Electronic Research Archive of Blekinge Institute of Technology
http://www.bth.se/fou/

This is an author produced version of a journal paper. The paper has been peer-reviewed but may not include the final publisher proof-corrections or journal pagination.

Citation for the published Journal paper:
Title:
The Normalised Compression Distance as a file fragment classifier

Author:
Stefan Axelsson

Journal:
Digital Investigation

Year:
2010
Vol.
7
Issue:
Suppl 1
Pagination:
S24-S31
URL/DOI to the paper:
doi:10.1016/j.diin.2010.05.004

Access to the published version may require subscription.
Published with permission from:

THE NORMALIZED COMPRESSION DISTANCE AS A FILE FRAGMENT CLASSIFIER

Stefan Axelsson

Blekinge Institute of Technology
stefan.axelsson@bth.se

ABSTRACT

We have applied the generalised and universal distance measure NCD—Normalised Compression Distance—to the problem of determining the type of file fragments. To enable later comparison of the results, the algorithm was applied to fragments of a publicly available corpus of files. The NCD algorithm in conjunction with the k-nearest-neighbour (k ranging from one to ten) as the classification algorithm was applied to a random selection of circa 3000 512-byte file fragments from 28 different file types. This procedure was then repeated ten times. While the overall accuracy of the n-valued classification only improved the prior probability from approximately 3.5% to circa 32%–36%, the classifier reached accuracies of circa 70% for the most successful file types.

A prototype of a file fragment classifier was then developed and evaluated on new set of data (from the same corpus). Some circa 3000 fragments were selected at random and the experiment repeated five times. This prototype classifier remained successful at classifying individual file types with accuracies ranging from only slightly lower than 70% for the best class, down to similar accuracies as in the prior experiment.

1. INTRODUCTION

When performing computer forensics the investigator often finds herself with a collection of file fragments from slack space on disks, usb-keys etc. Sometimes enough meta data (file links, headers etc.) survives to make reconstruction easy, but often enough what the investigator is left with is a random collection of file fragments, that need to be put together to form (partially complete) files. Knowing or having an indication of the types of the file fragments (i.e. picture, executable, text etc.) greatly aids in the reconstruction process, as the available resources can be put to use where they best are needed.

To that end we have investigated applying a fairly recent distance measure—normalised compression distance—to the problem of classifying file fragments. This field of research is not new, but it is unfortunately difficult to compare the results we obtain with previous results, since the data was not available for comparison. In order to further research in the field we have performed our experiment on the publicly available corpus by Garfinkel et. al. [1] in the hope that future work will be performed in a way which makes comparison possible and to foster a spirit of friendly competition.

2. NORMALISED COMPRESSION DISTANCE (NCD)

The problem as we have framed it is one of supervised machine learning. Of the several available algorithms we have chosen to base our classifier on a fairly recent algorithm that computes distances between arbitrary data vectors: *Normalised Compression Distance* (*NCD*) [2] as it is generally applicable [3], parameter free [4], noise resistant [5] and demonstrated theoretically optimal [6]. The NCD is an approximation to the uncomputable *Normalised Information Distance*, that is based on the notion of Kolmogorov Complexity.

NCD is based on the idea that by using a compression algorithm on data vectors (in whatever shape or form these may come) both individually and concatenated, we will receive a measure of how distant they are. The better the combination of the two vectors compress, compared to how the individual vectors compress on their own (normalised to remove differences in length between the set of all vectors), the more similar they are. More formally, NCD is a metric:

$$NCD(x,y) = \frac{C(x,y) - min(C(x), C(y))}{max(C(x), C(y))}$$

where $C(x)$ is the compressed length of x and $C(x,y)$ the compressed length of x concatenated with y.

In order to apply this metric as a supervised learner one selects features of the input data to train on and then calculates the distances from the features of the training instances to the features of the instances under classification. Another advantage of the NCD is that anything that can be put through the compression algorithm can in theory be used as a vector. [7] That means that the NCD is parameter free and resistant to errors in feature selection (as one does not *have* to do any of the latter). Not having to make any feature selection is a substantial advantage, as there is an almost infinite number of ways to select features, and these have then to be identified, selected and evaluated. The latter in particular is often a substantial task.

When actually using NCD for classification, a classification algorithm has to be used. We have chosen distance to the closest examples in the set e.g. a fragment that is closest in distance to several particular instances in e.g. the *exe* set is classified as *exe*, and so on. Thus the classification algorithm proper is *k-nearest-neighbour* for various values of *k*. The *k* nearest feature vectors are selected and the class is assigned according to a majority vote. For example, with $k = 10$ if three of the closest feature vectors were of type *zip* and seven were of type *exe*, then the feature vector under classification is assigned to the class *exe*, even though the closest example might have been a *zip* feature vector.

