Applying Persistent Homology to Machine Learning with Persistence Images
CMSE 491 Project Write-up

Victor Ramirez

1 Introduction

With the ongoing "data revolution", the demand for efficient methods to analyze the massive amounts of large datasets has grown. Topology, a field that has remained relatively dormant in providing wide-scale applications until recently, has been infused with computational techniques in an attempt to better represent the properties of space. Computational topology has shown to be a promising approach to analyzing noisy, complex data sets and has been used in computer vision, medical imaging, and biological aggregation models. One of the tools used in computational topology is persistent homology, which can describe features of interest in this dataset by representing the homological information through a collection of points called *Persistence Diagrams* (PDs).

Interest in computational topology has coincided with machine learning (ML), a prominent data analysis technique. ML recognizes patterns among data in order to classify clusters within data and make predictions. Techniques in ML require metrics to draw from in order to make predictions. There's a growing interest to combine both persistence homology and machine learning in order to refine the workflow in data analysis. It's possible to perform ML techniques using PDs and their features as a metric when PDs are mapped into spaces which allow for it to be "fed" into a ML algorithm.

This project will attempt to demonstrate such a ML pipeline. Section 2 discusses the methodology, starting from data extraction, to data transformation, to classifying using a ML algorithm. In this case, I used scikitlearn's linear SVM classifier. Section 3 shows how well this model classified our data and is compared to results from conventional methods. Section 4 concludes with a discussion of the limitations of this model and suggets future improvements. Much of the basis of this project was inspired by the work of Adcock et al [**PI**].

2 Workflow

To understand the workflow, we must define the following entities:

- M = Matrix of pixel values
- P = Point cloud from M
- D_k = k-dimension Persistence diagram; k = 1 for this project
- L_{max} = Max lifetime
- f_n = Carlsson Coordinates; n = [1,4]

The workflow follows these steps:

1. Extract M's of each digit from dataset
2. Transform M into P

3. Calculate D_k
4. Calculate L_{max}
5. Using D_k and L_{max}, find f_n
6. Classify using svm with features = X = f_n

Step 1 is simply data extraction and is laid out in the next subsection. Steps 2-4 are about calculating simple persistence features to use as a backbone for feature selection. Step 5 is about transformations from persistence features calculated in steps 2-4 in order to give each digit more distinct features. The workflow concludes with step 6 of feeding these features into a ML classification algorithm.

2.1 About the Data

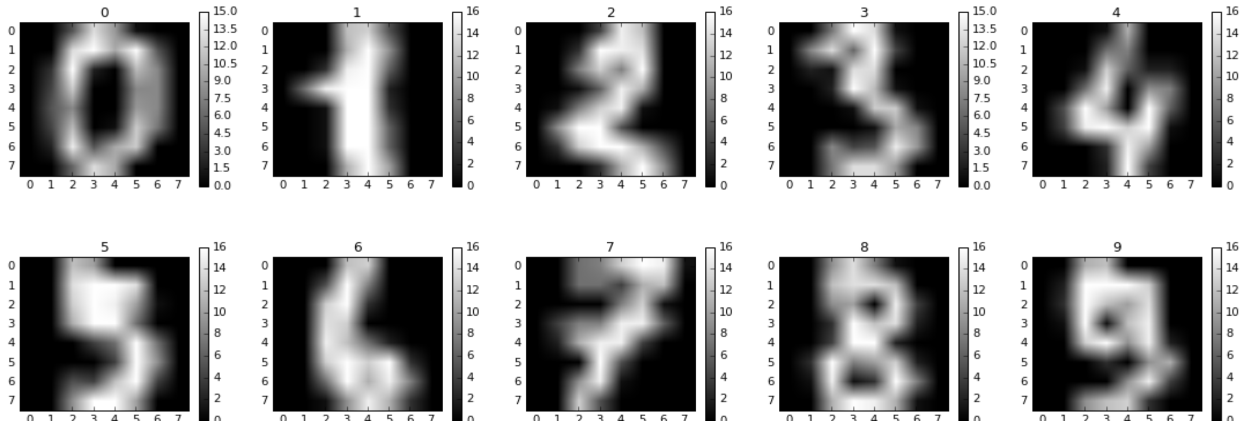


Figure 1: The sample pixels of each digit.

