31/01/2019

Ben-Gurion University of the Negev

Faculty of Engineering Sciences

Department of Software and Information systems

Implementations of Machine Learning Algorithms

Prof. Lior Rokach

Assignment 3

Maya Schvetz 305249708

Yuval Heffetz 302957139

Introduction

In this exercise we implemented a machine learning method for generating ensembles, Decorate, that directly constructs diverse hypotheses using additional artificially-constructed training examples. We had evaluated two methods to generate the data, first one by using the distribution of each column as was explain in [1]. The second method for generating the data was conducted using the GAN (generative adversarial network) algorithm. After implementing the Decorate algorithm and combined it with the GAN algorithm we had evaluated the two data generation methods on 10 different datasets using 10-fold cross validation.

Implementation

We chose to implement the models and the evaluation experiment in Python. A short explanation of our implementation in the google Colaboratory's notebook **hw3_yuval_maya.ipynb**:

1. We created a class called ***Decorate*** that implements the general meta-learner that can use any strong learner as a base classifier, in our case, a decision tree classifier, to build diverse committees. All the methods that the meta-learner uses were included in this class, as detailed:
2. The ***__init__*** method that initializes all the algorithm parameters as were introduced in the article and H.W assignment in addition to the GAN parameters that should be used for the integrated model. The attribute ***generator*** determines which method to use for generating synthetic data (GAN or the method presented in [1]).
3. The ***fit*** method runs the DECORATE algorithm. It either generates data by the method detailed in the article or samples data from a set of synthetic data generated with GAN algorithm, depending on ***self.generator*** attribute. It follows the same steps for both models (DECORATE and DECORATE+GAN) except for the data generation step.
4. The ***predict_proba*** method predicts class probabilities of the input samples X, based on the following equation from the article:

$$\hat{P}_y(x) = \frac{\sum_{C_i \in C^*} \hat{P}_{C_i,y}(x)}{|C^*|}$$

- Where: x – sample, C^*- ensemble, y- class label, P- probability of a sample belonging to class base on a classifier from the ensemble

5. The ***predict*** method selects the most probable class as the label of the sample, based on the following equation:

$$C^*(x) = \arg\max_{y \in Y} P_y(x)$$

- Where: x – sample, C^*- ensemble, y- class label, P- probability of a sample

6. The ***compute_error*** method computes the ensemble's error based on the following equation:

$$\epsilon = \frac{\sum_{x_j \in T, C^*(x_j) \neq y_j}}{m}$$

- Where: m – number of samples, x – sample, C^*- ensemble, y- class label

7. The ***generate_artificial_examples_basic*** method generates artificial training data by randomly picking data points from an approximation of the training-data distribution.
8. The ***generate_artificial_labels*** method is first assigning to each training artificial example the class membership probabilities predicted by the ensemble. Then, replaces zero probabilities with 0.001 and normalizes it. Next, labels are selected, such that the probability of selection is inversely proportional to the current ensemble's predictions.
9. The ***normalize_probas*** method replaces zero probabilities with 0.001 and normalize the probabilities to make it a distribution.
10. The ***generate_artificial_examples_GAN*** function is generating synthetic data using the GAN algorithm. It uses the GAN implementation from this repository. It creates a parameters-configuration file according to the number of attributes and rows to sample we define for the dataset. The number of samples is determined so it will be sufficient for all the algorithm's iterations with no repetitions. The GAN is trained on a few randomly selected

parameters configurations (several epochs each) and the synthetic data of the configuration that yielded the best performance (accuracy) is selected. Because of runtime considerations we generated data only once for each dataset, before splitting to 10-fold CV, by calling this function.

11. ***MAIN*** function is where the experiment is conducted by loading each of the 8 datasets, performing preprocessing steps to prevent missing values and for labels and categorical attributes encoding and then performing 10-fold cross validation, each fold trained with each model, DECORATE and DECORATE with GAN. Due to runtime and memory considerations we limited the size of the datasets to 50k. When the size was unlimited the Colab machine would simply collapse so this limitation was necessary.

