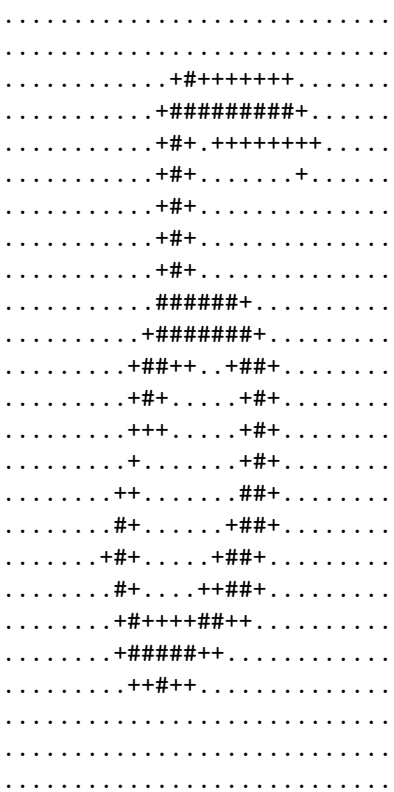
Project 4
Machine Learning: Optical Character Recognition

Due: Monday, 8/17/09

The goal of this project is to become familiar with a simple Machine Learning system. Statistics and machine learning are becoming increasingly important in computer science and are widely used for applications such as spam filtering, robot movement learning, computer vision, natural language processing, etc. In this project, we explore a part of computer vision: Optical Character Recognition through Naive Bayes algorithm.

OCR is the problem of analyzing a scanned image of a hand-written number, something like:



and be able to automatically decide which number it actually is.

You will need the following files:

```
~cs61a/lib/ocr/classify.scm
~cs61a/lib/ocr/samples.scm
~cs61a/lib/ocr/trainingimages
~cs61a/lib/ocr/traininglabels
~cs61a/lib/ocr/validationimages
~cs61a/lib/ocr/validationlabels
```

You will only be working with `classify.scm` The other files are for support purposes only. Although the purpose of this project is to perform OCR, the Naive Bayes Algorithm of the code is written to be very general and you will see that this program can be very easily adapted for other classification problems.

The General Abstract Framework:

We would like to assign **labels** to **instances** based on **features of the instances**. We will learn how distinguishing the features are from **training set**, which contains **labeled instances**.

For example: in OCR, the instances are the images in the form of a vector, the labels are the numbers `0,1,2,3,4,5,6,7,8,9`, the features will be every 2 by 2 pixel squares and whether they contain hand-writing or not.

Structure of Program:

The program contains two main procedures:

```
(train  in-stream  feat-getter  label-getter  max-samples)
```

`train` takes a training set of general instances in the form of a stream, a procedure `feat-getter` which takes an instance and returns a list of features, a procedure `label-getter` which takes an instance and returns the correct label of the instance, and a limit to how many samples to train.

```
(infer-label  instance  feat-getter   all-feats)
```

infer-label takes an instance, a procedure `feat-getter` which takes an instance and returns a list of features, and `all-feats` which is a lst of all possible features.

Thus, by having `feat-getter` and `label-getter` as input, we make our classification program very general. To classify new instances, the user just needs to provide a `feat-getter` procedure and a `label-getter` procedure and then call `train` and `infer-label`.

To actually work with images, we represent each raw scanned image by an Abstract Data Type called `image` (which actually uses vectors). The selectors are:

```
(get-img-label img)

(is-img? img)

(get-xy x y img)
```

Where `get-img-label` returns the correct label of the image, `get-xy` takes a coordinate and returns 0 if the coordinate has no handwriting, 1 if the coordinate has light handwriting, and 2 if the coordinate has heavy handwriting.

We can run the program by `(load ‘‘classify.scm’’)` but you will get an error until you finish exercise 1 and 2.

