1. Linear.java

```
/*
 * Copyright (c) 2007-2021 The LIBLINEAR Project.
 * All rights reserved.
 */
import java.io.BufferedReader;
import java.io.BufferedWriter;
import java.io.EOFException;
import java.io.File;
import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.io.IOException;
import java.io.InputStreamReader;
import java.io.OutputStreamWriter;
import java.io.PrintStream;
import java.io.Reader;
import java.io.Writer;
import java.nio.charset.Charset;
import java.nio.charset.StandardCharsets;
import java.util.ArrayList;
import java.util.Arrays;
import java.util.Formatter;
import java.util.HashMap;
import java.util.LinkedHashMap;
import java.util.List;
import java.util.Locale;
import java.util.Map;
import java.util.Random;
import java.util.TreeMap;
import java.util.concurrent.Callable;
import java.util.concurrent.ExecutorService;
import java.util.concurrent.Executors;
import java.util.concurrent.Future;
import java.util.concurrent.TimeUnit;
import java.util.regex.Pattern;
class TrainOneMulThread implements Callable<Object>{
        private Problem sub_prob;
        private Parameter param;
        private int class_idx;
```

```
        private int w_size;
        private int nr_class;
                private volatile double[] w;
        public TrainOneMulThread(Problem sub_prob, Parameter param, int
class_idx, int w_size, int nr_class) {
                this.sub_prob = sub_prob;
        this.param =param;
                this.class_idx = class_idx;
                this.w_size = w_size;
                this.nr_class = nr_class;

        }
                public double[] getWeight() {
                return this.w;
        }
        public int getClassIdx() {
                return this.class_idx;
        }
        @Override
        public Object call() throws Exception {
                this.w = new double[w_size];

        for (int j = 0; j < w_size; j++)
                this.w[j] = 0;

        Linear.solve_l2r_l1l2_svc(this.sub_prob, this.w, param.eps, param.C,
param.C, SolverType.L2R_L2LOSS_SVC_DUAL, param.max_iters);
                return this;
        }
}public class Linear {
    static final int VERSION = 211;
    static final Charset FILE_CHARSET = StandardCharsets.ISO_8859_1;
    static final Locale DEFAULT_LOCALE = Locale.ENGLISH;
    private static Object OUTPUT_MUTEX = new Object();
    private static PrintStream DEBUG_OUTPUT = System.out;
    private static final long DEFAULT_RANDOM_SEED = 0L;
    static Random random = new
Random(DEFAULT_RANDOM_SEED);

    /** used as complex return type */
    private static class GroupClassesReturn {
```

```
    final int[] count;
    final int[] label;
    final int    nr_class;
    final int[] start;
    GroupClassesReturn(int nr_class, int[] label, int[] start, int[] count) {
        this.nr_class = nr_class;
        this.label = label;
        this.start = start;
        this.count = count;
    }
}
private static GroupClassesReturn groupClasses(Problem prob, int[] perm) {
    int l = prob.l;
    int max_nr_class = 16;
    int nr_class = 0;
    int[] label = new int[max_nr_class];
    int[] count = new int[max_nr_class];
    int[] data_label = new int[l];
    int i;
    for (i = 0; i < l; i++) {
        int this_label = (int) prob.y[i];
        int j;
        for (j = 0; j < nr_class; j++) {
            if (this_label == label[j]) {
                ++count[j];
                break;
            }
        }
        data_label[i] = j;

        if (j == nr_class) {
            if (nr_class == max_nr_class) {
                max_nr_class *= 2;
                label = copyOf(label, max_nr_class);
                count = copyOf(count, max_nr_class);
            }
            label[nr_class] = this_label;
            count[nr_class] = 1;
            ++nr_class;
        }
    }
```

```
        if (nr_class == 2 && label[0] == -1 && label[1] == 1) {
            swap(label, 0, 1);
            swap(count, 0, 1);
            for (i = 0; i < l; i++) {
                if (data_label[i] == 0)
                    data_label[i] = 1;
                else
                    data_label[i] = 0;
            }
        }
        int[] start = new int[nr_class];
        start[0] = 0;
        for (i = 1; i < nr_class; i++)
            start[i] = start[i - 1] + count[i - 1];
        for (i = 0; i < l; i++) {
            perm[start[data_label[i]]] = i;
            ++start[data_label[i]];
        }
        start[0] = 0;
        for (i = 1; i < nr_class; i++)
            start[i] = start[i - 1] + count[i - 1];
        return new GroupClassesReturn(nr_class, label, start, count);
    }
    static void info(String message) {
        synchronized (OUTPUT_MUTEX) {
            if (DEBUG_OUTPUT == null) return;
            DEBUG_OUTPUT.printf(message);
            DEBUG_OUTPUT.flush();
        }
    }
    static void info(String format, Object... args) {
        synchronized (OUTPUT_MUTEX) {
            if (DEBUG_OUTPUT == null) return;
            DEBUG_OUTPUT.printf(format, args);
            DEBUG_OUTPUT.flush();
        }
    }
    /**
     * @param s the string to parse for the double value
     * @throws IllegalArgumentException if s is empty or represents NaN or
Infinity
```

```
     * @throws NumberFormatException see {@link double#parsedouble(String)}
     */
    static double atof(String s) {
        if (s == null || s.length() < 1) throw new IllegalArgumentException("Can't
convert empty string to integer");
        double d = Double.parseDouble(s);
        if (Double.isNaN(d) || Double.isInfinite(d)) {
            throw new IllegalArgumentException("NaN or Infinity in input: " + s);
        }
        return (d);
    }
    /**
     * @param s the string to parse for the integer value
     * @throws IllegalArgumentException if s is empty
     * @throws NumberFormatException see {@link Integer#parseInt(String)}
     */
    static int atoi(String s) throws NumberFormatException {
        if (s == null || s.length() < 1) throw new IllegalArgumentException("Can't
convert empty string to integer");

        if (s.charAt(0) == '+') s = s.substring(1);
        return Integer.parseInt(s);
    }
    /**
     * Java5 'backport' of Arrays.copyOf
     */
    public static double[] copyOf(double[] original, int newLength) {
        double[] copy = new double[newLength];
        System.arraycopy(original, 0, copy, 0, Math.min(original.length, newLength));
        return copy;
    }
    /**
     * Java5 'backport' of Arrays.copyOf
     */
    public static int[] copyOf(int[] original, int newLength) {
        int[] copy = new int[newLength];
        System.arraycopy(original, 0, copy, 0, Math.min(original.length, newLength));
        return copy;
    }
    /**
     * Loads the model from inputReader.
```

```
     * It uses {@link java.util.Locale#ENGLISH} for number formatting.
