Project 3

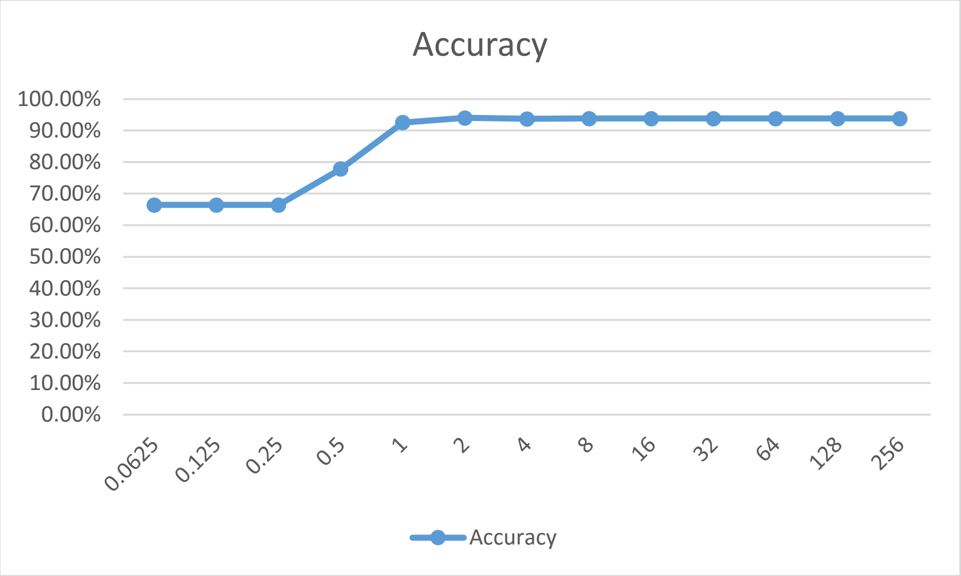
Jian Wang

I implemented this project in Java. I downloaded libSVM from the link and import the jar file for using libsvm. I also use two given source code, one is svm_train, the other is svm_predict. I believe that both of them are packaged into the libsvm jar file in order for us to use. I made some modifications on svm_predict source code. What I did basically is to store the accuracies for each iterations and rename it to predictor class.

1. Here is the result table

C	Accuracy
0.0625	66.43356643%
0.125	66.43356643%
0.25	66.43356643%
0.5	77.82217782%
1	92.50749251%
2	94.00599401%
4	93.70629371%
8	93.80619381%
16	93.80619381%
32	93.80619381%
64	93.80619381%
128	93.80619381%
256	93.80619381%

Here is the result graph



C = 2 is the best parameters with respect to accuracy

2. I first read and store all train data from train file, and then shuffle the data using build-in shuffle methods to get top 50% data from training set which indicated a random selection process. Then I divided the validation data into 5 parts to do the cross validation. For cross validation, I did exactly what we learnt in class and built 5 models for each alpha and C, and averaged the accuracies calculated by the 5 models. Here is the result table.

C\alpha	0.0625	0.125	0.25	0.5	1	2	4	8	16	32	64	128	256
0.0625	69.70%	69.70%	69.70%	69.70%	69.70%	69.70%	69.70%	69.70%	69.70%	69.70%	69.70%	69.70%	69.70%
0.125	69.70%	69.70%	69.70%	69.70%	69.70%	69.70%	69.70%	69.70%	69.70%	69.70%	69.70%	69.70%	69.70%
0.25	69.70%	69.70%	69.70%	69.70%	69.70%	69.70%	69.70%	72.20%	74.90%	75.00%	73.60%	70.10%	69.70%
0.5	69.70%	69.70%	69.70%	69.70%	69.70%	71.70%	78.30%	81.60%	82.50%	80.10%	76.90%	74.50%	70.20%
1	69.70%	69.70%	69.70%	69.70%	77.10%	88.20%	90.00%	90.20%	88.80%	86.70%	81.70%	78.20%	74.30%
2	69.70%	69.70%	70.40%	82.20%	92.70%	92.50%	92.20%	90.70%	90.10%	87.40%	84.20%	79.80%	75.30%
4	69.70%	70.70%	84.70%	93.30%	93.40%	93.40%	92.80%	91.10%	89.60%	88.30%	84.90%	79.80%	75.10%
8	70.80%	86.10%	93.70%	93.80%	93.90%	94.10%	93.20%	91.90%	90.20%	88.30%	84.40%	79.30%	75.10%
16	86.70%	93.90%	94.00%	94.10%	94.20%	94.30%	93.30%	92.10%	90.40%	87.40%	84.10%	79.30%	75.10%
32	94.30%	94.30%	94.10%	94.20%	94.70%	94.50%	93.10%	92.30%	90.20%	87.20%	83.80%	79.40%	75.10%
64	94.60%	94.50%	94.30%	93.90%	95.00%	94.50%	93.30%	92.10%	89.40%	86.30%	83.80%	79.40%	75.10%
128	94.30%	94.60%	94.30%	94.70%	95.20%	94.40%	93.10%	90.90%	88.60%	85.60%	83.70%	79.30%	75.20%
256	94.60%	94.60%	94.30%	94.70%	**95.60%**	94.20%	92.50%	90.60%	88.40%	85.50%	83.90%	79.30%	75.20%

