Weka'ya algoritma nasıl eklenir?

Bu doküman Affan Hasan (affan.hasan@hotmail.com) tarafından Yıldız Teknik Üniversitesin'de verilen Kolektif Öğrenme dersi için hazırlanmıştır.

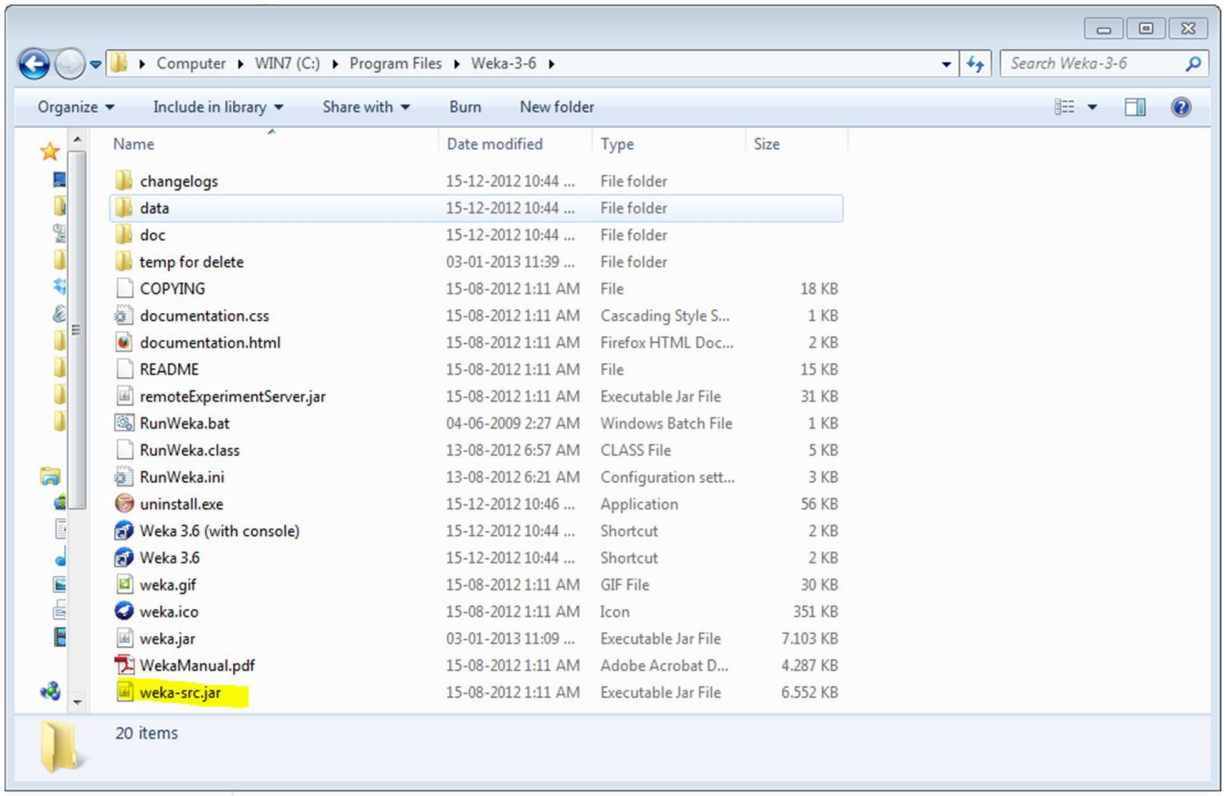
Amaç

Extended Space Forest: Extended Space Forest yöntemini sınıflandırma ve regresyon problemleri için Weka yazılımına eklemek ve diğer yöntemlerle karşılaştırmak (Java)

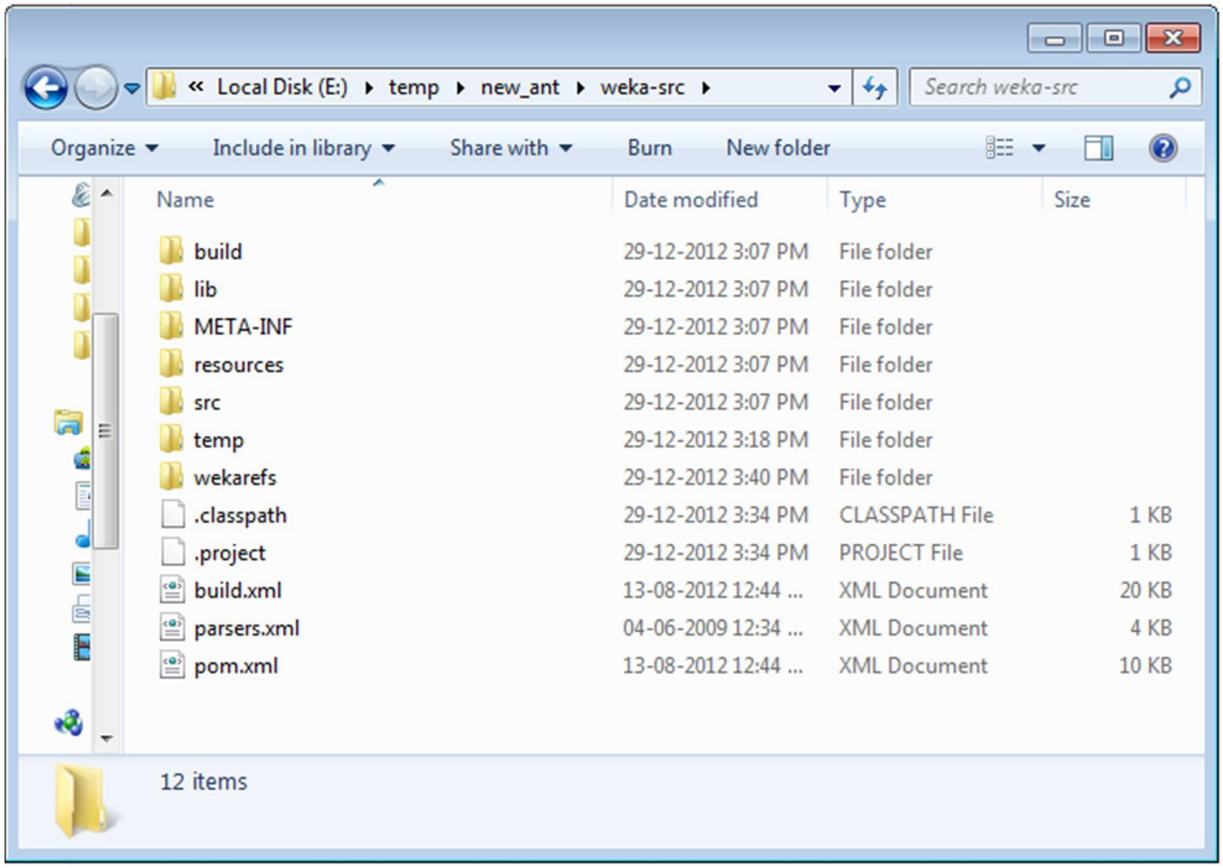
Adım adım algoritmanın eklenmesi.