     *
     * <p>Note: The inputReader is <b>NOT closed</b> after reading or in case of an exception.</p>
     */
    public static Model loadModel(Reader inputReader) throws IOException {
        Model model = new Model();
        model.label = null;
        Pattern whitespace = Pattern.compile("\\s+");
        BufferedReader reader = null;
        if (inputReader instanceof BufferedReader) {
            reader = (BufferedReader) inputReader;
        } else {
            reader = new BufferedReader(inputReader);
        }
        String line = null;
        while ((line = reader.readLine()) != null) {
            String[] split = whitespace.split(line);
            if (split[0].equals("solver_type")) {
                SolverType solver = SolverType.valueOf(split[1]);
                if (solver == null) {
                    throw new RuntimeException("unknown solver type");
                }
                model.solverType = solver;
            } else if (split[0].equals("nr_class")) {
                model.nr_class = atoi(split[1]);
                Integer.parseInt(split[1]);
            } else if (split[0].equals("nr_feature")) {
                model.nr_feature = atoi(split[1]);
            } else if (split[0].equals("bias")) {
                model.bias = atof(split[1]);
            } else if (split[0].equals("w")) {
                break;
            } else if (split[0].equals("label")) {
                model.label = new int[model.nr_class];
                for (int i = 0; i < model.nr_class; i++) {
                    model.label[i] = atoi(split[i + 1]);
                }
            } else {
                throw new RuntimeException("unknown text in model file: [" + line + "]");
```

```
            }
        }
        int w_size = model.nr_feature;
        if (model.bias >= 0) w_size++;
        int nr_w = model.nr_class;
        if (model.nr_class == 2 && model.solverType != SolverType.MCSVM_CS)
nr_w = 1;
        model.w = new double[w_size * nr_w];
        int[] buffer = new int[128];
        for (int i = 0; i < w_size; i++) {
            for (int j = 0; j < nr_w; j++) {
                int b = 0;
                while (true) {
                    int ch = reader.read();
                    if (ch == -1) {
                        throw new EOFException("unexpected EOF");
                    }
                    if (ch == ' ') {
                        model.w[i * nr_w + j] = atof(new String(buffer, 0, b));
                        break;
                    } else {
                        if (b >= buffer.length) {
                            throw new RuntimeException("illegal weight in model file
at index " + (i * nr_w + j) + ", with string content '" +
                                    new String(buffer, 0, buffer.length) + "', is not
terminated " +
                                    "with a whitespace character, or is longer than
expected (" + buffer.length + " characters max).");
                        }
                        buffer[b++] = ch;
                    }
                }
            }
        }
        return model;
    }
    /**
     * Loads the model from the file with ISO-8859-1 charset.
     * It uses {@link java.util.Locale#ENGLISH} for number formatting.
     */
    public static Model loadModel(File modelFile) throws IOException {
```

```
        try (FileInputStream in = new FileInputStream(modelFile);
                InputStreamReader inputStreamReader = new InputStreamReader(in, FILE_CHARSET);
                BufferedReader inputReader = new BufferedReader(inputStreamReader)) {
            return loadModel(inputReader);
        }
    }
    public static double predict(Model model, Feature[] x) {
        double[] dec_values = new double[model.nr_class];
        return predictValues(model, x, dec_values);
    }

    public static TreeMap<Integer, Double> predictAllClass(Model model, Feature[] x) {
        double[] dec_values = new double[model.nr_class];
        return predictAllClassValues(model, x, dec_values);
    }
     /**
     * @throws IllegalArgumentException if model is not probabilistic (see {@link Model#isProbabilityModel()})
     */
    public static double predictProbability(Model model, Feature[] x, double[] prob_estimates) throws IllegalArgumentException {
        if (!model.isProbabilityModel()) {
            StringBuilder sb = new StringBuilder("probability output is only supported for logistic regression");
            sb.append(". This is currently only supported by the following solvers: ");
            int i = 0;
            for (SolverType solverType : SolverType.values()) {
                if (solverType.isLogisticRegressionSolver()) {
                    if (i++ > 0) {
                        sb.append(", ");
                    }
                    sb.append(solverType.name());
                }
            }
            throw new IllegalArgumentException(sb.toString());
        }
        int nr_class = model.nr_class;
```

```
        int nr_w;
        if (nr_class == 2)
            nr_w = 1;
        else
            nr_w = nr_class;
        double label = predictValues(model, x, prob_estimates);
        for (int i = 0; i < nr_w; i++)
            prob_estimates[i] = 1 / (1 + Math.exp(-prob_estimates[i]));
        if (nr_class == 2)
            prob_estimates[1] = 1. - prob_estimates[0];
        else {
            double sum = 0;
            for (int i = 0; i < nr_class; i++)
                sum += prob_estimates[i];
            for (int i = 0; i < nr_class; i++)
                prob_estimates[i] = prob_estimates[i] / sum;
        }
        return label;
    }
    public static double predictValues(Model model, Feature[] x, double[]
dec_values) {
        int n;
        if (model.bias >= 0)
            n = model.nr_feature + 1;
        else
            n = model.nr_feature;
        double[] w = model.w;
        int nr_w;
        if (model.nr_class == 2 && model.solverType != SolverType.MCSVM_CS)
            nr_w = 1;
        else
            nr_w = model.nr_class;
        for (int i = 0; i < nr_w; i++)
            dec_values[i] = 0;


        for (Feature lx : x) {
            int idx = lx.getIndex();
            if (idx <= n) {
                for (int i = 0; i < nr_w; i++) {
                    dec_values[i] += w[(idx - 1) * nr_w + i] * lx.getValue();
```

```
                }
            }
        }

        if (model.nr_class == 2) {
            if (model.solverType.isSupportVectorRegression())
                return dec_values[0];
            else
                return (dec_values[0] > 0) ? model.label[0] : model.label[1];
        } else {
            int dec_max_idx = 0;
            for (int i = 1; i < model.nr_class; i++) {
                if (dec_values[i] > dec_values[dec_max_idx]) dec_max_idx = i;
            }
            return model.label[dec_max_idx];
        }
    }

    public static TreeMap<Integer, Double> predictAllClassValues(Model model,
Feature[] x, double[] dec_values) {
        TreeMap<Integer, Double> result = new TreeMap<Integer, Double>();
        int n;
        if (model.bias >= 0)
            n = model.nr_feature + 1;
        else
            n = model.nr_feature;
        double[] w = model.w;
        int nr_w;
        if (model.nr_class == 2 && model.solverType != SolverType.MCSVM_CS)
            nr_w = 1;
        else
            nr_w = model.nr_class;
        for (int i = 0; i < nr_w; i++)
            dec_values[i] = 0;


        for (Feature lx : x) {
            int idx = lx.getIndex();
            if (idx <= n) {
                for (int i = 0; i < nr_w; i++) {
                    dec_values[i] += w[(idx - 1) * nr_w + i] * lx.getValue();
```

```
                }
            }
        }

        for(int i=0; i<model.nr_class;i++)
                result.put(model.label[i], dec_values[i]);

        return result;
    }
```

```
    static void printf(Formatter formatter, String format, Object... args) throws IOException {
        formatter.format(format, args);
        IOException ioException = formatter.ioException();
        if (ioException != null) throw ioException;
    }
    /**
     * Writes the model to the modelOutput.
     * It uses {@link java.util.Locale#ENGLISH} for number formatting.
     *
     * <p><b>Note: The modelOutput is closed after reading or in case of an exception.</b></p>
     */

    public static void saveModel(Writer modelOutput, Model model) throws IOException {
        int nr_feature = model.nr_feature;
        int w_size = nr_feature;
        if (model.bias >= 0) w_size++;
        int nr_w = model.nr_class;
        if (model.nr_class == 2 && model.solverType != SolverType.MCSVM_CS) nr_w = 1;
        try (Formatter formatter = new Formatter(modelOutput, DEFAULT_LOCALE)) {
            printf(formatter, "solver_type %s\n", model.solverType.name());
            printf(formatter, "nr_class %d\n", model.nr_class);
            if (model.label != null) {
                printf(formatter, "label");
```

```
                for (int i = 0; i < model.nr_class; i++) {
                    printf(formatter, " %d", model.label[i]);
                }
                printf(formatter, "\n");
            }
            printf(formatter, "nr_feature %d\n", nr_feature);
            printf(formatter, "bias %.16g\n", model.bias);
            printf(formatter, "w\n");
            for (int i = 0; i < w_size; i++) {
                for (int j = 0; j < nr_w; j++) {
                    double value = model.w[i * nr_w + j];
                    /** this optimization is the reason for {@link
Model#equals(double[], double[])} */
                    if (value == 0.0) {
                        printf(formatter, "%d ", 0);
                    } else {
                        printf(formatter, "%.16g ", value);
                    }
                }
                printf(formatter, "\n");
            }
            formatter.flush();
            IOException ioException = formatter.ioException();
            if (ioException != null) throw ioException;
        }
    }
    /**
     * Writes the model to the file with ISO-8859-1 charset.