Exercise 1: Gathering Statistics

NOTE: The actual code you have to fill in is VERY SHORT (as in a couple of lines) for all exercises. Most of your time should be spent understanding the procedures provided.
NOTE 2: Throughout the entire project, you should only modify `train` procedure, `infer-label` procedure, and code in the Cat vs. Bear mini-test section. YOU DO NOT NEED to modify any other part of the code.

The `train` procedure is almost fully defined. Your job here is to complete the definition of `train`.

Recall that `train` gathers 2 types of statistics:

1. For every label, we gather P(label), percentage of instance with the given label
2. Given a label, for every feature, we gather P(feat|label), percentage of instances OF THE GIVEN LABEL that has the given feature

The P(label) count we store inside `label-counter` ADT, which just associates a label with a count. The P(feat|label) count we store inside `label/feat-counter` ADT, which associates every label with a Feature-Counter. Each Feature-Counter associates a Feature with a count.

For example, suppose we are trying to classify whether an animal is a Cat or a Bear and we have the following labeled-instances in our Training Set:

1. Label: Cat; Features: Furry, Medium, Brown
2. Label: Cat; Features: Furry, Small, Black
3. Label: Bear; Features: Furry, Big, Black
4. Label: Bear; Features: Furry, Medium, Black
5. Label: Bear; Features: Furry, Medium, Brown

Then we will tally the following statistics:

1. Label-Counter: (Cat⇒2, Bear⇒3)
2. Label/Feat-Counter:
   (Cat ⇒ (Furry ⇒ 2,Medium⇒1,Brown⇒1, Small⇒1, Black⇒1),
   Bear⇒(Furry⇒3, Medium⇒2, Big⇒1, Black⇒2, Brown⇒1))

Notice that Label/Feat-Counter associates each Label with a Feature-Counter. Each Feature-Counter associates Features with a count. Here, Furry is associated with count of 2 for Cat because 2 of the Cats have the feature Furry.

Now, we normalize, meaning we want to turn raw Counts into percentages.

1. Label-Counter: (Cat⇒0.4, Bear⇒0.6)
2. Label/Feat-Counter:
   (Cat ⇒ (Furry ⇒ 1, Medium⇒0.5, Brown⇒0.5, Small⇒0.5, Black ⇒ 0.5) Bear ⇒ (Furry ⇒ 1, Medium⇒0.666, Big⇒0.333, Black⇒0.666, Brown⇒0.333))

Here, in Label-Counter, Cat⇒0.4 because two-fifth of all Training instances are Cats. Note that in Label/Feat-Counter, we normalize each of the feature-Counters separately. For example: in Label/Feat-Counter, for Cats, Black ⇒ 0.5 (i.e. P(Black | Cat) = 0.5), because half of **ALL CATS** are black. For bears, Black ⇒ 0.666 (i.e. P(Black | Bear) = 0.666) because two-third of **ALL BEARS** are black.

The part of `train` you have to fill out is to get the raw counts. The normalization is done for you. You should pay attention to the two data structures: `Label-Counter` and `Label/Feat-Counter` provided for you.

Exercise 2: Bayes Rule Implementation

Here, we need to fill in `infer-label`, that is, to be able to guess a label for a new instance. Suppose a new instance has features $F_1, F_2, ...F_n$. We will then, for every possible label, compute P(label$|F_1, F_2, ...F_n$)

We will then return the label that maximizes the above probability. (In program, we actually return a pair of both the label and the probability P(label$|F_1, F_2, ...F_n$) associated with the label)

By Bayes Rule, the probability of each label is:

$$P(\text{label}|F_1, F_2, ...F_n) = \frac{\prod_{i=1}^{n} P(F_i|\text{label})}{Z}$$

Where the Z term on the bottom is the same for all labels and

$$Z = \sum_{j=1}^{m} \left(\prod_{i=1}^{n} P(F_i|\text{label}_j) \right)$$

If the new instance does not have feature F_1, F_2, then when we compute the $\prod_{i=1}^{n} P(F_i|\text{label})$ value, we use the terms $P(NF_1|\text{label})$ and $P(NF_2|\text{label})$ terms for the product instead of $P(F_1|\text{label})$ or $P(NF_2|\text{label})$. Remember that $P(NF_1|\text{label}) = 1 - P(F_1|\text{label})$.
(**Note: This is not as complicated as it seems. Jump to the example below if you find this all this way over your head**)

