Area Under the Receiver Operating Characteristics Curve

This evaluation metric is a single score that is a measure of how well the model is able to distinguish between classes.

The Receiver Operating Characteristics curve

This curve is a plot of the True Positive Rate versus the False Positive Rate as the threshold value is increased. The threshold value is a value between zero and one that specifies the probability above which an example is classified as positive.

Area under the curve as a probability

The area under the ROC curve can be used as a measure of how well the model classifies because it represents the probability that a random positive example is ranked higher than a random negative example.

This feature will be used to calculate the value of the metric.

Design of the implementation

Item	Values
Parameters	DATASET(Classify_Result) predicted, DATASET(DiscreteField) actual
Returned values	{INTEGER wi, INTEGER classifier, REAL8 value} AUC

Proposed location

ML_Core.Analysis.Classification

Parameters

DATASET(Classify_result) predicted - The predictions as received by the model, that include the probability of the prediction.

DATASET(DiscreteField) actual - The actual or real values of the fields the model tried to predict

Return value

{INTEGER wi, INTEGER classifier, REAL8 value} AUC - The area under the curve, per classifier, per work item.

Dependencies

ML_Core.Types

Implementation

Since AUC is the probability that a random positive example is ranked higher than a random negative example, one way to calculate it would be to create a set of all positive-negative pairs (per work item, per classifier) and determine what fraction of them are *good* i.e, in how many of them the positive example is marked higher than the negative example.

First, the raw predictions and their true classifications are combined for convenience.

```
combined := JOIN(predicted_raw, actual,
                 LEFT.wi = RIGHT.wi and LEFT.number = RIGHT.number and LEFT.id =
RIGHT.id);
```

Next, the combined data is JOINed with itself to produce pairs. To eliminate duplicates and pairs of the same item, only positive examples are taken on the left, and only negative examples are taken on the right. This way, a pair *isGood* if the left's value is greater than the right's value. Since the join places a restriction on 'wi' and 'number', these pairs are formed only within the same work units *and* the same regressors.

```
pairs := JOIN(combined, combined,
              LEFT.wi = RIGHT.wi and LEFT.number = RIGHT.number and LEFT.raw = 1 and
RIGHT.raw = 0,
              TRANSFORM({INTEGER wi, INTEGER classifier, BOOLEAN isGood},
                        SELF.wi = LEFT.wi,
                        SELF.classifier = LEFT.classifier,
                        SELF.isGood = IF(LEFT.value >= RIGHT.value,true,false)));
```

Now these values are counted per work unit per classifier, grouped using the table functionality.

```
auc := TABLE(pairs,
             {wi, classifier, value := COUNT(GROUP,isGood=true) / COUNT(GROUP)},
             wi, classifier);
```

F-score

The F-score in statistical analysis of binary classification is a measure of accuracy. The traditional F-measure or balanced F-score (F1 score) is the harmonic mean of precision and recall:

$$F_1 = 2 \cdot \frac{precision \cdot recall}{precision + recall}$$

While precision is the fraction of True positives among all predicted as positive, recall is the fraction of True positives among all the points that are truly (actually) positive.

Calculation

The F-score, or more specifically the precision and recall, can be obtained from the confusion matrix.

The precision is the ratio of the diagonal element to the row (column) sum, and

The recall is the ratio of the diagonal element to the column (row) sum depending on the orientation of the matrix.

Design of implementation

Item	Values
Parameters	Parameters used by the AccuracyByClass function
Return value	Fscore is added to the result of Accuracy by class

Proposed Location

ML_Core.Analysis.Classification, where it will be combined with the **AccuracyByClass** function to display the f-score as one of the results.

Dependencies

ML_Core.Types

ML_Core

Implementation

The precision and recall values may be obtained from the ML_Core.Analysis.Classification.AccuracyByClass function.

The f-score may be obtained as a simple projection of the result of AccuracyByClass

```
Class_f_score := RECORD(Class_Accuracy)
  REAL8 fscore;
END;

Class_f_score fScoreTransform(DATASET(Class_Accuracy) abc) := TRANSFORM
  SELF.fscore := 2*(abc.precision*abc.recall)/(abc.precision + abc.recall);
  SELF := LEFT;
END;

f_Score := PROJECT(accuracyByClass, fScoreTransform(LEFT));
```

Hamming Loss

Hamming loss is a simple measure that is used to conveniently evaluate multi-label classification models. It is given by the fraction of labels that are incorrectly classified, among all the labels.

