Final Project of Machine Learning

Jonas Uhrig
03616049
jonas.uhrig@in.tum.de

Michael Wolfram
03616011
michaelwolfram@gmx.de

Abstract

In this paper we describe our final project for the Machine Learning class of 2013/14 at TUM. We use data set 4) containing information about body postures and movements and apply two similar Machine Learning algorithms that were implemented within this project. The used algorithms are called 'Ferns' and 'Random Tree' or 'Random Forest'.
The programming language of our choice is Matlab - some of the used approaches on Ferns are based on ideas from Mustafa Özuyal et al. [1], the development of the trees and forests was only based on ideas from the lecture.

1 Approach

We already know about Trees and Forests from the lecture - additionally we read of a Machine Learning algorithm called *Ferns* [1], which was recently (2007) introduced to detect objects in images. Ferns have a very similar behaviour as Forests and, in fact, are completely replaceable, i.e. intentionally building a Tree with the same tests in each layer makes it interchangeable with a Fern containing the same binary tests - though the important difference lies in the striking evaluation speed of Ferns for the classification of new samples.

For object detection purposes, Ferns are built using binary values as feature vector which initially posed a problem for the chosen PUC-Rio Data Set containing continuous (e.g the sensor data) and categorical (e.g gender, names) features. A key point of Ferns is chance so we thought it might serve to randomly choose a threshold within the range of each feature and do a simple comparison, resulting in a new binary feature value - this approach was also used by Miron Bartosz Kursa [2], which encouraged us to continue the implementation.

2 Data

sitting: 50631 sittingdown: 11827 standing: 47370 standingup: 12415 walking: 43390

3 Method

This section gives a short overview of the properties of both methods. The algorithms and basic methods to tweak them were taken from the original papers (see section "References") and some lecture slides but in the final implementation we made some slight changes to cope with the given restrictions of this project.

3.1 Forest

3.1.1 Basic Principle

As known from the lecture, we implemented a Forest as ensemble of a certain number of Decision Trees. Our Trees are defined by the following parameters:
First, there is the possibility to prune the tree. Referring to the lecture, since we implemented bagging and feature subset selection for the training of each Tree of a Forest to overcome overfitting on the whole data set pruning as third method became superfluous.
Other properties of a Tree are its criteria when to stop growing. We implemented the four suggested ones from the lecture, i.e. the depth of the tree can be limited to a certain maximum, a lower bound for the benefit of another split can be specified, if the number of training samples in a certain node is below a given threshold it will not be split any more and is marked as leaf and lastly, the tree stops growing at a particular node if the distribution of labels/classes of the training data in this node is pure.
For the last stopping heuristic we implemented the impurity measures Gini index, Entropy and misclassification rate from the lecture. After some small initial tests we decided to concentrate on the Gini index for further testing since the Gini index was more accurate on a subset of the given training data even though the Entropy was slightly faster.

3.1.2 Optimizations

A Random Forest consists of Trees not trained with exact the same training data. This would Due to the fact that we were faced with limitations regarding time and computation power we did minor modifications to the building process of a Forest.

3.2 Ferest

In this section we describe Ferns as well as *Ferests*, which is a forest of Ferns [1]. We describe methods for training and testing together with the mathematical background and then consider different adjustments that have been made for testing and optimization issues.

3.2.1 Mathematical Background

Generally, what training is supposed to achieve is a distribution over given feature vectors that classifies a new sample into one of the given classes. In other words, we want to obtain the class which has the highest posterior probability over all class labels, given the specific features of a sample: $\arg\max_k P(C_k|f_1, f_2, \cdots, f_N)$. Applying Bayes' rule results in the following expression: $\arg\max_k P(f_1, f_2, \cdots, f_N|C_k)P(C_k)$, meaning that if we find the joint distribution over all features, we know to which class a sample belongs to - unfortunately, this is very hard to get. Instead, we make the Naive Bayes assumption of independent and identically distributed features which leads us to the first classification formula:

$$Class(f) \equiv \arg\max_k P(C_k) \prod_{n=1}^{N} P(f_n|C_k)$$

The assumption that all features in our data set have nothing to do with each other is rather strong and often incorrect. Thus, this usually delivers quite accurate results and is easy and fast to learn.

The idea of a Ferest is to combine joint class-conditional distributions with a Naive assumption to a *Semi-Naive* method which achieves accurate results within a good range of complexity and computational effort. Instead of assuming conditional independence of features, a Ferest F_l uses L Ferns of size S and assumes independence between those. The new and final classification formula is then:

$$Class(f) \equiv \arg\max_k P(C_k) \prod_{l=1}^{L} P(F_l|C_k)$$

3.2.2 Training

As described in the section above, a single Fern has to learn the conditional probability distributions for each of the given classes during the training phase, given a certain feature vector $\mathbf{f} = (f_1, f_2, \cdots, f_N)$ of size N.

In the original implementation, every Fern randomly selects S features and does a binary test on them, resulting in a binary number between 0 and $2^S - 1$. However, the original implementation was working on image patches to detect objects and the binary tests were picked by comparing two random pixel intensities in the given patch. For our initial implementation, as Kursa [2] suggests, we chose a random threshold within the currently given feature space and compare this threshold with the respective value of each sample.

Like this, we get a decimal number for every training sample by iteratively applying the random tests - this number is then used to put the current sample into one of 2^S *buckets*, comparable to hashing.
Using this pattern of dropping samples into buckets for every sample of a class present in the training set delivers a multinomial distribution (*histogram*) of the new features (random binary tests) for the regarded class. This has to be normalized prior to be an actual probability distribution.

To train a Ferest, we simply train L Ferns and store the specific histograms together with the randomly picked thresholds for the tests.

3.2.3 Testing

The classification of a new sample using a single Fern is very efficient: The previously stored tests are applied to the tested sample in the same order to find the bucket into which this sample drops. The combined buckets of all classes, after normalizing, then represent the corresponding class posterior distribution, given the features of the tested sample. This means that every Fern knows with which certainty a sample would be of which class.

For a Ferest, the results of all Ferns are combined using the Naive Bayes' assumption - which is now more appropriate as all Ferns check on random features and thresholds.

3.2.4 Adjustments

Trying to achieve more accurate results,

4 Results

4.1 Forest

4.2 Fererst

4.3 Comparison

References

[1] M. Özuysal, P. Fua, and V. Lepetit. Fast Keypoint Recognition in Ten Lines of Code. *2007 IEEE Conference on Computer Vision and Pattern Recognition*, pages 1-8, June 2007

[2] M. B. Kursa. Random ferns method implementation for the general-purpose machine learning. *Interdisciplinary Centre for Mathematical and Computational Modelling, University of Warsaw*, February 2012