Wekanın en son sürümünü indirip kurulması gerek.(Downloading and installing Weka - http://www.cs.waikato.ac.nz/ml/weka/index_downloading.html)

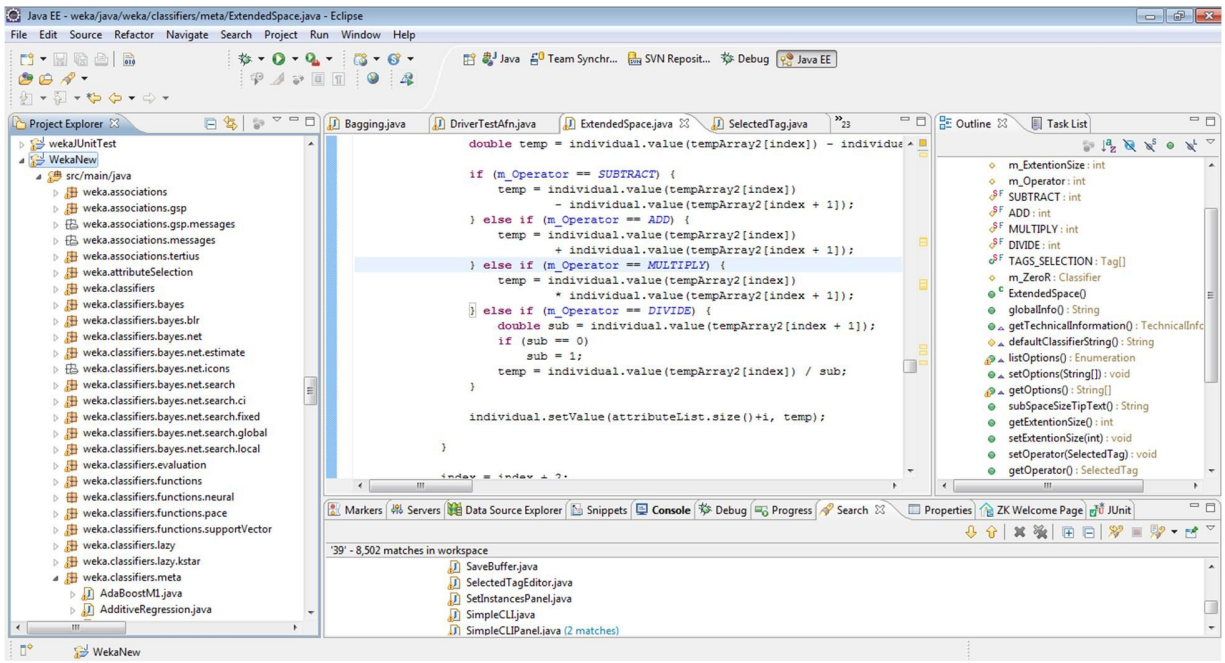
Weka 'C:\Program Files\Weka-3-6' klasörünün altına kurulur. Projenin açık koduna 'weka-src.jar' dosyasından ulaşabilirsiniz. Tabi sıkıştırılımş weka-src.jar dosyasını .zip ile açılabilir. Dosyayı açmak için benzer uygulamalar da kullanılabilir.



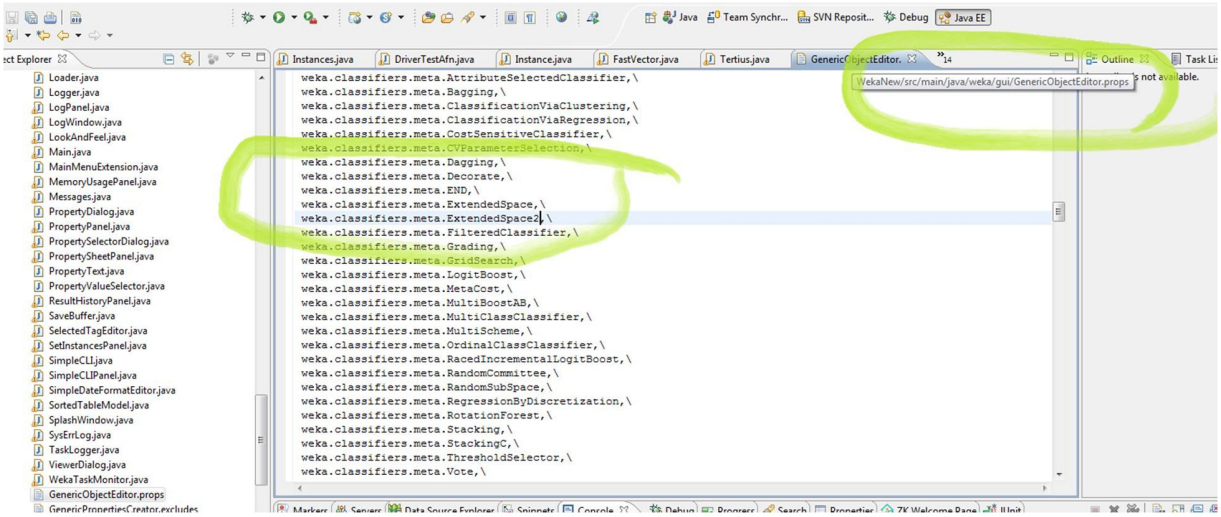
- Açılan kaynak kodun dosya yapısı:



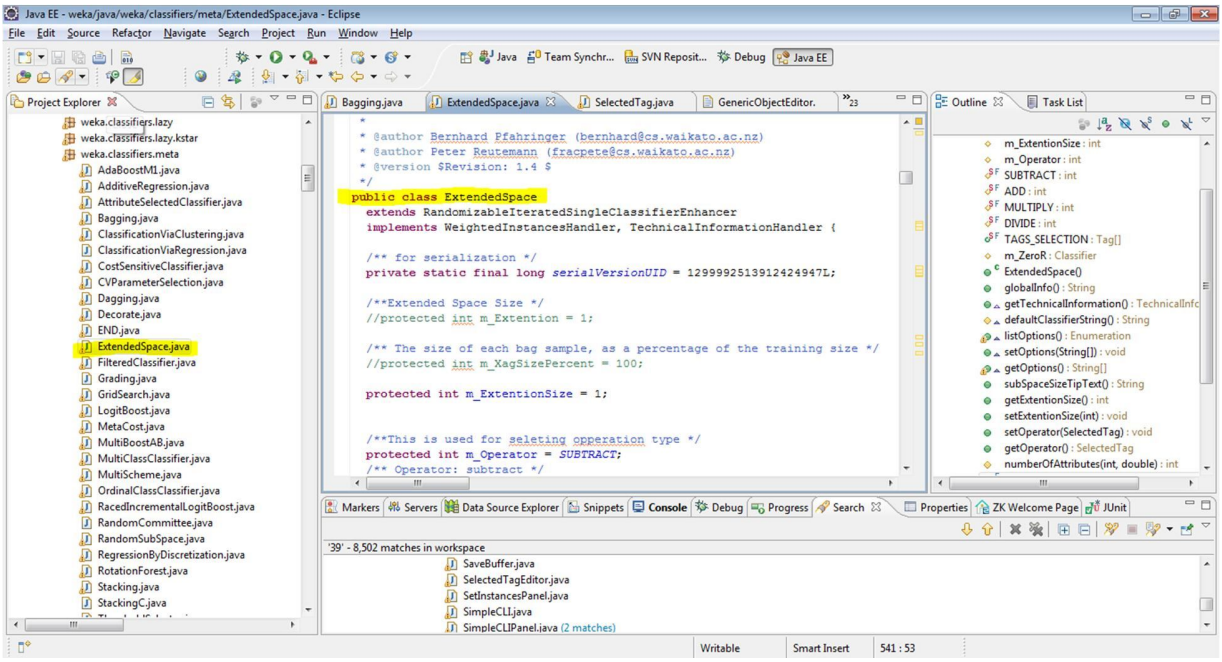
- Kaynak kod Eclipste açılır:



- İlk önce “GenericObjectEditor.props” dosyasına algoritmanın ismini ve bulunacağı yer tanımlanır. Bizim algoritmamız için “weka.classifiers.meta.ExtendedSpace,\” tanımı yapılır.



