Name Generator

Project Report

Index

Introduction

Problem Statement

Many new parents' wants their new born babies to have new and unique names. They might find name suggestions online and using other applications but most of those names are old and common. Also the name depends on the person's country, gender, ethnicity etc.

So the one way proposed was to use Machine Learning to train an Artificial Intelligence model which takes the baby's gender, birth place, ethnicity, etc into consideration and generates new and unique names suggestions.

The project goal is to generate new first name suggestions for new born babies based on parents' name, surname, gender, ethnicity and birth place. Also do develop a User Interface for people to easily use the software which generates names.

Data

[[First_name, last_name, father_first_name, Father_last_name, mother_first_name, Mother_last_name, gender, birth_place, ethnicity]]

Above mentioned data was found to be available only for celebrities (public figures) and IMDB.com provides film actors' data. So I am using Movie Actors' data for this project. Which is publicly available on the internet.

Used BeautifulSoup[2] for HTML parsing and requests[3] and some other modules to develop a web scraper.

	first_name	last_name	birth_date	birth_place_state	birth_place_country	father_first_name	father_last_name	mother_first_name	mother_last_name	gender
0	Cole	Hauser	1975-3-22	California	USA	Cass	Warner	Wings	Hauser	m
1	Christian	Bale	1974-1-30	Wales	UK	Jennifer	(James)	Christian	Bale	m
2	Donna	(Davis)	1987-2-9	California	USA	Donna	(Davis)	Michael	Jordan	m
3	Nick	Offerman	1970-6-26	Illinois	USA	Cathy	Roberts	Ric	Offerman	m
4	Alexander	Skarsgård	1976-8-25	Stockholms län	Sweden	Stellan	Skarsgård	My	Skarsgård	m

Data Preparation

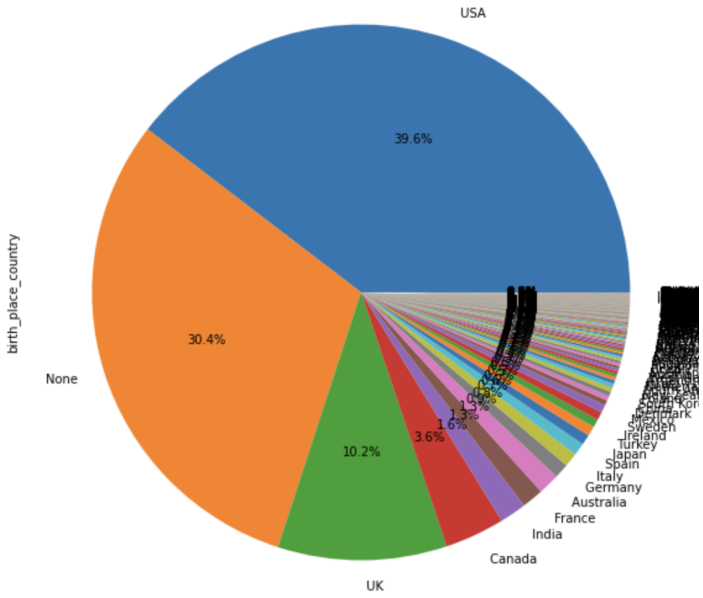
The web scraped data contained many missing fields and invalid values. So I handled those data with the following replacements if possible…

Feature Name		Affected number of rows
First_name	fillNA with Father_first_name or mother_first_name	1668
last_name	fillNA with Father_last_name or mother_last_name	182
Birth_date	fillNA with random pre-decided date	0
Birth_place_state	fillNA with most common state	0
birth_place_country	fillNA with most common country	0
Father_first_name	fillNA with first_name	35971
father_last_name	fillNA with last_name	35972
Mother_first_name	fillNA with first_name	39007
mother_last_name	fillNA with lastt_name or father_last_name	39008
Gender	No missing values	No missing values
ethnicity	fillNA with most common one	fillNA with most common one
	Else drop the row	13

