Algebraic Ground Truth Inference Penn ML Benchmarks demonstrations:twonorm

The Penn ML benchmark `twonorm` is a binary classification dataset. This is an artificial or synthetic dataset consisting of 7,400 20-dimensional vectors drawn from two Gaussian distributions. For this demonstration we took 1000 from each of the two labels as our training set and tested on the 5,400 remaining records (2,703 “0” + 2,697 “1”).

Code

Ingesting the data from Github and initializing the mushroom data table

In[1]:=
```
datasetName = "twonorm"
filename =
  "https://github.com/EpistasisLab/penn-ml-benchmarks/blob/master/datasets/" <>
   datasetName <> "/" <> datasetName <> ".tsv.gz?raw=true";
tsvHeader = Import[filename, "TSV"] // First;
benchmarkData =
  Import[filename, "TSV"] // Rest // GroupBy[#, Last] & // Map[Most, #, {2}] &;
```

Out[1]=
```
twonorm
```

In[5]:=
```
tsvHeader
numberOfFeatures = Length@tsvHeader - 1
```

Out[5]=
```
{A1, A2, A3, A4, A5, A6, A7, A8, A9, A10, A11,
 A12, A13, A14, A15, A16, A17, A18, A19, A20, target}
```

Out[6]=
```
20
```

Training/testing code

In[7]:=
```
Clear[TrainClassifiers]
TrainClassifiers[classifiersData_, classifierTypes_List,
  nTrainTotal_Integer, nEachRandom_Integer] := Module[
  { trainingSamples, classifiers},
  (* All this code is standard Mathematica code.  The workhorse command is -
   Classify. We even just go with the default settings for
    the algoritms Mathematica implements with it *)
  (* Picking about 1/2 chance of overlap seems best with our current settings *)
  trainingSamples =
   Table[RandomSample[Range@nTrainTotal, nEachRandom], {Length@classifiersData}];
  classifiers = Transpose@{classifierTypes, Map[First, classifiersData, {2}] //
       MapIndexed[#1⟦trainingSamples⟦#2⟦1⟧⟧⟧ &, #, {2}] &} //
    Map[Classify[#⟦2⟧, Method → #⟦1⟧, TrainingProgressReporting → None] &, #] &;
  classifiers]
```

In[60]:=
```
Clear[LabelRMSE]
LabelRMSE[gtDiff_, label_] :=
 KeySelect[gtDiff, (Length@# == 3) &] // KeySelect[#, (Last@# == label) &] & // Values //
    Map[#^2 &, #] & // Mean // Sqrt
Clear[AlgebraicallyEvaluateClassifiers]
AlgebraicallyEvaluateClassifiers[classifiers_, classifiersData_] := Module[
  {nTestAlpha, nTestBeta, decisions,
   alphaSamples, betaSamples,
   byLabelDecisions, votingPatternCountsByLabel,
   sols, equationsToSolve, vars,
   gt, avgSol, gtDiff},
  (* Calculate the size of the test sets *)
  {nTestAlpha, nTestBeta} =
   classifiersData // First // Map[Length, #, {2}] & // {#[0], #[1]} & // Last /@ # &;
  (* We arbitrarily define "0" as the alpha label, and "1" as the beta label *)
  decisions =
   Table[classifiers⟦i⟧[classifiersData⟦i⟧ // Map[Last, #] & // Join[#[0], #[1]] &],
    {i, Length@classifiers}];
  alphaSamples = RandomSample[Range@nTestAlpha, nTestAlpha];
  betaSamples = RandomSample[Range@nTestBeta, nTestBeta];
  byLabelDecisions = decisions // Map[TakeDrop[#, 2703] &, #] & //
    Map[Association[{0 → #⟦1⟧⟦alphaSamples⟧, 1 → #⟦2⟧⟦betaSamples⟧}] &, #] &;
  votingPatternCountsByLabel = byLabelDecisions // Merge[#, Identity] & //
     Map[Transpose, #] & // Map[Counts, #] &;
  Map[Length, votingPatternCountsByLabel];
   sols = Table[
```

```
    equationsToSolve = MakeIndependentVotingIdeal[trio];
    vars = Variables /@ equationsToSolve // Flatten // DeleteDuplicates // Sort //
      Cases[#, Except[f__]] &;
    Solve[Map[(# == 0) &, equationsToSolve] /. VotingFrequenciesData[
        votingPatternCountsByLabel, trio], vars] // N,
    {trio, Subsets[Range@Length@classifiers, {3}]}];
  gt = Join[{Pα → (votingPatternCountsByLabel // Map[Values, #] & // Map[Total, #] & //
         #[0] / Total@# &)}, Table[Subscript[P, i, α] →
       DataGroundTruth[votingPatternCountsByLabel, 0, i], {i, Length@classifiers}],
     Table[Subscript[P, i, β] → DataGroundTruth[votingPatternCountsByLabel, 1, i],
      {i, Length@classifiers}]] //
    Association;
  avgSol =
   Select[sols, IsAPhysicalSolutionQ] // Map[PickHighestAverageAccuracy, #] & //
        Flatten // GroupBy[#, First] & // Map[Last, #, {2}] & //
     Map[Mean, #] & // KeySortBy[#, Last@# &] &;
  gtDiff = Merge[{gt // N, -avgSol}, Total];
  {Pα /. gtDiff, LabelRMSE[gtDiff, α], LabelRMSE[gtDiff, β], gt, avgSol}
 ]
```

