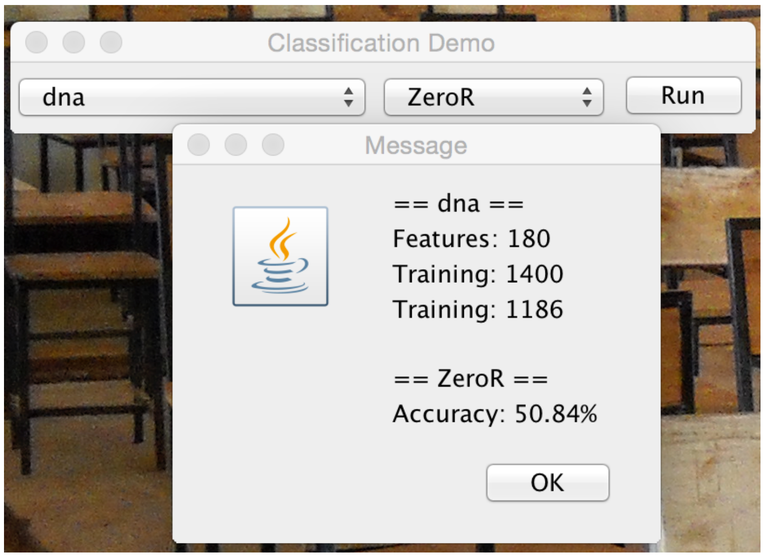
Wentworth Institute of Technology
COMP1050 – Computer Science II
Spring 2015, Derbinsky

Final Project

For the final project you are to implement a fully functioning program in Java. The program will require implementing several classes, including one that provides the user with a graphical user interface (GUI). You must document the classes using Javadoc and turn in the source files, in the appropriate folder structure, as a zip file. You will be evaluated based upon correctness (adhering to the problem description, passing test cases), object-oriented design (e.g. appropriate usage of inheritance, member access control, etc.), and source code documentation (via Javadoc).

1 Project Overview

In this project you will implement the building blocks necessary to evaluate *Machine Learning*[1] (ML) algorithms. In particular, you are going to work on classification, "the problem of identifying to which of a set of categories (sub-populations) a new observation belongs, on the basis of a training set of data containing observations (or instances) whose category membership is known"[2]. ML and classification algorithms are becoming ubiquitous, because they apply to many types of intelligent systems (e.g. recommendation engines, image/voice recognition, text translation, . . .) and they allow your program to solve problems by *learning* from examples, which is especially useful when solutions are difficult to encode directly as code. An example run of your final project is depicted below.



```
== dna ==
Features: 180
Training: 1400
Training: 1186

== ZeroR ==
Accuracy: 50.84%
```

To build up to this system, this project is divided into sub-parts that focus on logical subsets of this functionality. In total, you will have at least 16 source files – a considerable project indeed, and one which touches on every part of object-oriented programming (OOP)! The remainder of this document describes each class you will implement, and its relation to Machine Learning. You should also look to the included `final-doc.zip` file for an in-depth breakdown of all the components involved.

[1]See `http://en.wikipedia.org/wiki/Machine_learning`
[2]See `http://en.wikipedia.org/wiki/Statistical_classification`

1.1 Examples

At the core of every ML algorithm is the concept of an *example* (also referred to as an *instance* or *observation*). Each example is composed of a sequence of *features*, typically numbers that correspond to some real-world aspect of the object, as well as a an associated *result*, which depends on the task (for classification, the result is a single *class* or *category*, while in *regression* problems it is a real number). A set of examples forms a *dataset*. To facilitate interaction with other tools and dataset files, we will adopt the LibSVM data format[3] for inputting/outputting examples and datasets.

Consider, for instance, the `Iris` dataset[4], which is composed of 150 examples, each with 4 features (sepal length in cm, sepal width in cm, petal length in cm, petal width in cm) and the result is the type of iris plant (Setosa, Versicolour, or Virginica).

The process of classification is to (i) *train* on some examples and then (ii) attempt to accurately predict future *test* examples. For example, you might be given 100 Iris examples, with the correct type of plant, and then be asked to classify the remaining 50. There are many *algorithms* that seek to solve this problem, but for the most part they operate in the sequence above, and all work with sets of examples. Thus, in your project, you are going to implement all the building blocks surrounding examples, features, datasets, and algorithms, and then have a GUI that allows you to mix and match datasets and algorithms – a perfect demonstration of object-oriented programming!

1.1.1 FeatureEntry

We begin with the `FeatureEntry` interface, which is an abstraction to facilitate a sequence of features, without enforcing an implementation.

1.1.2 TestingExample

We then build up to the `TestingExample` interface, which provides the necessary aspects of an example (i.e. enumeration and iteration of features), but does not provide access to the result classification (this is useful for the *testing* phase of classification).

1.1.3 Example

Next we have the `Example` base class, which provides many useful functions, but is still abstract, as it does not require an implementation of feature storage.

1.1.4 DenseExample

The most straightforward implementation of an example is via an array, which is what `DenseExample` does. This class is fully instantiable, and appropriate when most if not all features are non-zero valued.

1.1.5 SparseExample

In many datasets, it is common that there are many features, but most examples make use of relatively few (i.e. the remaining are zero-valued) – this is the ideal circumstance for the `SparseExample` class, which stores a sorted pairing of feature number-value pairs. Exploiting this representation can be a big memory and time savings for large datasets.

[3]See `http://www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets/`
[4]See `http://www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets/multiclass.html#iris`

1.2 Datasets

Now we move up to a collection of examples – a *dataset*.

1.2.1 DataSet

The `DataSet` class provides an abstraction for iterating over examples, without committing to a particular backend mechanism of storage/representation.