3. THE FILE FRAGMENT CORPUS

Previous research has, with one exception [8], exclusively used private data sets. Using a private data set has many advantages, the chief reason being that these are quickly and simply put together e.g. by downloading "random" files of the Internet. However, this most probably means that the files cannot be redistributed to other researchers as that would violate copyright law. To counteract the resulting problem of the research not being reproducible by other researchers, and promoting friendly competition between approaches in the field, we chose instead to base our research on the freely available corpus of forensics research data by Garfinkel et.al. [1]. The corpus is large and contains several different parts. We chose to base our research on the file corpus described in section 4 of [1]. That corpus contains 1 million files of different file types. Of the available file types we chose to concentrate on an easily identifiable (based on filename endings) large subset consisting of the following 28 file types:, *pdf*, *html*, *jpg*, *text*, *doc*, *xls*, *ppt*, *xml*, *gif*, *ps*, *csv*, *gz*, *eps*, *png*, *swf*, *pps*, *sql*, *java*, *pptx*, *docx*, *ttf*, *js*, *pub*, *bmp*, *xbm*, *xlsx*, *jar*, and *zip*.

4. THE EXPERIMENT

Our experiment follows the example put forth in [8] closely as we wish to make our results comparable to that work.

We decided to perform our experiment on 512 byte fragments from the corpus described above. The reason for choosing 512 byte fragments was that this is currently the smallest available sector size on magnetic media and hence one does not typically have to deal with fragments smaller than this. If one has knowledge of the file system that was in use, then one can profitably chose a larger fragment size. However, we would argue that using the smaller size when doing research is a more conservative choice as larger fragments allows for the machine learning algorithm to be exposed to more structure per fragment on which to base its decision.

Other research has selected features by doing processing on the fragments under study (such as calculating byte frequency distributions etc.) but since an advantage of the NCD algorithm is that it can process most available data without preprocessing, instead the entire fragment is fed to the compression algorithm. The *bzip2* [9] algorithm was chosen as it can comfortably handle the lengths of strings (2×512) that we are exposing it to, without much overhead. For more discussion of what to look for in a compression algorithm that is to be used in NCD see Cebrian et.al. [7]

Ideally we would like to have performed one big experiment on several tens of thousands of blocks chosen from the corpus and then done our evaluation of the results by ten-fold cross validation [10]. However, as we have to calculate the distance between all pairs of blocks, and hence the runtime of the classification algorithm is quadratic in nature, that was unfortunately impossible. Instead, a smaller subset of blocks, ca 3000 per trial, was chosen and the experiment repeated ten times.

In order to achieve comparable results, during each of the ten runs, we selected ten files at random of each type and then selected 14 blocks randomly (using GNU sort's '-R', random, argument) from each file, except where the file was not sufficiently long (i.e. less than $512 \times 14 = 7168$ bytes).

The selection of files was done in such a way that all files in the corpus were equally likely to be selected in any one run of the experiment (i.e. the file were placed back in the "urn"). Once a file set had been selected for an experiment the fragments were selected at random uniquely, hence no fragment was selected twice. Thus the entire experiment was conducted on ca 32000 fragments (divided onto ten separate trials). Table 1 details the types and number of fragments that were selected.

In labelling a file as an actual type of files for the experiment we have relied mostly on the file name endings. This method is far from fool proof though. While we, for example, could find only one or two instances of a file with the ending *.eps* that was not actually an encapsulated postscript file, other types, notably the *.pub* ending, was less successful with several examples of e.g. addresses for persons, etc. We have yet to do an actual full classification by hand which would answer once and for all what the actual type of the files would be. Until that is done, we advise some caution in interpreting the results above as a certain (small) percentage of the files assumed to be e.g. PostScript files could in fact be of type e.g. *html*. We decided to still include the problematic classes, i.e. *pub* and *text* as, even though we may not expect perfect hitrates on these file types, they still provide a target of opportunity for the classifier to misclassify *other* file types as either *pub* or *text*, making the experiment more conservative.

The NCD algorithm was run once for each of the ten trial sets and the distance between each pair of fragments was computed (actually since the distance in this case does not necessarily commute, due to imperfections in the compression algorithms, both the distances $a - b$ and $b - a$ were computed, when classifying a only the $a - b$ distance was actually used). Using a dual core Opteron 2214HE with 4 GBytes of memory, computing the complete distance matrix took ap-

Ending	Type	Fragments
bmp	Bitmap picture	1131
csv	Comma separated values	1055
doc	Microsoft Word (various vs)	1319
docx	Microsoft Word (xml format)	1302
eps	Encapsulated Postscript	1563
gif	Graphics interchange format	1281
gz	GNU zip compression	1185
html	Hypertext markup language	1171
jar	Java archive	765
java	Java source code	1151
jpg	Joint Photographic Experts Group image	1143
js	Javascript source code	994
pdf	Portable document format	1286
png	Portable network graphics	1078
pps	Microsoft Powerpoint Show	1317
ppt	Microsoft Powerpoint	1276
pptx	Microsoft Powerpoint (xml format)	1190
ps	PostScript	819
pub	ssh public key files	643
sql	Sql database scripts and dumps	846
swf	Shockwave flash	1297
text	Text files (DOS/Windows)	1030
ttf	True type font	961
xbm	X Bitmap	694
xls	Microsoft Excel (various vs)	1290
xlsx	Microsoft Excel (xml format)	1379
xml	Extensible markup language	1233
zip	Data compression and archiving	1142
total		31541

Table 1. Experimental data

proximately 40 minutes per trial, for a total of just over six hours for all trials.

In presenting the results of the classification we will start with the overall results and then present the data in increasing detail.

