Machine Learning 2017/2018 – Homework 1 Luca Vargiu 1422135

I. **Description of the leaning problem**

The learning problem taken in consideration in respect of Malware Analysis during this homework is the Malware detection problem on Drebin dataset in Android environment.

Malware detection problem, in general, implies as a result the binary classification, from a passed dataset, of applications or softwares into "malware" or "not malware" sets.
The learning procedure can take place in a supervised way, that uses a labeled and predefined training set or in a unsupervised way, that doesn't need a labeled training set. Malware instances can be classified and represented by a finite set of features and/or behaviors, so supervised learning allows to infer a good categorization based on a labeled training dataset.
Feature extraction, so the process that returns the features' values from an instance, can be based on static features, that can be extracted simply by looking at the sample files or on dynamic features, that can be extracted executing the sample and looking at the execution traces or on an hybrid analysis, based on the previous ones.
In the Android environment, static features can be derived directly from the ".apk" file of the application, giving informations about components, permissions, API calls, strings and flow graphs. Dynamic features from the execution trace relate informations on resource consumption, system calls and download patterns.

Drebin dataset is a huge collection of samples of Android applications, that uses static analysis to extract features from samples.
It contains 123,453 bening applications and 5,560 malware, where features are extracted from the manifest.xml file (requested hardware components, requested permissions, app components and filtered intents) and from the disassembled code (restricted API calls, used permissions, suspicious API calls and network addresses).
Features extracted from an app are stored inside a text file, whose name is the SHA1 hash of the apk. Each line of these files contains a key-value entry, that defines a type of a feature and its found content. A dictionary file in ".csv" format is also provided as a ground truth to identify malwares in the whole dataset, with also an indication to the family they belongs to.

II. **Algorithms used**

In the current homework, there has been choosen to use an implementation of Naive-Bayes classifier, in particular based on Multinomial distribution, and an integration in the code of the library libSvm to implement an SVM classifier, that supports up to three different types of kernels (linear, polynomial and radial basys function). In parallel to these two main classes of algorithms, there have been implemented methods to make a k-fold cross evaluation of a passed dataset by dividing samples in k test subsets and finding the correspont training subsets.
Furthermore, tested datasets have been randomly generated through an integrated method, that can control the percentage of malware samples in the resultant dataset (it has also been considered a pure random generation).
Generated datasets contain on first line the accepted features (by default the most numerous three) and in the rest the names of the files, that represent the samples.
All write and read operations on file have been implemented with two static methods, defined in the main class, that acts on LinkedLists to memorize informations and to facilitate access, modification and processing(ex. Random accesses).

Naive-Bayes classifier implementation takes place with a Text-Classifier approach, implemented through three classes:

1) Document class, that is the representation of a feature document, created in order to take trace of number of words and to check if certain word is contained in the document. It also makes a more inner division of feature text through URL splitting. The input paramaters of it's constructor are an ID, that identifies the document, and a formatted string (all words divided by space charachters), that is the document content. The class is also used in the categorization of the samples by matching a category to the current document.

2) MultinomialClassifier class, that is a container able to take trace and to categorize the informations of all Document objects added. In particular, it takes trace of the total number of words in the training set, how many words are in a category, how many occurencies of a word appear in a category and the number of documents for each category. Is used in the prior probabilities estimation in the Naive-Bayes algorithm with a Multinomial Distribution.

3) NaiveBayesCrossFolding class, that implements the actual code for the Naive-Bayes Text-Classifier and the k-fold cross validation. In the constructor, a LinkedList, containing the dataset information (that is processed to obtain the file names and the accepted features), an integer k, that represents the folding degree (by default 10), and the dataset file name, used for producing the report file, are passed as parameters.
Execute() is the main method, used to produce the final Confusion Matrix and the final Evaluation Metrics returned in the report file.

