CSC 412 Fall 2019 Programming Assignment 2

Naive Bayes Classification

Due time: November 8, 2019 11:59 PM

Checkpoint	Task	Deadline
1	Project groups formation (*via Canvas*) due date	October 16
2	Report and submission	November, 8

Grade: 100 marks in total, while it accounts for **20%** towards the final grade.

Type: 2 students group work, or independent work if you prefer.

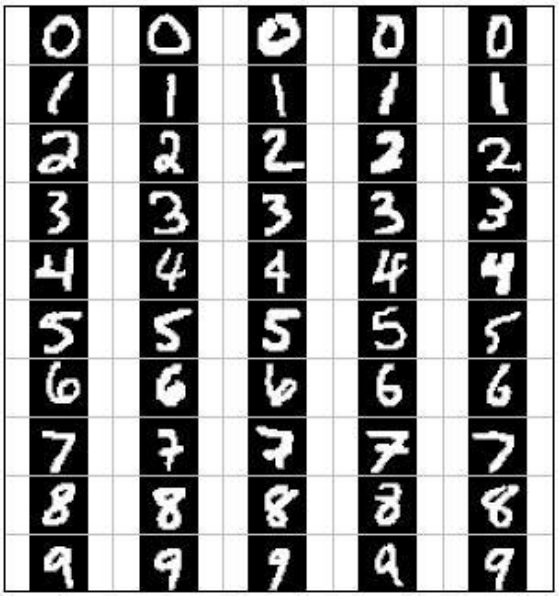
Programming language: Python (preferred), Java, C/C++, or any other advanced programming languages.

Hints: Textbook online repository files (probability.py, probability.ipynb; probability4e.py, probability-4e.ipynb).

Project Overview: The goal of this assignment is to implement Naive Bayes classifiers and to apply it to the task of classifying visual patterns. As before, you can work in teams of up to two people.

Contents

Digit classification



(Adapted from Berkeley CS 188 project 5)

Data: The attached file "digitdata.zip" is a zip archive containing training and test digits, together with their ground truth labels (see `readme.txt` in the zip archive for an explanation of the data format). There are `5000` training exemplars (roughly `500` per class) and `1000` test exemplars (roughly `100` per class).

Part 1: Single pixels as features

- **Features:** The basic feature set consists of a single binary indicator feature for each pixel. Specifically, the feature F_{ij} indicates the status of the `(i,j)-th` pixel. Its value is `1` if the pixel is foreground (no need to distinguish between the two different foreground values), and `0` if it is background. The images are of size `28*28`, so there are `784` features in total.
- **Training:** The goal of the training stage is to estimate the **likelihoods** $P(F_{ij} \mid class)$ for every pixel location `(i,j)` and for every digit class from `0` to `9`. The likelihood estimate is defined as

  $$P(F_{ij} = f \mid class) = (\#\text{ of times pixel (i,j) has value f in training examples from this class}) / (\text{Total \# of training examples from this class})$$

  In addition, as discussed in the lecture, you have to **smooth** the likelihoods to ensure that there are no zero counts. *Laplace smoothing* is a very simple method that increases the observation count of every value `f` by some constant `k`. This corresponds to adding `k` to the numerator above, and `k*V` to the denominator (where V is the number of possible values the feature can take on). The higher the value of k, the stronger the smoothing. 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** $P(class)$ by the empirical frequencies of different classes in the training set.
- **Testing:** You will perform **maximum a posteriori (MAP)** classification of test digits according to the learned Naive Bayes model. Suppose a test image has feature values $f_{1,1}, f_{1,2}, \ldots, f_{28,28}$. According to this model, the posterior probability (up to scale) of each class given the digit is given by

  $$P(class) \cdot P(f_{1,1} \mid class) \cdot P(f_{1,2} \mid class) \cdot \ldots \cdot P(f_{28,28} \mid class)$$

  Note that in order to avoid underflow, it is standard to work with the log of the above quantity:

  $$\log P(class) + \log P(f_{1,1} \mid class) + \log P(f_{1,2} \mid class) + \ldots + \log P(f_{28,28} \mid class)$$

  After you compute the above decision function values for all ten classes for every test image, you will use them for MAP classification.