As we can see, the best C is 256 and the best alpha is 1, so in this case, I apply those two parameters to the whole training set and get the result. The accuracy under C = 256 and alpha = 1 is 95.00499500499501%.

The source code is below. I generate the validation sets and do cross validation in Main class. I make some modifications on svm_predict and rename it to predictor.

```
import java.io.*;
import java.util.ArrayList;
import java.util.Collections;

public class Main {

    static ArrayList<Double> accuracies = new ArrayList<Double>();
    static ArrayList<String> lines = new ArrayList<>();
    static double[][] p2accuracies = new double[13][13];

    public static void seperateTrain() throws IOException {
        BufferedReader br = new BufferedReader(new FileReader("ncrna_s.train"));
        String line = br.readLine();

        while (line != null) {
            lines.add(line);
            line = br.readLine();
        }

        br.close();
        Collections.shuffle(lines);
        PrintWriter w1 = new PrintWriter("trainset0.txt", "UTF-8");
        PrintWriter w2 = new PrintWriter("trainset1.txt", "UTF-8");
        PrintWriter w3 = new PrintWriter("trainset2.txt", "UTF-8");
        PrintWriter w4 = new PrintWriter("trainset3.txt", "UTF-8");
        PrintWriter w5 = new PrintWriter("trainset4.txt", "UTF-8");

        for (int i = 0; i < 5; i++) {
            PrintWriter writer = new PrintWriter("testset" + i + ".txt", "UTF-8");
            for (int j = 0; j < 200; j++) {
                String temp = lines.get(i * 200 + j);
                if (i == 0) {
                    w2.println(temp);
                    w3.println(temp);
                    w4.println(temp);
                    w5.println(temp);
                }
                if (i == 1) {
                    w1.println(temp);
                    w3.println(temp);
                    w4.println(temp);
                    w5.println(temp);
                }
                if (i == 2) {
                    w1.println(temp);
                    w2.println(temp);
                    w4.println(temp);
                    w5.println(temp);
                }
                if (i == 3) {
                    w1.println(temp);
                    w2.println(temp);
                    w3.println(temp);
                    w5.println(temp);
                }
                if (i == 4) {
                    w1.println(temp);
                    w2.println(temp);
                    w3.println(temp);
                    w4.println(temp);
                }

                writer.println(temp);
            }
            writer.close();
        }
        w1.close();
        w2.close();
        w3.close();
        w4.close();
        w5.close();
    }

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

        String[] params = new String[] { "0.0625", "0.125", "0.25", "0.5", "1", "2", "4", "8", "16",
"32", "64", " 128",
                "256" };

        for (int i = 0; i < 13; i++) {
            String[] train_args = new String[] { "-t", "0", "-c", params[i], "ncrna_s.train",
"model.txt" };
            svm_train.main(train_args);
            String[] predict_args = new String[] { "ncrna_s.test", "model.txt", "output.txt" };
            Predictor predictor = new Predictor();
            predictor.main(predict_args);
            accuracies.add(predictor.accuracy);
        }
        PrintWriter writer = new PrintWriter("p1accuracy.txt", "UTF-8");

        for (int i = 0; i < accuracies.size(); i++) {
            writer.println(accuracies.get(i)+"%");
        }
        writer.close();

        seperateTrain();
        for (int i = 0; i < 13; i++) {
            for (int j = 0; j < 13; j++) {
                System.out.println(i*13+j);
                double aver = 0;
                for (int k = 0; k < 5; k++) {
                    String[] train_args = new String[] { "-t", "2", "-c", params[i], "-g", params[j],
                            "trainset" + k + ".txt", "model.txt" };
                    svm_train.main(train_args);
                    String[] predict_args = new String[] { "testset" + k + ".txt", "model.txt",
"output.txt" };

                    Predictor predictor = new Predictor();
                    predictor.main(predict_args);
                    aver += predictor.accuracy;
                }
                p2accuracies[i][j] = (aver / 5);
            }
        }

        PrintWriter p2w = new PrintWriter("p2accuracy.csv", "UTF-8");
        double max = 0;
        ArrayList<Integer> maxc = new ArrayList<Integer>();
        ArrayList<Integer> maxa = new ArrayList<Integer>();
        for (int i = 0; i < 13; i++) {
            for (int j = 0; j < 13; j++) {
                if (j == 12) p2w.print(p2accuracies[i][j]+"%");
                else p2w.print(p2accuracies[i][j] + "%,");
                if (max < p2accuracies[i][j]){
                    max = p2accuracies[i][j];
                }
            }
            p2w.println();
        }


        for (int i = 0; i < 13; i++) {
            for (int j = 0; j < 13; j++) {
                if (max == p2accuracies[i][j]){
                    maxc.add(i);
                    maxa.add(j);
                }
            }
        }
        p2w.close();
        PrintWriter p3w = new PrintWriter("p3accuracy.txt", "UTF-8");
        System.out.println("best C is " + maxc + ", best Gamma is " + maxa);
        p3w.println("best C is " + maxc + ", best Gamma is " + maxa);
        for (int i = 0; i < maxc.size(); i++) {
            String[] train_args = new String[] { "-t", "2", "-c", params[maxc.get(i)], "-g",
params[maxa.get(i)],"ncrna_s.train", "model.txt" };
            svm_train.main(train_args);
            String[] predict_args = new String[] { "ncrna_s.test", "model.txt", "output.txt" };
            Predictor predictor = new Predictor();
            predictor.main(predict_args);

            p3w.println("C is " + params[maxc.get(i)] + " a is " + params[maxa.get(i)] +" accuracy
is "+ predictor.accuracy+"%");
        }

        p3w.close();
    }
}
```