Firstly, prepareKcrossSubsets() method divides the dataset into k trainsets and testsets, then in range of k each trainset is trained and tested with the corresponding testset and, basing the result on the ground truth reported in the dictionary file, the result is registered in an intermediate confusion matrix and evaluated for each step. Finally, all confusion matrices are merged into the final one, the total evaluation metrics are calculated and the report file is printed.
Train(int step) method analyzes all words, that are found in the sample files of the trainset for the current step, then it memorizes in a Document object (one for each sample) the selected words related to the accepted features only, then all the Documents are added to a MultinomialClassifier object related to the current step, that will be used in the test phase.
TestData(int step) method starts retrieving the MultinomialClassifier object related to the current step and gaining from it all useful informations in order to compute prior probabilities for categories and words of the trainset. Then all derived Documents from the correspondent testset are tested confronting the max probability between the categories with a logarithmic transformation, passing from the product of all probabilities to the sum of them and then memorizing the result category in the Document object. Finally, after confrontation of the results with the ground truth, the confusion matrix for the intermediate step is produced together with the partial evaluation metrics.

SVM classifier implementation and in particular the integration of libSvm in the code together with the adaptation of the samples features has been implemented through the usage of two main classes:

1) Categorizer class, that acts as a dictionary for passed features and categories, matching them with integer values that represent labels and indexes for the libSvm vector representation (<label> <index1>:<value1> ...).
Integer values are correlated to new strings by assigning consecutive id values in respect of the last addition. If a string has already been mapped, the matching integer value is returned.

2) SvmCrossFolding class integrates and launches the libSvm classifier, adjusting the parameters for the requested kernel, during the train phase, and also implements a k-fold cross validation procedure, in the same way as seen in the Naive-Bayes classifier (execute() method acts in the same way). In the constructor, a LinkedList containing the dataset information (that is processed to obtain the file names and the accepted features), an integer k that represents the folding degree (by default 10), the dataset file name, used for producing the report file, and an integer ker, that indicates the type of kernel that will be used, are passed as parameters.
TransformToSVMFormat(String sha) method translates the sample content, denoted by the file name sha, into a string in the libSvm format, selecting only the accepted features strings and using the Categorizer istance during the translation. The production of the vector follows the representation of

sparse format, reporting only the indexes that appear in the sample with value equal to “1” and omitting the others, as described in libSvm FAQ.
Train(int step) method translates the training set related to the step into file, where each line represents a sample in the libSvm format, then a model file is produced depending on the selected kernel. In relation to the type of kernel, paramaters are defined as it follows:

- Linear : "-s 0 -t 0";

- Polynomial : "-s 0 -t 1 -c 100 -g 1 -r 1 -d 4";

- Radial Basis Function: "-s 0 -t 2 -c 100 -g 1 -r 1 -p 0.01".

Predict(int step) method translates, sample by sample, the testset related to the step, predicting each time which category the sample belongs to and printing the result into an output file. Then the output file is read and, through the Categorizer istance traduction, the confusion matrix for the intermediate step is compiled and produced as seen in the Naive-Bayes classifier implementation.

All the provided code is written in Java language.

III. **Description of the results obtained**

Tests executed in order to compile this report concern on five datasets (test1.txt, test2.txt, test3.txt, test4.txt, test5.txt), where the consecutive ones are a generalization of the previous ones (for example text2.txt contains the same samples of text1.txt plus other ones) in a linear order, based on the first dataset's size. In our case, the first set has a size of 1000 samples and the final one has a size of 5000 samples. The structure of a dataset file results the same as it has been described in the previous paragraph.
Furthermore, these five cases were tested under four different scenarios of malware concentration (random but under 25%, 25%, 50% and 75%) and with the two presented algorithms with their variants (Naive-Bayes Text-Classifier, SVM Classifier – linear kernel, SVM Classifier – polynomial kernel, SVM Classifier – radial basis function kernel).

In the following subparagraphs, one for each scenario, will be represented in a table the final Confusion Matrix values obtained for each algorithm on each dataset and the computed evaluation metrics, concerning Precision (real malwares among the considered ones), Recall (malware spotted among those of the testset), False positive rate (samples considered malwares among the benign files), Accuracy (samples classified correctly) and F-measure (weighted average of precision and recall).

Each confusion matrix is represented as it follows:

True positive (TP)	False Negative (FN)
False positive (FP)	True Negative (TN)

- TP : is a malware and also detected;
- FP : isn't a malware but was detected;
- TN: isn't a malware and it was recognized;
- FN: is a malware but it wasn't detected.

