University of Aveiro
Computer Engineering Masters
Algorithmic Theory of Information
Armando J. Pinho
Diogo Pratas

Diogo Almeida, Rodrigo Ferreira

November 29, 2020

1 Data Compression to Measure Similarity Between Files

On this assignment, we take advantage of the code used in assignment 1 related to finite context models. But this time there's a twist though, we're not generating text, we're evaluating the similarity between files using our Markov models. This can be done by iterating through the target's text, through the different contexts and events and by making use of its probability in a model, to measure the estimated amount of bits it takes to encode it with said model. This approach can be used in many different ways, but we're focusing on language detection. To do this, we built models for different languages based on various texts which are in itself, an important part of the process (quality and quantity wise).

1.1 Picking the references

It's not a trivial task to determine which texts better represent a language, there's a lot of factors we have to keep in mind, when making this choice. For instance, the text should be rich, it should have variety in its vocabulary, it shouldn't contain foreign words (not a problem if only a few, but when there's a lot of them, we start running into problems when classifying other files), it shouldn't be too specific (for instance, a model built upon english medicine articles could possibly describe a french medicine text than a model built upon random french text). Thus, we made mainly two choices to simplify this procedure. We chose to build our models up from the language's famous books (so we ensure richness in our model), and also from kid's books, this might seem odd at first, but children's books are perfect to represent the most elementary parts of a language, due to their simple phrase structure and minimalistic vocabulary (solidifying the probabilities of the most often occurring contexts, while the other novels we choose also do this to a certain extent but get more specific with rarer words).

1.2 Across the models

Now that we have a method for picking references for each language, we need to make sure that all the models are somewhat similar in terms of "knowledge" obtained. A model built on poor text can mess with our results as we will see later on. Though this is a hard problem to manage, we took a somewhat simplistic approach of having a similar reference size for each language while also ensuring that all of them have both kid's books and regular novels. Needless to mention, to keep all things equal, we must use the same smoothing parameter and context size for all the models, as we can also see in our observations, later, the context size has a huge impact on the ability to recognize a language.

2 Tasks

2.1 Task 1

Task 1 asked us to calculate the estimated number of bits it would take a certain model to encode a given target. This is a very simple procedure as we simply have to sum $-log_2(P(s|C))$ for every context and event we find, as we iterate through the target text (where $P(s|C)$ is the probability of event s occurring after context C in our model). This is done in our language.py, it takes a context size, a smoothing parameter, a model file path and a target file path and prints out the estimated number of bits it would take that model to encode that target.

2.2 Task 2

Task 2 had us guess a certain file's language, given some language models. This is what recognizer.py does, it takes a context size, smoothing parameter, the folder path containing the models and the target file's path. Similar to language.py in task 1, it calculates the estimated number of bits it takes to describe the target, for every language, and ranks them from lowest to highest, where the lowest value is our best guess (assuming we have well balanced models).

2.3 Task 3

To gather the files that would later build our models, we made use of Project Gutenberg's library, where we got our text files from in bulk. As it was previously mentioned, we decided to use kid's books as well as novels to serve as references. In the end we came up to 18 total models, representing 18 languages, which are in our models folder.

2.4 Task 4

This task is addressed in our observations section.

2.5 Task 5

This was by far the hardest part of the assignment, langsegments.py's purpose is to detect ocurrences of various languages within the same text. We do this by iterating through the text, but this time, instead of summing (as we would for task 1) $-log_2(P(s|C))$ for all contexts and respective events found, we save them all on a list, which we will later pass through a low pass filter, to remove noise and smoothen our results, and then, we can see when the values go below each language's threshold of $-log_2(\frac{|\Sigma|}{2})$. When the values are below this threshold for a certain interval, we consider this to be our model's way of telling us that such interval was considered similar to whatever the model describes (language in this case).

2.6 Task 6

Task 6 asked us to make the report you've been reading for the last few minutes, the illustrations will come about in our observations section.

3 Development

3.1 First steps

We started out by making a class to represent each model, containing useful attributes, mainly its name, alphabet and finite context model (a dictionary). Using the previous assignment's code, the first few tasks weren't too difficult, and our difficulties came mostly when deciding what references to use and how to check on our results.

