1. Введение

Здесь надо простыми словами сказать, кому и почему это надо.

Этот пункт написать труднее всего, так что пока не заморачивайтесь.

2. Смежные исследования

Что делают люди вокруг вашей задачи

Это тоже пока не надо

2. Постановка задачи

Провести сравнительную оценку методов машинного обучения на примере сравнения методов бинарной классификации по теме «рак легких, отличие аденокарциномы от мезотелиомы».

Исследуемые методы машинного обучения:

- Байесовская логистическая регрессия
- Конъюнктивный метод
- Дискриминационная полиномиальная наивная байесовская классификация
- Решающий пень
- JRip
- Метод локального взвешивания
- Сеть радиально-базисных функций
- Дерево решений
- Случайный лес
- Линейная логистическая регрессионная модель
- Перцептрон
- Метод ближайших соседей
- Boosting

Исследование производится при помощи оценивания характеристик ROC-кривых (зависимости верно-положительно классифицированных от ложно-положительно классифицированных результатов).

3. Benchmark (Методы исследования)

Датасет по выбранной теме был взят с сайта datam.i2r.a-star.edu.sg. Датасет содержит 12533 атрибута, описывающих различные гены. Выборка состоит из 149 записей. Датасеты обрабатывались автоматически, с помощью пакета машинного обучения weka для языка java. Данный пакет был выбрал всвязи с хорошим знанием и опытом использования последнего. Приложение было написано с использованием системы сборки gradle, с помощью которого пакет машинного обучения и библиотека для визуализации графиков были интегрированы в конечную программу. Реализации всех исследуемых алгоритмов представлены в выбранном пакете. Собственноручно было написано приложение, которое считывает информацию из датасета, анализирует его с использованием выбранных алгоритмов, и выводит ROC кривые на общих графиках. Данные для построения ROC-кривых, вывода их характеристик, а также вычисления коэффициента Мэттьюса получены с помощью используемого пакета.

При запуске программы открывается форма, представленная на рисунке 1. Изначально на графике нет ни одногой кривой. В процессе обсчета датасета (расчет выполняется многопоточно для более эфективного использования системных ресурсов), кривые добавляются на график и результаты расчетов выводятся в сводную таблицу. Имеется возможность включения/отключения уже просчитанных кривых двойным нажатием по строке с ее результатами в сводной таблице. Также имеется возможность приблизить/отдалить часть графика с помощью выделения мышью (отдаление производится через контекстное меню с помощью нажатия правой кнопкой мыши по графику).