     * It uses {@link java.util.Locale#ENGLISH} for number formatting.
     */
    public static void saveModel(File modelFile, Model model) throws IOException {
        BufferedWriter modelOutput = new BufferedWriter(new
OutputStreamWriter(new FileOutputStream(modelFile), FILE_CHARSET));
        saveModel(modelOutput, model);
    }




    public static void saveModelBinary(String modelPath, Model model) throws
IOException {
```

```
        int nr_feature = model.nr_feature;
        int w_size = nr_feature;
        if (model.bias >= 0) w_size++;
        int nr_w = model.nr_class;
        if (model.nr_class == 2 && model.solverType != SolverType.MCSVM_CS)
nr_w = 1;
        VirtualFileOutputStream os = null;
                try {
                        System.out.println("start save model to "+ modelPath);
                        os = new VirtualFileOutputStream(modelPath);
                        ByteUtil.writeString(os,       String.format("solver_type       %s",
model.solverType.name()));
                        ByteUtil.writeString(os,        String.format("nr_class        %d",
model.nr_class));
                        ByteUtil.writeString(os,       String.format("nr_feature       %d",
nr_feature));
                        ByteUtil.writeString(os,        String.format("bias          %.16g",
model.bias));



                        String label = "label";
            for (int i = 0; i < model.nr_class; i++) {
                label+= String.format(" %d", model.label[i]);
            }
            ByteUtil.writeString(os,label);



                                ByteUtil.writeString(os,"w");

            for (int i = 0; i < w_size; i++) {
                for (int j = 0; j < nr_w; j++) {
                    double value = model.w[i * nr_w + j];
                    if (value == 0.0) {
                        ByteUtil.writeString(os,String.format("%d ", 0));
                    } else {
                        ByteUtil.writeString(os,String.format("%.16g ", value));
                    }
                }
            }
            os.flush();
                        System.out.println("success save model to "+ modelPath);
```

```
            } catch (IOException e) {
                    System.out.println("fail save model to "+ modelPath);
                    e.printStackTrace();
            }finally {
                    os.close();
            }
    }



    public static Model loadModelBinary(String modelPath) throws IOException {
        Model model = new Model();
        VirtualFileInputStream is = null;

        try {
                is = new VirtualFileInputStream(modelPath);
            model.solverType = SolverType.valueOf(ByteUtil.readString(is).split(" ")[1]);
            model.nr_class = atoi(ByteUtil.readString(is).split(" ")[1]);
            model.nr_feature = atoi(ByteUtil.readString(is).split(" ")[1]);
            model.bias = atof(ByteUtil.readString(is).split(" ")[1]);
            model.label = new int[model.nr_class];
            String[] labelStr = ByteUtil.readString(is).split(" ");
            for (int i = 0; i < model.nr_class; i++) {
                model.label[i] = atoi(labelStr[i + 1]);
            }

            int w_size = model.nr_feature;
            if (model.bias >= 0) w_size++;
            int nr_w = model.nr_class;
            if (model.nr_class == 2 && model.solverType != SolverType.MCSVM_CS)
nr_w = 1;

            model.w = new double[w_size * nr_w];
            int weightCheckCount = 0;
            if(!"w".equals(ByteUtil.readString(is))) {
                System.out.println("weight were not saved propery");
                return null;
            }
```

```
            for (int i = 0; i < w_size; i++) {

                for (int j = 0; j < nr_w; j++) {
                  model.w[i * nr_w + j] = atof(ByteUtil.readString(is));
                  weightCheckCount++;
                }

            }
            System.out.println("setting weight count : " + nr_w*w_size);
            System.out.println("actual weight count : " + weightCheckCount);

        }catch(Exception e) {
                System.out.println("Loading fail");
                e.printStackTrace();
                return null;
        }finally {
                is.close();
        }
        System.out.println("Model Loading successs!");
        return model;
    }


    public static ModelInfo getModelInfo(Model model, HashMap<CharVector, Integer> vocabulary , CategoryInfo categoryInfo) throws IOException{
        int nr_class = model.getNrClass();
        int nr_feature = model.getNrFeature();

        System.out.println("nr_class : "+nr_class);
        System.out.println("nr_feature : "+nr_feature);

        Map<Integer, CharVector> reverseVoca = new HashMap<>();
        for(Map.Entry<CharVector, Integer> entry : vocabulary.entrySet()){
                reverseVoca.put(entry.getValue(), entry.getKey());
        }


        ModelInfo modelinfo = new ModelInfo(model);

        for(int i=0 ; i < nr_class; i++) {
```

```
                ArrayList<ModelFeatureNode> classInfo = new
ArrayList<ModelFeatureNode>();
                String className =  categoryInfo.getCategoryID(model.label[i]);
                for(int j=0 ; j < nr_feature; j++) {
                        double rawFeature = model.w[j * nr_class + i];
                        double roundedFeature =
(double)Math.round(rawFeature*1000000000D)/1000000000D;
                        if (Math.abs(roundedFeature) > 0.01D) {
                                classInfo.add(new ModelFeatureNode(j,
roundedFeature));
                        }
                }


                ModelFeatureNode [] classInfoArr = classInfo.toArray(new
ModelFeatureNode[classInfo.size()]);
                classInfo=null;

                Arrays.sort(classInfoArr);

                LinkedHashMap<String, Double> classInfoMap = new
LinkedHashMap<String, Double>();
                for(int n=0; n<classInfoArr.length; n++) {

classInfoMap.put(reverseVoca.get(classInfoArr[n].getIndex()).toString(),
classInfoArr[n].getValue());
                }


                modelinfo.addClassInfo(className, classInfoMap);
                classInfoArr=null;
        }
        return modelinfo;
    }

    /*
     * this method corresponds to the following define in the C version:
     * #define GETI(i) (y[i]+1)
     */
    private static int GETI(byte[] y, int i) {
        return y[i] + 1;
```

```
}
/**
 * A coordinate descent algorithm for
 * L1-loss and L2-loss SVM dual problems
 *<pre>
 *  min_\alpha  0.5(\alpha^T (Q + D)\alpha) - e^T \alpha,
 *    s.t.      0 <= \alpha_i <= upper_bound_i,
 *
 *  where Qij = yi yj xi^T xj and
 *  D is a diagonal matrix
 *
 * In L1-SVM case:
 *     upper_bound_i = Cp if y_i = 1
 *     upper_bound_i = Cn if y_i = -1
 *     D_ii = 0
 * In L2-SVM case:
 *     upper_bound_i = INF
 *     D_ii = 1/(2*Cp) if y_i = 1
 *     D_ii = 1/(2*Cn) if y_i = -1
 *
 * Given:
 * x, y, Cp, Cn
 * eps is the stopping tolerance
 *
 * solution will be put in w
 *
 * See Algorithm 3 of Hsieh et al., ICML 2008
 *</pre>
 */
public static void solve_l2r_l1l2_svc(Problem prob, double[] w, double eps, double Cp, double Cn, SolverType solver_type, int max_iter) {


        int l = prob.l;
        int w_size = prob.n;
        int i, s, iter = 0;
        double Upper, d, G;
        double[] QD = new double[l];
        int[] index = new int[l];
        double[] alpha = new double[l];
        byte[] y = new byte[l];
```

```
            int active_size = l;

            double PG;
            double PGmax_old = Double.POSITIVE_INFINITY;
            double PGmin_old = Double.NEGATIVE_INFINITY;
            double PGmax_new, PGmin_new;

            double diag[] = new double[] {0.5 / Cn, 0, 0.5 / Cp};
              double upper_bound[] = new double[] {Double.POSITIVE_INFINITY, 0,
Double.POSITIVE_INFINITY};
            if (solver_type == SolverType.L2R_L1LOSS_SVC_DUAL) {
                diag[0] = 0;
                diag[2] = 0;
                upper_bound[0] = Cn;
                upper_bound[2] = Cp;
            }

