1 Assignment

1.1 Intro

Handwriting recognition is the ability of a computer to interpret hand written text as the characters. In this assignment we will be trying to recognize numbers from images. To accomplish this task, we will be using a Naive Bayes classifier.

Naive Bayes classifiers are a family of probabilistic classifiers that are based on Bayes' theorem. These algorithms work by combining the probabilities that an instance belongs to a class based on the value of a set of features. In this case we will be testing if images belong to the class of the digits 0 to 9 based on the state of the pixels in the images.

1.2 Provided Files

Fork the starting git repository using the URL below. This time we are providing you `catch.hpp` and an initial Cmake file to help you get started. You are not required to sue them but we thought they might be helpful. We have also provided a directory to store the data files you will need to download separately.

`https://classroom.github.com/a/sjp09mv-`

We are also providing data files representing handwriting images along with the correct classification of these images. These are provided as a zip file that you can find at the following URL.

`https://subversion.ews.illinois.edu/svn/sp18-cs126/_shared/digitdata.zip`

This zip file contains the following.

- `readme.txt` - description of the files in the zip
- `testimages` - 1000 images to test
- `testlabels` - the correct class for each test image
- `trainingimages` - 5000 images for training
- `traininglabels` - the correct class for each training image

We will be using the training images to train our classifier specifically to compute probabilities that we will use and then try to recognize the test images. You should use the .gitignore file to prevent these files from being added to your git repository.

1.3 Image Data Description

The image data for this consist of 28×28 pixel images stored as text. Each image has 28 lines of text with each line containing 28 characters. The values of the pixels are encoded as `' '` for white and `'+'` for gray and finally `'#'` for black. You can view the images by simply looking at them with a fixed width font in a text editor. We will be classifying images of the digits 0-9, meaning that given a picture of a number we should be able to accurately label which number it is.

Since computers can't tell what a picture is we're going to have to describe the input data to it in a

way that it can understand. We do this by taking an input image and extracting a set of features from it. A feature simply describes something about our data. In the case of our images we can simply break the image up into the grid formed by its pixels and say each pixel has a feature that describes its value.

We will be using these images as a set of binary features to be used by our classifier. Since the image is 28×28 pixels we will have $28 \times 28 = 784$ features for each input image. To produce binary features from this data we will treat pixel i, j as a feature and say that $F_{I,j}$ has a value of 0 if it is a background pixel or white and a value of 1 if it is a foreground pixel and not white. In this case we are generalizing the two foreground values of gray and black as the same to simplify our task.

2 Naive Bayes Algorithm

2.1 Overview

Naive Bayes classifiers are based on Bayes' theorem which describes the probability of an event based on prior knowledge of conditions that might be related to the event. In our case we are computing the probability that an image might belong to a class based on the value of a feature. This idea is called conditional probability and is the likelihood of event A occurring given event B and is written as $P(A|B)$. From this we get Bayes' theorem.

$$P(A|B) = \frac{P(B|A)P(A)}{P(B)}$$

This says that if A and B are events and $P(B) \neq 0$ then

- $P(A|B)$ is the probability that A occurs given B
- $P(B|A)$ is the probability that B occurs given A
- $P(A)$ the probability A occurs independently of B
- $P(B)$ the probability B occurs independently of A

Applying this to our case we want to compute for each feature the probability that given the feature $F_{i,j}$ has a value $f \in \{0, 1\}$ that the image will belong to a given class $c \in \{0, 1, ..., 9\}$. To do this we will train our model on a large set of images where we know the class that they belong to. Once we have computed these probabilities we will be able to classify images by computing the class with the highest probability given all the known features of the image and assign it to that class.

From this we get two phases of the algorithm.

- Training - compute the conditional probability that for each feature that an image is part of each class.
- Classifying - for each image compute the class that it is most likely to be a member of given the assumed independent probabilities computed in the training stage.

2.2 Training

The goal of the training stage is to teach the computer the likelihoods that a feature $F_{i,j}$ has value $f \in \{0, 1\}$ as we have binary features, given that the current image is of class $c \in \{0, 1, ..., 9\}$, which we can write as $P(F_{i,j} = f | class = c)$

This can be simply computed as follows:

$$P(F_{i,j} = f | class = c) = \frac{\#\text{ of times } F_{i,j} = f \text{ when } class = c}{\text{Total number of training examples where } class = c}$$

Given how we will use these probabilities we need to ensure that they are not zero since if they are it will cause them to cancel out any non-zero probability from other features. To do this we will use a technique called Laplace Smoothing. This technique works by adding a small positive value k to the numerator above, and $k \cdot V$ to the denominator (where V is the number of possible values our feature can take on so 2 in our case). The higher the value of k, the stronger the smoothing is. So for our binary features that would give us:

$$P(F_{i,j} = f | class = c) = \frac{k + \#\text{ of times } F_{i,j} = f \text{ when } class = c}{2k + \text{ Total number of training examples where } class = c}$$

You can experiment with different values of k (say, from 0.1 to 10) and find the one that gives the highest classification accuracy.