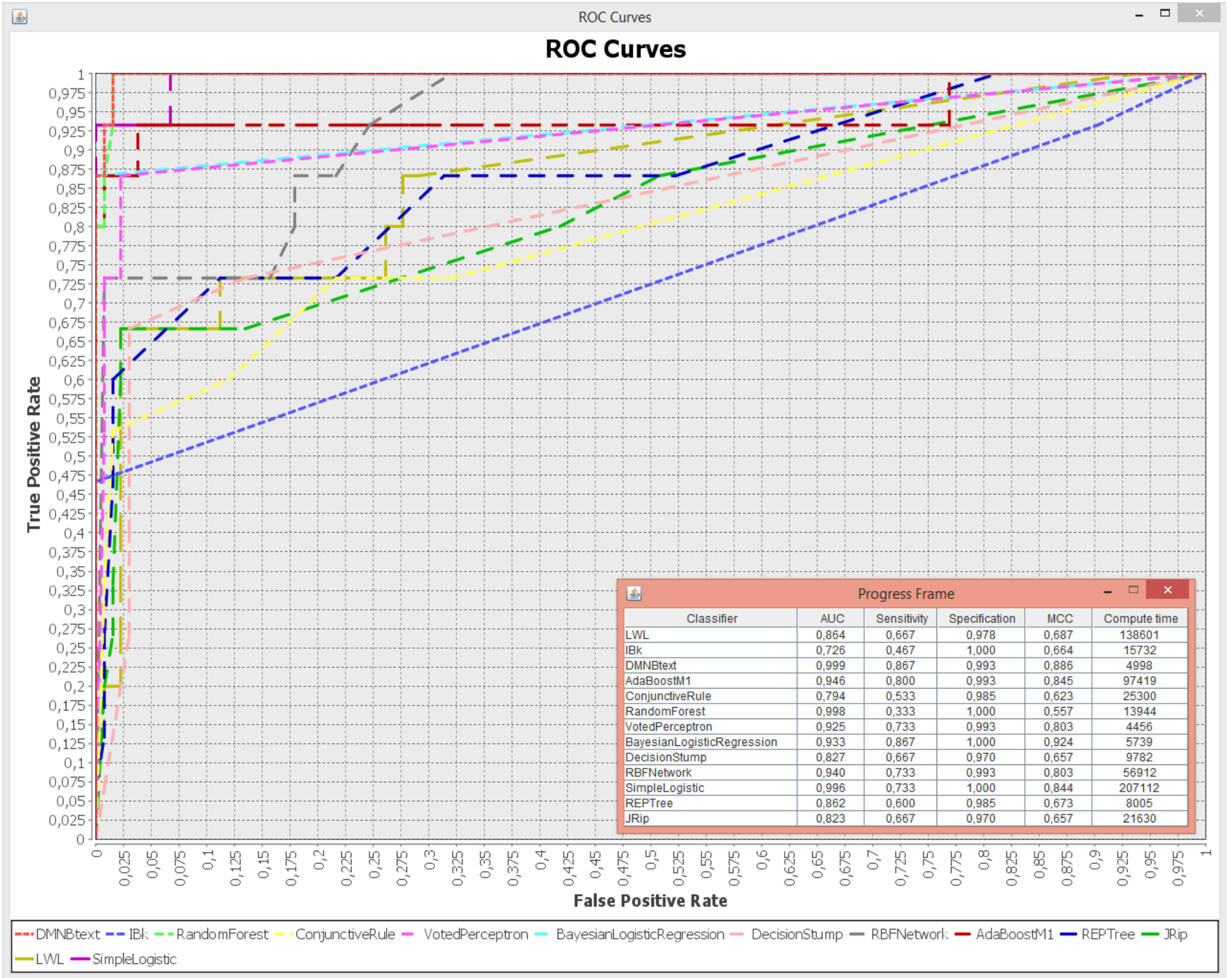


Рис. 1. Общий вид интерфейса программы

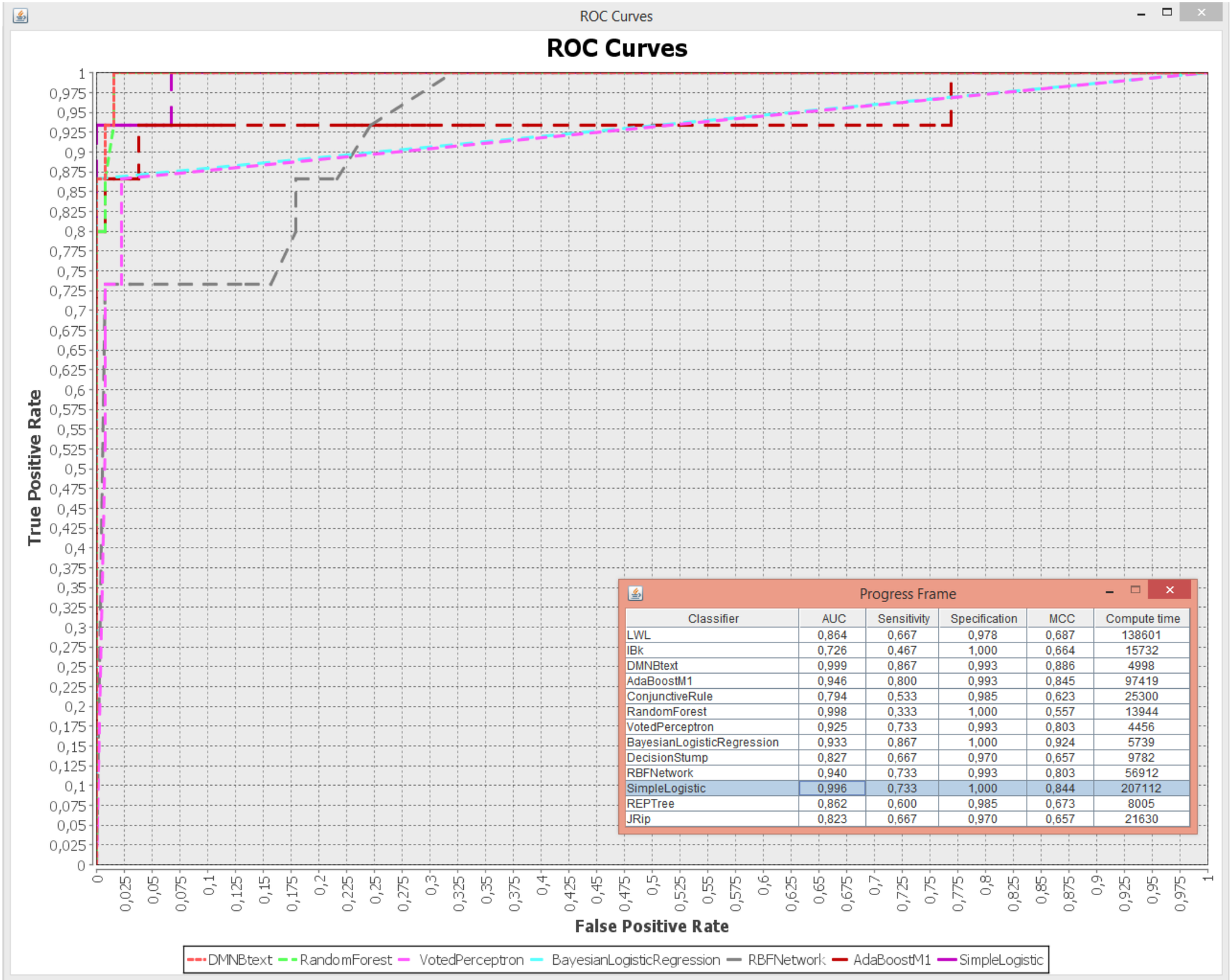
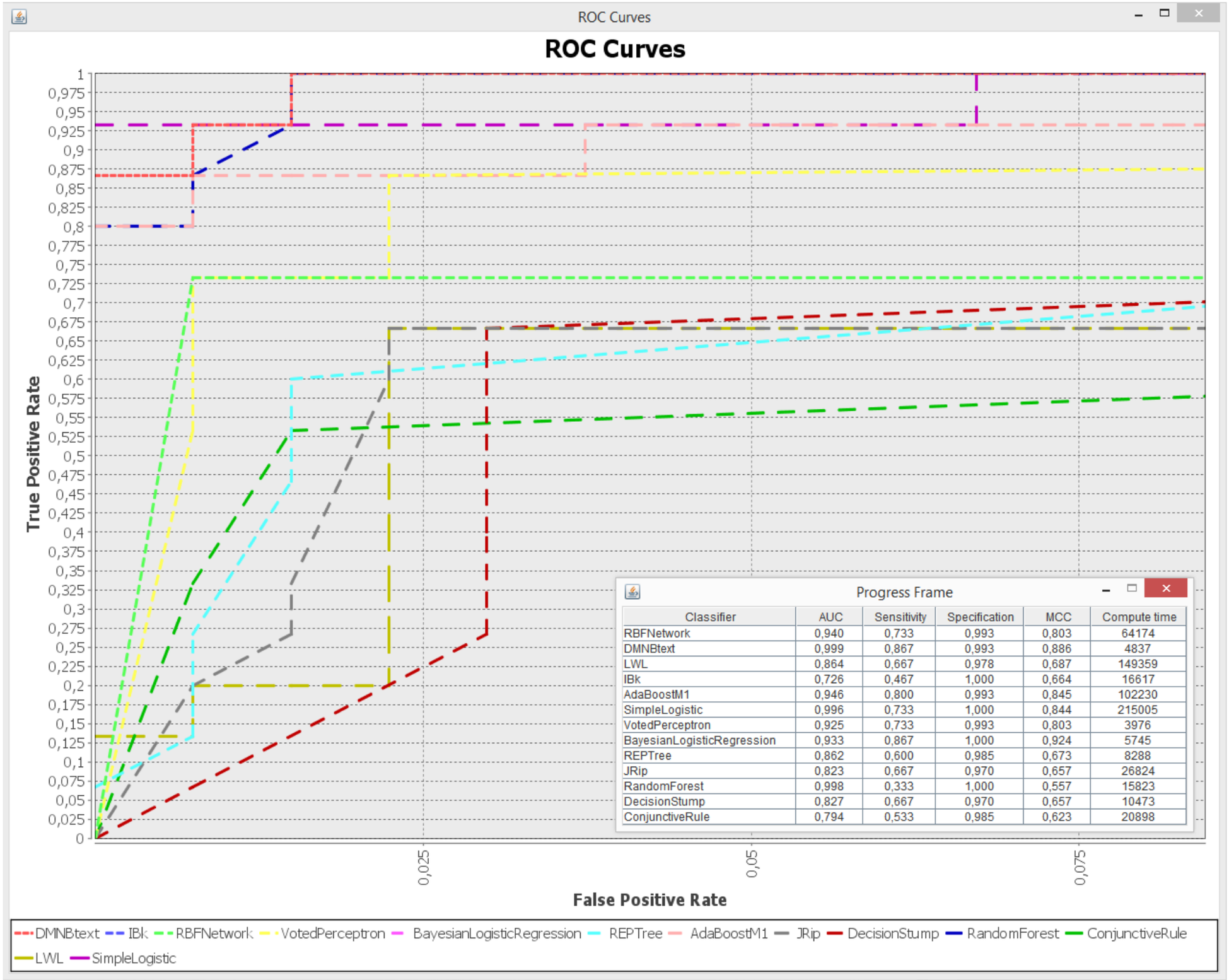


Рис. 2. Пример отключения показа некоторых кривых



Progress Frame

Classifier	AUC	Sensitivity	Specification	MCC	Compute time
RBFNetwork	0,940	0,733	0,993	0,803	64174
DMNBtext	0,999	0,867	0,993	0,886	4837
LWL	0,864	0,667	0,978	0,687	149359
IBk	0,726	0,467	1,000	0,664	16617
AdaBoostM1	0,946	0,800	0,993	0,845	102230
SimpleLogistic	0,996	0,733	1,000	0,844	215005
VotedPerceptron	0,925	0,733	0,993	0,803	3976
BayesianLogisticRegression	0,933	0,867	1,000	0,924	5745
REPTree	0,862	0,600	0,985	0,673	8288
JRip	0,823	0,667	0,970	0,657	26824
RandomForest	0,998	0,333	1,000	0,557	15823
DecisionStump	0,827	0,667	0,970	0,657	10473
ConjunctiveRule	0,794	0,533	0,985	0,623	20898