Design of implementation

Item	Values
Parameters	Parameters used by the ML_Core.Analysis.Classification.Accuracy function
Returns	Adds field **hl** to the result of Accuracy

Proposed location

ML_Core.Analysis.Classification, where the value will be added to the **Accuracy** function as one of the results

Dependencies

ML_Core

Implementation

To calculate the Hamming loss, the predicted and actual values are combined, where the *isCorrect* value is set to true if label is classified correctly. This data is then grouped by wi and classifier to get the hamming loss as the number of labels predicted wrong.

```
combined := JOIN(predicted, actual,
                 LEFT.wi=RIGHT.wi and LEFT.id=RIGHT.id and LEFT.number=RIGHT.number,
                 TRANSFORM({INTEGER wi, INTEGER classifier, BOOLEAN isCorrect},
                           SELF.wi := LEFT.wi,
                           SELF.classifier := LEFT.number,
                           SELF.isCorrect := IF(LEFT.value=RIGHT.value,true,false)));

h := TABLE(combined, {wi,classifier,hl:=COUNT(GROUP,isCorrect=false)}, wi,
classifier);
```

Silhouette coefficient

Silhouette analysis is a way to measure how close each point in a cluster is to the points in its neighboring clusters as well as in its own cluster. It provides an easy way to find the optimum value for k during k-means clustering.

Silhouette values lie in the range of (-1, 1). A value of +1 indicates that the sample point is far away from its neighboring cluster and very close to the cluster it is assigned. Similarly, a value of -1 indicates that the point is closer to its neighboring cluster than to the cluster it is assigned.

Computation

The Silhouette score for each individual sample is calculated as follows. Let a be the average of the distances of the sample from all other samples in its assigned cluster. Let b be the average of the distances of the sample from all points assigned to the closest cluster to the sample, not assigned to it. Here, the closest cluster is the one that essentially produces the smallest b value. Once these values are computed, the coefficient is given by the formula:

$$S_i = \frac{b_i - a_i}{max(a_i, b_i)}$$

Design of implementation

Item	Values
Parameters	DATASET(Cluster_Types.KMeans_Model.Labels) labels, DATASET(Types.NumericField) samples
Returns	{UNSIGNED2 wi, UNSIGNED8 id, REAL8 s}

Parameters

TABLE({UNSIGNED2 wi, UNSIGNED8 id, UNSIGNED8 label}) labels - The labels (closest cluster) assigned to the training samples as returned by KMeans.Labels or KMeans.predict

DATASET(Types.NumericField) samples - The training samples used to train the model, as passed to KMeans.Fit

These parameters can also be any samples and their corresponding labelling, not necessarily the training samples.

Returns

{UNSIGNED2 wi, UNSIGNED8 id, REAL8 s} - The silhouette coefficient per work unit, per cluster.

Proposed location

NEW Sub-module ML_Core.Analysis.Clustering

Dependencies

KMeans.Cluster_Types

ML_Core

Implementation

The implementation involves three parts.

1. Calculating the 'a' value for all samples
2. Calculating the 'b' value for all samples
3. Calculating the silhouette coefficient

Preparation

For convenience, the samples and labels are JOINed to form a single dataset.

```
points := JOIN(samples, labels,LEFT.wi = RIGHT.wi and LEFT.id = RIGHT.id);
```

Part 1 : Calculating a value for all samples

To calculate the a value for all samples, all pairs of points within a cluster are formed, and in the process the square of the distance between their number values for each number (coordinate) is found.

```
a1 := JOIN(samples, samples,
           LEFT.wi=RIGHT.wi and
           LEFT.number=RIGHT.number and
           LEFT.id <> RIGHT.id and
           LEFT.label=RIGHT.label,
           TRANSFORM({INTEGER wi, INTEGER id1, INTEGER id2,
                      INTEGER number, INTEGER label, REAL8 sq_diff},
                     SELF.wi := LEFT.wi,
                     SELF.id1 := LEFT.id,
                     SELF.id2 := RIGHT.id,
                     SELF.number := LEFT.number,
                     SELF.label := LEFT.label,
                     SELF.sq_diff := POWER(LEFT.value-RIGHT.value,2)));
```