Writing out the independent evaluation ideal generator set

```
In[13]:= Clear[MakeIndependentVotingIdeal]
         MakeIndependentVotingIdeal[{i_, j_, k_}] :=
```

$$\{P_\alpha P_{i,\alpha} P_{j,\alpha} P_{k,\alpha} + (1-P_\alpha)\times(1-P_{i,\beta})\times(1-P_{j,\beta})\times(1-P_{k,\beta}) - f_{\alpha,\alpha,\alpha},$$
$$P_\alpha P_{i,\alpha} P_{j,\alpha} (1-P_{k,\alpha}) + (1-P_\alpha)\times(1-P_{i,\beta})\times(1-P_{j,\beta}) P_{k,\beta} - f_{\alpha,\alpha,\beta},$$
$$P_\alpha P_{i,\alpha} (1-P_{j,\alpha}) P_{k,\alpha} + (1-P_\alpha)\times(1-P_{i,\beta}) P_{j,\beta} (1-P_{k,\beta}) - f_{\alpha,\beta,\alpha},$$
$$P_\alpha P_{i,\alpha} (1-P_{j,\alpha})\times(1-P_{k,\alpha}) + (1-P_\alpha)\times(1-P_{i,\beta}) P_{j,\beta} P_{k,\beta} - f_{\alpha,\beta,\beta},$$
$$P_\alpha (1-P_{i,\alpha}) P_{j,\alpha} P_{k,\alpha} + (1-P_\alpha) P_{i,\beta} (1-P_{j,\beta})\times(1-P_{k,\beta}) - f_{\beta,\alpha,\alpha},$$
$$P_\alpha (1-P_{i,\alpha}) P_{j,\alpha} (1-P_{k,\alpha}) + (1-P_\alpha) P_{i,\beta} (1-P_{j,\beta}) P_{k,\beta} - f_{\beta,\alpha,\beta},$$
$$P_\alpha (1-P_{i,\alpha})\times(1-P_{j,\alpha}) P_{k,\alpha} + (1-P_\alpha) P_{i,\beta} P_{j,\beta} (1-P_{k,\beta}) - f_{\beta,\beta,\alpha},$$
$$P_\alpha (1-P_{i,\alpha})\times(1-P_{j,\alpha})\times(1-P_{k,\alpha}) + (1-P_\alpha) P_{i,\beta} P_{j,\beta} P_{k,\beta} - f_{\beta,\beta,\beta}\}$$

In[15]:=
```
Clear[DataGroundTruth]
DataGroundTruth[data_List, label_, classifier_Integer] := Module[
  {withoutHeaderData = Rest@data, byLabel},
  byLabel = GroupBy[withoutHeaderData, #[[5]] &];
  Map[#[[{1, 2, 3, 4, 7}]] &, byLabel, {2}] //
        #[label] & //
       Map[#[[{classifier, -1}]] &, #] & //
      GroupBy[#, First] & //
     Map[Last, #, {2}] & //
    Total /@ # & //
   #[label] / Total@# &]
DataGroundTruth[data_Association, label_, classifier_Integer] := Module[
  {},
  data[label] // Normal // Map[(#[[1]][[classifier]] → #[[2]]) &, #] & //
     Map[Association, #] & // Merge[#, Total] & // #[label] / Total@# &]
```