Рис. 3. Пример увеличения части графика

4. Метрики

Одним из критериев оценки была выбрана площадь под графиком ROC-кривой. Ввиду того, что диапазон допустимых значений по обоим осям лежит на замкнутом интервале [0; 1], то максимальным значением площади под графиком ROC-кривой может быть 1, что будет соответствовать «идеальному» классифицированию. Тем самым, чем больше данное значение для конкретного метода обучения, тем лучше. При этом значение меньшее 0,5 является неприемлемым (значение 0,5 является признаком случайных гаданий, меньшие – о преобладающем количестве ошибочных классификаций)

Вторым методом оценки является корреляционный коэффициент Мэттьюса, который, основываясь на количестве верно – положительных, ложно – положительных, верно – отрицательных и ложно – отрицательных классификаций, показывает качество классификации выбраненного метода по шкале от -1 до 1, где 1 соответствует «идеальному» методу, 0 – случайному гаданию и -1 - постоянным ошибкам классификации.

5. Результаты и обсуждение

ROC – кривая отображает соотношение между долей объектов от общего количества носителей признака, верно классифицированных, как несущих признак, (True positive rate, TPR) и долей объектов от общего количества объектов, не несущих признака, ошибочно классифицированных, как несущих признак (False positive rate, FPR).

Величина TPR называется чувствительностью (sensivity), она отложена по оси Oy на приведенных графиках. Величина (1 – FPR) называется специфичностью (specificity), она отложена по оси Ox на приведенных графиках.

ROC – кривая получается следующим образом:

Предполагается, что у классификатора имеется некоторый параметр, варьируя который, мы будем получать то или иное разбиение на два класса. Этот параметр называют порогом, или точкой отсечения. Начальное значение порога принимается равным нулю. Максимальное возможное значение равно 1. Для каждого значения порога отсечения, которое меняется от 0 до 1 (новое значение порога устанавливается, если рейтинг (вероятность верной классификации) классификации на соответствующей итерации превышает заданное значение порога), рассчитываются значения чувствительности и специфичности. По ним строится график зависимости: по оси Y откладывается чувствительность Se, по оси X – (100%–Sp). В результате получается неубывающая ломаная линия, называющаяся ROC-кривой. Количество изломов это кривой при описанном методе изменения порога зависит исключительно от величины рейтинга классификатора на каждой итерации.

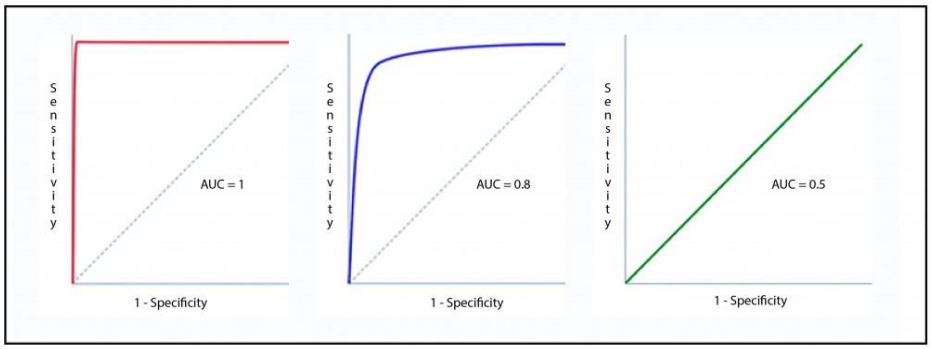


Рис. 4. Пример различных вариантов ROC-кривых

На рисунке 4 приведены три варианта ROC-кривых (слева направо): идеальный случай (AUC = 1б идеально точное предсказание), общий случай (0.5 < AUC < 1), худший случай (AUC = 0.5, случайные гадания).

Пересечение методов из статьи с методами в программе

Название метода в статье	Название метода в программе
Boosting	AdaBoostM1
Decision trees	REPTree
k-nearest neighbor	IBk
LASSO	SimpleLogistic
Random forests	RandomForest

Характеристики исследуемых методов

Название метода	Area Under Curve	Sensitivity	Specification	MCC
LWL	0,864	0,667	0,978	0,687
IBk	0,726	0,467	1,000	0,664
DMNBtext	0,999	0,867	0,993	0,886
AdaBoostM1	0,946	0,800	0,993	0,845
ConjunctiveRule	0,794	0,533	0,985	0,623
RandomForest	0,998	0,333	1,000	0,557
VotedPerceptron	0,925	0,733	0,993	0,803
BayesianLogisticRegression	0,933	0,867	1,000	0,924
DecisionStump	0,827	0,667	0,970	0,657
RBFNetwork	0,940	0,733	0,993	0,803
SimpleLogistic	0,996	0,733	1,000	0,844
REPTree	0,862	0,600	0,985	0,673
JRip	0,823	0,667	0,970	0,657

