
hyperSMURF Documentation

Release

Max Schubach

Jun 12, 2017

1	API Documentation	1
2	Indices and tables	33
	Bibliography	35

The friendly people at javadoc.io host our API documentation:

hyperSMURF <https://javadoc.io/doc/de.charite.compbio/hyperSMURF/0.3/>

Quickstart

This short How-To guides you from downloading the hyperSMURF weka plugin and load it into weka.

1. Download the current stable 0.3 release from our [GitHub project](#) by clicking 0.30.3.
2. Download the latest 3.9 development version of Weka [here](#) and install it.
3. Open Weka and navigate to the Package Manager (Tools -> Package manager).
4. Install the 0.3 file (Unofficial File/URL -> Browse -> OK).
5. Restart Weka. If you open the package explorer again you will see the hyperSMURF classifier under the Installed tab.
6. Download example data [quickstart_example.arff.gz](#)
7. Open the Weka Explorer. In the Preprocess tab click *Open file...* then navigate to the downloaded *quickstart_example.arff.gz* and select it. Choose as filetype **.arff.gz* and open the data.
8. Now you should be able to switch to the Classify tab and choose the hyperSMURF classifier under weka -> classifiers -> trees. After that you can start the classification and the classifier output should display results without errors. The end of the output should look like:

```
Time taken to build model: 7.6 seconds
```

```
=== Stratified cross-validation ===
```

```
=== Summary ===
```

Correctly Classified Instances	8744	87.44	%
Incorrectly Classified Instances	1256	12.56	%
Kappa statistic	0.7322		

```
Mean absolute error          0.2089
Root mean squared error      0.2881
Relative absolute error      49.7147 %
Root relative squared error   62.8514 %
Total Number of Instances    10000

=== Detailed Accuracy By Class ===

      ROC Area  PRC Area  Class  TP Rate  FP Rate  Precision  Recall  F-Measure  MCC
↪      0,993    0,997    c0     0,823    0,006    0,997     0,823    0,902     0,
↪758  0,993    0,997    c0     0,994    0,177    0,707     0,994    0,826     0,
↪758  0,993    0,985    c1     0,994    0,177    0,707     0,994    0,826     0,
↪      0,879    0,758    0,993    Weighted Avg.  0,874    0,057    0,910    0,874
      0,879    0,758    0,993    0,993

=== Confusion Matrix ===

      a      b  <-- classified as
5759 1239 |      a = c0
 17   2985 |      b = c1
```

Installation

Pre-built Binaries

Note: This is the recommended way of installing for normal users.

Pre-built binaries are available from [Maven Central](#). Download hyperSMURF-0.3 .jar from [here](#).

Install from Source

Note: You only need to install from source if you want to develop hyperSMURF in Java yourself.

There are two options of installing hyperSMURF. The recommended way for most users is to download a prebuilt binary and is well-described in the [Quickstart](#) section. This section describes how to build hyperSMURF from scratch.

Prerequisites

For building hyperSMURF, you will need

1. [Java JDK 8 or higher](#) for compiling hyperSMURF,
2. [Maven 3](#) for building hyperSMURF, and
3. [Git](#) for getting the sources.

Git Checkout

In this tutorial, we will download the hyperSMURF sources and build them in `~/hyperSMURF`.

```
~ # mkdir -p ~/hyperSMURF
~ # cd ~/hyperSMURF
hyperSMURF # git clone https://github.com/charite/hyperSMURF.git hyperSMURF
hyperSMURF # cd hyperSMURF
```

Maven Proxy Settings

If you are behind a proxy, you will get problems with Maven downloading dependencies. If you run into problems, make sure to also delete `~/.m2/repository`. Then, execute the following commands to fill `~/.m2/settings.xml`.

```
hyperSMURF # mkdir -p ~/.m2
hyperSMURF # test -f ~/.m2/settings.xml || cat >~/.m2/settings.xml <<END
<settings>
  <proxies>
    <proxy>
      <active>true</active>
      <protocol>http</protocol>
      <host>proxy.example.com</host>
      <port>8080</port>
      <nonProxyHosts>*.example.com</nonProxyHosts>
    </proxy>
  </proxies>
</settings>
END
```

Building

You can build hyperSMURF using `mvn package`. This will automatically download all dependencies, build hyperSMURF, and run all tests.

```
hyperSMURF # mvn package
```

In case that you have non-compiling test, you can use the `-DskipTests=true` parameter for skipping them.

```
hyperSMURF # mvn install -DskipTests=true
```

Creating Eclipse Projects

Maven can be used to generate Eclipse projects that can be imported by the Eclipse IDE. This can be done calling `mvn eclipse:eclipse` command after calling `mvn install`:

```
hyperSMURF # mvn install
hyperSMURF # mvn eclipse:eclipse
```

Examples using the Java package

In this section we will create a new Maven project, called hyperSMURF-tutorial, and using hyperSMURF together with Weka to train some tasks. Therefore we first have to set up an Maven project which will handle all libraries for us. Then we will start with a synthetic example. Afterwards we are using real genetic data for training. All files of this tutorial are available [here](#)

Requirements

First we have to build a maven project and include the hyperSMURF library into the *pom.xml* file. Therefore we generate a new folder (*hyperSMURF-tutorial*) with a new *pom.xml* file.

```
mkdir hyperSMURF-tutorial
cd hyperSMURF-tutorial
touch pom.xml
```

Then we open the *pom.xml* file in an editor and put in the following lines:

```
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/
↪XMLSchema-instance"
    xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/
↪maven-v4_0_0.xsd">
  <modelVersion>4.0.0</modelVersion>
  <groupId>de.charite.compbio.hypersmurf</groupId>
  <artifactId>hyperSMURF-tutorial</artifactId>
  <packaging>jar</packaging>
  <version>0.2</version>
  <name>hyperSMURF-tutorial</name>
  <properties>
    <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
  </properties>
  <dependencies>
    <dependency>
      <groupId>nz.ac.waikato.cms.weka</groupId>
      <artifactId>weka-dev</artifactId>
      <version>3.9.0</version>
    </dependency>
    <dependency>
      <groupId>de.charite.compbio</groupId>
      <artifactId>hyperSMURF</artifactId>
      <version>0.2</version>
    </dependency>
  </dependencies>
  <build>
    <plugins>
      <plugin>
        <artifactId>maven-assembly-plugin</artifactId>
        <executions>
          <execution>
            <phase>package</phase>
            <goals>
              <goal>single</goal>
            </goals>
          </execution>
        </executions>
      </plugin>
    </plugins>
  </build>
</project>
```



```

                                <descriptorRefs>
                                    <descriptorRef>jar-with-dependencies</
↪descriptorRef>
                                </descriptorRefs>
                            </configuration>
                        </plugin>
                    </plugins>
                </build>
            </project>

```

Now it we can generate Java files under the the folder `src/main/java` and to generate a final runnable jar-file we simply use the command `mvn clean package` to generate a jar fine into the `target` folder. Then we can run the `hyperSMURF-tutorial-0.2-jar-with-dependencies.jar` jar in `target` folder by specifying our main class (here `SyntheticExample` from the next section):

```

java -cp target/hyperSMURF-tutorial-0.2-jar-with-dependencies.jar de.charite.compbio.
↪hypersmurf.SyntheticExample

```

If you use Eclipse for developing a useful maven command might be `mvn eclipse:eclipse` to generate a project that can be imported into eclipse.

Simple usage examples using synthetic data

In this section we will add a new class `SyntheticExample.java` under the folder `src/main/java/de/charite/compbio/hypersmurf` to our maven project. This class has a main function to run it and two other main functions: (1) `generateSyntheticData` to generate synthetic imbalanced data and (2) `classify` to classify instances with a classifier using k-fold cross-validation.

