Assignment 2: Discriminative and Generative Classifiers

Simone Jovon 882028

December 22, 2022

1 Task Description

The goal of the assignment is to write a handwritten digit classifier for the **MNIST database**[1]. These are composed of 70000 28x28 pixel gray-scale images of handwritten digits divided into 60000 training set and 10000 test set.

We have to train the following classifiers on the dataset:

1. SVM using **linear**, **polynomial of degree 2**, and **RBF** kernels;
2. Random forests;
3. Naive Bayes classifier where each pixel is distributed according to a Beta distribution of parameters α, β:

$$d(x, \alpha, \beta) = \frac{\Gamma(\alpha + \beta)}{\Gamma(\alpha)\Gamma(\beta)} x^{(\alpha-1)}(1-x)^{(\beta-1)}$$

4. k-NN.

For SVM and random forests we can use any library we want, but we must implement the Naive Bayes and k-NN classifiers.

[1] *http://yann.lecun.com/exdb/mnist/*

2 Theoretical Background

Let's start by defining what is **supervised learning**: given a set of example input and output pairs find a function that that does a good job of predicting the the output associated to a new input. Having said that all classifiers that we will see in this report are supervised learning models.

A classifier needs to be trained on a dataset to be able to predicting, in the best possible way, the output of new data (classification). On large datasets the data is split in two parts:

- **Tainig set**, to train the classifier;
- **Test set**, to test how accurate are the prediction of the classifier over new data.

During the training process the classifier don't have to fit the training date too precisely because this can lead to bad results on new data, this problem is konwn as **overfitting**.

2.1 Support vector machine (SVM)

If a data set can be split by linear separator the data space is called **linear separable**, when we faced a non-linearly-separable dataset we can take two approaches:

1. Use a more complex separator;
2. Use a linear saparator and accept some errors.

Support vector machine perform a linear classification, but with trick called **kernel trick** it can also perform non-linear classification.

In general, there can be multiple separators for a data set, a natural choise, and the one adopted by SVM, is to choose the one with the largest geometric margin.

In the case of a "nearly" separable dataset we may be willing to have a separator that allows a small misclassification by making the margin condition less strict. So it can be introduced a variable C that controll how much strictly the margin must be enforced, choosing a big value of C the classifier will work very hard to correctly classify all the data set, a lower value of C

instead allows more easily a misclassification of the point to achive a better margin.

On non-linearly-separable dataset if we trandform the feature values of the points inside the dataset in a non-linear way, we can trandform the dataset into one that is linearly separable. To adapt the dataset to the now feature space is used a function $\phi(x)$, the ϕ function can map the values of the dataset also to an higher-dimentional space and it can do a non-linear transformation.

Since SVM only use a dot product of the data

- $\phi(x_i) \cdot \phi(x_j)$, to train the classifier
- $\phi(x_i) \cdot \phi(u)$, to classify a new point

there is no need to compute $\phi(x)$ to every single point x, but having

$$\phi(x_i) \cdot \phi(x_j) = K(x_i, x_j)$$

we can simply define the function $K : X \times X \to \mathbb{R}, \{x_i, \dots x_n\} \subseteq X$, such function is called **kernel**.

There are various choises for kernels:

- **Linear kernel**: $K(x_i, x_j) = x_i \cdot x_j$
- **Polynomial kernel**: $K(x_i, x_j) = (1 + x_i \cdot x_j)^n$
- **Radial Basis Function**: $exp(-\frac{1}{2}\frac{\|x_i - x_j\|}{\sigma^2})$

2.2 Random forests

Decision trees can be seen as rules to perform categorization, but how to create a good decision tree that makes corrects categorization of new data? The main problem is to decide which node put in which position. This problem can be easily solved using an alghorithm like the ID3 Alghorithm, that using a measure called Information Gain it decides which node to put next on the tree.

Decision trees after the training can be perfectly fit the given dataset but may not be so accurate to classify new data, so there maight be overfitting. The Random forests solves this problem by constructing a multitude of decision tree during the training.

Before going further let's introduce a training technique called Bootstrap Aggregating (**Bagging**); given a training set it selects some random samples with replacement of the training set and it creates the associated decision trees. Having this set of decision trees, the predictions of the classifier is the average of the predictions of the various decision trees.

The training alghorithm for random forests follows the Bagging procedure but with a small difference; the alghorithm each time a split is to be performed, only a random subset of features is selected to create the associated decision tree.

2.3 Naive Bayes classifier

2.4 k-nearest neighbors (k-NN)

3 Implementation