Pattern recognition systems – Lab 9

Naive Bayesian Classifier: Digit Recognition Application

1. Objectives

In this lab session we will study the Naive Bayes algorithm and we will apply it to a specific recognition problem: we will learn to distinguish between images of handwritten digits.

2. Theoretical Background

The Naive Bayes classifier applies the Bayes rule and an independence assumption to calculate the posterior probability of each class. The class with the highest probability is chosen as the output.

Due to the independence assumption it can handle an arbitrary number of independent variables whether continuous or categorical. Given a set of random variables, $\boldsymbol{x} = \{x_1, x_2, \dots, x_d\}$, we want to construct the posterior probability for the random variable *C* having the set of possible outcomes $C = \{c_1, c_2, \dots, c_J\}$. In a more familiar language, the elements $\boldsymbol{x}$ are the predictors or features and $\boldsymbol{C}$ is the set of categorical levels or classes present in the dependent variable. Using Bayes' rule we can write:

$$p(c|x_1, x_2, \dots, x_d) \propto P(c)p(x_1, x_2, \dots, x_d|c)$$

where $p(c|x_1, x_2, \dots, x_d)$ is the posterior probability of class membership, i.e., the probability that $\boldsymbol{x}$ belongs to *C* given the features; $p(x_1, x_2, \dots, x_d|c)$ is the likelihood and $P(c)$ is the prior. Naive Bayes assumes that the feature values are independent given the class so we can decompose the likelihood into a product of terms:

$$p(\boldsymbol{x}|c) = \prod_{k=1}^{d} p(x_k|c)$$

and rewrite the posterior probability as:

$$p(c|\boldsymbol{x}) \propto P(c) \prod_{k=1}^{d} p(x_k|c)$$

Using Bayes' rule from above, we can implement a classifier that predicts the class based on the input features $\boldsymbol{x}$. This is achieved by selecting the class *c* that achieves the highest posterior probability.

$$c^* = argmax_j p(c_j|\boldsymbol{x})$$

Although the assumption that the predictor variables (features) are independent is not always accurate, it does simplify the classification task dramatically, since it allows the class conditional densities $p(x_k|c)$ to be calculated separately for each variable. In effect, Naive Bayes reduces a high-dimensional density estimation task to multiple one-dimensional kernel density estimations. Furthermore, the assumption does not seem to affect the posterior probabilities, especially in regions near decision boundaries, thus, it leaves the classification task unaffected. The probability density functions can be modeled in several different ways including normal, log-normal, gamma and Poisson density functions and discrete versions.

3. Implementation details

3.1. MNIST handwritten dataset

We will use a standard benchmark for digit recognition to evaluate the performance of the classifier. The MNIST dataset was assembled by Yann LeCun from multiple datasets. The training set contains 60000 images of handwritten digits from approximately 250 writers. The test set contains 10000 instances. The distribution of the digits is roughly uniform. For more details visit the link from [2]. We will use binomial probability distributions to model the probability density functions.

3.2. Training algorithm

Let *X* denote the feature matrix for the training set, as usual. In this case *X* contains on every row the binarized values of each training image to either 0 or 255 based on a selected threshold. *X* has the dimension *n x d*, where *n* is the number of training instances and *d=28 x 28* is the number of features which is equal to the size of an image. The class labels are stored in the vector *y* of dimension *n*.

The prior for class *i* is calculated as the fraction of instances from class *i* from the total number of instances:

$$P(C=i) = {}^{n_i}/_{n}$$

The likelihood of having feature *j* equal to 255 given class *i* is given by the fraction of the training instances which have feature *j* equal to 255 and are from class *i*:

$$p(x_j = 255|C=i) = \frac{count(X_{kj} = 255 \wedge y_k = i)}{n_i}$$

The likelihood of having feature *j* equal to 0 is the complementary event so:

$$p(x_j = 0|C=i) = 1 - p(x_j = 255|C=i)$$

To avoid multiplication by zero in the posterior probability, likelihoods having the value of 0 need to be treated carefully. A simple solution is to change all values smaller than 10^{-5} to 10^{-5}. Another alternative is to use Laplace smoothing, where $|C|$ signifies the number of classes:

$$p(x_j = 255|C=i) = \frac{count(X_{kj} = 255 \wedge y_k = i) + 1}{n_i + |C|}$$

3.3. Classification algorithm

Once the likelihood values and priors are calculated classification is possible. The values for the likelihood are in the interval [0,1] and the posterior is a product of 784 numbers each less than 1. To avoid precision problems, it is recommended to work with the logarithm of the posterior. Denote the test vector as *T* and its elements as T_j. These are the binarized values from the test image in the form of a vector. The log posterior of each class can be evaluated as:

$$\log\big(p(C=i|T)\big) \propto \log\big(P(C=i)\big) + \sum_{j=1}^{d} \log\Big(p\big(x_j = T_j \big| C=i\big)\Big)$$

Since the ordering of the posteriors does not change when the log function is applied, the predicted class will be the one with the highest log posterior probability value.

Load the first 100 images from class c:

```
char fname[256];
int c = 1;
int index = 0;
while(index<100){
      sprintf(fname, "train/%d/%06d.png", c, index);
      Mat img = imread(fname, 0);
      if (img.cols==0) break;
      //process img
      index++;
}
```

The prior is a Cx1 vector:

```
const int C = 3; //number of classes
Mat priors(C,1,CV_64FC1);
```

The likelihood is a Cxd vector (we only store the likelihood for 255):

```
const int d = 28*28;
Mat likelihood(C,d,CV_64FC1);
```

Header suggestion for the classifier:

```
int classifyBayes(Mat img, Mat priors, Mat likelihood);
```

4. Practical work

1. Load each image from the training set, perform binarization and save the values in the training matrix *X*. Save the class label in the label vector *y*. For the initial version use only the first 100 images from the first two classes.
2. Implement the training method.
2.1. Compute and save the priors for each class.
2.2. Compute and save the likelihood values for each class and each feature. Apply Laplace smoothing to avoid zero values.
3. Implement the Naive Bayes classifier for an unknown image.
4. Display the log posterior for each class. Optionally, convert the values to proper probabilities.
5. Evaluate the classifier on the test images and calculate the confusion matrix and the error rate. The error rate is the fraction of misclassified test instances (the complementary metric to the accuracy).
6. Train and evaluate on the full dataset

5. References

[1] *Electronic Statistics Textbook* – http://www.statsoft.com/textbook/stnaiveb.html
[2] LeCun, Yann, Corinna Cortes, and Christopher JC Burges. "The MNIST database." URL http://yann. lecun. com/exdb/mnist (1998).