
Code

Data

```
In[*]:= Clear[ImportPennMLBenchmarksDataset]
ImportPennMLBenchmarksDataset[datasetName_String] :=
Module[{filename, tsvHeader, benchmarkData}, 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}] &;
{tsvHeader, benchmarkData}]
```

Mathematica's Classify function is sometimes confused on what data types to use for feature values. This list specifies the type.

```
In[*]:= uciAdultFeatures =
  {{1, "Numerical"}, {2, "Nominal"}, {3, "Numerical"}, {4, "Nominal"}, {5, "Nominal"},
   {6, "Nominal"}, {7, "Nominal"}, {8, "Nominal"}, {9, "Nominal"}, {10, "Nominal"},
   {11, "Numerical"}, {12, "Numerical"}, {13, "Numerical"}, {14, "Nominal"}}

Out[*]:= {{1, Numerical}, {2, Nominal}, {3, Numerical}, {4, Nominal}, {5, Nominal},
  {6, Nominal}, {7, Nominal}, {8, Nominal}, {9, Nominal}, {10, Nominal},
  {11, Numerical}, {12, Numerical}, {13, Numerical}, {14, Nominal}}

In[*]:= mushroomFeatures = {{1, "Nominal"}, {2, "Nominal"}, {3, "Nominal"}, {4, "Nominal"},
  {5, "Nominal"}, {6, "Nominal"}, {7, "Nominal"}, {8, "Nominal"}, {9, "Nominal"},
  {10, "Nominal"}, {11, "Nominal"}, {12, "Nominal"}, {13, "Nominal"}, {14, "Nominal"},
  {15, "Nominal"}, {16, "Nominal"}, {17, "Nominal"}, {18, "Nominal"},
  {19, "Nominal"}, {20, "Nominal"}, {21, "Nominal"}, {22, "Nominal"}}

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

Data sketch

True or Ground Truth Evaluation Point

```

In[*]:= Clear[VotingFrequenciesData]
VotingFrequenciesData[testAlignedDecisions_Association,
  classifiers: {_Integer ..}] := Module[
  {eventCounts},
  eventCounts =
    Values@testAlignedDecisions // Merge[#, Identity] & // Map[Total, #] & //
      KeyValueMap[(#1[[classifiers]] → #2) &, #] & //
      GroupBy[#, First] & //
      Map[Last, #, {2}] & //
      Map[Total, #] & //
      KeyMap[Subscript[f, Sequence@@Map[If[# == 0,  $\alpha$ ,  $\beta$ ] &, #]] &, #] & //
      # / Total@# &]

In[*]:= Clear[TurnVotesToIndicators]
TurnVotesToIndicators[key_, label_] := Map[If[# == label, 1, 0] &, key]

Clear[ClassifierLabelAccuracy]
ClassifierLabelAccuracy[
  voteCountsByLabel_Association, classifier_Integer, label_] :=
  voteCountsByLabel[label] // KeyMap[TurnVotesToIndicators[#, label] &, #] & //
    Normal // GroupBy[#, #[[1, classifier]] &] & //
    Map[Last, #, {2}] & // Map[Total, #] & //
    #[1] / (Total@#) &

In[*]:= Clear[ProjectVoteCounts]
ProjectVoteCounts[labelVoteCounts_, classifiers_List] :=
  Normal[labelVoteCounts] // GroupBy[#, #[[1, classifiers]] &] & //
    Map[Last, #, {2}] & // Map[Total, #] &

```

```
In[*]:=
```

```
Clear[CorrelationProduct]
CorrelationProduct[indicators_List, accuracies_List] :=
  Times@@ (indicators - accuracies)

Clear[LabelCorrelations]
LabelCorrelations[voteCountsByLabel_Association,
  classifiers_List, label_] := Module[
  {labelAccuracies},
  labelAccuracies =
    Map[ClassifierLabelAccuracy[voteCountsByLabel, #, label] &, classifiers];
  voteCountsByLabel[label] // ProjectVoteCounts[#, classifiers] & //
    KeyMap[TurnVotesToIndicators[#, label] &, #] & //
    KeyMap[CorrelationProduct[#, labelAccuracies] &, #] & // Normal //
  ((Map[Times@@# &, #] // Total) / (Map[Last, #] // Total)) &
```

