Pattern recognition systems – Lab 8

K-Nearest Neighbor Classifier

1. Objectives

The purpose of this laboratory session is to introduce perhaps the simplest classifier: k-Nearest Neighbor classifier. The classifier is applied on a small image dataset with multiple classes.

2. Theoretical Background

Introduction

The purpose of a classifier is to assign a class to an unknown sample. Each sample is described by a feature vector. Perhaps one the simplest classifiers is the k-NN classifier. It makes the decision about the input sample based on the K nearest neighbors from a labeled training dataset. The next figure illustrates this by showing the sample as a blue square among the labeled samples. A circle enclosing the 5 closest neighbors indicates the region which is used to infer the class of the test sample. The radius of the circle is variable and always encloses K neighbors.

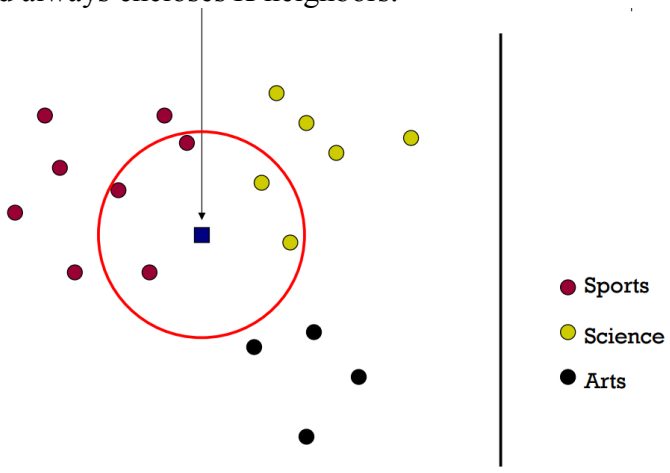

Figure 1. 5-NN classifier example with three classes

k-NN classifier is a non-parametric classifier, meaning that it does not construct a model for the classes it tries to distinguish. Instead it remembers the whole training set and at classification time the instance is classified online. It can be labeled as a type of instance-based learning, or lazy learning, since the classifier function is only approximated locally and all computation is deferred until classification.

Classification algorithm

Let the training dataset be defined in the form of a matrix of dimensions nxd denoted by X. Each line from X contains a single d dimensional feature vector called X_i, corresponding to a training instance. Also, let $\boldsymbol{y}$ denote the vector containing class labels. The dimension of $\boldsymbol{y}$ is $nx1$, each training instance having a class assigned to it. The elements of $\boldsymbol{y}$ are restricted to the set $\{1,2,...,C\}$, where C is the number of classes.

For an unknown test instance $\boldsymbol{x}$ the distance from each training example is calculated:

$$d_i = dist(x, X_i)$$

The distances are sorted in ascending order and the closest *K* instances are considered based on the distance. Each instance casts a vote for their class which is known from $\boldsymbol{y}$. The instance is classified as the class which has the most votes. A more formal description follows.

Let **p** be the permutation that sorts the distances in increasing order:

$$d_{p_1} < d_{p_2} < \cdots < d_{p_n}$$

The vote histogram is a *Cx1* vector constructed as:

$$\boldsymbol{h} = \sum_{k=1}^{K} \mathbf{1}(y_{p_k})$$

where $\mathbf{1}(y_{p_k})$ is a *Cx1* the indicator vector containing 1 only at the position $\boldsymbol{y}_{p_k}$ and 0 elsewhere. The sum accumulates the votes from the *K* closest neighbors. The class of the unknown instance is selected as:

$$c = argmax_i \boldsymbol{h}_i$$

There are multiple versions of the algorithm depending on the distance function used and the voting scheme. For example, votes can be weighed based on inverse distance using the following formula:

$$\boldsymbol{h} = \sum_{k=1}^{K} \frac{\mathbf{1}(y_{p_k})}{1 + d_{p_k}}$$

where we have added 1 to the distance to avoid division by 0 and to obtain a weight of 1 for distances equal 0.

The parameter *K* controls how many neighbors are considered. If *K=1*, only the nearest neighbor is considered. Increasing its value reduces the effect of noise on the result but makes the boundaries between the classes less distinct. In the extreme case when *K=n*, the whole training set is considered. If the votes are not weighted, this would classify an instance based on the prior distribution of the classes from the training set. In practice *K* is chosen to be an odd number to break ties when there are only two classes. Tests are performed on a validation set to obtain a proper value for *K* (hyper-parameter optimization).

The presented approach can also be used to perform regression if instead of choosing the class; we construct a weighted sum of the training instances as a response. The error rate of a k-NN classifier approaches that of the ideal Bayes error rate and is bounded by twice the Bayes error for two classes and for $n \to \infty$.