            for (i = 0; i < l; i++) {
                if (prob.y[i] > 0) {
                    y[i] = +1;
                } else {
                    y[i] = -1;
                }
            }


            for (i = 0; i < l; i++)
                alpha[i] = 0;
            for (i = 0; i < w_size; i++)
                w[i] = 0;

            for (i = 0; i < l; i++) {


                /*
                D_{ii}=diag[GETI(y, i)]값
                   - 만약 y[i]가 positive class인 (+1)라면 0.5/Cp, negative class(-1)이
라면 0.5/Cn
                   -  그런데 어짜피 Cp=Cn=C 이므로 positive class, negative class 모
두 0.5/C가 됨
                 */
```

```
    QD[i] = diag[GETI(y, i)];
    Feature[] xi = prob.x[i];
    QD[i] += SparseOperator.nrm2_sq(xi);

    SparseOperator.axpy(y[i] * alpha[i], xi, w);
    index[i] = i;
}


while (iter < max_iter) {
    PGmax_new = Double.NEGATIVE_INFINITY;
    PGmin_new = Double.POSITIVE_INFINITY;

    for (i = 0; i < active_size; i++) {
        int j = i + random.nextInt(active_size - i);
        swap(index, i, j);
    }









    for (s = 0; s < active_size; s++) {

        i = index[s];
        byte yi = y[i];

        Feature[] xi = prob.x[i];
        G = yi * SparseOperator.dot(w, xi) - 1;
        Upper = upper_bound[GETI(y, i)];

        G += alpha[i] * diag[GETI(y, i)];
        PG = 0;
        if (alpha[i] == 0) {


            if (G > PGmax_old) {
```

```
                    active_size--;
                    swap(index, s, active_size);
                    s--;
                    continue;
                } else if (G < 0) {
                    PG = G;
                }
            } else if (alpha[i] == Upper) {

                if (G < PGmin_old) {
                    active_size--;
                    swap(index, s, active_size);
                    s--;
                    continue;
                } else if (G > 0) {
                    PG = G;
                }
            } else {
                PG = G;
            }


            PGmax_new = Math.max(PGmax_new, PG);
            PGmin_new = Math.min(PGmin_new, PG);

            if (Math.abs(PG) > 1.0e-12) {
                double alpha_old = alpha[i];

                    alpha[i] = Math.min(Math.max(alpha[i] - G / QD[i], 0.0),
Upper);
                d = (alpha[i] - alpha_old) * yi;
                SparseOperator.axpy(d, xi, w);
            }
        }

        iter++;
        if (PGmax_new - PGmin_new <= eps) {


            if (active_size == l)
                break;
```

```
                else {

                    active_size = l;
                    PGmax_old = Double.POSITIVE_INFINITY;
                    PGmin_old = Double.NEGATIVE_INFINITY;
                    continue;
                }
            }

            PGmax_old = PGmax_new;
            PGmin_old = PGmin_new;

            if (PGmax_old <= 0)
                PGmax_old = Double.POSITIVE_INFINITY;

            if (PGmin_old >= 0)
                PGmin_old = Double.NEGATIVE_INFINITY;
        }
    }


static Problem transpose(Problem prob) {
    int l = prob.l;
    int n = prob.n;
    int[] col_ptr = new int[n + 1];
    Problem prob_col = new Problem();
    prob_col.l = l;
    prob_col.n = n;
    prob_col.y = new double[l];
    prob_col.x = new Feature[n][];
    for (int i = 0; i < l; i++)
        prob_col.y[i] = prob.y[i];
    for (int i = 0; i < l; i++) {
        for (Feature x : prob.x[i]) {
            col_ptr[x.getIndex()]++;
        }
    }
```

```
        for (int i = 0; i < n; i++) {
            prob_col.x[i] = new Feature[col_ptr[i + 1]];
            col_ptr[i] = 0;
        }
        for (int i = 0; i < l; i++) {
            for (int j = 0; j < prob.x[i].length; j++) {
                Feature x = prob.x[i][j];
                int index = x.getIndex() - 1;
                prob_col.x[index][col_ptr[index]] = new FeatureNode(i + 1,
x.getValue());
                col_ptr[index]++;
            }
        }
        return prob_col;
    }
    static void swap(double[] array, int idxA, int idxB) {
        double temp = array[idxA];
        array[idxA] = array[idxB];
        array[idxB] = temp;
    }
    static void swap(int[] array, int idxA, int idxB) {
        int temp = array[idxA];
        array[idxA] = array[idxB];
        array[idxB] = temp;
    }
    static void swap(IntArrayPointer array, int idxA, int idxB) {
        int temp = array.get(idxA);
        array.set(idxA, array.get(idxB));
        array.set(idxB, temp);
    }
    /**
     * @throws IllegalArgumentException if the feature nodes of prob are not
sorted in ascending order
     */


    public static int properThreadNum(int w_size, int nr_class) {
        int threadNum=1;

                long eachThreadMem = (long)20*w_size*nr_class;
```

```
                long FreeMemory = Runtime.getRuntime().freeMemory();


                threadNum+=(int)(FreeMemory/eachThreadMem);
                System.out.println(String.format("[SVM]detected proper thread num is %d", threadNum));
        return threadNum;

    }


    public static Model train(Problem prob, Parameter param, int svmMultiThread) {
        float progress = 0.0f;

        if (prob == null) throw new IllegalArgumentException("problem must not be null");
        if (param == null) throw new IllegalArgumentException("parameter must not be null");
        if (prob.n == 0) throw new IllegalArgumentException("problem has zero features");
        if (prob.l == 0) throw new IllegalArgumentException("problem has zero instances");

        for (int i=0; i< prob.x.length ; i++) {
                Feature[] nodes = prob.x[i];
            int indexBefore = 0;

            for (int j=0; j< nodes.length ; j++) {
                Feature n = nodes[j];
                if (n==null) {
                        prob.x[i][j] = new FeatureNode(indexBefore+1,1.0);
                        n = prob.x[i][j];
                }
                if (n.getIndex() <= indexBefore) {
                        throw new IllegalArgumentException("feature nodes must be sorted by index in ascending order");
                }
                indexBefore = n.getIndex();
            }
        }
        int l = prob.l;
```

```
int n = prob.n;
int w_size = prob.n;
Model model = new Model();
if (prob.bias >= 0)
    model.nr_feature = n - 1;
else
    model.nr_feature = n;
model.solverType = param.solverType;
model.bias = prob.bias;
int[] perm = new int[l];

GroupClassesReturn rv = groupClasses(prob, perm);
int nr_class = rv.nr_class;
int[] label = rv.label;
int[] start = rv.start;
int[] count = rv.count;
checkProblemSize(n, nr_class);
model.nr_class = nr_class;
model.label = new int[nr_class];
for (int i = 0; i < nr_class; i++)
    model.label[i] = label[i];


double[] weighted_C = new double[nr_class];
for (int i = 0; i < nr_class; i++)
    weighted_C[i] = param.C;


Feature[][] x = new Feature[l][];
for (int i = 0; i < l; i++)
    x[i] = prob.x[perm[i]];
if (nr_class == 2) {
    Problem sub_prob = new Problem();
    sub_prob.l = l;
    sub_prob.n = n;
    sub_prob.x = new Feature[sub_prob.l][];
    sub_prob.y = new double[sub_prob.l];
    for (int k = 0; k < sub_prob.l; k++)
        sub_prob.x[k] = x[k];

        model.w = new double[w_size];
```

```
            int e0 = start[0] + count[0];
            int k = 0;
            for (; k < e0; k++)
                sub_prob.y[k] = +1;
            for (; k < sub_prob.l; k++)
                sub_prob.y[k] = -1;
            if (param.init_sol != null)
                for (int i = 0; i < w_size; i++)
                    model.w[i] = param.init_sol[i];
            else
                for (int i = 0; i < w_size; i++)
                    model.w[i] = 0;
            train_one(sub_prob, param, model.w, param.C, param.C);
        } else {



            model.w = new double[w_size * nr_class];
            double[] w = new double[w_size];
                boolean useMulTread = true;

                if (useMulTread) {
                        int mulTreadNum = properThreadNum(w_size, nr_class);

                        mulTreadNum = svmMultiThread>mulTreadNum?