So the evaluation metrics are calculated in this way:

- Precision = TP / (TP + FP);
- Recall = TP / (TP + FN);
- False Positive rate = FP / (FP+TN);
- Accuracy = (TP + TN) / (TP + FN + TN + FP);
- F-measure = 2 x (Precision x Recall) / (Precision + Recall).

All decimal values will be represented in the percentual format.

A) **Datasets with random concentration < 25%**

	NAIVE-BAYES 1000	NAIVE-BAYES 2000	NAIVE-BAYES 3000	NAIVE-BAYES 4000	NAIVE-BAYES 5000	SVM-LINEAR 1000	SVM-LINEAR 2000	SVM-LINEAR 3000	SVM-LINEAR 4000	SVM-LINEAR 5000	SVM-POLYNOMIAL 1000	SVM-POLYNOMIAL 2000	SVM-POLYNOMIAL 3000	SVM-POLYNOMIAL 4000	SVM-POLYNOMIAL 5000	SVM-RBF 1000	SVM-RBF 2000	SVM-RBF 3000	SVM-RBF 4000	SVM-RBF 5000
Precision	40%	75%	73,58%	70,73%	72,00%	60,00%	81,25%	88,89%	91,93%	90,59%	NaN	100%	100%	100%	98,41%	NaN	100%	100%	100%	100%
Recall	6,06%	21,95%	31,45%	32,95%	41,09%	9,09%	15,85%	25,80%	32,39%	35,15%	0,00%	12,19%	20,97%	25,00%	28,31%	0,00%	12,19%	20,97%	23,29%	26,48%
F. P. rate	0,31%	0,31%	0,49%	0,63%	0,73%	0,20%	0,16%	0,13%	0,13%	0,17%	0,00%	0,00%	0,00%	0,00%	< 0,01%	0,00%	0,00%	0,00%	0,00%	0,00%
Accuracy	96,6%	96,5%	96,70%	96,45%	96,72%	96,80%	96,40%	96,80%	96,90%	97,00%	96,70%	96,40%	96,73%	96,70%	96,84%	96,70%	96,40%	96,73%	96,62%	96,78%
F-measure	10,52%	33,96%	44,07%	44,96%	52,32%	15,78%	26,53%	0,40%	47,90%	50,66%	NaN	21,74%	34,67%	40,00%	43,97%	NaN	21,74%	34,67%	37,79%	41,88%
TP	2	18	39	58	90	3	13	32	57	77	0	10	26	44	62	0	10	26	41	58
FN	31	64	85	118	129	30	69	92	119	142	33	72	98	132	157	33	72	98	135	161
FP	3	6	14	24	35	2	3	4	5	8	0	0	0	0	1	0	0	0	0	0
TN	964	1912	2862	3800	4746	965	1915	2872	3819	4773	967	1918	2876	3824	4780	967	1918	2876	3824	4781

