A List of 68 Metrics Calculated by WEKA

http://weka.sourceforge.net/doc.dev/weka/classifiers/Evaluation.html

By: Noureddin Sadawi

31 Jan 2014

1. **Basic performance stats - right vs wrong**

   (a) **unclassified()** Gets the number of instances not classified (that is, for which no prediction was made by the classifier).

   (b) **correct()** Gets the number of instances correctly classified (that is, for which a correct prediction was made).

   (c) **pctCorrect()** Gets the percentage of instances correctly classified (that is, for which a correct prediction was made).

   (d) **pctIncorrect()** Gets the percentage of instances incorrectly classified (that is, for which an incorrect prediction was made).

   (e) **pctUnclassified()** Gets the percentage of instances not classified (that is, for which no prediction was made by the classifier).

   (f) **incorrect()** Gets the number of instances incorrectly classified (that is, for which an incorrect prediction was made).

   (g) **kappa()** Returns value of kappa statistic if class is nominal.

2. **IR stats**

   (a) **areaUnderROC(int classIndex)** Returns the area under ROC for those predictions that have been collected in the evaluateClassifier(Classifier, Instances) method.

   (b) **falseNegativeRate(int classIndex)** Calculate the false negative rate with respect to a particular class.

   (c) **falsePositiveRate(int classIndex)** Calculate the false positive rate with respect to a particular class.

   (d) **fMeasure(int classIndex)** Calculate the F-Measure with respect to a particular class.

   (e) **numTrueNegatives(int classIndex)** Calculate the number of true negatives with respect to a particular class.

(f) **numTruePositives(int classIndex)** Calculate the number of true positives with respect to a particular class.

(g) **numFalseNegatives(int classIndex)** Calculate number of false negatives with respect to a particular class.

(h) **numFalsePositives(int classIndex)** Calculate number of false positives with respect to a particular class.

(i) **precision(int classIndex)** Calculate the precision with respect to a particular class.

(j) **recall(int classIndex)** Calculate the recall with respect to a particular class.

(k) **trueNegativeRate(int classIndex)** Calculate the true negative rate with respect to a particular class.

(l) **truePositiveRate(int classIndex)** Calculate the true positive rate with respect to a particular class.

3. **Weighted IR stats**

(a) **weightedAreaUnderROC()** Calculates the weighted (by class size) AUC.

(b) **weightedFalseNegativeRate()** Calculates the weighted (by class size) false negative rate.

(c) **weightedFalsePositiveRate()** Calculates the weighted (by class size) false positive rate.

(d) **weightedFMeasure()** Calculates the macro weighted (by class size) average F-Measure.

(e) **weightedPrecision()** Calculates the weighted (by class size) precision.

(f) **weightedRecall()** Calculates the weighted (by class size) recall.

(g) **weightedTrueNegativeRate()** Calculates the weighted (by class size) true negative rate.

(h) **weightedTruePositiveRate()** Calculates the weighted (by class size) true positive rate.

4. **SF stats**

(a) **SFEntropyGain()** Returns the total SF, which is the null model entropy minus the scheme entropy.

(b) **SFMeanEntropyGain()** Returns the SF per instance, which is the null model entropy minus the scheme entropy, per instance.

   (c) **SFMeanPriorEntropy()** Returns the entropy per instance for the null model.
   (d) **SFMeanSchemeEntropy()** Returns the entropy per instance for the scheme
   (e) **SFPriorEntropy()** Returns the total entropy for the null model.
   (f) **SFSchemeEntropy()** Returns the total entropy for the scheme.

5. **Sensitive stats - certainty of predictions**
   (a) **relativeAbsoluteError()** Returns the relative absolute error.
   (b) **rootMeanSquaredError()** Returns the root mean squared error.
   (c) **rootRelativeSquaredError()** Returns the root relative squared error if the class is numeric.
   (d) **meanAbsoluteError()** Returns the mean absolute error.

6. **K&B stats**
   (a) **KBInformation()** Return the total Kononenko & Bratko Information score in bits.
   (b) **KBMeanInformation()** Return the Kononenko & Bratko Information score in bits per instance.
   (c) **KBRelativeInformation()** Return the Kononenko & Bratko Relative Information score.

7. **areaUnderPRC(int classIndex)** Returns the area under precision-recall curve (AUPRC) for those predictions that have been collected in the evaluateClassifier(Classifier, Instances) method.

8. **avgCost()** Gets the average cost, that is, total cost of misclassifications (incorrect plus unclassified) over the total number of instances.

9. **confusionMatrix()** Returns a copy of the confusion matrix.

10. **correlationCoefficient()** Returns the correlation coefficient if the class is numeric.

11. **coverageOfTestCasesByPredictedRegions()** Gets the coverage of the test cases by the predicted regions at the confidence level specified when evaluation was performed.

12. **errorRate()** Returns the estimated error rate or the root mean squared error (if the class is numeric).

13. **getClassPriors()** Get the current weighted class counts.

14. **matthewsCorrelationCoefficient(int classIndex)** Calculates the matthews correlation coefficient (sometimes called phi coefficient) for the supplied class

15. **meanPriorAbsoluteError()** Returns the mean absolute error of the prior.

16. **numInstances()** Gets the number of test instances that had a known class value (actually the sum of the weights of test instances with known class values

17. **priorEntropy()** Calculate the entropy of the prior distribution.

18. **rootMeanPriorSquaredError()** Returns the root mean prior squared error.

19. **setMetricsToDisplay(java.util.List¡java.lang.String¿ display)** Set a list of the names of metrics to have appear in the output.

20. **sizeOfPredictedRegions()** Gets the average size of the predicted regions, relative to the range of the target in the training data, at the confidence level specified when evaluation was performed

21. **totalCost()** Gets the total cost, that is, the cost of each prediction times the weight of the instance, summed over all instances.

22. **unweightedMacroFmeasure()** Unweighted macro-averaged F-measure.

23. **unweightedMicroFmeasure()** Unweighted micro-averaged F-measure.

24. **weightedAreaUnderPRC()** Calculates the weighted (by class size) AUPRC.

25. **weightedMatthewsCorrelation()** Calculates the weighted (by class size) matthews correlation coefficient.

26. Number_of_training_instances

27. Number_of_testing_instances

28. Elapsed_Time_training

29. Elapsed_Time_testing

30. UserCPU_Time_training

31. UserCPU_Time_testing

32. Serialized_Model_Size

33. Serialized_Train_Set_Size
34. Serialized_Test_Set_Size