1.2.2 ListDataSet

Many sources of examples (e.g. files) would work well in a list, and so a `ListDataSet` is an abstract class that facilitates such storage, using an `ArrayList` backend.

1.2.3 LibSVMDataSet

The `LibSVMDataSet` builds on the `ListDataSet`: it parses an input stream, in LibSVM format, and adds each example to the dataset. Each example starts with the result object and, to provide maximum flexibility, the LibSVMDataSet takes as an argument a `ResultInterpreter` object to convert the first string on each file into an appropriate Java object (e.g. an integer). The LibSVMDataSet provides the `ResultInterpreter` interface, as well as two static implementations, one for strings (`StringInterpreter`) and the other for integers (`IntegerInterpreter`).

1.2.4 HiddenPatternDataSet

The last dataset class you are to implement is the `HiddenPatternDataSet`, which allows you to define a dataset via simple patterns. Each pattern states that for a particular result object, there are a certain set of features that will have a fixed value. For example, all "dog" examples will have feature 1 equal to 1 and feature 3 equal to 2, whereas all "cat" examples will have feature 2 equal to 1 and feature 4 equal to 2. Any unspecified features will either be 0 or a random value in a known range, as defined by class configuration.

This dataset has [at least] two interesting aspects. First, the object **should not store any examples**, but should instead generate them reliably and repeatedly on demand. Thus, no matter how many examples are generated, the amount of memory needed by the class should only be as big as the number of patterns provided by the user. Second, if examples are made that use 0 for unspecified features, the class should generate `SparseExample` objects; if instead the features are random-valued, the class should generate `DenseExample` objects.

1.3 Classifiers

Given datasets of examples, we can now consider the problems of learning and classifying!

1.3.1 TieBreaker

While not a classifier in and of itself, the `TieBreaker` class is useful in a variety of contexts: it will help you solve the problem of finding the "best" item in a *stream* of items in which you don't know the length ahead of time (and/or it is very large).

1.3.2 Classifier

The `Classifier` base class provides the basic process for evaluating an algorithm, given a training dataset and a testing dataset. As a part of this process, it defines methods that all classification algorithms must implement.

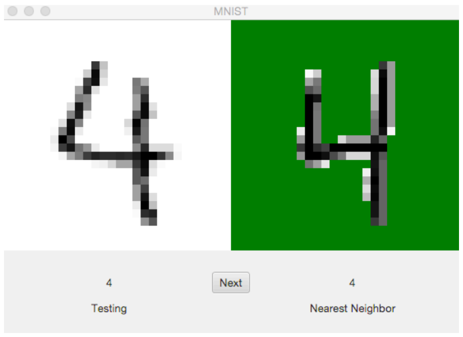
1.3.3 ZeroRClassifier

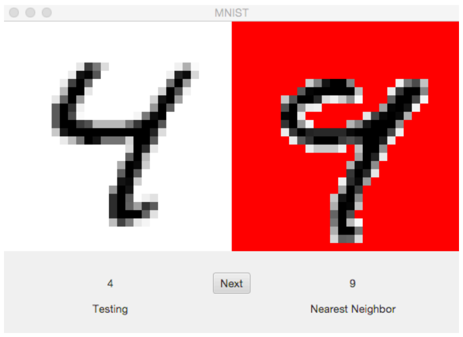
The first algorithm you will implement, ZeroR[5], is sometimes seen as a sanity check: it determines which example label is most common in the training set, and responds to that for all testing examples.

1.3.4 NearestNeighborClassifier

The second algorithm you will implement, NearestNeighbor[6] (NN), is another useful baseline: it finds the closest training example to each test example, according to a *distance function*, and uses that's match's result as the prediction.

For example, consider the MNIST dataset[7], which is composed of 60,000 examples, each with 780 features (each a value from 0-255, representing a greyscale pixel value in a 28x28 picture) and the result is the handwritten number that is being represented in the picture (0-9). The images below illustrate how this algorithm, when presented a new example, finds the most similar previously seen digit and guesses that they are the same.





1.3.5 DistanceFunction

As a part of NN, you will need to implement the `DistanceFunction` interface, which will provide a flexible abstraction for indicating the "distance" between two testing examples (i.e. via the features only).

1.3.6 EuclideanDistanceSquared

While there are several commonly used distance functions, the most common baseline is Euclidean (or L2). (We will use this, but not take the square root of the final computation, just to be more efficient.)

1.4 GUI

The final aspect of the project is developing a graphical user interface to allow the user to choose a dataset and an algorithm, click run, and have the accuracy provided as a result (see the example screenshot on the first page).

1.4.1 ClassificationGUI

The `ClassificationGUI` class will implement the GUI.

[5]See `http://www.saedsayad.com/zeror.htm`
[6]See `http://www.saedsayad.com/k_nearest_neighbors.htm`
[7]See `http://www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets/multiclass.html#mnist`

2 Opportunities

Listed above is the skeleton of the final project, most of which will be covered in varying degrees in the coming assignments. However, if you wish to further your understanding of OOP and/or ML, here are some possible further steps you can take in your implementation:

- Parse other file format(s)
- Implement other ML algorithm(s)
- Extend the GUI to allow the user to parameterize the algorithm/dataset (e.g. RNG seed, file location)
- Extend the GUI to provide status feedback as to the classification progress
- Implement cross-validation of data[8] and/or additional evaluation metrics (e.g. training/testing time, confusion matrix)

If you propose one or more extension(s), we agree ahead of time, and you successfully implement, you will receive extra credit.

[8]See `http://en.wikipedia.org/wiki/Cross-validation_(statistics)`