Appendix A. Datasets

All datasets are stored under *src/main/resources/datasets*. While Java class codes are stored under *src/main/java*, user resources are stored under *src/main/resources*. In general, we use the JAR loader functionality to retrieve contents of a file directly from the JAR, not from the filesystem.

Anscombe's Quartet

Anscombe's quartet is a set of four *x-y* pairs of data with remarkable properties. Although the x-y plots of each pair look completely different, the data has the properties that make statistical measures almost identical. The values for each of the four x-y data series are in Table A-1.

Table A-1. Anscombe's quartet data

x1	y1	x2	y2	x3	y3	x4	y4
10.0	8.04	10.0	9.14	10.0	7.46	8.0	6.58
8.0	6.95	8.0	8.14	8.0	6.77	8.0	5.76
13.0	7.58	13.0	8.74	13.0	12.74	8.0	7.71
9.0	8.81	9.0	8.77	9.0	7.11	8.0	8.84
11.0	8.33	11.0	9.26	11.0	7.81	8.0	8.47
14.0	9.96	14.0	8.10	14.0	8.84	8.0	7.04
6.0	7.24	6.0	6.13	6.0	6.08	8.0	5.25
4.0	4.26	4.0	3.10	4.0	5.39	19.0	12.50
12.0	10.84	12.0	9.13	12.0	8.15	8.0	5.56
7.0	4.82	7.0	7.26	7.0	6.42	8.0	7.91
5.0	5.68	5.0	4.74	5.0	5.73	8.0	6.89

We can easily hardcode the data as static members of the class:

```
public class Anscombe {
    public static final double[] x1 = {10.0, 8.0, 13.0, 9.0, 11.0,
                                       14.0, 6.0, 4.0, 12.0, 7.0, 5.0};
    public static final double[] y1 = {8.04, 6.95, 7.58, 8.81, 8.33,
                                       9.96, 7.24, 4.26, 10.84, 4.82, 5.68};
    public static final double[] x2 = {10.0, 8.0, 13.0, 9.0, 11.0,
                                       14.0, 6.0, 4.0, 12.0, 7.0, 5.0};
    public static final double[] y2 = {9.14, 8.14, 8.74, 8.77, 9.26,
                                       8.10, 6.13, 3.10, 9.13, 7.26, 4.74};
    public static final double[] x3 = {10.0, 8.0, 13.0, 9.0, 11.0,
                                       14.0, 6.0, 4.0, 12.0, 7.0, 5.0};
    public static final double[] y3 = {7.46, 6.77, 12.74, 7.11, 7.81,
                                       8.84, 6.08, 5.39, 8.15, 6.42, 5.73};
    public static final double[] x4 = {8.0, 8.0, 8.0, 8.0, 8.0, 8.0,
                                       8.0, 19.0, 8.0, 8.0, 8.0};
    public static final double[] y4 = {6.58, 5.76, 7.71, 8.84, 8.47,
                                       7.04, 5.25, 12.50, 5.56, 7.91, 6.89};
}
```

Then we can call any array:

```
double[] x1 = Anscombe.x1;
```

Sentiment

This is the sentiment-labeled dataset from *https://archive.ics.uci.edu/ml/datasets/Sentiment+Labelled+Sentences*. Download three files and place them in *src/main/resources/datasets/sentiment*. They contain data from IMDb, Yelp, and Amazon. There is a single sentence and then a tab-delimited 0 or 1 corresponding to respective negative or positive sentiment. Not all sentences have a corresponding label.

IMDb has 1,000 sentences, with 500 positive (1) and 500 negative (0). Yelp has 3,729 sentences, with 500 positive (1) and 500 negative (0). Amazon has 15,004 sentences, with 500 positive (1) and 500 negative (0):

```
public class Sentiment {

    private final List<String> documents = new ArrayList<>();
    private final List<Integer> sentiments = new ArrayList<>();
    private static final String IMDB_RESOURCE =
    "/datasets/sentiment/imdb_labelled.txt";
    private static final String YELP_RESOURCE =
    "/datasets/sentiment/yelp_labelled.txt";
    private static final String AMZN_RESOURCE =
    "/datasets/sentiment/amazon_cells_labelled.txt";
    public enum DataSource {IMDB, YELP, AMZN};

    public Sentiment() throws IOException {
        parseResource(IMDB_RESOURCE); // 1000 sentences
        parseResource(YELP_RESOURCE); // 1000 sentences
        parseResource(AMZN_RESOURCE); // 1000 sentences
    }

    public List<Integer> getSentiments(DataSource dataSource) {
        int fromIndex = 0; // inclusive
        int toIndex = 3000; // exclusive
        switch(dataSource) {
            case IMDB:
                fromIndex = 0;
                toIndex = 1000;
                break;
            case YELP:
                fromIndex = 1000;
                toIndex = 2000;
                break;
            case AMZN:
                fromIndex = 2000;
                toIndex = 3000;
```

```
                break;
            }
            return sentiments.subList(fromIndex, toIndex);
        }

        public List<String> getDocuments(DataSource dataSource) {
            int fromIndex = 0; // inclusive
            int toIndex = 3000; // exclusive
            switch(dataSource) {
                case IMDB:
                    fromIndex = 0;
                    toIndex = 1000;
                    break;
                case YELP:
                    fromIndex = 1000;
                    toIndex = 2000;
                    break;
                case AMZN:
                    fromIndex = 2000;
                    toIndex = 3000;
                    break;
            }
            return documents.subList(fromIndex, toIndex);
        }

        public List<Integer> getSentiments() {
            return sentiments;
        }

        public List<String> getDocuments() {
            return documents;
        }

        private void parseResource(String resource) throws IOException {
            try(InputStream inputStream = getClass().getResourceAsStream(resource)) {
                BufferedReader br =
                    new BufferedReader(new InputStreamReader(inputStream));
                String line;
                while ((line = br.readLine()) != null) {
                    String[] splitLine = line.split("\t");
                    // both yelp and amzn have many sentences with no label
                    if (splitLine.length > 1) {
                        documents.add(splitLine[0]);
                        sentiments.add(Integer.parseInt(splitLine[1]));
                    }
                }
            }
        }
    }
```