How to Run the Script

1. A google drive folder called ***ml_hw3_yuval_maya*** was shared with you (with the account: nathanie@post.bgu.ac.il). Open the folder that is located under your ***Team Drives*** directory (אחסון שיתופי).
2. Right click on the Google Colab notebook ***hw3_yuval_maya.ipynb*** → open with → Colaboratory
3. The Colaboratory notebook will open. You can now run the cells one by one. Make sure that you mount the drive (in the first cell) of the correct account where the folder is shared with. Notice that you will be asked to copy a code from a link.
4. Make sure you run on Python 3 with GPU.

The notebook file is also attached in the zip file ***HW3-305249708_302957139.zip*** but it will not run by itself. Notice that if you run the scripts now, the GAN algorithm will not generate the data again since there are already files containing the generated data for each dataset in the folder ***expdir*** (it skips existing files).

Evaluation

Our evaluation compares the performance of each of the two data generation methods – basic VS GAN. We used four different metrics suited for multi-class classification to evaluate the performance of each method:

1. **Accuracy**

$$\texttt{accuracy}(y, \hat{y}) = \frac{1}{n_{\text{samples}}} \sum_{i=0}^{n_{\text{samples}}-1} 1(\hat{y}_i = y_i)$$

   This metric could be problematic in unbalanced datasets, so we made sure the datasets are approximately balanced and added three more metrics.

2. **Macro-average precision**

   Precision is computed by:

$$\text{precision} = \frac{tp}{tp + fp}.$$

   For multi-class problems we can use macro-averaging of the precision (or recall) which is a method for averaging the sum of precision of each label.

3. **Macro-average Recall**

   Similarly, we use macro-averaging over the labels' recall values:

$$\text{recall} = \frac{tp}{tp + fn}$$

4. **Macro-average F1**

   F1 is the harmonic-mean of precision and recall and computed by:

$$F_1 = 2 \cdot \frac{\text{precision} \cdot \text{recall}}{\text{precision} + \text{recall}}.$$

Another important factor we decided to evaluate, and compare is **time** of the training process and prediction of each method.

Results

The two methods were tested on ten different datasets of classification problems, obtained from here and here.

1. **UK Accidents 10 years history with many variables [link]**

   Number of instances: 2.22m

   Number of attributes: 15

   Number of classes: 4

	Basic	GAN
Accuracy	0.4879	0.6411
Precision*	0.3107	0.321
Recall*	0.3213	0.2676
F1*	0.3052	0.2383
Training Time [sec]	64.178	2933.196
Testing Time [sec]	0.013	0.01

*macro-averaged

2. **Bureau [link]**

   Number of instances: 1.72 M

   Number of attributes: 300

   Number of classes: 2

	Basic	GAN
Accuracy	0.8581	0.9226
Precision*	0.5407	0.5618
Recall*	0.5662	0.5236
F1*	0.5465	0.5255
Training Time [sec]	3.345	162.003
Testing Time [sec]	0.001	0

*macro-averaged

3. **Home Credit [link]**

Number of instances: 48.7 K

Number of attributes: 121

Number of classes: 2

	Basic	GAN
Accuracy	0.8816	0.92
Precision*	0.577	0.5886
Recall*	0.5762	0.5284
F1*	0.565	0.5269
Training Time [sec]	1.834	98.16
Testing Time [sec]	0.008	0.001

*macro-averaged

4. **LendingClub Issued Loans [link]**

Number of instances: 760 K

Number of attributes: 56

Number of classes: 13

	Basic	GAN
Accuracy	0.3546	0.5332
Precision*	0.1329	0.1174
Recall*	0.1445	0.1001
F1*	0.1228	0.0926
Training Time [sec]	89.225	2345.716
Testing Time [sec]	0.041	0.024

*macro-averaged

5. **ISOLET [link]**

Number of instances: 7797

Number of attributes: 617

Number of classes: 26

	Basic	GAN
Accuracy	0.5632	0.5545
Precision*	0.5631	0.5588
Recall*	0.5641	0.5551
F1*	0.5546	0.5406
Training Time [sec]	14.736	558.85
Testing Time [sec]	0.01	0.001