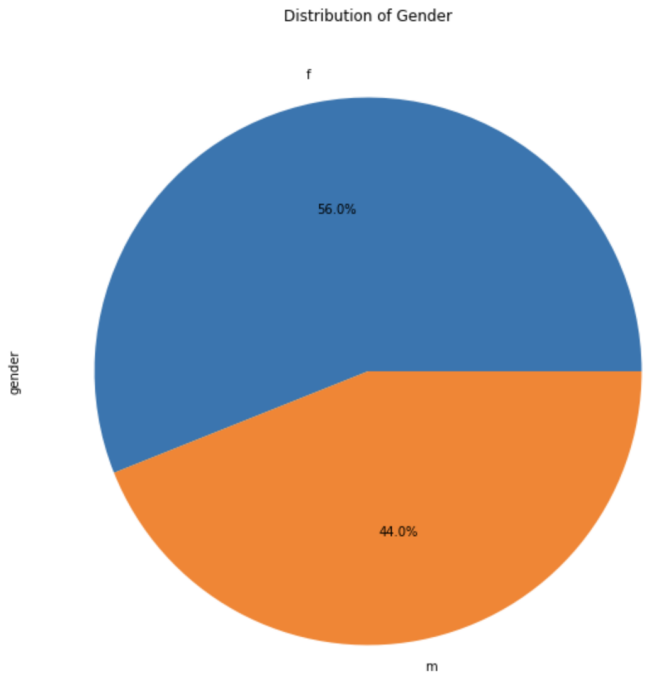
Then I did feature engineering and used SelectKBest from SciKitLearn to select 5 best most correlated features for training.

```
mother_last_name  last_name          0.932420
last_name         mother_last_name   0.932420
                  father_last_name   0.917198
father_last_name  last_name          0.917198
mother_last_name  father_last_name   0.888516
father_last_name  mother_last_name   0.888516
```

Data Visualization



Above figure shows the places the people data is from.



Methods

1. GANs

- First I used Generative Adversarial Networks (GANs) to build a generative model to generate new names.
- By design, GAN can only generate continuous data but I was supposed to generate categorical data where categories being A-Z uppercase alphabets, a-z lowercase alphabets and white space; total of 53 possible values. And the final generated output would be a string made out of a combination of those 53 different values.
- So Initially I only used first_names and trained a basic GAN based model on that data. Where I converted first_names into ASCII values for their numerical representation.
    - “Jeel” → [74, 101, 101, 108]
- Then the GAN based basic model would generate a value between -1 to 1 since I used TanH at the end to make sure that the values have finite range.
- So I have to convert the ordinal values, which would range from (32, 65 to 97 + 90 to 122) to new normalized values which would range from 0 to 1.
- Then the GAN based model was trained on many different hyperparameters and I tried to fine tune it.
- I found that if I use High Epoch or more hidden layers the model Over Fits generates the same names all the time.
- And if I use Low Epoch or less hidden layers the model Under Fits generates gibberish names Like “jfskdlak”.
- For a more detailed method explanation please refer to the Jupyter Notebook which contains well commented code for this.
- Later upon more research, I found that “the Generative Adversarial Networks, were designed to cope with continuous information (image) instead of discrete data (text).” [1]

- So I had to change my model from GAN to LSTM Networks.
- GANs (Generative Adversarial Networks) are a type of deep learning model that are typically used for generating images, videos, and other types of visual media. While it's possible to use GANs for generating text, it's generally more difficult and less common than using other models like LSTMs.
- LSTMs (Long Short-Term Memory Networks) are a type of recurrent neural network that are well-suited for processing sequential data, like text. They have been widely used for tasks like language modeling, machine translation, and text generation.