mulTreadNum : svmMultiThread;

                        System.out.println("[SVM]actual muli-thread number :
"+mulTreadNum);

                        int i = 0;
                        while (i < nr_class) {
                                try{
                                ExecutorService service =
Executors.newFixedThreadPool(mulTreadNum);
                                List<Future<Object>> futures = new
ArrayList<Future<Object>>();

                                for(int t=0; t<mulTreadNum; t++ ) {
                                        if(i < nr_class) {
```

```
                Problem sub_prob = new Problem();
                sub_prob.l = l;
                sub_prob.n = n;
                sub_prob.x = new Feature[sub_prob.l][];
                sub_prob.y = new double[sub_prob.l];

                for (int k = 0; k < sub_prob.l; k++)
                    sub_prob.x[k] = x[k];




                int si = start[i];
                    int ei = si + count[i];
                    int k = 0;
                    for (; k < si; k++)
                        sub_prob.y[k] = -1;
                    for (; k < ei; k++)
                        sub_prob.y[k] = +1;
                    for (; k < sub_prob.l; k++)
                        sub_prob.y[k] = -1;

                                                Future f =
service.submit(new TrainOneMulThread(sub_prob, param, i, w_size, nr_class));

                      futures.add(f);
                      i++;
                        }
                }
                for (Future<Object> f : futures){
                        TrainOneMulThread res =
(TrainOneMulThread)f.get();
                        double[] w1 = res.getWeight();
                        int class_idx1 = res.getClassIdx();
            for (int j = 0; j < n; j++)
                synchronized(model.w) {
                    model.w[j * nr_class + class_idx1] = w1[j];

                }
                }
                service.shutdown();
```

```
                                        service.awaitTermination(Long.MAX_VALUE,
TimeUnit.MINUTES);

                            progress = 100*(float)(i+1)/(float)nr_class;
                            if (progress > 100f) {
                                progress = 100f;
                            }
                              System.out.println(String.format("[%d Thread]SVM train
progress : %.2f",mulTreadNum, progress)+"% completed" );

                                } catch(Exception e){
                                        e.printStackTrace();
                                }
                        }
                }else {
                for (int i = 0; i < nr_class; i++) {

                            Problem sub_prob = new Problem();
                            sub_prob.l = l;
                            sub_prob.n = n;
                            sub_prob.x = new Feature[sub_prob.l][];
                            sub_prob.y = new double[sub_prob.l];
                            for (int k = 0; k < sub_prob.l; k++)
                                sub_prob.x[k] = x[k];

                        int si = start[i];
                    int ei = si + count[i];
                    int k = 0;
                    for (; k < si; k++)
                        sub_prob.y[k] = -1;
                    for (; k < ei; k++)
                        sub_prob.y[k] = +1;
                    for (; k < sub_prob.l; k++)
                        sub_prob.y[k] = -1;
                    if (param.init_sol != null)
                        for (int j = 0; j < w_size; j++)
                            w[j] = param.init_sol[j * nr_class + i];
                    else
                        for (int j = 0; j < w_size; j++)
                            w[j] = 0;
```

```
                    train_one(sub_prob, param, w, param.C, param.C);

                    progress = 100*(float)(i+1)/(float)nr_class;
                    System.out.println(String.format("SVM train progress : %.2f",
progress)+"% completed" );
                    for (int j = 0; j < n; j++)
                        model.w[j * nr_class + i] = w[j];
                }
                }
        }
        return model;
    }
    /**
     * verify the size and throw an exception early if the problem is too large
     */
    private static void checkProblemSize(int n, int nr_class) {
        if (n >= Integer.MAX_VALUE / nr_class || n * nr_class < 0) {
            throw new IllegalArgumentException("'number of classes' * 'number of
instances' is too large: " + nr_class + "*" + n);
        }
    }
    private static void train_one(Problem prob, Parameter param, double[] w,
double Cp, double Cn) {
        double eps = param.eps;
        switch (param.solverType) {
            case L2R_L2LOSS_SVC_DUAL:
                solve_l2r_l1l2_svc(prob, w, eps, Cp, Cn,
SolverType.L2R_L2LOSS_SVC_DUAL, param.max_iters);
                break;
            case L2R_L1LOSS_SVC_DUAL:
                solve_l2r_l1l2_svc(prob, w, eps, Cp, Cn,
SolverType.L2R_L1LOSS_SVC_DUAL, param.max_iters);
                break;
            default:
                throw new IllegalStateException("unknown solver type: " +
param.solverType);
        }
    }
    private static double calc_start_C(Problem prob, Parameter param) {
        int i;
        double xTx, max_xTx;
```

```
            max_xTx = 0;
            for (i = 0; i < prob.l; i++) {
                xTx = 0;
                for (Feature xi : prob.x[i]) {
                    double val = xi.getValue();
                    xTx += val * val;
                }
                if (xTx > max_xTx)
                    max_xTx = xTx;
            }
            double min_C = 1.0;
            if (param.getSolverType() == SolverType.L2R_LR)
                min_C = 1.0 / (prob.l * max_xTx);
            else if (param.getSolverType() == SolverType.L2R_L2LOSS_SVC)
                min_C = 1.0 / (2 * prob.l * max_xTx);
            return Math.pow(2, Math.floor(Math.log(min_C) / Math.log(2.0)));
        }
        public static void disableDebugOutput() {
            setDebugOutput(null);
        }
        public static void enableDebugOutput() {
            setDebugOutput(System.out);
        }
        public static void setDebugOutput(PrintStream debugOutput) {
            synchronized (OUTPUT_MUTEX) {
                DEBUG_OUTPUT = debugOutput;
            }
        }
        public static int getVersion() {
            return VERSION;
        }
        public static void resetRandom() {
            random = new Random(DEFAULT_RANDOM_SEED);
        }
    }
```

1. Model.java

```
/*
 * Copyright (c) 2007-2021 The LIBLINEAR Project.
