2016-09-30

What's in a name?

Introduction

Machine learning works by taking examples of a certain domain and generalizing them to unseen data. It is essential that there is correlation in the data, otherwise one can not hope to generalize. In order to see how machine learning is applicable to the current task, we have to find out where there is correlation. The task at hand is to decide whether a certain string, expected to contain a few words, is a name. In order to simplify the problem, I'll discuss single words first and afterwards consider strings consisting of several words.

Looking at single words

When considering a single word, it can be either

a) a name
b) a word which is not a name
c) a random sequence of characters

If the task were to identify whether a string is a valid word or name, or just random characters, a simple machine learning algorithm trained on some words and some random sequences would probably be able to say with some certainty that “btrrwkkwq” is not a word, whereas something like “baren”, which I just invented, does sound like a word. But when it comes to differentiating words from names, the difference seems arbitrary: In German, why is “Maren” and “Karen” a name, while “garen” is just a regular word (ignoring capitalization, which I'll get to later)? There is seemingly no structure in these datapoints, meaning datapoints that lie close together are not more likely to belong to the same group. Although I do not claim that there is no structure at all, I do think it is very subtle and beyond the scope of this challenge, because it would probably require training a classifier for an extended period of time on a very big dataset.

My previous argument is seemingly disproved in Andrej Karpathy's excellent blog post “The unreasonable effectiveness of recurrent neural networks” [1], in which the author uses a recurrent neural network not only to generate mostly valid latex-code and some linux C source code, but also names, after being trained on a list of 8000 first names. The list of names that the network came up with is eerily realistic [2]. Am I wrong and is there something special to names that makes the difference to other words non-arbitrary? There are two caveats: the network did not only generate valid sounding names, but also words that are non-names, like “can”, “killer”, “baby”, “R”, and “hi”. This would lead to the conclusion that also a powerful neural network didn't find a substantial difference between a regular word and a name. Additionally, the network was only trained on US American first names. If one would include first names from other countries, the network would probably generate much more random sounding words, an effect that would likely be even worse once one includes last names.

Using first and last names structure?

So now that single words are hard to distuingish into words and names, is there hope to make use of the typical first name, last name structure? There probably is. In some languages, last names are longer than first names, in others it is the other way around. In Icelandic, last names always have the same ending, only dependent on whether the person is male or female. So there are clues to be

picked up here. One might think that capitalization of names can be helpful (when classifying words from a text), but this is language specific, because in German, for instance, all nouns are capitalized.

Method

In the end, I decided to go the most pragmatic way: Machine learning is powerful, and in a lot of cases it is much better than handcrafted rules, but in this case it seemed to be that handcrafted rules may be the way to go. In this case, I decided that due to the fact that the difference between words and names is so arbitrary, a quick and effective solution is the simple dictionary lookup.

For this lookup method, I used pythons set datastructure, which is using hashes to determine whether an object is part of the set, which makes the computation time essentially O(1) in the number of objects in the set. That means I can include massive amounts of names in the lookuptable and classification speed will still be significantly faster than any machine learning classifier ever could be. When classifying, the rule for classifying a name is extremely simple: take the input string and split it at every space. Look up every resulting word in the lookuptable. If all words are present, classify it as a word.
With this method, the classifier easily ignores a couple of problems that could potentially pose a problem for a machine learning classifier: some names have non-name helper words in them, like the “van” in Ludwig van Beethoven. Other names are artists names, like Liberace, and have no first and last name. By placing these types of words into the lookuptable, the classifier can handle them. Besides the wikipedia names dataset, I found another big database of names and included it in the lookup table.

In order to evaluate the classifier performance, I generated names and words from www.listofrandomnames.com and www.listofrandomwords.com, respectively. I used 1000 words and names for evaluation.

Results

For the few sample items that were given in the task description, the classifier actually didn't do very well: Nishant Dahad was thought to be not a name. The reason was that “Dahad” does not appear in any of the databases. “Thames river” on the other hand was incorrectly classified as a name, because both “thames” and “river” appeared to be in some peoples names. For the data from listofrandomnames and listofrandomwords.com, the classifier did much better: the scores were:

Accuracy: 0.995
Precision: 0.990
Recall: 1.0
F1-score: 0.995

Notice that there is no area under the curve. Since this classifier has no sliding threshold the area under the curve can not be computed. Since the recall score was 1.0, the classifier found every single name. It misclassified a couple of strings as names:

kellet quin was incorrectly predicted to be a name
november mask was incorrectly predicted to be a name
bernoulli mahala was incorrectly predicted to be a name
frost criminal was incorrectly predicted to be a name
prov november was incorrectly predicted to be a name
puny pater was incorrectly predicted to be a name

Discussion

Can the good results of the classifier be trusted? No. On the one hand, the recall score is reasonable, and the classifier really is good at finding a name if it sees one. There may be a bias towards English names, because the extra databases I included were mostly English names, but obviously it would be easy to add more databases of names in other languages. On the other hand, in real world settings there will be a lot of cases where a named entity (like “thames river”, or “eiffel tower”, which was also misclassified) appears, and all words of that named entity are also part of some name. The classifier would incorrectly classify these non-person named entities as names. Therefore the precision score should be lower. I would have liked to include named entities in the testing list of non-words to get a more realistic precision score, but I could not find a good database (besides wikidata, which is a bit complicated, or rather, time-consuming to parse). However, note that with a database of non-person named entities, I could blacklist these non-person named entities and the classifier would not misclassify them in the first place.

There is another problem with the lookup table, which appears whenever it is to classify two or more words that all happen to be also names. Fortunately this happens quite rarely, as evidenced by the good precision score, but it does happen. Distinguishing these strings that are both real words and real names would require a lot more handcrafted rules.

If I were tasked to create a really well performing classifier, first I would download a wikidata dump and parse it for named entities. Then I would look for more databases of names in other languages to include in the lookuptable.

[1] https://karpathy.github.io/2015/05/21/rnn-effectiveness/
[2] https://cs.stanford.edu/people/karpathy/namesGenUnique.txt