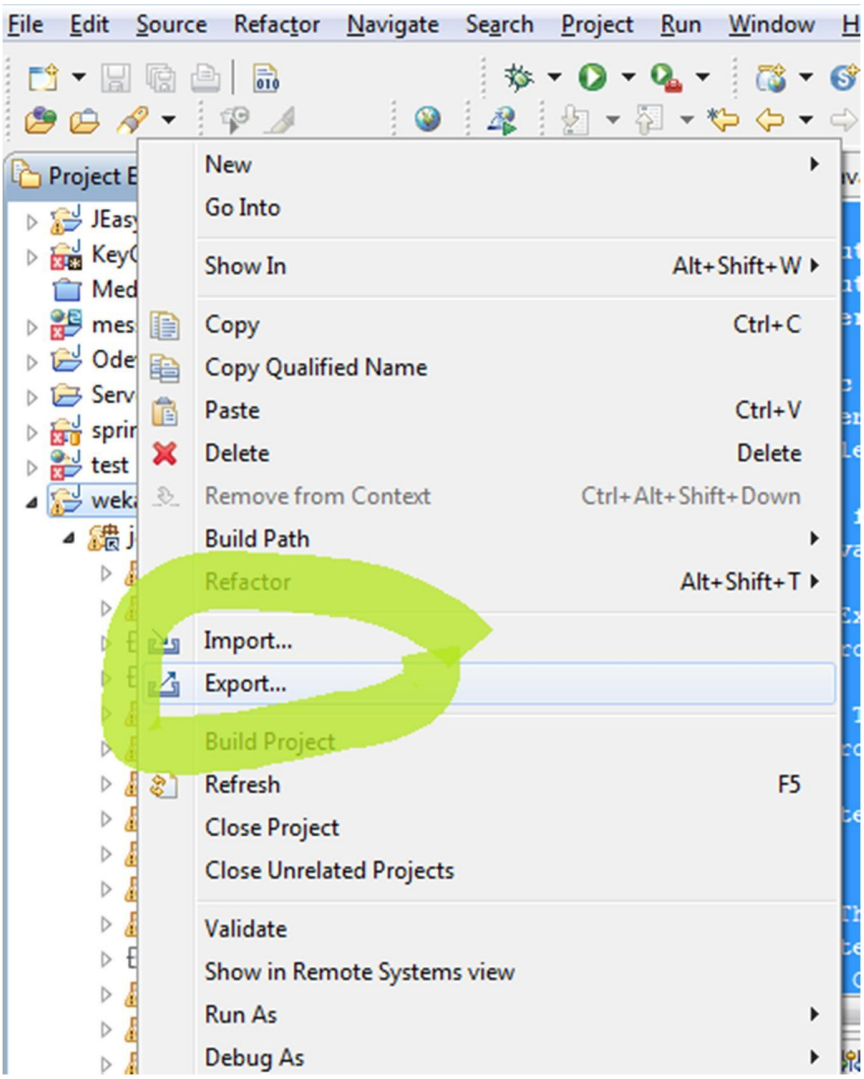
- Daha sonra Algoritmayı ekleyeceğimiz paketin altına yeni sınıfımız ekliyoruz.



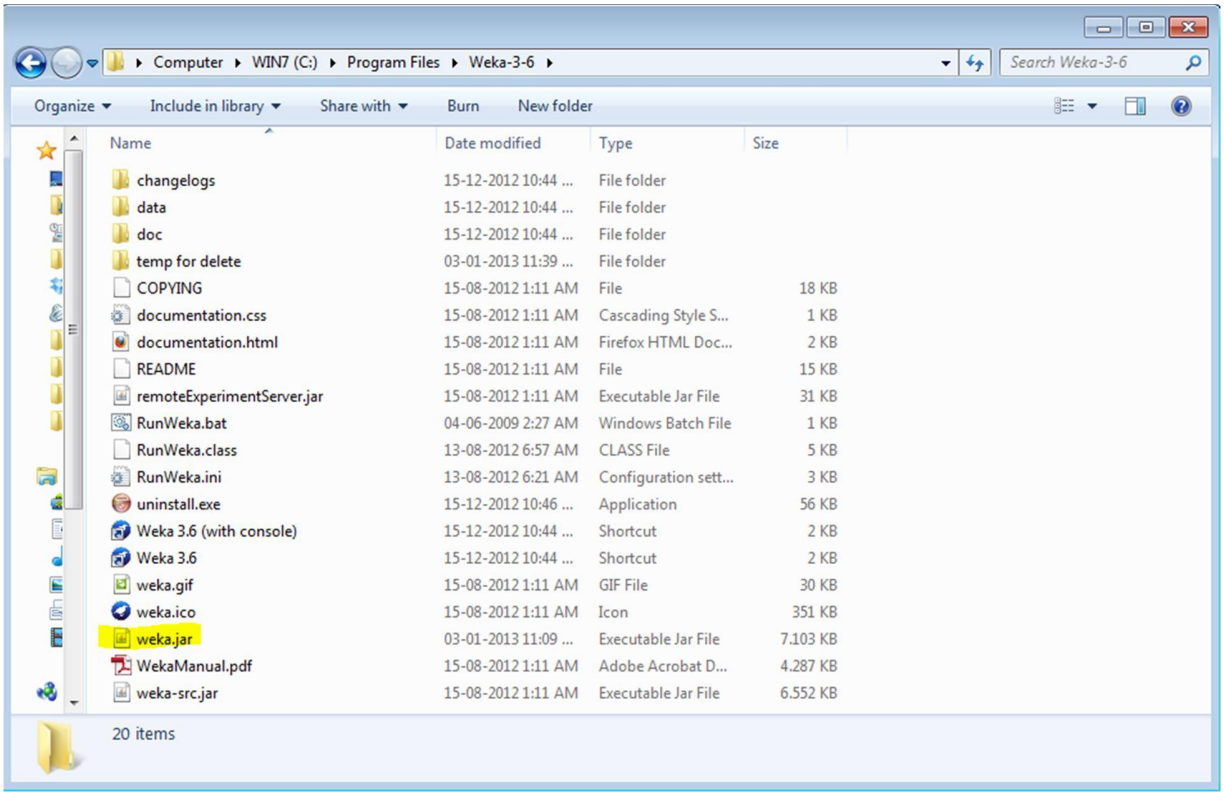
- Yeni oluşturduğumz sınıfta algoritmamızı yazıyoruz.

Bizim eklediğimiz algoritma “ExtendedSpace.java” nın kodu ektedir.

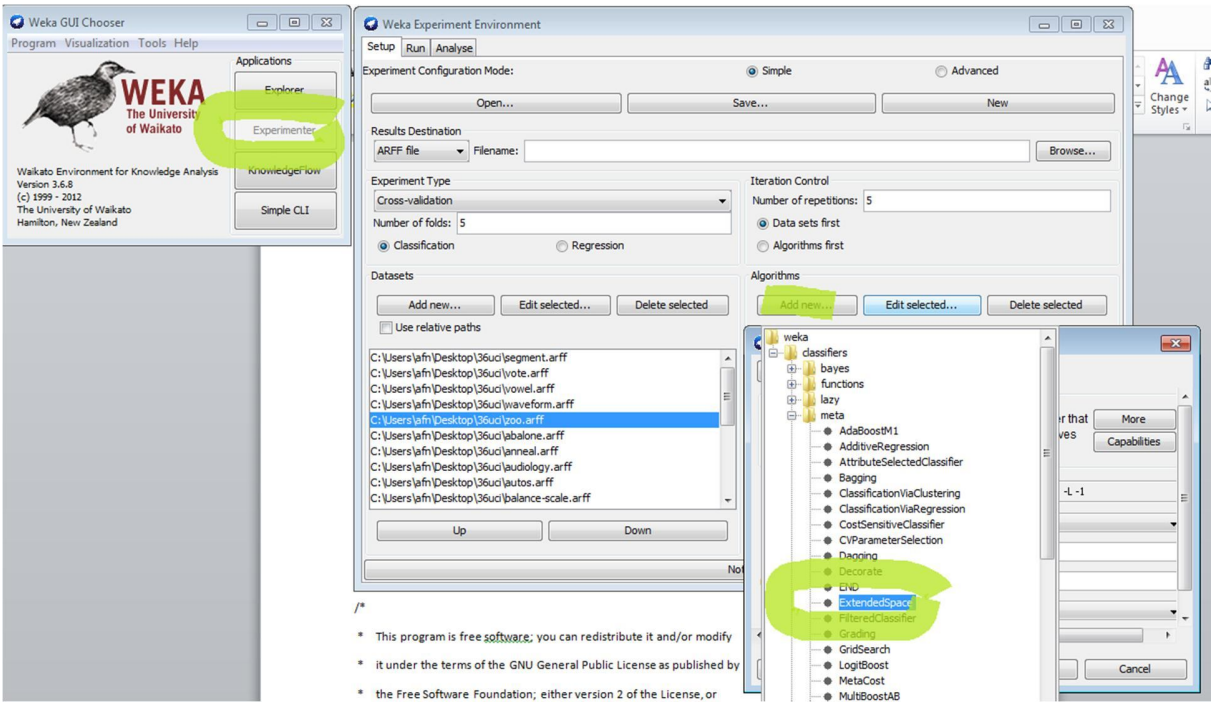
- Kod yazılıp derlendikten sonra proje “weka.jar” olarak export edilir.