Сравнение характеристик пересекающихся методов

Название метода	Area Under Curve	Sensitivity	Specification	MCC
Boosting (статья)	0,860	0,860	0,860	0,720
AdaBoostM1 (программа)	0,946	0,800	0,993	0,845
Decision trees (статья)	0,730	0,820	0,660	0,480
REPTree (программа)	0,862	0,600	0,985	0,673
k-nearest neighbor (статья)	0,720	0,750	0,620	0,370
IBk (программа)	0,726	0,467	1,000	0,664
LASSO (статья)	0,910	0,930	0,830	0,750
SimpleLogistic (программа)	0,996	0,733	1,000	0,844
Random forests (статья)	0,850	0,860	0,830	0,680
RandomForest (программа)	0,998	0,333	1,000	0,557

Сравнивая результаты, полученные в статье с результатами, полученными опытным путем, по тому же набору методов, можно заметить, что в значения AUC и MCC, полученные опытным путем больше, чем значения этих параметров в статье.

Анализируя все выбранные методы по двум выбранным критериям оценки, хорошим результатом можно считать методы AdaBoostM1 (0,946 AUC и 0.845 MCC), BayesianLogisticRegression (0.933 AUC и 0.924 MCC), DMNBtext (0.999 AUC и 0.886 MCC), VotedPerceptron (0.925 AUC и 0.803 MCC), RBFNetwork (0.940 AUC и 0.803 MCC), SimpleLogistic (0.996 AUC и 0.844 MCC).

6. Приложения

6.1. Файл для сборки

```
apply plugin: 'java'

group = 'ru.ifmo.itip.trofiv'
compileJava.options.encoding = 'UTF-8'
sourceCompatibility = 1.8
targetCompatibility = 1.8
version = '1.0'

repositories {
    mavenLocal()
    mavenCentral()
}

jar {
    baseName = 'Lab03'
    version = '1.0'
    manifest {
        attributes 'Main-Class': 'ru.ifmo.itip.trofiv.Main'
    }
    from {
        configurations.compile.collect { it.isDirectory() ? it : zipTree(it) }
    }
}

dependencies {
    compile(
            [group: 'nz.ac.waikato.cms.weka', name: 'weka-stable', version: '3.6.12'],
            [group: 'com.google.guava', name: 'guava', version: '18.0'],
            [group: 'jfree', name: 'jfreechart', version: '1.0.13']
    )
}
```

