» Predicted tally: {{survived, 201}, {died, 127}}

» Accuracy: 0.594512

» Actual vs predicted:

	died	survived
died	100	106
survived	27	95

,

	died	survived
died	0.485437	0.514563
survived	0.221311	0.778689

```
Iteration with range:{701, 900}
```

» Size of trie: ⟨|total → 143, internal → 72, leaves → 71|⟩

» Predicted tally: {{survived, 125}, {died, 203}}

» Accuracy: 0.716463

» Actual vs predicted:

	died	survived
died	158	48
survived	45	77

	died	survived
died	0.76699	0.23301
survived	0.368852	0.631148

```
Iteration with range:{901, 981}
```

» Size of trie: ⟨|total → 164, internal → 83, leaves → 81|⟩

» Predicted tally: {{survived, 100}, {died, 228}}

» Accuracy: 0.780488

» Actual vs predicted:

	died	survived
died	181	25
survived	47	75

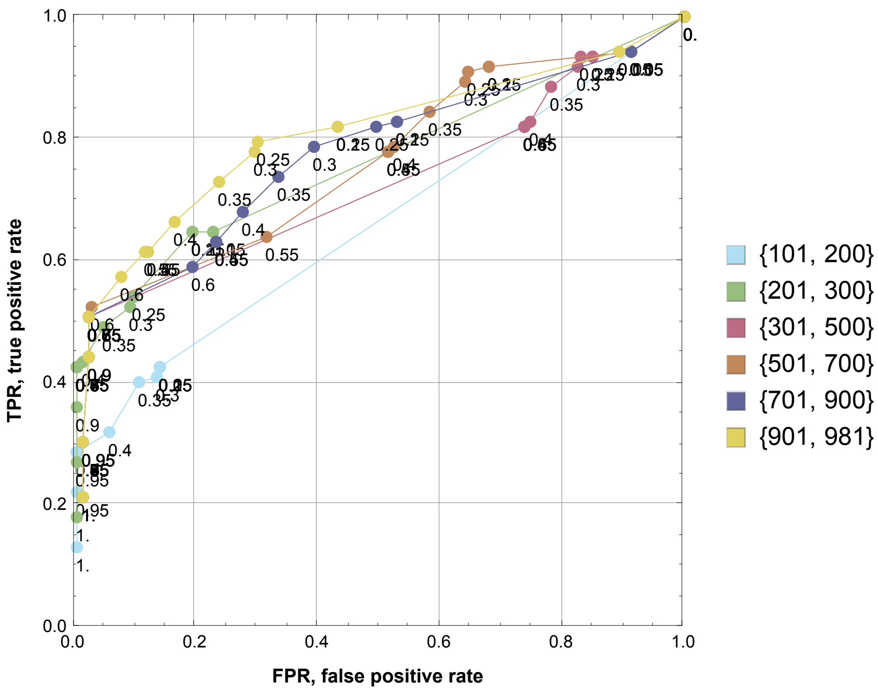
	died	survived
died	0.878641	0.121359
survived	0.385246	0.614754

Plot ROC curves

Here we plot the Receiver Operating Characteristic (ROC) curves using the package [AAp7]:

In[252]:=
```
ROCPlot[aROCStats, GridLines -> Automatic, PlotRange → {{0, 1}, {0, 1}}]
```

Out[252]=



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[253]:=
```
ds = dsTitanic;
```

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

In[254]:=
```
labelColumnName = "passengerSurvival";
useTFIDFQ = False;
```

Here we determine the class label to focus on:

```
In[256]:= focusLabel = SortBy[Normal[Tally[ds[All, labelColumnName]]], #[[2]] &][[1, 1]]
Out[256]= survived
```

Data separation and preparation

```
In[257]:= SeedRandom[1232]
{dsTraining, dsTest} = Sort[ds][Sort@#] & /@ TakeDrop[RandomSample[Range[Length[ds]]], Floor[Length[ds] .75]];
```

```
In[259]:= testRecords = Map[ToString /@ Values[KeyDrop[#, {"id", labelColumnName}]] &, Normal[dsTest]];
```

Classification

Here are the training data splitting ranges:

```
In[260]:= 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[261]//MatrixForm=

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

```
In[262]:= (*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, "dropZeroScoredLabels" → 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]]]|>];
(*ROC over the top NNs parameter.*)
nnsROCStats =
 Table[(
   clRes = ItemRecommenderClassify[mainSMR, #, "Scores", "nTopNNs" → i, "Normalize" → True] & /@ testRecords;
   clDS = Dataset[clRes];
   Join[ROCValues[clDS, Normal[dsTest[All, labelColumnName]],
      {0.7}, "ClassLabel" -> focusLabel][[1]], <|"ROCParameter" → i|>]
  ), {i, Range[5, 70, 5]}];
aNNsROCStats = Join[aNNsROCStats, <|rng → nnsROCStats|>],
{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, 260}, {died, 68}}

