ROC curves in WEKA ←

Visually estimating variance via null permutations

Comparing classifiers

An ROC curve is a efficient visualization to observe classification performance

The “big data” approach to classification problems

Is a voter democratic or republican?

Voters with known affiliation

Training set

Test set

Find features
(income, zip code, gender)
that predict affiliation

Build statistical model

Make predictions on test set

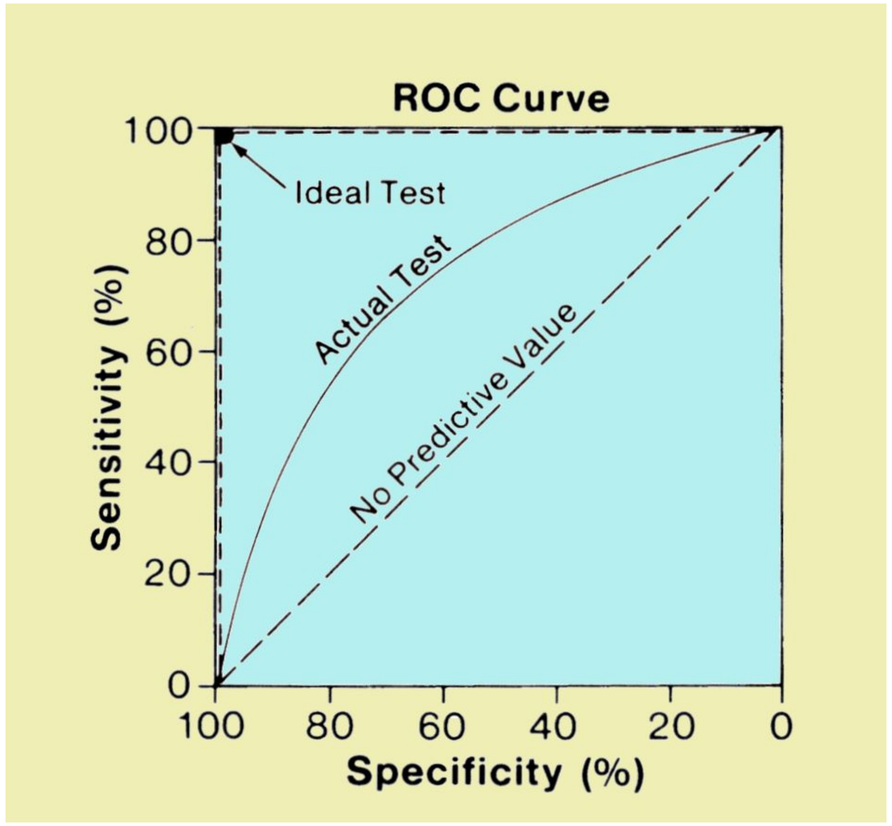


Image: http://www.sprawls.org/ppmi2/IMGCHAR/1IMCHAR12.gif

The “big data” approach to classification problems

Is a sample case or control?

Whole-genome metagenome sequencing
on samples with known disease status

Training set

Test set

Find features
(taxa, genes)
that predict status

Build statistical model

Make predictions on test set

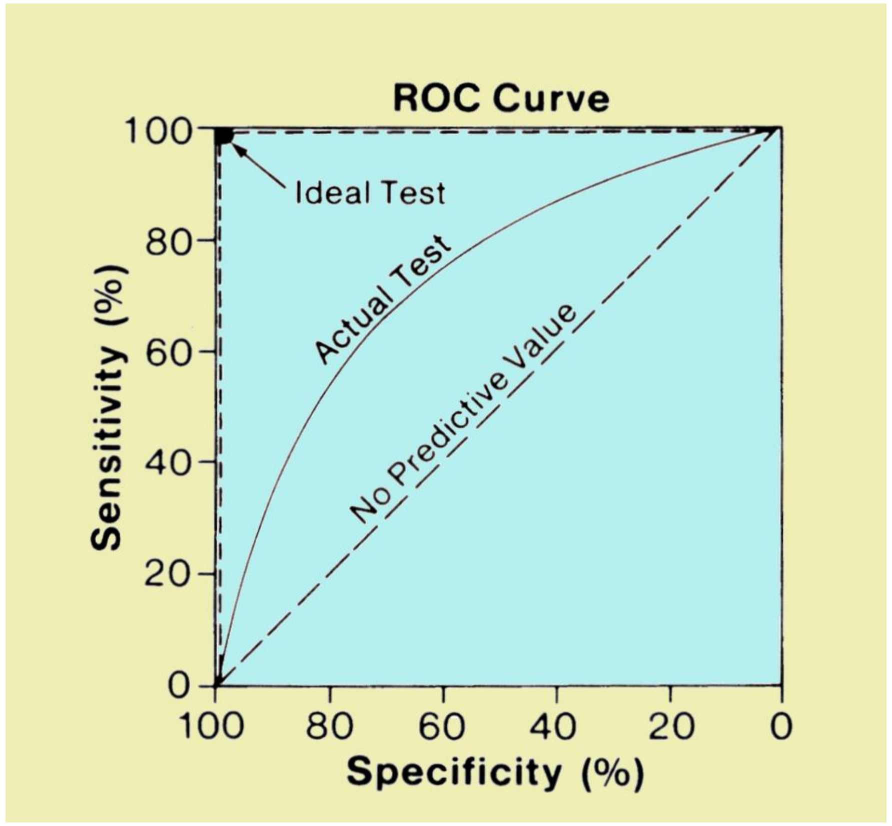
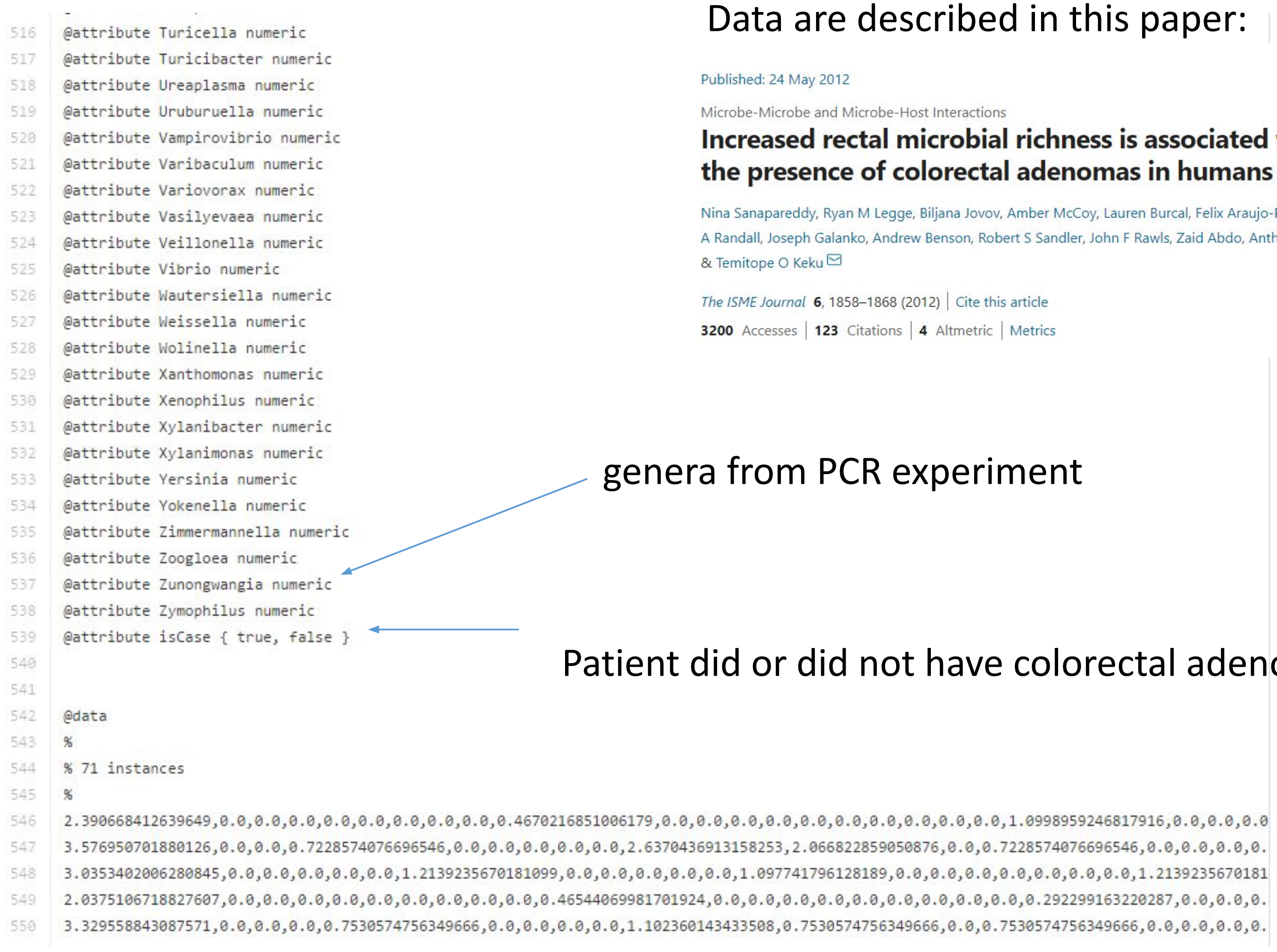


Image: http://www.sprawls.org/ppmi2/IMGCHAR/1IMCHAR12.gif

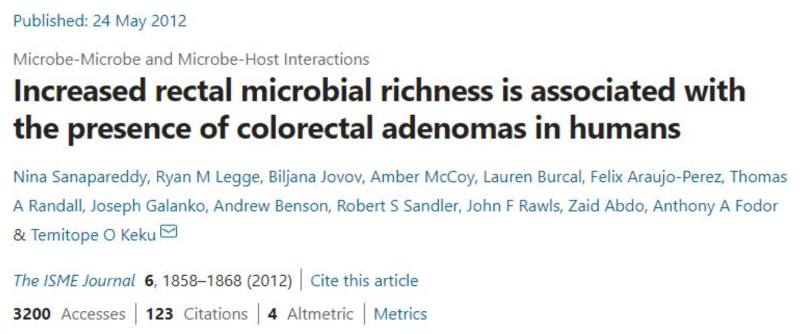
Our data file contains data from 71 patients…

https://github.com/afodor/afodor.github.io/blob/master/classes/prog2016/pivoted_genusLogNormalWithMetadata.arff



```
@attribute Turicella numeric
@attribute Turicibacter numeric
@attribute Ureaplasma numeric
@attribute Uruburuella numeric
@attribute Vampirovibrio numeric
@attribute Varibaculum numeric
@attribute Variovorax numeric
@attribute Vasilyevaea numeric
@attribute Veillonella numeric
@attribute Vibrio numeric
@attribute Wautersiella numeric
@attribute Weissella numeric
@attribute Wolinella numeric
@attribute Xanthomonas numeric
@attribute Xenophilus numeric
@attribute Xylanibacter numeric
@attribute Xylanimonas numeric
@attribute Yersinia numeric
@attribute Yokenella numeric
@attribute Zimmermannella numeric
@attribute Zoogloea numeric
@attribute Zunongwangia numeric
@attribute Zymophilus numeric
@attribute isCase { true, false }


@data
%
% 71 instances
%
2.390668412639649,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.4670216851006179,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,1.0998959246817916,0.0,0.0,0.0,0.0
3.576950701880126,0.0,0.0,0.7228574076696546,0.0,0.0,0.0,0.0,0.0,2.6370436913158253,2.066822859050876,0.0,0.7228574076696546,0.0,0.0,0.0,0.0,0.
3.0353402006280845,0.0,0.0,0.0,0.0,0.0,0.0,1.2139235670181099,0.0,0.0,0.0,0.0,0.0,0.0,1.097741796128189,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,1.2139235670181
2.0375106718827607,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.46544069981701924,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.292299163220287,0.0,0.0,0.0,0.
3.329558843087571,0.0,0.0,0.0,0.0,0.7530574756349666,0.0,0.0,0.0,0.0,0.0,1.102360143433508,0.7530574756349666,0.0,0.7530574756349666,0.0,0.0,0.0,0.0,0.
```

Data are described in this paper:



Published: 24 May 2012

Microbe-Microbe and Microbe-Host Interactions

Increased rectal microbial richness is associated with the presence of colorectal adenomas in humans

Nina Sanapareddy, Ryan M Legge, Biljana Jovov, Amber McCoy, Lauren Burcal, Felix Araujo-Perez, Thomas A Randall, Joseph Galanko, Andrew Benson, Robert S Sandler, John F Rawls, Zaid Abdo, Anthony A Fodor & Temitope O Keku

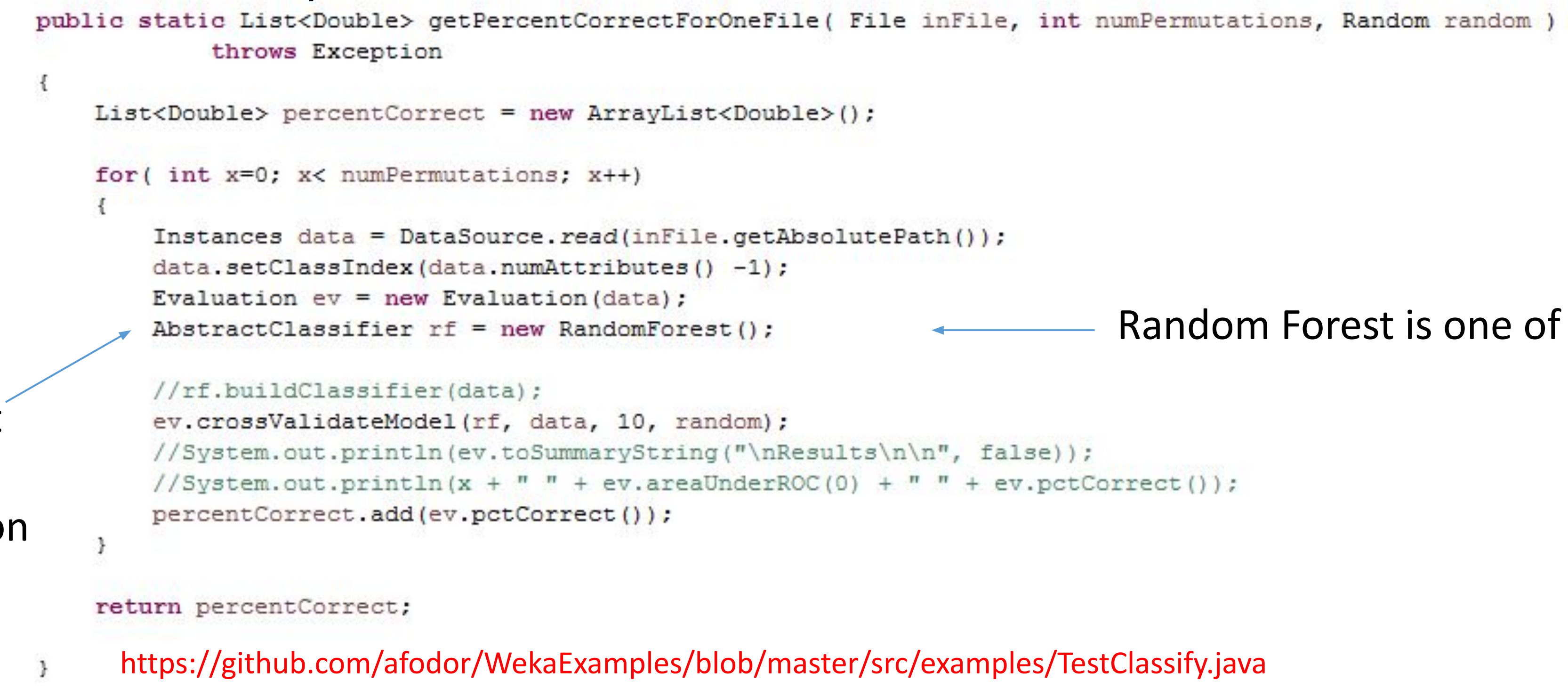
The ISME Journal **6**, 1858–1868 (2012) | Cite this article

3200 Accesses | **123** Citations | **4** Altmetric | Metrics

genera from PCR experiment

Patient did or did not have colorectal adenomas

This code performs 10-fold cross validation…



```java
public static List<Double> getPercentCorrectForOneFile( File inFile, int numPermutations, Random random )
            throws Exception
{
    List<Double> percentCorrect = new ArrayList<Double>();

    for( int x=0; x< numPermutations; x++)
    {
        Instances data = DataSource.read(inFile.getAbsolutePath());
        data.setClassIndex(data.numAttributes() -1);
        Evaluation ev = new Evaluation(data);
        AbstractClassifier rf = new RandomForest();

        //rf.buildClassifier(data);
        ev.crossValidateModel(rf, data, 10, random);
        //System.out.println(ev.toSummaryString("\nResults\n\n", false));
        //System.out.println(x + " " + ev.areaUnderROC(0) + " " + ev.pctCorrect());
        percentCorrect.add(ev.pctCorrect());
    }

    return percentCorrect;

}
```

Random Forest is one of many classifiers

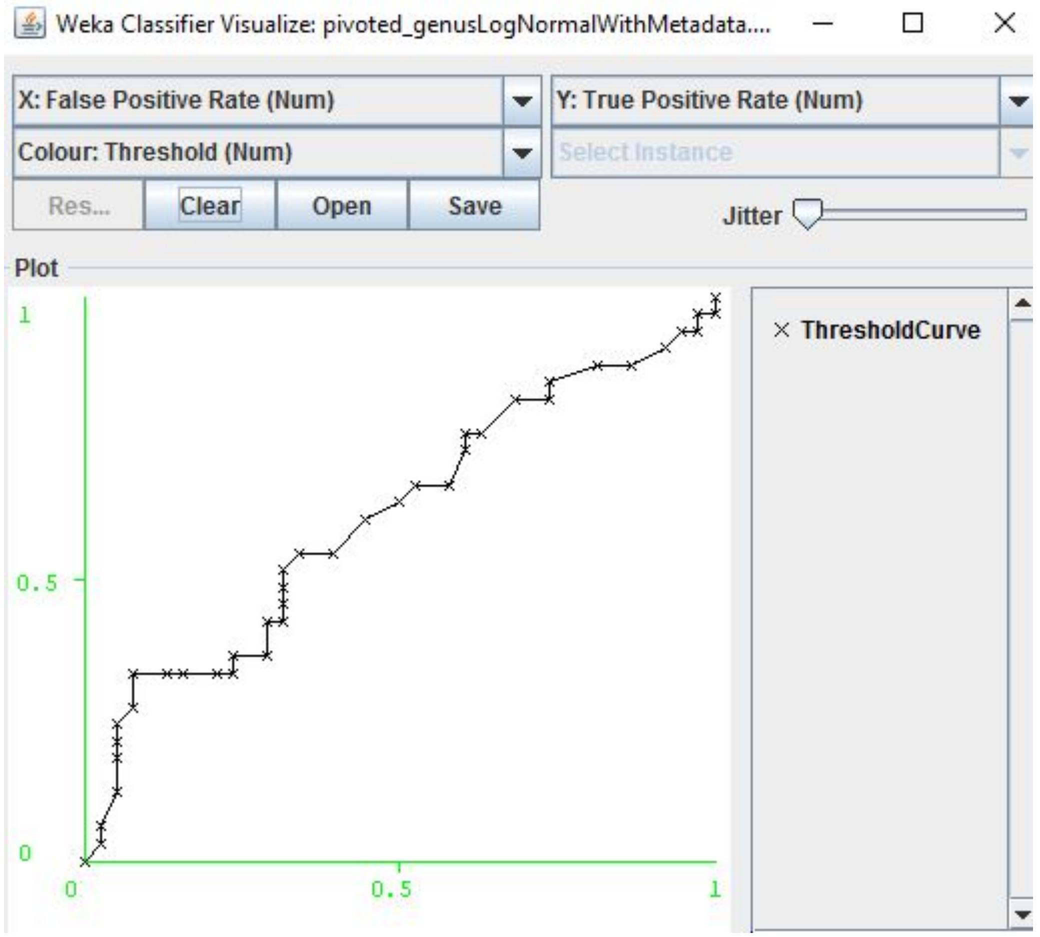
This is a great
Example of
OO abstraction

https://github.com/afodor/WekaExamples/blob/master/src/examples/TestClassify.java

