Outline

- Introduction
  - Multilabel Setting
  - Applications & Datasets
- Theoretical Foundations
  - Probabilities in Multilabel
    - joint vs. marginal
  - Losses
    - Ranking
- Programming in MULAN
  - data loading
  - training and evaluation
  - implementation of new approach
- Algorithms
  - Transformation vs. Holistic
  - Transformational Approaches
    - BR, LP, Pairwise
  - Label Dependencies
    - Classifier Chains
  - Holistic Approaches
    - Overview
  - Large Number of Labels
    - Adaptations
    - HOMER
    - Label Space Transformation

MULAN

Framework for
- handling multilabel data
- training state-of-the-art multilabel classifiers
- evaluate multilabel classifiers
- do multilabel specific feature selection, cross validation, data transformations etc.
- implement new ideas and approaches
- ...

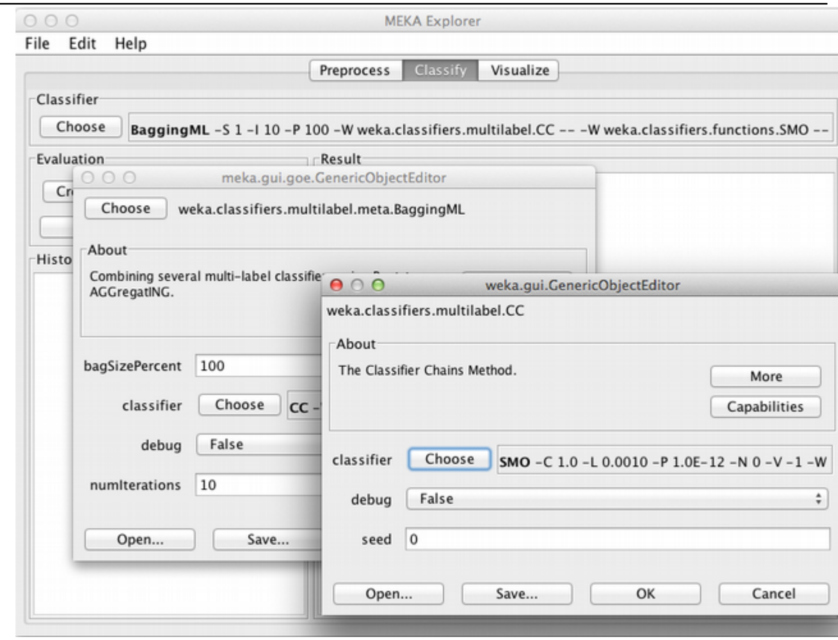
- built up on current WEKA version
  - usage of variety and abundance of learners and techniques available
- developed by team around Greg Tsoumakas, Univ. Thessaloniki
  - good mailing list available
  - stable code
  - http://mulan.sourceforge.net

MULAN Alternatives

TECHNISCHE UNIVERSITÄT DARMSTADT

MEKA

- multilabel extension to WEKA
- developed mainly by Jesse Read (Classifier Chains)
- only few algorithms, mostly ones developed by Author
  - but interface to MULAN
- GUI
- http://meka.sourceforge.net/



LPCforSOS

- developed by KE Group
- specific to pairwise decomposition
- alpha stage (contact us before using it)
- http://sourceforge.net/projects/lpcforsos/

Data Format

Based on WEKA ARFF file format

- labels are additional binary features
- accompanied by XML-file containing information to label features
  - also hierarchical information possible

```
@relation MultiLabelExample

@attribute feature1 numeric
@attribute feature2 numeric
@attribute feature3 numeric
@attribute label1 {0, 1}
@attribute label2 {0, 1}
@attribute label3 {0, 1}
@attribute label4 {0, 1}
@attribute label5 {0, 1}

@data
2.3,5.6,1.4,0,1,1,0,0
1.0,0.0,3.4,0,1,0,1,0
...
```

```
<?xml version="1.0"
      encoding="utf-8"?>
<labels xmlns="http://mulan.">
<label name="label1"></label>
<label name="label2"></label>
<label name="label3"></label>
<label name="label4"></label>
<label name="label5"></label>
</labels>
```