3.2 Difficulties

As we noted, we had some trouble deciding on the references. We went through a few ideas, first we thought about literal dictionaries, which proved to be a decent but not so efficient solution, then we thought about news and other articles, but these could at times be too specific and not the best way to describe a language, and eventually we landed at the aforementioned mix of novels and kid's books, which left us satisfied with the results, though we still had to take what we could get, when gathering material for a specific language was too difficult. Though we still had some difficulty picking references for more "obscure" languages, as our understanding of norwegian, for instance, is quite limited. We ended up choosing the most famous, in most cases. We also had some trouble checking our results, mainly when trying to recognize multiple languages in a single file. It's hard to assess results when looking at a huge stream of numbers, so we had to find a way to help us visualize it, we pulled through by plotting line graphs with the values, making the values much clearer. Originally, the books from the Gutenberg Project, all brought a lot of copyright information in English, which messed with our results at first until we removed it, we also had some issues with different encodings.

3.3 Observations

Aside from choosing the references, this was the most interesting part of the assignment, getting to see how different parameters bring about different results, and speculating on the logic behind them.

3.3.1 Context Size

We will start by discussing the effect the context size had in our experiments, increasing the context size up to a point, seems to be one of the ways to improve the accuracy of our recognizer, though it has diminishing returns.

With a consistent ($\alpha = 0.001$) and changing only the context size, we get the following results with the following target text using $k = 2, 3, 4$ and 5:

```
Este exemplo de texto tem como objetivo testar os diferentes valores para o tamanho do conte
```

pt	262.0083376438982	pt	228.62283562313905	pt	216.16716312565836	pt	244.5834074387547
es	320.7192395024263	es	369.8663734142126	es	387.37821043618806	es	442.72547654223877
sl	363.58329026221344	sl	372.4159722524319	ro	528.9351872121678	ro	511.8277154525378
fr	385.060599458956	ro	470.54279058615253	sl	557.1540045882795	fr	597.9988130343571
cs	410.8709859715635	de	481.30114549028684	de	557.5317144769834	de	605.116805905348
en	414.1371640952463	fr	485.60775849790383	it	590.5317238104499	da	607.4880839777908
it	419.71702293334636	it	489.7279076670687	fr	610.7253803032614	pl	626.3218620825683
ro	422.1717220461304	en	523.1067203107391	pl	637.2531318233191	no	626.7472623635387
pl	425.89021046472027	pl	527.8276813461323	ru	659.9378142370589	el	629.9754281183098
de	443.14197059995286	cs	528.9755029672698	en	668.5636682509269	en	632.3763302949762
af	450.4047800795321	no	562.2456439766436	af	672.0776119051562	ru	640.3801192222469
da	455.25028297926565	af	582.6825059275848	fi	672.2688239189787	it	647.7886282951491
no	455.98586104515	fi	594.448356701668	da	685.6476314540303	sl	663.9362741281683
fi	513.1683459832853	da	645.457943960858	el	698.3550947701033	fi	667.418246822906
el	533.6823754400847	el	689.673158786976	cs	701.6649513082198	af	677.9024028186897
ru	553.6401480146016	ru	693.6847402775427	no	704.7739102043207	cs	704.9610846251172
ja	651.197171738928	ja	916.6088498743045	ja	1051.7713726947454	ja	1070.3580093850078

3.3.2 Smoothing parameter

Our tests seemed to show that as we decrease the value of our smoothing parameter, we increase our accuracy in recognizing a language, and the number of bits needed to encode the target increase to all except for its real language, as seen in the following example.

For:

```
este es un texto de muestra para probar nuestro programa
```

Keeping all other parameters consistent (k=3) and changing only the smoothing parameter, we get the following results for $\alpha = 0.99, 0.5, 0.1$ and 0.001 respectively:

el	130.7451557911626	es	145.5239876004103	es	145.89387280062584	es	145.9878294674318
es	145.08961724552785	el	162.56129061583133	pt	191.4545560528285	de	203.29734364395608
pt	175.8517487497949	pt	180.99072477484737	de	196.0349212593264	fr	215.5162707912749
ru	177.00004093204245	de	191.41904528948317	fr	201.79722545459916	pt	218.36163129463654
de	187.9518715125199	fr	195.5953115790927	ro	213.7515659044759	it	242.38014276559255
fr	191.99811773577505	ru	199.99058710870185	it	215.31635277012538	ro	247.523895924651
ro	193.4315473017022	ro	200.2346854790554	sl	228.80413220087922	no	258.5335730278769
it	198.82991382299937	it	204.36935603605252	el	230.6835248899112	sl	262.8271494587907
pl	204.6628148598797	pl	213.58175117003182	no	231.30964784247715	pl	278.7283349962282
sl	206.6956437180137	sl	214.4007521579106	pl	231.71263800300756	da	292.0487297306274
no	213.3199892827698	no	219.68864500260727	ru	248.39725885342426	en	299.1358277665546
cs	217.40337304412537	cs	229.27574919678239	da	251.3946312626155	af	308.991155607897 6
af	222.7909719561925	af	233.65667185702205	en	252.13959270540582	cs	321.37444011929995
en	225.33327323355337	en	234.1065463446121	cs	254.40966674274875	ru	375.85986824387953
da	225.9346493605155	da	234.67241093582982	af	255.06203937996042	fi	385.47630695083296
fi	243.911822805487	fi	261.9623079433931	fi	297.53605111322423	el	411.66088255684366
ja	268.0562603336762	ja	296.62224185126433	ja	358.94911378318534	ja	526.3397406343855