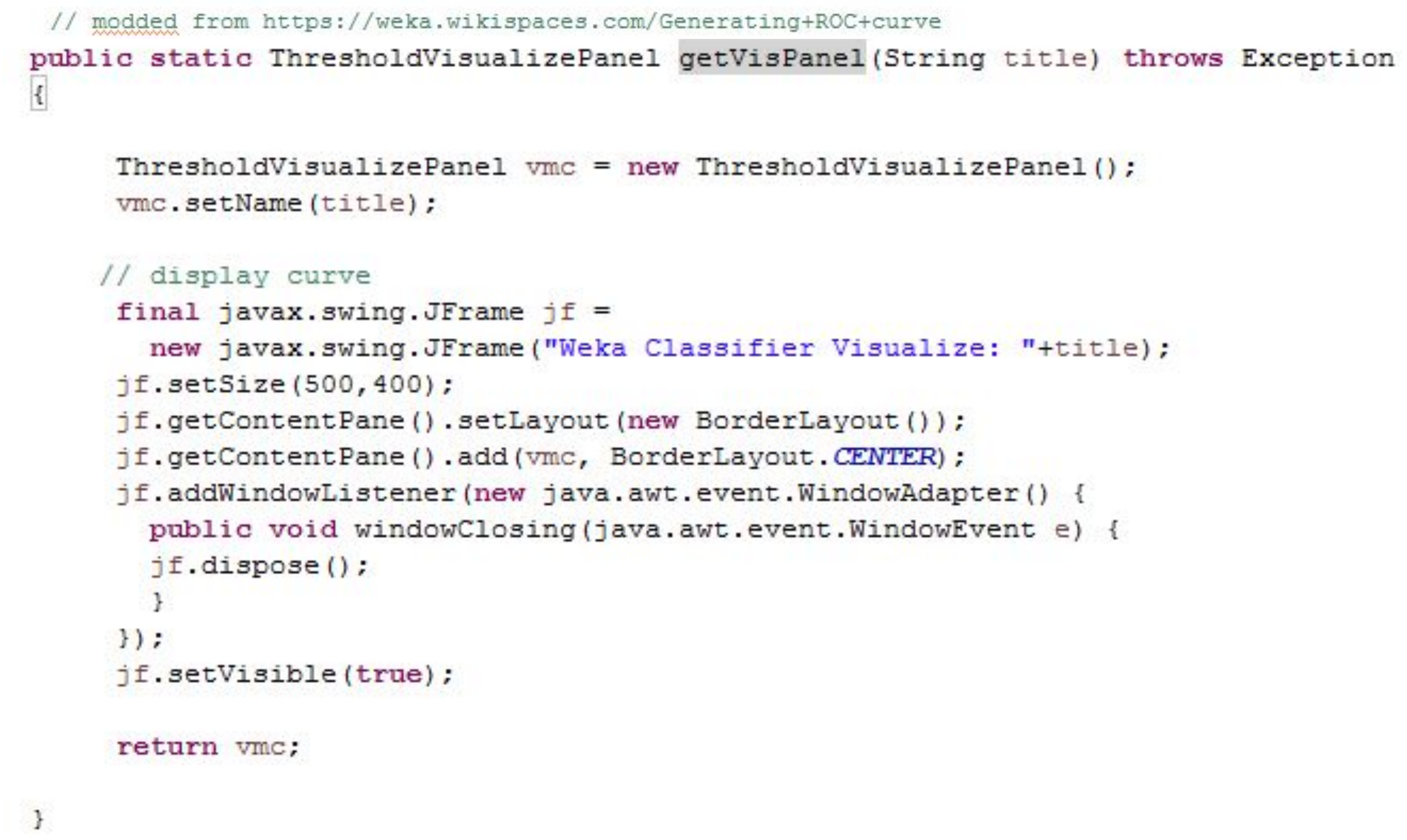
We can easily visualize the ROC code associated with this ten-fold cross validation

```
public static void main(String[] args) throws Exception
{
    Random random = new Random();
    // this file is at
    //https://github.com/afodor/afodor.github.io/blob/master/classes/prog2016/pivoted_genusLogNormalWithMetadata.arff
    File inArff= new File(
            "C:\\Users\\corei7\\git\\afodor.github.io\\classes\\prog2016\\pivoted_genusLogNormalWithMetadata.arff");

    ThresholdVisualizePanel tvp = TestClassify.getVisPanel(inArff.getName());

    TestClassify.plotROCForAnArff(inArff, 1,random,false,tvp);
}
```

https://github.com/afodor/WekaExamples/blob/master/src/examples/RunOneROCCurve.java


Weka Classifier Visualize: pivoted_genusLogNormalWithMetadata....
X: False Positive Rate (Num)
Y: True Positive Rate (Num)
Colour: Threshold (Num)
Select Instance
Res...
Clear
Open
Save
Jitter
Plot
1
0.5
0
0
0.5
1
ThresholdCurve

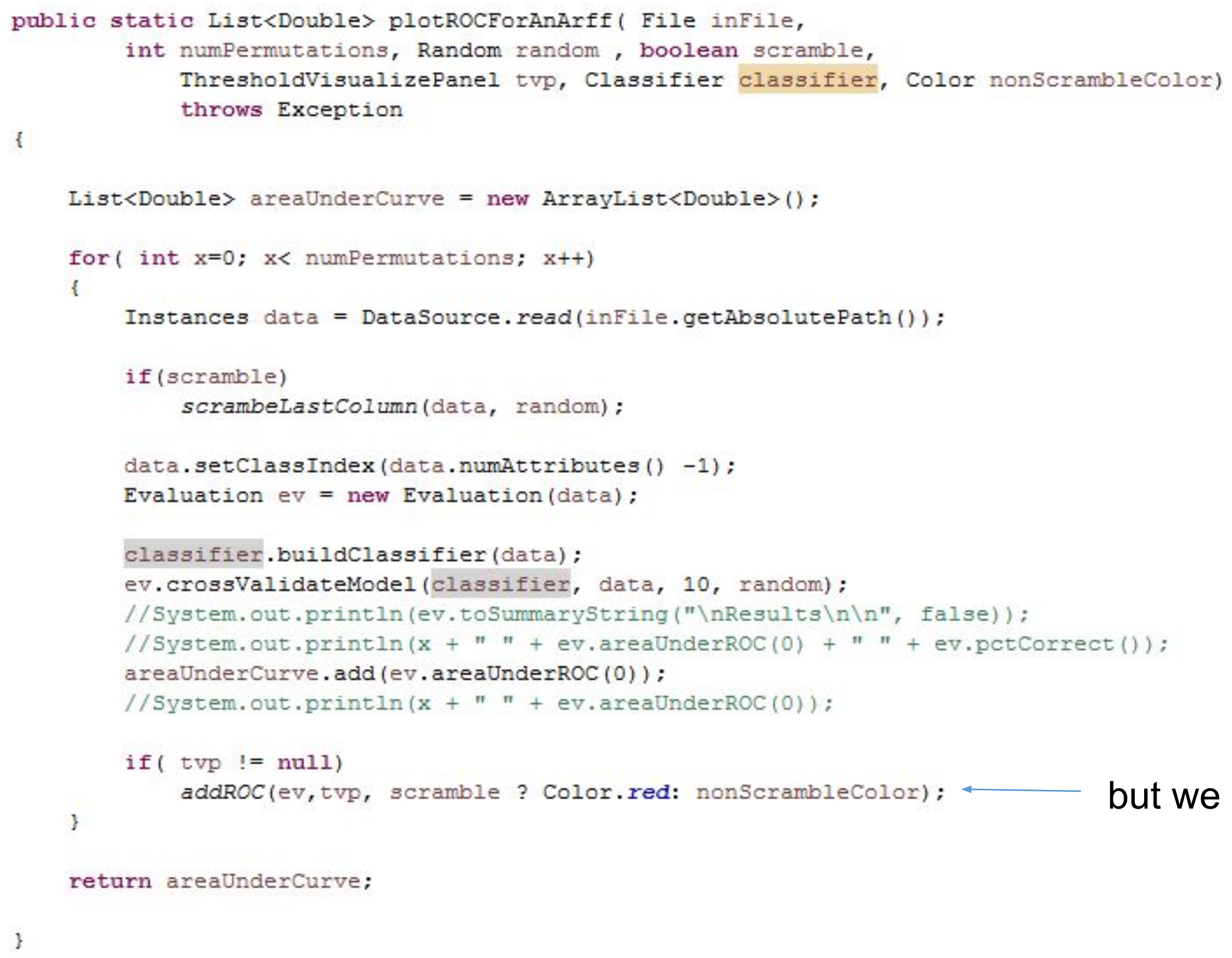
Our initial method just creates a “ThresholdVisualizePanel” and returns it…
(this won’t make a lot of sense till we get to GUIs later..)



```
 // modded from https://weka.wikispaces.com/Generating+ROC+curve
public static ThresholdVisualizePanel getVisPanel(String title) throws Exception
{

     ThresholdVisualizePanel vmc = new ThresholdVisualizePanel();
     vmc.setName(title);

    // display curve
     final javax.swing.JFrame jf =
       new javax.swing.JFrame("Weka Classifier Visualize: "+title);
     jf.setSize(500,400);
     jf.getContentPane().setLayout(new BorderLayout());
     jf.getContentPane().add(vmc, BorderLayout.CENTER);
     jf.addWindowListener(new java.awt.event.WindowAdapter() {
       public void windowClosing(java.awt.event.WindowEvent e) {
       jf.dispose();
       }
     });
     jf.setVisible(true);

     return vmc;

}
```

https://github.com/afodor/WekaExamples/blob/master/src/examples/TestClassify.java

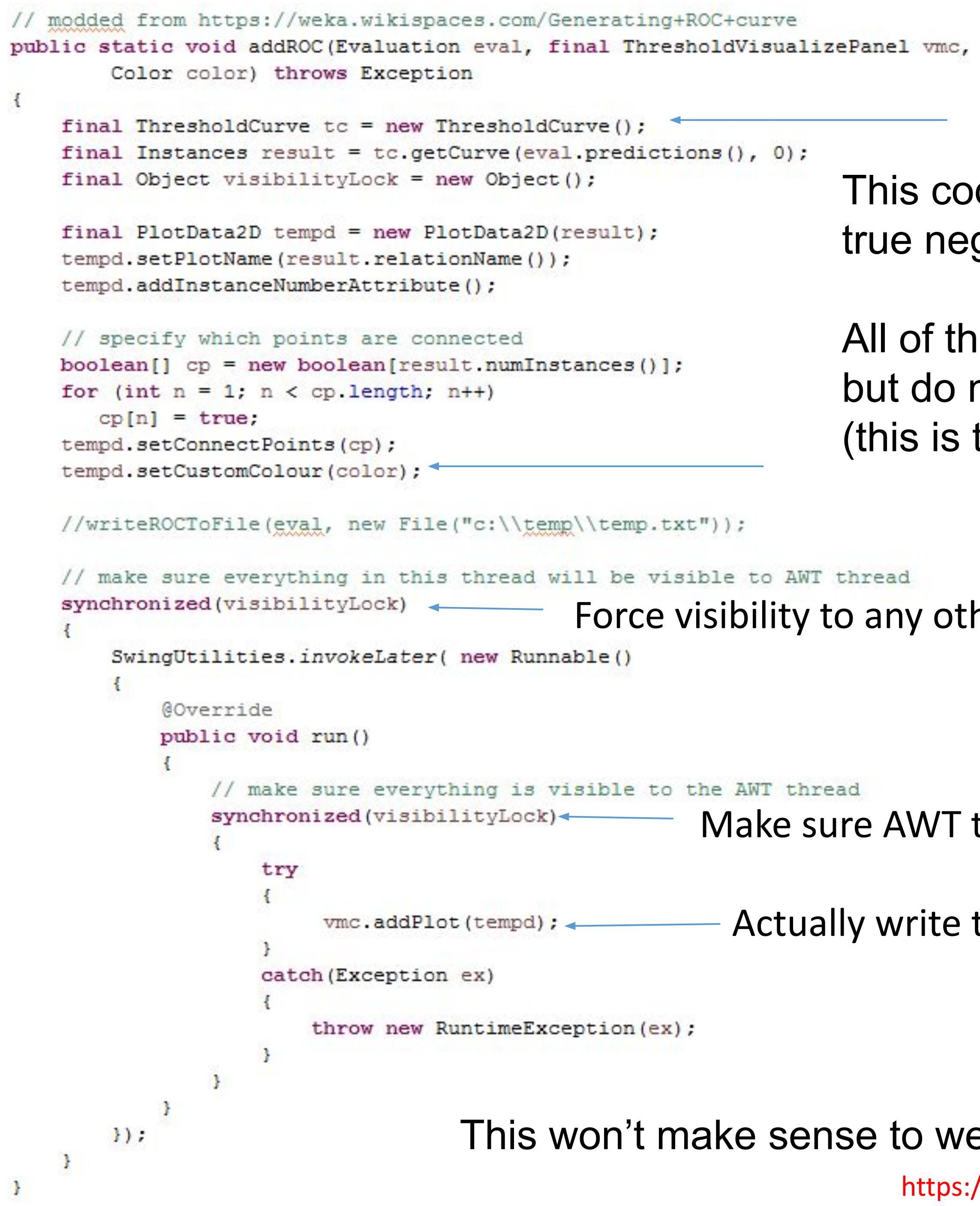
This method proceeds as before, building a classifier and testing it via ten-fold cross-validation



```java
public static List<Double> plotROCForAnArff( File inFile,
        int numPermutations, Random random , boolean scramble,
            ThresholdVisualizePanel tvp, Classifier classifier, Color nonScrambleColor)
            throws Exception
{

    List<Double> areaUnderCurve = new ArrayList<Double>();

    for( int x=0; x< numPermutations; x++)
    {
        Instances data = DataSource.read(inFile.getAbsolutePath());

        if(scramble)
            scrambeLastColumn(data, random);

        data.setClassIndex(data.numAttributes() -1);
        Evaluation ev = new Evaluation(data);

        classifier.buildClassifier(data);
        ev.crossValidateModel(classifier, data, 10, random);
        //System.out.println(ev.toSummaryString("\nResults\n\n", false));
        //System.out.println(x + " " + ev.areaUnderROC(0) + " " + ev.pctCorrect());
        areaUnderCurve.add(ev.areaUnderROC(0));
        //System.out.println(x + " " + ev.areaUnderROC(0));

        if( tvp != null)
            addROC(ev,tvp, scramble ? Color.red: nonScrambleColor);
    }

    return areaUnderCurve;

}
```

but we add a line for visualization

https://github.com/afodor/WekaExamples/blob/master/src/examples/TestClassify.java



```java
// modded from https://weka.wikispaces.com/Generating+ROC+curve
public static void addROC(Evaluation eval, final ThresholdVisualizePanel vmc,
        Color color) throws Exception
{
    final ThresholdCurve tc = new ThresholdCurve();
    final Instances result = tc.getCurve(eval.predictions(), 0);
    final Object visibilityLock = new Object();

    final PlotData2D tempd = new PlotData2D(result);
    tempd.setPlotName(result.relationName());
    tempd.addInstanceNumberAttribute();

    // specify which points are connected
    boolean[] cp = new boolean[result.numInstances()];
    for (int n = 1; n < cp.length; n++)
       cp[n] = true;
    tempd.setConnectPoints(cp);
    tempd.setCustomColour(color);

    //writeROCToFile(eval, new File("c:\\temp\\temp.txt"));

    // make sure everything in this thread will be visible to AWT thread
    synchronized(visibilityLock)
    {
        SwingUtilities.invokeLater( new Runnable()
        {
            @Override
            public void run()
            {
                // make sure everything is visible to the AWT thread
                synchronized(visibilityLock)
                {
                    try
                    {
                        vmc.addPlot(tempd);
                    }
                    catch(Exception ex)
                    {
                        throw new RuntimeException(ex);
                    }
                }
            }
        });
    }
}
```

This code does the calculations of true positive and true negative

All of the helper classes are stack confined, but do not make visibility guarantees (this is thread stuff we will talk about later)

Force visibility to any other thread that grabs this lock

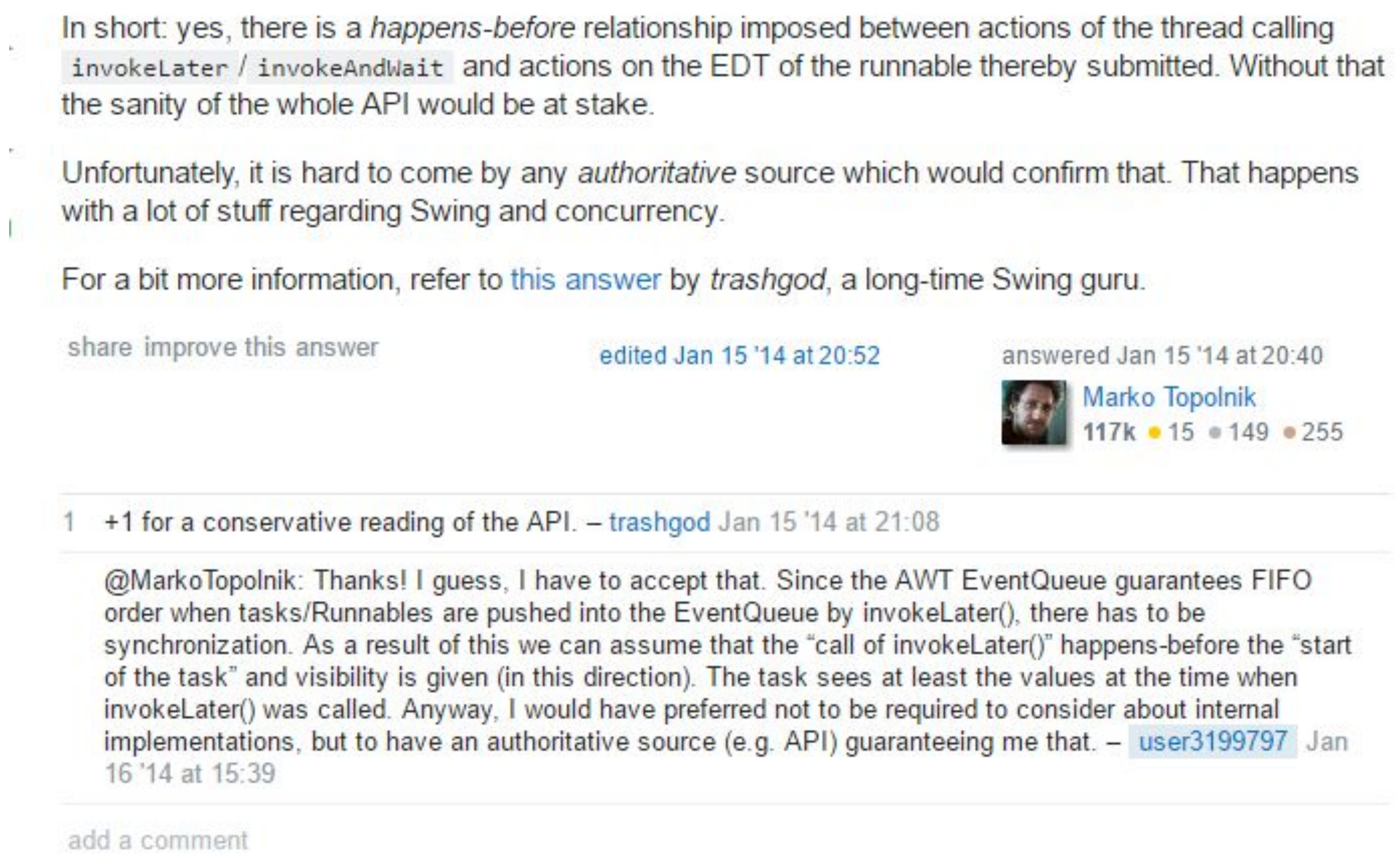
Make sure AWT thread has full visibility

Actually write the plot; should happen on AWT thread

This won't make sense to we talk about GUIs and multi-threaded code later

https://github.com/afodor/WekaExamples/blob/master/src/examples/TestClassify.java

Do I need to manually force the visibility?
Probably not, but the documentation is a bit unclear, so better safe than sorry.
(It is a very small performance hit to grab the lock twice….)



In short: yes, there is a *happens-before* relationship imposed between actions of the thread calling `invokeLater` / `invokeAndWait` and actions on the EDT of the runnable thereby submitted. Without that the sanity of the whole API would be at stake.

Unfortunately, it is hard to come by any *authoritative* source which would confirm that. That happens with a lot of stuff regarding Swing and concurrency.

