Project 4
Machine Learning: Optical Character Recognition

Due: Noon Friday, 8/13/10

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, stock portfolio optimization, etc. In this project, we explore a part of computer vision: Optical Character Recognition through Naive Bayes algorithm.

Optical Character Recognition (OCR) is the problem of analyzing a scanned image of a hand-written number, something like (ascii representation):

```
............................
............................
............+#+++++++.......
...........+#########+......
...........+#+.++++++++.....
...........+#+.......+......
...........+#+..............
...........+#+..............
...........+#+..............
...........######+..........
..........+########+........
.........+##++..+##+........
.........+#+.....+#+........
.........+++.....+#+........
.........+.......+#+........
........++.......##+........
........#+......+##+........
.......+#+.....+##+.........
........#+....++##+.........
........+#++++##++..........
........+######++............
.........++#++..............
............................
............................
............................
```

and be able to automatically decide which number it actually is. Machine learning is not the only way to do OCR, but it is currently the best way.

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 Machine Learning Framework:

Note: this part only describes a sub-field of Machine Learning called Classification. Machine Learning is a much broader field.

We would like to assign **labels** to **instances** based on **features of the instances**.

- Example 1: in spam filtering, the instances are the actual emails and the labels can either be spam or not-spam. We can choose the features to be various characteristics of the email, i.e. whether the email contains the word "viagra", whether the email comes from an unknown sender, etc.
- Example 2: 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`, and we will choose the features to be every 2 by 2 pixel region of the image and whether they contain hand-writing or not.

In Machine Learning, we take **Training Instances** (instances with features and labels that we know are correct) and gather data from these Training Instances to help us assign the correct label to **Testing Instances** (instances with only features). For our algorithm, the data we gather are various probabilities; we will describe these probabilities in detail later.

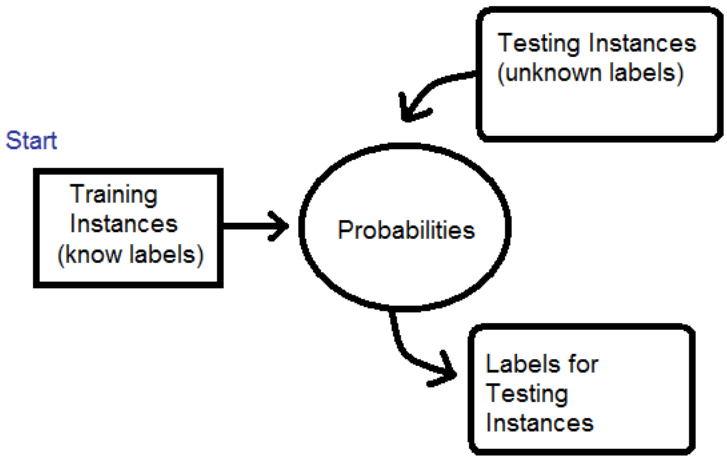

Figure 1: The general Machine Learning (classification) framework

Structure of Program:

Main Procedures

The program contains two main procedures: `train` and `infer-label`

```
(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.

`train` itself returns nothing. It instead stores probabilities in the `label-counter` and `label/feat-counter` data structure.

```
(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.

`infer-label` returns a label.

Important Note: 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`.

Data Structures to Store Probabilities

We define two Abstract Data Types to store all necessary probabilities: `label-counter` and `label/feat-counter`. See description of exercise 1 for a concrete example.

- `label-counter` associates a number with every label
  - `(make-label-counter)` returns a new empty `label-counter`
  - `(add-to-label-counter label label-counter)` increments by one the number associated with the input label
  - `(get-label-count label label-counter)` returns the number associated with the input label
  - `(normalize-label-counter label-counter)` divide every number by the sum of all numbers to turn the counts into probabilities
- `label/feat-counter` associates every label with an association list, which associates every feature with a number
  - `(make-label/feat)` returns a new empty `label/feat-counter`
  - `(get-label/feat label feat label/feat)` returns the number associated with the input feat in the association list associated with the input label

- `(add-to-label/feat label feat-list label/feat)` increments by one the number associated with every feature in `feat-list` in the association list associated with the input label
- `(normalize-label/feat label-counter label/feat)` for every label, normalizes every association list associated with that label

Interfacing with Mini-Tests

Training on actual images will take a long time (around 15 minutes). Thus, the code provides mini-test cases for you to debug your code.

Interfacing with Images

Note: You actually do not need to understand this part to do the project. This is just for your curiosity.

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.

Running the program

We can run the program by uncommenting the last two lines of the program and then using `(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 in this exercise 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:
   (a) Cat ⇒ (Furry ⇒ 1, Medium⇒0.5, Brown⇒0.5, Small⇒0.5, Black ⇒ 0.5)
   (b) 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

Note: P(A | B) represents probability of A GIVEN B and is different from P(A), P(B), and P(B | A). For example, the probability you have the swine flu is low because swine flu is rare. The probability

you have the swine flu GIVEN that you are running a fever is higher but still pretty small because so many other more common diseases cause fever. The probability you are running a fever GIVEN you have the swine flu is very very high because fever is a symptom of swine flu. You need not understand the math precisely; you just need to have an intuition on how these different probabilities are different.

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

$$Score_{\text{label}} = P(\text{label}) \prod_{i=1}^{n} P(F_i|\text{label})$$

We will then return the label that maximizes the above score. (In program, we actually return a pair of both the label and the score associated with the label)

We can interprete the score as a un-normalized probability. By Bayes Rule, the probability of each label is:

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

Where the Z term on the bottom is the normalization term to make sure the probability sum to 1:

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

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 $P(\text{label}) \prod_{i=1}^{n} P(F_i|\text{label})$ value for all labels and put all these values in a list called `label-prob-pairs`. And then we sum together all the $P(\text{label}) \prod_{i=1}^{n} P(F_i|\text{label})$ 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 $P(\text{label}) \prod_{i=1}^{n} P(F_i|\text{label})$ 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 $P(\text{label}) \prod_{i=1}^{n} P(F_i|\text{label})$ score value for every labels. See `infer-label` procedure in the code.

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.

Note: In our framework, it is possible for the same cat instance to have both "meows" and "roars" or both "big" and "small" features. Such non-sensical training instances will not signal any error but it could very well throw off your probabilities. This shortcoming shows a fundamental weakness of Machine Learning approach: the quality of your classifications depends heavily on the quality of your training data. You don't need to correct this shortcoming.

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.