This dataset is then used to calculate the distances between all these points.

```
a2 := TABLE(a1, {wi,id1,id2,label,dist:=SQRT(SUM(GROUP,sq_diff))},wi,id1,id2,label);
```

These distances are then averaged to finally get the a values.

```
a3 := TABLE(a2, {wi, id:=id1, label,value:=AVE(GROUP,dist)}, wi,id1,label);
```

Part 2 : Calculating b value for all samples

To calculate the b value for all samples, all pairs of points which do not belong to the same cluster are formed, and in the process the square of the distance between their values for each number is found.

```
b1 := JOIN(samples, samples,
           LEFT.wi=RIGHT.wi and
           LEFT.number=RIGHT.number and
           LEFT.id <> RIGHT.id,
           LEFT.label <> RIGHT.label,
           TRANSFORM({INTEGER wi, INTEGER id1, INTEGER id2, INTEGER Llabel,
                      INTEGER number, INTEGER Rlabel, REAL8 sq_diff},
                     SELF.wi := LEFT.wi,
                     SELF.id1 := LEFT.id,
                     SELF.id2 := RIGHT.id,
                     SELF.number := LEFT.number,
                     SELF.Llabel := LEFT.label,
                     SELF.Rlabel := RIGHT.label,
                     SELF.sq_diff := POWER(LEFT.value-RIGHT.value,2)));
```

Now, the distances between these points is calculated.

```
b2 := TABLE(b1,
            {wi,id1,id2,Llabel,Rlabel,dist:=SQRT(SUM(GROUP,sq_diff))},
            wi,id1,id2,Llabel,Rlabel);
```

Now, the average distance of a sample, from all samples in each cluster is calculated.

```
b3 := TABLE(b2,
            {wi,id:=id1,Llabel,Rlabel,avgDist:=AVE(GROUP,dist)},
            wi,id1,Llabel,Rlabel);
```

The minimums among these values for every point are the required b values.

```
b4 := TABLE(b3,
            {wi,id,label:=Llabel,value:=MIN(GROUP,avgDist)},
            wi,id,Llabel);
```

Part 3 : Calculating the Silhouette coefficient

To calculate the coefficient, the a and b values are used to find them, first for all samples, and then for each cluster.

```
sampleCoeffs := JOIN(a3,b4,
                     LEFT.id=RIGHT.id and LEFT.wi=RIGHT.wi,
                     TRANSFORM({INTEGER wi, INTEGER id, INTEGER label, REAL8 s},
                               SELF.wi := LEFT.wi,
                               SELF.id := LEFT.id,
                               SELF.label := LEFT.label,
                               SELF.value := (RIGHT.value-
LEFT.value)/MAX(RIGHT.value,LEFT.value)));
```

These sample coefficients are averaged to form the coefficients for the cluster, which is the required result.

```
clusterCoeffs := TABLE(sampleCoeffs,{wi,label,s:=AVE(GROUP,value)},wi,label);
```

Chi squared test for feature selection

The Chi squared test for feature selection is statistical measure that helps establish the dependence of two categorical variables. In machine learning, it can be used to determine whether a classifier is dependent on a certain feature, and thus helps in feature selection. **This test can only be used when both variables are categorical**

Contingency table

The contingency table represents the number of samples present in the data for each combination of sample category and feature category. To start with the chi2 test, a contingency table of the data is first constructed.

Chi2 value and Null hypothesis testing

In the Chi2 test, we first assume the null hypothesis that the two variables are independent. We then proceed to construct a contingency table that supports this claim, based off of the sums of rows and columns of our existing contingency table. We then calculate the chi2 value as

$$\chi^2 = \sum \frac{(e_i - o_i)^2}{e_i}$$

For a large population, the probability distribution of this value for a random sampling (small subset) of the population is known to approximately follow a chi2 distribution, and hence the name of the coefficient. A large value of chi2 indicates that it is **less probable** that the sample is **not an outlier** and represents the population well.

Hence, when the null hypothesis that the sample is independent of a feature produces a large chi2 value, it indicates that there is a low probability that the null hypothesis is true and hence **null hypothesis is rejected** and the dependence of the two variables is established. A lower chi2 value indicates that it is more probable that the null hypothesis is true. It is not known for certain, however **the null hypothesis cannot be rejected** and the dependence of the variables cannot be established.