For a bit more information, refer to this answer by *trashgod*, a long-time Swing guru.

share improve this answer   edited Jan 15 '14 at 20:52   answered Jan 15 '14 at 20:40
Marko Topolnik
117k ● 15 ● 149 ● 255

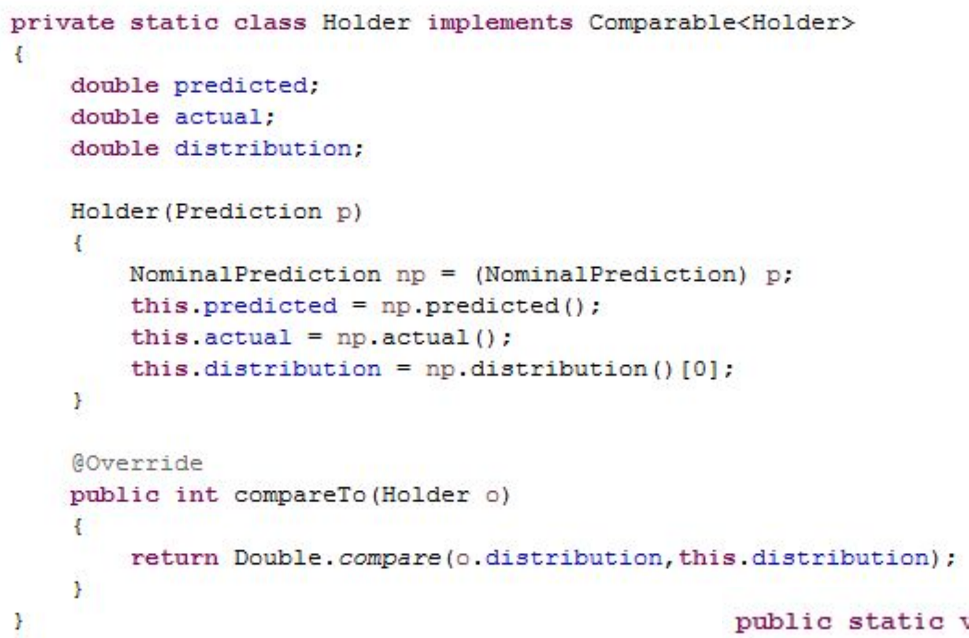
1 +1 for a conservative reading of the API. – trashgod Jan 15 '14 at 21:08

@MarkoTopolnik: Thanks! I guess, I have to accept that. Since the AWT EventQueue guarantees FIFO order when tasks/Runnables are pushed into the EventQueue by invokeLater(), there has to be synchronization. As a result of this we can assume that the "call of invokeLater()" happens-before the "start of the task" and visibility is given (in this direction). The task sees at least the values at the time when invokeLater() was called. Anyway, I would have preferred not to be required to consider about internal implementations, but to have an authoritative source (e.g. API) guaranteeing me that. – user3199797 Jan 16 '14 at 15:39

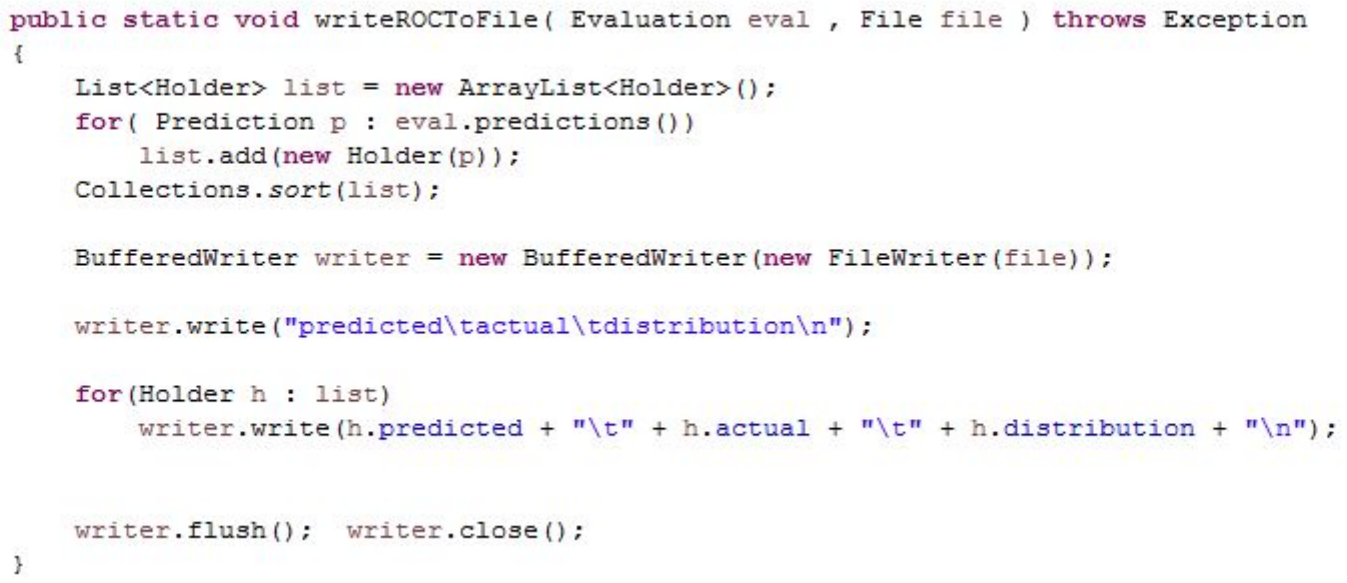
add a comment

http://stackoverflow.com/questions/21147240/is-happens-before-relation-given-in-case-of-invokelater-or-invokeandwait

We can easily write out the ROC data to a file if we want to use an alternative visualization or analysis platform….



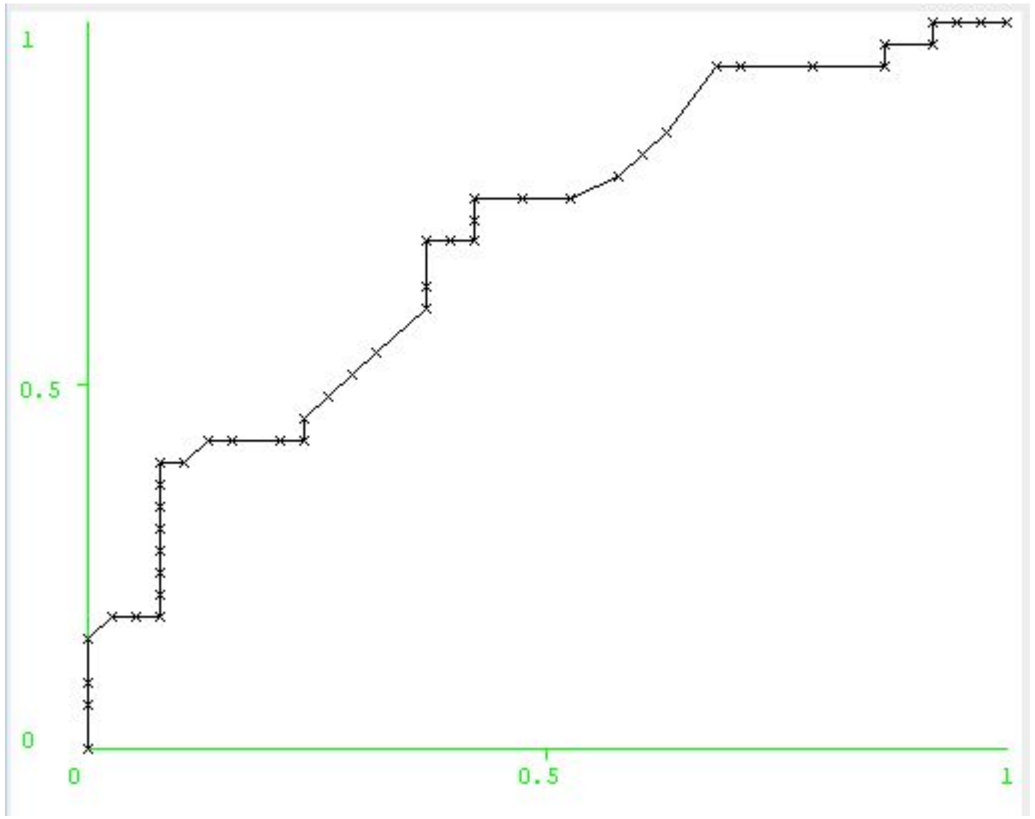
```java
private static class Holder implements Comparable<Holder>
{
    double predicted;
    double actual;
    double distribution;

    Holder(Prediction p)
    {
        NominalPrediction np = (NominalPrediction) p;
        this.predicted = np.predicted();
        this.actual = np.actual();
        this.distribution = np.distribution()[0];
    }

    @Override
    public int compareTo(Holder o)
    {
        return Double.compare(o.distribution,this.distribution);
    }
}
```

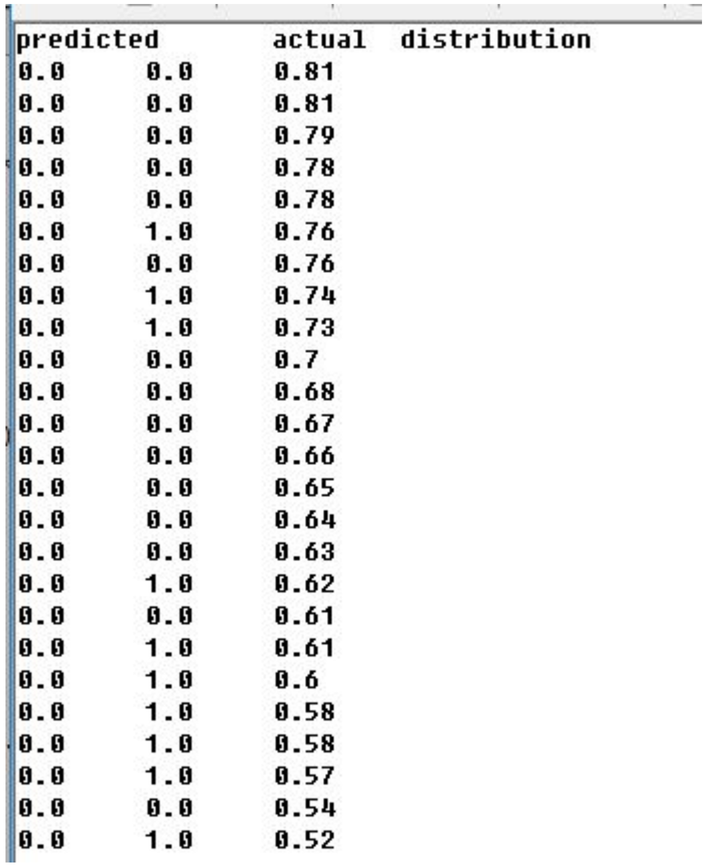


```java
public static void writeROCToFile( Evaluation eval , File file ) throws Exception
{
    List<Holder> list = new ArrayList<Holder>();
    for( Prediction p : eval.predictions())
        list.add(new Holder(p));
    Collections.sort(list);

    BufferedWriter writer = new BufferedWriter(new FileWriter(file));

    writer.write("predicted\tactual\tdistribution\n");

    for(Holder h : list)
        writer.write(h.predicted + "\t" + h.actual + "\t" + h.distribution + "\n");


    writer.flush();  writer.close();
}
```

https://github.com/afodor/WekaExamples/blob/master/src/examples/TestClassify.java

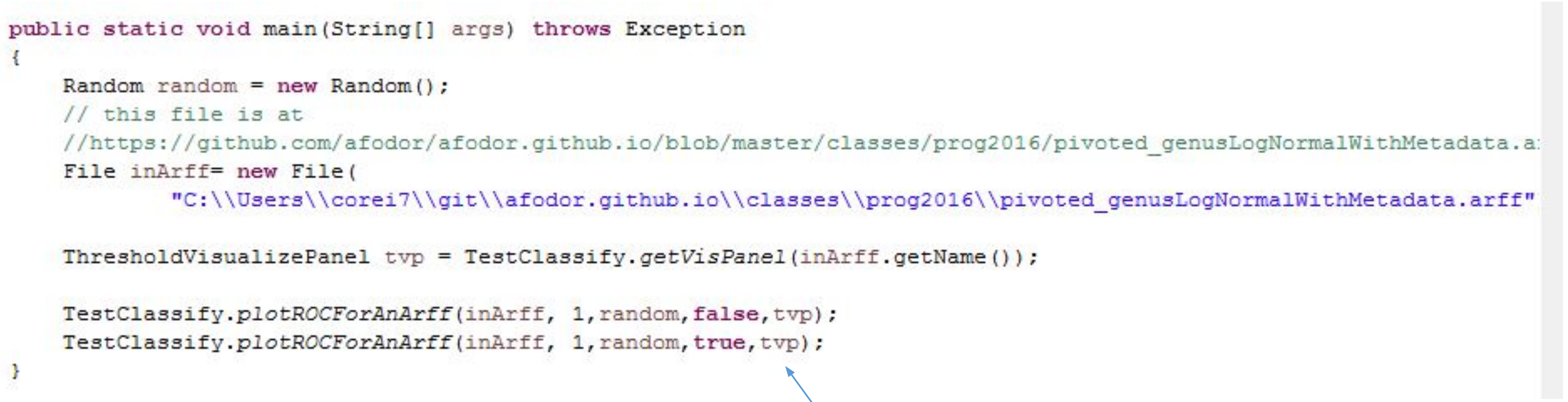
The “distribution” variable shows how much confidence the predictor has in each prediction and is used to order the ROC curve





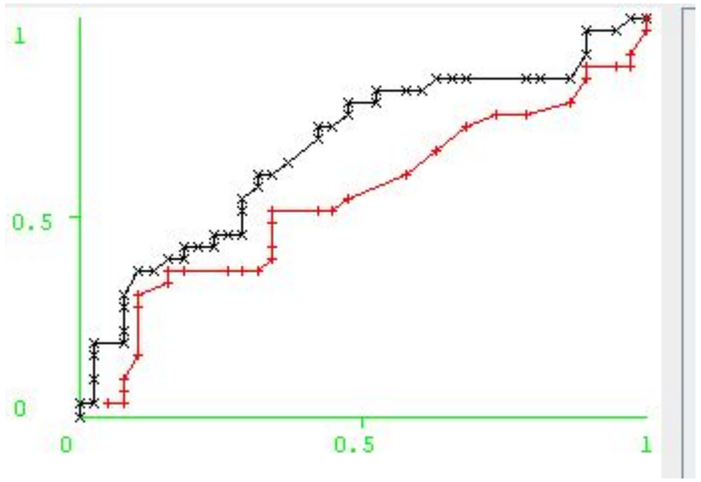
```
predicted         actual   distribution
0.0       0.0       0.81
0.0       0.0       0.81
0.0       0.0       0.79
0.0       0.0       0.78
0.0       0.0       0.78
0.0       1.0       0.76
0.0       0.0       0.76
0.0       1.0       0.74
0.0       1.0       0.73
0.0       0.0       0.7
0.0       0.0       0.68
0.0       0.0       0.67
0.0       0.0       0.66
0.0       0.0       0.65
0.0       0.0       0.64
0.0       0.0       0.63
0.0       1.0       0.62
0.0       0.0       0.61
0.0       1.0       0.61
0.0       1.0       0.6
0.0       1.0       0.58
0.0       1.0       0.58
0.0       1.0       0.57
0.0       0.0       0.54
0.0       1.0       0.52
```

ROC curves in WEKA
Visually estimating variance via null permutations ←
Comparing classifiers

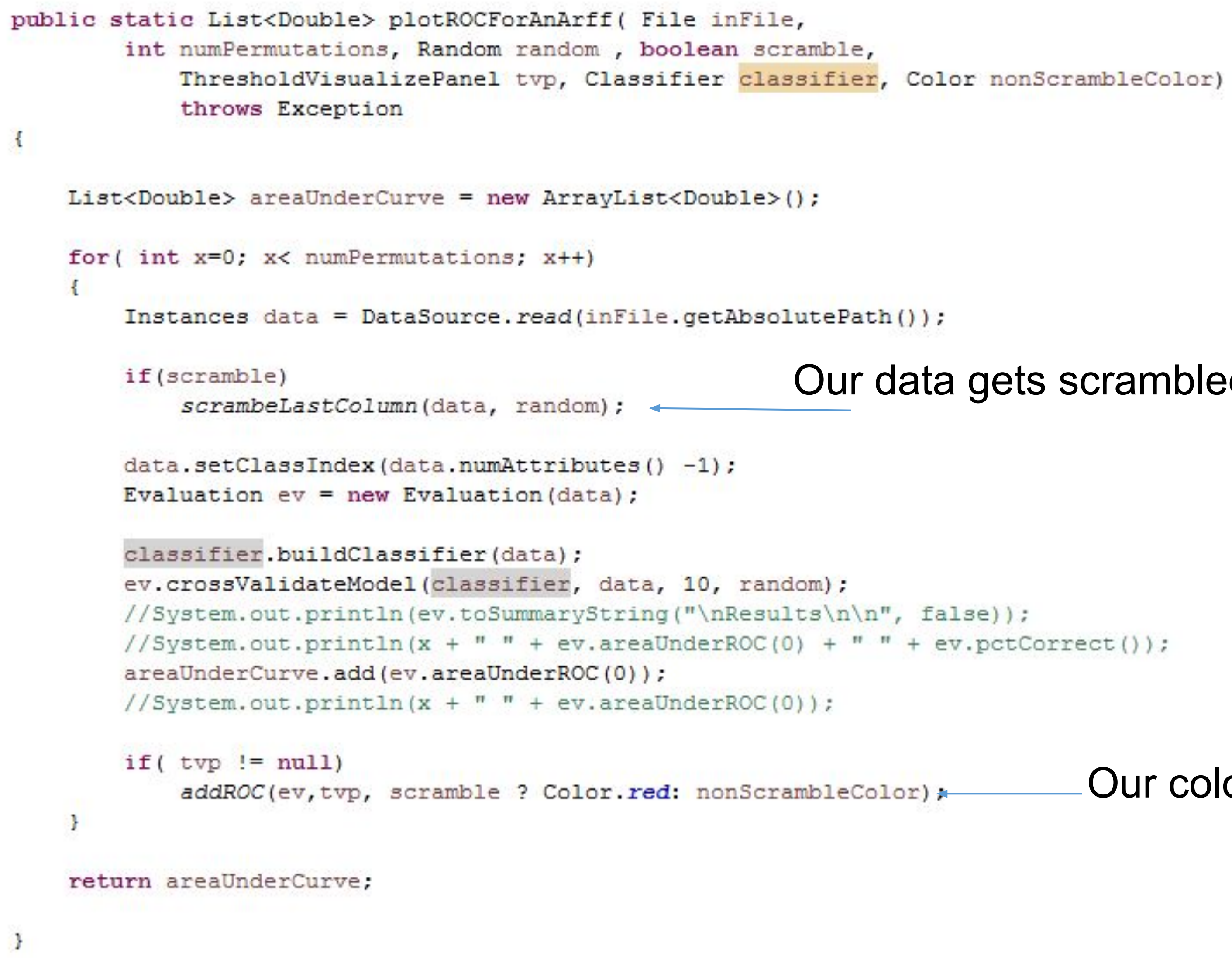


```
public static void main(String[] args) throws Exception
{
    Random random = new Random();
    // this file is at
    //https://github.com/afodor/afodor.github.io/blob/master/classes/prog2016/pivoted_genusLogNormalWithMetadata.a
    File inArff= new File(
            "C:\\Users\\corei7\\git\\afodor.github.io\\classes\\prog2016\\pivoted_genusLogNormalWithMetadata.arff"

    ThresholdVisualizePanel tvp = TestClassify.getVisPanel(inArff.getName());

    TestClassify.plotROCForAnArff(inArff, 1,random,false,tvp);
    TestClassify.plotROCForAnArff(inArff, 1,random,true,tvp);
}
```

We can compare our "true" classification to
a classification with the case/control labels scrambled



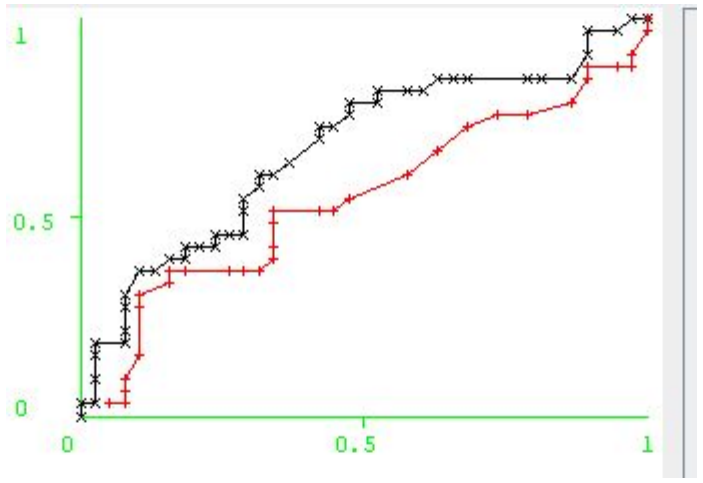
https://github.com/afodor/WekaExamples/blob/master/src/examples/RunOneROCCurve.java



```
public static List<Double> plotROCForAnArff( File inFile,
        int numPermutations, Random random , boolean scramble,
            ThresholdVisualizePanel tvp, Classifier classifier, Color nonScrambleColor)
            throws Exception
{

    List<Double> areaUnderCurve = new ArrayList<Double>();

    for( int x=0; x< numPermutations; x++)
    {
        Instances data = DataSource.read(inFile.getAbsolutePath());

        if(scramble)
            scrambeLastColumn(data, random);

        data.setClassIndex(data.numAttributes() -1);
        Evaluation ev = new Evaluation(data);

        classifier.buildClassifier(data);
        ev.crossValidateModel(classifier, data, 10, random);
        //System.out.println(ev.toSummaryString("\nResults\n\n", false));
        //System.out.println(x + " " + ev.areaUnderROC(0) + " " + ev.pctCorrect());
        areaUnderCurve.add(ev.areaUnderROC(0));
        //System.out.println(x + " " + ev.areaUnderROC(0));

        if( tvp != null)
            addROC(ev,tvp, scramble ? Color.red: nonScrambleColor);
    }

    return areaUnderCurve;

}
```