The outline of the Java class `SyntheticExample.java` looks like this:

```

package de.charite.compbio.hypersmurf;

public class SyntheticExample {

    /**
     * We need a seed to make consistent predictions.
     */
    private static int SEED = 42;

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

    }

}

```

So the class only defines a seed to make predictions consistent. Then we use the RDG1 data generator from Weka to generate synthetic data. For example we will generate 10000 instances, each with 20 numeric attributes and set the index to the last attribute which contains class `c0` and `c1` by default. Then we randomize the data using our predefined seed:

```

RDG1 dataGenerator = new RDG1();
dataGenerator.setRelationName("SyntheticData");
dataGenerator.setNumExamples(10000);
dataGenerator.setNumAttributes(20);
dataGenerator.setNumNumeric(20);
dataGenerator.setSeed(SEED);
dataGenerator.defineDataFormat();

```

```
Instances instances = dataGenerator.generateExamples();

// set the index to last attribute
instances.setClassIndex(instances.numAttributes() - 1);

// randomize the data
Random random = new Random(SEED);
instances.randomize(random);
```

The problem is, that this data is not imbalanced. We can check this writing a short helper function.

```
private static int[] countClasses(Instances instances) {
    int[] counts = new int[instances.numClasses()];
    for (Instance instance : instances) {
        if (instance.classIsMissing() == false) {
            counts[(int) instance.classValue()]++;
        }
    }
    return counts;
}
```

Now if we add `int[] counts = countClasses(instances);` to our instance generation and print it using `System.out.println("Before imbalancing: " + Arrays.toString(counts));` we will see that `c0` has 2599 and `c1` has 7401 instances.

To imbalance the data we will write some own code. For example we want to use only 50 instances of `c0`. So we have to generate a new `Instances` object and assign all `c1` class instances and only 50 `c0` class instances to it.

```
// imbalance data
int numberOfClassOne = 50;
Instances imbalancedInstances = new Instances(instances, counts[1] +
    numberOfClassOne);
for (int i = 0; i < instances.numInstances(); i++) {
    if (instances.get(i).classValue() == 0.0) {
        if (numberOfClassOne != 0) {
            imbalancedInstances.add(instances.get(i));
            numberOfClassOne--;
        }
    } else {
        imbalancedInstances.add(instances.get(i));
    }
}
imbalancedInstances.randomize(random);
counts = countClasses(imbalancedInstances);
System.out.println("After imbalancing: " + Arrays.toString(counts));
```

The last line prints out the new imbalance. Now `c0` has only 50 instances.

Now we have to set up our classifier. We will use hyperSMURF with 10 partitions, oversampling factor of 2 (200%), no undersampling and each forest should have a size on 10.

```
// setup the hyperSMURF classifier
HyperSMURF clsHyperSMURF = new HyperSMURF();
clsHyperSMURF.setNumIterations(10);
clsHyperSMURF.setNumTrees(10);
clsHyperSMURF.setDistributionSpread(0);
clsHyperSMURF.setPercentage(200.0);
clsHyperSMURF.setSeed(SEED);
```

The next step will be the performance testing of hyperSMURF on the new generated imbalanced dataset. Therefore we will use a 5-fold cross-validation. To rerun this performance test using other classifiers we write everything into a new function `classify(AbstractClassifier cls, Instances instances, int folds)`. The `classify` function will collect the predictions over all 5 folds in the *Evaluation* object which then can be used to print out the performance results. Here is the complete `classify` function:

```
private static void classify(AbstractClassifier cls, Instances instances, int folds)
    throws Exception {
    // perform cross-validation and add predictions
    Instances predictedData = null;
    Evaluation eval = new Evaluation(instances);
    for (int n = 0; n < folds; n++) {
        System.out.println("Training fold " + n + " from " + folds + "...");
        Instances train = instances.trainCV(folds, n);
        Instances test = instances.testCV(folds, n);

        // build and evaluate classifier
        Classifier clsCopy = AbstractClassifier.makeCopy(cls);
        clsCopy.buildClassifier(train);
        eval.evaluateModel(clsCopy, test);

        // add predictions
        AddClassification filter = new AddClassification();
        filter.setClassifier(cls);
        filter.setOutputClassification(true);
        filter.setOutputDistribution(true);
        filter.setOutputErrorFlag(true);
        filter.setInputFormat(train);
        Filter.useFilter(train, filter); // trains the classifier
        // perform predictions on test set
        Instances pred = Filter.useFilter(test, filter);
        if (predictedData == null)
            predictedData = new Instances(pred, 0);
        for (int j = 0; j < pred.numInstances(); j++)
            predictedData.add(pred.instance(j));
    }

    // output evaluation
    System.out.println();
    System.out.println("=== Setup ===");
    System.out.println("Classifier: " + cls.getClass().getName() + " " + Utils.
    joinOptions(cls.getOptions()));
    System.out.println("Dataset: " + instances.relationName());
    System.out.println("Folds: " + folds);
    System.out.println("Seed: " + SEED);
    System.out.println();
    System.out.println(eval.toSummaryString("=== " + folds + "-fold Cross-
    validation ===", false));
    System.out.println();
    System.out.println(eval.toClassDetailsString("=== Details ==="));
}
```

Finally we can test hyperSMURF by running `classify(clsHyperSMURF, imbalancedInstances, 5);`. The output of the performance should be similar to the next text:

```
=== 5-fold Cross-validation ===
Correctly Classified Instances      7406      99.3961 %
```

Incorrectly Classified Instances	45	0.6039 %							
Kappa statistic	0.3809								
Mean absolute error	0.0858								
Root mean squared error	0.1278								
Relative absolute error	637.5943 %								
Root relative squared error	156.5741 %								
Total Number of Instances	7451								
=== Details ===									
	TP Rate	FP Rate	Precision	Recall	F-Measure	MCC	ROC Area		
↪PRC Area	Class								
↪0.337	c0	0.280	0.001	0.609	0.280	0.384	0.410	0.895	
↪0.999	c1	0.999	0.720	0.995	0.999	0.997	0.410	0.895	
Weighted Avg.		0.994	0.715	0.993	0.994	0.993	0.410	0.895	
↪0.995									

So we will get an AUROC of 0.895 and an AUPRC of 0.337 for our minority class *c0*. We can also use a Random Forest classifier using the same number of random trees to see the differences:

```
// setup a RF classifier
RandomForest clsRF = new RandomForest();
clsRF.setNumIterations(10);
clsRF.setSeed(SEED);

// classify RF
classify(clsRF, imbalancedInstances, 5);
```

Now we see that the RandomForest is only able to get an AUROC of 0.706 and an AUPRC of 0.109.

Usage examples with genetic data

HyperSMURF was designed to predict rare genomic variants, when the available examples of such variants are substantially less than *background* examples. This is a typical situation with genetic variants. For instance, we have only a small set of available variants known to be associated with Mendelian diseases in non-coding regions (positive examples) against the sea of background variants, i.e. a ratio of about 1 : 36,000 between positive and negative examples [Smedley2016].

Here we show how to use hyperSMURF to detect these rare features using data sets obtained from the original large set of Mendelian data [Smedley2016]. To provide usage examples that do not require more than 1 minute of computation time on a modern desktop computer, we considered data sets downsampled from the original Mendelian data. In particular we constructed Mendelian data sets with a progressive larger imbalance between Mendelian associated mutations and background genetic variants. We start with an artificially balanced data set and then we consider progressively imbalanced data sets with ratio *positive:negative* varying from 1 : 10, 1 : 100 and 1 : 1000. These data sets are downloadable as compressed *.arff* files, easily usable by Weka, from <https://www.github.com/charite/hyperSMURF-tutorial/data>.

The *Mendelian.balanced.arff.gz* file include 26 features, a column *class* howing the belonging class (1=positive, 0=negative) and a column *fold*. This is a numeric attribute with the number of the fold in which each example will be included according to the 10-fold cytogenetic band-aware CV procedure (0 to 9). In total the file contains 406 positives and 400 negatives.

Now we have to write the following code in our new Java file *MendelianExample.java* in folder *src/main/java/de/charite/compbio/hypersmurf*:

- Loader of the Instances.
- Cross-validation strategy that takes the the column *fold* into account when partitioning and removing the column *fold* for training.
- Setting up our hyperSMURF classifier.