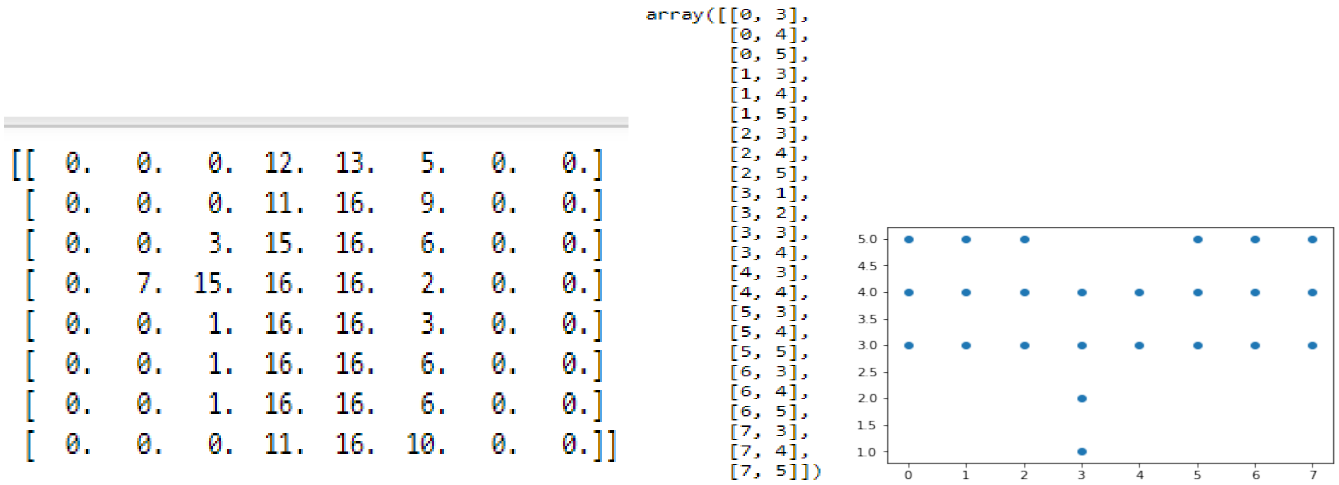
The data I used is from the MNIST handwritten digits, a subset of which can be imported on sklearn as their "digits" dataset. The entire dataset is often used as a barometer for implementing different ML techniques and is used by [**PI**].

The sklearn digits dataset contains 1798 handwritten digits, each with a matrix of pixel values M. M is an 8x8 matrix that has a pixel value in each point ranging from 0-16, 0 being empty space and 16 being a well defined point. Each digit has a label from 0-9 that corresponds to what that digit is supposed to represent. We represent this label vector as what we're trying to classify, y.

2.2 Extracting Persistence Features

The process for extracting the persistence features is summarized as follows:

1. Let pixel value = v
2. Define threshold pixel value t. Here t = 5
3. If v > t, store coordinates in P. Discard otherwise.



```
[[ 0.  0.  0. 12. 13.  5.  0.  0.]
 [ 0.  0.  0. 11. 16.  9.  0.  0.]
 [ 0.  0.  3. 15. 16.  6.  0.  0.]
 [ 0.  7. 15. 16. 16.  2.  0.  0.]
 [ 0.  0.  1. 16. 16.  3.  0.  0.]
 [ 0.  0.  1. 16. 16.  6.  0.  0.]
 [ 0.  0.  1. 16. 16.  6.  0.  0.]
 [ 0.  0.  0. 11. 16. 10.  0.  0.]]
```

```
array([[0, 3],
       [0, 4],
       [0, 5],
       [1, 3],
       [1, 4],
       [1, 5],
       [2, 3],
       [2, 4],
       [2, 5],
       [3, 1],
       [3, 2],
       [3, 3],
       [3, 4],
       [4, 3],
       [4, 4],
       [5, 3],
       [5, 4],
       [5, 5],
       [6, 3],
       [6, 4],
       [6, 5],
       [7, 3],
       [7, 4],
       [7, 5]])
```

(a) (b) (c)

Figure 2: Transformation from M with digit label 1 (a) to P (b) and plotted in (c)

The output from this process is a 2D point cloud P with points corresponding to the coordinates in which the pixel value v $\geq$ threshold value t. We can calculate a 1-D persistence diagram D_1 using the Ripser library with the code:
`persistence_diagrams = teaspoon.TDA.Persistence.VR_Ripser(P,1)`
and the max lifetime with:
`max_lifetime_1d = max(pd_1d[:,1] - pd_1d[:,0])`
These features are stored in a dataframe as shown on 3. Notice the lack of distinction in the persistence features for the first 10 digits; digits 1,2,3,5,6 have the exact same max lifetime and largely similar PD's. These issues are solved in part by using Carlsson coordinates, as described in the next stage of our workflow.