B) **Datasets with concentration of 25%**

	NAIVE-BAYES 1000	NAIVE-BAYES 2000	NAIVE-BAYES 3000	NAIVE-BAYES 4000	NAIVE-BAYES 5000	SVM-LINEAR 1000	SVM-LINEAR 2000	SVM-LINEAR 3000	SVM-LINEAR 4000	SVM-LINEAR 5000	SVM-POLYNOMIAL 1000	SVM-POLYNOMIAL 2000	SVM-POLYNOMIAL 3000	SVM-POLYNOMIAL 4000	SVM-POLYNOMIAL 5000	SVM-RBF 1000	SVM-RBF 2000	SVM-RBF 3000	SVM-RBF 4000	SVM-RBF 5000
Precision	76,61%	77,37%	77,68%	77,43%	77,99%	93,24%	94,68%	94,99%	94,66%	95,95%	98,09%	98,77%	97,96%	98,18%	98,47%	100%	100%	99,41%	99,15%	99,51%
Recall	85,20%	89,60%	90,53%	89,90%	89,84%	55,20%	60,60%	65,73%	69,10%	72,08%	41,20%	48,40%	51,33%	54,00%	56,64%	37,60%	43,80%	45,20%	46,70%	49,12%
F. P. rate	8,67%	8,73%	8,66%	8,73%	8,45%	1,33%	1,13%	1,15%	1,30%	1,01%	0,26%	0,20%	0,35%	0,33%	0,29%	0,00%	0,00%	< 0,01%	0,13%	< 0,01%
Accuracy	89,80%	90,85%	91,13%	90,92%	91,12%	87,80%	89,30%	90,56%	91,30%	92,26%	85,10%	86,95%	87,57%	88,25%	88,94%	84,40%	85,95%	86,23%	86,57%	87,22%
F-measure	80,68%	83,04%	83,62%	83,20%	83,49%	69,34%	73,90%	77,69%	79,88%	82,32%	58,02%	64,97%	67,37%	69,68%	71,91%	54,65%	60,92%	62,14%	63,49%	65,77%
TP	213	448	679	899	1123	138	303	493	691	901	103	242	385	540	708	94	219	339	467	614
FN	37	52	71	101	127	112	197	257	309	349	147	258	365	460	542	156	281	411	533	636
FP	65	131	195	262	317	10	17	26	39	38	2	3	8	10	11	0	0	2	4	3
TN	685	1369	2055	2738	3433	740	1483	2224	2961	3712	748	1497	2242	2990	3739	750	1500	2248	2996	3747

C) **Datasets with concentration of 50%**

	NAIVE-BAYES 1000	NAIVE-BAYES 2000	NAIVE-BAYES 3000	NAIVE-BAYES 4000	NAIVE-BAYES 5000	SVM-LINEAR 1000	SVM-LINEAR 2000	SVM-LINEAR 3000	SVM-LINEAR 4000	SVM-LINEAR 5000	SVM-POLYNOMIAL 1000	SVM-POLYNOMIAL 2000	SVM-POLYNOMIAL 3000	SVM-POLYNOMIAL 4000	SVM-POLYNOMIAL 5000	SVM-RBF 1000	SVM-RBF 2000	SVM-RBF 3000	SVM-RBF 4000	SVM-RBF 5000
Precision	74,80%	76,37%	78,92%	79,00%	78,68%	95,01%	96,70%	96,67%	96,91%	96,67%	95,00%	96,99%	98,03%	99,06%	98,07%	99,54%	99,60%	99,75%	99,82%	99,86%
Recall	94,40%	95,00%	94,87%	94,25%	94,32%	68,60%	73,30%	77,40%	79,85%	82,48%	49,40%	54,70%	59,80%	63,00%	69,16%	43,00%	49,20%	53,33%	55,80%	58,84%
F. P. rate	31,80%	29,40%	25,33%	25,05%	25,56%	3,60%	2,50%	2,67%	2,55%	2,84%	2,60%	1,70%	1,20%	0,60%	1,36%	0,20%	0,20%	0,13%	0,10%	0,08%
Accuracy	81,30%	82,80%	84,77%	84,60%	84,38%	82,50%	85,40%	87,37%	88,65%	89,82%	73,40%	76,50%	79,30%	81,20%	83,90%	71,40%	74,50%	76,60%	77,85%	79,38%
F-measure	83,47%	84,67%	86,16%	85,96%	85,79%	79,67%	83,39%	85,97%	87,55%	89,01%	65,00%	69,95%	74,29%	77,02%	81,12%	60,06%	65,86%	69,50%	71,58%	74,05%
TP	472	950	1423	1885	2358	343	733	1161	1597	2062	247	547	897	1260	1729	215	492	800	1116	1471
FN	28	50	77	115	142	157	267	339	403	438	253	453	603	740	771	285	508	700	884	1029
FP	159	294	380	501	639	18	25	40	51	71	13	17	18	12	34	1	2	2	2	2
TN	341	706	1120	1499	1861	482	975	1460	1949	2429	487	983	1482	1988	2466	499	998	1498	1998	2498