So this will be the blank *MendelianExample.java* class:

```
package de.charite.compbio.hypersmurf;

public class MendelianExample {
    /**
     * We need a seed to make consistent predictions.
     */
    private static int SEED = 42;
    /**
     * The number of folds are predefined in the dataset
     */
    private static int FOLDS = 10;

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

    }
}
```

To read the data we simply can use the *ArffLoader* from Weka. We will use the first argument of the command-line arguments as our input file.

```
// read the file from the first argument of the command line input
ArffLoader reader = new ArffLoader();
reader.setFile(new File(args[0]));
Instances instances = reader.getDataSet();
```

Then we have to set the class attribute. This is the last attribute of our instances. So we write `instances.setClassIndex(instances.numAttributes() - 1);`. Because we have a balanced dataset of the Mendelian data we do not need to do over- or undersampling. So we simply run hyperSMURF with two partitions and a forest size of ten. Over- and undersampling settings have to be set to 0.

```
// setup the hyperSMURF classifier
HyperSMURF clsHyperSMURF = new HyperSMURF();
clsHyperSMURF.setNumIterations(2);
clsHyperSMURF.setNumTrees(10);
clsHyperSMURF.setDistributionSpread(0);
clsHyperSMURF.setPercentage(0.0);
clsHyperSMURF.setSeed(SEED);
```

Now we arrived at the special cytogenetic band-aware cross-validation. The folds are predefined as attribute *fold* in the instances object. So we have to select the instances on that fold but have to remove the fold attribute before training or testing a classifier. So we will write a small helper method that gives us a given fold for testing or the inverse for training. The blank method can be written like this:

We will use the filter *SubsetbyExpression* to get the instances with the fold and we can simply use the *Instances* method *deleteAttributeAt(int index)* to remove the fold attribute. For *SubsetbyExpression* filter we write a regular expression like *Attribute = n* or *!(Attribute = n)* to get the *n*'th fold (or all other folds). *Attribute* will be written by like 'ATT' with the index (one based) of the attribute. This we can get using `int indexFold = instances.attribute("fold").index();` (zero based) and we have to increment it by one for our filter method. So the

content of our *getFold* method can look like:

```
// filter on fold variable
int indexFold = instances.attribute("fold").index();
SubsetByExpression filterFold = new SubsetByExpression();
if (invert)
    filterFold.setExpression("!(ATT" + (indexFold + 1) + " = " + fold + ")");
else
    filterFold.setExpression("ATT" + (indexFold + 1) + " = " + fold);
filterFold.setInputFormat(instances);
Instances filtered = Filter.useFilter(instances, filterFold);

// remove fold attribute
filtered.deleteAttributeAt(indexFold);

return filtered;
```

Now it is time for the cross-validation this is similar to the Synthetic Example but we will use the *getFold* method to make the train/test partitioning.

```
// perform cross-validation and add predictions
Instances predictedData = null;
Evaluation eval = new Evaluation(instances);
for (int n = 0; n < FOLDS; n++) {
    System.out.println("Training fold " + (n+1) + " from " + FOLDS + "...");
    Instances train = getFold(instances, n, true);
    Instances test = getFold(instances, n, false);

    // build and evaluate classifier
    Classifier clsCopy = AbstractClassifier.makeCopy(cls);
    clsCopy.buildClassifier(train);
    eval.evaluateModel(clsCopy, test);





    // add predictions
    AddClassification filter = new AddClassification();
    filter.setClassifier(cls);
    filter.setOutputClassification(true);
    filter.setOutputDistribution(true);
    filter.setOutputErrorFlag(true);
    filter.setInputFormat(train);
    Filter.useFilter(train, filter); // trains the classifier
    // perform predictions on test set
    Instances pred = Filter.useFilter(test, filter);
    if (predictedData == null)
        predictedData = new Instances(pred, 0);
    for (int j = 0; j < pred.numInstances(); j++)
        predictedData.add(pred.instance(j));
}

// output evaluation
System.out.println();
System.out.println("=== Setup ===");
System.out.println("Classifier: " + cls.getClass().getName() + " " + Utils.
    → joinOptions(cls.getOptions()));
System.out.println("Dataset: " + instances.relationName());
System.out.println("Folds: " + FOLDS);
System.out.println("Seed: " + SEED);
System.out.println();
System.out.println(eval.toSummaryString("=== " + FOLDS + "-fold Cross-validation ===",
    → false));
```

```
System.out.println();
System.out.println(eval.toClassDetailsString("== Details =="));
```

If we run hyperSMURF with the settings above the command-line output will show an AUPRC 0.989 of and an AUROC of 0.989 of our class 1 which are the Mendelian regulatory mutations. This is the complete output:

```
=== 10-fold Cross-validation ===
Correctly Classified Instances      770          95.5335 %
Incorrectly Classified Instances    36           4.4665 %
Kappa statistic                    0.9107
Mean absolute error                 0.0898
Root mean squared error             0.1925
Relative absolute error             17.9538 %
Root relative squared error         38.4915 %
Total Number of Instances          806




=== Details ===
      TP Rate  FP Rate  Precision  Recall   F-Measure  MCC      ROC Area  
↪PRC Area  Class
      0.985    0.074    0.929    0.985    0.956      0.912    0.989    
↪0.983      0
      0.926    0.015    0.984    0.926    0.954      0.912    0.989    
↪0.989      1
Weighted Avg. 0.955    0.044    0.957    0.955    0.955      0.912    0.989    
↪0.986
```

Then we can perform the same computation using the progressively imbalanced data sets: *Mendelian.1_10.arff.gz*, *Mendelian.1_100.arff.gz*, and *Mendelian.1_1000.arff.gz*. Of course every time we have to adapt the settings of hyperSMURF.

Using *Mendelian.1_10.arff.gz*, hyperSMURF and the output can look like:

```
// setup the hyperSMURF classifier
clsHyperSMURF = new HyperSMURF();
clsHyperSMURF.setNumIterations(5);
clsHyperSMURF.setNumTrees(10);
clsHyperSMURF.setDistributionSpread(0);
clsHyperSMURF.setPercentage(100.0);
clsHyperSMURF.setSeed(SEED);
```

```
=== 10-fold Cross-validation ===
Correctly Classified Instances      4310          97.8212 %
Incorrectly Classified Instances     96           2.1788 %
Kappa statistic                    0.8779
Mean absolute error                 0.0577
Root mean squared error             0.1427
Relative absolute error             34.4437 %
Root relative squared error         49.3333 %
Total Number of Instances          4406

=== Details ===
      TP Rate  FP Rate  Precision  Recall   F-Measure  MCC      ROC Area  
↪PRC Area  Class
      0.981    0.044    0.995    0.981    0.988      0.880    0.990    
↪0.999      0
      0.956    0.020    0.833    0.956    0.890      0.880    0.990    
↪0.950      1
```

Weighted Avg.	0.978	0.042	0.980	0.978	0.979	0.880	0.990	↵
	↵0.994							

Increasing the imbalance with *Mendelian.1_100.arff.gz*:

```
// setup the hyperSMURF classifier
clsHyperSMURF = new HyperSMURF();
clsHyperSMURF.setNumIterations(5);
clsHyperSMURF.setNumTrees(10);
clsHyperSMURF.setDistributionSpread(0);
clsHyperSMURF.setPercentage(100.0);
clsHyperSMURF.setSeed(SEED);
```

```
=== 10-fold Cross-validation ===
Correctly Classified Instances      39987      99.1348 %
Incorrectly Classified Instances    349      0.8652 %
Kappa statistic                    0.6795
Mean absolute error                0.0249
Root mean squared error            0.0851
Relative absolute error            124.7001 %
Root relative squared error        85.3023 %
Total Number of Instances          40336

=== Details ===
      TP Rate  FP Rate  Precision  Recall   F-Measure  MCC      ROC Area  ↵
↵PRC Area  Class
      0.992    0.071    0.999    0.992    0.996      0.705    0.991    ↵
↵1.000      0
      0.929    0.008    0.541    0.929    0.684      0.705    0.991    ↵
↵0.900      1
Weighted Avg.  0.991    0.071    0.995    0.991    0.992      0.705    0.991    ↵
↵0.999
```

Again increasing the imbalance with *Mendelian.1_1000.arff.gz*:

```
// setup the hyperSMURF classifier
clsHyperSMURF = new HyperSMURF();
clsHyperSMURF.setNumIterations(10);
clsHyperSMURF.setNumTrees(10);
clsHyperSMURF.setDistributionSpread(3);
clsHyperSMURF.setPercentage(200.0);
clsHyperSMURF.setSeed(SEED);
```

```
=== 10-fold Cross-validation ===
Correctly Classified Instances      392436      99.2597 %
Incorrectly Classified Instances    2927      0.7403 %
Kappa statistic                    0.2021
Mean absolute error                0.0233
Root mean squared error            0.0805
Relative absolute error            1135.2254 %
Root relative squared error        251.4735 %
Total Number of Instances          395363

=== Details ===
      TP Rate  FP Rate  Precision  Recall   F-Measure  MCC      ROC Area  ↵
↵PRC Area  Class
```


↪1.000	0	0.993	0.079	1.000	0.993	0.996	0.323	0.989	↪
↪0.773	1	0.921	0.007	0.114	0.921	0.204	0.323	0.989	↪
Weighted Avg.		0.993	0.079	0.999	0.993	0.995	0.323	0.989	↪
↪1.000									

As we can see, we have a certain decrement of the performances when the imbalance increases. Indeed when we have perfectly balanced data the AUPRC is very close to 1, while by increasing the imbalance we have a progressive decrement of the AUPRC to 0.950, 0.900, till to 0.773 when we have a 1 : 1000 imbalance ratio. Nevertheless this decline in performance is relatively small compared to other machine-learning methods.