Predictor class

```
import libsvm.*;
import java.io.*;
import java.util.*;

class Predictor {
    public double accuracy;

    private svm_print_interface svm_print_null = new svm_print_interface() {
        public void print(String s) {
        }
    };

    private static svm_print_interface svm_print_stdout = new svm_print_interface() {
        public void print(String s) {
            System.out.print(s);
        }
    };

    private static svm_print_interface svm_print_string = svm_print_stdout;

    static void info(String s) {
        svm_print_string.print(s);
    }

    private double atof(String s) {
        return Double.valueOf(s).doubleValue();
    }

    private int atoi(String s) {
        return Integer.parseInt(s);
    }

    private void predict(BufferedReader input, DataOutputStream output, svm_model model, int
predict_probability)
            throws IOException {
        int correct = 0;
        int total = 0;
        double error = 0;
        double sumv = 0, sumy = 0, sumvv = 0, sumyy = 0, sumvy = 0;

        int svm_type = svm.svm_get_svm_type(model);
        int nr_class = svm.svm_get_nr_class(model);
        double[] prob_estimates = null;

        if (predict_probability == 1) {
            if (svm_type == svm_parameter.EPSILON_SVR || svm_type ==
svm_parameter.NU_SVR) {
                Predictor
                        .info("Prob. model for test data: target value = predicted value + z,\nz:
Laplace distribution e^(-|z|/sigma)/(2sigma),sigma="
                                + svm.svm_get_svr_probability(model) + "\n");
            } else {
                int[] labels = new int[nr_class];
                svm.svm_get_labels(model, labels);
                prob_estimates = new double[nr_class];
                output.writeBytes("labels");
                for (int j = 0; j < nr_class; j++)
                    output.writeBytes(" " + labels[j]);
                output.writeBytes("\n");
            }
        }
        while (true) {
            String line = input.readLine();
            if (line == null)
                break;

            StringTokenizer st = new StringTokenizer(line, " \t\n\r\f:");

            double target = atof(st.nextToken());
            int m = st.countTokens() / 2;
            svm_node[] x = new svm_node[m];
            for (int j = 0; j < m; j++) {
                x[j] = new svm_node();
                x[j].index = atoi(st.nextToken());
                x[j].value = atof(st.nextToken());
            }

            double v;
            if (predict_probability == 1 && (svm_type == svm_parameter.C_SVC || svm_type ==
svm_parameter.NU_SVC)) {
                v = svm.svm_predict_probability(model, x, prob_estimates);
                output.writeBytes(v + " ");
                for (int j = 0; j < nr_class; j++)
                    output.writeBytes(prob_estimates[j] + " ");
                output.writeBytes("\n");
            } else {
                v = svm.svm_predict(model, x);
                output.writeBytes(v + "\n");
            }

            if (v == target)
                ++correct;
            error += (v - target) * (v - target);
            sumv += v;
            sumy += target;
            sumvv += v * v;