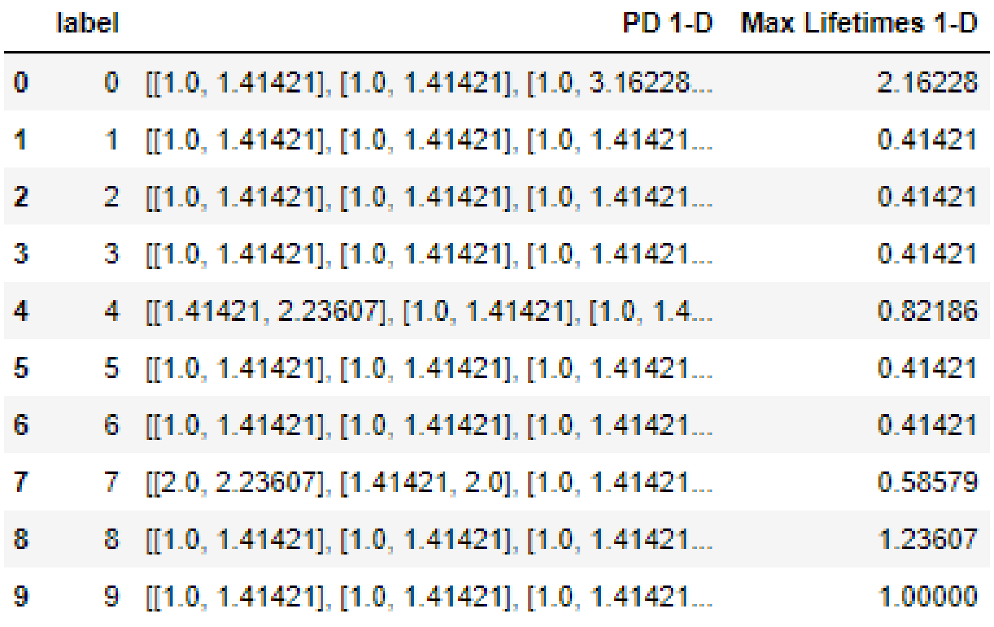
2.3 Carlsson Coordinates

Using the basic persistence features in 1-D is not enough to classify between digits. Thus we define rotationally invariant features called the Carlsson coordinates. These features are described from the following function values: Let

$$
\begin{aligned}
f_1(x,y) &= \textstyle\sum_i^n x_i(y_i - x_i)\\
f_2(x,y) &= \textstyle\sum_i^n (y_{max} - y_i)(y_i - x_i)\\
f_3(x,y) &= \textstyle\sum_i^n x_i^2(y_i - x_i)^4\\
f_4(x,y) &= \textstyle\sum_i^n (y_{max} - y_i)^2(y_i - x_i)^4
\end{aligned}
\tag{1}
$$

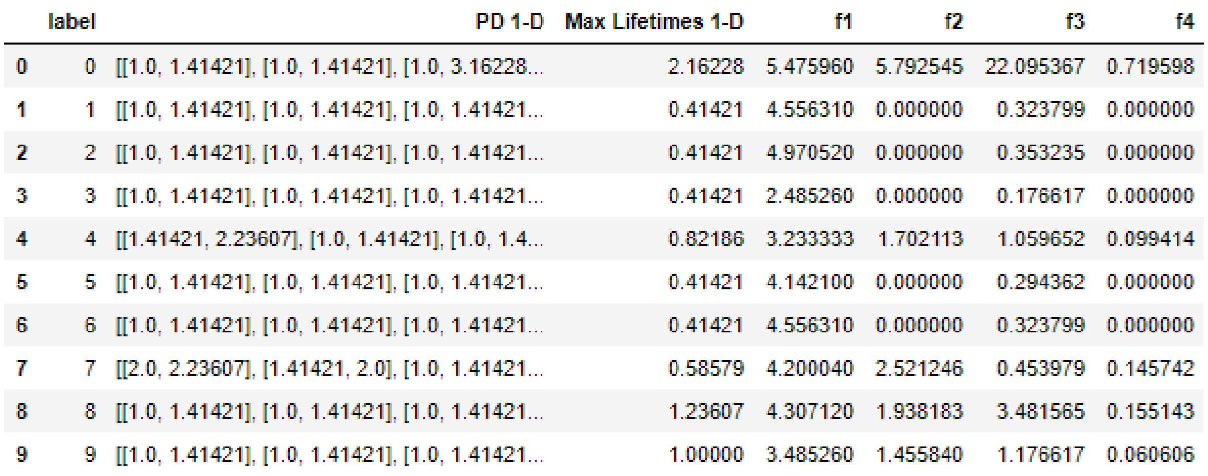
where x,y are points in a persistence diagram

The first two features take all of the lifetimes, lengths and endpoints, into account. The second two features heavily favor the arrangement max-lifetime. These attributes serve to expose and amplify the persistence features in our PD's in order to give a better chance for our classification algorithm of choice to distinguish between each digit. Our new persistence diagram now looks like figure 4.



