# Maschinenlernen mit XML-Daten und Weka


+

# Buzzword Bingo

| | | |
|---|---|---|
| Data Mining | Künstliche Intelligenz | Knowledge Discovery |
| Mustererkennung | Statistik | Big Data |

# ML-Teilbereich: Klassifikation

- Der Computer lernt, Daten zu klassifizieren
- Klassifikation „ohne Lehrer“: Clustering
- „Gewöhnliche“ Klassifikation:
  1. Trainingsphase mit vorab klassifizierten Trainingsdaten
  2. Evaluationsphase mit vorab klassifizierten Testdaten
  3. Anwendungsphase: Gelerntes Modell aus Phase 1 wird in der Praxis eingesetzt
  4. Ggf. iteratives Vorgehen zur Verbesserung des Modells

# Weka

- Java-Bibliothek, die Implementierungen von sehr vielen Maschinenlern-Algorithmen bereitstellt.
- Mehrere grafische Benutzeroberflächen zum Arbeiten und Experimentieren mit Maschinenlernen.
- XRFF: Ein einfaches XML-Format zur Dateneingabe für alle Weka-Algorithmen.
- https://www.cs.waikato.ac.nz/ml/weka/

# Saxon

- XSLT-Prozessor mit der Möglichkeit, Erweiterungsfunktionen in Java zu schreiben:
- https://www.saxonica.com/documentation/index.html#!extensibility/integratedfunctions
- Idee: Weka und Saxon verbinden, so dass aus XSLT-Skripten auf die Maschinenlern-Algorithmen von Weka zugegriffen werden kann.

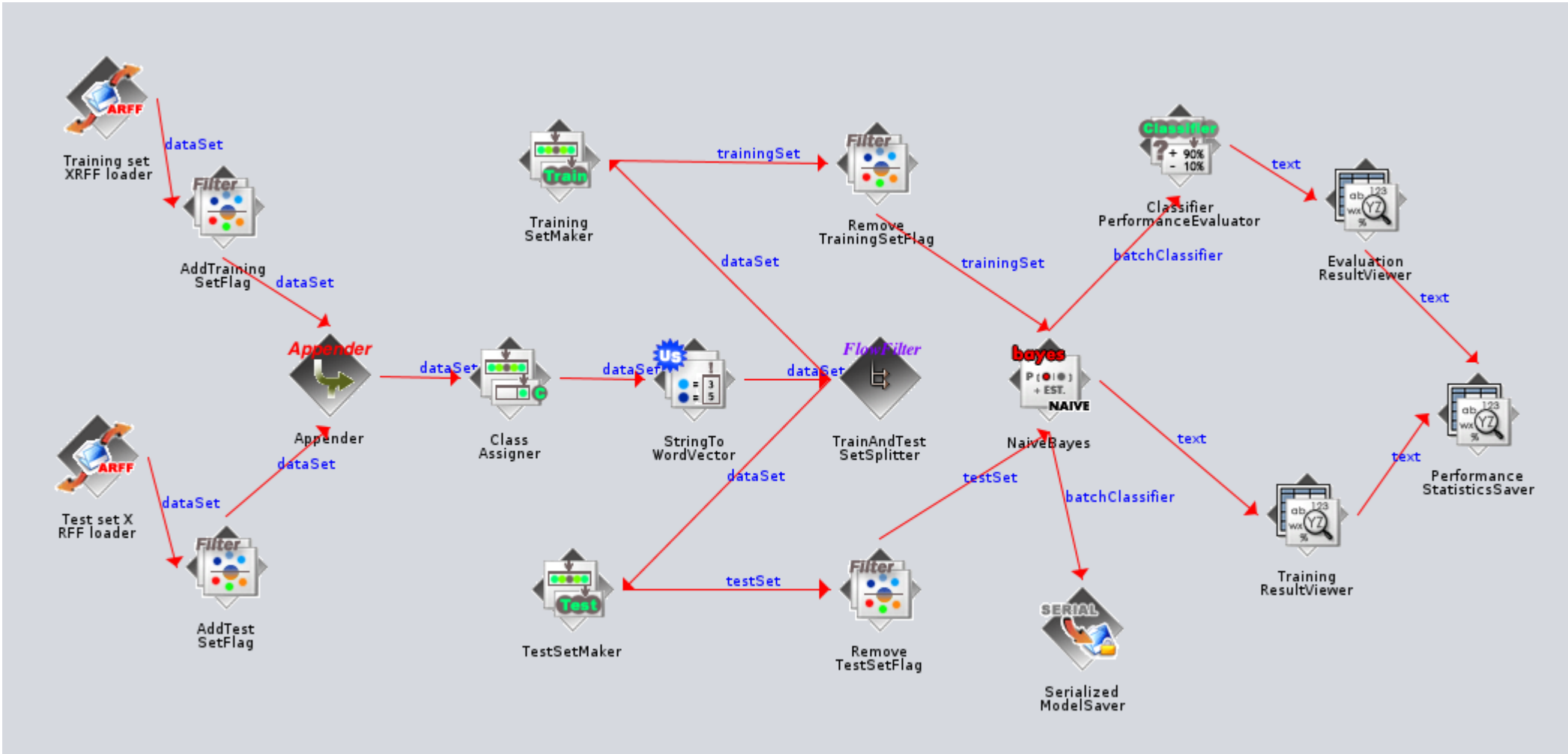
# Beispiel: Ebooks auf VLB-Warengruppen abbilden