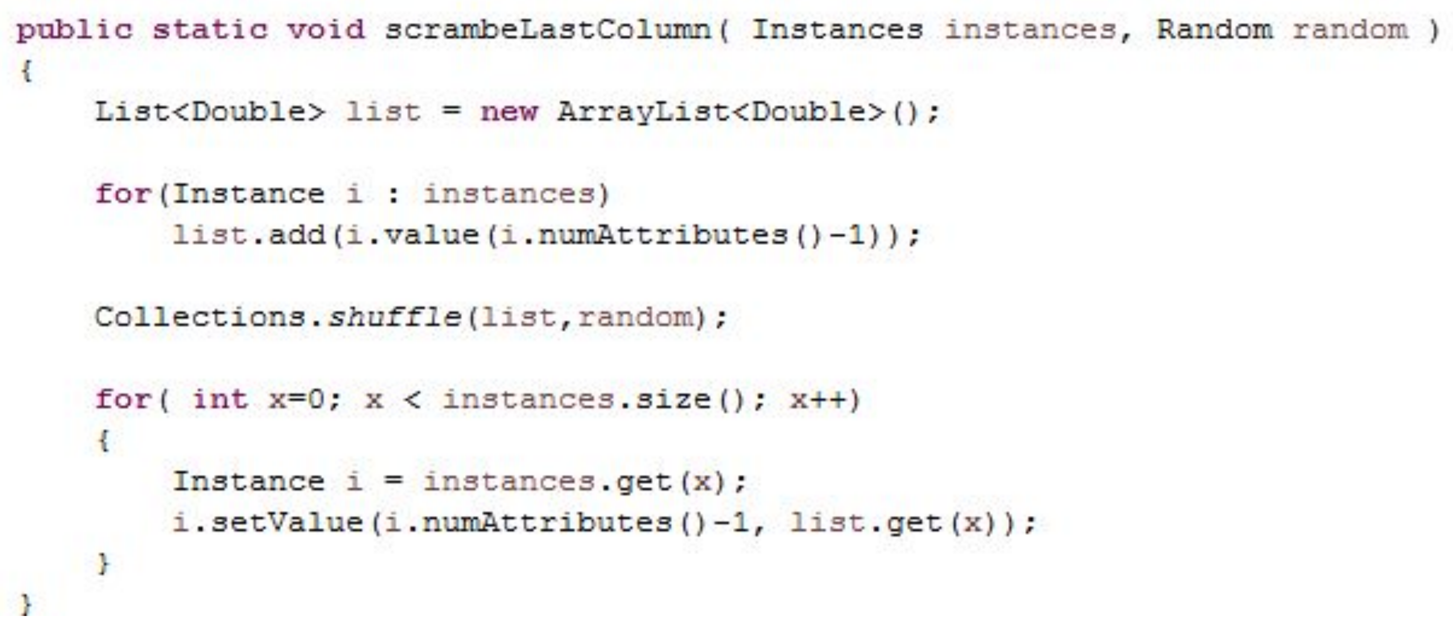
Our data gets scrambled

Our color turns red



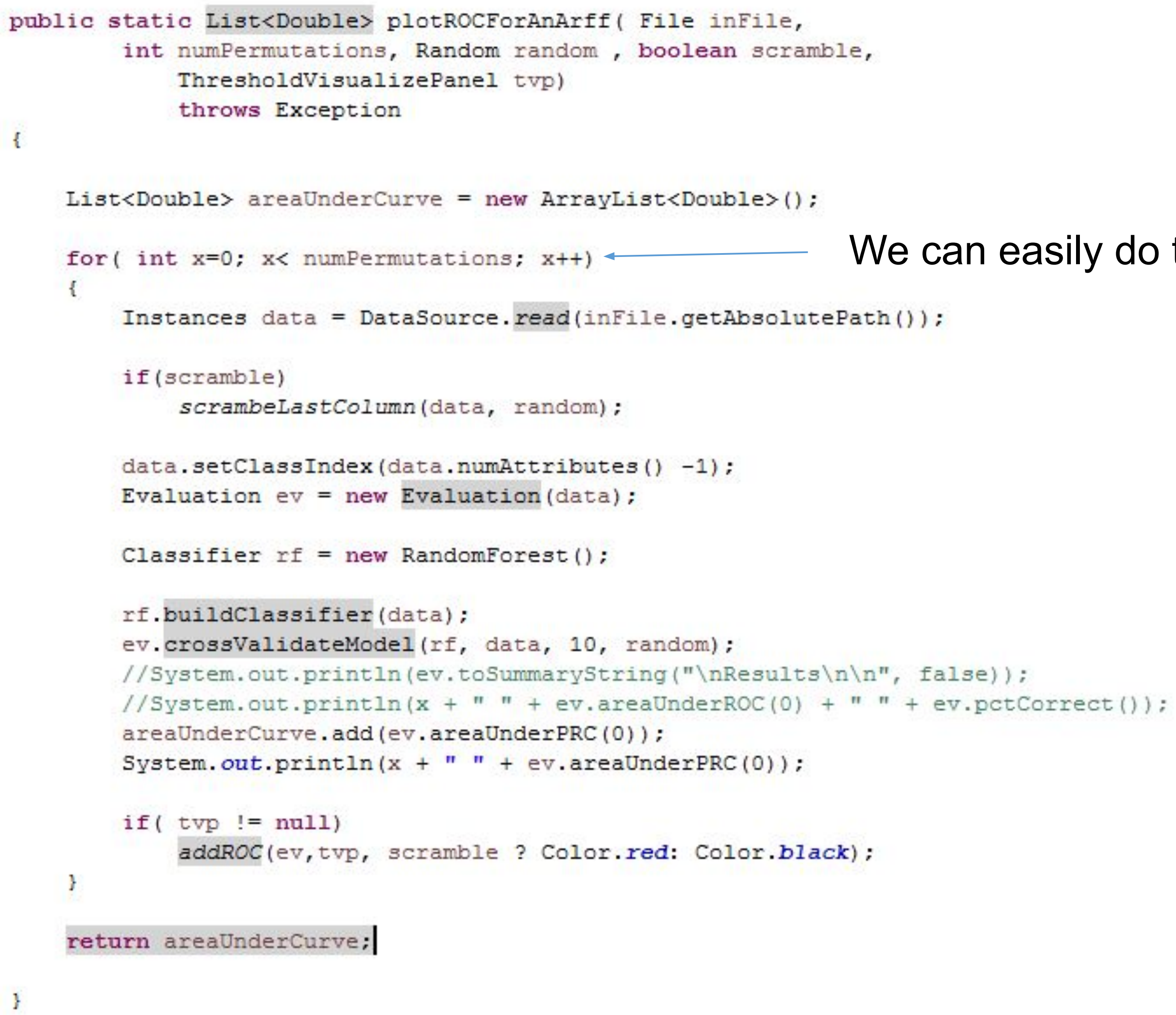
https://github.com/afodor/WekaExamples/blob/master/src/examples/TestClassify.java

Scrambling the columns is trivial…



```java
public static void scrambeLastColumn( Instances instances, Random random )
{
    List<Double> list = new ArrayList<Double>();

    for(Instance i : instances)
        list.add(i.value(i.numAttributes()-1));

    Collections.shuffle(list,random);

    for( int x=0; x < instances.size(); x++)
    {
        Instance i = instances.get(x);
        i.setValue(i.numAttributes()-1, list.get(x));
    }
}
```

https://github.com/afodor/WekaExamples/blob/master/src/examples/TestClassify.java

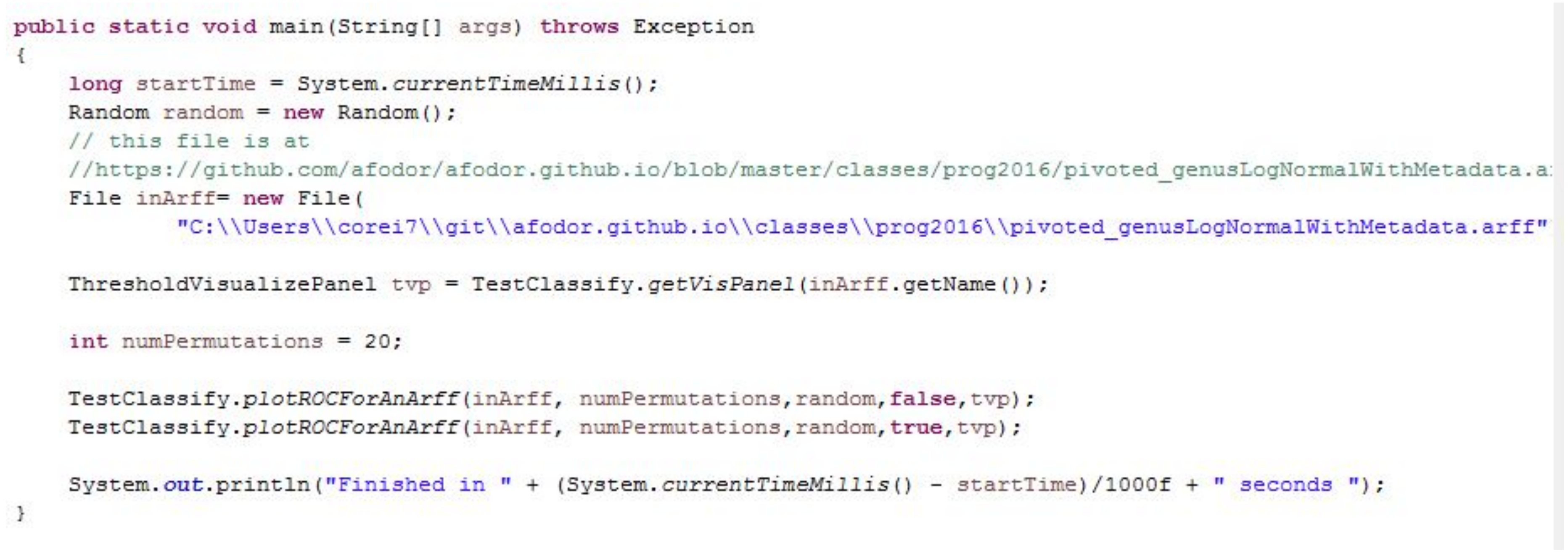


```
public static List<Double> plotROCForAnArff( File inFile,
        int numPermutations, Random random , boolean scramble,
            ThresholdVisualizePanel tvp)
            throws Exception
{

    List<Double> areaUnderCurve = new ArrayList<Double>();

    for( int x=0; x< numPermutations; x++)
    {
        Instances data = DataSource.read(inFile.getAbsolutePath());

        if(scramble)
            scrambeLastColumn(data, random);

        data.setClassIndex(data.numAttributes() -1);
        Evaluation ev = new Evaluation(data);

        Classifier rf = new RandomForest();

        rf.buildClassifier(data);
        ev.crossValidateModel(rf, data, 10, random);
        //System.out.println(ev.toSummaryString("\nResults\n\n", false));
        //System.out.println(x + " " + ev.areaUnderROC(0) + " " + ev.pctCorrect());
        areaUnderCurve.add(ev.areaUnderPRC(0));
        System.out.println(x + " " + ev.areaUnderPRC(0));

        if( tvp != null)
            addROC(ev,tvp, scramble ? Color.red: Color.black);
    }

    return areaUnderCurve;

}
```

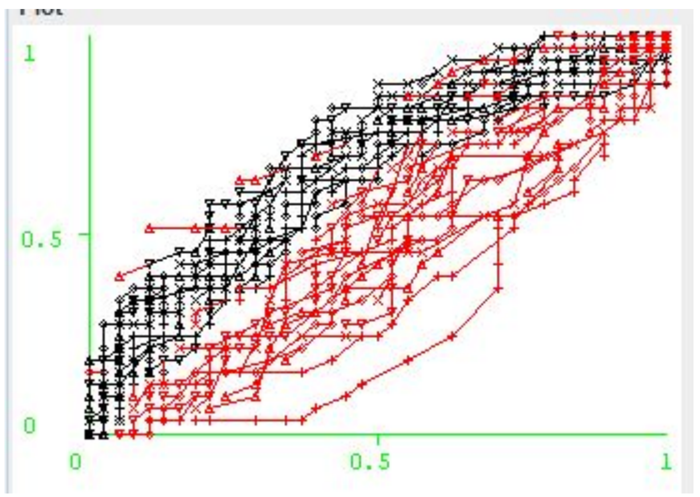
We can easily do this for multiple permutations…

https://github.com/afodor/WekaExamples/blob/master/src/examples/TestClassify.java

This makes a very compelling and informative visualization (here for 20 permutations)



```
public static void main(String[] args) throws Exception
{
    long startTime = System.currentTimeMillis();
    Random random = new Random();
    // this file is at
    //https://github.com/afodor/afodor.github.io/blob/master/classes/prog2016/pivoted_genusLogNormalWithMetadata.a
    File inArff= new File(
            "C:\\Users\\corei7\\git\\afodor.github.io\\classes\\prog2016\\pivoted_genusLogNormalWithMetadata.arff"

    ThresholdVisualizePanel tvp = TestClassify.getVisPanel(inArff.getName());

    int numPermutations = 20;

    TestClassify.plotROCForAnArff(inArff, numPermutations,random,false,tvp);
    TestClassify.plotROCForAnArff(inArff, numPermutations,random,true,tvp);

    System.out.println("Finished in " + (System.currentTimeMillis() - startTime)/1000f + " seconds ");
}
```



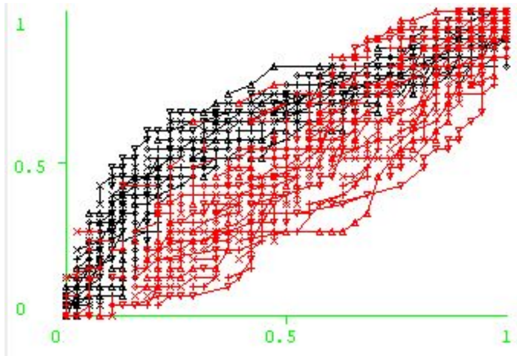
```
Finished in 99.502 seconds
```

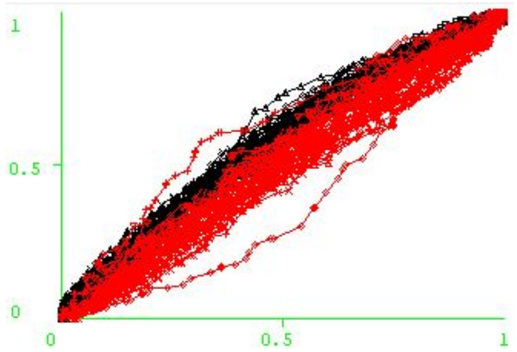
https://github.com/afodor/WekaExamples/blob/master/src/examples/RunOneROCCurve.java
Commit comment: Do 20 permuations

Comparison of two 16S microbiome datasets for colorectal adenomas from my lab…

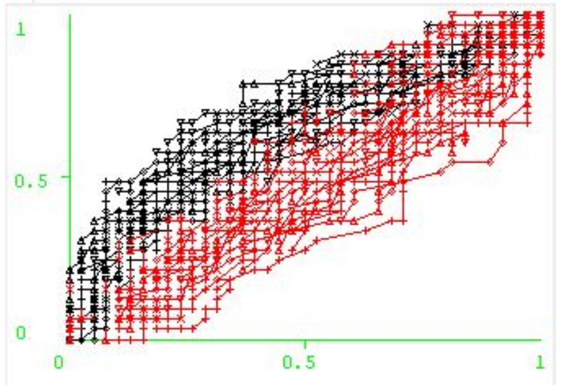
2012 2015

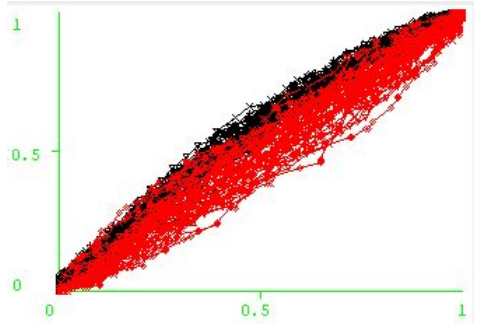
phylum



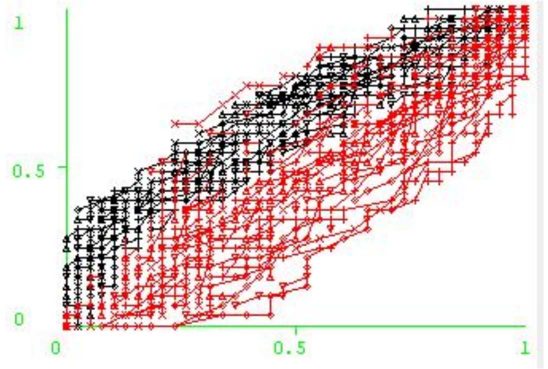


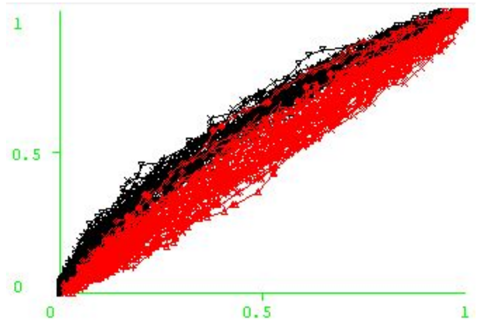
class





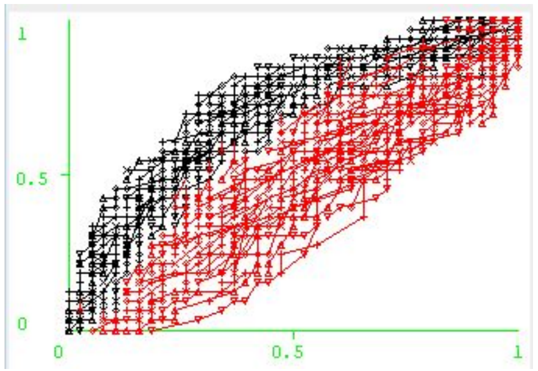
order

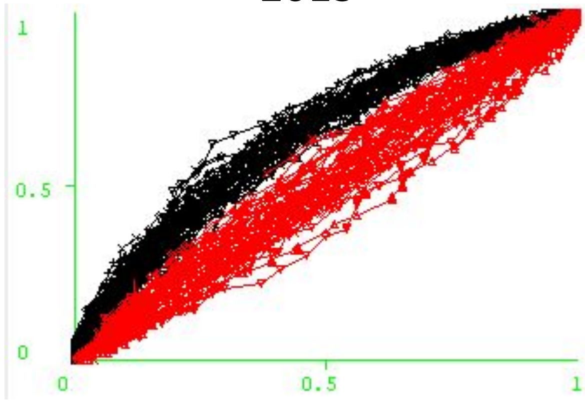




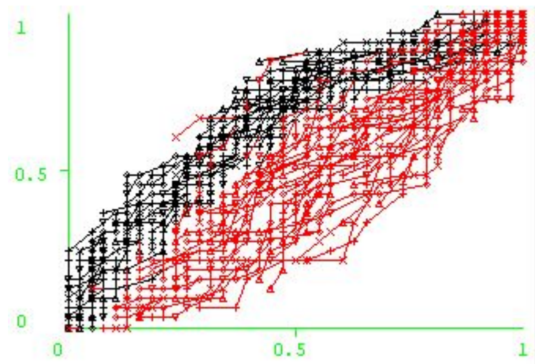
2012 2015

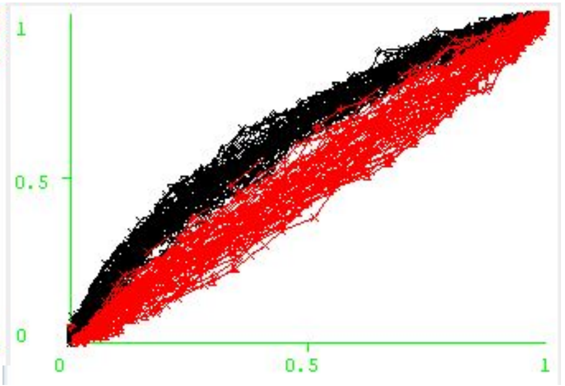
family





genus

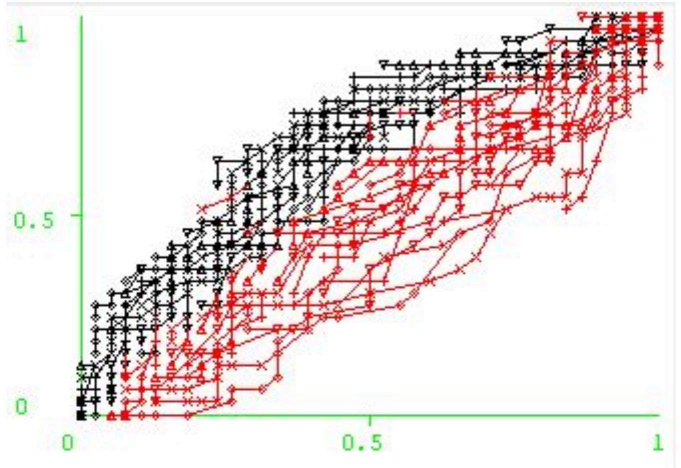




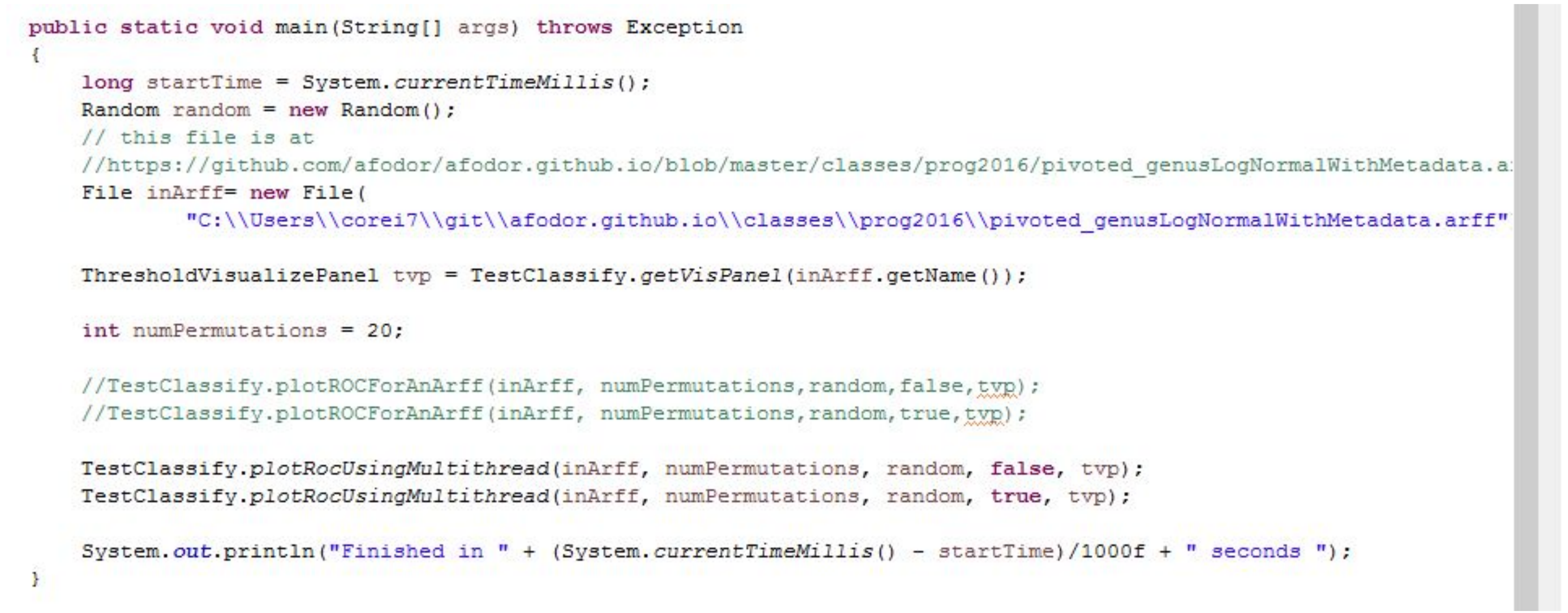
Multi-threaded application seems straight-forward…

A very robust 7.87 fold speedup on an 8 core box!