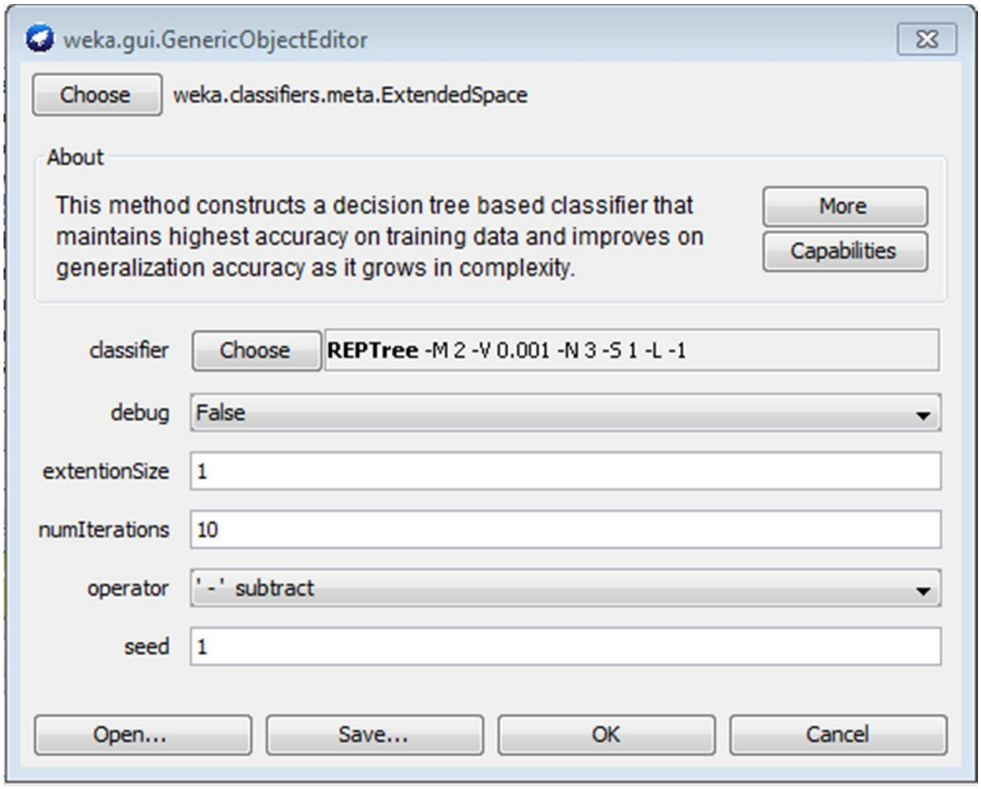
- Export edilen yeni “weka.jar”, weka projesinin içinde bulunan “weka.jar” ile değiştirilir.



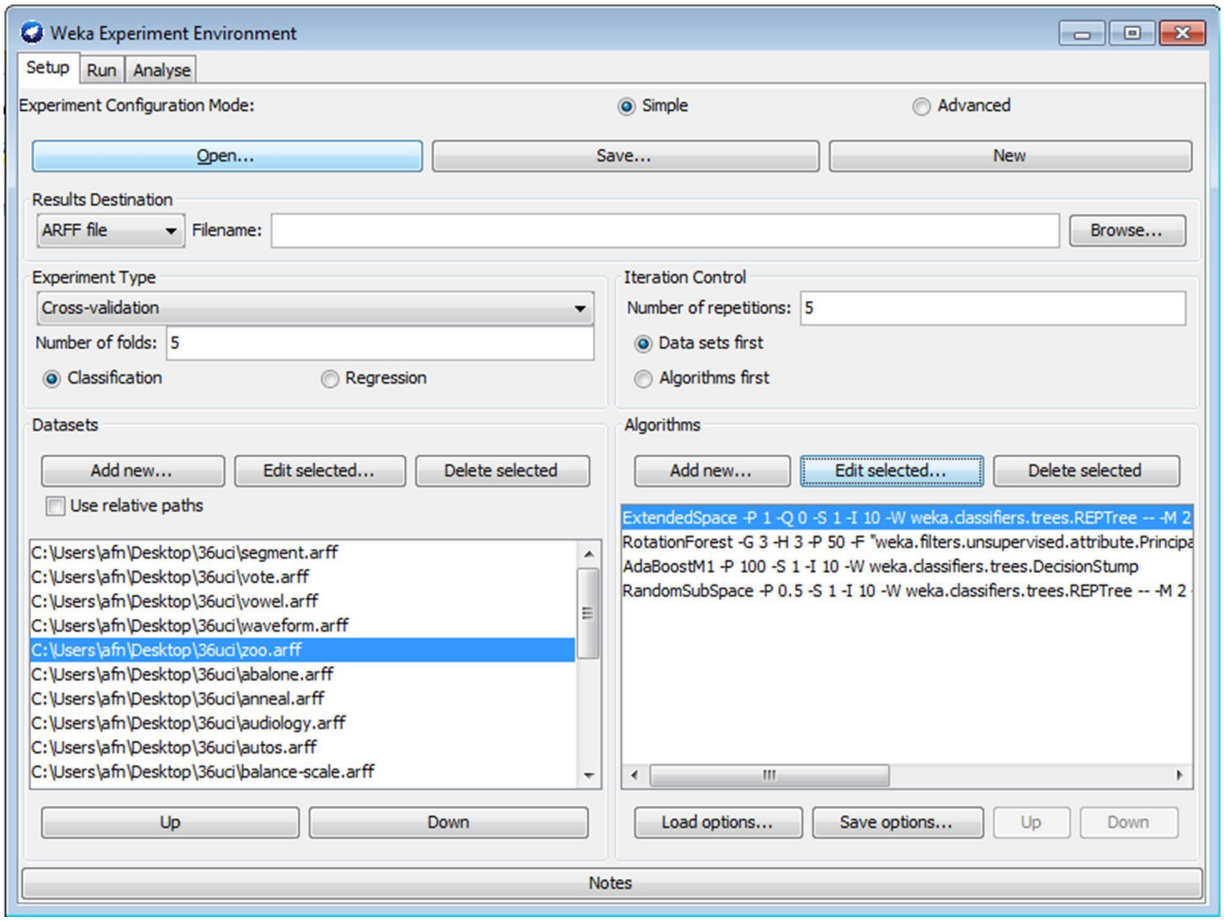
Daha sonra da Weka.exe çalıştırılıp algoritma test edilir.