We can perform the same task using parallel computation. For instance, by using 4 cores with an Intel i7-2670QM CPU, 2.20GHz, we perform a full 10-fold cytogenic band-aware cross-validation using 406 genetic variants known to be associated with Mendelian diseases and 400000 background variants in less than 5 minutes. The best performance boost from the implementation is if we do the training of the partitioning in parallel. So we can set the number of execution slots to 4 using `clsHyperSMURF.setNumExecutionSlots(4);`.

Of course the training and cross-validation functions allow to set also the parameters of the Random Forest ensembles, that constitute the base learners of the hyperSMURF hyper-ensemble, such as the number of decision trees to be used for each Random Forest (`setNumTrees(int num)`) or the number of features to be randomly selected from the set of available input features at each step of the inductive learning of the decision tree (`setNumFeatures(int num)`). The full description of the hyperSMURF class can be found in the *HyperSMURF* Java API <https://javadoc.io/doc/de.charite.compbio/hyperSMURF/>.

References

Examples using the R package

Simple usage examples using synthetic data

Here we introduce some simple usage examples using the generator of synthetic imbalanced data included in the R package. At first we load the library:

```
library(hyperSMURF)
```

Then we construct two imbalanced data sets (training and test set) having both 20 *positive* and 2000 *negative* examples with 10 features (dimension of input data equal to 10 - see the [Reference manual](#) on CRAN for details about the synthetic data generator):

```
train <- imbalanced.data.generator(n.pos=20, n.neg=2000, n.features=10, n.inf.
↪features=3, sd=0.1, seed=1);
test <- imbalanced.data.generator(n.pos=20, n.neg=2000, n.features=10, n.inf.
↪features=3, sd=0.1, seed=2);
```

Then we can train and test the model with the following code:

```
HSmodel <- hyperSMURF.train(train$data, train$label, n.part = 10, fp = 2, ratio = 3);
res <- hyperSMURF.test(test$data, HSmodel);
```

Note that we used 10 partitions of the training data (parameter *n.part* that corresponds to the parameter *n* in the pseudo-code of the algorithm in Supplementary Note 1), a SMOTE oversampling equal to 2 (parameter *fp* corresponding to the *f* parameter in the pseudo-code), and undersampling ratio equal to 3 (parameter *ratio* corresponding to the parameter *m* of the modified second line of the Hy~algorithm in Supplementary Note~1). In other words the negative examples

were partitioned in 10 sets of equal size (200 examples). Then a different RF was trained using: a) the available 20 positive examples, plus the “augmented” 40 synthetic positive examples obtained by SMOTE convex combination of close positive examples, and b) a set of $3 \times 60 = 180$ negative examples randomly extracted from the partition (see Supplementary Note 1). The obtained hyperSMURF model (*HSMModel*), that includes 10 different RF (one for each partition), is finally tested on the test set.

We can easily obtain the confusion matrix:

```
y <- ifelse(test$labels==1,1,0);
pred <- ifelse(res>0.5,1,0);
table(pred,y);
```

y		pred	0	1
	0	1979	1	
	1	21	19	

The accuracy is 0.9891 and the F-score (more informative in this unbalanced context) is 0.6333. Note that with a RF that does not adopt unbalance-aware learning strategies on the same data we obtain significantly worse results in terms of the F-score:

```
library(randomForest);
RF <- randomForest(train$data, train$label);
res <- predict(RF, test$data);
y <- ifelse(test$labels==1,1,0);
pred <- ifelse(res==1,1,0);
table(pred,y);
```

y		pred	0	1
	0	2000	16	
	1	0	4	

The accuracy of the RF is high (0.9930), but the F-score is 0.3333, only about half that of hyperSMURF¹.

To perform a 5 fold CV on a given data set we need only 1 line of R code:

```
res <- hyperSMURF.cv(train$data, train$labels, kk = 5, n.part = 10, fp = 1, ratio = 1);
```

To compute the AUROC and the AUPRC (respectively the area under the ROC curve and the area under the precision/recall curve) we can use the `precrec` package:

```
library(precrec);
labels <- ifelse(train$labels==1,1,0);
digits=4;
sscurves <- evalmod(scores = res, labels = labels);
m<-attr(sscurves,"auc",exact=FALSE);
AUROC <- round(m[1,"aucs"],digits);
AUPRC <- round(m[2,"aucs"],digits);
cat ("AUROC = ", AUROC, "\n", "AUPRC = ", AUPRC, "\n");
AUROC = 0.9972
AUPRC = 0.8540
```

We can also apply the version of hyperSMURF that embeds a feature selection step on the training data to select the features most correlated with the labels:

¹ Note that the results may vary slightly due to the randomization in the algorithm.

```
res <- hyperSMURF.corr.cv.parallel(train$data, train$labels, kk = 5, n.part = 10, fp =
↪ 1, ratio = 1, mtry=3, n.feature = 6);
sscurves <- evalmod(scores = res, labels = labels);
m<-attr(sscurves, "auc", exact=FALSE);
AUROC <- round(m[1, "aucs"], digits);
AUPRC <- round(m[2, "aucs"], digits);
cat ("AUROC = ", AUROC, "\n", "AUPRC = ", AUPRC, "\n");
AUROC = 0.9982
AUPRC = 0.9190
```

Usage examples with genetic data

HyperSMURF was designed to predict rare genomic variants, when the available examples of such variants are substantially less than *background* examples. This is a typical situation with genetic variants. For instance, we have only a small set of available variants known to be associated with Mendelian diseases in non-coding regions (positive examples) against the sea of background variants, i.e. a ratio of about 1 : 36,000 between positive and negative examples [Smedley2016].

Here we show how to use hyperSMURF to detect these rare features using data sets obtained from the original large set of Mendelian data [Smedley2016]. To provide usage examples that do not require more than 1 minute of computation time on a modern desktop computer, we considered data sets downsampled from the original Mendelian data set described in the *mendelian data* section of the main manuscript (this data set includes more than 14 millions of genetic variants). In particular we constructed Mendelian data sets with a progressive larger imbalance between Mendelian associated mutations and background genetic variants. We start with an artificially balanced data set, and then we consider progressively imbalanced data sets with ratio *positive:negative* varying from 1 : 10, to 1 : 100 and 1 : 1000. These data sets are downloadable as compressed *.rda* R objects from <http://homes.di.unimi.it/valentini/DATA/Mendelian>.