(but more on that later; we will return to how we did the speed up after we review multi-threading)

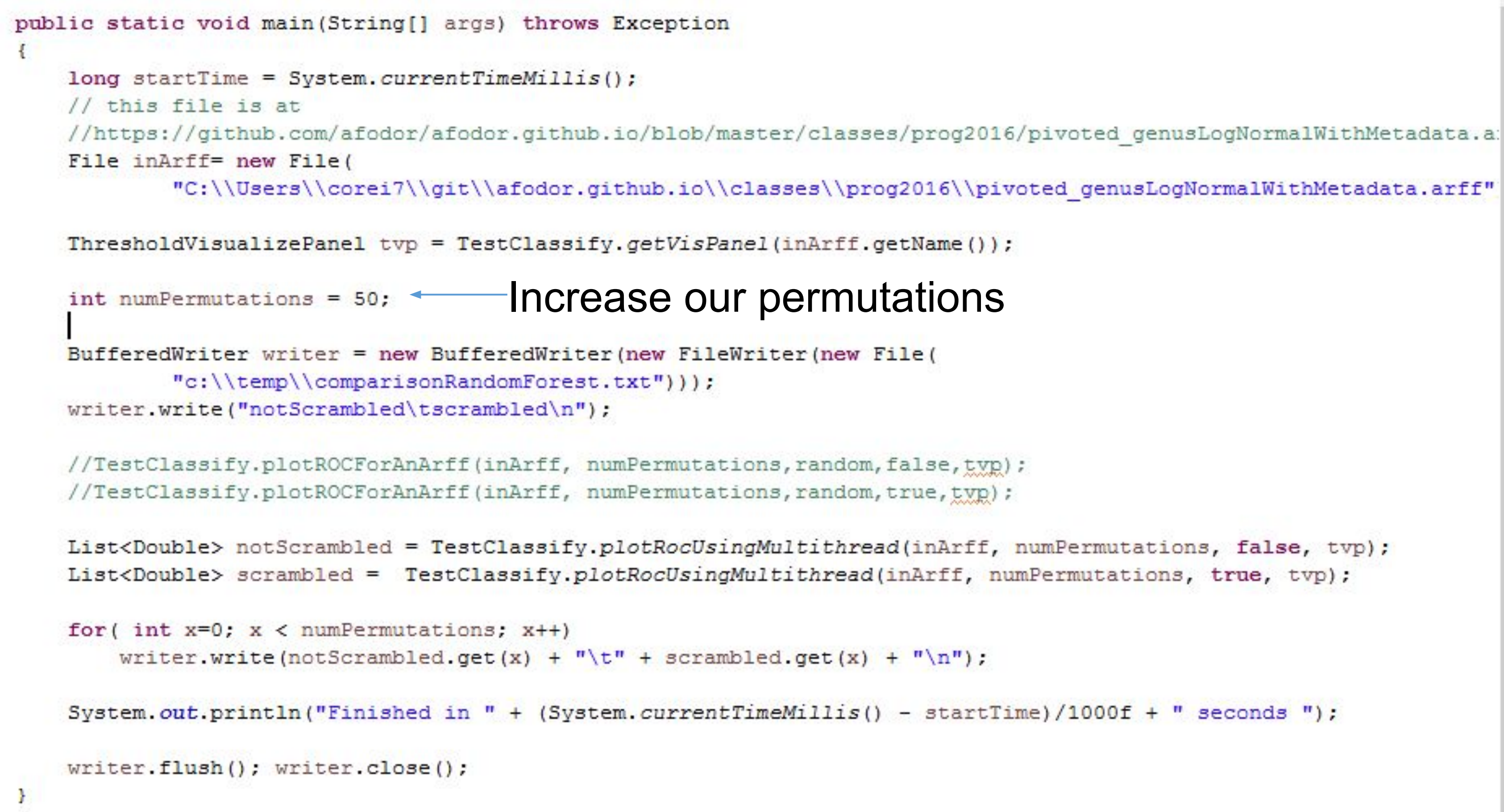


```
Finished in 12.637 seconds
```



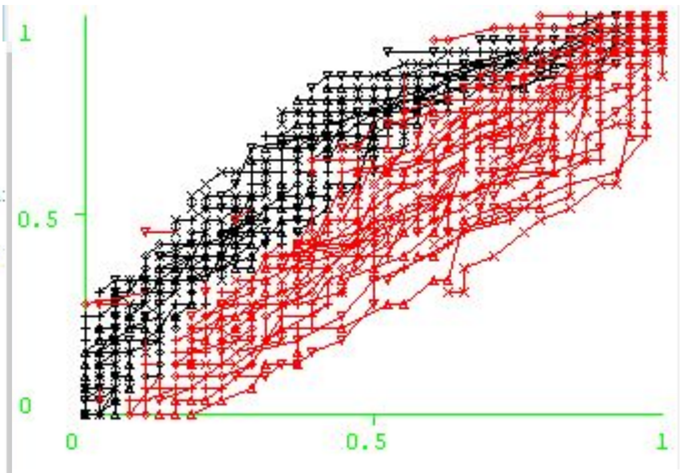
```
public static void main(String[] args) throws Exception
{
    long startTime = System.currentTimeMillis();
    Random random = new Random();
    // this file is at
    //https://github.com/afodor/afodor.github.io/blob/master/classes/prog2016/pivoted_genusLogNormalWithMetadata.a
    File inArff= new File(
            "C:\\Users\\corei7\\git\\afodor.github.io\\classes\\prog2016\\pivoted_genusLogNormalWithMetadata.arff"

    ThresholdVisualizePanel tvp = TestClassify.getVisPanel(inArff.getName());

    int numPermutations = 20;

    //TestClassify.plotROCForAnArff(inArff, numPermutations,random,false,tvp);
    //TestClassify.plotROCForAnArff(inArff, numPermutations,random,true,tvp);

    TestClassify.plotRocUsingMultithread(inArff, numPermutations, random, false, tvp);
    TestClassify.plotRocUsingMultithread(inArff, numPermutations, random, true, tvp);

    System.out.println("Finished in " + (System.currentTimeMillis() - startTime)/1000f + " seconds ");
}
```

We can ask, does scrambled do worse than not scrambled to a statistically significant degree?

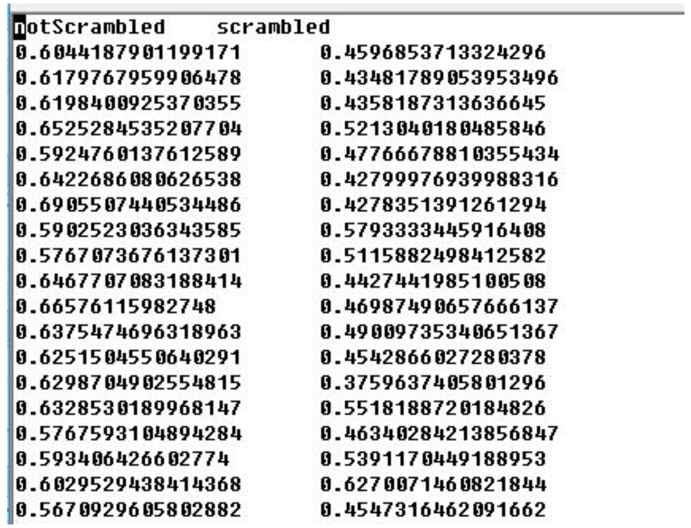


```java
public static void main(String[] args) throws Exception
{
    long startTime = System.currentTimeMillis();
    // this file is at
    //https://github.com/afodor/afodor.github.io/blob/master/classes/prog2016/pivoted_genusLogNormalWithMetadata.a
    File inArff= new File(
            "C:\\Users\\corei7\\git\\afodor.github.io\\classes\\prog2016\\pivoted_genusLogNormalWithMetadata.arff"

    ThresholdVisualizePanel tvp = TestClassify.getVisPanel(inArff.getName());

    int numPermutations = 50;

    BufferedWriter writer = new BufferedWriter(new FileWriter(new File(
            "c:\\temp\\comparisonRandomForest.txt")));
    writer.write("notScrambled\tscrambled\n");

    //TestClassify.plotROCForAnArff(inArff, numPermutations,random,false,tvp);
    //TestClassify.plotROCForAnArff(inArff, numPermutations,random,true,tvp);

    List<Double> notScrambled = TestClassify.plotRocUsingMultithread(inArff, numPermutations, false, tvp);
    List<Double> scrambled =  TestClassify.plotRocUsingMultithread(inArff, numPermutations, true, tvp);

    for( int x=0; x < numPermutations; x++)
        writer.write(notScrambled.get(x) + "\t" + scrambled.get(x) + "\n");

    System.out.println("Finished in " + (System.currentTimeMillis() - startTime)/1000f + " seconds ");

    writer.flush(); writer.close();
}
```

Increase our permutations

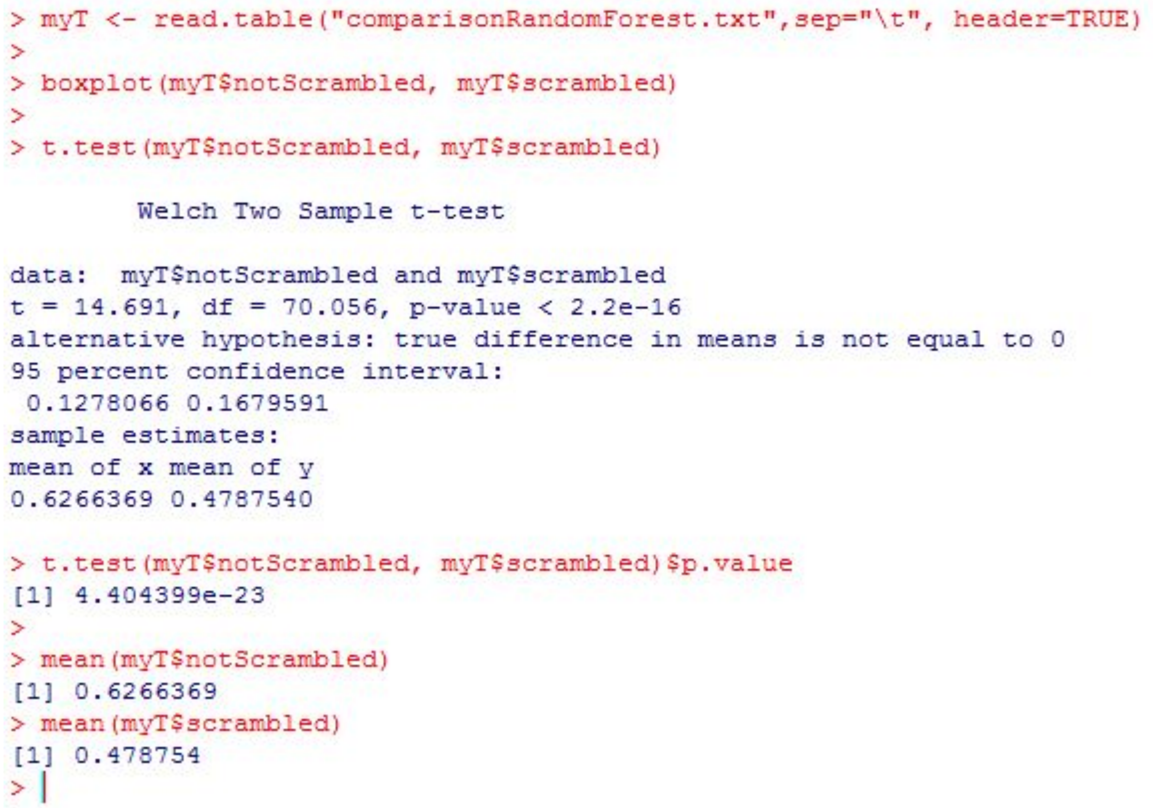


```
Finished in 18.88 seconds
```

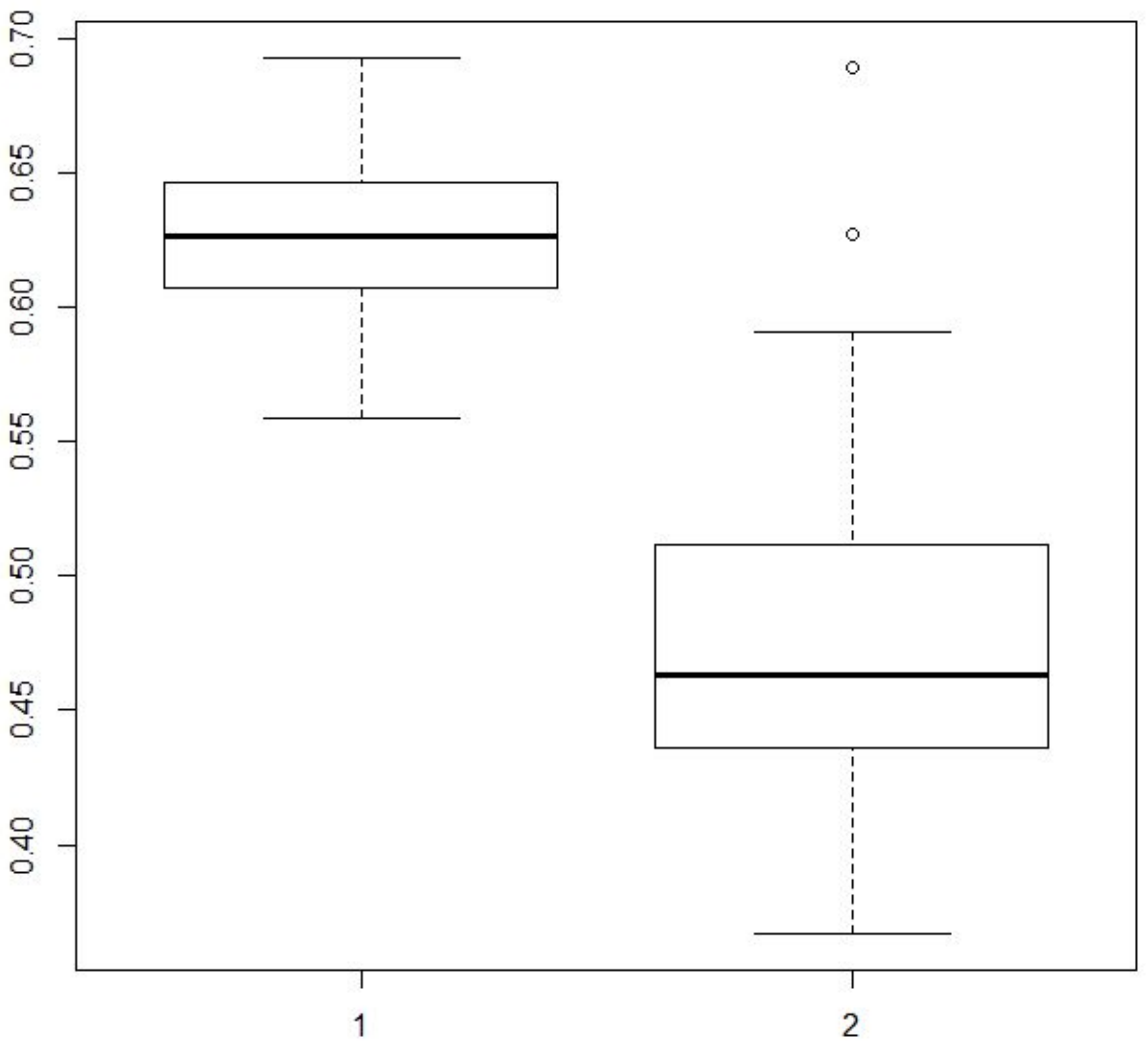


```
notScrambled    scrambled
0.6044187901199171      0.4596853713324296
0.6179767959906478      0.43481789053953496
0.6198400925370355      0.435818731363 6645
0.6525284535207704      0.5213040180485846
0.592476013761258 9     0.47766678810355434
0.6422686080626538      0.42799976939988316
0.6905507440534486      0.4278351391261294
0.5902523036343585      0.5793333445916408
0.5767073676137301      0.5115882498412582
0.6467707083188414      0.4427441985100508
0.66576115982748        0.46987490657666137
0.6375474696318963      0.49009735340651367
0.6251504550640291      0.4542866027280378
0.6298704902554815      0.3759637405801296
0.6328530189968147      0.5518188720184826
0.5767593104894284      0.46340284213856847
0.59340642660 2774      0.5391170449188953
0.6029529438414368      0.6270071460821844
0.5670929605802882      0.4547316462091662
```

We switch to R for the visualization and the statistical test



```
> myT <- read.table("comparisonRandomForest.txt",sep="\t", header=TRUE)
>
> boxplot(myT$notScrambled, myT$scrambled)
>
> t.test(myT$notScrambled, myT$scrambled)

        Welch Two Sample t-test

data:  myT$notScrambled and myT$scrambled
t = 14.691, df = 70.056, p-value < 2.2e-16
alternative hypothesis: true difference in means is not equal to 0
95 percent confidence interval:
 0.1278066 0.1679591
sample estimates:
mean of x mean of y
0.6266369 0.4787540

> t.test(myT$notScrambled, myT$scrambled)$p.value
[1] 4.404399e-23
>
> mean(myT$notScrambled)
[1] 0.6266369
> mean(myT$scrambled)
[1] 0.478754
> |
```

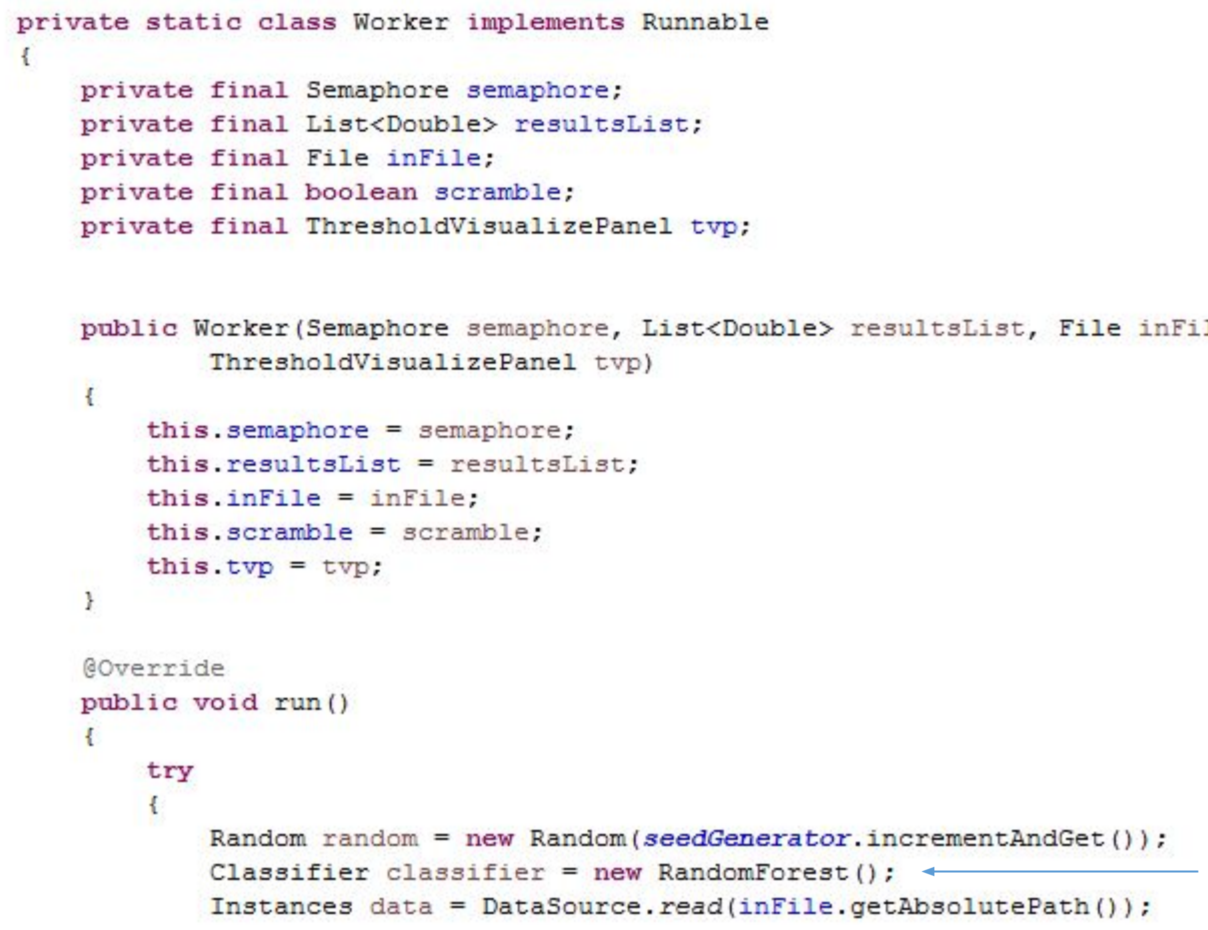


On the hand, our predictor has modest power (mean AUC = 0.63)

On the other hand, clearly better than shuffling labels!