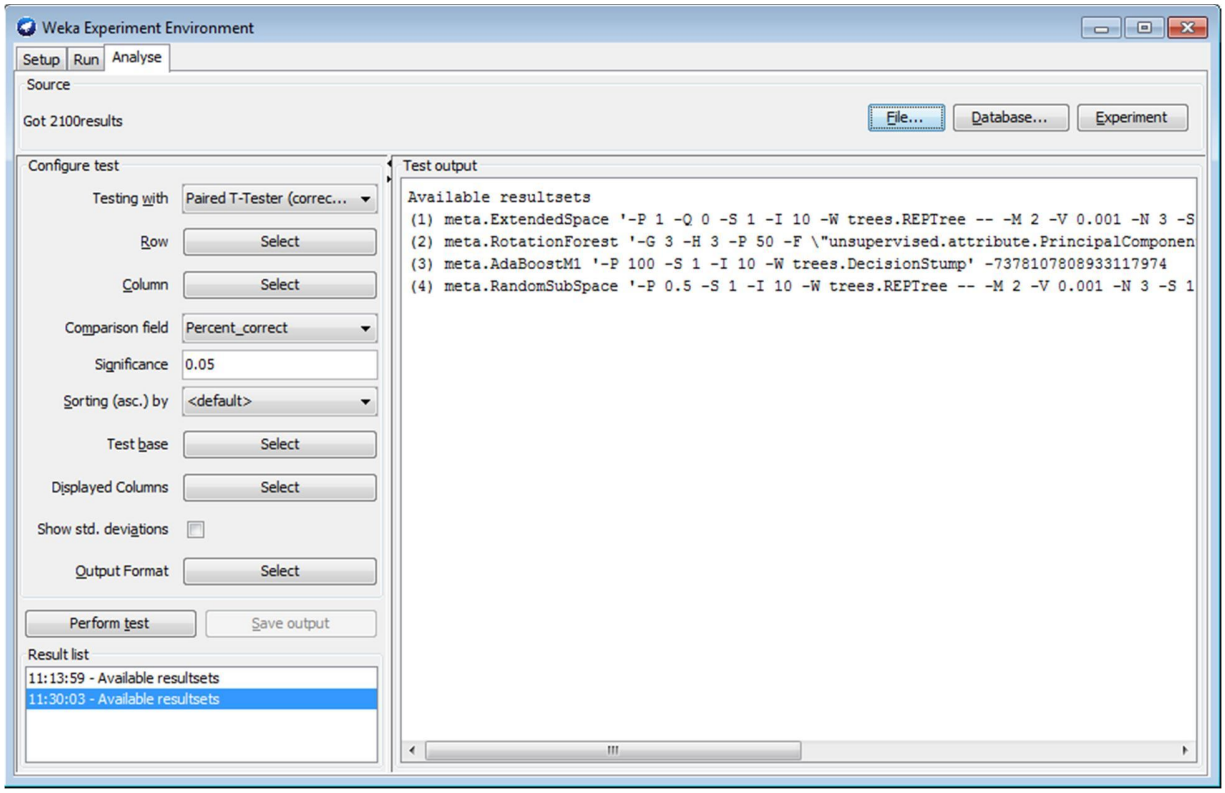


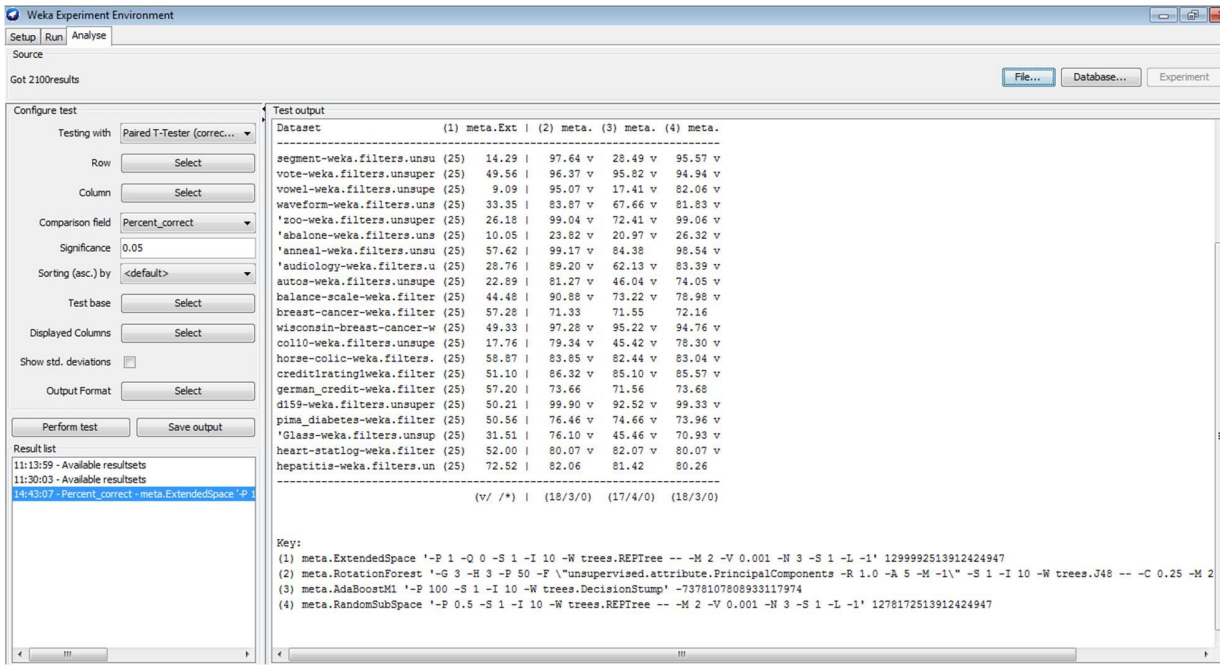
Algoritma seçildikten sonra parametreler şu şekilde setlenir:



Eklenen yeni algortimanın diğer algoritmalarla kıyaslanması :


Weka Experiment Environment
Setup
Run
Analyse
Experiment Configuration Mode:
Simple
Advanced
Open...
Save...
New
Results Destination
ARFF file
Filename:
Browse...
Experiment Type
Cross-validation
Number of folds: 5
Classification
Regression
Iteration Control
Number of repetitions: 5
Data sets first
Algorithms first
Datasets
Add new...
Edit selected...
Delete selected
Use relative paths
C:\Users\afn\Desktop\36uci\segment.arff
C:\Users\afn\Desktop\36uci\vote.arff
C:\Users\afn\Desktop\36uci\vowel.arff
C:\Users\afn\Desktop\36uci\waveform.arff
C:\Users\afn\Desktop\36uci\zoo.arff
C:\Users\afn\Desktop\36uci\abalone.arff
C:\Users\afn\Desktop\36uci\anneal.arff
C:\Users\afn\Desktop\36uci\audiology.arff
C:\Users\afn\Desktop\36uci\autos.arff
C:\Users\afn\Desktop\36uci\balance-scale.arff
Up
Down
Algorithms
Add new...
Edit selected...
Delete selected
ExtendedSpace -P 1 -Q 0 -S 1 -I 10 -W weka.classifiers.trees.REPTree -- -M 2
RotationForest -G 3 -H 3 -P 50 -F "weka.filters.unsupervised.attribute.Principa
AdaBoostM1 -P 100 -S 1 -I 10 -W weka.classifiers.trees.DecisionStump
RandomSubSpace -P 0.5 -S 1 -I 10 -W weka.classifiers.trees.REPTree -- -M 2
Load options...
Save options...
Up
Down
Notes


Weka Experiment Environment
Setup
Run
Analyse
Source
Got 2100results
File...
Database...
Experiment
Configure test
Testing with
Paired T-Tester (correc...
Row
Select
Column
Select
Comparison field
Percent_correct
Significance
0.05
Sorting (asc.) by
<default>
Test base
Select
Displayed Columns
Select
Show std. deviations
Output Format
Select
Perform test
Save output
Result list
11:13:59 - Available resultsets
11:30:03 - Available resultsets
Test output
Available resultsets
(1) meta.ExtendedSpace '-P 1 -Q 0 -S 1 -I 10 -W trees.REPTree -- -M 2 -V 0.001 -N 3 -S
(2) meta.RotationForest '-G 3 -H 3 -P 50 -F \"unsupervised.attribute.PrincipalComponen
(3) meta.AdaBoostM1 '-P 100 -S 1 -I 10 -W trees.DecisionStump' -7378107808933117974
(4) meta.RandomSubSpace '-P 0.5 -S 1 -I 10 -W trees.REPTree -- -M 2 -V 0.001 -N 3 -S 1