The *Mendelian_balanced.rda* file include 3 objects: *m.subset*, that includes the input features of the balanced examples (406 positives and 400 negatives), *labels.subset*, i.e. the corresponding labels, and *folds.subset* a vector with the number of the fold in which each example will be included according to the 10-fold cytoband-aware CV procedure (see Supplementary Note~2). The following lines of code load the data and perform a 10-fold cytoband-aware CV and compute the AUROC and AUPRC:

```
load("Mendelian_balanced.rda");
res <- hyperSMURF.cv(m.subset, factor(labels.subset, levels=c(1,0)), kk = 10, n.part
↪ 2, fp = 0, ratio = 1, k = 5, ntree = 10, mtry = 6, seed = 1, fold.partition =
↪ folds.subset);

sscurves <- evalmod(scores = res, labels = labels.subset);
m<-attr(sscurves, "auc", exact=FALSE);
AUROC <- round(m[1, "aucs"], digits);
AUPRC <- round(m[2, "aucs"], digits);
cat ("AUROC = ", AUROC, "\n", "AUPRC = ", AUPRC, "\n");
AUROC = 0.9903
AUPRC = 0.9893
```

Then we can perform the same computation using the progressively imbalanced data sets:

```
# Imbalance 1:10. about 400 positives and 4000 negative variants
load("Mendelian_1:10.rda");

res <- hyperSMURF.cv(m.subset, factor(labels.subset, levels=c(1,0)), kk = 10, n.part
↪ 5,
fp = 1, ratio = 1, k = 5, ntree = 10, mtry = 6, seed = 1, fold.partition = folds.
↪ subset);
```

```

sscurves <- evalmod(scores = res, labels = labels.subset);
m<-attr(sscurves,"auc",exact=FALSE);
AUROC <- round(m[1,"aucs"],digits);
AUPRC <- round(m[2,"aucs"],digits);
cat ("AUROC = ", AUROC, "\n", "AUPRC = ", AUPRC, "\n");
AUROC = 0.9915
AUPRC = 0.9583

# Imbalance 1:100. about 400 positives and 40000 negative variants
load("Mendelian_1:100.rda");
res <- hyperSMURF.cv(m.subset, factor(labels.subset, levels=c(1,0)), kk = 10, n.part_
↳= 10, fp = 2, ratio = 3, k = 5, ntree = 10, mtry = 6, seed = 1, fold.partition = 
↳folds.subset);

sscurves <- evalmod(scores = res, labels = labels.subset);
m<-attr(sscurves,"auc",exact=FALSE);
AUROC <- round(m[1,"aucs"],digits);
AUPRC <- round(m[2,"aucs"],digits);
cat ("AUROC = ", AUROC, "\n", "AUPRC = ", AUPRC, "\n");
AUROC = 0.9922
AUPRC = 0.9

# Imbalance 1:1000. about 400 positives and 400000 negative variants
load("Mendelian_1:1000.rda");

res <- hyperSMURF.cv(m.subset, factor(labels.subset, levels=c(1,0)), kk = 10, n.part_
↳= 10,
fp = 2, ratio = 3, k = 5, ntree = 10, mtry = 6, seed = 1, fold.partition = folds.
↳subset);

sscurves <- evalmod(scores = res, labels = labels.subset);
m<-attr(sscurves,"auc",exact=FALSE);
AUROC <- round(m[1,"aucs"],digits);
AUPRC <- round(m[2,"aucs"],digits);
cat ("AUROC = ", AUROC, "\n", "AUPRC = ", AUPRC, "\n");
AUROC = 0.9901
AUPRC = 0.7737

```

As we can see, we have a certain decrement of the performances when the imbalance increases. Indeed when we have perfectly balanced data the AUPRC is very close to 1, while by increasing the imbalance we have a progressive decrement of the AUPRC to 0.9583, 0.9000, till to 0.7737 when we have a 1 : 1000 imbalance ratio. Nevertheless this decline in performance is relatively small compared to that of state-of-the-art imbalance-unaware learning methods (see Fig. 5 in the main manuscript).

We can perform the same task using parallel computation. For instance, by using 4 cores with an Intel i7-2670QM CPU, 2.20GHz, less than 1 minute is necessary to perform a full 10-fold cytoband-aware CV using 406 genetic variants known to be associated with Mendelian diseases and 400,000 background variants:

```

res <- hyperSMURF.cv.parallel(m.subset, factor(labels.subset, levels=c(1,0)), kk = 10,
↳ n.part = 10, fp = 2, ratio = 3, k = 5, ntree = 10, mtry = 6, seed = 1, fold.
↳partition = folds.subset, ncores=4);

```

Of course the training and CV functions allow to set also the parameters of the RF ensembles, that constitute the base learners of the hyperSMURF hyper-ensemble, such as the number of decision trees to be used for each RF (parameter *ntree*) or the number of features to be randomly selected from the set of available input features at each step of the inductive learning of the decision tree (parameter *mtry*). The full description of all the parameters and the output of

each function is available in the PDF and HTML documentation included in the hyperSMURF R package.

References

Frequently Asked Questions

Where are the questions?

Right now, there are no frequently asked questions.

Cite hyperSMURF

If you use this algorithm please cite our [Scientific Reports](#) article:

M. Schubach, M. Re, P. N. Robinson, and G. Valentini. (2017). Imbalance-Aware Machine Learning for Predicting Rare and Common Disease-Associated ↪Non-Coding Variants. *Scientific Reports*, 7.

Contributing

Contributions are welcome, and they are greatly appreciated! Every little bit helps, and credit will always be given.

You can contribute in many ways:

Types of Contributions

Report Bugs

Report bugs at <https://github.com/charite/hyperSMURF/issues>

If you are reporting a bug, please include:

- Your operating system name and version.
- Any details about your local setup that might be helpful in troubleshooting.
- Detailed steps to reproduce the bug.

Fix Bugs

Look through the Github issues for bugs. If you want to start working on a bug then please write short message on the issue tracker to prevent duplicate work.

Implement Features

Look through the Github issues for features. If you want to start working on an issue then please write short message on the issue tracker to prevent duplicate work.

Write Documentation

hyperSMURF could always use more documentation, whether as part of the official vcfpy docs, in docstrings, or even on the web in blog posts, articles, and such.

hyperSMURF uses [Sphinx](#) for the user manual (that you are currently reading). See *doc_guidelines* on how the documentation reStructuredText is used. See *doc_setup* on creating a local setup for building the documentation.

Submit Feedback

The best way to send feedback is to file an issue at <https://github.com/charite/hyperSMURF/issues>

If you are proposing a feature:

- Explain in detail how it would work.
- Keep the scope as narrow as possible, to make it easier to implement.
- Remember that this is a volunteer-driven project, and that contributions are welcome :)

Documentation Guidelines

For the documentation, please adhere to the following guidelines:

- Put each sentence on its own line, this makes tracking changes through Git SCM easier.
- Provide hyperlink targets, at least for the first two section levels.
- Use the section structure from below.

```
.. heading_1:
=====
Heading 1
=====

.. heading_2:
-----
Heading 2
-----

.. heading_3:
Heading 3
=====

.. heading_4:
Heading 4
-----

.. heading_5:
Heading 5
```

```
~~~~~

.. heading_6:

Heading 6
:~:~:~:~:~:~:
```

Documentation Setup

For building the documentation, you have to install the Python program Sphinx. This is best done in a virtual environment. The following assumes you have a working Python 3 setup.

Use the following steps for installing Sphinx and the dependencies for building the SIMdrom documentation:

```
$ cd hyperSMURF/manual
$ virtualenv -p python3 .venv
$ source .venv/bin/activate
$ pip install --upgrade -r requirements.txt
```

Use the following for building the documentation. The first two lines is only required for loading the virtualenv. Afterwards, you can always use `make html` for building.

```
$ cd hyperSMURF/manual
$ source .venv/bin/activate
$ make html # rebuild for changed files only
$ make clean && make html # force rebuild
```

Get Started!

Ready to contribute? First, create your Java/Documentation development setup as described in *install_from_source/doc_setup*.

1. Fork the *hyperSMURF* repo on GitHub.
2. Clone your fork locally:

```
$ git clone git@github.com:your_name_here/hyperSMURF.git
```

3. Create a branch for local development:

```
$ git checkout -b name-of-your-bugfix-or-feature
```

Now you can make your changes locally.

5. When you're done making your changes, make sure that the build runs through. For Java:

```
$ mvn package
```

For documentation:

```
$ cd manual && make clean && make html
```

6. Commit your changes and push your branch to GitHub:

```
$ git add .
$ git commit -m "Your detailed description of your changes."
$ git push origin name-of-your-bugfix-or-feature
```

7. Submit a pull request through the GitHub website.

Pull Request Guidelines

Before you submit a pull request, check that it meets these guidelines:

1. The pull request should include tests.
2. If the pull request adds functionality, the docs should be updated.
3. Describe your changes in the `CHANGELOG.md` file.

Authors

in alphabetical order

- Max Schubach

History

```
# hyperSMURF Changelog

## development

## v0.3

* Documentation release
* move java api to javadoc.io
* Upgrading Weka from 3.9. to 3.9.1