First, the overall average hit rate (*accuracy*) of the *k-nearest-neighbour* classification for different values of k are presented in table 2. The average is computed over the ten trials. Note that we treat the classification as an n-valued problem, i.e. we do not compare pairs of file types as others have done, but instead we classify each file fragment into one of the available 28 file type classes. This should realistically affect our hit rate negatively as there are now many more choices for misclassifying a file fragment as opposed to a straight yes/no classification. In order to not commit the sin of including our "training" data in the final results, we exclude all file fragments from the same file as the fragment under study from our classification.

As we can see from table 2, the results are unremarkable. However, note that as there are 28 different possible classes, a 34% hit rate does not mean that a coin toss would have done just as well. A completely random classification process

k	Average hit rate (%)
1	36.43
2	34.95
3	34.96
4	34.31
5	34.17
6	33.65
7	33.50
8	33.06
9	33.00
10	32.86

Table 2. k-nearest-neighbour overall results

would instead result in a $\approx 3.5\%$ hit rate ($1/28$ to be precise). It is interesting to note that the value of k has little effect on the result of the classification. The single nearest neighbour does as well as any other small k. Too large a value of k would lead to a drop off in hit rate as the number of file fragments of the same (correct) type is limited and less than the total number of file fragments.

However, if we break down the result per class of fragment, the picture changes. In table 3 the average hit rate and standard deviation (in percentage points) per fragment type for $1 \leq k \leq 10$ is presented. Note that as for the overall results the specific value of k changes the accuracy little.

We calculated the statistical significance of the above results when put against the purely random classifier (that in effect would roll a 28-sided die). This is not entirely correct as the total number of fragments per file type varies slightly, a more clever "random" classifier would take that into effect and adjust the probability ever so slightly for a class, to be the number of fragments for that class divided by the total number of fragments. However, as the lion's share of the results above are highly statistically significant and the difference is small, we have decided to ignore this refinement. In summary, all the above results are statistically significant at the 1% level ($P < 0.01$) with the exception of the result for *pptx*, that is significant at the 5% level ($P < 0.05$) and *pps*, *gz*, and *png* that are not statistically significantly better than the equal likelihood classifier. However, that e.g. the *gif* classifier is statistically better than random chance is not in itself much of an endorsement, as the accuracy is still quite low (even though it is twice as good as pure chance). As a comparison against random chance is setting the bar very low, we will not mention statistical significance further.

As we can see from table 3, there is a notable difference between different types of file fragments when it comes to how successful we are at classifying them. In order to make the data in table 3 more clear we have included a bar chart that lists the average of the average accuracies over all k-values in figure 1. Unfortunately for the further analysis no clear groups can be easily identified. However, it becomes clear that the classifier is most successful for the *eps*, *java*, *csv*, and

Class	1	σ	2	σ	3	σ	4	σ	5	σ	6	σ	7	σ	8	σ	9	σ	10	σ
eps	69	(8.1)	64	(13)	70	(10)	67	(13)	70	(10)	69	(12)	71	(9.1)	68	(12)	70	(11)	70	(12)
java	68	(12)	62	(12)	66	(12)	67	(12)	68	(12)	67	(13)	66	(13)	67	(13)	67	(13)	67	(13)
csv	67	(12)	61	(14)	63	(14)	64	(12)	65	(13)	65	(13)	66	(13)	67	(12)	66	(13)	67	(12)
ttf	69	(12)	73	(12)	69	(11)	66	(11)	62	(10)	62	(9.0)	59	(10)	60	(8.3)	58	(9.0)	58	(7.5)
xml	58	(15)	56	(16)	58	(15)	58	(15)	59	(15)	58	(14)	58	(14)	58	(15)	58	(15)	57	(15)
js	62	(12)	59	(13)	58	(15)	56	(16)	58	(17)	57	(18)	57	(18)	55	(18)	54	(19)	52	(22)
bmp	51	(12)	39	(12)	48	(9.2)	53	(11)	54	(8.9)	56	(9.0)	57	(10)	56	(11)	57	(10)	58	(10)
doc	48	(10)	56	(13)	54	(11)	53	(12)	50	(10)	52	(11)	49	(11)	50	(13)	48	(14)	52	(15)
xls	51	(13)	46	(10)	48	(10)	47	(12)	50	(13)	49	(13)	51	(14)	50	(13)	51	(14)	50	(13)
xbm	61	(14)	51	(17)	51	(17)	50	(18)	50	(18)	44	(18)	45	(19)	40	(19)	40	(20)	39	(19)
docx	47	(14)	44	(13)	35	(15)	45	(18)	49	(20)	49	(21)	50	(22)	50	(22)	51	(23)	51	(22)
jar	59	(8.9)	55	(9.2)	47	(11)	49	(13)	48	(13)	44	(15)	44	(14)	40	(12)	39	(11)	39	(12)
zip	61	(17)	44	(24)	51	(21)	45	(22)	44	(21)	43	(21)	43	(22)	42	(23)	41	(22)	42	(23)
ps	40	(14)	43	(14)	39	(14)	39	(15)	37	(15)	38	(14)	37	(14)	37	(14)	36	(14)	36	(15)
sql	40	(12)	38	(11)	37	(12)	37	(12)	37	(12)	37	(14)	36	(14)	36	(14)	37	(15)	36	(15)
pub	52	(26)	43	(22)	36	(22)	33	(22)	31	(21)	29	(19)	27	(20)	24	(19)	22	(18)	21	(19)
html	25	(10)	36	(10)	35	(10)	34	(8)	31	(9)	31	(9)	30	(10)	30	(9)	29	(10)	28	(10)
xlsx	43	(13)	20	(11)	30	(14)	31	(17)	32	(16)	31	(18)	29	(17)	29	(19)	30	(20)	29	(20)
text	19	(10)	27	(12)	25	(13)	23	(12)	23	(12)	23	(12)	22	(12)	23	(12)	23	(12)	24	(13)
pdf	23	(9)	26	(10)	25	(10)	24	(10)	24	(11)	23	(10)	22	(11)	21	(11)	21	(10)	21	(10)
ppt	14	(7.7)	14	(9.5)	11	(7.8)	13	(9.2)	13	(8.1)	12	(9.4)	14	(7.8)	15	(8.5)	14	(8.7)	15	(8.6)
swf	11	(16)	11	(15)	11	(14)	10	(15)	10	(16)	12	(15)	12	(14)	12	(13)	10	(12)	10	(11)
jpg	7.2	(6.5)	12	(10)	11	(9.4)	9.2	(7.7)	7.8	(6.8)	6.3	(6.4)	5.4	(6.0)	5.7	(5.7)	5.4	(5.4)	5.2	(4.0)
gif	7.4	(5.7)	14	(10)	12	(7.7)	8.4	(5.2)	6.2	(4.5)	5.3	(4.6)	5.5	(4.4)	5.4	(3.6)	5.0	(3.7)	4.9	(3.8)
pptx	8.9	(8.9)	12	(10)	7.9	(4.3)	5.8	(5.7)	6.0	(7.3)	6.2	(7.5)	5.9	(5.9)	6.1	(6.1)	5.8	(5.3)	4.7	(5.0)
pps	6.8	(4.0)	4.1	(2.9)	5.4	(4.4)	4.7	(3.8)	5.1	(5.0)	5.1	(4.8)	5.7	(5.3)	6.3	(5.2)	7.0	(5.2)	7.8	(5.1)
gz	4.4	(3.8)	8.7	(6.3)	8.2	(6.0)	5.2	(3.8)	4.0	(5.6)	4.8	(6.2)	5.0	(6.0)	5.7	(6.1)	5.8	(4.9)	4.9	(3.8)
png	5.5	(4.0)	11	(8.1)	8.0	(7.3)	4.3	(4.7)	4.2	(3.4)	3.6	(3.1)	3.6	(3.2)	4.2	(4.0)	3.9	(3.6)	3.8	(4.7)