  Evaluation: Use the true class labels of the test images from the `testlabels` file to check the correctness of the estimated label for each test digit. Report your performance in terms of the

classification rate for each digit (percentage of all test images of a given digit correctly classified). Also report your **confusion matrix**. This is a 10x10 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 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).

Important: The ground truth labels of test images should be used *only* to evaluate classification accuracy. They should not be used in any way during the decision process.

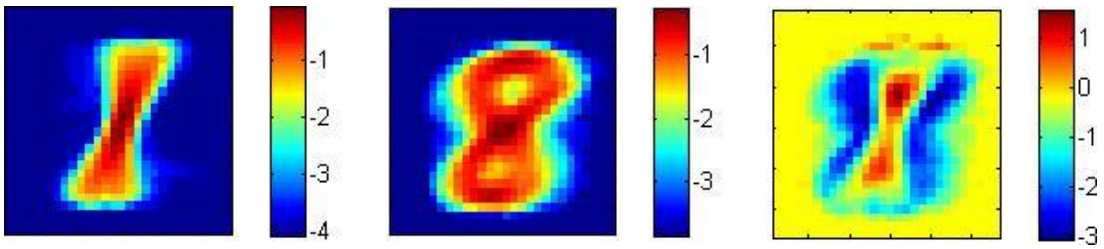
Tip: You should be able to achieve at least 70% accuracy on the test set. One "warning sign" that you have a bug in your implementation is if some digit gets 100% or 0% classification accuracy (that is, your system either labels all the test images as the same class, or never wants to label any test images as some particular class).

- **Odds ratios:** When using classifiers in real domains, it is important to be able to inspect what they have learned. One way to inspect a naive Bayes model is to look at the most likely features for a given label. Another tool for understanding the parameters is to look at odds ratios. For each pixel feature F_{ij} and pair of classes c_1, c_2, the odds ratio is defined as

  $$odds(F_{ij}=1, c_1, c_2) = P(F_{ij}=1 \mid c_1) / P(F_{ij}=1 \mid c_2).$$

  This ratio will be greater than one for features which cause belief in c_1 to increase over the belief in c_2. The features that have the greatest impact on classification are those with both a high probability (because they appear often in the data) and a high odds ratio (because they strongly bias one label versus another).

  Take four pairs of digits that have the highest confusion rates according to your confusion matrix, and for each pair, display the maps of feature likelihoods for both classes as well as the odds ratio for the two classes. For example, the figure below shows the log likelihood maps for 1 (left), 8 (center), and the log odds ratio for 1 over 8 (right):

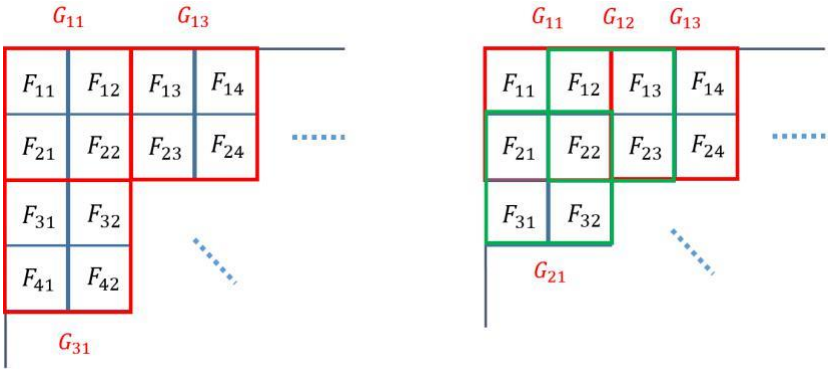


  If you cannot do a graphical display like the one above, you can display the maps in ASCII format using some coding scheme of your choice. For example, for the odds ratio map, you can use '+' to denote features with positive log odds, ' ' for features with log odds close to 1, and '-' for features with negative log odds.