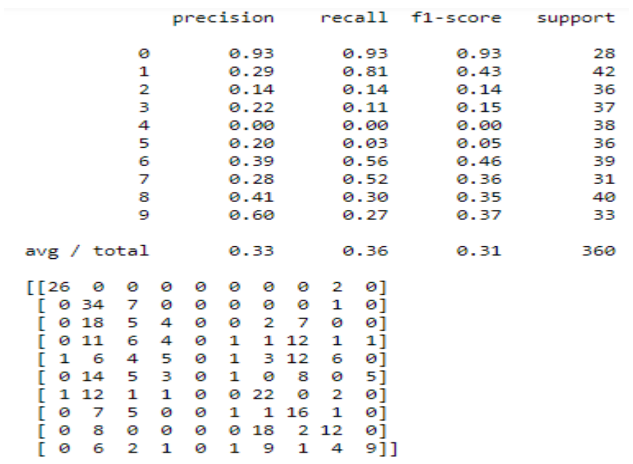
	label	PD 1-D	Max Lifetimes 1-D
0	0	[[1.0, 1.41421], [1.0, 1.41421], [1.0, 3.16228...	2.16228
1	1	[[1.0, 1.41421], [1.0, 1.41421], [1.0, 1.41421...	0.41421
2	2	[[1.0, 1.41421], [1.0, 1.41421], [1.0, 1.41421...	0.41421
3	3	[[1.0, 1.41421], [1.0, 1.41421], [1.0, 1.41421...	0.41421
4	4	[[1.41421, 2.23607], [1.0, 1.41421], [1.0, 1.4...	0.82186
5	5	[[1.0, 1.41421], [1.0, 1.41421], [1.0, 1.41421...	0.41421
6	6	[[1.0, 1.41421], [1.0, 1.41421], [1.0, 1.41421...	0.41421
7	7	[[2.0, 2.23607], [1.41421, 2.0], [1.0, 1.41421...	0.58579
8	8	[[1.0, 1.41421], [1.0, 1.41421], [1.0, 1.41421...	1.23607
9	9	[[1.0, 1.41421], [1.0, 1.41421], [1.0, 1.41421...	1.00000

Figure 3: Data Frame with D_1 and L_{max}



	label	PD 1-D	Max Lifetimes 1-D	f1	f2	f3	f4
0	0	[[1.0, 1.41421], [1.0, 1.41421], [1.0, 3.16228...	2.16228	5.475960	5.792545	22.095367	0.719598
1	1	[[1.0, 1.41421], [1.0, 1.41421], [1.0, 1.41421...	0.41421	4.556310	0.000000	0.323799	0.000000
2	2	[[1.0, 1.41421], [1.0, 1.41421], [1.0, 1.41421...	0.41421	4.970520	0.000000	0.353235	0.000000
3	3	[[1.0, 1.41421], [1.0, 1.41421], [1.0, 1.41421...	0.41421	2.485260	0.000000	0.176617	0.000000
4	4	[[1.41421, 2.23607], [1.0, 1.41421], [1.0, 1.4...	0.82186	3.233333	1.702113	1.059652	0.099414
5	5	[[1.0, 1.41421], [1.0, 1.41421], [1.0, 1.41421...	0.41421	4.142100	0.000000	0.294362	0.000000
6	6	[[1.0, 1.41421], [1.0, 1.41421], [1.0, 1.41421...	0.41421	4.556310	0.000000	0.323799	0.000000
7	7	[[2.0, 2.23607], [1.41421, 2.0], [1.0, 1.41421...	0.58579	4.200040	2.521246	0.453979	0.145742
8	8	[[1.0, 1.41421], [1.0, 1.41421], [1.0, 1.41421...	1.23607	4.307120	1.938183	3.481565	0.155143
9	9	[[1.0, 1.41421], [1.0, 1.41421], [1.0, 1.41421...	1.00000	3.485260	1.455840	1.176617	0.060606