ROC curves in WEKA
Visually estimating variance via null permutations
Comparing classifiers $\leftarrow$

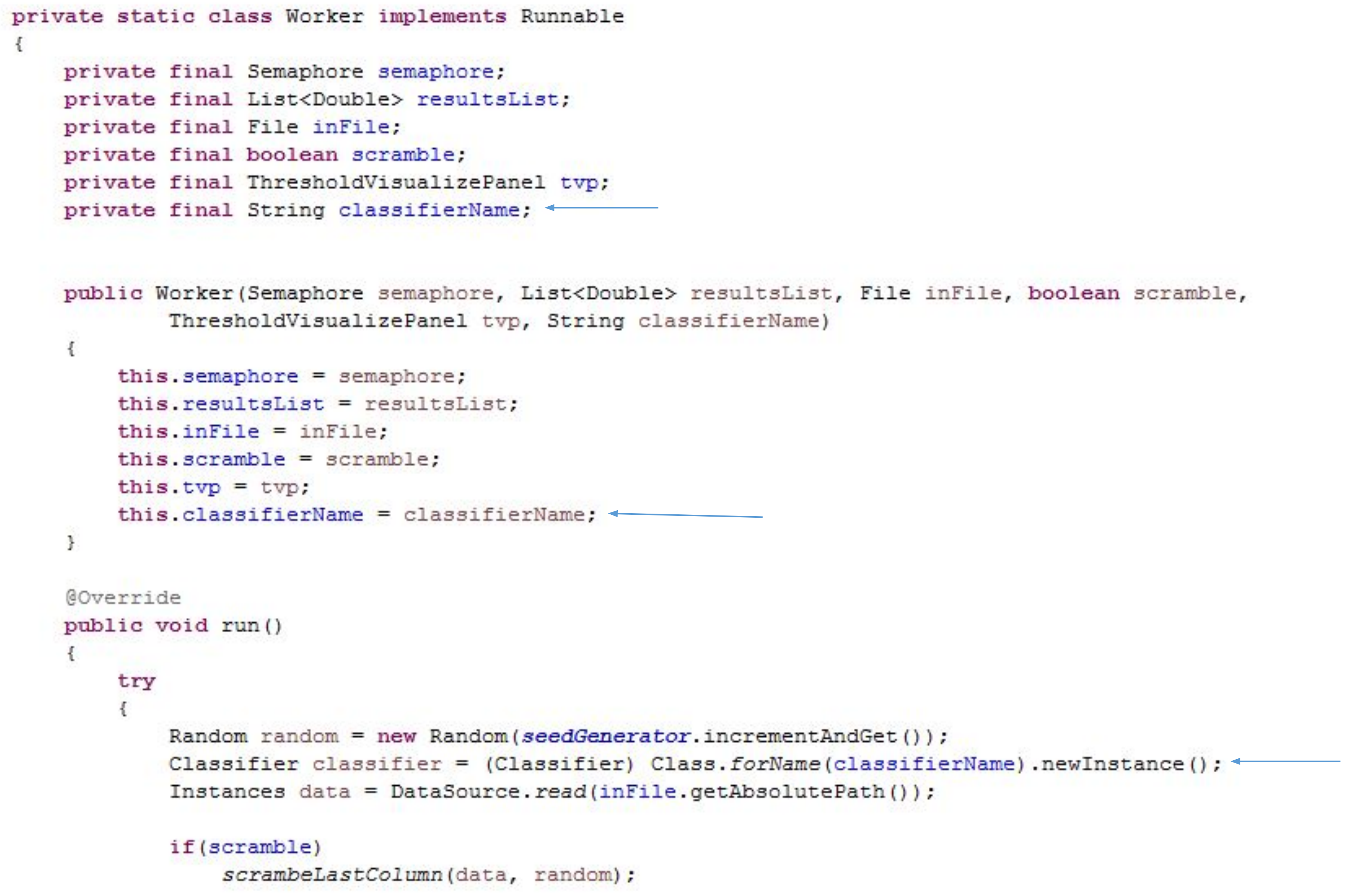
We'd like to be able to pass in a Classifier to our Worker, but we don't want our workers to share a Classifier object (as that would violate thread safety on mutable objects)



```java
private static class Worker implements Runnable
{
    private final Semaphore semaphore;
    private final List<Double> resultsList;
    private final File inFile;
    private final boolean scramble;
    private final ThresholdVisualizePanel tvp;


    public Worker(Semaphore semaphore, List<Double> resultsList, File inFile, boolean scramble,
            ThresholdVisualizePanel tvp)
    {
        this.semaphore = semaphore;
        this.resultsList = resultsList;
        this.inFile = inFile;
        this.scramble = scramble;
        this.tvp = tvp;
    }

    @Override
    public void run()
    {
        try
        {
            Random random = new Random(seedGenerator.incrementAndGet());
            Classifier classifier = new RandomForest();
            Instances data = DataSource.read(inFile.getAbsolutePath());
```

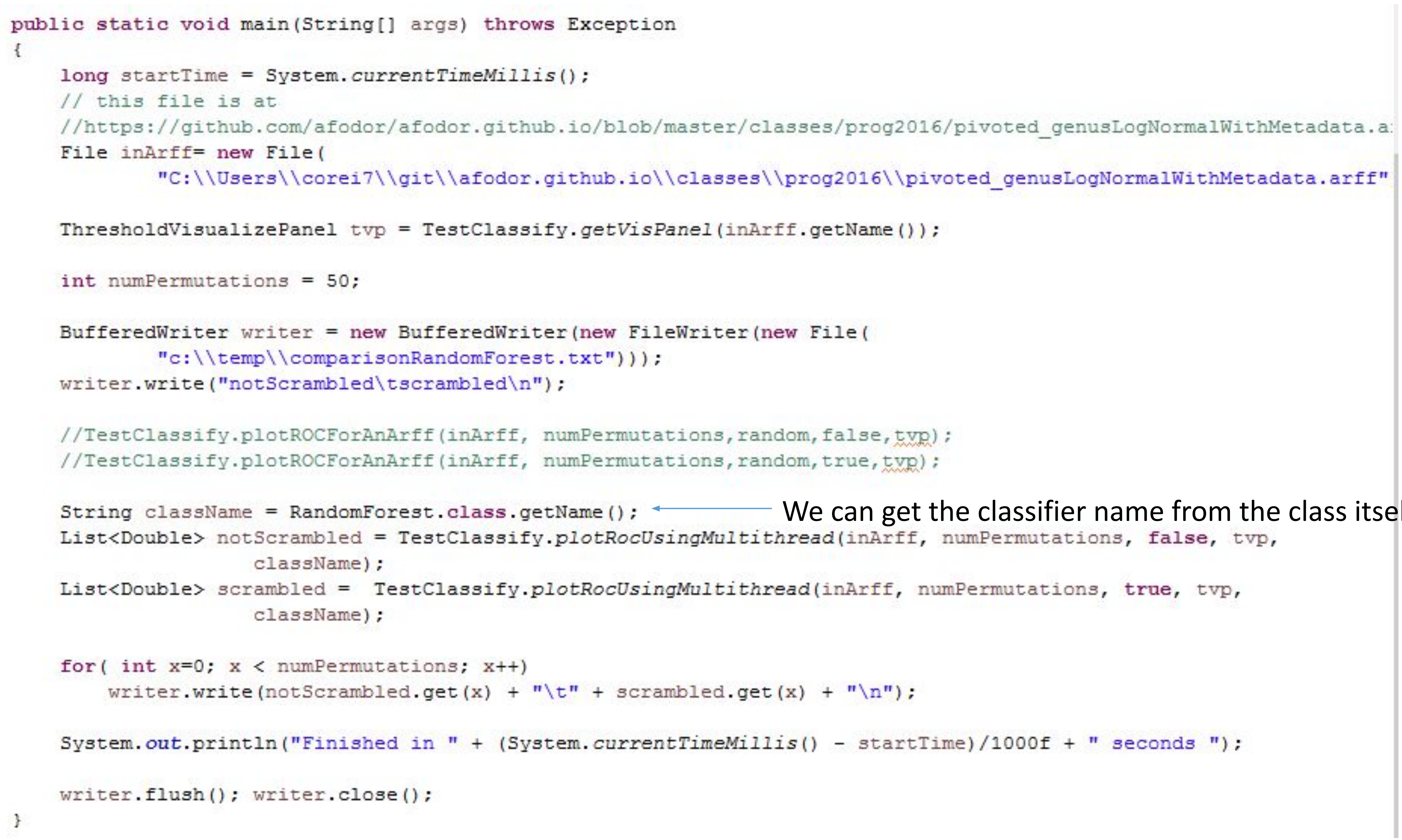
We can solve this problem by using Java's ability to dynamically control which class is instantiated…

We can tell the Worker at run time which classifier to instantiate…
(This will, of course, throw a runtime Exception if classifierName does not name a valid Classifier with a default constructor…)



```java
private static class Worker implements Runnable
{
    private final Semaphore semaphore;
    private final List<Double> resultsList;
    private final File inFile;
    private final boolean scramble;
    private final ThresholdVisualizePanel tvp;
    private final String classifierName;


    public Worker(Semaphore semaphore, List<Double> resultsList, File inFile, boolean scramble,
            ThresholdVisualizePanel tvp, String classifierName)
    {
        this.semaphore = semaphore;
        this.resultsList = resultsList;
        this.inFile = inFile;
        this.scramble = scramble;
        this.tvp = tvp;
        this.classifierName = classifierName;
    }

    @Override
    public void run()
    {
        try
        {
            Random random = new Random(seedGenerator.incrementAndGet());
            Classifier classifier = (Classifier) Class.forName(classifierName).newInstance();
            Instances data = DataSource.read(inFile.getAbsolutePath());

            if(scramble)
                scrambeLastColumn(data, random);
```

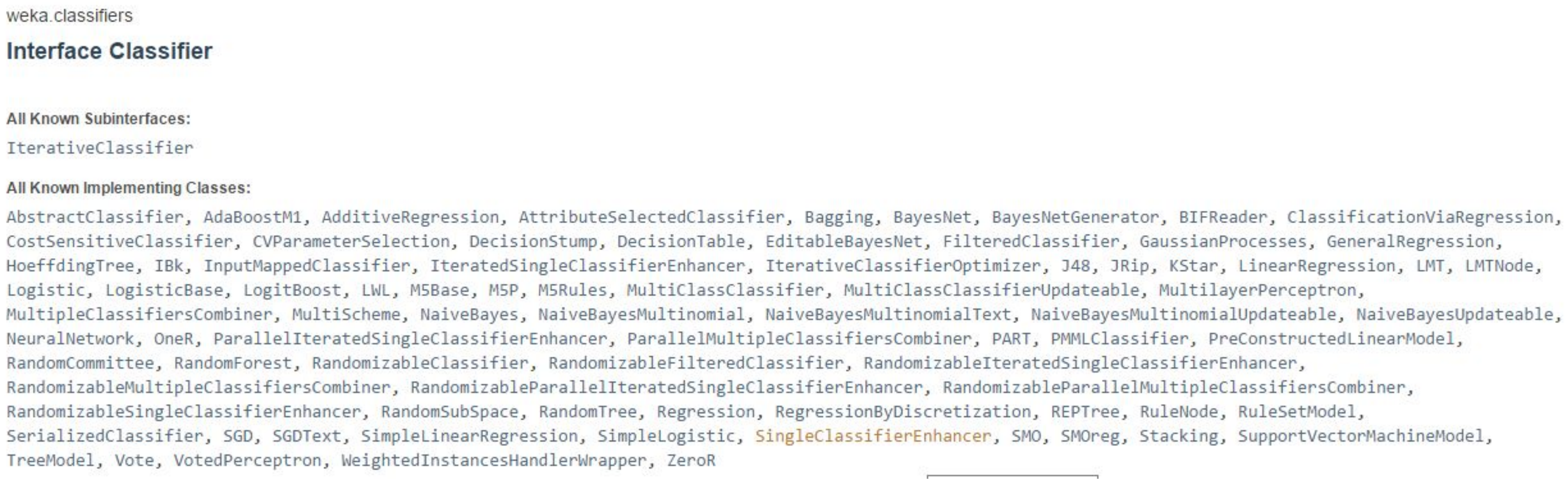
https://github.com/afodor/WekaExamples/blob/master/src/examples/TestClassify.java



```
public static void main(String[] args) throws Exception
{
    long startTime = System.currentTimeMillis();
    // this file is at
    //https://github.com/afodor/afodor.github.io/blob/master/classes/prog2016/pivoted_genusLogNormalWithMetadata.a
    File inArff= new File(
            "C:\\Users\\corei7\\git\\afodor.github.io\\classes\\prog2016\\pivoted_genusLogNormalWithMetadata.arff"

    ThresholdVisualizePanel tvp = TestClassify.getVisPanel(inArff.getName());

    int numPermutations = 50;

    BufferedWriter writer = new BufferedWriter(new FileWriter(new File(
            "c:\\temp\\comparisonRandomForest.txt")));
    writer.write("notScrambled\tscrambled\n");

    //TestClassify.plotROCForAnArff(inArff, numPermutations,random,false,tvp);
    //TestClassify.plotROCForAnArff(inArff, numPermutations,random,true,tvp);

    String className = RandomForest.class.getName();
    List<Double> notScrambled = TestClassify.plotRocUsingMultithread(inArff, numPermutations, false, tvp,
                className);
    List<Double> scrambled =  TestClassify.plotRocUsingMultithread(inArff, numPermutations, true, tvp,
                className);

    for( int x=0; x < numPermutations; x++)
        writer.write(notScrambled.get(x) + "\t" + scrambled.get(x) + "\n");

    System.out.println("Finished in " + (System.currentTimeMillis() - startTime)/1000f + " seconds ");

    writer.flush(); writer.close();
}
```

We can get the classifier name from the class itself

This allows us to use all of the classifiers that are implemented in Weka…



weka.classifiers

Interface Classifier

All Known Subinterfaces:

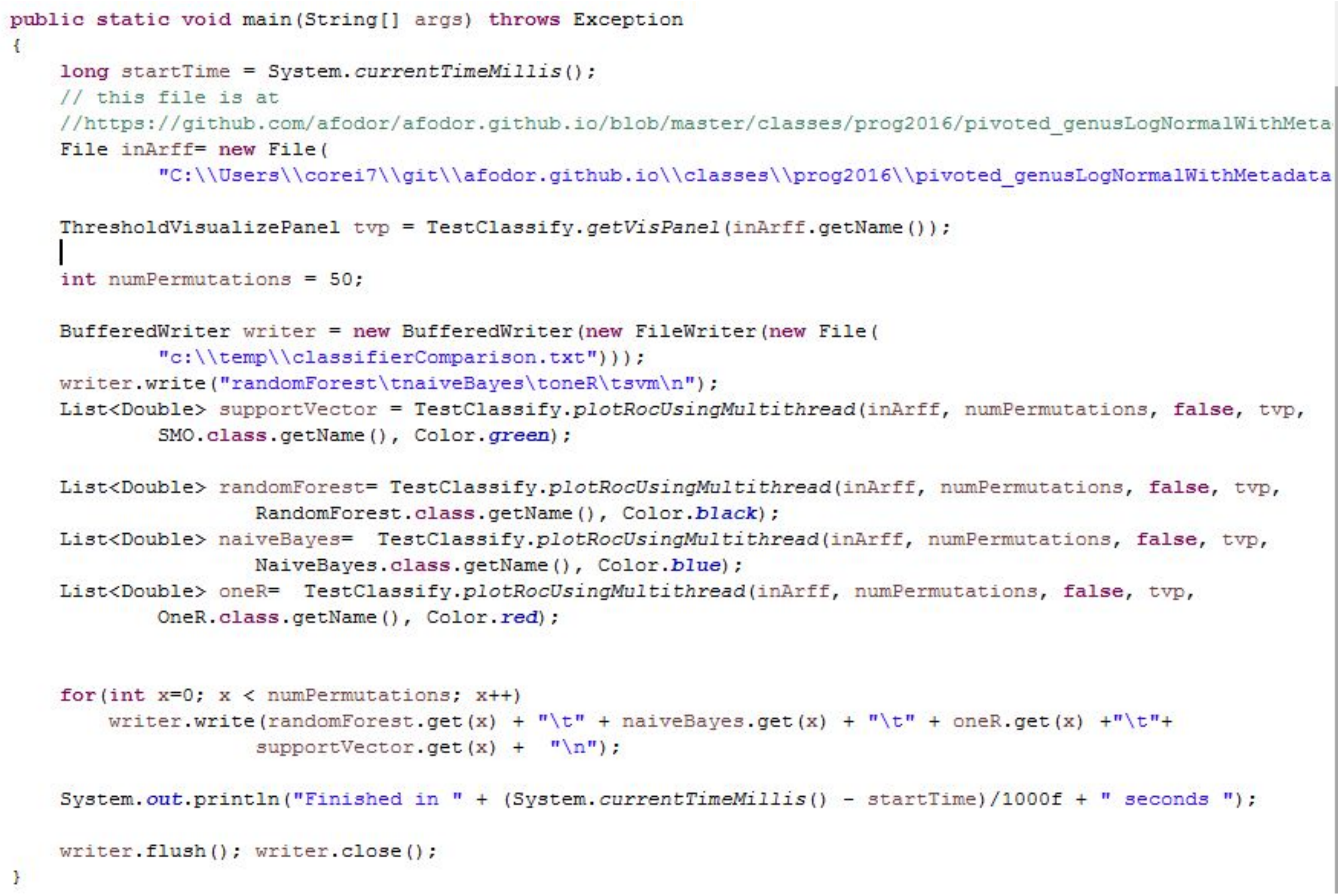
`IterativeClassifier`

All Known Implementing Classes:

```
AbstractClassifier, AdaBoostM1, AdditiveRegression, AttributeSelectedClassifier, Bagging, BayesNet, BayesNetGenerator, BIFReader, ClassificationViaRegression,
CostSensitiveClassifier, CVParameterSelection, DecisionStump, DecisionTable, EditableBayesNet, FilteredClassifier, GaussianProcesses, GeneralRegression,
HoeffdingTree, IBk, InputMappedClassifier, IteratedSingleClassifierEnhancer, IterativeClassifierOptimizer, J48, JRip, KStar, LinearRegression, LMT, LMTNode,
Logistic, LogisticBase, LogitBoost, LWL, M5Base, M5P, M5Rules, MultiClassClassifier, MultiClassClassifierUpdateable, MultilayerPerceptron,
MultipleClassifiersCombiner, MultiScheme, NaiveBayes, NaiveBayesMultinomial, NaiveBayesMultinomialText, NaiveBayesMultinomialUpdateable, NaiveBayesUpdateable,
NeuralNetwork, OneR, ParallelIteratedSingleClassifierEnhancer, ParallelMultipleClassifiersCombiner, PART, PMMLClassifier, PreConstructedLinearModel,
RandomCommittee, RandomForest, RandomizableClassifier, RandomizableFilteredClassifier, RandomizableIteratedSingleClassifierEnhancer,
RandomizableMultipleClassifiersCombiner, RandomizableParallelIteratedSingleClassifierEnhancer, RandomizableParallelMultipleClassifiersCombiner,
RandomizableSingleClassifierEnhancer, RandomSubSpace, RandomTree, Regression, RegressionByDiscretization, REPTree, RuleNode, RuleSetModel,
SerializedClassifier, SGD, SGDText, SimpleLinearRegression, SimpleLogistic, SingleClassifierEnhancer, SMO, SMOreg, Stacking, SupportVectorMachineModel,
TreeModel, Vote, VotedPerceptron, WeightedInstancesHandlerWrapper, ZeroR
```