3.3.3 Model Size

The model's size isvery important when it comes to its reliability. A big enough model is well rounded, and has seen lots of different contexts and knows better which are the most often occurring than the smaller models. Thus, we expect the smaller model values to be more volatile, because it's more likely that there might be things in the target that weren't found in the model. This is still possible, but less likely in bigger models. We tested the following target text with all the same parameters, changing only the model size:

```
the quick brown fox jumps over the lazy dog
```

Results for 500kb, 1Mb, 3Mb models respectively:

```
en 208.61329017989493
da 244.8109942470424
no 246.96333115269712
el 260.31178560794814
ro 264.68667902071303
ru 269.3992900715342
es 273.7291383950071
de 278.9397207889708
sl 281.2137322554052
cs 285.9413056705968
pt 287.74128665440384
fi 292.85901928049697
is 296.85181573373313
pl 297.89841914959504
fr 302.38346822143313
it 306.3729356226249
af 310.28962822098765
ja 457.00863613197515
```

```
en 161.3141708614379
fr 255.80769698638298
af 258.7956252476772
ru 269.3992900715342
pt 271.60248494605537
da 282.2250433580781
no 283.0797754506967
el 285.16683157385563
es 286.4819890037262
de 294.2774707471198
fi 299.7404427779337
cs 306.8586703499899
sl 311.83002321206
is 316.0159582798144
it 322.06304093588557
pl 322.2329072842463
ro 330.6677103305796
ja 463.8289477640654
```

```
en 135.17510918181227
pl 232.43461976012546
af 251.19587011785296
sl 254.49970096844135
da 258.6533725246216
no 259.213168297051
fr 273.9945934963774
de 287.9386149527243
el 292.36787313258196
ro 292.42625669524523
ru 299.31994000192526
es 302.89514277953543
pt 307.63005156124564
fi 310.1305102066452
is 329.95328008641786
cs 340.481176199153
it 342.035754899971
ja 365.79864523942234
```

As we expected, a bigger model takes less bits to describe the target, probably because of the reasons we previously speculated.

3.3.4 Target size

Finally we get to test the target size's impact on language recognition. For this test we're gonna run our language recognizer on an excerpt of english test from the second Harry Potter book. We will run it first on the first 50 characters, then the first 200 and in the end, the first 500.

```
en 97.33138202595886
no 251.36002532243157
da 251.38596750048075
sl 258.74461959094225
af 265.7495941345859
fr 268.3306770031271
pl 283.4029979843326
ro 292.168529557297
de 326.96550238659034
el 329.52805966270614
ru 333.1616396306562
pt 369.4042019439559
fi 370.1965879687469
es 373.307888048716
it 378.9573881353686
ja 396.2049180437759
cs 396.3641192118369
```

```
en 599.8761101088976
af 1044.1861922272421
sl 1108.1813663839482
pl 1167.1883880724702
fr 1195.4732609240452
ro 1200.3808389761114
no 1206.1571673040428
de 1286.9980309668033
da 1366.3256235864976
ru 1391.5150396074384
el 1406.2187514797308
fi 1501.608414312878
cs 1523.3250053448335
pt 1541.2068988966216
it 1577.4914344525337
es 1614.1495075850194
ja 1806.0029141235623
```

```
en 1390.154045582166
af 2702.981794494724
sl 2873.0330888078533
pl 2937.014373655799
no 3070.553753838996
fr 3194.89101780211
de 3306.386319090596
ro 3307.0272511742082
da 3396.57955165556
el 3547.4207336655495
ru 3574.364378008591
fi 3742.8275917469937
pt 3756.7854947200735
es 3829.168898375244
it 3880.8661756174697
cs 4000.946296587434
ja 4723.582292419517
```

As we can see, the program guesses correctly in the three instances, but the ratios of the first ranked and the other languages decreases slightly.

3.3.5 Language similarity

We can see, mainly with our recognizer, that languages with the same origins tend to have similar scores, or at least, be grouped together in the rankings returned for a certain target. This was to be expected, as languages that share origins tend to have many similar contexts, this can be noticed especially with lower context sizes.