Table 3. Average accuracy per fragment type (Standard deviation) for different k (both % rounded)

ttf file types with accuracies ranging from 68%–63% averaged over all values of k. Then the *xml*, *js*, *bmp*, *doc*, *xls*, *xbm*, *docx*, *jar* and *zip*-classes range until there is a small drop. (The *zip*-class averaging 46% accuracy.) We see the next step at *xls* (at 30%) and then surprisingly the *text* class at 23% and the *pdf*-class likewise at 23%. Then we have the low performers; *ppt*, *swf*, *jpg*, *gif*, *pptx*, *pps*, *gz* and *png*, which all score so low as to be scarcely better than random classification.

However, to not inadvertently lure the reader to accept the average accuracy (over the ten trials) as the final word on the matter; for completeness sake we also list the minimum (worst) accuracies of the ten trials in table 4. Listing the minimum as opposed to, for example, the lower quartile, is of course more conservative. We see from the table that considering the instances where the classifier performs the worst, we can hardly rely on it for more than a third or so of the top scoring file types. If we study the table 4 more closely we see that the rough order of success does not change much though a few classes do worse or better than they do overall. We have not examined this effect in any detail.

So far we have only discussed accuracy in terms of hits and misses. A fuller picture develops if we also consider the actual classes that were predicted by the classifier. As space doesn't permit us to present the complete confusion matrix we will focus on the cases where the classifier did not do well, as that tells us which types are most likely to be mistaken

Class	min
java	45.2
ttf	38.3
eps	37.9
csv	33.3
doc	25.3
xml	25.2
xls	23.4
jar	21.3
bmp	16.7
html	14.2
js	12
sql	11.1
xbm	11
pdf	10.7
xlsx	5.7
zip	5.4
docx	5.3
pub	4
text	2.5
ps	2.4
gif	0.7
gz	0.7
jpg	0.7
png	0.7
ppt	0.7
pptx	0.7
swf	0.7
pps	0.6

Table 4. Minimum accuracy per fragment type (% rounded)