Level of significance and critical value

The level of significance used is the highest probability of correctness of a hypothesis that can be rejected. For example, at 5% level of significance, a null hypothesis that has 0-5% chance of being correct is rejected. The corresponding value of chi2 for a given level of significance is called the critical value, and can be found using the level of significance and the number of degrees of freedom of the data.

Design of implementation

The implementation consists of two separate functionalities. The first is to create a contingency table of the data, and the second to find the chi2 values.

Contingency table

Item	Values
Parameter	DATASET(DiscreteField) samples, DATASET(DiscreteField) features
Returns	TABLE({UNSIGNED2 wi, UNSIGNED4 fnumber, UNSIGNED4 snumber, INTEGER4 fclass, INTEGER4 sclass, UNSIGNED8 cnt})

Parameters

DATASET(DiscreteField) samples - The classifiers which are categorical (dependent data)

DATASET(DiscreteField) features - The features which are also categorical (independent data)

Returns

TABLE({UNSIGNED2 wi, UNSIGNED4 fnumber, UNSIGNED4 snumber, INTEGER4 fclass, INTEGER4 sclass, UNSIGNED8 cnt}) - The contingency tables per work item for each combination of feature (fnumber) and classifier (snumber).

Proposed Location

NEW Sub module ML_Core.Analysis.FeatureSelection

Implementation

To obtain the contingency tables, the samples and features are first combined into a single table, with every feature mapped to every classifier.

```
combined := JOIN(samples, features,
                 LEFT.wi=RIGHT.wi and LEFT.id=RIGHT.id,
                 TRANSFORM({INTEGER wi, INTEGER id, INTEGER fnumber,
                            INTEGER snumber, INTEGER fclass, INTEGER sclass},
                           SELF.wi := LEFT.wi,
                           SELF.id := LEFT.id,
                           SELF.fnumber := RIGHT.number,
                           SELF.snumber := LEFT.number,
                           SELF.fclass := RIGHT.value,
                           SELF.sclass := LEFT.value));
```

This combined data is then grouped using the table functionality to obtain the contingency tables.

```
ct := TABLE(combined, {wi,fnumber,snumber,fclass,sclass,cnt:=COUNT(GROUP)},
            wi,fnumber,snumber,fclass,sclass);
```

Chi2

Item	Values
Parameter	DATASET(DiscreteField) samples, DATASET(DiscreteField) features
Returns	TABLE({UNSIGNED2 wi, UNSIGNED4 fnumber, UNSIGNED4 snumber, REAL8 chi2, UNSIGNED8 dof})

Parameters

DATASET(DiscreteField) samples - The classifiers which are categorical (dependent data)

DATASET(DiscreteField) features - The features which are also categorical (independent data)

Returns

TABLE({UNSIGNED2 wi, UNSIGNED4 fnumber, UNSIGNED4 snumber, REAL8 chi2, UNSIGNED8 dof}) - The chi2 value and degrees of freedom per work item for each combination of feature (fnumber) and classifier (snumber).

Proposed Location

NEW Sub module ML_Core.Analysis.FeatureSelection

Implementation

To obtain the chi2 values, the expected contingency table of the data is formed based on the null hypothesis that the data is independent and the existing contingency table of the data (ct).

The expected value for a cell is as follows, where Eij is a cell of the expected contingency table and Oij is a cell of the observed contingency table.

$$E_{ij} = \frac{(\sum_i O_{ij}) \cdot (\sum_j O_{ij})}{\sum_i \sum_j O_{ij}}$$

To achieve this, the row and column (i.e, the feature and sample) sums, as well as the whole sum, are first calculated from the contingency table.

```
featureSums := TABLE(ct,
{wi,fnumber,snumber,fclass,cnt:=SUM(GROUP,cnt)},wi,fnumber,snumber,fclass);

sampleSums := TABLE(ct,
{wi,fnumber,snumber,sclass,cnt:=SUM(GROUP,cnt)},wi,fnumber,snumber,sclass);

allSum := TABLE(ct, {wi,fnumber,snumber,cnt:=SUM(GROUP,cnt)},wi,fnumber,snumber);
```