### manual

* creating readthedocs manual
* adding examples

## v0.2

* Publication release
```

hyperSMURF License

hyperSMURF is licensed under the GNU GPLv3 license:

```
GNU GENERAL PUBLIC LICENSE
    Version 3, 29 June 2007

Copyright (C) 2007 Free Software Foundation, Inc. <http://fsf.org/>
```


Everyone is permitted to copy and distribute verbatim copies of this license document, but changing it is not allowed.

Preamble

The GNU General Public License is a free, copyleft license for software and other kinds of works.

The licenses for most software and other practical works are designed to take away your freedom to share and change the works. By contrast, the GNU General Public License is intended to guarantee your freedom to share and change all versions of a program--to make sure it remains free software for all its users. We, the Free Software Foundation, use the GNU General Public License for most of our software; it applies also to any other work released this way by its authors. You can apply it to your programs, too.

When we speak of free software, we are referring to freedom, not price. Our General Public Licenses are designed to make sure that you have the freedom to distribute copies of free software (and charge for them if you wish), that you receive source code or can get it if you want it, that you can change the software or use pieces of it in new free programs, and that you know you can do these things.

To protect your rights, we need to prevent others from denying you these rights or asking you to surrender the rights. Therefore, you have certain responsibilities if you distribute copies of the software, or if you modify it: responsibilities to respect the freedom of others.

For example, if you distribute copies of such a program, whether gratis or for a fee, you must pass on to the recipients the same freedoms that you received. You must make sure that they, too, receive or can get the source code. And you must show them these terms so they know their rights.

Developers that use the GNU GPL protect your rights with two steps: (1) assert copyright on the software, and (2) offer you this License giving you legal permission to copy, distribute and/or modify it.

For the developers' and authors' protection, the GPL clearly explains that there is no warranty for this free software. For both users' and authors' sake, the GPL requires that modified versions be marked as changed, so that their problems will not be attributed erroneously to authors of previous versions.

Some devices are designed to deny users access to install or run modified versions of the software inside them, although the manufacturer can do so. This is fundamentally incompatible with the aim of protecting users' freedom to change the software. The systematic pattern of such abuse occurs in the area of products for individuals to use, which is precisely where it is most unacceptable. Therefore, we have designed this version of the GPL to prohibit the practice for those products. If such problems arise substantially in other domains, we stand ready to extend this provision to those domains in future versions of the GPL, as needed to protect the freedom of users.

Finally, every program is threatened constantly by software patents. States should not allow patents to restrict development and use of

software on general-purpose computers, but in those that do, we wish to avoid the special danger that patents applied to a free program could make it effectively proprietary. To prevent this, the GPL assures that patents cannot be used to render the program non-free.

The precise terms and conditions for copying, distribution and modification follow.

TERMS AND CONDITIONS

0. Definitions.

"This License" refers to version 3 of the GNU General Public License.

"Copyright" also means copyright-like laws that apply to other kinds of works, such as semiconductor masks.

"The Program" refers to any copyrightable work licensed under this License. Each licensee is addressed as "you". "Licensees" and "recipients" may be individuals or organizations.

To "modify" a work means to copy from or adapt all or part of the work in a fashion requiring copyright permission, other than the making of an exact copy. The resulting work is called a "modified version" of the earlier work or a work "based on" the earlier work.

A "covered work" means either the unmodified Program or a work based on the Program.

To "propagate" a work means to do anything with it that, without permission, would make you directly or secondarily liable for infringement under applicable copyright law, except executing it on a computer or modifying a private copy. Propagation includes copying, distribution (with or without modification), making available to the public, and in some countries other activities as well.

To "convey" a work means any kind of propagation that enables other parties to make or receive copies. Mere interaction with a user through a computer network, with no transfer of a copy, is not conveying.

An interactive user interface displays "Appropriate Legal Notices" to the extent that it includes a convenient and prominently visible feature that (1) displays an appropriate copyright notice, and (2) tells the user that there is no warranty for the work (except to the extent that warranties are provided), that licensees may convey the work under this License, and how to view a copy of this License. If the interface presents a list of user commands or options, such as a menu, a prominent item in the list meets this criterion.

1. Source Code.

The "source code" for a work means the preferred form of the work for making modifications to it. "Object code" means any non-source form of a work.

A "Standard Interface" means an interface that either is an official standard defined by a recognized standards body, or, in the case of interfaces specified for a particular programming language, one that

is widely used among developers working in that language.

The "System Libraries" of an executable work include anything, other than the work as a whole, that (a) is included in the normal form of packaging a Major Component, but which is not part of that Major Component, and (b) serves only to enable use of the work with that Major Component, or to implement a Standard Interface for which an implementation is available to the public in source code form. A "Major Component", in this context, means a major essential component (kernel, window system, and so on) of the specific operating system (if any) on which the executable work runs, or a compiler used to produce the work, or an object code interpreter used to run it.

The "Corresponding Source" for a work in object code form means all the source code needed to generate, install, and (for an executable work) run the object code and to modify the work, including scripts to control those activities. However, it does not include the work's System Libraries, or general-purpose tools or generally available free programs which are used unmodified in performing those activities but which are not part of the work. For example, Corresponding Source includes interface definition files associated with source files for the work, and the source code for shared libraries and dynamically linked subprograms that the work is specifically designed to require, such as by intimate data communication or control flow between those subprograms and other parts of the work.

The Corresponding Source need not include anything that users can regenerate automatically from other parts of the Corresponding Source.

The Corresponding Source for a work in source code form is that same work.

2. Basic Permissions.

All rights granted under this License are granted for the term of copyright on the Program, and are irrevocable provided the stated conditions are met. This License explicitly affirms your unlimited permission to run the unmodified Program. The output from running a covered work is covered by this License only if the output, given its content, constitutes a covered work. This License acknowledges your rights of fair use or other equivalent, as provided by copyright law.

You may make, run and propagate covered works that you do not convey, without conditions so long as your license otherwise remains in force. You may convey covered works to others for the sole purpose of having them make modifications exclusively for you, or provide you with facilities for running those works, provided that you comply with the terms of this License in conveying all material for which you do not control copyright. Those thus making or running the covered works for you must do so exclusively on your behalf, under your direction and control, on terms that prohibit them from making any copies of your copyrighted material outside their relationship with you.

Conveying under any other circumstances is permitted solely under the conditions stated below. Sublicensing is not allowed; section 10 makes it unnecessary.

3. Protecting Users' Legal Rights From Anti-Circumvention Law.

No covered work shall be deemed part of an effective technological measure under any applicable law fulfilling obligations under article 11 of the WIPO copyright treaty adopted on 20 December 1996, or similar laws prohibiting or restricting circumvention of such measures.

When you convey a covered work, you waive any legal power to forbid circumvention of technological measures to the extent such circumvention is effected by exercising rights under this License with respect to the covered work, and you disclaim any intention to limit operation or modification of the work as a means of enforcing, against the work's users, your or third parties' legal rights to forbid circumvention of technological measures.

4. Conveying Verbatim Copies.

You may convey verbatim copies of the Program's source code as you receive it, in any medium, provided that you conspicuously and appropriately publish on each copy an appropriate copyright notice; keep intact all notices stating that this License and any non-permissive terms added in accord with section 7 apply to the code; keep intact all notices of the absence of any warranty; and give all recipients a copy of this License along with the Program.

You may charge any price or no price for each copy that you convey, and you may offer support or warranty protection for a fee.

5. Conveying Modified Source Versions.

You may convey a work based on the Program, or the modifications to produce it from the Program, in the form of source code under the terms of section 4, provided that you also meet all of these conditions:

- a) The work must carry prominent notices stating that you modified it, and giving a relevant date.
- b) The work must carry prominent notices stating that it is released under this License and any conditions added under section 7. This requirement modifies the requirement in section 4 to "keep intact all notices".
- c) You must license the entire work, as a whole, under this License to anyone who comes into possession of a copy. This License will therefore apply, along with any applicable section 7 additional terms, to the whole of the work, and all its parts, regardless of how they are packaged. This License gives no permission to license the work in any other way, but it does not invalidate such permission if you have separately received it.
- d) If the work has interactive user interfaces, each must display Appropriate Legal Notices; however, if the Program has interactive interfaces that do not display Appropriate Legal Notices, your work need not make them do so.

A compilation of a covered work with other separate and independent works, which are not by their nature extensions of the covered work,

and which are not combined with it such as to form a larger program, in or on a volume of a storage or distribution medium, is called an "aggregate" if the compilation and its resulting copyright are not used to limit the access or legal rights of the compilation's users beyond what the individual works permit. Inclusion of a covered work in an aggregate does not cause this License to apply to the other parts of the aggregate.

6. Conveying Non-Source Forms.

You may convey a covered work in object code form under the terms of sections 4 and 5, provided that you also convey the machine-readable Corresponding Source under the terms of this License, in one of these ways:

- a) Convey the object code in, or embodied in, a physical product (including a physical distribution medium), accompanied by the Corresponding Source fixed on a durable physical medium customarily used for software interchange.
- b) Convey the object code in, or embodied in, a physical product (including a physical distribution medium), accompanied by a written offer, valid for at least three years and valid for as long as you offer spare parts or customer support for that product model, to give anyone who possesses the object code either (1) a copy of the Corresponding Source for all the software in the product that is covered by this License, on a durable physical medium customarily used for software interchange, for a price no more than your reasonable cost of physically performing this conveying of source, or (2) access to copy the Corresponding Source from a network server at no charge.
- c) Convey individual copies of the object code with a copy of the written offer to provide the Corresponding Source. This alternative is allowed only occasionally and noncommercially, and only if you received the object code with such an offer, in accord with subsection 6b.
- d) Convey the object code by offering access from a designated place (gratis or for a charge), and offer equivalent access to the Corresponding Source in the same way through the same place at no further charge. You need not require recipients to copy the Corresponding Source along with the object code. If the place to copy the object code is a network server, the Corresponding Source may be on a different server (operated by you or a third party) that supports equivalent copying facilities, provided you maintain clear directions next to the object code saying where to find the Corresponding Source. Regardless of what server hosts the Corresponding Source, you remain obligated to ensure that it is available for as long as needed to satisfy these requirements.
- e) Convey the object code using peer-to-peer transmission, provided you inform other peers where the object code and Corresponding Source of the work are being offered to the general public at no charge under subsection 6d.

A separable portion of the object code, whose source code is excluded from the Corresponding Source as a System Library, need not be

included in conveying the object code work.

A "User Product" is either (1) a "consumer product", which means any tangible personal property which is normally used for personal, family, or household purposes, or (2) anything designed or sold for incorporation into a dwelling. In determining whether a product is a consumer product, doubtful cases shall be resolved in favor of coverage. For a particular product received by a particular user, "normally used" refers to a typical or common use of that class of product, regardless of the status of the particular user or of the way in which the particular user actually uses, or expects or is expected to use, the product. A product is a consumer product regardless of whether the product has substantial commercial, industrial or non-consumer uses, unless such uses represent the only significant mode of use of the product.

"Installation Information" for a User Product means any methods, procedures, authorization keys, or other information required to install and execute modified versions of a covered work in that User Product from a modified version of its Corresponding Source. The information must suffice to ensure that the continued functioning of the modified object code is in no case prevented or interfered with solely because modification has been made.