» Dimensions[mainSMR["M"]]: {102, 13}

» Accuracy: 0.52439

» Actual vs predicted:

	died	survived
died	55	143
survived	13	117

,

	died	survived
died	0.277778	0.722222
survived	0.1	0.9

Iteration with range:{201, 300}

» Predicted tally: {{survived, 156}, {died, 172}}

» Dimensions[mainSMR["M"]]: {202, 15}

» Accuracy: 0.762195

» Actual vs predicted:

	died	survived
died	146	52
survived	26	104

,

	died	survived
died	0.737374	0.262626
survived	0.2	0.8

Iteration with range:{301, 500}

» Predicted tally: {{survived, 96}, {died, 232}}

» Dimensions[mainSMR["M"]]: {402, 16}

» Accuracy: 0.756098

» Actual vs predicted:

	died	survived
died	175	23
survived	57	73

,

	died	survived
died	0.883838	0.116162
survived	0.438462	0.561538

Iteration with range:{501, 700}

» Predicted tally: {{survived, 96}, {died, 232}}

» Dimensions[mainSMR["M"]]: {602, 16}

» Accuracy: 0.75

» Actual vs predicted:

	died	survived
died	174	24
survived	58	72

,

	died	survived
died	0.878788	0.121212
survived	0.446154	0.553846

Iteration with range:{701, 900}

» Predicted tally: {{survived, 102}, {died, 226}}

» Dimensions[mainSMR["M"]]: {802, 16}

» Accuracy: 0.756098

» Actual vs predicted:

	died	survived
died	172	26
survived	54	76

,

	died	survived
died	0.868687	0.131313
survived	0.415385	0.584615

Iteration with range:{901, 981}

» Predicted tally: {{survived, 95}, {died, 233}}

» Dimensions[mainSMR["M"]]: {883, 16}

» Accuracy: 0.771341

» Actual vs predicted:

	died	survived
died	178	20
survived	55	75

,

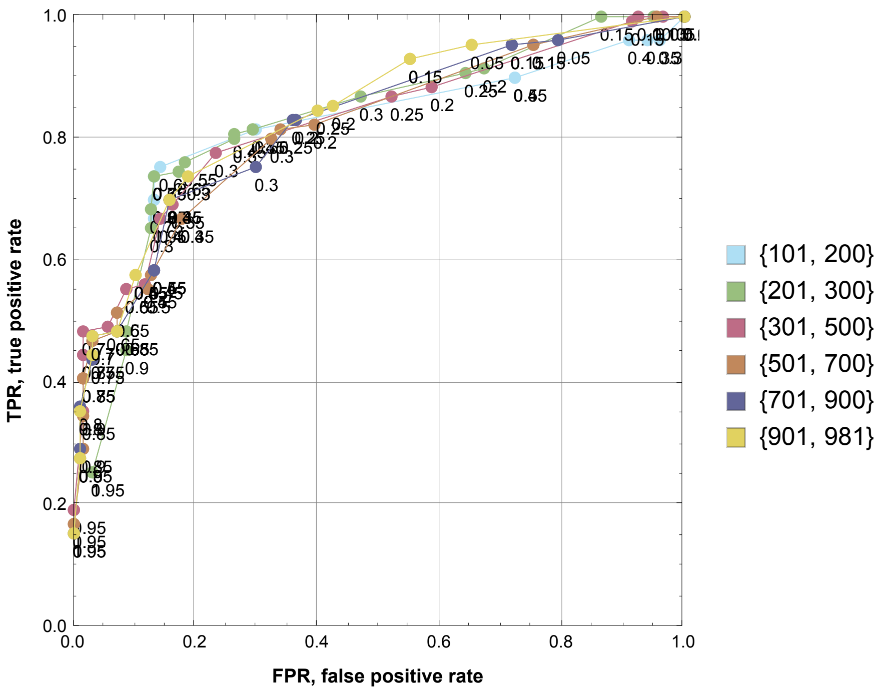
	died	survived
died	0.89899	0.10101
survived	0.423077	0.576923

ROC curves

Here are ROC plots for the threshold of selection.

In[266]:=
```
ROCPlot[aROCStats, "PlotJoined" → True, GridLines -> Automatic, PlotRange → {{0, 1}, {0, 1}}]
```

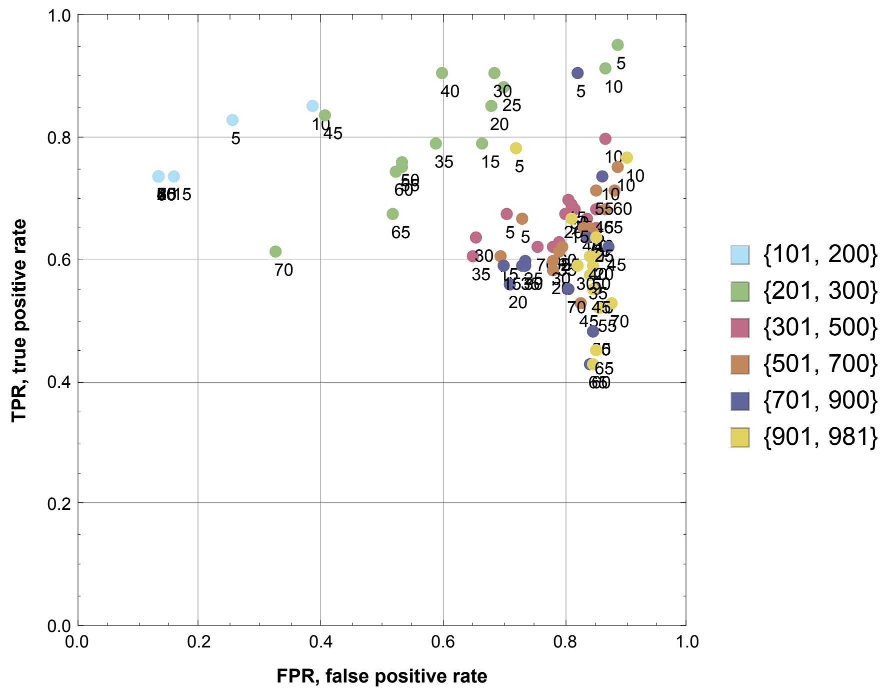
Out[266]=



Here are ROC plots over the number of top NNs used to determine the predicted class label.

In[267]:=
```
ROCPlot[aNNsROCStats, "PlotJoined" → False, GridLines -> Automatic, PlotRange → {{0, 1}, {0, 1}}]
```

Out[267]=



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/ .