CLASSIFICATION OF BRAZILIAN IMMIGRANTS IN THE U.S.A
Group 1 - Final Report

Federico Siano
fsiano@bu.edu

Runze Liu
runzeliu@bu.edu

Yafei Guo
guoyafei@bu.edu

Yu Zhao
yuzhao95@bu.edu

1. Project Task

The project is aimed at identifying immigrants from Brazil in the U.S.A. using their full names as distinguishing features. Interestingly, this is not a trivial task for at least two reasons. First, immigrants might be second-generation Brazilians and therefore be characterized by an American first name, by an American last name or both. This circumstance makes the likelihood of making false positive / false negative mistakes non-trivial. Second, the Brazilian language derives from the Portuguese colonization of the Country and its therefore inherently challenging to distinguish a Brazilian name from a Portuguese one. Given the available data set, we do not focus on this second point and leave it to future work.

2. Proposed Approach

We implement multiple classification methodologies and then **down-select the algorithm with the best balance of computational intensiveness and evaluation metric** adopted. In more details, we use: (i) *Logistic Regression/Classification*, (ii) *K-nearest neighbor (KNN)*, (iii) *Cosine Similarity*, (iv) *Character-based Recurrent Neural Network (RNN)*. Here follows a description of the main steps through which we have implemented each algorithm.

(i) *Logistic Regression/Classification*: each name in both the training set and test set is split into unique syllables using the Brazilian *hyphenator* in the **Python Natural Language Toolkit Library**. Each unique syllable is then used as a training feature in the Logistic Model. The structure of the data is characterized by a design matrix that comprises all the unique syllables as columns and 0 or 1 dummies for each name, depending on whether that name contains or not a particular syllable. The output vector contains labels (1 for a Brazilian name and 0 otherwise).

(ii) *K-nearest neighbor (KNN)*: all special ASCII characters are discarded in a preliminary step. Subsequently, modified **names are converted to 27-base system numbers** (an example is provided in the "Results"). Converted first and last names are used to create a 2-dimension feature space. Names are classified as Brazilian based on the proximity to a name that was previously classified as such.

(iii) *Cosine Similarity*: the cosine of the angle between two names is calculated implementing the formula:

$$cos(\theta) = \frac{\vec{a} \cdot \vec{b}}{\|\vec{a}\| \left\|\vec{b}\right\|}$$

Where a and b represent features vectors. The distance among vectors of features is calculated and minimized for each name in order to classify the test data.

(iii) *Character-based Recurrent Neural Network (RNN)*: first and last names are first concatenated and saved to distinguish whether they come from the training or test set. Subsequently, **names are converted to tensors using one-hot vectors** which are then joined in a matrix (details are included in the "Results"). The hidden layers structure and the hyper-parameters are calibrated through a basic simulation approach.

3. Dataset and Metric

We are given **48,014 full names** (of which 23,965 are Brazilian) as a **training set**. We are also given **11,942 full names** (unlabeled) as **test** set. We know what the first and last names are and whether a training name is Brazilian. For the Logistic Regression we use the **area under the ROC curve (AUC)** as the effectiveness metric (for consistency with the effectiveness metric used by Kaggle). We also evaluate the **computational time and efficiency**. We implement those metrics also for the KNN, Cosine Similarity and RNN algorithms.

4. Results

4.1. Logistic Classification

The analysis begins with a pilot study aimed at evaluating the in-sample performance of the algorithm. In more details, we initially divide the training sample into a training sub-sample (40,0000 data points) and a test sub-sample (8,014 data points). We then train the model using only first names to extract training features (i.e. syllables). We finally use a **confusion matrix** to evaluate in detail the overall performance of this strategy.