You should also estimate the priors $P(class = c)$ or the probability of each class independent of features by the empirical frequencies of different classes in the training set.

$$P(class = c) = \frac{\#\text{ of training examples where } class = c}{\#\text{ of training examples}}$$

With the training done you will have an empirically generated set of probabilities that can be referred to as the model that we will use to do classification on unknown data. This model once generated can be used in the future without needing to be regenerated.

2.3 Classifying

To classify the unknown images using the trained model you will perform maximum a posteriori (MAP) classification of the test data using the trained model. This technique computes the posterior probability of each class for the particular image. Then classifies the image as belonging to the class which has the highest posterior probability.

Since we assume that all the probabilities are all independent of each other we can compute the probability that the image belongs to the class by simply multiplying all the probabilities together and dividing by the probability of the feature set. Finally, since we don't actually need the probability but merely to be able to compare the values we can ignore the probability of the feature set and thus compute the value as follows:

$$P(class) * P(f_{1,1}|class) * P(f_{1,2}|class) * ... * P(f_{28,28}|class)$$

Unfortunately, when computers do floating point arithmetic they are not quite computing as we do by hand and they can produce a type of error called underflow. More information on this can be found here:

`https://en.wikipedia.org/wiki/Arithmetic_underflow`

To avoid this problem, we will compute using the log of each probability rather than the actual value. This way the actual computation will be the following:

$$\log(P(class)) + \log(P(f_{1,1}|class)) + \log(P(f_{1,2}|class)) + ... + \log(P(f_{28,28}|class))$$

Where $f_{i,j}$ is the feature value of the input data. You should be able to use the priors and probabilities you calculated in the training stage to calculate these posterior probabilities.

Once you have a posterior probability for each class from 0-9 you should assign the test input to the class with the highest posterior probability. For example for an input P, if we had the posterior probabilities:

class	posterior probability
0	.3141
1	.432
2	0
3	0
4	.4
5	.004
6	.1
7	.2
8	.7
9	.5

We'd classify P as an 8 as that is the class with the highest posterior probability.

3 Requirements

To complete this assignment you will need to use C++ to write a program the uses the Naive Bayes Algorithm described above to classify the provided images. This program should run on the command line and must have some interface to implement at least the following features.

- Read training data from files and generate a model
- Save a model as a file
- Load a model from a file
- Classify images from a file

The interface you design to allow the user to accomplish these tasks is up to you but is expected to run correctly from the command line.

Remember to plan your assignment out before writing it and breaking it apart into the appropriate necessary functions and book keeping values that you will need to keep.

Finally you will also need to evaluate your classifier.

3.1 Evaluation

Use the true class labels of the test images from the `testlabels` file to check the correctness of your model. Report your confusion matrix. This is a 10×10 matrix whose entry in row r and column c is the percentage of test images from class r that are classified as class c.

In addition, for each digit class, show the test examples from that class that have the highest and the lowest posterior probabilities for its actual class according to your classifier. You can think of these as the most and least "prototypical" instances of each digit class (and the least "prototypical" one is probably misclassified).

Play around with the Laplace smoothing factor to figure out how to get the best performance out of your classifier.

Try changing the order you feed in your training data, randomizing it to see if it changes the results of your classifier.

3.2 ~~Extra Credit~~

- ~~Perform K-cross form validation with different values of K to improve classification accuracy~~
- ~~Use ternary features (by taking into account the two foreground values to see if classification accuracy improves~~
- ~~Use groups of pixels as features, perhaps 2×2 pixel squares where the value of a feature is the majority value of its composing pixels~~

3.3 Glossary

We're aware that you were presented with many terms and mathematical concepts so we've assembled a glossary here for you to refer to.

Probability $P(X)$ is the probability that some event X happens and is a value between 0 and 1.

Conditional probability $P(A|B)$ is the probability that some event A happens given that B has already happened.

$$P(A|B) = \frac{P(A \cap B)}{P(B)}$$

Bayes Theorem Predicts probability of an event based on knowledge we already have

$$P(C|X) = \frac{P(C) * P(X|C)}{P(X)}$$

Feature We use features to describe a characteristic of the input data. A feature set is the set of all features that describe a piece of input data. In our case we have 1 feature for each pixel in a 28×28 image so we have 784 features $F_{i,j}$

Class Class describes the label of a piece of data. In our case our classes can take any value from 0-9

Laplace Smoothing Add some k to the numerator in $P(F_{i,j} = f|class = c)$ and $k \cdot V$ where V is the number of values the features can take to the denominator. This ensures there are no zero counts and the posterior probability is not zero

Maximum A Posteriori MAP classification is used to classify what the label of a test input. We calculate the posterior probabilities of all the classes and assign the class with highest posterior probability to our input.

Confusion Matrix A 10×10 matrix where entry $(R \times C)$ tells you what percentage of test images from class R where classified as class C

Prototypical Examples The least and most prototypical examples should tell you which test inputs are the most and least ideal inputs for a given class.