2. LSTM Networks

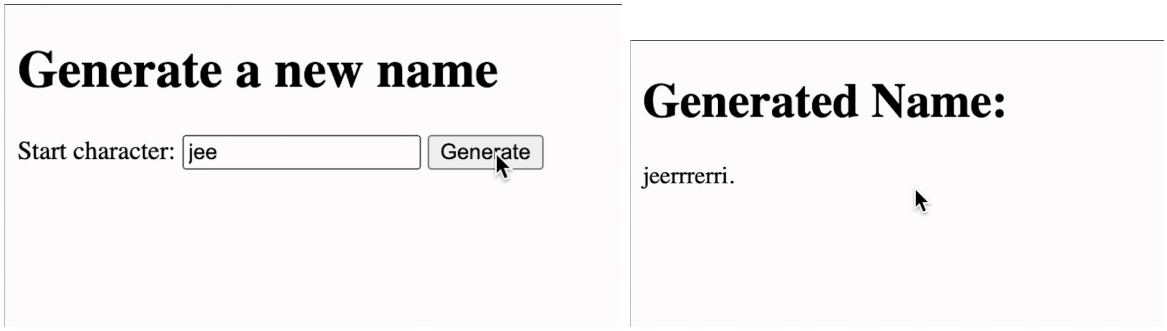
- After changing my model from GAN to LSTI, I again created a basic model only using first_names.
- This time I wanted to have less categorical output values, so instead of allowing both uppercase and lowercase letters, I converted the names to all lowercase only. That way the output categories were only 27 which includes a-z alphabets and “.” which was used as post-padding for the names.
- I again used ASCII values of alphabets to convert text to numerical form.
- The LSTM based model architecture looks like the following…

```
DataParallel(
  (module): Model(
    (lstm): LSTM(27, 50, batch_first=True)
    (fc_1): Linear(in_features=50, out_features=27, bias=True)
    (fc_2): Linear(in_features=27, out_features=27, bias=True)
  )
)
```

- I used CrossEntropy at loss function and Adam as optimiser. For better performance, I used the Learning Rate Scheduler for changing Learning Rate every epoch.
- I trained the model for 50 epoch twice and for 5 epoch twice, a total of (110 epoch).
- I was satisfied with the generated names…
  - ['Jeaie', 'Jerir', 'Jei', 'Jeeaai', 'J', 'Jeraaeareer', 'Jeeer', 'Jeiare', 'Jeaeri', 'Je', 'Jeraeaa']

User Interface

I used Flask[4] to develop a simple User Interface website so that users can easily use the name generator.



Evaluation

- GAN based model outputs:

```
Code cell <mZXtXX-mbMrY>
# %% [code]
for _ in range(4):
  generate_new_name()
Execution output from 20 Apr 2023 17:02
0KB
    Stream
        Generated new name:  X          C
        Generated new name:  X          A
        Generated new name:  X          C
        Generated new name:  X          E
```

-

- LSTM based model outputs:

```
Epoch:1 --> Loss:1.982473324326908
['Je', 'Jeei', 'Jeehiilelel', 'Jeenrsnirrr', 'Jeeslhiae', 'Jeeerr']
Epoch:2 --> Loss:1.3250390747014213
['Jeeanlrie', 'Jeehalt', 'Jeestsrthls', 'Jeehher', 'Jeenantaasl', 'Jeeiennaesn']
Epoch:3 --> Loss:1.303139230784248
['Jeee', 'Jeentiessn', 'Jeehreer', 'Jeehtlireis', 'Jeetlhsisrs', 'Je']
Epoch:4 --> Loss:1.2996042686350204
['Jeeltleller', 'Jeeliiarsie', 'Je', 'Jeernllhlii', 'Jeehhenti', 'Jeelrs']
Epoch:5 --> Loss:1.302516306147856
['Jeet', 'Jeettln', 'Jeeahaharhn', 'Jeeannllese', 'Jeea', 'Je']
```

Conclusion

I learned that the GAN based model is good for generating continuous data while the LSTM based model is good for working for text based sequence data.
There are some workarounds to use GAN for text as well which are discussed here [1] but that does not perform as well as the LSTM based models.

This project is not completed yet, since the current model was trained only using first_names, so for future work, I will develop a final model with all features for better and personalized name generation.

References

[1] https://www.sciencedirect.com/science/article/abs/pii/S0031320321002855
[2] https://www.crummy.com/software/BeautifulSoup/bs4/doc/
[3] https://pypi.org/project/requests/
[4] https://flask.palletsprojects.com/en/2.3.x/