Part 2: Pixel groups as features

Instead of each feature corresponding to a single pixel, we can form features from groups of adjacent pixels. We can view this as a relaxation of the Naive Bayes assumption that allows us to have a more accurate model of the dependencies between the individual random variables. Specifically, consider a 2*2 square of pixels with top left coordinate i,j and define a feature $G_{i,j}$ that corresponds to the ordered tuple of the four pixel values. For example, in the figure below, we have

$$G_{1,1} = (F_{1,1}, F_{1,2}, F_{2,1}, F_{2,2}).$$



(The exact ordering of the four pixel values is not important as long as it's consistent throughout your implementation.) Clearly, this feature can have 16 discrete values. The 2*2 squares can be disjoint (left side of figure) or overlapping (right side of figure). In the case of disjoint squares, there are 14 * 14 = 196 features; in the case of overlapping squares, there are 27 * 27 = 729 features.

We can generalize the above examples of 2*2 features to define features corresponding to n*m disjoint or overlapping pixel patches. An n*m feature will have 2^{n*m} distinct values, and as many entries in the conditional probability table for each class. Laplace smoothing applies to these features analogously as to the single pixel features.

In this part, you should build Naive Bayes classifiers for feature sets of n*m disjoint/overlapping pixel patches and report the following:

- Test set accuracies for disjoint patches of size 2*2, 2*4, 4*2, 4*4.
- Test set accuracies for overlapping patches of size 2*2, 2*4, 4*2, 4*4, 2*3, 3*2, 3*3.
- Discussion of the trends you have observed for the different feature sets (including single pixels), in particular, why certain features work better than others for this task.
- Brief discussion of running time for training and testing for the different feature sets (which ones are faster and why, and how does the running time scale with feature set size).

Tip: You should be able to achieve over 80% accuracy with your best feature set.

Extra Credit

- Experiment with yet more features to improve the accuracy of the Naive Bayes model. For example, instead of using binary pixel values, implement ternary features. Alternatively, try using features that check to see if there is a horizontal line of three consecutive foreground pixels, or a vertical or diagonal line.

- Apply your Naive Bayes classifier with various features to the attached face data facedata.zip. It is in a similar format to that of the digit data, and contains training and test images and binary labels, where 0 corresponds to 'non-face' and 1 corresponds to 'face'. The images themselves are higher-resolution than the digit images, and each pixel value is either '#', corresponding to an edge being found at that location, or ' ', corresponding to a non-edge pixel.

Report Checklist

Part 1.

- Briefly discuss your implementation, especially the choice of the smoothing constant.
- Report classification rate for each digit and confusion matrix.
- For each digit, show the test examples from that class that have the highest and lowest posterior probabilities according to your classifier.
- Take four pairs of digits that have the highest confusion rates, and for each pair, display feature likelihoods and odds ratio.

Part 2.

- Report test set accuracies for disjoint patches of size 2*2, 2*4, 4*2, 4*4, and for overlapping patches of size 2*2, 2*4, 4*2, 4*4, 2*3, 3*2, 3*3. Discuss trends for
- the different feature sets.
- Discuss training and testing running time for different feature sets.

Submission Instructions

As before, **one designated person from the group** will need to submit on Canvas by the deadline. Each submission must consist of the following two attachments:

1. A **report** in **PDF format**. As before, the report should briefly describe your implemented solution and fully answer all the questions posed above. **Remember: you will not get credit for any solutions you have obtained, but not included in the report.**

   All group reports need to include a brief **statement of individual contribution**, i.e., which group member was responsible for which parts of the solution and submitted material.

The name of the report file should be **lastname_firstname_assignment2.pdf**. Don't forget to include the names of all group members and the number of credit credits at the top of the report.

2. Your **source code** compressed to a **single ZIP file**. The code should be well commented, and it should be easy to see the correspondence between what's in the code and what's in the report. You don't need to include executables or various supporting files (e.g., utility libraries) whose content is irrelevant to the assignment. If we find it necessary to run your code in order to evaluate your solution, we will get in touch with you.

   The name of the code archive should be **lastname_firstname_assignment2.zip**.

Multiple attempts will be allowed but only the last submission will be graded. **We reserve the right to take off points for not following directions.**

Late policy: For every day that your assignment is late, your score gets multiplied by 0.75. The penalty gets saturated after four days, that is, you can still get up to about 32% of the original points by turning in the assignment at all. If you have a compelling reason for not being able to submit the assignment on time and would like to make a special arrangement, you must send me email **at least four days before the due date** (any genuine emergency situations will be handled on an individual basis).

Be sure to also refer to course policies on academic integrity, etc.