            sumyy += target * target;
            sumvy += v * target;
            ++total;
        }
        if (svm_type == svm_parameter.EPSILON_SVR || svm_type == svm_parameter.NU_SVR)
{
```

```
            Predictor.info("Mean squared error = " + error / total + " (regression)\n");
            Predictor
                    .info("Squared correlation coefficient = "
                            + ((total * sumvy - sumv * sumy) * (total * sumvy - sumv * sumy))
                                    / ((total * sumvv - sumv * sumv) * (total * sumyy -
sumy * sumy))
                            + " (regression)\n");
        } else {
            Predictor.info("Accuracy = " + (double) correct / total * 100 + "% (" + correct + "/" +
total
                    + ") (classification)\n");
            accuracy = ((double)correct/total * 100);
        }

    }

    private void exit_with_help() {
        System.err.print("usage: svm_predict [options] test_file model_file output_file\n" +
"options:\n"
                + "-b probability_estimates: whether to predict probability estimates, 0 or 1
(default 0); one-class SVM not supported yet\n"
                + "-q : quiet mode (no outputs)\n");
        System.exit(1);
    }

    public void main(String argv[]) throws IOException {
        int i, predict_probability = 0;
        svm_print_string = svm_print_stdout;

        // parse options
        for (i = 0; i < argv.length; i++) {
            if (argv[i].charAt(0) != '-')
                break;
            ++i;
            switch (argv[i - 1].charAt(1)) {
            case 'b':
                predict_probability = atoi(argv[i]);
                break;
            case 'q':
                svm_print_string = svm_print_null;
                i--;
                break;
            default:
                System.err.print("Unknown option: " + argv[i - 1] + "\n");
                exit_with_help();
            }
        }
        if (i >= argv.length - 2)
            exit_with_help();
        try {
            BufferedReader input = new BufferedReader(new FileReader(argv[i]));
            DataOutputStream output = new DataOutputStream(new
BufferedOutputStream(new FileOutputStream(argv[i + 2])));
            svm_model model = svm.svm_load_model(argv[i + 1]);
            if (model == null) {
                System.err.print("can't open model file " + argv[i + 1] + "\n");
                System.exit(1);
            }
            if (predict_probability == 1) {
                if (svm.svm_check_probability_model(model) == 0) {
                    System.err.print("Model does not support probabiliy estimates\n");
                    System.exit(1);
                }
            } else {
                if (svm.svm_check_probability_model(model) != 0) {
                    Predictor.info("Model supports probability estimates, but disabled in
prediction.\n");
                }
            }
            predict(input, output, model, predict_probability);
            input.close();
            output.close();
        } catch (FileNotFoundException e) {
            exit_with_help();
        } catch (ArrayIndexOutOfBoundsException e) {
            exit_with_help();
        }
    }
}
```