Figure 4: Data Frame now with Carlsson Coordinates



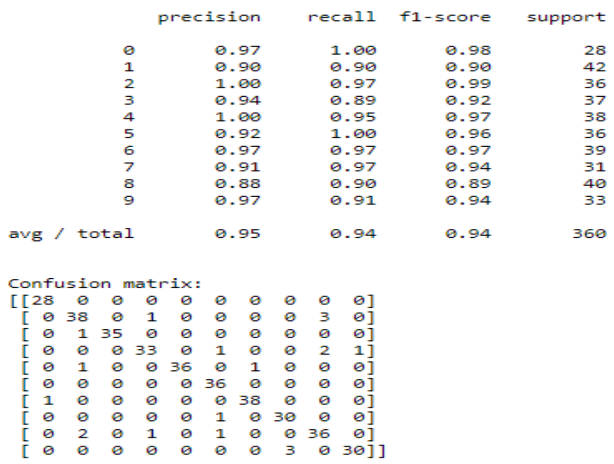
```
             precision    recall  f1-score   support

          0       0.93      0.93      0.93        28
          1       0.29      0.81      0.43        42
          2       0.14      0.14      0.14        36
          3       0.22      0.11      0.15        37
          4       0.00      0.00      0.00        38
          5       0.20      0.03      0.05        36
          6       0.39      0.56      0.46        39
          7       0.28      0.52      0.36        31
          8       0.41      0.30      0.35        40
          9       0.60      0.27      0.37        33

avg / total       0.33      0.36      0.31       360

[[26  0  0  0  0  0  0  0  2  0]
 [ 0 34  7  0  0  0  0  0  1  0]
 [ 0 18  5  4  0  0  2  7  0  0]
 [ 0 11  6  4  0  1  1 12  1  1]
 [ 1  6  4  5  0  1  3 12  6  0]
 [ 0 14  5  3  0  1  0  8  0  5]
 [ 1 12  1  1  0  0 22  0  2  0]
 [ 0  7  5  0  0  1  1 16  1  0]
 [ 0  8  0  0  0  0 18  2 12  0]
 [ 0  6  2  1  0  1  9  1  4  9]]
```

(a)



```
             precision    recall  f1-score   support

          0       0.97      1.00      0.98        28
          1       0.90      0.90      0.90        42
          2       1.00      0.97      0.99        36
          3       0.94      0.89      0.92        37
          4       1.00      0.95      0.97        38
          5       0.92      1.00      0.96        36
          6       0.97      0.97      0.97        39
          7       0.91      0.97      0.94        31
          8       0.88      0.90      0.89        40
          9       0.97      0.91      0.94        33
avg / total       0.95      0.94      0.94       360

Confusion matrix:
[[28  0  0  0  0  0  0  0  0  0]
 [ 0 38  0  1  0  0  0  0  3  0]
 [ 0  1 35  0  0  0  0  0  0  0]
 [ 0  0  0 33  0  1  0  0  2  1]
 [ 0  1  0  0 36  0  1  0  0  0]
 [ 0  0  0  0  0 36  0  0  0  0]
 [ 1  0  0  0  0  0 38  0  0  0]
 [ 0  0  0  0  0  1  0 30  0  0]
 [ 0  2  0  1  0  1  0  0 36  0]
 [ 0  0  0  0  0  0  0  3  0 30]]
```

(b)

Figure 5: Summary stats for using persistence features (a) vs. using features within sklearn dataset (b)

2.4 Applying ML to Persistence Features

Armed with more distinguishing features, we can now apply a ML classification algorithm of our choice to our dataset. I chose just the features from the Carlsson coordinates as my X. I chose sklearn's linear SVM for simplicity, but other classification algorithms should work using the same template. The code for this ML pipeline is as follows: Let X = [f1, f2, f3, f4] and y = [label]

```
X_train, X_test, y_train, y_test = train_test_split(
X, y, test_size=0.2, random_state=1349)
lin_clf = svm.LinearSVC()
lin_clf.fit(X_train, y_train)
y_pred = lin_clf.predict(X_test)
```

The results of this pipeline is discussed in the next section.