6.2. Код программы

```
package ru.ifmo.itip.trofiv;

import com.google.common.primitives.Doubles;
import org.jfree.chart.ChartFactory;
import org.jfree.chart.ChartPanel;
import org.jfree.chart.JFreeChart;
import org.jfree.chart.axis.NumberAxis;
import org.jfree.chart.axis.NumberTickUnit;
import org.jfree.chart.plot.PlotOrientation;
import org.jfree.chart.plot.XYPlot;
import org.jfree.chart.renderer.xy.XYItemRenderer;
import org.jfree.data.xy.DefaultXYDataset;
import weka.classifiers.Classifier;
import weka.classifiers.Evaluation;
import weka.classifiers.bayes.BayesianLogisticRegression;
import weka.classifiers.bayes.DMNBtext;
import weka.classifiers.evaluation.ThresholdCurve;
import weka.classifiers.functions.RBFNetwork;
import weka.classifiers.functions.SimpleLogistic;
import weka.classifiers.functions.VotedPerceptron;
import weka.classifiers.lazy.IBk;
import weka.classifiers.lazy.LWL;
import weka.classifiers.meta.AdaBoostM1;
import weka.classifiers.rules.ConjunctiveRule;
import weka.classifiers.rules.JRip;
import weka.classifiers.trees.DecisionStump;
import weka.classifiers.trees.REPTree;
import weka.classifiers.trees.RandomForest;
import weka.core.Instances;

import javax.swing.*;
import javax.swing.table.*;
import java.awt.*;
import java.awt.event.MouseAdapter;
import java.awt.event.MouseEvent;
import java.awt.event.WindowAdapter;
import java.awt.event.WindowEvent;
import java.io.BufferedReader;
import java.io.FileNotFoundException;
import java.io.FileReader;
import java.io.IOException;
import java.util.*;
import java.util.List;
import java.util.concurrent.*;

/**
 * @author vladislav.trofimov@emc.com
 */
public class Main {

    private static final String IN_QUEUE = "In queue";
    private static final String INPUT_FILE_NAME = "data.arff";
    private static final JTable TABLE;
    private static final List<Classifier> CLASSIFIERS;
    private static final Map<Integer, Integer> COMPUTED = new ConcurrentHashMap<>();
    private static final Map<Integer, Boolean> VISIBLE = new ConcurrentHashMap<>();
    private static final DefaultXYDataset DATASET = new DefaultXYDataset();
    private static final String ROC_CURVES = "ROC Curves";
    private static final JFreeChart CHART = ChartFactory.createXYLineChart(
            ROC_CURVES, "False Positive Rate", "True Positive Rate",
            DATASET, PlotOrientation.VERTICAL, true, true, false);

    static {
        CLASSIFIERS = new CopyOnWriteArrayList<>();
        CLASSIFIERS.add(new AdaBoostM1());
        CLASSIFIERS.add(new BayesianLogisticRegression());
        CLASSIFIERS.add(new ConjunctiveRule());
        CLASSIFIERS.add(new DMNBtext());
        CLASSIFIERS.add(new DecisionStump());
        CLASSIFIERS.add(new JRip());
        CLASSIFIERS.add(new LWL());
        CLASSIFIERS.add(new RBFNetwork());
        CLASSIFIERS.add(new REPTree());
        CLASSIFIERS.add(new RandomForest());
        CLASSIFIERS.add(new SimpleLogistic());
        CLASSIFIERS.add(new IBk());
        CLASSIFIERS.add(new VotedPerceptron());
        Collections.shuffle(CLASSIFIERS);
        final DefaultTableModel model = new DefaultTableModel(new String[]{
                "Classifier", "AUC", "Sensitivity", "Specification", "MCC", "Compute time"
        }, CLASSIFIERS.size()) {
            @Override
            public boolean isCellEditable(final int row, final int column) {
                return false;
            }
        };
        TABLE = new JTable(model);
        final XYPlot plot = (XYPlot) CHART.getPlot();
        //noinspection MagicNumber
        plot.setBackgroundPaint(new Color(235, 235, 235));
        plot.setRangeGridlinePaint(Color.DARK GRAY);
        plot.setDomainGridlinePaint(Color.DARK GRAY);
        final NumberAxis domain = (NumberAxis) plot.getDomainAxis();
        domain.setRange(0.00, 1.00);
        //noinspection MagicNumber
        domain.setTickUnit(new NumberTickUnit(0.025));
        domain.setVerticalTickLabels(true);
```

```
        final NumberAxis range = (NumberAxis) plot.getRangeAxis();
        range.setRange(0.0, 1.0);
        //noinspection MagicNumber
        range.setTickUnit(new NumberTickUnit(0.025));
        plot.setDomainGridlinesVisible(true);
    }

    public static void main(final String[] args) {
        final Instances data = readData();
        if (data != null) {
            displayMainFrame();
            displayProgressFrame();
            startClassifying(data);
        }
    }

    private static void startClassifying(final Instances data) {
        final int n = Runtime.getRuntime().availableProcessors();
        final ExecutorService executor = Executors.newFixedThreadPool(n);
        final Collection<Future> futures = new LinkedList<>();
        for (final Classifier classifier : CLASSIFIERS) {
            try {
                //noinspection ObjectAllocationInLoop
                final Runnable task = new ClassifierThread(classifier, data);
                futures.add(executor.submit(task));
            } catch (Exception e) {
                e.printStackTrace(System.out);
            }
        }
        executor.shutdown();
        for (final Future future : futures) {
            try {
                future.get();
            } catch (CancellationException e) {
                System.out.println("Computation canceled");
                e.printStackTrace(System.out);
            } catch (InterruptedException e) {
                System.out.println("Computation interrupted");
                e.printStackTrace(System.out);
            } catch (ExecutionException e) {
                System.out.println("Computation failed");
                e.printStackTrace(System.out);
            }
        }
        System.out.println("Classifying complete");
    }

    private static void addPlot(final ClassifierResults data, final Classifier classifier, final long time) {
        final double[][] points = {