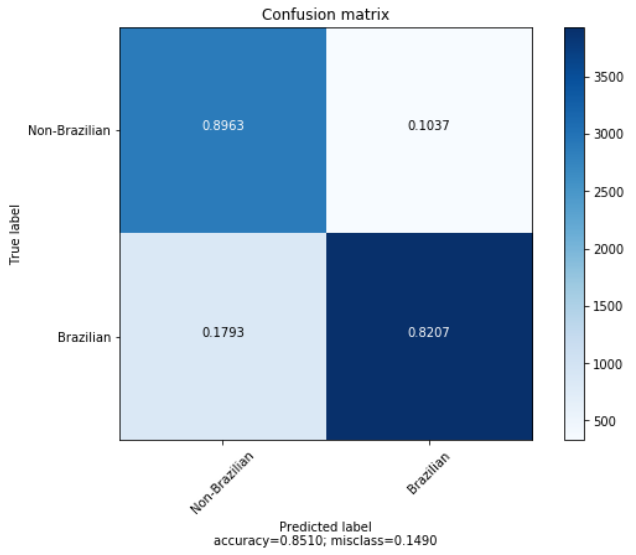


Figure 1. Pilot Confusion Matrix

The result is promising. In fact, with only **5 minutes** of computational time, and a limited number of features, we obtain an overall **accuracy** (i.e. the proportion of true positive and true negative observations over the total test subsample) of **85%**. Acknowledging that the results might be biased by the use of just one in-sample sub-sample, we run a **cross-validation test** in which we divide the sample in 5 sub-samples of around 10,000 observations each. We then use each of them as test sets and the remaining observations as training sets. We train the Logistic model and then average the results. We make sure, whenever possible, to have a total number of test observations labeled as Brazilian (i.e. 1) equal to the total number of test observations labeled as non-Brazilian (i.e. 0) to draw cleaner inferences (i.e. ”Support” in the table). We report the averages (approximated) in tabulated form.

Table 1. Logistic Classification Cross-Validation Results

Average In-Sample Results				
Label	Precision	Recall	F-1 Score	Support
0	0.96	0.84	0.91	5,000
1	0.87	0.85	0.89	5,000

Again, we obtain an average total accuracy (defined above and untabulated) around 85% for a computation time of 5 minutes. At this point, we decide to **increase the number of features** to improve the algorithm performance and conduct the **out-of-sample analysis**. In more details, we adopt **5 strategies**, train the model on test data and check the performance on Kaggle. We summarize the results.

- Strategy 1: only first names, AUC: 86.9%, time: 5 min.
- Strategy 2: only last names, AUC: 83.7%, time: 5 min.
- Strategy 3: ”if-or”, AUC: 86.3%, time: 5 min.
- Strategy 4: ”if-and”, AUC: 84.4%, time: 5 min.
- Strategy 5: concatenation, **AUC: 89.4%, time: 6 min.**

Where: (i) in the ”if-or” strategy we classify a name as Brazilian whenever either the first or the last name is predicted to be Brazilian; (ii) in the ”if-and” case, we classify a name as Brazilian when both first and last name are predicted to be Brazilian and (iii) in the concatenation case, we concatenate first and last names and use the combined set of features (i.e. syllables) to train the model.

4.2. Cosine Similarity

Cosine similarity is a measure of similarity between two non-zero vectors of an inner product space that measures the cosine of the angle between them. The cosine of 0 is 1, and it is less than 1 for any angle in the interval $(0, \pi]$ radians. It is thus a judgment of orientation and not magnitude: two vectors with the same orientation have a cosine similarity of 1, two vectors oriented at $\frac{2}{\pi}$ relative to each other have a similarity of 0, and two vectors diametrically opposed have a similarity of -1, independent of their magnitude.

As we want to classify Brazilian and Non-Brazilian Names, we use **k-means for clustering, and cosine similarity as the distance metric**.

We code the algorithm and train the model using first/last names as features. The results are not satisfactory - they are similar to what one could get by choosing the label randomly. The algorithm is unable to distinguish the two groups of names, as the difference between Brazilian and Non-Brazilian names is not sufficiently significant as to make the clustering effective. Probably, name features are not enough to predict the nationality of an individual in Cosine Similarity. For many mature cosine similarity applications, each sample has a lot of features for model training. So, in our one-feature case, Cosine Similarity could not be ideal.

4.3. KNN Method

Generally, we assume that similar names will have the same labels. We first convert first and last names into unique numbers and then make the comparison. By using the numbers instead of the characters, we anticipate the risk of modeling just a ”match-like” type of classification (a type of classification that is not particularly informative).

We notice that there are some special ASCII character in the training set. We discard all special characters and only care about the letters. We use the 27-base system to transfer the characters. For letter a to z, we transfer them from number 1 to number 26. If there is no number, then it will be zero. To make all the names have a similar range, we preset their digits to a constant number d, which we will adjust later to find a better result. Besides, the leftmost letter is the top digit. **For example, if we set $d = 5$ and a name Prasetyo, it will be transfered to:** $Prasetyo = 16 \cdot 26^4 + 18 \cdot 26^3 + 1 \cdot 26^2 + 19 \cdot 26 + 5 + 20 \cdot 26^{-1} + 25 \cdot 26^{-2} + 15 \cdot 26^{-3} = 7.629159807066454 * 10^6$**.** In this way, every pair of first names and last names will be converted to a pair of unique numbers. We use MATLAB to implement this algorithm. We present in figure 2 the **training data distribution (fist names - last names) in the 2-D plane**, where red stands for Brazilian names and blue stands for non-Brazilian names:

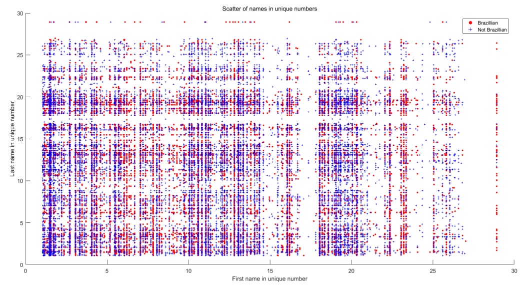


Figure 2. Training Data Distribution in 2D Coordinate

From the figure of data distribution in numbers, we could see that the distribution of names appears almost random. Even though some names are really close, they still have different labels, which is consistent with common sense. For classification purposes, **we assume that if two points are closest to each other, they will share the same label**. For example, same names will appear at the same point, which have a distance of zero. Hence they will be treated as same label. To do such a classification, we just use the KNN algorithm, where k equals to 1. We always classify the new points with the same label as its nearest point. We use again MATLAB to implement this strategy. We use the first name and last name together and make a two-dimension feature vector, which is as the figure 2 shows. However, we only get a AUC of 0.74. We rethink our problem and find that if some names are only the same with respect to first name or last name, they may be far away in the 2D coordinate plane and hence have a high probability of being misclassified. Therefore, we separately train the model for first names and last names, and choose the best prediction. After modifying the algorithm, we obtain a **AUC of 0.83** for a **negligible computational time (i.e. seconds)**.

4.4. Character-level RNN

A recurrent neural network (RNN) is a class of artificial neural network in which connections between nodes form a directed graph along a sequence. This allows it to exhibit temporal dynamic behavior for a time sequence. Unlike feedforward neural networks, RNNs can use their internal state (memory) to process sequences of inputs. This makes them applicable to tasks such as unsegmented, connected handwriting recognition or speech recognition, as well as sentiment classification. A character-level RNN reads names as a series of characters - outputting a prediction and hidden state at each step, feeding its previous hidden state into each next step. We take the final prediction to be the output, i.e. which class the name belongs to (Brazilian or American). We present the structure and architecture of a RNN.

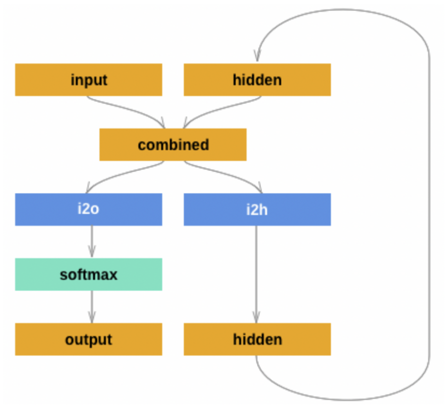


Figure 3. RNN Structure

In the analysis, we proceed through the following steps:

- Create Names Training Set

  We extract Brazilian names and non-Brazilian names from original training set and save them as separate files. Each file contains a bunch of names, one name per line. First name and last name are given the same weight, so we trained the whole name directly as one word. By the way, all words are encoded in UTF-8 for training convenience. See Figure 4.

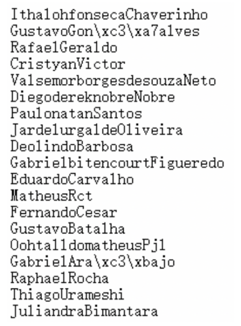
- Turning Names in to Tensors To represent a single letter, we use a one-hot vector of size $1 \times n$. A one-hot vector is filled with 0s except for a 1 at index of the current letter, e.g. $b = [0, 1, 0, 0, 0, \cdots]$. To make a word we join a bunch of those into a 2D matrix $m \times n$.

- Network Structure

  Here,we set $n_{hidden} = 128$ and $learningrate = 0.005$.