Simple Experiment
Data Loading

```
public class RunExperiment1readdata {

    public static void main(String[] args) throws Exception {

        String trainFile="data/emotions-train.arff";
        String testFile="data/emotions-test.arff";
        String xmlFile="data/emotions.xml";


        MultiLabelInstances trainInstances = new MultiLabelInstances(trainFile, xmlFile);
        MultiLabelInstances testInstances = new MultiLabelInstances(testFile, xmlFile);

        System.out.println("numLabels: "+trainInstances.getNumLabels());
        System.out.println("labelNames: "+Arrays.toString(trainInstances.getLabelNames()));
        System.out.println("numTrainInstances: "+trainInstances.getNumInstances());
        System.out.println("numTestInstances: "+testInstances.getNumInstances());
        System.out.println(testInstances.getDataSet());




        Instances empty = new Instances(testInstances.getDataSet(), 0);
        System.out.println(empty);
        System.out.println(testInstances.getDataSet().instance(1));

    }
}
```

Simple Experiment Training and Evaluation

```java
public static void main(String[] args) throws Exception {

    String trainFile="data/emotions-train.arff";
    String testFile="data/emotions-test.arff";
    String xmlFile="data/emotions.xml";


    MultiLabelInstances trainInstances = new MultiLabelInstances(trainFile, xmlFile);
    MultiLabelInstances testInstances = new MultiLabelInstances(testFile, xmlFile);


    //set binary base learner and multilabel learner
    J48 baseLearner = new J48();
    MultiLabelLearner multilabelLearner = new BinaryRelevance(baseLearner);

    //train
    multilabelLearner.build(trainInstances);
    System.out.println(multilabelLearner);

    //make a single prediction
    MultiLabelOutput prediction = multilabelLearner.makePrediction(testInstances.getDataSet().instance(1));
    System.out.println(prediction);

    //testing and evaluating on test data
    Evaluator evaluator = new Evaluator();
    Evaluation results = evaluator.evaluate(multilabelLearner, testInstances, trainInstances ); //second parameter only for statistics
    System.out.println(results);
```

Implementation Example: Dependent Binary Relevance

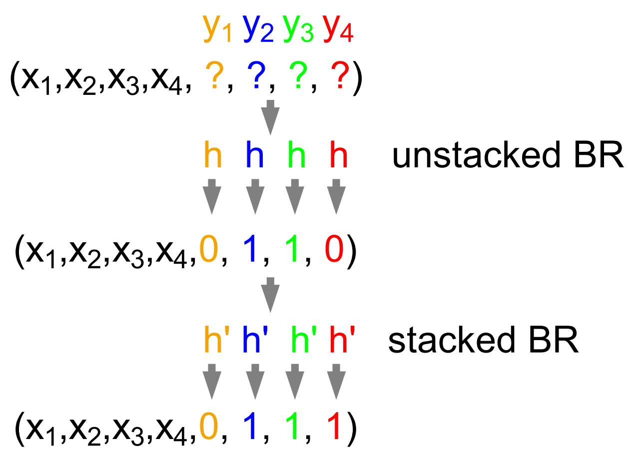
TECHNISCHE UNIVERSITÄT DARMSTADT

Training:

- learn binary single-label classifiers for each label, but include true label information as additional features (stacked BR)
  - like in CC, but include all labels!

Prediction:

  - first stage: fill up label features of test instances with predictions of unstacked BR
  - second stage: use stacked BR to predict labels
- in contrast to CC
  - can model dependencies in both directions
  - no label ordering, result is always the same



Montañesa, Senge, Barranquero, Quevedo, del Coz, Hüllermeier: *Dependent binary relevance models for multi-label classification.* In: Pattern Recognition, to appear
Eneldo Loza Mencía and Frederik Janssen, Towards Multilabel Rule Learning, in: Proceedings of KDML – LWA 2013

Dependent Binary Relevance: Example Result

One objective of DBR:

- find dependencies and model them explicitly
  - preferably direct, informative (symbolic) representation
  - compress knowledge / compressed view

approach	**yeast**	**enron**
binary relevance (using Ripper rule learner)	x23 > 0.08, x49 < -0.09 → **Class4** x68 < 0.05, x33 > 0.00, x24 > 0.00, x66 > 0.00, x88 > -0.06 → **Class4** x3 < -0.03, x71 > 0.03, x91 > -0.01 → **Class4** x68 < 0.03, x83 > -0.00, x44> 0.029, x93 < 0.01 → **Class4** x96 < -0.03, x10 > 0.01, x78< -0.07 → **Class4**	"mail", "fw", "didn" → **Joke**
dependent binary relevance	**Class3**, ~~**Class2**~~ → **Class4** **Class5**, ~~**Class6**~~ → **Class4** **Class3**, **Class1**, x22 > -0.02 → **Class4**	**Personal**, "day", "mail" → **Joke**