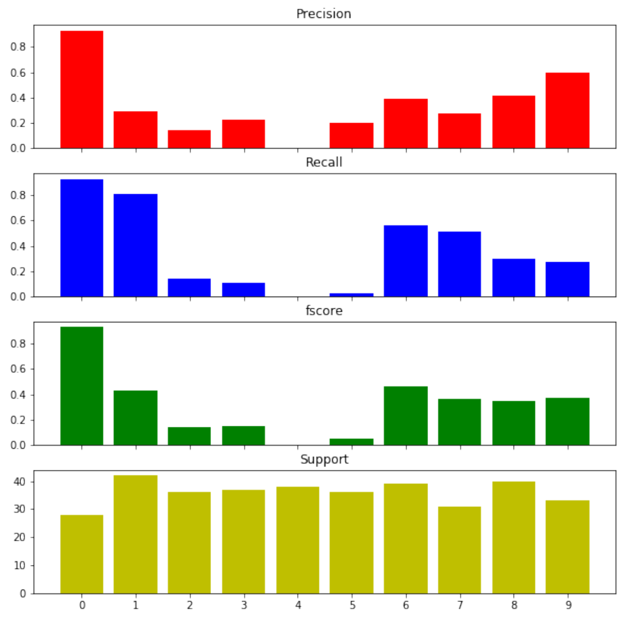
3 Results

To understand the context of these results, we must define the metrics for our classification:

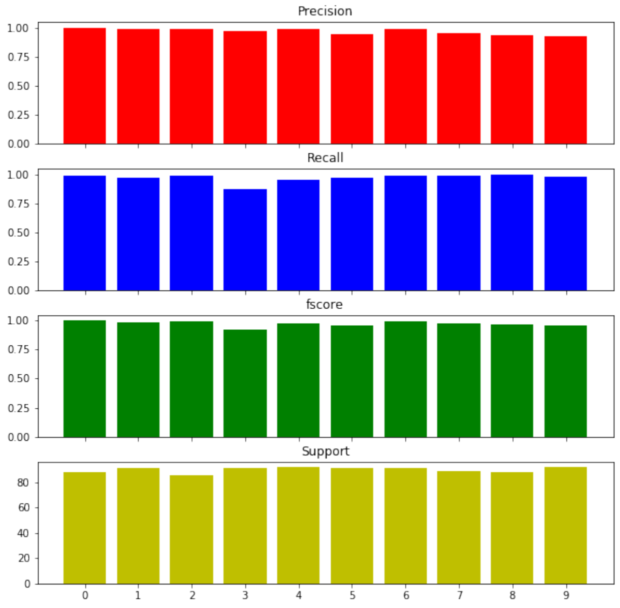
$$\begin{aligned} precision &= \tfrac{tp}{tp+fp} \\ recall &= \tfrac{tp}{tp+fn} \end{aligned} \tag{2}$$

where tp is a "true positive" or correct classifcations, "fp" is a false positive or a different digit being incorrectly classified as the digit of interest, and "fn" is a false negative or a digit of interest being classified as a different digit. The f1 score is the harmonic mean between precision and recall. Lastly the "support" is how many correct digits were classified.

The results from this pipeline ranged from fairly accurate to completely inaccurate when compared to a conventional machine learning method of using all pixel features. Our model did the best with classifying 0's as those have marked persistence features. The rest of the digits were less than 60% precise. 4's had it especially rough, registering 0 correct classifications. One observation of note is that some digits fare better on some metrics than others. For example, 1 has a low



(a)



(b)

Figure 6: Classification results for using persistence features (a) vs. using features within sklearn dataset (b)

precision but a high recall. This means that there are many false positives being detected for 1 but not a lot of false negatives.

The limitations of this model defined to create a simple workflow partially explain the poor results. The main culprit is using just 1-D persistence which cascades into many other issues. I set an abitrary threshold in order to capture all of the data, but this has the drawback of capturing too many pixel values that are irrelevant to the digit, blurring the picture. I also had to throw away a lot of pixel data in order to fit this into a 2-D point cloud. The low resolution of pixel values did not finely distinguish between digits. Lastly, some digits are simply too similar in 1-D. Digits with no loops were grouped to be largely the same as a result of the features chosen.

4 Conclusion

This paper shows how persistence features can be used to funnel into a machine learning pipeline. Though limited by simplicity, the model outlined can be easily expanded to more complex workflows. This model would undoubtedly benefit from higher dimensionality as this would allow the access to various other persistence features. Future work can be expanded to examine how different ML algorithms fare with the given persistence features.

5 References

[1] Adcock, Aaron, et al. “The Ring of Algebraic Functions on Persistence Bar Codes.” Homology, Homotopy and Applications, vol. 18, no. 1, 2016, pp. 381–402., doi:10.4310/hha.2016.v18.n1.a21.

[2] ScikitLearn http://scikit-learn.org/stable/index.html