If you convey an object code work under this section in, or with, or specifically for use in, a User Product, and the conveying occurs as part of a transaction in which the right of possession and use of the User Product is transferred to the recipient in perpetuity or for a fixed term (regardless of how the transaction is characterized), the Corresponding Source conveyed under this section must be accompanied by the Installation Information. But this requirement does not apply if neither you nor any third party retains the ability to install modified object code on the User Product (for example, the work has been installed in ROM).

The requirement to provide Installation Information does not include a requirement to continue to provide support service, warranty, or updates for a work that has been modified or installed by the recipient, or for the User Product in which it has been modified or installed. Access to a network may be denied when the modification itself materially and adversely affects the operation of the network or violates the rules and protocols for communication across the network.

Corresponding Source conveyed, and Installation Information provided, in accord with this section must be in a format that is publicly documented (and with an implementation available to the public in source code form), and must require no special password or key for unpacking, reading or copying.

7. Additional Terms.

"Additional permissions" are terms that supplement the terms of this License by making exceptions from one or more of its conditions. Additional permissions that are applicable to the entire Program shall be treated as though they were included in this License, to the extent that they are valid under applicable law. If additional permissions apply only to part of the Program, that part may be used separately under those permissions, but the entire Program remains governed by this License without regard to the additional permissions.

When you convey a copy of a covered work, you may at your option remove any additional permissions from that copy, or from any part of it. (Additional permissions may be written to require their own removal in certain cases when you modify the work.) You may place additional permissions on material, added by you to a covered work, for which you have or can give appropriate copyright permission.

Notwithstanding any other provision of this License, for material you add to a covered work, you may (if authorized by the copyright holders of that material) supplement the terms of this License with terms:

- a) Disclaiming warranty or limiting liability differently from the terms of sections 15 and 16 of this License; or
- b) Requiring preservation of specified reasonable legal notices or author attributions in that material or in the Appropriate Legal Notices displayed by works containing it; or
- c) Prohibiting misrepresentation of the origin of that material, or requiring that modified versions of such material be marked in reasonable ways as different from the original version; or
- d) Limiting the use for publicity purposes of names of licensors or authors of the material; or
- e) Declining to grant rights under trademark law for use of some trade names, trademarks, or service marks; or
- f) Requiring indemnification of licensors and authors of that material by anyone who conveys the material (or modified versions of it) with contractual assumptions of liability to the recipient, for any liability that these contractual assumptions directly impose on those licensors and authors.

All other non-permissive additional terms are considered "further restrictions" within the meaning of section 10. If the Program as you received it, or any part of it, contains a notice stating that it is governed by this License along with a term that is a further restriction, you may remove that term. If a license document contains a further restriction but permits relicensing or conveying under this License, you may add to a covered work material governed by the terms of that license document, provided that the further restriction does not survive such relicensing or conveying.

If you add terms to a covered work in accord with this section, you must place, in the relevant source files, a statement of the additional terms that apply to those files, or a notice indicating where to find the applicable terms.

Additional terms, permissive or non-permissive, may be stated in the form of a separately written license, or stated as exceptions; the above requirements apply either way.

8. Termination.

You may not propagate or modify a covered work except as expressly provided under this License. Any attempt otherwise to propagate or

modify it is void, and will automatically terminate your rights under this License (including any patent licenses granted under the third paragraph of section 11).

However, if you cease all violation of this License, then your license from a particular copyright holder is reinstated (a) provisionally, unless and until the copyright holder explicitly and finally terminates your license, and (b) permanently, if the copyright holder fails to notify you of the violation by some reasonable means prior to 60 days after the cessation.

Moreover, your license from a particular copyright holder is reinstated permanently if the copyright holder notifies you of the violation by some reasonable means, this is the first time you have received notice of violation of this License (for any work) from that copyright holder, and you cure the violation prior to 30 days after your receipt of the notice.

Termination of your rights under this section does not terminate the licenses of parties who have received copies or rights from you under this License. If your rights have been terminated and not permanently reinstated, you do not qualify to receive new licenses for the same material under section 10.

9. Acceptance Not Required for Having Copies.

You are not required to accept this License in order to receive or run a copy of the Program. Ancillary propagation of a covered work occurring solely as a consequence of using peer-to-peer transmission to receive a copy likewise does not require acceptance. However, nothing other than this License grants you permission to propagate or modify any covered work. These actions infringe copyright if you do not accept this License. Therefore, by modifying or propagating a covered work, you indicate your acceptance of this License to do so.

10. Automatic Licensing of Downstream Recipients.

Each time you convey a covered work, the recipient automatically receives a license from the original licensors, to run, modify and propagate that work, subject to this License. You are not responsible for enforcing compliance by third parties with this License.

An "entity transaction" is a transaction transferring control of an organization, or substantially all assets of one, or subdividing an organization, or merging organizations. If propagation of a covered work results from an entity transaction, each party to that transaction who receives a copy of the work also receives whatever licenses to the work the party's predecessor in interest had or could give under the previous paragraph, plus a right to possession of the Corresponding Source of the work from the predecessor in interest, if the predecessor has it or can get it with reasonable efforts.

You may not impose any further restrictions on the exercise of the rights granted or affirmed under this License. For example, you may not impose a license fee, royalty, or other charge for exercise of rights granted under this License, and you may not initiate litigation (including a cross-claim or counterclaim in a lawsuit) alleging that any patent claim is infringed by making, using, selling, offering for

sale, or importing the Program or any portion of it.

11. Patents.

A "contributor" is a copyright holder who authorizes use under this License of the Program or a work on which the Program is based. The work thus licensed is called the contributor's "contributor version".

A contributor's "essential patent claims" are all patent claims owned or controlled by the contributor, whether already acquired or hereafter acquired, that would be infringed by some manner, permitted by this License, of making, using, or selling its contributor version, but do not include claims that would be infringed only as a consequence of further modification of the contributor version. For purposes of this definition, "control" includes the right to grant patent sublicenses in a manner consistent with the requirements of this License.

Each contributor grants you a non-exclusive, worldwide, royalty-free patent license under the contributor's essential patent claims, to make, use, sell, offer for sale, import and otherwise run, modify and propagate the contents of its contributor version.

In the following three paragraphs, a "patent license" is any express agreement or commitment, however denominated, not to enforce a patent (such as an express permission to practice a patent or covenant not to sue for patent infringement). To "grant" such a patent license to a party means to make such an agreement or commitment not to enforce a patent against the party.

If you convey a covered work, knowingly relying on a patent license, and the Corresponding Source of the work is not available for anyone to copy, free of charge and under the terms of this License, through a publicly available network server or other readily accessible means, then you must either (1) cause the Corresponding Source to be so available, or (2) arrange to deprive yourself of the benefit of the patent license for this particular work, or (3) arrange, in a manner consistent with the requirements of this License, to extend the patent license to downstream recipients. "Knowingly relying" means you have actual knowledge that, but for the patent license, your conveying the covered work in a country, or your recipient's use of the covered work in a country, would infringe one or more identifiable patents in that country that you have reason to believe are valid.

If, pursuant to or in connection with a single transaction or arrangement, you convey, or propagate by procuring conveyance of, a covered work, and grant a patent license to some of the parties receiving the covered work authorizing them to use, propagate, modify or convey a specific copy of the covered work, then the patent license you grant is automatically extended to all recipients of the covered work and works based on it.

A patent license is "discriminatory" if it does not include within the scope of its coverage, prohibits the exercise of, or is conditioned on the non-exercise of one or more of the rights that are specifically granted under this License. You may not convey a covered work if you are a party to an arrangement with a third party that is in the business of distributing software, under which you make payment

to the third party based on the extent of your activity of conveying the work, and under which the third party grants, to any of the parties who would receive the covered work from you, a discriminatory patent license (a) in connection with copies of the covered work conveyed by you (or copies made from those copies), or (b) primarily for and in connection with specific products or compilations that contain the covered work, unless you entered into that arrangement, or that patent license was granted, prior to 28 March 2007.

Nothing in this License shall be construed as excluding or limiting any implied license or other defenses to infringement that may otherwise be available to you under applicable patent law.

12. No Surrender of Others' Freedom.

If conditions are imposed on you (whether by court order, agreement or otherwise) that contradict the conditions of this License, they do not excuse you from the conditions of this License. If you cannot convey a covered work so as to satisfy simultaneously your obligations under this License and any other pertinent obligations, then as a consequence you may not convey it at all. For example, if you agree to terms that obligate you to collect a royalty for further conveying from those to whom you convey the Program, the only way you could satisfy both those terms and this License would be to refrain entirely from conveying the Program.

13. Use with the GNU Affero General Public License.

Notwithstanding any other provision of this License, you have permission to link or combine any covered work with a work licensed under version 3 of the GNU Affero General Public License into a single combined work, and to convey the resulting work. The terms of this License will continue to apply to the part which is the covered work, but the special requirements of the GNU Affero General Public License, section 13, concerning interaction through a network will apply to the combination as such.

14. Revised Versions of this License.

The Free Software Foundation may publish revised and/or new versions of the GNU General Public License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns.

Each version is given a distinguishing version number. If the Program specifies that a certain numbered version of the GNU General Public License "or any later version" applies to it, you have the option of following the terms and conditions either of that numbered version or of any later version published by the Free Software Foundation. If the Program does not specify a version number of the GNU General Public License, you may choose any version ever published by the Free Software Foundation.

If the Program specifies that a proxy can decide which future versions of the GNU General Public License can be used, that proxy's public statement of acceptance of a version permanently authorizes you to choose that version for the Program.

Later license versions may give you additional or different

permissions. However, no additional obligations are imposed on any author or copyright holder as a result of your choosing to follow a later version.

15. Disclaimer of Warranty.

THERE IS NO WARRANTY FOR THE PROGRAM, TO THE EXTENT PERMITTED BY APPLICABLE LAW. EXCEPT WHEN OTHERWISE STATED IN WRITING THE COPYRIGHT HOLDERS AND/OR OTHER PARTIES PROVIDE THE PROGRAM "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE PROGRAM IS WITH YOU. SHOULD THE PROGRAM PROVE DEFECTIVE, YOU ASSUME THE COST OF ALL NECESSARY SERVICING, REPAIR OR CORRECTION.

16. Limitation of Liability.

IN NO EVENT UNLESS REQUIRED BY APPLICABLE LAW OR AGREED TO IN WRITING WILL ANY COPYRIGHT HOLDER, OR ANY OTHER PARTY WHO MODIFIES AND/OR CONVEYS THE PROGRAM AS PERMITTED ABOVE, BE LIABLE TO YOU FOR DAMAGES, INCLUDING ANY GENERAL, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OR INABILITY TO USE THE PROGRAM (INCLUDING BUT NOT LIMITED TO LOSS OF DATA OR DATA BEING RENDERED INACCURATE OR LOSSES SUSTAINED BY YOU OR THIRD PARTIES OR A FAILURE OF THE PROGRAM TO OPERATE WITH ANY OTHER PROGRAMS), EVEN IF SUCH HOLDER OR OTHER PARTY HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

17. Interpretation of Sections 15 and 16.

If the disclaimer of warranty and limitation of liability provided above cannot be given local legal effect according to their terms, reviewing courts shall apply local law that most closely approximates an absolute waiver of all civil liability in connection with the Program, unless a warranty or assumption of liability accompanies a copy of the Program in return for a fee.

END OF TERMS AND CONDITIONS

How to Apply These Terms to Your New Programs

If you develop a new program, and you want it to be of the greatest possible use to the public, the best way to achieve this is to make it free software which everyone can redistribute and change under these terms.

To do so, attach the following notices to the program. It is safest to attach them to the start of each source file to most effectively state the exclusion of warranty; and each file should have at least the "copyright" line and a pointer to where the full notice is found.