Gaussian Mixtures

Generate a mixture of multivariate normal distributions data:

```
    public class MultiNormalMixtureDataset {
        int dimension;
        List<Pair<Double, MultivariateNormalDistribution>> mixture;
        MixtureMultivariateNormalDistribution mixtureDistribution;

        public MultiNormalMixtureDataset(int dimension) {
            this.dimension = dimension;
            mixture = new ArrayList<>();
        }

        public MixtureMultivariateNormalDistribution getMixtureDistribution() {
            return mixtureDistribution;
        }

        public void createRandomMixtureModel(
        int numComponents, double boxSize, long seed) {
            Random rnd = new Random(seed);
            double limit = boxSize / dimension;
            UniformRealDistribution dist =
```

```
            new UniformRealDistribution(-limit, limit);
        UniformRealDistribution disC = new UniformRealDistribution(-1, 1);
        dist.reseedRandomGenerator(seed);
        disC.reseedRandomGenerator(seed);

        for (int i = 0; i < numComponents; i++) {
            double alpha = rnd.nextDouble();
            double[] means = dist.sample(dimension);
            double[][] cov = getRandomCovariance(disC);
            MultivariateNormalDistribution multiNorm =
            new MultivariateNormalDistribution(means, cov);
            addMultinormalDistributionToModel(alpha, multiNorm);
        }

        mixtureDistribution = new MixtureMultivariateNormalDistribution(mixture);
        mixtureDistribution.reseedRandomGenerator(seed);
        // calls to sample() will return same results
    }

    /**
     * NOTE this is for adding both internal and external, known distros but
     * need to figure out clean way to add the mixture to mixtureDistribution!!!
     * @param alpha
     * @param dist
     */
    public void addMultinormalDistributionToModel(
    double alpha, MultivariateNormalDistribution dist) {
        // note all alpha will be L1 normed
        mixture.add(new Pair(alpha, dist));
    }

    public double[][] getSimulatedData(int size) {
        return mixtureDistribution.sample(size);
    }

    private double[] getRandomMean(int dimension, double boxSize, long seed) {
        double limit = boxSize / dimension;
        UniformRealDistribution dist =
            new UniformRealDistribution(-limit, limit);
        dist.reseedRandomGenerator(seed);
        return dist.sample(dimension);
    }

    private double[][] getRandomCovariance(AbstractRealDistribution dist) {
        double[][] data = new double[2*dimension][dimension];
        double determinant = 0.0;
        Covariance cov = new Covariance();
        while(Math.abs(determinant) == 0) {
            for (int i = 0; i < data.length; i++) {
                data[i] = dist.sample(dimension);
            }
            // check if cov matrix is singular ... if so, keep going
            cov = new Covariance(data);
            determinant = new CholeskyDecomposition(
            cov.getCovarianceMatrix()).getDeterminant();

        }
        return cov.getCovarianceMatrix().getData();
    }

}
```

Iris

Iris is the famous dataset containing measurements of irises and three types:

```
package com.datascience.javabook.datasets;

import java.io.BufferedReader;
import java.io.FileReader;
import java.io.IOException;
import java.util.ArrayList;
import java.util.List;
```

```
/**
 * Sentiment-Labeled Sentences
 * https://archive.ics.uci.edu/ml/datasets/Sentiment+Labelled+Sentences
 * @author mbrzusto
 */
public class IMDB {

    private final List<String> documents = new ArrayList<>();
    private final List<Integer> sentiments = new ArrayList<>();
    private static final String FILEPATH = "datasets/imdb/imdb_labelled.txt";

    public IMDB() throws IOException {
        ClassLoader classLoader = getClass().getClassLoader();
        String filename = classLoader.getResource(FILEPATH).getFile();
        try(BufferedReader br = new BufferedReader(new FileReader(filename))) {
            String line;
            while ((line = br.readLine()) != null) {
                String[] splitLine = line.split("\t");
                documents.add(splitLine[0]);
                sentiments.add(Integer.parseInt(splitLine[1]));
            }
        }
    }

    public List<Integer> getSentiments() {
        return sentiments;
    }

    public List<String> getDocuments() {
        return documents;
    }
}
```