```
In[*]:=
```

```
Clear[GTClassifiers]
GTClassifiers[votingPatternCountsByLabel_Association] := Module[
  {alphaLabel = 0, nClassifiers},
  nClassifiers =
    votingPatternCountsByLabel // First // Keys // RandomChoice // Length;
  Join[{P $\alpha$  → (votingPatternCountsByLabel // Map[Values, #] & // Map[Total, #] & //
    #[alphaLabel] / Total@# &)},
    Table[P $i,\alpha$  → ClassifierLabelAccuracy[votingPatternCountsByLabel, i, 0],
      {i, nClassifiers}],
    Table[P $i,\beta$  → ClassifierLabelAccuracy[votingPatternCountsByLabel, i, 1],
      {i, nClassifiers}],
    Table[ΓSequence@@pair, $\alpha$  → LabelCorrelations[votingPatternCountsByLabel, pair, 0],
      {pair, Subsets[Range@nClassifiers, {2}]}],
    Table[ΓSequence@@pair, $\beta$  → LabelCorrelations[votingPatternCountsByLabel, pair, 1],
      {pair, Subsets[Range@nClassifiers, {2}]}],
    Table[ΓSequence@@trio, $\alpha$  → LabelCorrelations[votingPatternCountsByLabel, trio, 0],
      {trio, Subsets[Range@nClassifiers, {3}]}],
    Table[ΓSequence@@trio, $\beta$  → LabelCorrelations[votingPatternCountsByLabel, trio, 1],
      {trio, Subsets[Range@nClassifiers, {3}]}]] // Association
]
```

Algebra

```

In[*]:= Clear[IndependentGeneratingSet]
IndependentGeneratingSet[{i_, j_, k_}] :=
{Pα Pi,α Pj,α Pk,α + (1 - Pα) (1 - Pi,β) (1 - Pj,β) (1 - Pk,β) - fα,α,α,
Pα Pi,α Pj,α (1 - Pk,α) + (1 - Pα) (1 - Pi,β) (1 - Pj,β) Pk,β - fα,α,β,
Pα Pi,α (1 - Pj,α) Pk,α + (1 - Pα) (1 - Pi,β) Pj,β (1 - Pk,β) - fα,β,α,
Pα Pi,α (1 - Pj,α) (1 - Pk,α) + (1 - Pα) (1 - Pi,β) Pj,β Pk,β - fα,β,β,
Pα (1 - Pi,α) Pj,α Pk,α + (1 - Pα) Pi,β (1 - Pj,β) (1 - Pk,β) - fβ,α,α,
Pα (1 - Pi,α) Pj,α (1 - Pk,α) + (1 - Pα) Pi,β (1 - Pj,β) Pk,β - fβ,α,β,
Pα (1 - Pi,α) (1 - Pj,α) Pk,α + (1 - Pα) Pi,β Pj,β (1 - Pk,β) - fβ,β,α,
Pα (1 - Pi,α) (1 - Pj,α) (1 - Pk,α) + (1 - Pα) Pi,β Pj,β Pk,β - fβ,β,β}

In[*]:= Clear[AlgebraicallyEvaluateClassifiers]
AlgebraicallyEvaluateClassifiers[vcbl_] := Module[
{ evaluationIdeal,
equationsToSolve, vars, sols, gt, observableFrequencies},
(* Calculate the size of the test sets *)
observableFrequencies = VotingFrequenciesData[vcbl, {1, 2, 3}];
evaluationIdeal = IndependentGeneratingSet[{1, 2, 3}];
equationsToSolve = Map[(# == 0) &, evaluationIdeal] /. observableFrequencies;
vars = Variables /@ evaluationIdeal // Flatten // DeleteDuplicates //
SortBy[#, {Last@#} &] &;
Solve[equationsToSolve, vars]]

In[*]:= Clear[FirstSolutionPoint]
FirstSolutionPoint[eval_List] := Last@eval // First // First // Association
Clear[SecondSolutionPoint]
SecondSolutionPoint[eval_List] := Last@eval // First // Last // Association

In[*]:= Clear[GroundTruthCoordinates]
GroundTruthCoordinates[point_] := {Pα, P1,α, P1,β, P2,α, P2,β, P3,α, P3,β} /. point

In[*]:= Clear[IsNotComplexQ]
IsNotComplexQ[eval_List] :=
eval // Last // Flatten // Last /@ # & // Not@MemberQ[#, _Complex, Infinity] &

In[*]:= Clear[MetricVsDistance]
MetricVsDistance[featuresRunEval_List, metric_] := featuresRunEval //
{(* we want the ground truth evaluation *)
First@First@# //
metric,
Last@#} &

```

Algebraic evaluation failure mode tests