Weka Experiment Environment
Setup
Run
Analyse
Source
Got 2100results
File...
Database...
Experiment
Configure test
Testing with
Paired T-Tester (correc...
Row
Select
Column
Select
Comparison field
Percent_correct
Significance
0.05
Sorting (asc.) by
<default>
Test base
Select
Displayed Columns
Select
Show std. deviations
Output Format
Select
Perform test
Save output
Result list
11:13:59 - Available resultsets
11:30:03 - Available resultsets
14:43:07 - Percent_correct - meta.ExtendedSpace '-P 1
Test output
Dataset (1) meta.Ext | (2) meta. (3) meta. (4) meta.
segment-weka.filters.unsu (25) 14.29 | 97.64 v 28.49 v 95.57 v
vote-weka.filters.unsuper (25) 49.56 | 96.37 v 95.82 v 94.94 v
vowel-weka.filters.unsupe (25) 9.09 | 95.07 v 17.41 v 82.06 v
waveform-weka.filters.uns (25) 33.35 | 83.87 v 67.66 v 81.83 v
'zoo-weka.filters.unsuper (25) 26.18 | 99.04 v 72.41 v 99.06 v
'abalone-weka.filters.uns (25) 10.05 | 23.82 v 20.97 v 26.32 v
'anneal-weka.filters.unsu (25) 57.62 | 99.17 v 84.38 98.54 v
'audiology-weka.filters.u (25) 28.76 | 89.20 v 62.13 v 83.39 v
autos-weka.filters.unsupe (25) 22.89 | 81.27 v 46.04 v 74.05 v
balance-scale-weka.filter (25) 44.48 | 90.88 v 73.22 v 78.98 v
breast-cancer-weka.filter (25) 57.28 | 71.33 71.55 72.16
wisconsin-breast-cancer-w (25) 49.33 | 97.28 v 95.22 v 94.76 v
col10-weka.filters.unsupe (25) 17.76 | 79.34 v 45.42 v 78.30 v
horse-colic-weka.filters. (25) 58.87 | 83.85 v 82.44 v 83.04 v
credit1rating1weka.filter (25) 51.10 | 86.32 v 85.10 v 85.57 v
german_credit-weka.filter (25) 57.20 | 73.66 71.56 73.68
d159-weka.filters.unsuper (25) 50.21 | 99.90 v 92.52 v 99.33 v
pima_diabetes-weka.filter (25) 50.56 | 76.46 v 74.66 v 73.96 v
'Glass-weka.filters.unsup (25) 31.51 | 76.10 v 45.46 v 70.93 v
heart-statlog-weka.filter (25) 52.00 | 80.07 v 82.07 v 80.07 v
hepatitis-weka.filters.un (25) 72.52 | 82.06 81.42 80.26
(v/ /*) | (18/3/0) (17/4/0) (18/3/0)
Key:
(1) meta.ExtendedSpace '-P 1 -Q 0 -S 1 -I 10 -W trees.REPTree -- -M 2 -V 0.001 -N 3 -S 1 -L -1' 1299992513912424947
(2) meta.RotationForest '-G 3 -H 3 -P 50 -F \"unsupervised.attribute.PrincipalComponents -R 1.0 -A 5 -M -1\" -S 1 -I 10 -W trees.J48 -- -C 0.25 -M 2
(3) meta.AdaBoostM1 '-P 100 -S 1 -I 10 -W trees.DecisionStump' -7378107808933117974
(4) meta.RandomSubSpace '-P 0.5 -S 1 -I 10 -W trees.REPTree -- -M 2 -V 0.001 -N 3 -S 1 -L -1' 1278172513912424947

ExtendedSpace.java

```
/*
 *   This program is free software; you can redistribute it and/or modify
 *   it under the terms of the GNU General Public License as published by
 *   the Free Software Foundation; either version 2 of the License, or
 *   (at your option) any later version.
 *
 *   This program is distributed in the hope that it will be useful,
 *   but WITHOUT ANY WARRANTY; without even the implied warranty of
 *   MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 *   GNU General Public License for more details.
 *
 *   You should have received a copy of the GNU General Public License
 *   along with this program; if not, write to the Free Software
 *   Foundation, Inc., 675 Mass Ave, Cambridge, MA 02139, USA.
 */

/*
 *   ExtendedSpace2.java
 *   Copyright (C) 2006 University of Waikato, Hamilton, New Zealand
 *
 */

package weka.classifiers.meta;

import weka.filters.unsupervised.attribute.Remove;
import weka.classifiers.Classifier;
import weka.classifiers.RandomizableIteratedSingleClassifierEnhancer;
import weka.core.Attribute;
```

```
import weka.core.Instance;

import weka.core.Instances;

import weka.core.Option;

import weka.core.Randomizable;

import weka.core.RevisionUtils;

import weka.core.SelectedTag;

import weka.core.Tag;

import weka.core.TechnicalInformation;

import weka.core.TechnicalInformationHandler;

import weka.core.Utils;

import weka.core.WeightedInstancesHandler;

import weka.core.TechnicalInformation.Field;

import weka.core.TechnicalInformation.Type;


import java.util.ArrayList;

import java.util.Enumeration;

import java.util.List;

import java.util.Random;

import java.util.Vector;

import java.util.Arrays;

import java.util.Collections;


/**

 <!-- globalinfo-start -->

 * This method constructs a decision tree based classifier that maintains highest accuracy on training data and improves on generalization accuracy as it grows in complexity. The classifier consists of multiple trees constructed systematically by pseudorandomly selecting subsets of components of the feature vector, that is, trees constructed in randomly chosen subspaces.<br/>

 * <br/>

 * For more information, see<br/>
```

*

* Tin Kam Ho (1998). The Random Subspace Method for Constructing Decision Forests. IEEE Transactions on Pattern Analysis and Machine Intelligence. 20(8):832-844. URL http://citeseer.ist.psu.edu/ho98random.html.

* <p/>

<!-- globalinfo-end -->

*

<!-- technical-bibtex-start -->

* BibTeX:

* <pre>

* @article{Ho1998,

* author = {Tin Kam Ho},

* journal = {IEEE Transactions on Pattern Analysis and Machine Intelligence},

* number = {8},

* pages = {832-844},

* title = {The Random Subspace Method for Constructing Decision Forests},

* volume = {20},

* year = {1998},

* ISSN = {0162-8828},

* URL = {http://citeseer.ist.psu.edu/ho98random.html}

* }

* </pre>

* <p/>

<!-- technical-bibtex-end -->

*

<!-- options-start -->

* Valid options are: <p/>

*

* <pre> -P

* How many times from the original data will be

* (default 1)

* </pre>

*

* <pre> -Q

* Use Operator for Extention.

* (defult '-' subtract)

* </pre>

*

* <pre> -S <num>

* Random number seed.

* (default 1)</pre>

*

* <pre> -I <num>

* Number of iterations.

* (default 10)</pre>

*

* <pre> -D

* If set, classifier is run in debug mode and

* may output additional info to the console</pre>

*

* <pre> -W

* Full name of base classifier.

* (default: weka.classifiers.trees.REPTree)</pre>

*

* <pre>

* Options specific to classifier weka.classifiers.trees.REPTree:

* </pre>

*

* <pre> -M <minimum number of instances>

* Set minimum number of instances per leaf (default 2).</pre>

```
 *
 * <pre> -V &lt;minimum variance for split&gt;
 *  Set minimum numeric class variance proportion
 *  of train variance for split (default 1e-3).</pre>
 *
 * <pre> -N &lt;number of folds&gt;
 *  Number of folds for reduced error pruning (default 3).</pre>
 *
 * <pre> -S &lt;seed&gt;
 *  Seed for random data shuffling (default 1).</pre>
 *
 * <pre> -P
 *  No pruning.</pre>
 *
 * <pre> -L
 *  Maximum tree depth (default -1, no maximum)</pre>
 *
 <!-- options-end -->
 *
 * Options after -- are passed to the designated classifier.<p>
 *
 * @author Bernhard Pfahringer (bernhard@cs.waikato.ac.nz)
 * @author Peter Reutemann (fracpete@cs.waikato.ac.nz)
 * @version $Revision: 1.4 $
 */
public class ExtendedSpace
  extends RandomizableIteratedSingleClassifierEnhancer
  implements WeightedInstancesHandler, TechnicalInformationHandler {
```

```
/** for serialization */
private static final long serialVersionUID = 12999925139124249 47L;

/**Extended Space Size */
//protected int m_Extention = 1;

/** The size of each bag sample, as a percentage of the training size */
//protected int m_XagSizePercent = 100;

protected int m_ExtentionSize = 1;



/**This is used for seleting opperation type */
protected int m_Operator = SUBTRACT;
/** Operator: subtract */
protected static final int SUBTRACT = 0;
/** Operator: add */
protected static final int ADD = 1;
/** Operator: multiply */
protected static final int MULTIPLY = 2;
/** Operator: divide */
protected static final int DIVIDE = 3;
/** search Operator */
public static final Tag [] TAGS_SELECTION = {
  new Tag(SUBTRACT, "' - ' subtract"),
  new Tag(ADD, "' + ' add"),
  new Tag(MULTIPLY, "' * ' multiply"),
  new Tag(DIVIDE, "' / ' divide"),
};
```

```
/** The size of each bag sample, as a percentage of the training size */
//protected double m_SubSpaceSize = 0.5;

/** a ZeroR model in case no model can be built from the data */
protected Classifier m_ZeroR;

/**
 * Constructor.
 */
public ExtendedSpace() {
  super();

  m_Classifier = new weka.classifiers.trees.REPTree();
}

/**
 * Returns a string describing classifier
 *
 * @return                    a description suitable for
 *                            displaying in the explorer/experimenter gui
 */
public String globalInfo() {
  return
      "This method constructs a decision tree based classifier that "
    + "maintains highest accuracy on training data and improves on "
    + "generalization accuracy as it grows in complexity. The classifier "
    + "consists of multiple trees constructed systematically by "
    + "pseudorandomly selecting subsets of components of the feature vector, "
```

```
    + "that is, trees constructed in randomly chosen subspaces.\n\n"
    + "For more information, see\n\n"
    + getTechnicalInformation().toString();
}

/**
 * Returns an instance of a TechnicalInformation object, containing
 * detailed information about the technical background of this class,
 * e.g., paper reference or book this class is based on.
 *
 * @return                  the technical information about this class
 */
public TechnicalInformation getTechnicalInformation() {
  TechnicalInformation     result;

  result = new TechnicalInformation(Type.ARTICLE);
  result.setValue(Field.AUTHOR, "Affan HASAN");
  result.setValue(Field.YEAR, "2013");
  result.setValue(Field.TITLE, "The Random Subspace Method for Constructing Decision Forests");
  result.setValue(Field.JOURNAL, "IEEE Transactions on Pattern Analysis and Machine Intelligence");
  result.setValue(Field.VOLUME, "20");
  result.setValue(Field.NUMBER, "8");
  result.setValue(Field.PAGES, "832-844");
  result.setValue(Field.URL, "http://citeseer.ist.psu.edu/ho98random.html");
  result.setValue(Field.ISSN, "0162-8828");

  return result;
}
```

```
/**
 * String describing default classifier.
 *
 * @return                    the default classifier classname
 */
protected String defaultClassifierString() {
  return "weka.classifiers.trees.REPTree";
}


/**
 * Returns an enumeration describing the available options.
 *
 * @return                    an enumeration of all the available options.
 */
public Enumeration listOptions() {
  Vector result = new Vector();

  result.addElement(new Option(
            "\tSize of each subspace:\n"
            + "\t\t< 1: percentage of the number of attributes\n"
            + "\t\t>=1: absolute number of attributes\n",
            "P", 1, "-P"));

  Enumeration enu = super.listOptions();
  while (enu.hasMoreElements()) {
    result.addElement(enu.nextElement());
  }

  return result.elements();
```

```
}

/**
 * Parses a given list of options. <p/>
 *
 <!-- options-start -->
 * Valid options are: <p/>
 *
 * <pre> -P
 *  Size of each subspace:
 *   &lt; 1: percentage of the number of attributes
 *   &gt;=1: absolute number of attributes
 * </pre>
 *
 * <pre> -S &lt;num&gt;
 *  Random number seed.
 *  (default 1)</pre>
 *
 * <pre> -I &lt;num&gt;
 *  Number of iterations.
 *  (default 10)</pre>
 *
 * <pre> -D
 *  If set, classifier is run in debug mode and
 *  may output additional info to the console</pre>
 *
 * <pre> -W
 *  Full name of base classifier.
 *  (default: weka.classifiers.trees.REPTree)</pre>
```

```
*
* <pre>
* Options specific to classifier weka.classifiers.trees.REPTree:
* </pre>
*
* <pre> -M &lt;minimum number of instances&gt;
*  Set minimum number of instances per leaf (default 2).</pre>
*
* <pre> -V &lt;minimum variance for split&gt;
*  Set minimum numeric class variance proportion
*  of train variance for split (default 1e-3).</pre>
*
* <pre> -N &lt;number of folds&gt;
*  Number of folds for reduced error pruning (default 3).</pre>
*
* <pre> -S &lt;seed&gt;
*  Seed for random data shuffling (default 1).</pre>
*
* <pre> -P
*  No pruning.</pre>
*
* <pre> -L
*  Maximum tree depth (default -1, no maximum)</pre>
*
<!-- options-end -->
*
* Options after -- are passed to the designated classifier.<p>
*
* @param options          the list of options as an array of strings
```

```
 * @throws Exception      if an option is not supported
 */
public void setOptions(String[] options) throws Exception {



        String thresholdString = Utils.getOption('P', options);
        if (thresholdString.length() != 0) {
                setExtentionSize(Integer.parseInt(thresholdString));
        } else {
                setExtentionSize(1);
        }




  super.setOptions(options);
}


/**
 * Gets the current settings of the Classifier.
 *
 * @return                  an array of strings suitable for passing to setOptions
 */
public String [] getOptions() {
  Vector       result;
  String[]     options;
  int          i;


  result  = new Vector();
```

```
  result.add("-P");
  result.add("" + getExtentionSize());

  result.add("-Q");
  result.add("" + getOperator());

  options = super.getOptions();
  for (i = 0; i < options.length; i++)
    result.add(options[i]);

  return (String[]) result.toArray(new String[result.size()]);
}

/**
 * Returns the tip text for this property
 *
 * @return                  tip text for this property suitable for
 *                          displaying in the explorer/experimenter gui
 */
public String subSpaceSizeTipText() {
  return
      "Size of each subSpace: if less than 1 as a percentage of the "
    + "number of attributes, otherwise the absolute number of attributes.";
}

/**
 * Gets the size of each bag, as a percentage of the training set size.
 *
 * @return the bag size, as a percentage.
```

```
 */
/*
public int getXagSizePercent() {

  return m_XagSizePercent;
}
*/
/**
 * Sets the size of each bag, as a percentage of the training set size.
 *
 * @param newBagSizePercent the bag size, as a percentage.
 */
/*
public void setXagSizePercent(int newBagSizePercent) {

  m_XagSizePercent = newBagSizePercent;
}
*/
/**
 * Gets the size of each bag, as a percentage of the training set size.
 *
 * @return the ExtendedSpace size, as an integer.
 */
public int getExtentionSize() {

  return m_ExtentionSize;
}

/**
```

```
 * Sets the size of each bag, as a percentage of the training set size.
 *
 * @param newExtentionSize the ExtendedSpace size, as an integer.
 */
public void setExtentionSize(int newExtentionSize) {

        m_ExtentionSize = newExtentionSize;
}



/**
 * Gets the size of each ExtendedSpace.
 *
 * @return                  the ExtendedSpace size, as an integer.
 */
/*
public double getExtention() {
  return m_Extention;
}
*/
/**
 * Sets the size of each ExtendedSpace.
 *
 * @param value             the ExtendedSpace Size, as an integer.
 */
/*
public void setExtention(int value) {
        m_Extention = value;
}
```

```
*/
/**
 * Set the search direction
 *
 * @param d the direction of the search
 */
public void setOperator (SelectedTag d) {

  if (d.getTags() == TAGS_SELECTION) {
            m_Operator = d.getSelectedTag().getID();
  }
}



/**
 * Get the search direction
 *
 * @return the direction of the search
 */
public SelectedTag getOperator () {

  return new SelectedTag(m_Operator, TAGS_SELECTION);
}

/**
 * calculates the number of attributes
 *
 * @param total the available number of attributes
 * @param fraction          the fraction - if less than 1 it represents the
```

```
 *                          percentage, otherwise the absolute number of attributes
 * @return                  the number of attributes to use
 */
protected int numberOfAttributes(int total, double fraction) {
  int k = (int) Math.round((fraction < 1.0) ? total*fraction : fraction);

  if (k > total)
    k = total;
  if (k < 1)
    k = 1;

  return k;
}