These are then used to find the expected contingency table ex, using two JOINs

```
ex1 := JOIN(featureSums, sampleSums,
            LEFT.wi=RIGHT.wi and LEFT.fnumber=RIGHT.fnumber and
LEFT.snumber=RIGHT.snumber,
            TRANSFORM({INTEGER wi, INTEGER fnumber, INTEGER snumber,
                       INTEGER fclass, INTEGER sclass, REAL8 value},
                      SELF.wi := LEFT.wi,
                      SELF.value := LEFT.cnt * RIGHT.cnt,
                      SELF.fnumber := LEFT.fnumber,
                      SELF.snumber := LEFT.snumber,
                      SELF.fclass := LEFT.fclass,
                      SELF.sclass := RIGHT.sclass));

ex2 := JOIN(ex1, allSum,
            LEFT.wi=RIGHT.wi and LEFT.fnumber=RIGHT.fnumber and
LEFT.snumber=RIGHT.snumber,
            TRANSFORM(RECORDOF(ex1),
                      SELF.value := LEFT.value/RIGHT.cnt,
                      SELF := LEFT));
```

The number of degrees of freedom are calculated by preforming groupings on the row and column sums. The number of degrees of freedom is calculated as the number of (rows -1)*(cols - 1).

```
dof1 := TABLE(featureSums, {wi,fnumber,snumber,d:=COUNT(GROUP)-1}, wi,
fnumber,snumber);
dof2 := TABLE(sampleSums, {wi,fnumber,snumber,d:=COUNT(GROUP)-1}, wi,
fnumber,snumber);
dof := JOIN(dof1,dof2,
            LEFT.wi=RIGHT.wi and
            LEFT.fnumber=RIGHT.fnumber and
            LEFT.snumber=RIGHT.snumber,
            TRANSFORM(RECORDOF(dof1),
                      SELF.d := LEFT.d*RIGHT.d,
                      SELF := LEFT));
```

The two contingency tables are now used to calculate the chi2 values, first for each cell of the table, and then grouped for each table.

Here, the **LEFT OUTER** JOIN flag is used. This is because the contingency table obtained (ct) does not contain entries for combinations where no samples are available (i.e, when cnt = 0). However, the expected contingency table contains entries for all combinations of sample and feature classes due to the nature of its construction. Hence the OUTER condition is used to produce entries for all combinations of sample and feature classes.

chi2_2 is the required coefficient value for each combination of feature and classifier, per work item.

This is then combined with the computed dof values to give the final result.

```
chi2_1 := JOIN(ex2, ct,
               LEFT.wi=RIGHT.wi and
               LEFT.fnumber=RIGHT.fnumber and
               LEFT.snumber=RIGHT.snumber and
               LEFT.fclass=RIGHT.fclass and
               LEFT.sclass=RIGHT.sclass,
               TRANSFORM(RECORDOF(ex2),
                         SELF.value := POWER(RIGHT.value-LEFT.value,2)/LEFT.value,
                         SELF := LEFT), LEFT OUTER);

chi2_2 := TABLE(chi2_1,
{wi,fnumber,snumber,x2:=SUM(GROUP,value)},wi,fnumber,snumber);

chi2 := JOIN(chi2_2, dof,
             LEFT.wi=RIGHT.wi and LEFT.fnumber=RIGHT.fnumber and
LEFT.snumber=RIGHT.snumber);
```

Adjusted Rand Index (ARI)

Rand Index

The Rand index or Rand measure in statistics and in data clustering, is a measure of the similarity between two data clusterings. The rand index is related to accuracy. The Rand index has a value between 0 and 1, with 0 indicating that the two data clusterings do not agree on any pair of points and 1 indicating that the data clusterings are exactly the same.

Adjusted Rand Index

The rand index is not corrected for chance, and hence is not able to produce smaller values or neutral values for randomly guessed clusters. This is taken into account in the adjusted Rand Index, which is corrected for chace. The adjusted Rand index can yield negative values if the index is less than the expected index, that is, if the clustering is worse than a random attempt.