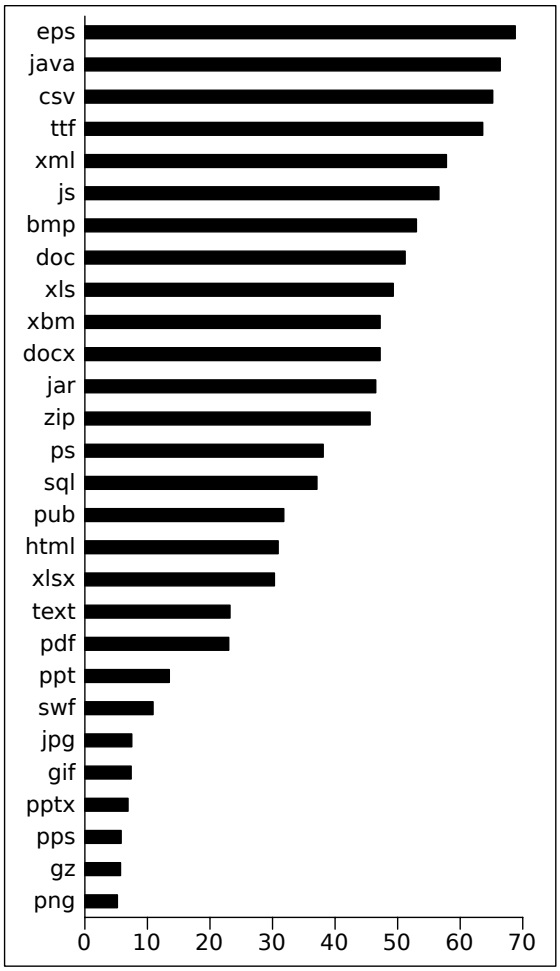

Fig. 1. Average of the average accuracy per class (%).

for other types of files. We will present the data for the k-value where the classifier did the best to avoid artificially low figures due to pure chance. Table 5 lists the results with one column per poor performer. A lack of space precludes us from also listing the percentages, but as luck would have it, the column totals are all fairly similar which makes comparison between them easier.

A few interesting patterns emerge in the table. The most striking is perhaps that as far as misclassification is concerned, every file looks like a *docx* (and to a lesser extent an *xlsx*) file. However, the *docx* (and *xlsx*) format is basically a *zip*-file (i.e. it is compressed). We would tend to think that a compressed file would look more or less random from the perspective of our detector, and that is probably why it matches other formats that are compressed, such as *pptx*, *gif*, *png* etc.

Since $k = 1$ did so well as a classifier its interesting to see which single specific fragments best describe their class (i.e. leads to the most correct classifications). In order to present this data we have chosen a rather conservative method; we count the nearest fragments as long as they are of the correct type. As soon as a fragment is of the wrong type, we stop the counting and emit the result. To not pollute the result with files that are self similar, we skip fragments from the same file and do not count them, instead we keep scanning for the next closest fragment as stopping would be a rather unreasonable. Table 6 lists how many fragments the five most successful fragments per class predicts using the method above. (It should be remembered that the method used here is more conservative than *k-nearest-neighbour* that we used in the beginning of the paper).

Thus, reading the table, we see e.g. that the best *csv* fragment correctly predicted the class of 54 other *csv* fragments using our rather restrictive definition for *predicted*. A lack of space precludes us from discuss the results in any detail, but a few examples suffice to demonstrate the idea. Five unique *csv* fragments each classify between 54–47 fragments each. While this is in itself not a great percentage of the total number of *csv* fragments, it does serve to give an indication of how successful a single fragment can be at correctly classifying a file type. Furthermore it is of interest when we develop a more practical detector in the next section.

Of course, it would be interesting to learn to what extent two different fragments classify the same or different fragments (in the latter case a better detector could be made using combining two or more fragments in the classifier, in the former this would lead to scant gain). A lack of space precludes us from taking this line of investigation further. It would however be interesting to delve further into exactly why these fragments tend to correctly classify its brethren.

5. A PROTOTYPE DETECTOR

In order to test the method further, we have also developed a prototype for a more practical detector that can be run on file fragments. The previous experiment pitted all (already classified) blocks against each other. This is of course not possible in a real scenario, as one does not then have access to already labelled data.

A reasonable amount of "good" fragments, i.e. fragments that were successful at classifying other fragments were selected as a reference to be used to classify unknown blocks against. In order to create this data set, the twenty most successful fragments (measured in the same way as in table 6) were selected as this gave a reasonable total of 560 fragments. The distance between these 560 fragments and the fragments under study was then calculated and the k-nearest-neighbour algorithm used for classification. Since the reference number of fragments was fixed at 560, the runtime of the NCD algorithm proper is no longer quadratic in nature, but rather linear in the number of fragments to classify. On a computer about half as fast as the Opteron detailed above, classifying approximately 3300 fragments (each against the 560 reference fragments) took on the order of 10 minutes. (No doubt, there is great room for improvement/optimisation by writing software

Class k-value	pps $k = 10$	gz $k = 2$	png $k = 2$	swf $k = 7$	jpg $k = 2$	pptx $k = 2$	gif $k = 2$	ppt $k = 10$
bmp	27	3	3	5	5	8	10	35
csv	1							
doc	74	2	2	50	5	33	32	117
docx	495	470	470	464	381	400	440	435
eps	8			2	2	9	3	4
gif	44	128	128	63	136	122		23
gz	42			26	97	87	85	40
html	7			12				4
jar	2							2
java								4
jpg	46	90	90	50		107	60	30
js		2	2	4				
pdf	20	25	25	9	44	22	26	6
png	36	122	122	66	134	125	111	31
pps		31	31	15	24	24	20	66
ppt	93	10	10	6	15	18	10	
pptx	44	138	138	75	117		118	20
ps	1							
pub	1			2		1	1	
sql				1			1	
swf	14	36	36		40	36	22	9
text	3					1	3	2
ttf		7	7					3
xbm		1	1					
xls	77	23	23	25	21	24	34	61
xlsx	339	117	117	309	119	147	130	223
xml	1			1	1		1	5
zip		3	3	5				5
Sum	1375	1208	1067	1190	1141	1164	1107	1125