protected final Instances generateExtendedSpaceData(Instances newData,int j) {

        //Instances data = null;
        //data = newData;
                Instances data = new Instances(newData, newData.numInstances());
                int k = 0;
                int l = 0;
                while ((k < newData.numInstances() && (l < newData.numInstances()))) {
                        while ((k < newData.numInstances())) {
                                data.add(newData.instance(l));
                                // newData.instance(k).setWeight(1);
                                k++;
                        }
                        l++;
                }
```

```
Enumeration attributesForCount2 = data.enumerateAttributes();

List<String> attributeList = new ArrayList<String>();
while (attributesForCount2.hasMoreElements()) {
        Attribute attr = (Attribute) attributesForCount2.nextElement();
        attributeList.add(attr.name());

}

//extention size cannot be less then 1
if (m_ExtentionSize<1)
        m_ExtentionSize=1;

int[] tempArray2 = getRandomAttributeArray(attributeList.size(), m_ExtentionSize);

String Operator = "_-_";

if (m_Operator==SUBTRACT){
        Operator = "_-_";
}else if(m_Operator==ADD){
        Operator = "_+_";
}else if(m_Operator==MULTIPLY){
        Operator = "_*_";
}else if(m_Operator==DIVIDE){
        Operator = "_/_";
}

int index = 0;
for (int i = 0; i < tempArray2.length / 2; i++) {
```

```
String newString = attributeList.get(tempArray2[index]) + "-" +
attributeList.get(tempArray2[index+1]) + "_" +(i+1)+"_"+j;

Attribute AttributeTemp = new Attribute(newString);

data.insertAttributeAt(AttributeTemp, attributeList.size()+i);


//Enumeration attributes = isTrainingSet2.enumerateAttributes();


Enumeration instances2 = data.enumerateInstances();

while (instances2.hasMoreElements()) {


    Instance individual = (Instance) instances2.nextElement();


    double temp = individual.value(tempArray2[index]) -
individual.value(tempArray2[index+1]);


    if (m_Operator == SUBTRACT) {

        temp = individual.value(tempArray2[index])

                - individual.value(tempArray2[index + 1]);

    } else if (m_Operator == ADD) {

        temp = individual.value(tempArray2[index])

                + individual.value(tempArray2[index + 1]);

    } else if (m_Operator == MULTIPLY) {

        temp = individual.value(tempArray2[index])

                * individual.value(tempArray2[index + 1]);

    } else if (m_Operator == DIVIDE) {

        double sub = individual.value(tempArray2[index + 1]);

        if (sub == 0)

            sub = 1;

        temp = individual.value(tempArray2[index]) / sub;
```

```
                                        }

                                        individual.setValue(attributeList.size()+i, temp);

                                }

                                index = index + 2;
                        }


                        return data;
                }

/**
 * generates an index string describing a random subspace, suitable for
 * the Remove filter.
 *
 * @param indices               the attribute indices
 * @param subSpaceSize the size of the subspace
 * @param classIndex            the class index
 * @param random                the random number generator
 * @return                      the generated string describing the subspace
 */
protected String ExtendedSpace2(Integer[] indices, int subSpaceSize, int classIndex, Random random) {
  Collections.shuffle(Arrays.asList(indices), random);
  StringBuffer sb = new StringBuffer("");
  for(int i = 0; i < subSpaceSize; i++) {
    sb.append(indices[i]+",");
  }
```

```
  sb.append(classIndex);

  if (getDebug())
    System.out.println("subSPACE = " + sb);

  return sb.toString();
}

/**
 * builds the classifier.
 *
 * @param data the training data to be used for generating the
 *                              classifier.
 * @throws Exception      if the classifier could not be built successfully
 */
          public void buildClassifier(Instances data) throws Exception {

                    // can classifier handle the data?
                    getCapabilities().testWithFail(data);

                    // remove instances with missing class
                    data = new Instances(data);
                    data.deleteWithMissingClass();

                    // only class? -> build ZeroR model
                    if (data.numAttributes() == 1) {
                              System.err
                                        .println("Cannot build model (only class attribute present in data!), "
                                                  + "using ZeroR model instead!");
```

```
            m_ZeroR = new weka.classifiers.rules.ZeroR();
            m_ZeroR.buildClassifier(data);
            return;
} else {
            m_ZeroR = null;
}

super.buildClassifier(data);

/*
Integer[] indices = new Integer[data.numAttributes() - 1];
int classIndex = data.classIndex();
int offset = 0;
for (int i = 0; i < indices.length + 1; i++) {
            if (i != classIndex) {
                        indices[offset++] = i + 1;
            }
}
*/

//int subSpaceSize = numberOfAttributes(indices.length, 0.5);
Random random = data.getRandomNumberGenerator(m_Seed);

for (int j = 0; j < m_Classifiers.length; j++) {
            if (m_Classifier instanceof Randomizable) {
                        ((Randomizable) m_Classifiers[j]).setSeed(random.nextInt());
            }

            Instances extendedSpaceData = null;
```

```
                    extendedSpaceData = generateExtendedSpaceData(data,j);

                    // build the classifier
                    m_Classifiers[j].buildClassifier(extendedSpaceData);
                }

        }

/**
 * Calculates the class membership probabilities for the given test
 * instance.
 *
 * @param instance        the instance to be classified
 * @return                preedicted class probability distribution
 * @throws Exception      if distribution can't be computed successfully
 */
public double[] distributionForInstance(Instance instance) throws Exception {

  // default model?
  if (m_ZeroR != null) {
    return m_ZeroR.distributionForInstance(instance);
  }

  double[] sums = new double [instance.numClasses()], newProbs;

  for (int i = 0; i < m_NumIterations; i++) {
    if (instance.classAttribute().isNumeric() == true) {
        sums[0] += m_Classifiers[i].classifyInstance(instance);
    } else {
```

```
            newProbs = m_Classifiers[i].distributionForInstance(instance);
            for (int j = 0; j < newProbs.length; j++)
              sums[j] += newProbs[j];
    }
  }
  if (instance.classAttribute().isNumeric() == true) {
    sums[0] /= (double)m_NumIterations;
    return sums;
  } else if (Utils.eq(Utils.sum(sums), 0)) {
    return sums;
  } else {
    Utils.normalize(sums);
    return sums;
  }
}


            protected final int[] getRandomAttributeArray(int attributeSize,
                                    int newAttributeSize) {


                        int AttributeArray[] = new int[2 * attributeSize * newAttributeSize];
                        /*
                        int AttributeArray[] = {0,1,2,3,4,5,6,7};//,10,11,10,9,8,7,6,5,4,3};//new int[4];



                        AttributeArray[0]=0;
                        AttributeArray[1]=1;
                        AttributeArray[2]=2;
                        AttributeArray[3]=3;
                        */
```

```
                Integer intArray[] = new Integer[attributeSize];
                for (int i = 0; i < attributeSize; i++) {
                        intArray[i] = i;
                }
                int count = 0;
                // Shuffle the elements in the array
                for (int i = 0; i < 2 * newAttributeSize; i++) {
                        Collections.shuffle(Arrays.asList(intArray));
                        while ((count > 0) && (AttributeArray[count - 1] == intArray[0])) {
                                Collections.shuffle(Arrays.asList(intArray));
                                System.err.println("Collections.shuffle(Arrays.asList(intArray));");
                        }
                        for (int j = 0; j < attributeSize; j++) {
                                AttributeArray[count] = intArray[j];
                                count++;
                        }
                }

                return AttributeArray;
        }

/**
 * Returns description of the bagged classifier.
 *
 * @return                  description of the bagged classifier as a string
 */
public String toString() {
```

```
  // only ZeroR model?
  if (m_ZeroR != null) {
    StringBuffer buf = new StringBuffer();
    buf.append(this.getClass().getName().replaceAll(".*\\.", "") + "\n");
    buf.append(this.getClass().getName().replaceAll(".*\\.", "").replaceAll(".", "=") + "\n\n");
    buf.append("Warning: No model could be built, hence ZeroR model is used:\n\n");
    buf.append(m_ZeroR.toString());
    return buf.toString();
  }

  if (m_Classifiers == null) {
    return "ExtendedSpace2: No model built yet.";
  }
  StringBuffer text = new StringBuffer();
  text.append("All the base classifiers: \n\n");
  for (int i = 0; i < m_Classifiers.length; i++)
    text.append(m_Classifiers[i].toString() + "\n\n");

  return text.toString();
}

/**
 * Returns the revision string.
 *
 * @return                    the revision
 */
public String getRevision() {
  return RevisionUtils.extract("$Revision: 1.4 $");
```

```
  }


  /**
   * Main method for testing this class.
   *
   * @param args the options
   */
  public static void main(String[] args) {
    runClassifier(new ExtendedSpace(), args);
  }
}
```