*macro-averaged

6. **MAGIC Gamma telescope [link]**

Number of instances: 19020

Number of attributes: 11

Number of classes: 2

	Basic	GAN
Accuracy	0.8651	0.8544
Precision*	0.8548	0.8498
Recall*	0.8461	0.8259
F1*	0.85	0.835
Training Time [sec]	32.629	1096.108
Testing Time [sec]	0.015	0.009

*macro-averaged

7. **Skin Segmentation [link]**

Number of instances: 245 K

Number of attributes: 3

Number of classes: 2

	Basic	GAN
Accuracy	0.993	0.9964
Precision*	0.9847	0.9918
Recall*	0.9951	0.9971
F1*	0.9896	0.9944
Training Time [sec]	49.302	2163.252
Testing Time [sec]	0	0.012

*macro-averaged

8. **The Broken Machine [link]**

Number of instances: 900 K

Number of attributes: 58

Number of classes: 2

	Basic	GAN
Accuracy	0.6223	0.6642
Precision*	0.5169	0.4954
Recall*	0.512	0.4963
F1*	0.5072	0.4534
Training Time [sec]	4.387	155.504
Testing Time [sec]	0.015	0.009

*macro-averaged

9. **Rain in Australia [link]**

Number of instances: 142 K

Number of attributes: 24

Number of classes: 2

	Basic	GAN
Accuracy	1	1
Precision*	1	1
Recall*	1	1
F1*	1	1
Training Time [sec]	71.995	3508.411
Testing Time [sec]	0.046	0.044

*macro-averaged

10. **Bank Marketing [link]**

Number of instances: 45.2 K

Number of attributes: 17

Number of classes: 2

	Basic	GAN
Accuracy	0.8535	0.8859
Precision*	0.693	0.7456
Recall*	0.7976	0.6775
F1*	0.7247	0.6927
Training Time [sec]	48.053	2357.168
Testing Time [sec]	0.012	0.002

*macro-averaged

Results summary

The following table summarize the results

	Basic Average	GAN Average	t Stat	t Critical one-tail	significant
Accuracy	0.74	0.79	2.34	1.83	GAN
Precision*	0.61	0.62	0.85	1.83	
Recall*	0.63	0.59	3.04	1.83	Basic
F1*	0.61	0.58	3.74	1.83	Basic
Training Time [sec]	37.96	1537.83	3.81	1.83	GAN
Testing Time [sec]	0.01	0.01	1.59	1.83	

Significance of the results

In order to test the significance of the results between the two methods, we performed a paired samples t-test (one tail, α=0.05, null H-there isn't a significant difference between basic and GAN methods) for each one of the tested scores: Accuracy, Precision, F1, training time and testing time.

Accuracy results

We got a T-value of 2.34 and we need a T value of at least 1.83 to reject the null hypothesis. Thus, we can reject the null hypothesis- the GAN method achieved significantly higher Accuracy results.

Precision results

We got a T-value of 0.85 and we need a T value of at least 1.83 to reject the null hypothesis. Thus, we can't reject the null hypothesis.

Recall results

We got a T-value of 3.04 and we need a T value of at least 1.83 to reject the null hypothesis. Thus, we can reject the null hypothesis- the Basic method achieved significantly higher Recall results.

F1 results

We got a T-value of 3.74 and we need a T value of at least 1.83 to reject the null hypothesis. Thus, we can reject the null hypothesis- the Basic method achieved significantly higher F1 results.

Training Time results

We got a T-value of 3.81 and we need a T value of at least 1.83 to reject the null hypothesis. Thus, we can reject the null hypothesis- the GAN method took significantly longer training time.

Testing Time results

We got a T-value of 1.59 and we need a T value of at least 1.83 to reject the null hypothesis. Thus, we can't reject the null hypothesis

The full results and analysis are available at the ***results.csv*** and ***results_statistical_analysis.xlsm*** files.