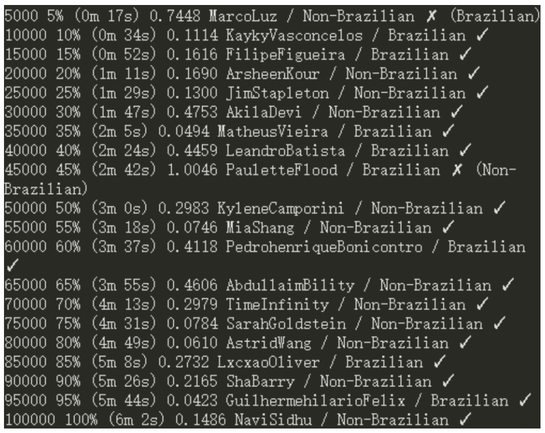
- Prediction and Results

  After $n_{iters} = 100000$ training, we obtain a more accurate and available prediction model. Applying the model to the test set, we finally achieve an **AUC of nearly 0.80** for a **computational time of around 10 min**. We have used different configurations of hidden layers and hyper-parameters following a basic simulation approach. See Figure 5.



```
IthalohfonsecaChaverinho
GustavoGon\xc3\xa7alves
RafaelGeraldo
CristyanVictor
ValsemorborgesdesouzaNeto
DiegodereknobreNobre
PaulonatanSantos
JardelurgaldeOliveira
DeolindoBarbosa
GabrielbitencourtFigueredo
EduardoCarvalho
MatheusRct
FernandoCesar
GustavoBatalha
OohtalldomatheusPjl
GabrielAra\xc3\xbajo
RaphaelRocha
ThiagoUrameshi
JuliandraBimantara
```

Figure 4. Brazilian Names Training Set



```
5000 5% (0m 17s) 0.7448 MarcoLuz / Non-Brazilian ✗ (Brazilian)
10000 10% (0m 34s) 0.1114 KaykyVasconcelos / Brazilian ✓
15000 15% (0m 52s) 0.1616 FilipeFigueira / Brazilian ✓
20000 20% (1m 11s) 0.1690 ArsheenKour / Non-Brazilian ✓
25000 25% (1m 29s) 0.1300 JimStapleton / Non-Brazilian ✓
30000 30% (1m 47s) 0.4753 AkilaDevi / Non-Brazilian ✓
35000 35% (2m 5s) 0.0494 MatheusVieira / Brazilian ✓
40000 40% (2m 24s) 0.4459 LeandroBatista / Brazilian ✓
45000 45% (2m 42s) 1.0046 PauletteFlood / Brazilian ✗ (Non-
Brazilian)
50000 50% (3m 0s) 0.2983 KyleneCamporini / Non-Brazilian ✓
55000 55% (3m 18s) 0.0746 MiaShang / Non-Brazilian ✓
60000 60% (3m 37s) 0.4118 PedrohenriqueBonicontro / Brazilian
✓
65000 65% (3m 55s) 0.4606 AbdullaimBility / Non-Brazilian ✓
70000 70% (4m 13s) 0.2979 TimeInfinity / Non-Brazilian ✓
75000 75% (4m 31s) 0.0784 SarahGoldstein / Non-Brazilian ✓
80000 80% (4m 49s) 0.0610 AstridWang / Non-Brazilian ✓
85000 85% (5m 8s) 0.2732 LxcxaoOliver / Brazilian ✓
90000 90% (5m 26s) 0.2165 ShaBarry / Non-Brazilian ✓
95000 95% (5m 44s) 0.0423 GuilhermehilarioFelix / Brazilian ✓
100000 100% (6m 2s) 0.1486 NaviSidhu / Non-Brazilian ✓
```

Figure 5. RNN Output

4.5. Kaggle Leaderboard

We report the final results from Kaggle, after running all the described algorithms. We include also results obtained from other teams. **Our team members are in boldface**.

Table 2. Kaggle Leaderboard

Test Set Results				
#	Change	Team	Score	Entries
1	-	Gonna_GG	0.908	6
2	new	Yuxi Jiang	0.903	2
3	-1	**Federico Siano**	**0.894**	5
4	new	RF_GG	0.838	1
5	-2	**Yu Zhao**	**0.831**	7
6	-2	BlamonT	0.822	2
7	-2	**RunzeLiu**	**0.798**	10

5. Roles

Table 3. The roles in the team

Roles	
Task	Person
Logistic Classification	Federico Siano
KNN Classification	Yu Zhao
RNN	Runze Liu
Cosine Similarity	Yafei Guo

6. Conclusion and Next Steps

In the analysis at issue, we found that the best performing algorithm (in terms of AUC / time) is the simplest: **Logistic Classification**. We believe that the use of a Recurrent Neural Network has the potential to provide with a stronger overall performance. One opportunity in this sense, would be to **use as input of the Network syllables instead of characters**. In fact, we think syllables could represent more informative features about a language than simple characters.

References

[1] Berry, M. W. *Survey of text mining: clustering, classification, and retrieval*. New York: Springer-Verlag, 2004.

[2] Yong, Z., Youwen, L., and Shixiong, X. *An improved KNN text classification algorithm based on Clustering*. Journal of Computers, 4(3): 230-237, 2009.

[3] *Pytorch Tutorial on Char-RNN*

`https://pytorch.org/tutorials`

[4] *Github link*

`https://github.com/YuZ1225/CS542-Project.git`