Table 5. Predicted class (rows) for the low performing fragment types (number of fragments), total for all trials

Class	Best	2:nd	3:d	4:th	5:th
csv	54	52	51	48	47
ttf	49	46	46	45	45
js	42	39	38	36	36
java	39	38	38	36	33
eps	37	37	34	34	32
xml	30	26	24	23	23
bmp	30	28	28	27	25
sql	29	29	28	26	24
ps	28	28	25	25	22
xls	24	20	18	17	17
pdf	20	18	18	16	16
jar	20	19	18	18	18
xbm	19	19	18	18	17
doc	19	18	17	14	14
text	18	13	12	11	11
zip	15	14	13	13	13
xlsx	14	14	13	13	12
swf	11	10	9	9	8
pub	11	9	8	7	7
docx	10	9	9	9	8
ppt	9	9	8	7	6
html	9	9	8	8	8
pps	7	7	6	6	5
pptx	5	4	4	4	4
jpg	4	3	2	2	2
png	3	2	2	2	2
gif	3	2	2	2	2
gz	2	1	1	1	1

Table 6. Fragments (class) most successful at conservatively predicting the type of other fragments

specialised for the task.)

Even though the proposed method does not perform well on all file types, we still include examples of the file types that perform poorly overall in the reference data set. The reason is that without these examples the detector could never emit those classes as a result of the classification procedure. This would most likely lead to a detector displaying serious signs of home blindness, i.e. it would misclassify other fragments with great certainty (however misguided). So in order to achieve more robust detector they were left in. We cannot quantify the effect that leaving the low performers out would have as we have not performed the alternative experiment (other than the obvious, if no examples at all were left in, that would render the detector completely blind to those file types).

The detector detailed above was tested on a new data set from the corpus. In much the same way as above, ten randomly selected files of each type were downloaded and likewise fifteen blocks were selected at random from each file (where the file was long enough). This procedure was repeated five times.

The detector was then run on each of the five data sets and the results are reported in table 7. The table lists the average percentage of fragments that were correctly detected, with the standard deviation in parenthesis.

We see that even though we validate the detector on a new data set the detector does quite well for many file types. (Though we admit that selecting the files from the same corpus does not ensure that the validating data set is independent of the training data set to quite the degree that would be desirable.) The detector performs roughly on par with our earlier experiments. When reading the accuracy data in table 7 it should again be kept in mind that the prior probability of any one class is on the order of 3.5%.

We would like to stress that this is still a prototype, and much work remains to be done before a similar tool could be put to practical use.

6. RELATED WORK

The work that is most similar to this is no doubt [8]. That work is unique amongst prior work in that it is also based on a freely available corpus of experimental data. The results on that data set, using much the same methods as here are somewhat different. For example, the NCD algorithm applied in that experiment did the very best for *svg*, *xls*, *html* and *text* type fragments. It also did well for *elf*, *png*, *gif* and *pdf*, while the performance for *exe*, *jpg*, *dll*, *mov*, *mp3* and *zip* is poor. This should be contrasted with the results presented here, where *html* is on the lower half of the tested file types as is *pdf*, *gif* and *png* (which is indeed among the worst performers here). However, the corpus used in [8] is an order of magnitude smaller than the one used here, and furthermore only one experimental run (as compared to ten in this experiment) was performed. Hence we have a higher degree of confidence in the results reported in the present paper.

Karresand et.al. [11, 12] introduced the idea of using the data of the file itself for classifying file fragments. In doing so they focused exclusively on developing the *Oscar* method for identifying *jpg* file fragments (that used special information about the structure of *jpg* files). They report a detection rate of 99.2% (and no false positives). As their approach in practice (though not necessarily in principle) is limited to only one fragment type it is difficult to compare their results with ours, as we address the more complex n-valued problem.

Veenman [13] also addressed the n-valued problem for $n = 11$ to a data set of about 450Mb of data using statistical methods (including one based on Kolmogorov complexity as is ours). He uses fragments of size 4096, i.e. eight times as large as we do. He reports an overall accuracy of 45%, which is only slightly larger than we do. He then develops a set of two class (cross type) classifiers that he reports better results with (as these are reported graphically it is difficult to summarise them here and also to compare them to our results).

More recently Calhoun and Coles [14] develop and test more than twenty different statistical methods for identifying file fragments. As the simplest methods they report on (ASCII frequency and entropy measurements) in their words, can correctly tell apart obviously different file formats such as *jpg*, *xls* and *txt*. They report results on using pairs of exper-