```
<one line to give the program's name and a brief idea of what it does.>
Copyright (C) <year> <name of author>
```

```
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 3 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, see <http://www.gnu.org/licenses/>.

Also add information on how to contact you by electronic and paper mail.

If the program does terminal interaction, make it output a short notice like this when it starts in an interactive mode:

```
<program> Copyright (C) <year> <name of author>
This program comes with ABSOLUTELY NO WARRANTY; for details type `show w'.
This is free software, and you are welcome to redistribute it
under certain conditions; type `show c' for details.
```

The hypothetical commands `show w' and `show c' should show the appropriate parts of the General Public License. Of course, your program's commands might be different; for a GUI interface, you would use an "about box".

You should also get your employer (if you work as a programmer) or school, if any, to sign a "copyright disclaimer" for the program, if necessary. For more information on this, and how to apply and follow the GNU GPL, see <http://www.gnu.org/licenses/>.

The GNU General Public License does not permit incorporating your program into proprietary programs. If your program is a subroutine library, you may consider it more useful to permit linking proprietary applications with the library. If this is what you want to do, use the GNU Lesser General Public License instead of this License. But first, please read <http://www.gnu.org/philosophy/why-not-lgpl.html>.

CHAPTER 2

Indices and tables

- `genindex`
- `modindex`
- `search`

Bibliography

- [Smedley2016] Smedley, Damian, et al. "A whole-genome analysis framework for effective identification of pathogenic regulatory variants in Mendelian disease." *The American Journal of Human Genetics* 99.3 (2016): 595-606.
- [Smedley2016] Smedley, Damian, et al. "A whole-genome analysis framework for effective identification of pathogenic regulatory variants in Mendelian disease." *The American Journal of Human Genetics* 99.3 (2016): 595-606.