In[18]:=
```
Clear[VotingFrequenciesData]
VotingFrequenciesData[data_List,
  classifiers : {_Integer, _Integer, _Integer}] := Module[
  {withoutHeaderData = Rest@data, eventCounts},
  eventCounts = GroupBy[withoutHeaderData, #[[classifiers]] &] //
     Map[Last, #, {2}] & //
    Total /@ # &;
  KeyMap[(Subscript[f, Sequence @@ #] /. {0 → α, 1 → β}) &, eventCounts] //
   # / (Total@#) &]
VotingFrequenciesData[data_Association, classifiers : {_Integer ..}] := Module[
  {eventCounts},
  eventCounts = Values@data // Merge[#, Identity] & // Map[Total, #] & //
         KeyValueMap[(#1[[classifiers]] → #2) &, #] & //
        GroupBy[#, First] & //
       Map[Last, #, {2}] & //
      Map[Total, #] & //
     KeyMap[Subscript[f, Sequence @@ Map[If[# == 0, α, β] &, #]] &, #] & //
    # / Total@# &]
```

Queries

In[21]:=
```
Clear[IsARealSolutionQ]
IsARealSolutionQ[sols_] :=
 If[Length@sols > 0, First@sols // Map[#⟦2⟧ &, #] & // Cases[#, _Complex] & //
   If[Length@# > 0, False, True] &,
  False]
Clear[IsAPhysicalSolutionQ]
IsAPhysicalSolutionQ[sols_] := And[IsARealSolutionQ[sols],
  First@sols // Map[#⟦2⟧ &, #] & // Map[(0 < # < 1) &, #] & // And @@ # &]
```

In[25]:=
```
Clear[PickHighestAverageAccuracy]
PickHighestAverageAccuracy[trioSols_] :=
 Map[Cases[#, HoldPattern[P_,_ → _]] &, trioSols] //
      Values /@ # & // Map[Mean, #] & // MapIndexed[{#1, #2⟦1⟧} &, #] & // Sort //
   Reverse // trioSols⟦Last@First@#⟧ &
Clear[PickHighestPrevalence]
PickHighestPrevalence[trioSols_] := If[Length@trioSols == 2,
  Map[({trioSols⟦#⟧ // Association // #[Pα] &, #}) &, Range@2] //
    Sort // trioSols⟦Last@Last@#⟧ &, Nothing]
```

A single evaluation

The first demonstration will evaluate a single test sample. This will introduce the reader to the steps necessary to carry out a purely algebraic evaluation of noisy binary classifiers when one does not have the ground truth. It will also make clear the design choices we make in carrying out an experimental demonstration.

1a. Train an ensemble of classifiers trying to make them as error independent as possible

The algebra we will use in this demonstration is the simplest case possible - the case of classifiers that are making errors independently of each other. Perfect independence at all times and for all samples would be nearly impossible to accomplish in the wild. The simpler task we want to accomplish in this demonstration notebook is to design classifiers that are nearly independent as possible.
This is accomplished here by following three design criteria:
1. Use disjoint feature sets for each classifier in the ensemble.
2. Use different algorithms for each classifier.
3. Try to use disjoint training sets for the classifiers.
Criteria 1 and 2 are easy enough to carry out. Implementing criteria 3 is harder because of the small dataset sizes. We partially carry it out by selecting random records from a common training pool so that the shared training records can be minimized.

This features 4 way split was found experimentally by scanning for partitions that resulted in classifiers that returned sensible algebraic answers. That test is discussed below in more detail and is one of the strengths of the algebraic approach - it can fail to return sensible answers. For now, take this as our preferred candidate for making classifiers that are independent enough on this dataset. You, of course, are free to experiment with our approach and perhaps find a much better partition of features that will get the classifiers closer to error independence.

