Character Identification Through Topological Data Analysis

Aidan Alai, Sonal Reddy Anuga, Caroline Bablin, Steven Caraballo, Hanjoon Choe, Sai Santosh Eedupuganti
Department of Mathematics and Statistics, University at Albany,
Albany Emails: {aalai, sanuga, cbablin, scaraballo, hchoe2, seedupuganti }@albany.edu

Abstract—***The objective is to formulate a classification scheme for a selection of characters with the ability to receive an unknown input and output its character identity. In creating the classification, several features of each character were considered and each these features created a unique vector to describe the character topologically. Such as one-dimensional homology, Betti-number and different kind of filtrations. In this project we used 32 unique characters in which 26 are Latin alphabet and rest are punctuation characters.***

Keywords—classification, 32 unique characters, homology, Betti number, filtrations, lower-star filtration, Ripser, Python.

I. INTRODUCTION

Topological data analysis (TDA) takes an approach to looking at data in terms of its shape. When using topological concepts, high-dimensional data can yield interesting and significant results that other approaches may lack and not consider. The most common method used to study data topologically is persistent homology. Persistent homology partitions data into simplicial complexes so that each simplicial complex is included or nested within the previous one. This sequence of simplicial complexes is first examined at the lowest level, i.e. the simplicial complex that is nested within all other simplicial complexes. Taking a stepwise approach through simplicial complexes, persistent homology tracks features that form in one simplicial complex that are still present in the following simplicial complex. This filtration process continues through all complexes providing a picture of the homological structure present at each step. It will detect the birth, where they first appear, and death, and where they last appear, features in data. The lifetime of a feature is denoted by the difference in the birth and death of that feature and are typically represented by barcodes. Note that following the completion of a filtration, one feature will persist forever.

II. BACKGROUND

A. Lower-Star Filtration and Ripser

Lower-Star Filtration is a function within the Ripser toolbox for Python. It allows for homological scans to be done against datasets that may have meaning in their features. The Lower-Star Filtration will begin scanning from the lowest values to the highest values and read features in that order. Preprocessing the dataset by manipulating it to be rescaled so the values of the features are in ascending order according to the desired direction of the scan.

The Ripser output can be used to create a barcode and the 'lifetime' of distinguishing features are tabulated. These 'lifetimes' represent how long the feature persisted in the letter, i.e. how long the given connected component continued to be present in a scan until it was combined with another connected component. The 'birth' of a component occurs when the feature was first detected, and the 'death' is where the component was last detected.

The standard point cloud Ripser function is also useful in creating a feature-based classification. One-dimensional homology, or the one-dimensional Betti number can be found through this method. Simply put, this number counts the number of 'holes' in a given letter. For example, the Betti number for the letter 'A' is 1, while the Betti number for the letter 'B' is 2. Note that most of letters and characters have a Betti number of zero. Each of these features were determined and then served as a signature for a character.

III. METHOD

A. Classification Scheme

The first step was to create a unique classification scheme for each letter in the Latin alphabetic characters and symbols from sample data. There were 32 unique characters, 26 characters of the Latin alphabet and 6 punctuation marks, all hand-drawn. To start, each hand drawn image was overlaid with a ten by ten array. Each square in the overlaid array that had a part of the character would represent a 1 in the 100 length binary string. All squares with no part of the character contained within it would receive a 0 in its corresponding location of the binary string. A single .csv file was produced to compile all binary strings for each image. An index value was added to the beginning of each row to identify each sample character.

1) Filtration

Once each character was represented by an array, several directional filtrations were applied to the original character arrays. These filtrations produced

rescaled versions of the original arrays. The location and magnitude of scaling were dependent on the desired direction of the lower-star filtration. Each filtration produces one new array. Seven directional arrays were chosen as optimal for encompassing all potential features of the 32 characters:

a) Lower left corner diagonal

The value of the matrix entries with a positive value are set to the number of the corresponding diagonal position with the lowest diagonal having a value of 1 and the highest having a value of 10. If the value of the original matrix entry is 0, then the value would be converted to 100.

b) Center in matrix creation

The value of the matrix entries with a positive value are set to the number of the corresponding diagonal position with the lowest diagonal having a value of 1 and the highest having a value of 10. If the value of the original matrix entry is 0, then the value would be converted to 100.

c) Center out matrix creation

The value of the matrix entries with a positive value are set to the number of the corresponding diagonal position with the lowest diagonal having a value of 1 and the highest having a value of 10. If the value of the original matrix entry is 0, then the value would be converted to 100.

d) Left to right matrix creation

The value of the matrix entries with a positive value are set to the number of the corresponding diagonal position with the lowest diagonal having a value of 1 and the highest having a value of 10. If the value of the original matrix entry is 0, then the value would be converted to 100.

e) Right to left matrix creation

The value of the matrix entries with a positive value are set to the number of the corresponding diagonal position with the lowest diagonal having a value of 1 and the highest having a value of 10. If the value of the original matrix entry is 0, then the value would be converted to 100.

f) Bottom top matrix creation

The value of the matrix entries with a positive value are set to the number of the corresponding diagonal position with the lowest diagonal having a value of 1 and the highest having a value of 10. If the value of the original matrix entry is 0, then the value would be converted to 100.

g) Top bottom matrix creation

The value of the matrix entries with a positive value are set to the number of the corresponding diagonal position with the lowest diagonal having a value of 1 and the highest having a value of 10. If the value of the original matrix entry is 0, then the value would be converted to 100.