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

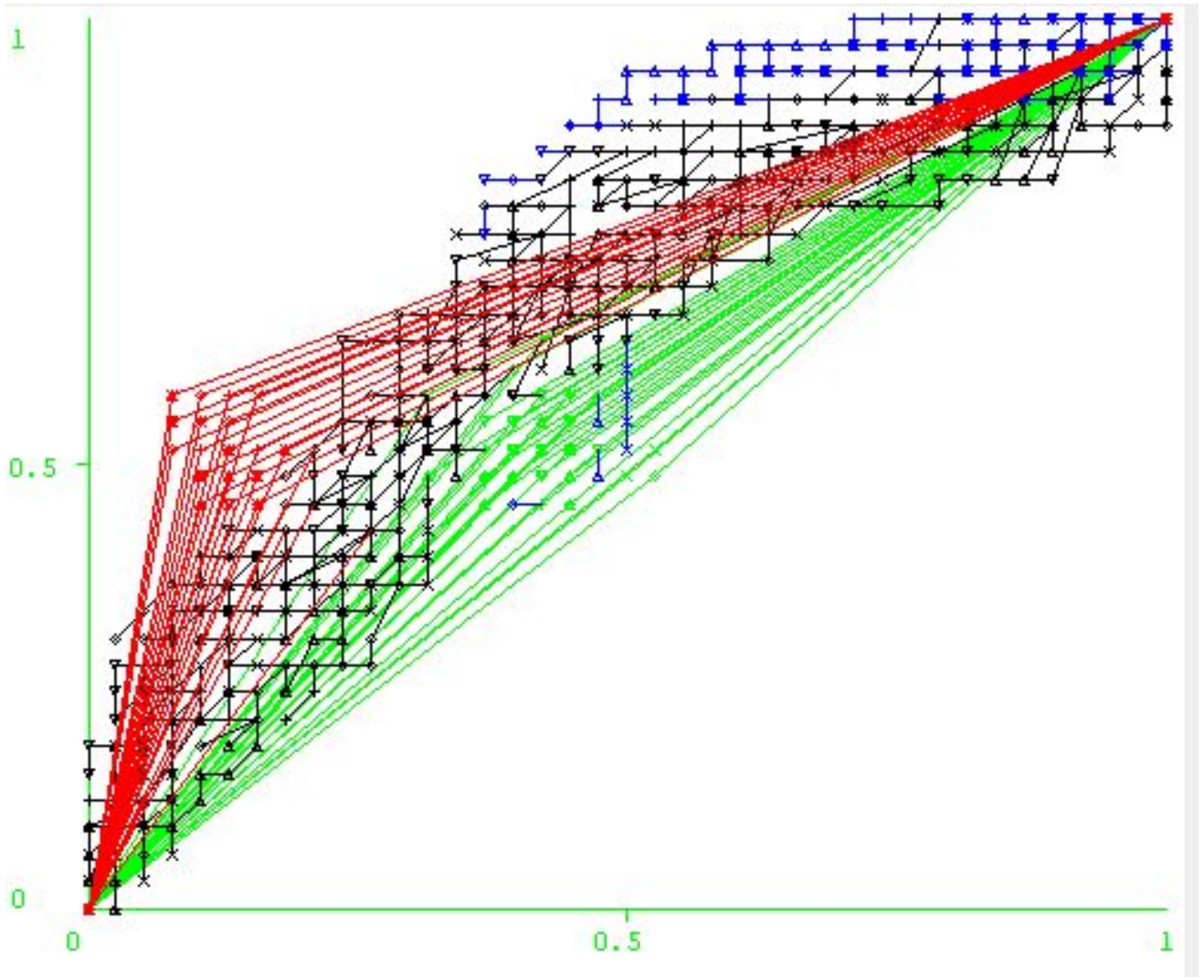
We can now directly compare classifiers (we modify the worker to pass in the color as well as the classifier)



```
public static void main(String[] args) throws Exception
{
    long startTime = System.currentTimeMillis();
    // this file is at
    //https://github.com/afodor/afodor.github.io/blob/master/classes/prog2016/pivoted_genusLogNormalWithMeta
    File inArff= new File(
            "C:\\Users\\corei7\\git\\afodor.github.io\\classes\\prog2016\\pivoted_genusLogNormalWithMetadata

    ThresholdVisualizePanel tvp = TestClassify.getVisPanel(inArff.getName());

    int numPermutations = 50;

    BufferedWriter writer = new BufferedWriter(new FileWriter(new File(
            "c:\\temp\\classifierComparison.txt")));
    writer.write("randomForest\tnaiveBayes\toneR\tsvm\n");
    List<Double> supportVector = TestClassify.plotRocUsingMultithread(inArff, numPermutations, false, tvp,
            SMO.class.getName(), Color.green);

    List<Double> randomForest= TestClassify.plotRocUsingMultithread(inArff, numPermutations, false, tvp,
                    RandomForest.class.getName(), Color.black);
    List<Double> naiveBayes=  TestClassify.plotRocUsingMultithread(inArff, numPermutations, false, tvp,
                    NaiveBayes.class.getName(), Color.blue);
    List<Double> oneR=  TestClassify.plotRocUsingMultithread(inArff, numPermutations, false, tvp,
            OneR.class.getName(), Color.red);


    for(int x=0; x < numPermutations; x++)
        writer.write(randomForest.get(x) + "\t" + naiveBayes.get(x) + "\t" + oneR.get(x) +"\t"+
                    supportVector.get(x) +  "\n");

    System.out.println("Finished in " + (System.currentTimeMillis() - startTime)/1000f + " seconds ");

    writer.flush(); writer.close();
}
```

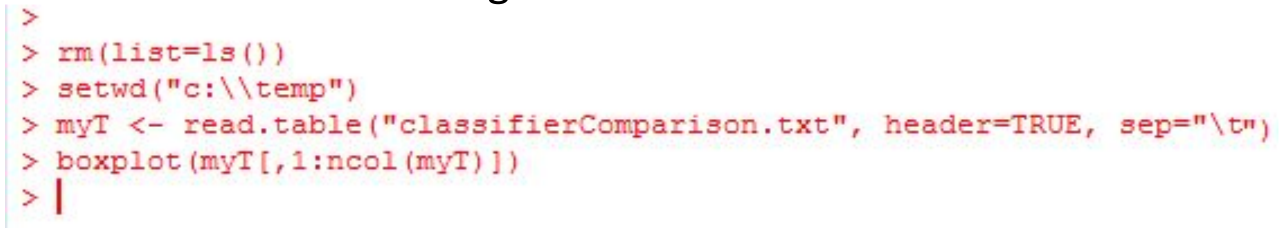
https://github.com/afodor/WekaExamples/blob/master/src/examples/CompareClassifiers.java

We can look at the ROC curves for a visual comparison…

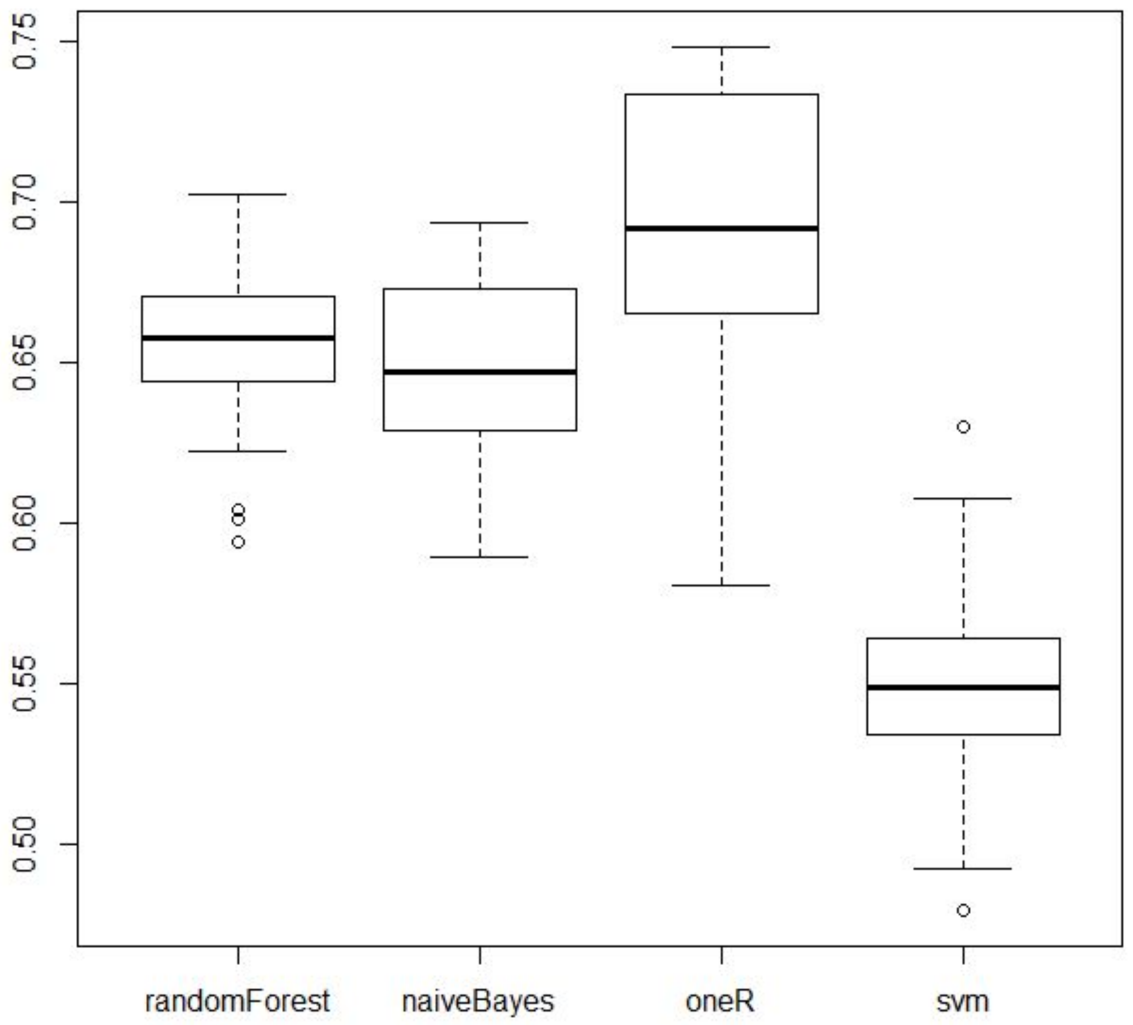


It has been noted many times in the literature, RandomForest works best for microbiome data…

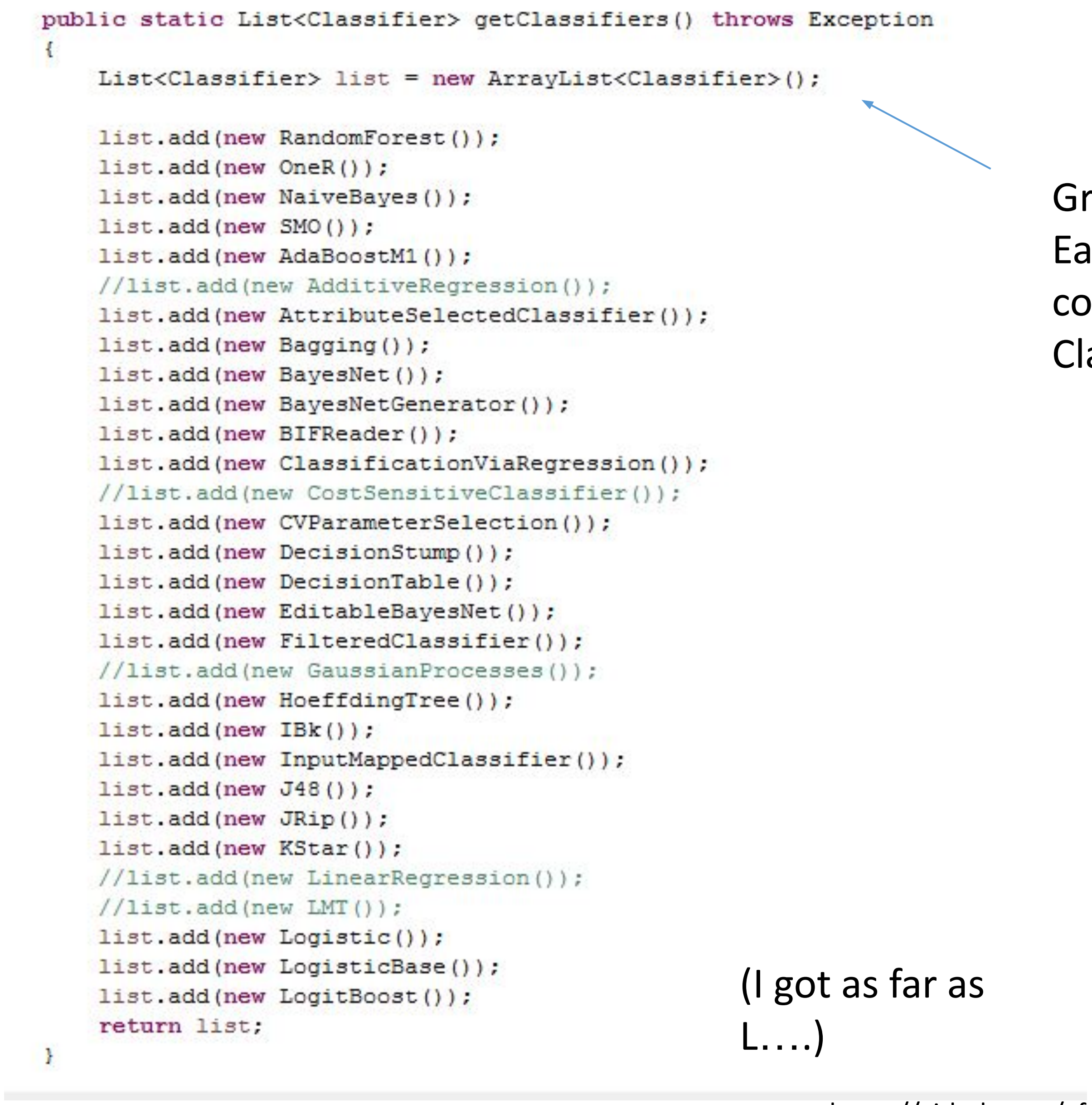
Here the "OneR" algorithm does well…



```
>
> rm(list=ls())
> setwd("c:\\temp")
> myT <- read.table("classifierComparison.txt", header=TRUE, sep="\t")
> boxplot(myT[,1:ncol(myT)])
> |
```



We can easily expand our classifier pool (commented out classifiers don't work on binary data or don't have Default constructors/parameter sets or otherwise threw an Exception…)



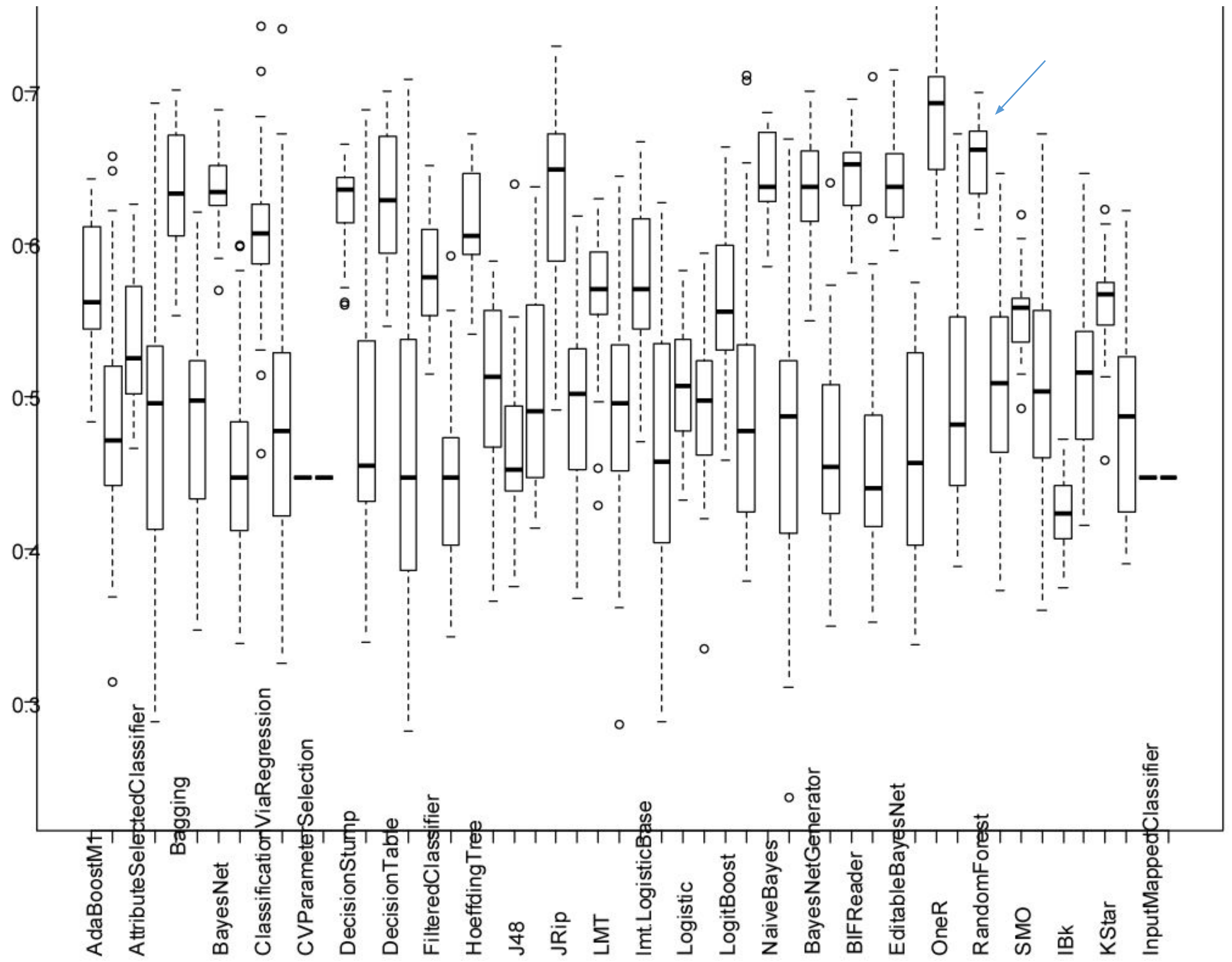
```java
public static List<Classifier> getClassifiers() throws Exception
{
    List<Classifier> list = new ArrayList<Classifier>();

    list.add(new RandomForest());
    list.add(new OneR());
    list.add(new NaiveBayes());
    list.add(new SMO());
    list.add(new AdaBoostM1());
    //list.add(new AdditiveRegression());
    list.add(new AttributeSelectedClassifier());
    list.add(new Bagging());
    list.add(new BayesNet());
    list.add(new BayesNetGenerator());
    list.add(new BIFReader());
    list.add(new ClassificationViaRegression());
    //list.add(new CostSensitiveClassifier());
    list.add(new CVParameterSelection());
    list.add(new DecisionStump());
    list.add(new DecisionTable());
    list.add(new EditableBayesNet());
    list.add(new FilteredClassifier());
    //list.add(new GaussianProcesses());
    list.add(new HoeffdingTree());
    list.add(new IBk());
    list.add(new InputMappedClassifier());
    list.add(new J48());
    list.add(new JRip());
    list.add(new KStar());
    //list.add(new LinearRegression());
    //list.add(new LMT());
    list.add(new Logistic());
    list.add(new LogisticBase());
    list.add(new LogitBoost());
    return list;
}
```

Great use of OO abstraction
Each classifier classifies in its own way, but our
code works at an abstract level to interact with many different
Classifiers.

(I got as far as
L….)