MNIST

The Modified National Institute of Standards (MNIST) database is the famous handwritten digits dataset of 70,000 images of digits 0 through 9. Of these, 60,000 are in a training set, and 10,000 are in a test set. The first 5,000 are good, and the last 5,000 are hard to read. All data is labeled.

All the integers in the files are stored in the MSB (Most Significant Bit) first (high endian) format used by most non-Intel processors. Users of Intel processors and other low-endian machines must flip the bytes of the header.

There are four files:

- *train-images-idx3-ubyte*: training set images
- *train-labels-idx1-ubyte*: training set labels
- *t10k-images-idx3-ubyte*: test set images
- *t10k-labels-idx1-ubyte*: test set labels

The training set contains 60,000 examples, and the test set 10,000 examples. The first 5,000 examples of the test set are taken from the original MNIST training set. The last 5,000 are taken from the original MNIST test set. The first 5,000 are cleaner and easier to read than the last 5,000.

```
public class MNIST {

    public RealMatrix trainingData;
    public RealMatrix trainingLabels;
    public RealMatrix testingData;
    public RealMatrix testingLabels;

    public MNIST() throws IOException {
        trainingData = new BlockRealMatrix(60000, 784); // image to vector
```

```
    trainingLabels = new BlockRealMatrix(60000, 10); // the one hot label
    testingData = new BlockRealMatrix(10000, 784); // image to vector
    testingLabels = new BlockRealMatrix(10000, 10); // the one hot label
    loadData();
}

private void loadData() throws IOException {
    ClassLoader classLoader = getClass().getClassLoader();
    loadTrainingData(classLoader.getResource(
    "datasets/mnist/train-images-idx3-ubyte").getFile());
    loadTrainingLabels(classLoader.getResource(
    "datasets/mnist/train-labels-idx1-ubyte").getFile());
    loadTestingData(classLoader.getResource(
    "datasets/mnist/t10k-images-idx3-ubyte").getFile());
    loadTestingLabels(classLoader.getResource(
    "datasets/mnist/t10k-labels-idx1-ubyte").getFile());
}

private void loadTrainingData(String filename)
throws FileNotFoundException, IOException {
    try (DataInputStream di = new DataInputStream(
    new BufferedInputStream(new FileInputStream(filename)))) {
        int magicNumber = di.readInt(); //2051
        int numImages = di.readInt(); // 60000
        int numRows = di.readInt(); // 28
        int numCols = di.readInt(); // 28
        for (int i = 0; i < numImages; i++) {
            for (int j = 0; j < 784; j++) {
                // values are 0 to 255, so normalize
                trainingData.setEntry(i, j, di.readUnsignedByte() / 255.0);
            }
        }
    }
}

private void loadTestingData(String filename)
throws FileNotFoundException, IOException {
    try (DataInputStream di = new DataInputStream(
    new BufferedInputStream(new FileInputStream(filename)))) {
        int magicNumber = di.readInt(); //2051
        int numImages = di.readInt(); // 10000
        int numRows = di.readInt(); // 28
        int numCols = di.readInt(); // 28
        for (int i = 0; i < numImages; i++) {
            for (int j = 0; j < 784; j++) {
                // values are 0 to 255, so normalize
                testingData.setEntry(i, j, di.readUnsignedByte() / 255.0);
            }
        }
    }
}

private void loadTrainingLabels(String filename)
throws FileNotFoundException, IOException {
    try (DataInputStream di = new DataInputStream(
    new BufferedInputStream(new FileInputStream(filename)))) {
        int magicNumber = di.readInt(); //2049
        int numImages = di.readInt(); // 60000
        for (int i = 0; i < numImages; i++) {
            // one-hot-encoding, column of 0-9 is given one all else 0
            trainingLabels.setEntry(i, di.readUnsignedByte(), 1.0);
        }
    }
}

private void loadTestingLabels(String filename)
throws FileNotFoundException, IOException {
    try (DataInputStream di = new DataInputStream(
    new BufferedInputStream(new FileInputStream(filename)))) {
        int magicNumber = di.readInt(); //2049
        int numImages = di.readInt(); // 10000
        for (int i = 0; i < numImages; i++) {
            // one-hot-encoding, column of 0-9 is given one all else 0
            testingLabels.setEntry(i, di.readUnsignedByte(), 1.0);
        }
    }
```

```
    }
}
```

PREV
6. Hadoop MapReduce

NEXT
Index