2) Vectorization

In addition to the lower-star filtration, the standard Ripser function was used to determine one-dimensional Betti numbers. In total, 24 possible features were recorded. Each character now had a 1x24 array representing the features discovered. At this point, all arrays were compared, and it was determined unique classification was achieved.

The arrays can be considered as the vectorized versions of the characters' image. Now that vectorization has been accomplished to create a unique classification of the known images it is possible to proceed to the next step and start analyzing unknown images for identification.

IV. Algorithm

See GitHub repository:

https://github.com/GitAlai/TDA-Practicum-GROUP-A.git

V. Experimentation

A. Unknown Classification

For testing, unknown test characters we used to validate the classification scheme and determine its reliability. These were created to have small variations from the known sample characters used for classification. The test character image was subjected to the same vectorization method received by the sample characters. This produces a 24 dimensional unknown character vector to then be evaluated against the known sample dataset.

24 features used for classification:

i) Right to Left Life Span (RLLS) 0,1,2
ii) Left to Right Life Span (LRLS) 0,1,2
iii) Top to Bottom Life Span (TBLS) 0,1,2
iv) Bottom to Top Life Span (BTLS) 0,1,2
v) Lower-Left Life Span (LLLS) 0,1,2
vi) Center-Out Life Span (COLS) 0,1,2
vii) Center-In Life Span (CILS) 0,1,2
viii)Betti 1-Count of 1D Features(H) 0,1,2

We used three different distance measurements to determine the closeness of the unknown vector space to our character classification vector spaces. The distances

used were Euclidean distance, Minkowski distance and Hamming distance. For each measurement, we captured up to 3 (0,1,2) Life Spans, and restrict inf span to 100 such that the maximum life span is limited to 100.

Euclidean:

$$argmin_{(letter1,letter2,letter3)\in SET} \sum_{i=1}^{21} \left(f^i{}_{letter} - f^i{}_{input}\right)^2$$

Minkowski:

$$argmin_{(letter1,letter2,letter3)\in SET} \sum_{i=1}^{21} \left|f^i{}_{letter} - f^i{}_{input}\right|$$

Hamming:

$$argmin_{(letter1,letter2,letter3)\in SET} \sum_{i=1}^{21} f^i{}_{letter} \oplus f^i{}_{input},$$

$where \ \oplus \ is \ XOR \ operator$

f^i *represents the ith measurement*

B. Accuracy Calculation

Next, it is necessary to calculate an accuracy percentage to understand the success of our classification. For Euclidean and Minkowski distances min-max scaling was applied to calculate relative percentage:

$$(1 - \frac{input\ dist\ -\ min\ dist}{max\ dist\ -\ min\ dist}) \times 100 \in [0,100]$$

And for Hamming distance an absolute percentage:

$$\frac{d_H(letter,input)}{\#\ of\ features} \times 100 \in [0,100]$$

d_H is hamming distance

The percentages calculation allows for the user to understand the variation between the top three candidate outputs. A greater difference in percentage between the first and second identification outputs could indicate that there is a greater certainty in the first candidate.

C. Results

In most cased the unknown character was correctly classified within the top three candidates. Some letters that were commonly confused are: ‘Z’ and ‘S’, ‘A’ and ‘R’, ‘O’ and ‘D’. The above pairs exhibited similar features and are seen to exist in relatively close vector spaces to each other making their classification commonly interchanged.

VI. DISSCUSION

Character classification has been achieved with close to perfect accuracy using convolutional neural networks. Typically, these experiments have been done using the MNIST dataset. Because this dataset is so large, it is possible to split into training and testing sets. These sets become crucial in the calculation of percent accuracy of classification. Contrary to this approach, our approach was on a much smaller scale, in that our dataset had only one copy of each letter. Due to this, we only tested our model on a few recreated letters. This inhibits the ability to determine the overall accuracy of our model. Instead, we were able to find an accuracy of each tested letter to the ground truth. In certain cases, our model performed very well, accurately predicting what a new letter was. In general, shifted letters with no noise were able to predict accurately. Letters with introduced noise, depending on how much noise, were not predicting with as much accuracy. To increase our robustness and accuracy, we could have used a higher resolution on our data. Instead of using 10 by 10 grids with each letter, using 100 by 100 would yield much more detail.