In[29]:=
```
classifiersFeatures =
 {{2, 15, 5, 7}, {3, 17, 9, 11}, {20, 4, 1, 8}, {12, 16, 13, 19}, {14, 6, 18, 10}}
```

Out[29]=
```
{{2, 15, 5, 7}, {3, 17, 9, 11}, {20, 4, 1, 8}, {12, 16, 13, 19}, {14, 6, 18, 10}}
```

In[64]:=
```
trainTestSplit = Map[TakeDrop[RandomSample@#, 1000] &, benchmarkData];
(* Each classifier is now trained and tested only using its specific features *)
classifiersData =
  Table[Map[#[[features]] &, trainTestSplit, {3}], {features, classifiersFeatures}];
classifierTypes = {{"NeuralNetwork", "NetworkDepth" → 5}, "GradientBoostedTrees",
  "NaiveBayes", "LogisticRegression", "SupportVectorMachine"}
classifiers = TrainClassifiers[classifiersData, classifierTypes, 1000, 800]
{prevError, alphaRMSE, betaRMSE, gt, avgSol} =
  AlgebraicallyEvaluateClassifiers[classifiers, classifiersData];
```

Out[66]=
```
{{NeuralNetwork, NetworkDepth → 5}, GradientBoostedTrees,
 NaiveBayes, LogisticRegression, SupportVectorMachine}
```

Out[67]=
```
{ClassifierFunction[ Input type: NumericalVector (length: 4)
                     Classes: 0, 1 ],
 ClassifierFunction[ Input type: NumericalVector (length: 4)
                     Classes: 0, 1 ],
 ClassifierFunction[ Input type: NumericalVector (length: 4)
                     Classes: 0, 1 ],
 ClassifierFunction[ Input type: NumericalVector (length: 4)
                     Classes: 0, 1 ],
 ClassifierFunction[ Input type: NumericalVector (length: 4)
                     Classes: 0, 1 ]}
```

How good was the purely algebraic evaluation of the binary classifiers?

In[69]:=
```
Grid[{{"Pα estimation error", "α accuracies RMSE", "β accuracies RMSE"},
  {prevError, alphaRMSE, betaRMSE}}, Dividers → All]
```

Out[69]=

P_α estimation error	α accuracies RMSE	β accuracies RMSE
0.00764635	0.00607469	0.00757921

Impressive results, but keep in mind that this is a synthetic dataset so we should expect these very clean results. Nonetheless, how well did it estimate each performance ground truth statistic?

In[99]:=
```
(* The prevalence error estimate *)
Grid[{{"Pα ground truth value", gt // #[Pα] & // N},
   {"Pα algebraic GTI", avgSol[Pα] // N}}, Dividers → All,
  Background → {None, {Green, LightGreen}}]
(* The alpha label accuracy estimates *)
alphaAccuracies = Table[Pi,α, {i, 5}];
Join[{{"α label accuracies", "ground truth", "algebraic estimate"}},
    Table[{accuracy, gt[accuracy] // N, avgSol[accuracy]},
     {accuracy, alphaAccuracies}]] //
   Transpose // Grid[#, Dividers → All, Background →
     {None, {LightBlue, Green, LightGreen}}] &
betaAccuracies = Table[Pi,β, {i, 5}];
Join[{{"β label accuracies", "ground truth", "algebraic estimate"}}, Table[
     {accuracy, gt[accuracy] // N, avgSol[accuracy]}, {accuracy, betaAccuracies}]] //
   Transpose // Grid[#, Dividers → All, Background →
     {None, {LightBlue, Green, LightGreen}}] &
```

Out[99]=

P_α ground truth value	0.500556
P_α algebraic GTI	0.492909

Out[101]=

α label accuracies	$P_{1,\alpha}$	$P_{2,\alpha}$	$P_{3,\alpha}$	$P_{4,\alpha}$	$P_{5,\alpha}$
ground truth	0.81724	0.803922	0.805401	0.796152	0.780614
algebraic estimate	0.814125	0.805416	0.815964	0.799477	0.787682

Out[103]=

β label accuracies	$P_{1,\beta}$	$P_{2,\beta}$	$P_{3,\beta}$	$P_{4,\beta}$	$P_{5,\beta}$
ground truth	0.798294	0.811272	0.812013	0.822395	0.826103
algebraic estimate	0.78484	0.802967	0.808055	0.824687	0.822064