- Ziel: Eine trainierte Maschine sollen den Volltext von E-Books einlesen und das E-Book anschließend einer VLB-Warengruppe zuordnen (z.B. WG 112 – Romane / Erzählungen oder WG 973 - Sachbuch Gesellschaft)
- Trainingsdaten: E-Books im EPUB-Format, die manuell (z.B. von Verlagen) mit einer VLB-Warengruppe versehen sind.

# Trainings- und Evaluationsphase

- XRFF-Daten erzeugen: Volltext aus EPUBs wird ausgelesen, statistisch aufbereitet (Wortarten-Quotienten, häufigste Wörter, usw.) und in XRFF-Format gespeichert.
- Maschinenlern-Verfahren wird ausgewählt und in das Training wird in Weka-GUI durchgeführt und evaluiert.
- Maschinenlern-Modell des Lernstands wird gespeichert.

# Weka Knowledge-Flow-Ansicht



# Anwendungsphase

- Extensionfunktion weka:classify() wird aus XSLT-Skript heraus aufgerufen.
- Funktionsargumente:
  1. XRFF-Dokument mit einem zu klassifierenden Datensatz.
  2. Pfad zu einer Weka-Modelldatei.
- Rückgabewert: Name der Klasse, zu der der Datensatz laut dem gelernten Model gehört.

# Noch Fragen?

- Codebeispiele im Anhang
- Kontakt:
  - Mail: sermo_de_arboribus@seznam.cz
  - Twitter: @fruehlingstag
  - Xing: https://www.xing.com/profile/Kai_Weber32
  - etc. (LinkedIn, Facebook, Github, Stackoverflow)

# Anhang: Codebeispiele

# Einfacher Aufruf aus XSLT-Skript

- Skript wird in diesem Fall ebenfalls auf XRFF-Datei angewendet
- XRFF-Datei könnte auch on-the-fly erzeugt werden

```
<xsl:template match="/">
    <xsl:message>Classifying ...<xsl:value-of select="document-uri(/)"/></xsl:message>

    <!-- Call weka classifier and put result into a variable -->
    <xsl:variable name="classificationResult">
        <xsl:value-of select="weka:classify(document(document-uri(/)),
            '/home/kai/git/Avve/Avve/weka/ergebnisse/model/naiveBayes__1_1_NaiveBayes.model')"/>
    </xsl:variable>
    <xsl:message>Document class is: <xsl:value-of select="$classificationResult"/></xsl:message>

    <!-- Write result to result document -->
    <result>
        <xsl:value-of select="$classificationResult"/>
    </result>
</xsl:template>
```