                prepareArrayForPlot(data.getFalsePositives()), prepareArrayForPlot(data.getTruePositives())};
        DATASET.addSeries(classifier.getClass().getSimpleName(), points);
        final int plotNumber = DATASET.getSeriesCount() - 1;
        final XYItemRenderer renderer = ((XYPlot) CHART.getPlot()).getRenderer();
        renderer.setSeriesStroke(plotNumber, getStroke(plotNumber + 2));
        CHART.fireChartChanged();
        TABLE.setValueAt(String.format("%5.3f",
                data.getAreaUnderCurve()), CLASSIFIERS.indexOf(classifier), 1);
        TABLE.setValueAt(String.format("%5.3f",
                data.getTruePositivesRate()), CLASSIFIERS.indexOf(classifier), 2);
        TABLE.setValueAt(String.format("%5.3f",
                data.getTrueNegativesRate()), CLASSIFIERS.indexOf(classifier), 3);
        TABLE.setValueAt(String.format("%5.3f",
                data.getMatthewsCorrelationCoefficient()), CLASSIFIERS.indexOf(classifier), 4);
        TABLE.setValueAt(Long.valueOf(time).toString(), CLASSIFIERS.indexOf(classifier), 5);
        synchronized (COMPUTED) {
            COMPUTED.put(CLASSIFIERS.indexOf(classifier), COMPUTED.size());
        }
    }

    private static void displayMainFrame() {
        final ChartPanel panel = new ChartPanel(CHART);
        final JFrame jf = new JFrame(ROC_CURVES);
        //noinspection MagicNumber
        jf.setSize(1280, 1024);
        jf.getContentPane().setLayout(new BorderLayout());
        jf.getContentPane().add(panel, BorderLayout.CENTER);
        final Dimension dim = Toolkit.getDefaultToolkit().getScreenSize();
        jf.setLocation(dim.width / 2 - jf.getSize().width / 2, dim.height / 2 - jf.getSize().height / 2);
        jf.addWindowListener(new WindowAdapter() {
            @Override
            public void windowClosing(final WindowEvent e) {
                jf.dispose();
                System.exit(0);
            }
        });
        jf.setVisible(true);
    }

    private static double[] prepareArrayForPlot(final double[] source) {
        final int n = source.length;
        final double max = Doubles.max(source);
        final double[] result = new double[n];
        for (int i = 0; i < n; i++) {
            result[i] = source[n - i - 1] / max;
        }
        return result;
    }

    private static void displayProgressFrame() {
```

```
        fillTable();
        fitTableToWidth();
        addTableToPanel();
    }

    private static void addTableToPanel() {
        final JFrame jf = new JFrame("Progress Frame");
        jf.getContentPane().setLayout(new BorderLayout());
        TABLE.getTableHeader().setResizingAllowed(false);
        TABLE.setBorder(BorderFactory.createEmptyBorder());
        TABLE.addMouseListener(new MouseAdapter() {
            @Override
            public void mousePressed(final MouseEvent e) {
                final JTable jTable = (JTable) e.getSource();
                final Point p = e.getPoint();
                final int row = jTable.rowAtPoint(p);
                //noinspection SuspiciousMethodCalls
                if (e.getClickCount() >= 2 && COMPUTED.containsKey(row)) {
                    final XYItemRenderer renderer = ((XYPlot) CHART.getPlot()).getRenderer();
                    final int plotNumber = COMPUTED.get(row);
                    //noinspection NegativelyNamedBooleanVariable
                    final boolean hidden = !VISIBLE.get(plotNumber);
                    renderer.setSeriesVisible(plotNumber, !VISIBLE.put(plotNumber, hidden));
                }
            }
        });
        final JScrollPane scrollPane = new JScrollPane(TABLE);
        scrollPane.setBorder(BorderFactory.createEmptyBorder());
        final Dimension size = TABLE.getPreferredSize();
        scrollPane.setPreferredSize(new Dimension(size.width - 10, size.height + 10));
        jf.getContentPane().add(scrollPane, BorderLayout.CENTER);
        jf.setDefaultCloseOperation(WindowConstants.EXIT ON CLOSE);
        jf.addWindowListener(new WindowAdapter() {
            @Override
            public void windowClosing(final WindowEvent e) {
                jf.dispose();
                System.exit(0);
            }
        });
        final Dimension dim = Toolkit.getDefaultToolkit().getScreenSize();
        jf.setLocation(dim.width / 2 - jf.getSize().width / 2, dim.height / 2 - jf.getSize().height / 2);
        jf.pack();
        jf.setResizable(false);
        jf.setVisible(true);
    }

    private static void fitTableToWidth() {
        TABLE.setAutoResizeMode(JTable.AUTO_RESIZE_OFF);
        for (int i = 0; i < TABLE.getColumnCount(); i++) {
            if (i != 0) {
                centerColumn(i);
            }
            packColumn(i, 10);
        }
    }

    private static void centerColumn(final int column) {
        final DefaultTableCellRenderer centerRenderer = new DefaultTableCellRenderer();
        centerRenderer.setHorizontalAlignment(SwingConstants.CENTER);
        TABLE.getColumnModel().getColumn(column).setCellRenderer(centerRenderer);
    }

    private static void packColumn(final int column, final int margin) {
        final TableColumnModel colModel = (DefaultTableColumnModel) TABLE.getColumnModel();
        final TableColumn col = colModel.getColumn(column);
        TableCellRenderer renderer = col.getHeaderRenderer();
        if (renderer == null) {
            renderer = TABLE.getTableHeader().getDefaultRenderer();
        }
        Component comp = renderer.getTableCellRendererComponent(TABLE, col.getHeaderValue(), false, false, 0, 0);
        int width = comp.getPreferredSize().width;
        for (int row = 0; row < TABLE.getRowCount(); row++) {
            renderer = TABLE.getCellRenderer(row, column);
            comp = renderer.getTableCellRendererComponent(
                    TABLE, TABLE.getValueAt(row, column), false, false, row, column);
            width = Math.max(width, comp.getPreferredSize().width);