For our `infer-label` procedure, we first compute the $\prod_{i=1}^{n} P(F_i|\text{label})$ value for all labels and put all these values in a list called `label-prob-pairs`. And then we sum together all the $\prod_{i=1}^{n} P(F_i|\text{label})$ to find Z and then divide every value in `label-prob-pairs` by Z. Finally, we take the label with the greatest probability.

For example, supposing that we have the Cat/Bear classification example as above.

Suppose we get a new instance that looks like:
Features: Furry, Medium, Brown.

Is this a Cat or a Bear? To figure that out, we will first assume we have a cat and compute:

$P(\text{Cat})P(\text{Furry}|\text{Cat})P(\text{Medium}|\text{Cat})P(\text{Brown}|\text{Cat})P(\text{NOTBig}|\text{Cat})P(\text{NOTSmall}|\text{Cat})P(\text{NOTBlack}|\text{Cat})$

We will just read the Label/Feat-Counter we build above to gather the numbers:

$$0.4 * 1 * 0.5 * 0.5 * (1 - 0) * (1 - 0.5) * (1 - 0.5) = 0.025$$

You should note two things: ONE. To find P(NOTBlack|Cat), we used (1 - P(Black|Cat)) and TWO. Since none of the cats are Big, P(NOTBig|Cat) is just 1.

We will now assume that we have a bear and compute:

$P(\text{Bear})P(\text{Furry}|\text{Bear})P(\text{Medium}|\text{Bear})P(\text{Brown}|\text{Bear})P(\text{NOTBig}|\text{Bear})P(\text{NOTSmall}|\text{Bear})P(\text{NOTBlack}|\text{Bear})$

and get

$$0.6 * 1 * 0.666 * 0.333 * (1 - 0.333) * (1 - 0) * (1 - 0.666) = 0.0295$$

What we just did is to compute the $\prod_{i=1}^{n} P(F_i|\text{label})$ values for each label.

Now, by Bayes Rule,

$$P(\text{Cat}|\text{Furry, Medium, Brown, NOTBig, NOTSmall, NOTBlack}) = \frac{0.025}{0.025 + 0.0295}$$

and

$$P(\text{Bear}|\text{Furry, Medium, Brown, NOTBig, NOTSmall, NOTBlack}) = \frac{0.0295}{0.025 + 0.0295}$$

Note that $0.025 + 0.0295$ is our Z value.

Since P(Bear|Furry, Medium, Brown, NOTBig, NOTSmall, NOTBlack) is larger, we would then classify this new instance as a BEAR.

Your job is to fill in part of the code that computes $\prod_{i=1}^{n} P(F_i|\text{label})$ value for every labels.

Exercise 3: Adding New Feature to Mini-test

After finishing exercise 1 and 2, you can now try running `classify.scm` by
tt (load "classify.scm") and it should begin the mini-tests. Make sure that the results to the mini-tests are correct; debug exercise 1 and 2 if not.

Now, the 5th instance that we classify in the mini-test should be a bear even though we classify it as a cat. So, add a new feature `'roars` and `'meows` to all of our instances. All cats and only cats should have the feature `'meows` and all bears and only bears should have the feature `'roars`. Be sure to look at `catbear-feat-getter`, `all-catbear-feat-list`, and don't forget to add these new features to the test-instances also.

After adding the new features, our classifier should correct classify the 5th test instance as a bear.

Running and Submitting OCR

After you complete all 3 exercises, you are now ready to run the OCR. Simply uncomment the last 2 lines in ``` ``classify.scm'' ``` and reload the file.

You should turn in a file called `XY.classify.scm` where you should replace `XY` with your actual login.