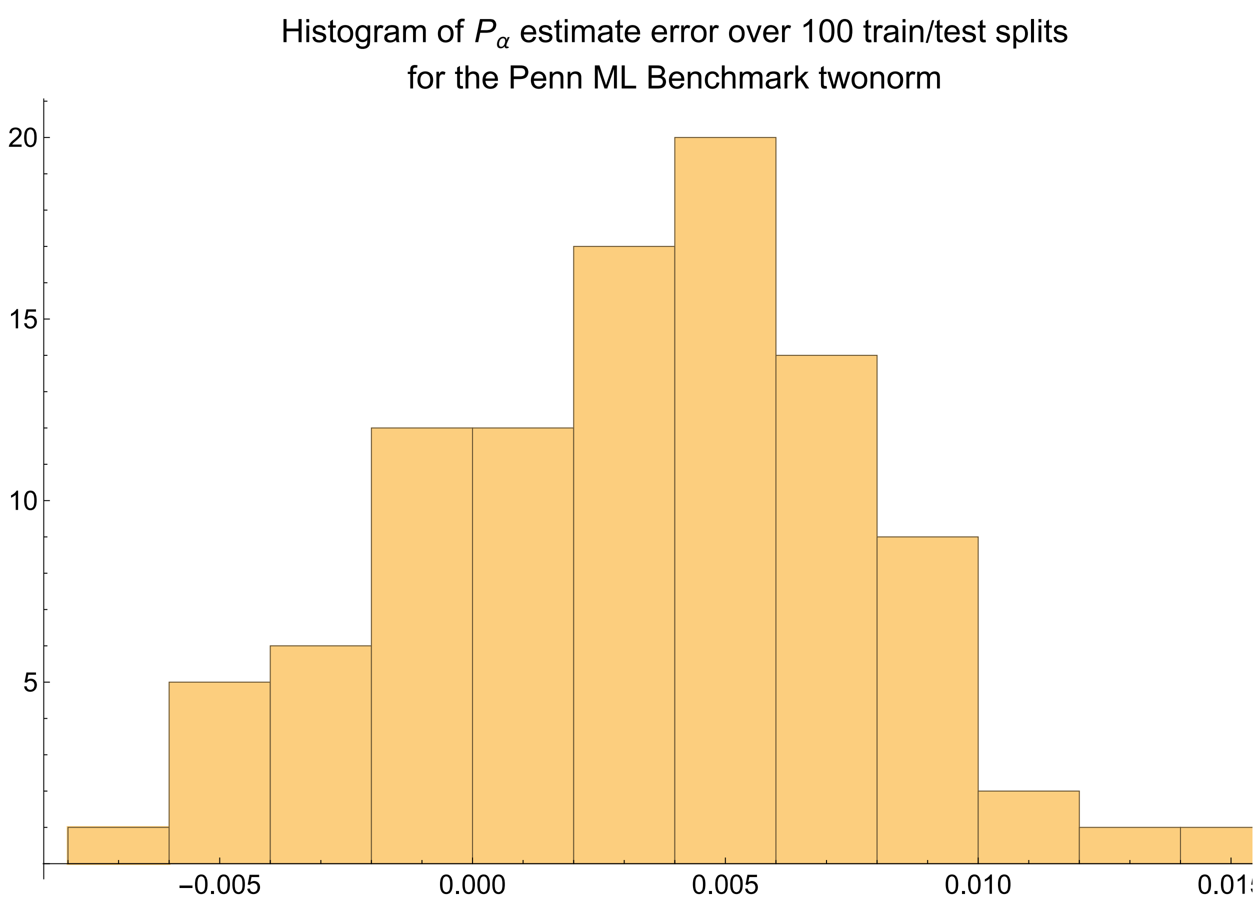
RMSE histogram for algebraic evaluation under random train/test splits of twonorm

The one-shot evaluation above is impressive. But maybe it is an exception. What happens when we do repeated random train/test splits with the twonorm dataset?

```
classifierTypes = {{"NeuralNetwork", "NetworkDepth" → 5}, "GradientBoostedTrees",
   "NaiveBayes", "LogisticRegression", "SupportVectorMachine"};
Monitor[runs = Table[
    trainTestSplit = Map[TakeDrop[RandomSample@#, 1000] &, benchmarkData];
    (* Each classifier is now
     trained and tested only using its specific features *)
    classifiersData = Table[Map[#[[features]] &, trainTestSplit, {3}],
      {features, classifiersFeatures}];
    classifiers = TrainClassifiers[classifiersData, classifierTypes, 1000, 800];
    AlgebraicallyEvaluateClassifiers[classifiers, classifiersData], {i, 100}];, i]
```