Conclusions

1. The most noticeable result is that the DECORATE algorithm improves when the data is generated with GAN algorithm instead of the basic method on the ***accuracy*** evaluation metric **but** achieves poor results on ***precision-recall*** metrics compared to the basic method. This can happen when the model over predicts the samples to the majority class in unbalanced datasets and so the accuracy is high, but precision-recall is low for the minority classes.
2. The ***time*** factor plays a significant role - adding the generation of data with GAN consumes a very long run-time compared to the basic algorithm. The time factor can be very important in some cases and might rule out using the model on the expense of reduction in performance in other metrics.
3. We suspecteed that as the number of rows\attributes rise between the data sets we will get better F1 and accuracy results for the GAN data generation method. However, we can see based on the four graphs below, figures 1-4, that the two methods aren't affected that match from the change by the number of rows\attributes in the data sets, an almost horizontal trend line in both graphs. We assume that an hyperparameter tuning for the GAN method is needed to get from this method better results where there exist more data.

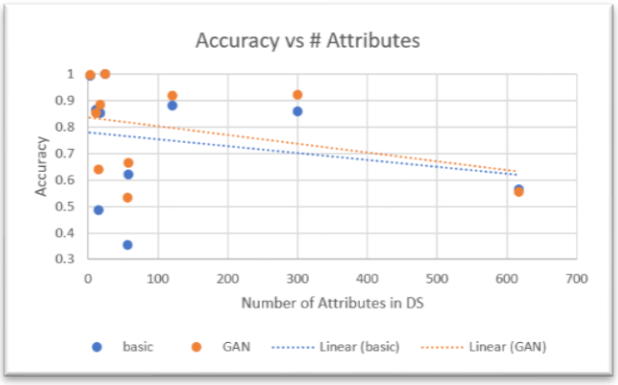

Figure2 Accuracy VS Number of Attributes

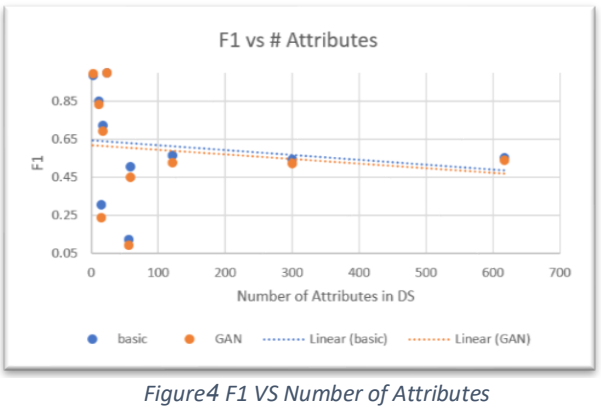

Figure4 F1 VS Number of Attributes

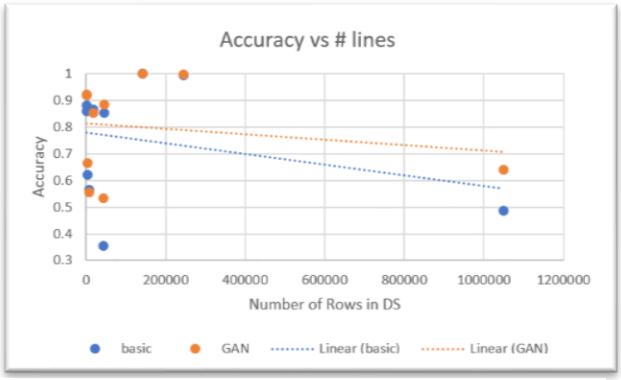

Figure3 Accuracy VS Number of Rows

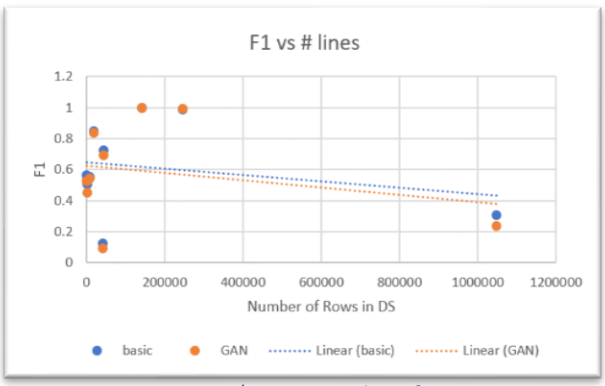

Figure1 F1 Vs Number of Rows

Reference

[1] P. Melville and R. J. Mooney, “Creating diversity in ensembles using artificial data,” *Inf. Fusion*, vol. 6, no. 1, pp. 99–111, 2005.