Global image features

Color images can be characterized by a global feature vector for the purpose of classification. A global feature vector of fixed dimension for any input image enables the process of classification. Global features usually describe certain relevant statistics of the image but lose information about the spatial layout of the image.

The image histogram can be viewed as a global feature vector for the image. The basic definition for a histogram of a grayscale image is that of a vector which counts the occurrences of each gray level intensity. It is a vector of dimension 256. In general, the histogram can be a vector of length *m* if we divide the [0,255] interval in *m* equal parts. In this case each bin in the histogram vector counts the number of gray level intensities falling in that particular bin. For example: if *m=8,* the first bin would count all intensities

between *0* and *256/m - 1=31;* the second bin between *32* and *63*; and so on. The histogram for a color image can be formed by concatenating the individual histograms for the separate channels. The size of the resulting histogram is of *3 x m*.

Evaluation of classifiers

Multiple metrics can be calculated to evaluate the performance of the classifier. The **confusion matrix** for a labeled dataset can be defined as a matrix containing in each cell M_{ij} the number of instances classified by the classifier into class i while having true class j. The ideal classifier would assign all instances to their correct class and would have large entries on the diagonal of the confusion matrix M_{ii}. In general, the values show which classes are confused with each other and can help to improve the classifier performance by identifying specific features that aid the discrimination between the two classes.

The accuracy for the classifier on a labeled test set is defined as the percentage of correctly classified instances. It is the complementary metric to the error rate. It does not offer relevant information if the classes are skewed. If the number of instances is unbalanced, a classifier that always predicts the most prevalent class will have a high accuracy. This is the typical situation, for example: pedestrian classifiers deal with a highly skewed distribution of much more background image samples than pedestrian samples. In this case, more relevant metrics are precision and recall for each class.

The accuracy can be calculated from the confusion matrix as:

$$Acc = \frac{\sum_{i=1}^{C} M_{ii}}{\sum_{i=1}^{C} \sum_{j=1}^{C} M_{ij}}$$

3. Dataset Statistics – scene recognition

The dataset for this session is for scene recognition. It contains 6 different classes: beach, city, desert, forest, landscape and snow. Images for each class are stored in subfolders and named as six digit numbers. The dataset is slightly imbalanced, the number of examples for each class ranging from 35 to 277. The training set contains 672 files, and the test set contains 85 files. Sample images from each class are given below:

beach city desert forest landscape snow

4. Implementation details

Suggestion for the histogram function header (the *hist* array is allocated previously):

```
void calcHist(Mat img, int nr_bins, int* hist)
```

Define the class names:

```
const int nrclasses = 6;
char classes[nrclasses][10] =
{"beach", "city", "desert", "forest", "landscape", "snow"};
```

Allocate the feature matrix and the label vector:

```
Mat X(nrinst, feature_dim, CV_32FC1);
Mat y(nrinst, 1, CV_8UC1);
```

Read all images from class *c*, calculate the histogram and insert the values in *X*:

```
int c = 0, fileNr = 0, rowX = 0;
while(1){
      sprintf(fname, "train/%s/%06d.jpeg", classes[c], fileNr++);
      Mat img = imread(fname);
      if (img.cols==0) break;

      //calculate the histogram in hist
      for(int d=0; d<hist_size; d++)
            X.at<float>(rowX, d) = hist[d];
      y.at<uchar>(rowX) = c;
      rowX++;
}
```

Allocate the confusion matrix:

```
Mat C(nrclasses, nrclasses, CV_32FC1);
```

5. Practical work

1. Implement a function for extracting the color histogram of an image.
2. Read all the images from the training set. For each image compute the color histogram with general bin size *m* and save it as a row in the feature matrix ***X***. Save the corresponding class label in the label vector ***y*.**
3. Implement the k-NN classifier for an unknown image and for a general *K* value.
4. Evaluate the classifier on the test set by calculating the confusion matrix and the overall accuracy.
5. Try out different values for the number of bins for the histogram and the parameter *K* to see which feature attains the best performance. Aim for over 65% accuracy.
6. Convert the input image into Luv or HSV color-space before histogram calculation.
7. Optionally, try out more complex features (such as histograms on image regions) or other distance metrics (Manhattan distance, weighted Euclidean).

References

[1] Wikipedia article - k-NN classifier
https://en.wikipedia.org/wiki/K-nearest_neighbors_algorithm
[2] Andrew Ng - Machine Learning: Nonparametric methods & Instance-based learning
http://www.cs.cmu.edu/~epxing/Class/10701-08s/Lecture/lecture2.pdf