        }
        width += 2 * margin;
        col.setPreferredWidth(width);
    }

    private static void fillTable() {
        for (int i = 0; i < CLASSIFIERS.size(); i++) {
            TABLE.setValueAt(CLASSIFIERS.get(i).getClass().getSimpleName(), i, 0);
            TABLE.setValueAt(IN_QUEUE, i, 1);
            TABLE.setValueAt(IN_QUEUE, i, 2);
            TABLE.setValueAt(IN_QUEUE, i, 3);
            TABLE.setValueAt(IN_QUEUE, i, 4);
            TABLE.setValueAt(IN_QUEUE, i, 5);
            VISIBLE.put(i, true);
        }
    }

    private static Instances readData() {
        try (final BufferedReader reader = new BufferedReader(new FileReader(INPUT_FILE_NAME))) {
            final Instances data = new Instances(reader);
            data.setClass(data.attribute(data.numAttributes() - 1));
            return data;
        } catch (final FileNotFoundException ignored) {
```

```
            System.out.println("Can't find file " + INPUT_FILE_NAME);
            return null;
        } catch (final IOException ignored) {
            System.out.println("Error parsing file " + INPUT_FILE_NAME);
            return null;
        }
    }

    private static Stroke getStroke(final int plotNumber) {
        //noinspection NumericCastThatLosesPrecision,MagicNumber
        return new BasicStroke(2.5f, BasicStroke.CAP_ROUND, BasicStroke.JOIN_ROUND,
                5.0f, new float[]{(float) StrictMath.pow(plotNumber, 0.75) * 2}, 0.0f);
    }

    private static class ClassifierResults {
        private final double[] truePositives;
        private final double truePositivesCount;
        private final double falseNegativesCount;
        private final double[] falsePositives;
        private final double falsePositivesCount;
        private final double trueNegativesCount;
        private final double areaUnderCurve;
        private final double truePositivesRate;
        private final double trueNegativesRate;

        public ClassifierResults(final double[] truePositives, final double truePositivesCount,
                                 final double[] falsePositives, final double falsePositivesCount,
                                 final double falseNegativesCount, final double trueNegativesCount,
                                 final double trueNegativesRate, final double truePositivesRate,
                                 final double areaUnderCurve) {
            this.truePositives = truePositives;
            this.truePositivesCount = truePositivesCount;
            this.falseNegativesCount = falseNegativesCount;
            this.falsePositives = falsePositives;
            this.falsePositivesCount = falsePositivesCount;
            this.trueNegativesCount = trueNegativesCount;
            this.areaUnderCurve = areaUnderCurve;
            this.trueNegativesRate = trueNegativesRate;
            this.truePositivesRate = truePositivesRate;
        }

        public double getMatthewsCorrelationCoefficient() {
            return (truePositivesCount * trueNegativesCount - falsePositivesCount * falseNegativesCount) /
                    Math.sqrt((truePositivesCount + falsePositivesCount) *
                            (truePositivesCount + falseNegativesCount) *
                            (trueNegativesCount + falsePositivesCount) *
                            (trueNegativesCount + falseNegativesCount));
        }

        public double[] getTruePositives() {
            return truePositives;
        }

        public double[] getFalsePositives() {
            return falsePositives;
        }

        public double getAreaUnderCurve() {
            return areaUnderCurve;
        }

        public double getTrueNegativesRate() {
            return trueNegativesRate;
        }

        public double getTruePositivesRate() {
            return truePositivesRate;
        }
    }

    private static class ClassifierThread implements Runnable {
        private final Classifier classifier;
        private final Instances data;
        private final Evaluation eval;

        public ClassifierThread(final Classifier classifier, final Instances data) throws Exception {
            this.classifier = classifier;
            this.data = data;
            this.eval = new Evaluation(data);
        }

        @Override
        public void run() {
            final Random random = new Random(1);
            try {
                System.out.println("START " + classifier.getClass().getSimpleName());
                final long before = System.currentTimeMillis();
                classifier.buildClassifier(data);
                eval.crossValidateModel(classifier, data, 10, random);
                final long after = System.currentTimeMillis();
                final ThresholdCurve thresholdCurve = new ThresholdCurve();
                final Instances curve = thresholdCurve.getCurve(eval.predictions(), 0);
                final double[] tp = curve.attributeToDoubleArray(curve.attribute("True Positives").index());
                final double tpc = eval.numTruePositives(0);
                final double fnc = eval.numFalseNegatives(0);
                final double[] fp = curve.attributeToDoubleArray(curve.attribute("False Positives").index());
                final double fpc = eval.numFalsePositives(0);
                final double tnc = eval.numTrueNegatives(0);
                final double auc = eval.areaUnderROC(0);
```

```
                final double tnr = eval.trueNegativeRate(0);
                final double tpr = eval.truePositiveRate(0);
                addPlot(new ClassifierResults(tp, tpc, fp, fpc, fnc, tnc, tnr, tpr, auc),
                        classifier, after - before);
            } catch (Exception e) {
                e.printStackTrace(System.out);
            } finally {
                System.out.println("FINISH " + classifier.getClass().getSimpleName());
            }
        }
    }
}
```