 * All rights reserved.
 */
import java.io.File;
import java.io.IOException;
import java.io.Reader;
import java.io.Serializable;
import java.io.Writer;
import java.util.Arrays;
public final class Model implements Serializable {
    private static final long serialVersionUID = -6456047576741854834L;
    double                    bias;
    /** label of each class */
    int[]                     label;
    int                       nr_class;
    int                       nr_feature;
    SolverType                solverType;
    /** feature weight array */
    double[]                  w;
    /**
     * @return number of classes
     */
    public int getNrClass() {
        return nr_class;
    }
    /**
     * @return number of features
     */
    public int getNrFeature() {
        return nr_feature;
    }
    public int[] getLabels() {
        return copyOf(label, nr_class);
    }
    public SolverType getSolverType() {
        return solverType;
    }
```

```
/**
 * The array w gives feature weights; its size is
 * nr_feature*nr_class but is nr_feature if nr_class = 2. We use one
 * against the rest for multi-class classification, so each feature
 * index corresponds to nr_class weight values. Weights are
 * organized in the following way
 *
 * <pre>
 * +------------------+------------------+------------+
 * | nr_class weights | nr_class weights |  ...
 * | for 1st feature  | for 2nd feature  |
 * +------------------+------------------+------------+
 * </pre>
 *
 * If bias &gt;= 0, x becomes [x; bias]. The number of features is
 * increased by one, so w is a (nr_feature+1)*nr_class array. The
 * value of bias is stored in the variable bias.
 * @see #getBias()
 * @return a <b>copy of</b> the feature weight array as described
 */
public double[] getFeatureWeights() {
    return Linear.copyOf(w, w.length);
}
/**
 * @return true for logistic regression solvers
 */
public boolean isProbabilityModel() {
    return solverType.isLogisticRegressionSolver();
}
/**
 * @see #getFeatureWeights()
 */
public double getBias() {
    return bias;
}
private double get_w_value(int idx, int label_idx) {
    if (idx < 0 || idx > nr_feature) {
        return 0;
    }
    if (solverType.isSupportVectorRegression()) {
        return w[idx];
```

```
        } else {
            if (label_idx < 0 || label_idx >= nr_class) {
                return 0;
            }
            if (nr_class == 2 && solverType != SolverType.MCSVM_CS) {
                if (label_idx == 0) {
                    return w[idx];
                } else {
                    return -w[idx];
                }
            } else {
                return w[idx * nr_class + label_idx];
            }
        }
    }
    /**
     * This function gives the coefficient for the feature with feature index =
     * feat_idx and the class with label index = label_idx. Note that feat_idx
     * starts from 1, while label_idx starts from 0. If feat_idx is not in the
     * valid range (1 to nr_feature), then a zero value will be returned. For
     * classification models, if label_idx is not in the valid range (0 to
     * nr_class-1), then a zero value will be returned; for regression models,
     * label_idx is ignored.
     *
     * @since 1.95
     */
    public double getDecfunCoef(int featIdx, int labelIdx) {
        if (featIdx > nr_feature) {
            return 0;
        }
        return get_w_value(featIdx - 1, labelIdx);
    }
    /**
     * This function gives the bias term corresponding to the class with the
     * label_idx. For classification models, if label_idx is not in a valid range
     * (0 to nr_class-1), then a zero value will be returned; for regression
     * models, label_idx is ignored.
     *
     * @since 1.95
     */
    public double getDecfunBias(int labelIdx) {
```

```
        int biasIdx = nr_feature;
        if (bias <= 0) {
            return 0;
        } else {
            return bias * get_w_value(biasIdx, labelIdx);
        }
    }
    @Override
    public String toString() {
        StringBuilder sb = new StringBuilder("Model");
        sb.append(" bias=").append(bias);
        sb.append(" nr_class=").append(nr_class);
        sb.append(" nr_feature=").append(nr_feature);
        sb.append(" solverType=").append(solverType);
        return sb.toString();
    }
    @Override
    public int hashCode() {
        final int prime = 31;
        int result = 1;
        long temp;
        temp = Double.doubleToLongBits(bias);
        result = prime * result + (int)(temp ^ (temp >>> 32));
        result = prime * result + Arrays.hashCode(label);
        result = prime * result + nr_class;
        result = prime * result + nr_feature;
        result = prime * result + ((solverType == null) ? 0 : solverType.hashCode());
        result = prime * result + Arrays.hashCode(w);
        return result;
    }
    @Override
    public boolean equals(Object obj) {
        if (this == obj) return true;
        if (obj == null) return false;
        if (getClass() != obj.getClass()) return false;
        Model other = (Model)obj;
        if (Double.doubleToLongBits(bias) != Double.doubleToLongBits(other.bias))
return false;
        if (!Arrays.equals(label, other.label)) return false;
        if (nr_class != other.nr_class) return false;
        if (nr_feature != other.nr_feature) return false;
```

```
        if (solverType == null) {
            if (other.solverType != null) return false;
        } else if (!solverType.equals(other.solverType)) return false;
        if (!equals(w, other.w)) return false;
        return true;
    }
    /**
     * don't use {@link Arrays#equals(double[], double[])} here, cause 0.0 and -0.0
should be handled the same
     *
     * @see Linear#saveModel(java.io.Writer, Model)
     */
    protected static boolean equals(double[] a, double[] a2) {
        if (a == a2) return true;
        if (a == null || a2 == null) return false;
        int length = a.length;
        if (a2.length != length) return false;
        for (int i = 0; i < length; i++)
            if (a[i] != a2[i]) return false;
        return true;
    }
    /**
     * see {@link Linear#saveModel(java.io.File, Model)}
     */


    public Model loadModelBinary(String modelPath) throws IOException{
        return Linear.loadModelBinary(modelPath);
    }

    public void saveModelBinary(String modelPath) throws IOException {
        Linear.saveModelBinary(modelPath, this);
    }

    public void save(File file) throws IOException {
        Linear.saveModel(file, this);
    }

    /**
     * see {@link Linear#saveModel(Writer, Model)}
     */
```

```
    public void save(Writer writer) throws IOException {
        Linear.saveModel(writer, this);
    }
    /**
     * see {@link Linear#loadModel(File)}
     */

    public static Model load(File file) throws IOException {
        return Linear.loadModel(file);
    }
    /**
     * see {@link Linear#loadModel(Reader)}
     */
    public static Model load(Reader inputReader) throws IOException {
        return Linear.loadModel(inputReader);
    }
}
```

3. Parameter.java

```
/*
 * Copyright (c) 2007-2021 The LIBLINEAR Project.
 * All rights reserved.