```

In[*]:= (* The independent evaluation ideal may be empty *)
Clear[HasAllDecisionFrequenciesQ]
HasAllDecisionFrequenciesQ[voteCountsByLabel_] := Module[
  {observableFrequencies, evaluationGS, empiricalSet},
  observableFrequencies = VotingFrequenciesData[voteCountsByLabel, {1, 2, 3}];
  observableFrequencies // If[Length@# == 8, True, False] &]

In[*]:= (* The independent evaluation ideal may be empty *)
Clear[IndependentEvaluationIdealIsNonEmptyQ]
IndependentEvaluationIdealIsNonEmptyQ[voteCountsByLabel_] := Module[
  {observableFrequencies,
   evaluationGS,
   observedGS,
   vars,
   gb},

  (* Set-up the generating set from the decisions data sketch *)
  observableFrequencies = VotingFrequenciesData[voteCountsByLabel, {1, 2, 3}];
  evaluationGS = IndependentGeneratingSet[{1, 2, 3}];
  observedGS = evaluationGS /. observableFrequencies;
  vars = Variables@observedGS // DeleteDuplicates // SortBy[#, Last] &;
  (* Does this generating set have a Groebner basis *)
  gb = GroebnerBasis[observedGS, Reverse@vars, CoefficientDomain -> Rationals];
  If[gb == {1}, False, True]]

In[*]:= (* The independent estimates may contain complex numbers *)
Clear[HasOnlyRealNumbersQ]
HasOnlyRealNumbersQ[sols_List] := First@sols //
  Last /@ # & //
  FreeQ[#, Complex, Heads -> True] &

In[*]:= (* The independent estimates may be outside the unit cube *)
Clear[InsideUnitCubeQ]
InsideUnitCubeQ[sols_List] := First@sols //
  Last /@ # & //
  Map[(# > 0 && # < 1) &, #] & //
  Counts //
  If[Lookup[#, False, 0] > 0, False, True] &

```

Universal Surface in Accuracies+Prevalence Space

```
In[*]:= Clear[GroundTruthImplicitRegion]
GroundTruthImplicitRegion[
  voteCountsByLabel_Association, classifiers: {__Integer}, label_] :=
Module[{eqs, vars, prevalence, prevalenceTerms},
  eqs = Table[{ $P_{i,\alpha} - f_\alpha$ ,  $P_{i,\beta} - f_\beta$ } /.
    VotingFrequenciesData[voteCountsByLabel, {i}], {i, classifiers}];
  vars = Flatten@eqs // Variables /@ # & // Flatten // DeleteDuplicates // Sort;
  prevalence = If[label ==  $\alpha$ ,  $P_\alpha$ ,  $P_\beta$ ];
  prevalenceTerms =
    If[label ==  $\alpha$ , {prevalence, 1 - prevalence}, {1 - prevalence, prevalence}];
  ImplicitRegion[
    Join[
      (* The single classifier equations *)
      Map[Transpose@{prevalenceTerms, #} &, eqs] //
        Map[Times@@# &, #, {2}] & //
        Map[(First@# - Last@# == 0) &, #] &,
      (* The pair equations *)
      Table[
        eqs[[pair]] // {First@#, Reverse@Last@#} & // Transpose // Times@@# & /@# & //
          (First@# - Last@# == 0) &,
        {pair, Subsets[classifiers, {2}]]}],
    Evaluate@Table[{var, 0, 1}, {var, Join[{prevalence}, vars]}]]]
```

Experiments code

```
In[*]:= Clear[DistancesToCV]
DistancesToCV[partitionRuns_] := Last@partitionRuns //
  {GroundTruthImplicitRegion[#, {1, 2, 3},  $\alpha$ ],
    AlgebraicallyEvaluateClassifiers@# // First // N //
    GroundTruthCoordinates} & /@ # & //
  Map[RegionDistance[First@#, Last@#] &, #] &

In[*]:= Clear[TrueEvaluations2Way]
TrueEvaluations2Way[partitionRuns_] := Map[GTClassifiers, Last@partitionRuns] //
  Map[KeySelect[#, (First@# == 1 && Length@# == 4) &] &, #] & //
  Values /@ # & // Map[SortBy[#, Abs] &, #] & // Map[Reverse, #] & // First /@ # &