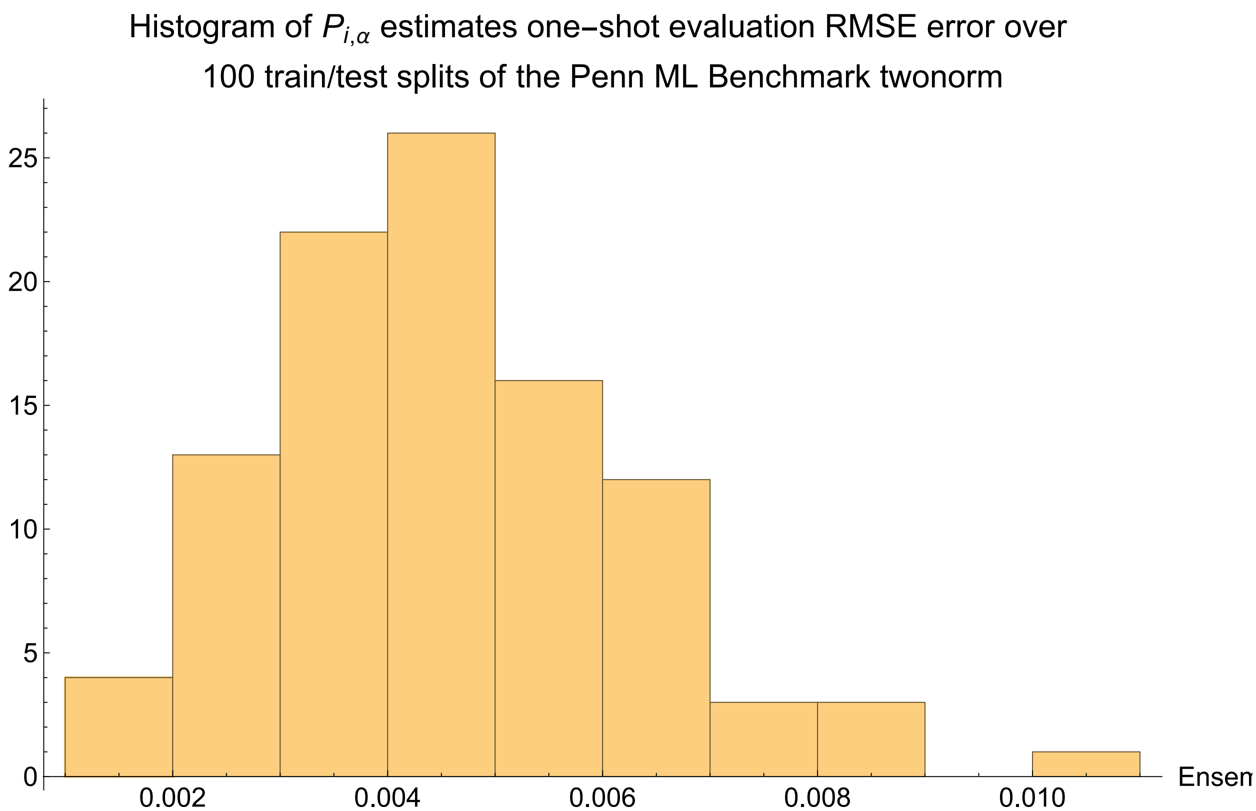
In[109]:=
```
runs // Take[#, 3] & /@ # & // Transpose // First //
  Histogram[#,
   PlotLabel → "Histogram of Pα estimate error over 100 train/test splits\nfor
      the Penn ML Benchmark twonorm",
   AxesLabel → {"α prevalence error", None},
   BaseStyle → {FontSize → 14}, ImageSize → 800] &
```

Out[109]=



In[111]:=
```
runs // Take[#, 3] & /@ # & // Transpose // #[[2]] & //
 Histogram[#,
   PlotLabel → "Histogram of P_{i,α} estimates one-shot evaluation RMSE error
     over \n100 train/test splits of the Penn ML Benchmark twonorm",
   AxesLabel → {"Ensemble α accuracies RMSE", None},
   BaseStyle → {FontSize → 14}, ImageSize → 800] &
```

Out[111]=



In[112]:=
```
runs // Transpose // #[[3]] & //
 Histogram[#,
   PlotLabel → "Histogram of P_{i,β} estimates one-shot evaluation RMSE error
     over \n100 train/test splits of the Penn ML Benchmark twonorm",
   AxesLabel → {"Ensemble β accuracies RMSE", None},
   BaseStyle → {FontSize → 14}, ImageSize → 800] &
```

Out[112]=