```java
package edu.gmu.classifier.naivebayes;

import java.io.File;
import java.io.FilenameFilter;
import java.io.IOException;
import java.util.ArrayList;
import java.util.Arrays;
import java.util.HashMap;
import java.util.List;
import java.util.Map;

import edu.gmu.classifier.io.DataLoader;
import edu.gmu.classifier.io.TrainingExample;
import edu.gmu.classifier.neuralnet.util.NeuralNetUtils;

public class Homework7
{
	public static void main( String[] args ) throws IOException
	{
		// list the test data files
		File dataDirectory = new File( "/home/ulman/CSI873/midterm/data" );
		String[] testDataFiles = dataDirectory.list( new FilenameFilter( )
		{
			@Override
			public boolean accept( File dir, String name )
			{
				return name.startsWith( "test" );
			}
		} );

		// sort the testDataFiles
		Arrays.sort( testDataFiles );

		// list the training data files
		String[] trainingDataFiles = dataDirectory.list( new FilenameFilter( )
		{
			@Override
			public boolean accept( File dir, String name )
			{
				return name.startsWith( "train" );
			}
		} );

		// sort the trainingDataFiles
		Arrays.sort( trainingDataFiles );

		// load all test data examples
		List<TrainingExample> testDataList = new ArrayList<TrainingExample>( );
		for ( String fileName : testDataFiles )
		{
			testDataList.addAll( DataLoader.loadFile( new File( dataDirectory, fileName ) ) );
		}

		// load all training data examples
		List<TrainingExample> trainingDataListAll = new ArrayList<TrainingExample>( );
		final Map<Integer, List<TrainingExample>> trainingDataMap = new HashMap<Integer, List<TrainingExample>>( );
		for ( String fileName : trainingDataFiles )
		{
			List<TrainingExample> list = DataLoader.loadFile( new File( dataDirectory, fileName ) );
			int digit = list.get( 0 ).getDigit( );
			trainingDataMap.put( digit, list );
			trainingDataListAll.addAll( list );
		}

		// create a data structure to store the training frequencies
		Map<P, Double> p0map = new HashMap<P, Double>( );

		// loop over digits and input indices, calculating the conditional probabilities:
		// p( index i = 0 | digit = d )
		for ( int d = 0; d < 10; d++ )
		{
			List<TrainingExample> examplesForDigit = trainingDataMap.get( d );

			for ( int i = 0; i < DataLoader.INPUT_SIZE; i++ )
			{
				P key = new P( i, d );
				Double p = calculateTrainingProbability( examplesForDigit, d, i );
				p0map.put( key, p );
			}
		}

		// calculate error and 95% confidence interval for test, training, and validation data sets
		double trainErrorRate = calculateErrorRate( p0map, trainingDataListAll );
		double testErrorRate = calculateErrorRate( p0map, testDataList );

		System.out.printf( "Training Error Rate: %.3f Testing ErrorRate: %.3f%n", trainErrorRate, testErrorRate );
	}
```

```
	/**
	 * Calculate the conditional probability p( index i = 0 | digit = d )
	 *
	 * @param examplesForDigit the set of training digits for digit d
	 * @param digit
	 * @param inputIndex
	 * @return the probability p( index i = 0 | digit = d )
	 */
	public static double calculateTrainingProbability( List<TrainingExample> examplesForDigit, int digit, int inputIndex )
	{
		int count = 0;

		for ( TrainingExample example : examplesForDigit )
		{
			if ( example.getInputs( )[inputIndex] == 0.0 ) count++;
		}

		if ( count == 0 )
		{
			System.out.printf( "Zero probability warning: index: %d digit: %d%n", inputIndex, digit );
		}

		return (double) count / ( double ) examplesForDigit.size( );
	}

	public static double calculateErrorRate( Map<P, Double> p0map, List<TrainingExample> dataList )
	{
		int correctCount = 0;

		for ( TrainingExample data : dataList )
		{
			double[] likelihoods = calculateOutputLikelihoods( p0map, data );
			int predictedDigit = NeuralNetUtils.getLargestIndex( likelihoods );

			if ( predictedDigit == data.getDigit( ) )
			{
				correctCount++;
			}
		}

		return 1.0 - ( double ) correctCount / (double) dataList.size( );
	}

	/**
	 * Use the pre-calculated conditional probabilities stored in p0map to calculate the probability
	 * that the TrainingExample represents each of the ten possible digits.
	 *
	 * @param p0map a map of pre-computed conditional probabilities p( index i = 0 | digit = d )
	 * @param data the training data element
	 * @return the probability that data represents each of the possible ten digits
	 */
	public static double[] calculateOutputLikelihoods( Map<P, Double> p0map, TrainingExample data )
	{
		// allocate an array to return the 10 calculated probabilities for each digit
		double[] likelihoods = new double[10];
		// get the input data for the training example
		double[] input = data.getInputs( );

		// apply the naive bayes classifier using the pre-calculated conditional probabilities
		// the result is one likelihood value for each digit
		for ( int d = 0; d < 10; d++ )
		{
			// in our case, all the digits have the same number of values, so p(v) is always 0.1
			double likelihood = 0.1;

			for ( int i = 0 ; i < DataLoader.INPUT_SIZE; i++ )
			{
				// get the pre-computed conditional probability for the current digit and input index
				double p = p0map.get( new P( i, d ) );
				// the pre-computed probabilities are for input = 0, subtract one minus the probability
				// if the input value at the index is a 1
				likelihood *= input[i] == 0 ? p : 1-p;
			}

			likelihoods[d] = likelihood;
		}

		return likelihoods;
	}
}
```