Clear[VCBLs]
VCBLs[partitionRuns_] := Last@partitionRuns
```

Training ensembles

```

In[*]:= Clear[MakeTrainTestSplit]
MakeTrainTestSplit[data_Association,
  aTrainIndices : {_Integer ..}, bTrainIndices : {_Integer ..}] :=
Module[{aLength, bLength},
  aLength = Length@data[0];
  bLength = Length@data[1];
  Association[
    0 → (data[0] //
      {#[[aTrainIndices]], #[[Complement[Range@aLength, aTrainIndices]]]} &),
    1 → (data[1] //
      {#[[bTrainIndices]], #[[Complement[Range@bLength, bTrainIndices]]]} &)]

In[*]:= Clear[RandomSampleTrainTestSplit]
RandomSampleTrainTestSplit[data_, nTestAlpha_Integer, nTrain_Integer] :=
Association[
  0 → {
    RandomSample[First@data[0]] // Take[#, nTrain] &,
    RandomSample[Last@data[0]] // Take[#, nTestAlpha] &},
  1 → {
    RandomSample[First@data[1]] // Take[#, nTrain] &,
    RandomSample[Last@data[1]] // Take[#, nTestAlpha // Floor] &}]

In[*]:= Clear[ClassifiersSampleIndices]
ClassifiersSampleIndices[data_, nSample_Integer] :=
  RandomSample[Range@Length@data] //
  Partition[#, nSample] & // Take[#, 3] &
Clear[SampleIndices]
SampleIndices[data_, nSample_Integer] := RandomSample[Range@Length@data, nSample]

In[*]:= Clear[SelectTrainDataFromSplit]
SelectTrainDataFromSplit[data_] := Map[First, data]
Clear[SelectTestDataFromSplit]
SelectTestDataFromSplit[data_] := Map[Last, data]

In[*]:= Clear[SelectFromTrainTestSplit]
SelectFromTrainTestSplit[data_, nTestAlpha_Integer, nTrain_Integer] :=
Association[
  0 → {
    RandomSample[First@data[0]] // Take[#, nTrain] &,
    RandomSample[Last@data[0]] // Take[#, nTestAlpha] &},
  1 → {
    RandomSample[First@data[1]] // Take[#, nTrain] &,
    RandomSample[Last@data[1]] // Take[#, nTestAlpha // Floor] &}]

```

```

In[*]:= Clear[SelectFromData]
SelectFromData[data_, aIndices : {_Integer..}, bIndices : {_Integer..}] :=
  KeySort@data // MapAt[#[[aIndices]] &, #, 1] & // MapAt[#[[bIndices]] &, #, 2] &

In[*]:= (* A helper function for splitting training data among the classifiers *)
Clear[FilterTraining]
FilterTraining[data_, {alphaIndices_, betaIndices_}] :=
  Association[0 → data[0] [[alphaIndices]], 1 → data[1] [[betaIndices]]]

In[*]:= Clear[TrainClassifiersDisjoint]
TrainClassifiersDisjoint[classifiersData_, classifierTypes_List,
  trainIndices_List, featureTypes_List, performanceGoal_String] := Module[
  { trainingSamples, classifiers},
  classifiers = Transpose@{
    classifierTypes,
    Transpose@{classifiersData, trainIndices} // Map[FilterTraining@@# &, #] &,
    featureTypes} // Map[
    Classify[#[[2]],
      Method → #[[1]],
      TrainingProgressReporting → None,
      PerformanceGoal → performanceGoal,
      FeatureTypes → #[[3]]] &,
    #] &;
  classifiers]

```


Classification

```

In[*]:= Clear[LabelCounts]
LabelCounts[classifiers_, classifiersData_] := Module[
  {nTestAlpha, nTestBeta, decisions,
   byLabelDecisions, votingPatternCountsByLabel,
   sols, equationsToSolve, vars,
   gt, alphaLabel = 0, betaLabel = 1},
  (* Calculate the size of the test sets *)
  {nTestAlpha, nTestBeta} =
    classifiersData // First // Map[Length, #] & // {#[alphaLabel], #[betaLabel]} &;
  (* We arbitrarily define "0" as the alpha label, and "1" as the beta label *)
  decisions =
    Table[classifiers[[i]][classifiersData[[i]] // Join[#[alphaLabel], #[betaLabel]] &],
      {i, Length@classifiers}];
  byLabelDecisions = decisions // Map[TakeDrop[#, nTestAlpha] &, #] & //
    Map[Association[{alphaLabel → #[[1]], betaLabel → #[[2]]}] &, #] &;
  votingPatternCountsByLabel = byLabelDecisions // Merge[#, Identity] & //
    Map[Transpose, #] & // Map[Counts, #] &;
  votingPatternCountsByLabel
]

```