# Kommandozeilenaufruf Saxon

```
java -cp "/home/kai/git/Avve/Avve/target/avve-1.0-jar-with-dependencies.jar" \
net.sf.saxon.Transform \
-s:"/home/kai/git/Avve/Avve/output/result_2017-08-22-090400.xrff" \
-xsl:/home/kai/git/Avve/Avve/src/main/resources/xml/SaxonExtensionTest.xsl \
-o:/home/kai/git/Avve/Avve/output/SaxonExtensionTest1.xrff \
-init:avve.classify.AvveSaxonInitializer
```

# Saxon: Initializer registriert ExtensionFunction

```
package avve.classify;


import javax.xml.transform.TransformerException;
import net.sf.saxon.Configuration;
import net.sf.saxon.lib.Initializer;


public class AvveSaxonInitializer implements Initializer
{
    @Override
    public void initialize(Configuration configuration) throws TransformerException
    {
        System.out.println("Initialisiere AvveSaxon");
        configuration.registerExtensionFunction(new SaxonClassPredictorExtension());
        configuration.setValidation(false);
    }
}
```

# SaxonClassPredictorExtension (1)

```
public class SaxonClassPredictorExtension extends ExtensionFunctionDefinition
{
    @Override public StructuredQName getFunctionQName()
    {
        return new StructuredQName("weka", "http://weka.sourceforge.net", "classify");
    }

    @Override public SequenceType[] getArgumentTypes()
    {
        return new SequenceType[]
        {
        // first argument should be the root element of an xrff document; second argument the full path to a weka model file
        SequenceType.SINGLE_NODE, SequenceType.SINGLE_STRING
        };
    }

    @Override public SequenceType getResultType(SequenceType[] suppliedArgumentTypes)
    {
        return SequenceType.SINGLE_STRING;
    }
```

# SaxonClassPredictorExtension (2)

```
@Override public ExtensionFunctionCall makeCallExpression()
{
    return new ExtensionFunctionCall()
    {
        @Override public Sequence call(XPathContext context, Sequence[] arguments) throws XPathException
        {
            String classString = "";

            Processor saxonProcessor = new Processor(false);
            String xrffDocumentAsString = "";
            InputStream xrffDocumentAsInputStream = null;
            XRFFLoader xrffLoader = new XRFFLoader();

            net.sf.saxon.om.NodeInfo rootNodeInfo = ((net.sf.saxon.om.NodeInfo)arguments[0].head());
            XdmNode xdmNode = null;
            DocumentBuilder builder = saxonProcessor.newDocumentBuilder();
```

# SaxonClassPredictorExtension (3)

```
try
{
     xdmNode = builder.build(rootNodeInfo);
}
catch (SaxonApiException exc)
{
    exc.printStackTrace();
}

Serializer serializer = saxonProcessor.newSerializer();
serializer.setOutputProperty(Serializer.Property.OMIT_XML_DECLARATION, "yes");
serializer.setOutputProperty(Serializer.Property.INDENT, "yes");
serializer.setOutputProperty(Serializer.Property.ENCODING, "UTF-8");
```

# SaxonClassPredictorExtension (4)

```
        try
        {
            String xdmNodeAsString = serializer.serializeNodeToString(xdmNode);
            // weka expects an internal dtd, so we need to add one to the document before we pass the xml doc
            xrffDocumentAsString = "<?xml version=\"1.0\" encoding=\"UTF-8\"?>" + System.lineSeparator() + XrffFileWriter.dtd + System.lineSeparator() + xdmNodeAsString;
            xrffDocumentAsInputStream = IOUtils.toInputStream(xrffDocumentAsString, "UTF-8");
            xrffLoader.setSource(xrffDocumentAsInputStream);
            Instances instances = xrffLoader.getDataSet();
            String modelFilePath = arguments[1].head().getStringValue();
            ClassPredictor classPredictor = new ClassPredictor(modelFilePath);
            classString = classPredictor.classify(instances);
        }
        catch (SaxonApiException | IOException exc)
        {
            exc.printStackTrace();
        }
        return StringValue.makeStringValue(classString);
    }
};
}
```