 */
public final class Parameter {
    double       C;
    /** stopping criteria */
    double       eps;
    int max_iters = 1000;
    SolverType solverType;
    double[]    weight       = null;
    int[]       weightLabel = null;
    double       p = 0.1;
    /**
     * Initial-solution specification (only supported for {@link SolverType#L2R_LR}
and {@link SolverType#L2R_L2LOSS_SVC})
     */
    double[]    init_sol = null;
    public Parameter(SolverType solver, double C, double eps) {
        setSolverType(solver);
        setC(C);
        setEps(eps);
    }
    public Parameter(SolverType solver, double C, int max_iters, double eps) {
        setSolverType(solver);
        setC(C);
        setEps(eps);
        setMaxIters(max_iters);
    }
    public Parameter(SolverType solverType, double C, double eps, double p) {
        setSolverType(solverType);
        setC(C);
        setEps(eps);
        setP(p);
    }
    public Parameter(SolverType solverType, double C, double eps, int max_iters,
double p) {
        setSolverType(solverType);
```

```
        setC(C);
        setEps(eps);
        setMaxIters(max_iters);
        setP(p);
    }
    /**
     * <p>nr_weight, weight_label, and weight are used to change the penalty
     * for some classes (If the weight for a class is not changed, it is
     * set to 1). This is useful for training classifier using unbalanced
     * input data or with asymmetric misclassification cost.</p>
     *
     * <p>Each weight[i] corresponds to weight_label[i], meaning that
     * the penalty of class weight_label[i] is scaled by a factor of weight[i].</p>
     *
     * <p>If you do not want to change penalty for any of the classes,
     * just set nr_weight to 0.</p>
     */
    public void setWeights(double[] weights, int[] weightLabels) {
        if (weights == null) throw new IllegalArgumentException("'weight' must not be null");
        if (weightLabels == null || weightLabels.length != weights.length)
            throw new IllegalArgumentException("'weightLabels' must have same length as 'weight'");
        this.weightLabel = copyOf(weightLabels, weightLabels.length);
        this.weight = copyOf(weights, weights.length);
    }
    /**
     * @see #setWeights(double[], int[])
     */
    public double[] getWeights() {
        return copyOf(weight, weight.length);
    }
    /**
     * @see #setWeights(double[], int[])
     */
    public int[] getWeightLabels() {
        return copyOf(weightLabel, weightLabel.length);
    }
    /**
     * the number of weights
     * @see #setWeights(double[], int[])
```

```
     */
    public int getNumWeights() {
        if (weight == null) return 0;
        return weight.length;
    }
    /**
     * C is the cost of constraints violation. (we usually use 1 to 1000)
     */
    public void setC(double C) {
        if (C <= 0) throw new IllegalArgumentException("C must not be <= 0");
        this.C = C;
    }
    public double getC() {
        return C;
    }
    /**
     * eps is the stopping criterion. (we usually use 0.01).
     */
    public void setEps(double eps) {
        if (eps <= 0) throw new IllegalArgumentException("eps must not be <= 0");
        this.eps = eps;
    }
    public double getEps() {
        return eps;
    }
    public void setMaxIters(int iters) {
        if (iters <= 0) throw new IllegalArgumentException("max iters not be <= 0");
        this.max_iters = iters;
    }
    public int getMaxIters() {
        return max_iters;
    }
    public void setSolverType(SolverType solverType) {
        if (solverType == null) throw new IllegalArgumentException("solver type
must not be null");
        this.solverType = solverType;
    }
    public SolverType getSolverType() {
        return solverType;
    }
    /**
```

```
     * set the epsilon in loss function of epsilon-SVR (default 0.1)
     */
    public void setP(double p) {
        if (p < 0) throw new IllegalArgumentException("p must not be less than 0");
        this.p = p;
    }
    public double getP() {
        return p;
    }
}
```

4. Problem.java

```
/*
 * Copyright (c) 2007-2021 The LIBLINEAR Project.
 * All rights reserved.
 */
import java.io.File;
import java.io.IOException;
import java.io.InputStream;
import java.nio.charset.Charset;
public class Problem {
    /** the number of training data */
    public int           l;
    /** the number of features (including the bias feature if bias &gt;= 0) */
    public int           n;
    /** an array containing the target values */
    public double[]      y;
    /** array of sparse feature nodes */
    public Feature[][] x;
    /**
     * If bias &gt;= 0, we assume that one additional feature is added
     * to the end of each data instance
     */

    public double        bias;
    /**
     * see {@link Train#readProblem(File, double)}
     */
    public static Problem readFromFile(File file, double bias) throws IOException,
InvalidInputDataException {
        return Train.readProblem(file, bias);
    }
    /**
     * see {@link Train#readProblem(File, Charset, double)}
     */
    public static Problem readFromFile(File file, Charset charset, double bias)
throws IOException, InvalidInputDataException {
        return Train.readProblem(file, charset, bias);
    }
    /**
```

```
     * see {@link Train#readProblem(InputStream, double)}
     */
    public static Problem readFromStream(InputStream inputStream, double bias) throws IOException, InvalidInputDataException {
        return Train.readProblem(inputStream, bias);
    }
    /**
     * see {@link Train#readProblem(InputStream, Charset, double)}
     */
    public static Problem readFromStream(InputStream inputStream, Charset charset, double bias) throws IOException, InvalidInputDataException {
        return Train.readProblem(inputStream, charset, bias);
    }
}
```

5. Train.java

```
/*
 * Copyright (c) 2007-2021 The LIBLINEAR Project.
 * All rights reserved.
 */
import java.io.BufferedReader;
import java.io.File;
import java.io.FileInputStream;
import java.io.IOException;
import java.io.InputStream;
import java.io.InputStreamReader;
import java.nio.charset.Charset;
import java.util.ArrayList;
import java.util.List;
import java.util.NoSuchElementException;
import java.util.StringTokenizer;
public class Train {
	private double bias = 1;
	private boolean find_C = false;
	private boolean C_specified = false;
	private boolean solver_specified = false;
	private boolean cross_validation = false;
	public boolean isCross_validation() {
		return cross_validation;
	}
	public void setCross_validation(boolean cross_validation) {
		this.cross_validation = cross_validation;
	}
	private String inputFilename;
	private String modelFilename;
	private int nr_fold;
	public int getNr_fold() {
		return nr_fold;
	}
	public void setNr_fold(int nr_fold) {
		this.nr_fold = nr_fold;
	}
	private Parameter param = null;
	public Parameter getParam() {
```

```
		return param;
	}
	public void setParam(Parameter param) {
		this.param = param;
	}
	private Problem prob = null;
	public Problem getProb() {
		return prob;
	}
	public void setProb(Problem prob) {
		this.prob = prob;
	}
	private void do_find_parameter_C() {
	}
	public void do_cross_validation() {
	}
	private void exit_with_help() {
		System.exit(1);
	}
	Problem getProblem() {
		return prob;
	}
	double getBias() {
		return bias;
	}
	Parameter getParameter() {
		return param;
	}
	void parse_command_line(String argv[]) {
		int i;
		param = new Parameter(SolverType.L2R_L2LOSS_SVC_DUAL, 2.0,
Double.POSITIVE_INFINITY, 0.1);
		cross_validation = false;
		for (i = 0; i < argv.length; i++) {
			if (argv[i].charAt(0) != '-')
				break;
			if (++i >= argv.length)
				exit_with_help();
			switch (argv[i - 1].charAt(1)) {
			case 's':
				param.solverType = SolverType.getById(atoi(argv[i]));
```

```
                        solver_specified = true;
                        break;
                case 'c':
                        param.setC(atof(argv[i]));
                        C_specified = true;
                        break;
                case 'p':
                        param.setP(atof(argv[i]));
                        break;
                case 'e':
                        param.setEps(atof(argv[i]));
                        break;
                case 'B':
                        bias = atof(argv[i]);
                        break;
                case 'w':
                        int weightLabel = atoi(argv[i - 1].substring(2));
                        double weight = atof(argv[i]);
                        param.weightLabel = addToArray(param.weightLabel,
weightLabel);
                        param.weight = addToArray(param.weight, weight);
                        break;
                case 'v':
                        cross_validation = true;
                        nr_fold = atoi(argv[i]);
                        if (nr_fold < 2) {
                                System.err.println("n-fold cross validation: n
must >= 2");
                                exit_with_help();
                        }
                        break;
                case 'q':
                        i--;
                        Linear.disableDebugOutput();
                        break;