Class	1	σ	2	σ	3	σ	4	σ	5	σ	6	σ	7	σ	8	σ	9	σ	10	σ
pub	67	(6)	59	(11)	50	(8)	57	(9)	48	(7)	48	(8)	47	(7)	51	(8)	46	(7)	49	(8)
eps	61	(6)	61	(9)	61	(8)	63	(9)	63	(9)	63	(10)	61	(8)	61	(9)	59	(7)	58	(9)
csv	59	(10)	55	(8)	59	(11)	59	(10)	61	(11)	62	(10)	62	(9)	63	(10)	63	(9)	63	(9)
java	53	(12)	52	(14)	50	(12)	52	(13)	52	(12)	53	(13)	53	(13)	53	(13)	53	(13)	52	(14)
xml	50	(8)	51	(11)	51	(9)	50	(9)	49	(8)	49	(7)	49	(6)	50	(7)	50	(7)	50	(5)
ttf	48	(16)	49	(16)	49	(16)	49	(16)	49	(16)	49	(16)	49	(16)	49	(16)	49	(16)	49	(16)
bmp	46	(10)	38	(10)	45	(11)	47	(10)	47	(12)	49	(12)	50	(12)	50	(11)	51	(12)	53	(12)
xlsx	43	(5)	22	(7)	33	(7)	28	(9)	24	(9)	20	(9)	15	(10)	13	(10)	12	(9)	12	(9)
sql	36	(20)	35	(20)	32	(20)	35	(21)	33	(20)	35	(21)	36	(20)	35	(20)	35	(18)	36	(18)
zip	35	(2)	26	(3)	35	(2)	39	(4)	34	(4)	36	(5)	36	(5)	39	(6)	45	(5)	47	(6)
js	35	(7)	35	(7)	35	(7)	36	(6)	37	(7)	38	(7)	37	(7)	38	(6)	37	(7)	38	(7)
xls	34	(7)	36	(4)	35	(4)	35	(4)	36	(5)	36	(4)	38	(5)	40	(5)	53	(8)	51	(8)
docx	31	(5)	39	(5)	26	(6)	42	(6)	52	(8)	50	(5)	43	(3)	43	(4)	45	(3)	49	(3)
text	29	(5)	27	(7)	26	(7)	23	(6)	23	(7)	22	(7)	22	(8)	23	(8)	22	(9)	22	(8)
html	28	(7)	39	(9)	36	(8)	33	(8)	31	(9)	31	(8)	30	(9)	30	(8)	29	(8)	29	(9)
jar	23	(13)	23	(13)	24	(12)	24	(12)	24	(12)	24	(12)	30	(15)	30	(15)	30	(15)	30	(15)
doc	21	(3)	27	(3)	30	(3)	29	(2)	30	(2)	32	(2)	34	(2)	34	(2)	35	(2)	36	(2)
pdf	19	(6)	22	(6)	24	(7)	25	(6)	23	(7)	21	(7)	22	(6)	23	(6)	23	(6)	23	(5)
xbm	18	(10)	19	(8)	17	(11)	18	(11)	18	(11)	18	(11)	18	(11)	18	(11)	18	(11)	18	(11)
png	18	(3)	24	(3)	14	(3)	4	(0)	6	(1)	10	(1)	10	(2)	8	(2)	9	(1)	9	(2)
swf	14	(8)	10	(6)	9	(6)	10	(7)	10	(7)	9	(5)	10	(5)	10	(6)	10	(6)	10	(6)
ppt	10	(2)	12	(5)	10	(3)	11	(3)	10	(4)	11	(4)	9	(3)	10	(4)	9	(3)	10	(5)
gz	7	(2)	15	(4)	18	(4)	13	(0)	13	(3)	19	(2)	24	(3)	24	(4)	24	(1)	22	(3)
gif	7	(3)	12	(3)	13	(4)	11	(5)	9	(3)	11	(3)	13	(1)	15	(2)	13	(1)	13	(1)
pps	6	(2)	3	(1)	4	(2)	3	(2)	3	(2)	2	(2)	4	(2)	2	(2)	4	(2)	3	(2)
jpg	6	(2)	10	(2)	14	(3)	13	(2)	9	(1)	6	(1)	7	(2)	9	(2)	10	(3)	9	(2)
ps	5	(3)	7	(5)	8	(5)	6	(4)	5	(4)	4	(3)	2	(2)	8	(0)	7	(0)	5	(0)
pptx	5	(4)	5	(4)	5	(4)	4	(3)	4	(3)	5	(4)	5	(3)	5	(3)	5	(4)	6	(5)

Table 7. Average accuracy per fragment type (Standard deviation) for different k (both % rounded) for the prototype detector

iments on *jpg*, *bmp*, *gif* and *pdf*. Their best methods correctly classify on the order of 80%–90% of fragments in the pairwise comparison. Again, as we address the n-valued problem directly a comparison is difficult, but we manage not too much worse accuracy for our best results; though not for the same file types, which is noteworthy.

Unfortunately, for all the latter of these results, differences in methods, fragment size, data sets etc., make a more useful comparison difficult. Taking that caveat into consideration we see no *obvious* sign that the method presented here would not compete successfully against the other proposed methods, quite the contrary. Since the differences are as great as they are, however, too much should not be read into the preceding statement. Further investigation is necessary.

7. CONCLUSION AND FUTURE WORK

Even though our method is rather straightforward (simplistic even) and easily implemented, we still manage to correctly classify quite a few file types with a worthwhile degree of accuracy. The work as it stands suffers from a few limitations though; the most notable being the use of files from only one source (i.e. publicly available files from the US government). Also one has to ask how realistic the examples of files are compared to what an investigator would see in the field. A problem here is that we cannot use files the contents of which would make possession itself illegal. Also, some types of files should probably be divided into subclasses, such as executable files one would expect to find on a computer of the same type as the one under investigation and others that one would not (e.g. Microsoft office versus viruses, worms etc. though finding the latter on the average computer would of course not be very surprising...)