D) **Datasets with concentration of 75%**

	NAIVE-BAYES 1000	NAIVE-BAYES 2000	NAIVE-BAYES 3000	NAIVE-BAYES 4000	NAIVE-BAYES 5000	SVM-LINEAR 1000	SVM-LINEAR 2000	SVM-LINEAR 3000	SVM-LINEAR 4000	SVM-LINEAR 5000	SVM-POLYNOMIAL 1000	SVM-POLYNOMIAL 2000	SVM-POLYNOMIAL 3000	SVM-POLYNOMIAL 4000	SVM-POLYNOMIAL 5000	SVM-RBF 1000	SVM-RBF 2000	SVM-RBF 3000	SVM-RBF 4000	SVM-RBF 5000
Precision	84,80%	85,93%	86,18%	86,99%	87,01%	95,93%	96,64%	96,87%	96,59%	97,43%	100%	99,89%	99,73%	99,37%	99,32%	77,88%	78,82%	79,19%	79,64%	79,57%
Recall	97,47%	96,87%	96,49%	96,03%	95,92%	78,67%	82,53%	84,95%	85,93%	87,04%	48,00%	60,67%	66,13%	67,97%	70,48%	100%	99,73%	99,78%	99,77%	99,68%
F. P. rate	52,40%	47,60%	46,40%	43,10%	42,96%	10,00%	8,60%	8,27%	9,10%	6,88%	0,00%	0,20%	0,53%	1,30%	1,44%	85,20%	80,40%	78,67%	76,50%	76,80%
Accuracy	85,00%	85,75%	85,77%	86,25%	86,20%	81,50%	84,75%	86,64%	87,18%	88,56%	61,00%	70,45%	74,47%	75,65%	77,50%	78,70%	79,70%	80,17%	80,70%	80,56%
F-measure	90,69%	91,07%	91,05%	91,29%	91,25%	86,45%	89,03%	90,52%	90,95%	91,94%	64,86%	75,49%	79,53%	80,72%	82,45%	87,57%	88,05%	88,30%	88,58%	88,49%
TP	731	1453	2171	2881	3597	590	1238	1919	2578	3264	360	910	1488	2039	2643	750	1496	2245	2993	3738
FN	19	47	79	119	153	160	262	340	422	486	390	590	762	961	1107	0	4	5	7	12
FP	131	238	348	431	537	25	43	62	91	86	0	1	4	13	18	213	402	590	765	960
TN	119	262	402	569	713	225	457	688	909	1164	250	499	746	987	1232	37	98	160	235	290

IV. **Conclusions**

From the tests carried out, it emerges that every algorithm, in all the possible concentration scenarios, shows a substantial improvement in evaluation statistics as samples grow in the dataset.

Naive-Bayes classifier reported high values of Recall and F-measure at the growing of the malware concentration, that starting from the 25% tend to oscillate with a few percentages, but remains stable.
Until the last measurements in concentration, Precision values tend to be the worst ones in oppositions to the other types of algorithms analyzed.
False positive rate appears to grow linearly in direct dependence to the malware percentage in the dataset.

SVM classifier with a linear kernel reported in every test good values for Precision and Accuracy, that in most cases exceeded their counterparts in the Naive-Bayes tests.
While malware concentration rises, also False positive rate tends to grow up, but with more and more larger datasets its value tends to decrease, provoking also improvements in Recall and F-measure estimations.

SVM classifiers with polynomial and radial basis function kernels registered very similar results according to the evaluation metrics until the 75% concentration tests.
Precision values remains in most cases around 100%-99%, denoting an high rate recognition of not-malware samples (also with very low False positive rate).
Recall, Accuracy and F-Measure values grow up with the number of samples, but they remain lower than those found in the two previous types of tests; in most cases, more in the radial basis function one than the other.
With a low percentage of malware in not too big datasets, both classificators have been seen to be uncapable to correclty classify samples in both two categories, as resulted with less than 25% of malwares and a dataset of 1000 samples.

In the tests’ scenario with the most concentration, the SVM classifier with radial basis function kernel reported a completely different behaviour than the polynomial one, that roughly mantained the same trend.
Despite a loss in Precision, Recall and F-measure have a big increase in their values.
Also the False positive rate starts very high in the first test, but tends to decrease with the growing of the dataset, denoting an high incidence in the recognition of secure samples as malwares.
In final seat, the number of false negative samples, so the number of true malwares not recognized, remains really low, denoting a very high possibility of true malware recognition.