                case 'C':
                        find_C = true;
                        i--;
                        break;
                default:
                        System.err.println("unknown option");
```

```
                        exit_with_help();
                    }
                }
                if (i >= argv.length)
                    exit_with_help();
                inputFilename = argv[i];
                if (i < argv.length - 1)
                    modelFilename = argv[i + 1];
                else {
                    int p = argv[i].lastIndexOf('/');
                    ++p;
                    modelFilename = argv[i].substring(p) + ".model";
                }
                if (find_C) {
                    if (!cross_validation)
                        nr_fold = 3;
                    if (!solver_specified) {
                        System.err.printf("Solver not specified. Using -s 2%n");
                        param.setSolverType(SolverType.L2R_L2LOSS_SVC);
                    } else if (param.getSolverType() != SolverType.L2R_LR &&
param.getSolverType() != SolverType.L2R_L2LOSS_SVC) {
                        System.err.printf("Warm-start parameter search only available for -s 0 and -s 2%n");
                        exit_with_help();
                    }
                }
                if (param.eps == Double.POSITIVE_INFINITY) {
                    switch (param.solverType) {
                    case L2R_LR:
                    case L2R_L2LOSS_SVC:
                        param.setEps(0.01);
                        break;
                    case L2R_L2LOSS_SVR:
                        param.setEps(0.001);
                        break;
                    case L2R_L2LOSS_SVC_DUAL:
                    case L2R_L1LOSS_SVC_DUAL:
                    case MCSVM_CS:
                    case L2R_LR_DUAL:
                        param.setEps(0.1);
```

```
                        break;
                    case L1R_L2LOSS_SVC:
                    case L1R_LR:
                        param.setEps(0.01);
                        break;
                    case L2R_L1LOSS_SVR_DUAL:
                    case L2R_L2LOSS_SVR_DUAL:
                        param.setEps(0.1);
                        break;
                    default:
                        throw new IllegalStateException("unknown solver type: " + param.solverType);
                    }
                }
            }
            /**
             * reads a problem from LibSVM format
             *
             * @param file
             *            the SVM file
             * @throws IOException
             *             obviously in case of any I/O exception ;)
             * @throws InvalidInputDataException
             *             if the input file is not correctly formatted
             */
            public static Problem readProblem(File file, double bias) throws IOException, InvalidInputDataException {
                try (InputStream inputStream = new FileInputStream(file)) {
                    return readProblem(inputStream, bias);
                }
            }
            public static Problem readProblem(File file, Charset charset, double bias) throws IOException, InvalidInputDataException {
                try (InputStream inputStream = new FileInputStream(file)) {
                    return readProblem(inputStream, charset, bias);
                }
            }
            public static Problem readProblem(InputStream inputStream, double bias) throws IOException, InvalidInputDataException {
                return readProblem(inputStream, Charset.defaultCharset(), bias);
            }
```

```
	public static Problem readProblem(InputStream inputStream, Charset charset, double bias) throws IOException, InvalidInputDataException {
		BufferedReader fp = new BufferedReader(new InputStreamReader(inputStream, charset));
		List<Double> vy = new ArrayList<>();
		List<Feature[]> vx = new ArrayList<>();
		int max_index = 0;
		int lineNr = 0;
		while (true) {
			String line = fp.readLine();
			if (line == null)
				break;
			lineNr++;
			StringTokenizer st = new StringTokenizer(line, " \t\n\r\f:");
			String token;
			try {
				token = st.nextToken();
			} catch (NoSuchElementException e) {
				throw new InvalidInputDataException("empty line", lineNr, e);
			}
			try {
				vy.add(atof(token));
			} catch (NumberFormatException e) {
				throw new InvalidInputDataException("invalid label: " + token, lineNr, e);
			}
			int m = st.countTokens() / 2;
			Feature[] x;
			if (bias >= 0) {
				x = new Feature[m + 1];
			} else {
				x = new Feature[m];
			}
			int indexBefore = 0;
			for (int j = 0; j < m; j++) {
				token = st.nextToken();
				int index;
				try {
					index = atoi(token);
				} catch (NumberFormatException e) {
```

```
                                        throw new InvalidInputDataException("invalid
index: " + token, lineNr, e);
                                }
                                if (index <= 0)
                                        throw new InvalidInputDataException("invalid
index: " + index, lineNr);
                                if (index <= indexBefore)
                                        throw new InvalidInputDataException("indices
must be sorted in ascending order", lineNr);
                                indexBefore = index;
                                token = st.nextToken();
                                try {
                                        double value = atof(token);
                                        x[j] = new FeatureNode(index, value);
                                } catch (NumberFormatException e) {
                                        throw new InvalidInputDataException("invalid
value: " + token, lineNr);
                                }
                        }
                        if (m > 0) {
                                max_index = Math.max(max_index, x[m -
1].getIndex());
                        }
                        vx.add(x);
                }
                return constructProblem(vy, vx, max_index, bias);
        }
        public void readProblem(String filename) throws IOException,
InvalidInputDataException {
                readProblem(filename, bias);
        }
        public void readProblem(String filename, double bias) throws IOException,
InvalidInputDataException {
                prob = Train.readProblem(new File(filename), bias);
        }
        private static int[] addToArray(int[] array, int newElement) {
                int length = array != null ? array.length : 0;
                int[] newArray = new int[length + 1];
                if (array != null && length > 0) {
                        System.arraycopy(array, 0, newArray, 0, length);
                }
```

```
		newArray[length] = newElement;
		return newArray;
	}
	private static double[] addToArray(double[] array, double newElement) {
		int length = array != null ? array.length : 0;
		double[] newArray = new double[length + 1];
		if (array != null && length > 0) {
			System.arraycopy(array, 0, newArray, 0, length);
		}
		newArray[length] = newElement;
		return newArray;
	}
	private static Problem constructProblem(List<Double> vy, List<Feature[]> vx,
int max_index, double bias) {
		Problem prob = new Problem();
		prob.bias = bias;
		prob.l = vy.size();
		prob.n = max_index;
		if (bias >= 0) {
			prob.n++;
		}
		prob.x = new Feature[prob.l][];
		for (int i = 0; i < prob.l; i++) {
			prob.x[i] = vx.get(i);
			if (bias >= 0) {
				assert prob.x[i][prob.x[i].length - 1] == null;
				prob.x[i][prob.x[i].length - 1] = new
FeatureNode(max_index + 1, bias);
			}
		}
		prob.y = new double[prob.l];
		for (int i = 0; i < prob.l; i++)
			prob.y[i] = vy.get(i).doubleValue();
		return prob;
	}
	private void run(String[] args) throws IOException,
InvalidInputDataException {
		parse_command_line(args);
		readProblem(inputFilename);
		System.out.println("Data num : " + prob.l);
		System.out.println("feature num : " + prob.n);
```

```
                if (find_C) {
                        do_find_parameter_C();
                } else if (cross_validation)
                        do_cross_validation();
                else {
                        Model model = Linear.train(prob, param, 3);
                        System.out.println("C is " + param.getC());
                        String testFilename =
"D:/liblinearCode/data/selcted_tfidf_test.txt";
                        Problem testProb = Train.readProblem(new
File(testFilename), bias);
                        double total_correct = 0.0;
                        for (int i = 0; i < testProb.l; i++) {
                                if (Linear.predict(model, testProb.x[i]) ==
testProb.y[i])
                                        ++total_correct;
                        }
                        System.out.printf("Test set Accuracy = %g%%%n", 100.0 *
total_correct / (testProb.l));
                        Linear.saveModel(new File(args[0]), model);
                }
                System.gc();
        }
        boolean isFindC() {
                return find_C;
        }
        int getNumFolds() {
                return nr_fold;
        }
}
```