The value of Adjusted rand index may be calculated using a contingency table as shown below:

$X \diagdown Y$	Y_1	Y_2	$\ldots$	Y_s	Sums
X_1	n_{11}	n_{12}	$\ldots$	n_{1s}	a_1
X_2	n_{21}	n_{22}	$\ldots$	n_{2s}	a_2
$\vdots$	$\vdots$	$\vdots$	$\ddots$	$\vdots$	$\vdots$
X_r	n_{r1}	n_{r2}	$\ldots$	n_{rs}	a_r
Sums	b_1	b_2	$\ldots$	b_s	

$$\overbrace{ARI}^{\text{Adjusted Index}} = \frac{\overbrace{\sum_{ij}\binom{n_{ij}}{2}}^{\text{Index}} - \overbrace{[\sum_i \binom{a_i}{2}\sum_j \binom{b_j}{2}]/\binom{n}{2}}^{\text{Expected Index}}}{\underbrace{\frac{1}{2}[\sum_i \binom{a_i}{2} + \sum_j \binom{b_j}{2}]}_{\text{Max Index}} - \underbrace{[\sum_i \binom{a_i}{2}\sum_j \binom{b_j}{2}]/\binom{n}{2}}_{\text{Expected Index}}}$$

**Images taken from Wikipedia*

Design of implementation

Item	Values

Item	Values
Parameters	DATASET(Cluster_Types.KMeans_Model.Labels) predicted, DATASET(Cluster_Types.KMeans_Model.Labels) true
Returns	TABLE({UNSIGNED2 wi, REAL8 ari})

Parameters

DATASET(Cluster_Types.KMeans_Model.Labels) predicted - Labels assigned by the model.**DATASET(Cluster_Types.KMeans_Model.Labels) true** - Actual labels or 'Ground Truth'

Returns

TABLE({UNSIGNED2 wi, REAL8 ari}) - The ARI per work unit

Proposed location

ML_Core.Analysis.Clustering

Dependencies

ML_Core.FeatureSelection (For the use of contingency table)

Implementation

First the contingency table of the given data is formed. In order to do this, the data is converted to the DiscreteField format and fed to the function proposed in the chi2 proposal.

```
conv1 := PROJECT(predicted, TRANSFORM(Types.DiscreteField,
                                      SELF.wi := LEFT.wi,
                                      SELF.number := 1,
                                      SELF.id := LEFT.id,
                                      SELF.value := LEFT.label));

conv2 := PROJECT(true, TRANSFORM(Types.DiscreteField,
                                 SELF.wi := LEFT.wi,
                                 SELF.number := 1,
                                 SELF.id := LEFT.id,
                                 SELF.value := LEFT.label));

ct := contingencyTable(conv1, conv2);
```

Next, the three sums required by ARI are calculated.

```
rowSumsC2 := TABLE(ct, {wi, fclass, c:=SUM(GROUP,cnt)*(SUM(GROUP,cnt)-1)/2}, wi,
fclass);
colSumsC2 := TABLE(ct, {wi, sclass, c:=SUM(GROUP,cnt)*(SUM(GROUP,cnt)-1)/2}, wi,
sclass);
allSumC2 := TABLE(ct, {wi, value:=SUM(GROUP,cnt)*(SUM(GROUP,cnt)-1)/2}, wi);

a := TABLE(rowSumsC2, {wi, value:=SUM(GROUP,c)}, wi);
b := TABLE(colSumsC2, {wi, value:=SUM(GROUP,c)}, wi);

ct1 := PROJECT(ct, TRANSFORM(REORDOF(ct),
                             SELF.cnt := LEFT.cnt*(LEFT.cnt-1)/2,
                             SELF := LEFT));

nij := TABLE(ct1(fclass=sclass), {wi, value:=SUM(GROUP,cnt)}, wi);
```

Using these three values, ARI is calculated.

ari1 is the required ARI per work item.

```
ari := JOIN([a,b,nij,allSumC2],
            LEFT.wi = RIGHT.wi,
            TRANSFORM({INTEGER wi, INTEGER a, INTEGER b,
                       INTEGER nij, INTEGER n, REAL8 value},
                      SELF.wi := LEFT.wi,
                      SELF.a := ROWS(LEFT)[1].value,
                      SELF.b := ROWS(LEFT)[2].value,
                      SELF.nij := ROWS(LEFT)[3].value,
                      SELF.n := ROWS(LEFT)[4].value,
                      SELF.value := (SELF.nij - SELF.a*SELF.b/SELF.n)/
                                    (0.5*(SELF.a + SELF.b) - SELF.a*SELF.b/SELF.n)));

ari1 := TABLE(ari,{wi,value});
```