More formally; this paper makes the following contributions: We have demonstrated that we can correctly classify file fragments even when they are short and perform an n-valued classification without any feature selection. The latter point is valuable in that there is an infinite number of ways to perform feature selection for such an open ended problem as this, and the selected ones must be verified before use. We have also developed a prototype classifier that manges to correctly classify fragments in an independent experiment in a reasonable amount of time proportional to the number of unknown fragments that need to be classified.

We plan to work in the future on the problem of moving the research to a realistic setting; studying actual captures of disk and memory slack space to develop a notion of what an investigator (or her tools) would meet in the area of problematic data types.

Regarding future work, we have for now addressed the problem as an n-valued problem where each file fragment is of equal value, this is probably not realistic. More likely, for certain types of investigations some types of files are more or less interesting, e.g. the investigator might be more interested

in images in one investigation and in email in another. The method should take this into account when classifying fragments in an attempt to reconstruct files. The reconstruction process itself in conjunction with the identification of fragment types needs more study, given that we have fragments without sufficient structure to make a more programmatic reconstruction possible. A prototype classifier would probably also benefit from a more sophisticated classification algorithm than *k-nearest-neighbour*, but this also makes the results of the experiments harder to interpret. Also, the method and data sets themselves need further study and, to a lesser extent, further development.

To foster friendly competition in research in the area, our specific data (lists of actual fragments and files selected etc.) will of course be made available on request.

8. REFERENCES

[1] Simson Garfinkel, Paul Farrell, Vassil Roussev, and George Dinolt, "Bringing science to digital forensics with standardized forensic corpora," in *9th annual conference of the Digital Forensics Research Workshop (DFRS'09)*, Montreal, Canada, 17–19 Aug. 2009, DFRWS, pp. S1–S11, Elsevier, doi:10.1016/j.diin.2009.06.016.

[2] Rudi Cilibrasi, *Statistical Inference Through Data Compression*, Ph.D. thesis, Institute for Logic, Language and Computation Universiteit van Amsterdam, Plantage Muidergracht 24, 1018 TV Amsterdam, 2007, http://www.illc.uva.nl/.

[3] Paolo Ferragina, Raffaele Giancarlo, Valentina Greco, Giovanni Manzini, and Gabriel Valiente, "Compression-based classification of biological sequences and structures via the universal similarity metric: experimental assessment," *BMC Bioinformatics*, vol. 8, no. 1, pp. 252, 2007.

[4] Eamonn Keogh, Stefano Lonardi, and Chotirat Ann Ratanamahatana, "Towards parameter-free data mining," in *KDD '04: Proceedings of the tenth ACM SIGKDD international conference on Knowledge discovery and data mining*, New York, NY, USA, 2004, pp. 206–215, ACM.

[5] M. Cebrian, M. Alfonseca, and A. Ortega, "The normalized compression distance is resistant to noise," *Information Theory, IEEE Transactions on*, vol. 53, no. 5, pp. 1895–1900, May 2007.

[6] P.M.B. Vitanyi, F.J. Balbach, R.L. Cilibrasi, and M. Li, *Information Theory and Statistical Learning*, chapter Chapter 3, Springer-Verlag, 2008.

[7] Manuel Cebrian, Manuel Alfonseca, and Alfonso Ortega, "Common pitfalls using normalized compression distance: What to watch out for in a compressor," *Communications in Information and Systems (CIS)*, vol. 5, no. 4, pp. 367–400, 2005.

[8] Stefan Axelsson, "Using normalized compression distance for classifying file fragments," in *The Third International Workshop on Digital Forensics (WSDF'10) (In The Proceedings of ARES 2010)*, Krakow, Poland, 15–18 Feb. 2010, IEEE, pp. 641–646, 978-0-7695-3965-2/10 DOI 10.1109/ARES.2010.100.

[9] J. Seward, "Space-time tradeoffs in the inverse b-w transform," in *DCC '01: Proceedings of the Data Compression Conference*, Washington, DC, USA, 2001, p. 439, IEEE Computer Society.

[10] Ron Kohavi, "A study of cross-validation and bootstrap for accuracy estimation and model selection," in *International Joint Conference on Artificial Intelligence (IJCAI)*. 1995, pp. 1137–1143, Morgan Kaufmann.

[11] M. Karresand and N. Shahmehri, "Oscar — file type identification of binary data in disk clusters and ram pages," *Security and Privacy in Dynamic Environments*, vol. 201/2006, pp. 413–424, July 2006, ISBN 978-0-387-33405-9.

[12] M. Karresand and N. Shahmehri, "File type identification of data fragments by their binary structure," in *IEEE Information Assurance Workshop*, West Point, NY, USA, 21–23 June 2006, IEEE, pp. 140–147, ISBN: 1-4244-0130-5.

[13] Cor J. Veenman, "Statistical disk cluster classification for file carving," in *IAS '07: Proceedings of the Third International Symposium on Information Assurance and Security*, Washington, DC, USA, 2007, pp. 393–398, IEEE Computer Society.

[14] William C. Calhoun and Drue Coles, "Predicting the types of file fragments," in *Digital Investigation, The Proceedings of the Eighth Annual DFRWS Conference*, Philadelphia, PA, USA, Aug. 2008, vol. 5:1, pp. S14–S20, doi:10.1016/j.diin.2008.05.005.