SAPIENZA UNIVERSITY OF ROME

Optimization Methods for Machine Learning

Homework 1

Ivan Bergonzani, Michele Cipriano

November 27, 2017

1 INTRODUCTION

The aim of the homework is to train and compare different neural networks on a regression problem. In particular, the regression task is performed on the Franke's function, building a dataset from it by sampling 100 random points (x^i, y^i) with noise, i.e. with $y^i = f(x^i) + \varepsilon^i$ where ε^i is a random number in $[-10^{-1}, 10^{-1}]$ and f is the Franke's function. The dataset has been split into a training set (70% of the dataset) and a test set (the remaining 30%).

Two architectures have been compared using different training methods and different hyperparameters. In particular, the multi-layer perceptron (MLP) and the radial basis function network (RBFN) have been trained with full minimization, two blocks method and decomposition method. In the full minimization, the training error is minimized w.r.t all the weights using the gradient descent algorithm. In the two blocks method, for the MLP the error is minimized via an extreme learning procedure and for the RBFN the error is minized by first selecting the centers through a clustering algorithm and then by solving a linear least squares problem. In the decomposition method the error is minimized by alternating a convex minimization w.r.t. the output weights and a non-convex minimization w.r.t. all the other weights.

The project has been developed in Python with Tensorflow and Numpy for the learning algorithms and the computation of the tensors.

TODO: RESULTS.

2 FULL MINIMIZATION

As mentioned before, the full minimization is done by using the gradient descent method on the training error function (DEFINED ABOVE?).

Let's consider a shallow MLP with linear output:

DREBIN is a dataset of Android malwares developed to study machine learning approaches for malware detection. It's composed of 123,453 benign applications and 5,560 malwares where each program is represented by a set of features extracted from the manifest file of Android (*AndroidManifest.xml*) and from the disassembled code of the application at runtime.

All the features are grouped in subsets. In particular, there are four groups from the manifest file:

- S_1: *Hardware components*
- S_2: *Requested permissions*
- S_3: *App components*
- S_4: *Filtered intents*

and four groups from the disassembled code:

- S_5: *Restricted API calls*
- S_6: *Used permissions*
- S_7: *Suspicious API calls*
- S_8: *Network addresses*

Researches on DREBIN have shown that not only machine learning approaches outperform most common anti-virus scanners, but that it's also possible to analyze a program directly on the smartphone with an average of 10 seconds[1]. This promising results make malware analysis worth studying and set up the basis for improvements in the future.

3 Two Blocks Method

In order to make it possible to train the classifiers on the dataset, all the data needs to be preprocessed. As done in [1] the features of each application are embedded into a vector space. Each element of the vectors can be either 1 or 0 and describes the presence of a feature in a program (e.g. `android.hardware.wifi`, `SEND_SMS`).

To speed up the training phase only a subset of features and a subset of applications have been considered. In particular subgroups S_1, S_3 and S_8 have been discarded and a subset of 9,567 applications have been used to train the classifiers.

This made it possible to embed the features in a vector space of dimension 1,773 against 545,000 used in the paper.

4 Decomposition Method

The first classifier trained is the decision tree. The idea behind this kind of classifier is simple: exploit the data in order to learn simple decision rules. Rules depend on the features of the data, which are represented in this case by a multi-dimensional vector, and are used to understand the category of each input.

Different decision trees have been trained and compared varying the maximum depth of the trees themselves, as it is possible to see from table 8.1. Increasing the depth of the tree seems to improve the behaviour of the classifier, but it's important to remember that in these experiments only a small subset of the whole dataset has been used. This makes it simpler to create rules that split the two classes.

Probably the most interesting aspect of decision trees is that it's easy to understand its behaviour just by seeing the generated rules. Taking as an example the decision tree with maximum depth 3 (figure **??**), the rules generated consider the following features:

- `X[489]: intent::android.intent.action.BOOT_COMPLETED`
- `X[1360]: permission::android.permission.SEND_SMS`
- `X[1336]: permission::android.permission.READ_PHONE_STATE`
- `X[394]: call::getSubscriberId`
- `X[75]: api_call::android/content/Context;->startActivity`
- `X[742]: intent::android.intent.action.SIG_STR`
- `X[839]: intent::android.intent.category.DEFAULT`

which suggests that most malwares (or viceversa most benign applications) needs to be executed directly after the booting of the smartphone, to send SMS or to access to the state of the phone.