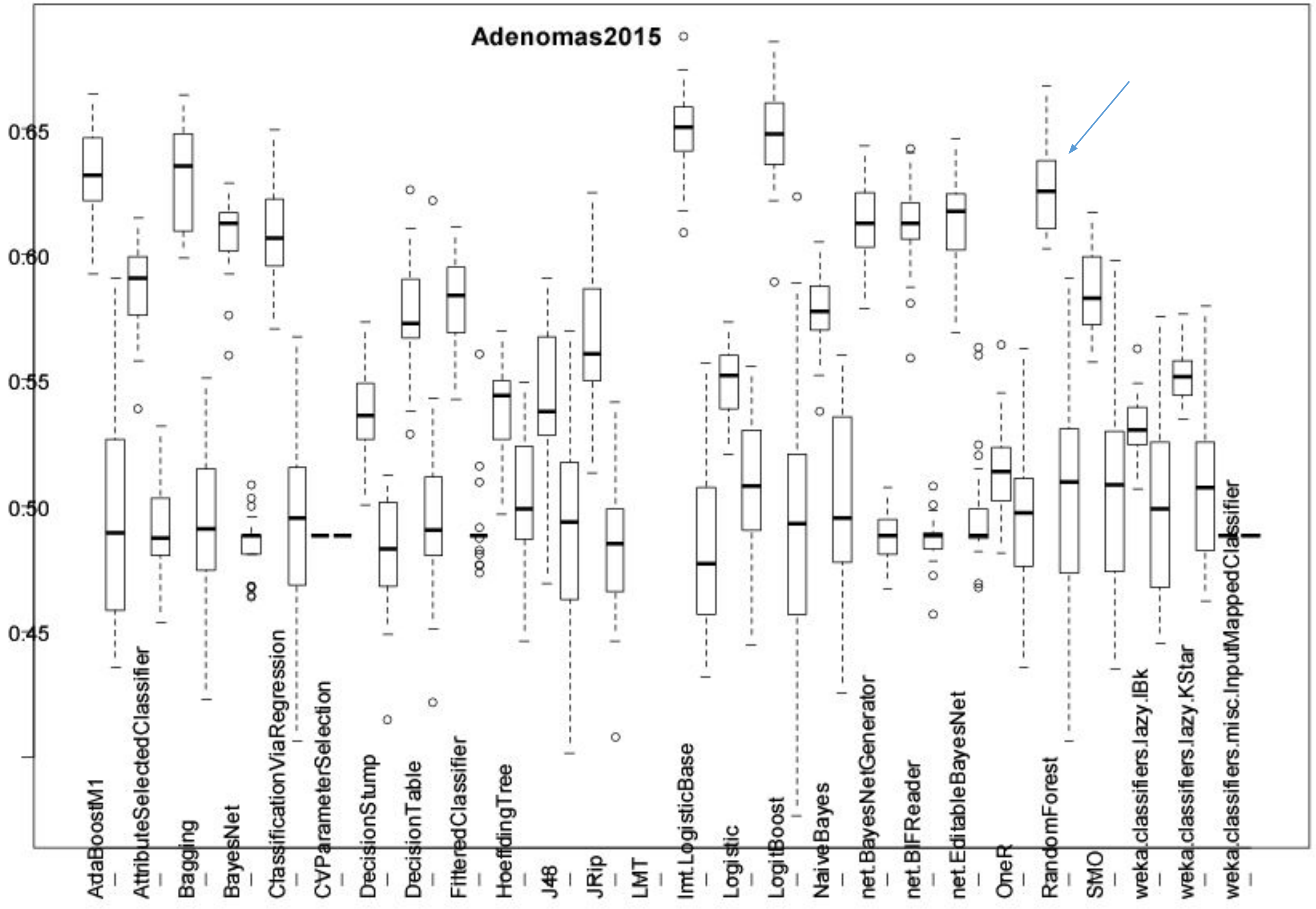
https://github.com/afodor/WekaExamples/blob/master/src/metaMergers/RunAllClassifiersVsAllDataLocal.java

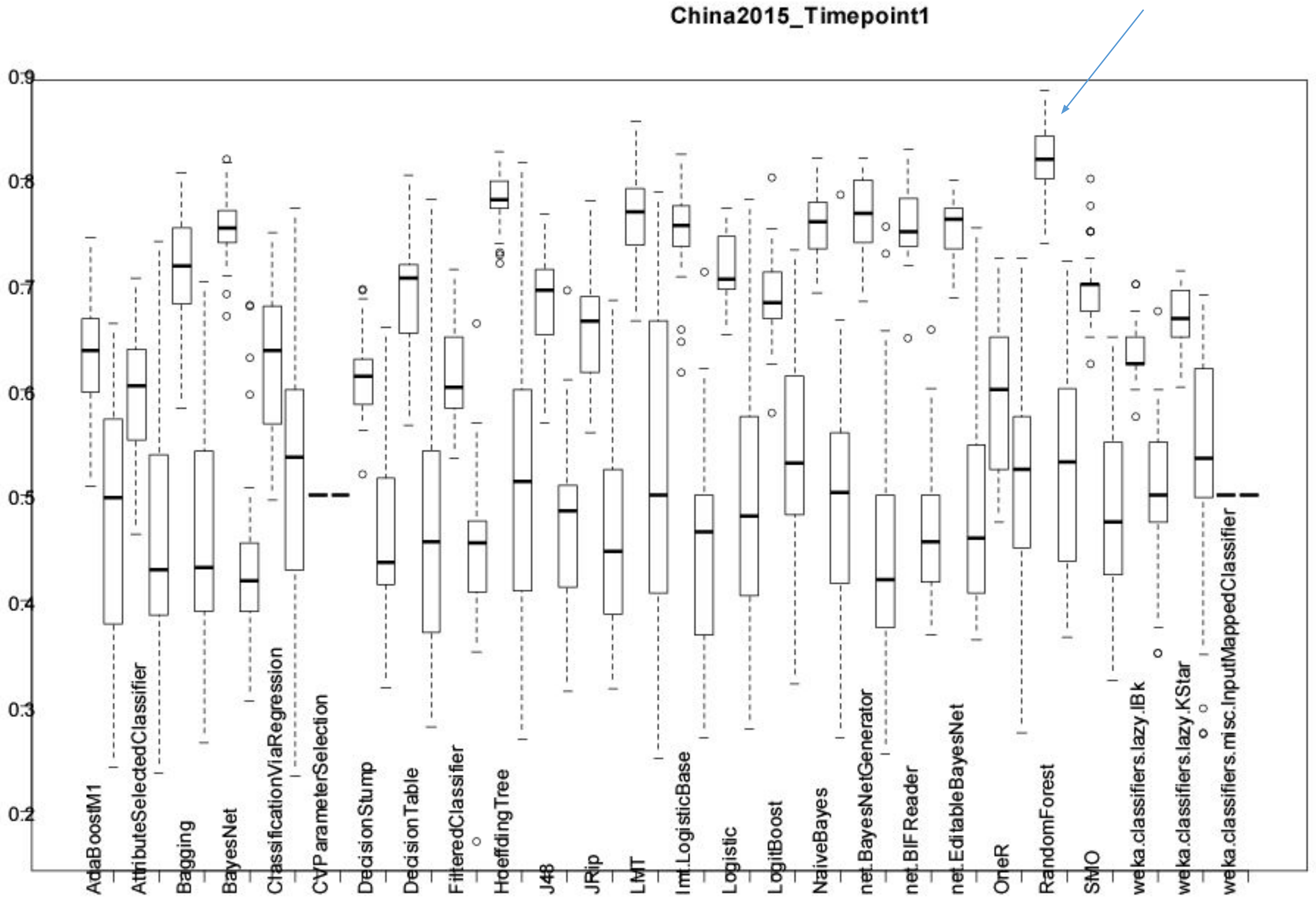
On our adenomas dataset at the genus level, OneR does well…



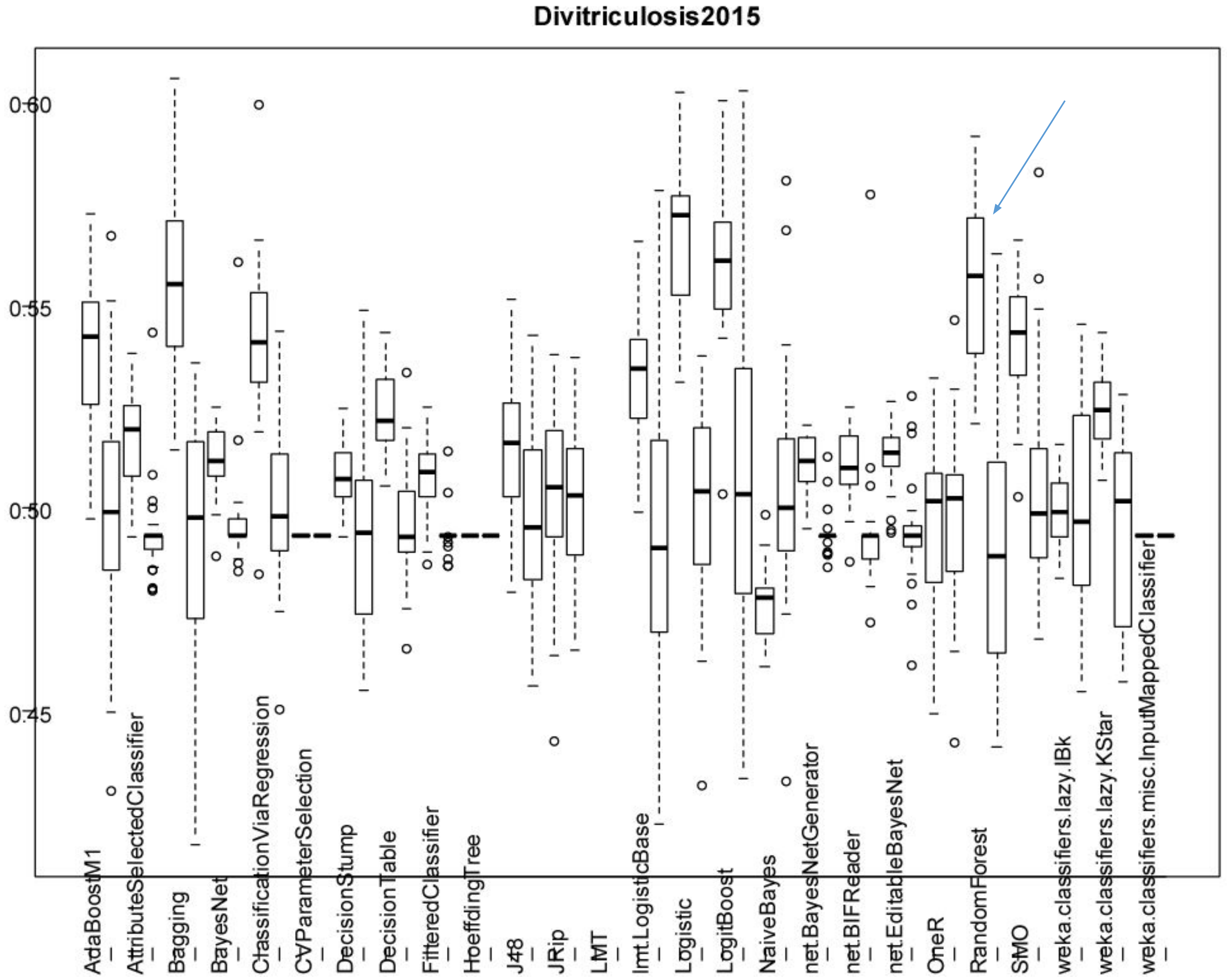
https://github.com/afodor/WekaExamples/blob/master/src/metaMergers/plotAllVsAll.txt

But RandomForest is more reliable across other datasets…




China2015_Timepoint1
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
AdaBoostM1
AttributeSelectedClassifier
Bagging
BayesNet
ClassificationViaRegression
CVParameterSelection
DecisionStump
DecisionTable
FilteredClassifier
HoeffdingTree
J48
JRip
LMT
lmt.LogisticBase
Logistic
LogitBoost
NaiveBayes
net.BayesNetGenerator
net.BIFReader
net.EditableBayesNet
OneR
RandomForest
SMO
weka.classifiers.lazy.IBk
weka.classifiers.lazy.KStar
weka.classifiers.misc.InputMappedClassifier

Models that have the words “boosting” or “Bagging” take a weighted average across many classifiers (Random Forest does this as well).
This is often a good strategy for genomic (and metagenomics data)

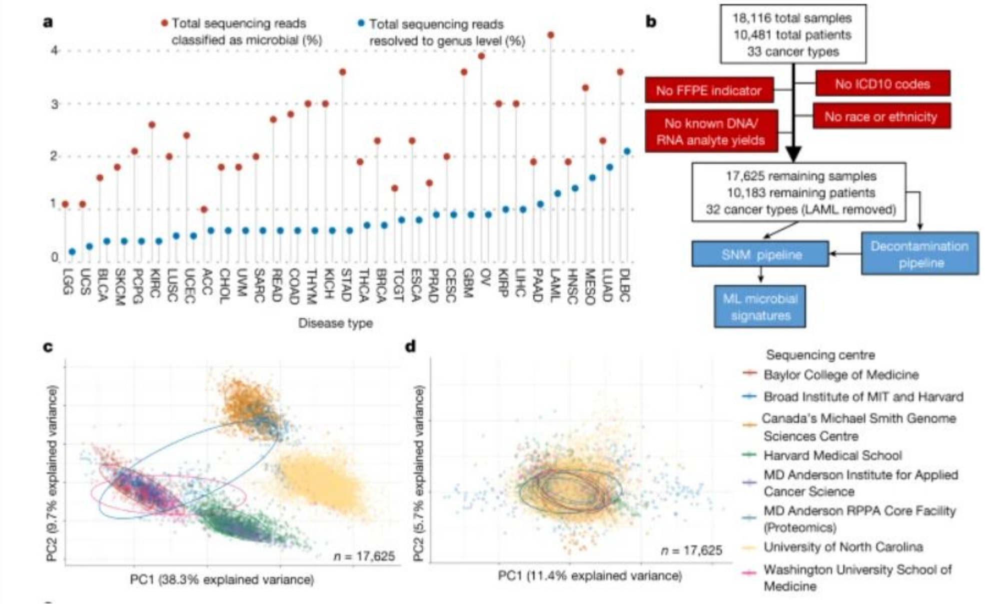


ROC curves are at the heart of the retracted paper…

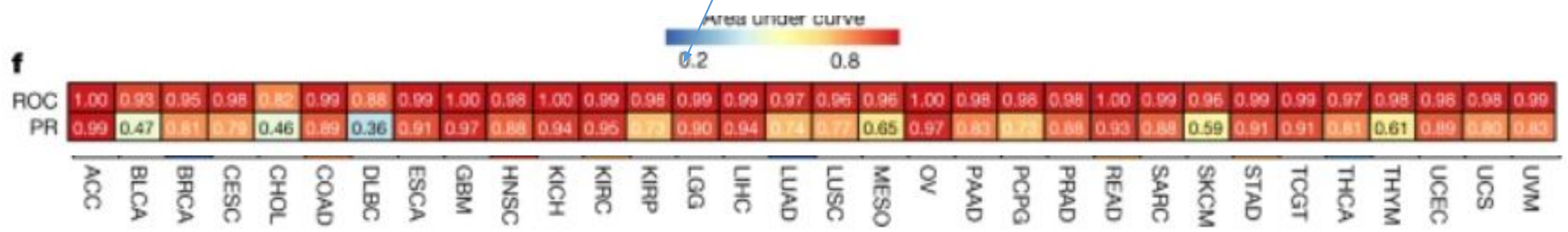
Microbiome analyses of blood and tissues suggest cancer diagnostic approach

https://www.nature.com/articles/s41586-020-2095-1

Fig. 1: Approach and overall findings of the cancer microbiome analysis of TCGA.



This paper claims close to perfect ability to discriminate different kinds of cancers



	ACC	BLCA	BRCA	CESC	CHOL	COAD	DLBC	ESCA	GBM	HNSC	KICH	KIRC	KIRP	LGG	LIHC	LUAD
ROC	1.00	0.93	0.95	0.98	0.82	0.99	0.88	0.99	1.00	0.98	1.00	0.99	0.98	0.99	0.99	0.97
PR	0.99	0.47	0.81	0.79	0.46	0.89	0.36	0.91	0.97	0.88	0.94	0.95	0.73	0.90	0.94	0.74

	LUSC	MESO	OV	PAAD	PCPG	PRAD	READ	SARC	SKCM	STAD	TCGT	THCA	THYM	UCEC	UCS	UVM
ROC	0.96	0.96	1.00	0.98	0.98	0.98	1.00	0.99	0.96	0.99	0.99	0.97	0.98	0.96	0.98	0.99
PR	0.77	0.65	0.97	0.83	0.75	0.88	0.93	0.88	0.59	0.91	0.91	0.81	0.61	0.89	0.80	0.83

| Human Microbiome | Research Article | 9 October 2023

Major data analysis errors invalidate cancer microbiome findings

Authors: Abraham Gihawi, Yuchen Ge, Jennifer Lu, Daniela Puiu, Amanda Xu, Colin S. Cooper, Daniel S. Brewer, Mihaela Pertea, Steven L. Salzberg | AUTHORS INFO & AFFILIATIONS

https://journals.asm.org/doi/10.1128/mbio.01607-23

This paper argued for a major flaw in the normalization scheme…

Fig 2

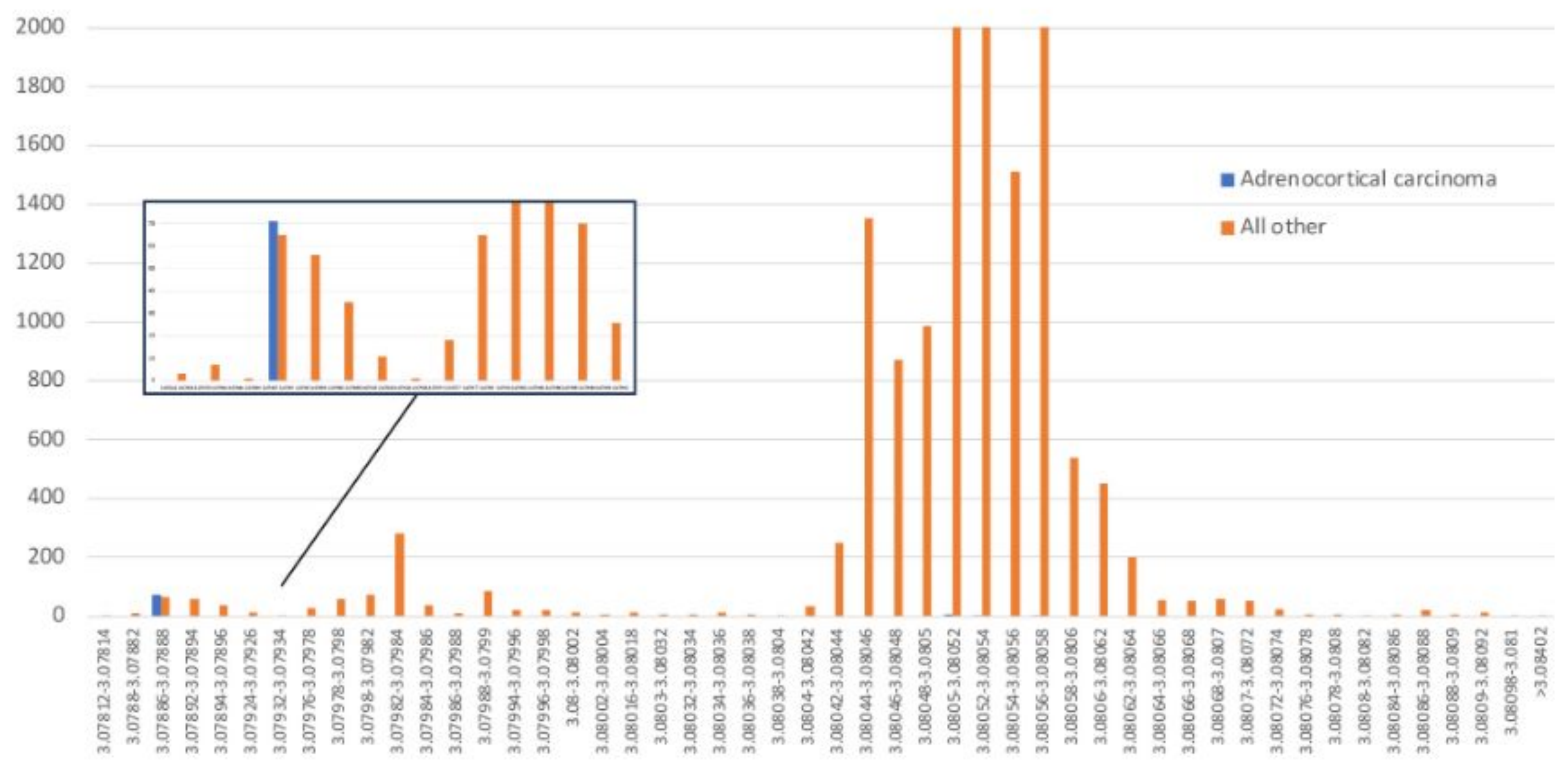
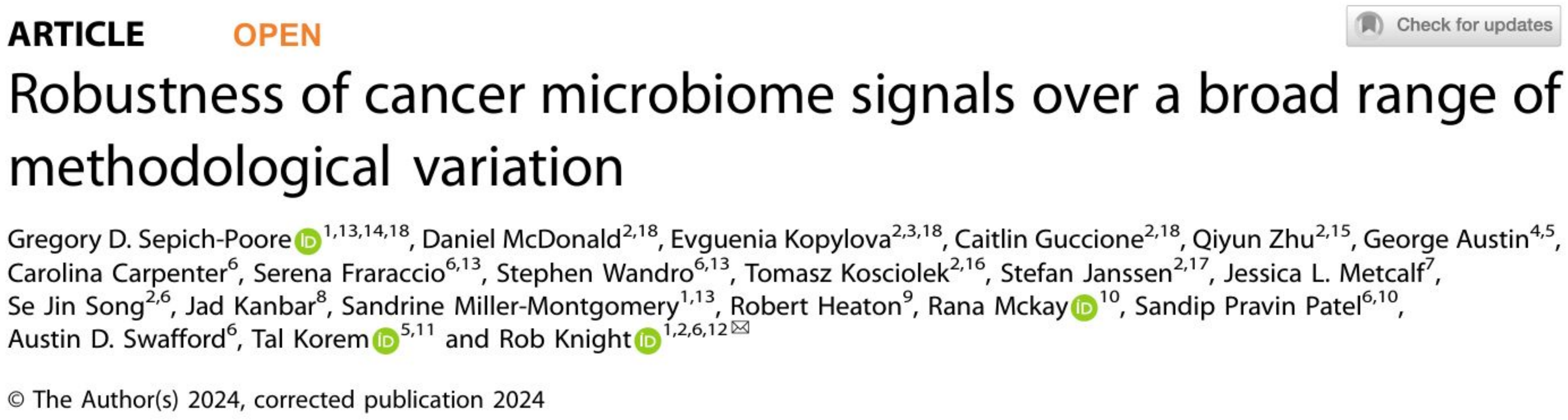


Fig 2 Distribution of normalized counts for *Hepandensovirus* for adrenocortical carcinoma (blue) vs all other samples (orange). The inset shows a zoomed-in view of the distribution for the small values. All raw values were zero.



ARTICLE OPEN

Check for updates

Robustness of cancer microbiome signals over a broad range of methodological variation

Gregory D. Sepich-Poore[1,13,14,18], Daniel McDonald[2,18], Evguenia Kopylova[2,3,18], Caitlin Guccione[2,18], Qiyun Zhu[2,15], George Austin[4,5], Carolina Carpenter[6], Serena Fraraccio[6,13], Stephen Wandro[6,13], Tomasz Kosciolek[2,16], Stefan Janssen[2,17], Jessica L. Metcalf[7], Se Jin Song[2,6], Jad Kanbar[8], Sandrine Miller-Montgomery[1,13], Robert Heaton[9], Rana Mckay[10], Sandip Pravin Patel[6,10], Austin D. Swafford[6], Tal Korem[5,11] and Rob Knight[1,2,6,12]

https://www.nature.com/articles/s41388-024-02974-w

This paper responded that the original analysis was, despite flaws, correct.

You can access the data in the supplementary materials.
Can you design an analysis to determine who is right? Possible independent project idea.

(You can use ChatGPT to generate Java or R code to run machine learning algorithms and produce ROC curves if you don't want to use WEKA!)s