Dependent Binary Relevance: Implementation in Mulan

```java
public class DBR1 extends MultiLabelLearnerBase {

    @Override
    protected void buildInternal(MultiLabelInstances trainingSet)
            throws Exception {
        // TODO Auto-generated method stub

    }

    @Override
    protected MultiLabelOutput makePredictionInternal(Instance instance)
            throws Exception, InvalidDataException {
        // TODO Auto-generated method stub
        return null;
    }

    @Override
    public TechnicalInformation getTechnicalInformation() {
        // TODO Auto-generated method stub
        return null;
    }

}
```

Dependent Binary Relevance: Implementation in Mulan

```
public class DBR2 extends MultiLabelLearnerBase {

    Classifier baseLearner;
    Classifier[] stackedBR;

    BinaryRelevance unstackedBR;


    public DBR2(Classifier baseLearner) {
        this.baseLearner = baseLearner;
    }
```

Dependent Binary Relevance: Implementation in Mulan

```java
@Override
protected void buildInternal(MultiLabelInstances multilabelTrainingInstances)
        throws Exception {

    //for repeated access
    numLabels=multilabelTrainingInstances.getNumLabels();
    labelNames=multilabelTrainingInstances.getLabelNames();

    //build unstacked BR classifier for initialization of predictions
    unstackedBR=new BinaryRelevance(baseLearner);
    unstackedBR.build(multilabelTrainingInstances);


    //build stacked BR
    labelIndices=multilabelTrainingInstances.getLabelIndices();
    stackedBR=new Classifier[numLabels];
    for(int labelNo=0;labelNo<labelIndices.length;labelNo++){

        //take original weka instances object and just set the correct class index
        int labelAttributeIndex=labelIndices[labelNo];
        Instances subProblemInstances = multilabelTrainingInstances.getDataSet();
        subProblemInstances.setClassIndex(labelAttributeIndex);  //TODO: this is not safe!!

        Classifier currentBaseLearner = AbstractClassifier.makeCopy(baseLearner);
        System.out.println("Bulding model " + (labelNo + 1) + "/" + numLabels);
        currentBaseLearner.buildClassifier(subProblemInstances);
        stackedBR[labelNo]=currentBaseLearner;

    }


}
```

Dependent Binary Relevance: Implementation in Mulan

```java
@Override
protected MultiLabelOutput makePredictionInternal(Instance instance)
        throws Exception, InvalidDataException {

    //in order to not change the given instance object
    Instances dataset=new Instances(instance.dataset(),0);
    Instance toPredict=(Instance) instance.copy();
    toPredict.setDataset(dataset);

    //make predictions for unstacked BR
    MultiLabelOutput pred = unstackedBR.makePrediction(instance);
    boolean[] biPartitions = pred.getBipartition();
    //not necessary
    double[] confidenceValues = pred.getConfidences();

    //put BR predictions into test instance
    for(int labelNo=0;labelNo<numLabels;labelNo++){
        int labelAttributeIndex=labelIndices[labelNo];
        if(biPartitions[labelNo]==true)
            toPredict.setValue(labelAttributeIndex, 1); //set positive
        else
            toPredict.setValue(labelAttributeIndex, 0); //set negative
    }

    //do predictions for stacked BR
```

Dependent Binary Relevance: Implementation in Mulan

```
//put BR predictions into test instance
for(int labelNo=0;labelNo<numLabels;labelNo++){
    int labelAttributeIndex=labelIndices[labelNo];
    if(biPartitions[labelNo]==true)
        toPredict.setValue(labelAttributeIndex, 1); //set positive
    else
        toPredict.setValue(labelAttributeIndex, 0); //set negative
}

//do predictions for stacked BR
for(int labelNo=0;labelNo<numLabels;labelNo++){
        Classifier singleClassifier = stackedBR[labelNo];

        //set the correct label index
        int labelAttributeIndex=labelIndices[labelNo];
        dataset.setClassIndex(labelAttributeIndex);

        //use WEKA base learner predictions
        double[] distr = singleClassifier.distributionForInstance(toPredict);
        confidenceValues[labelNo]=distr[1]; //assume class 0 is negative and class 1 positive one
        if (confidenceValues[labelNo] > 0.5){
            biPartitions[labelNo]=true;
        }else{  //distr[1]<0.5
            biPartitions[labelNo]=false;
        }
    }

//produce output -> final prediction
return new MultiLabelOutput(biPartitions,confidenceValues);
```