5 Conclusion

The second tested classifier is Naïve Bayes, which exploits the Bayes' Theorem and the frequency of the data contained in the dataset to compute the likelihood of a sample to be, in this case, a malware (or viceversa a benign application).

The Naïve Bayes classifier used makes use of Laplace smoothing, which is a technique used to smooth probabilities. The training was made by varying the smoothing parameter α in order to try to improve the accuracy. As it is possible to see from table 8.1 adding a smoothing parameter seems to slightly decrease the performances of the classifier.

6 Support Vector Machine

The idea behind Support Vector Machines (SVMs) is to construct a set of hyperplanes that split the data with the largest possible margin (distance between the hyperplane and the nearest sample in the training set). SVMs are built upon kernel functions, which implicitly map the samples to a high-dimensional space.

Here SVMs have been training using different kernel functions, as shown in table 8.1. It's interesting to notice that SVM with linear kernel performs better than the one with kernel based on radial basis function. This could be due to the fact that the dataset is smaller and the vector space dimension has been reduced a lot to reduce training time.

Classifier	hparams	Acc.	Prec.	Rec.	F1	FPR
Decision Tree	`max_depth=1`	0.788	0.886	0.675	0.767	0.092
Decision Tree	`max_depth=3`	0.888	0.905	0.876	0.890	0.098
Decision Tree	`max_depth=5`	0.925	0.918	0.939	0.928	0.090
Decision Tree	`max_depth=10`	0.948	0.942	0.959	0.950	0.063
Decision Tree	`max_depth=None`	0.950	0.945	0.960	0.952	0.059
Naïve Bayes	$\alpha = 0.0$	0.884	0.913	0.857	0.884	0.088
Naïve Bayes	$\alpha = 0.5$	0.876	0.903	0.851	0.876	0.097
Naïve Bayes	$\alpha = 1.0$	0.875	0.901	0.851	0.876	0.099
SVM	`kernel=linear`	0.959	0.960	0.961	0.960	0.043
SVM	`kernel=poly`	0.484	0.000	0.000	0.000	0.000
SVM	`kernel=rbf`	0.925	0.945	0.907	0.926	0.056
SVM	`kernel=sigmoid`	0.908	0.938	0.880	0.908	0.062
MLP	`solver=sgd`	0.959	0.960	0.960	0.960	0.042
MLP	`solver=adam`	**0.969**	**0.970**	**0.970**	**0.970**	**0.032**

Table 8.1: All the experiments performed on DREBIN showing the type of classifier, the hyperparameters, the accuracy, the precision, the recall, the F1-score and the false positive rate (FPR).

7 Multi-Layer Perceptron

The last tested classifier is the multi-layer perceptron (MLP), which is a feedforward artificial neural network consisting of three or more layers (one input layer, one output layer and multiple hidden layers in between).

Two different MLPs have been tested on DREBIN obtaining both good results. The first MLP has been trained using SGD algorithm with momentum ($\beta = 0.9$) while the second one has been trained using Adam. Both used *ReLU* as non-linear activation function, a constant learning rate $\eta = 0.001$ and a maximum number of iterations of 400. The second MLP obtained the best results in all the measures, outperforming all the previous classifiers, as shown in table 8.1.

8 Conclusion

Multiple experiments have been conducted on DREBIN dataset showing that machine learning methods are able to generalize the data and classify malware applications based on their features. Of course, all the experiments are simplified by the fact that the amount of samples used are only a part of the whole dataset. Future experiments could make use of deep learning, trying to optimize the hyperparameters of the MLP, increasing the number of layers, changing the activation functions and making use of the whole dataset. The use of GPUs should make the training much faster, optimizing computation of the matrices and performing the forward and the backward step with a larger number of samples altogether.

REFERENCES

[1] D. Arp, M. Spreitzenbarth, M. Hubner, H. Gascon, and K. Rieck, "DREBIN: Effective and Explainable Detection of Android Malware in Your Pocket.," in *NDSS*, The Internet Society, 2014.