Classification of Colon/Rectal Tissue

William Grayson Croom

The University of Texas at Dallas
Mathematical Sciences

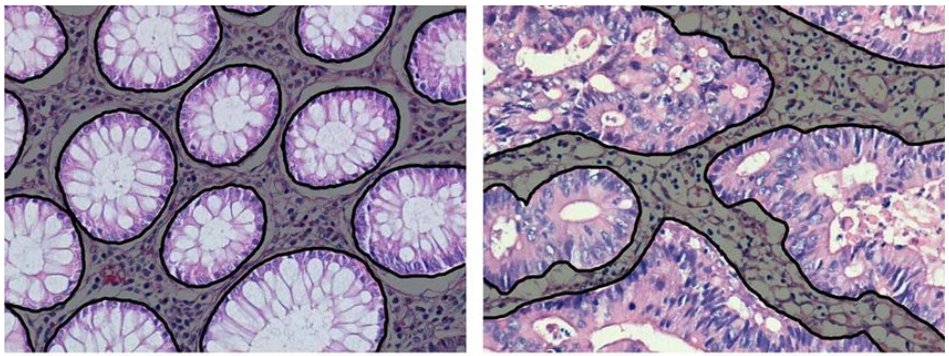
December 1, 2022

Summary

Main Idea

The geometric structure of cancerous tissue is very different than that of non-cancerous tissue under a microscope.

Figure: Cancerous vs Healthy Colon Tissue



(a) A normal and (b) a cancerous colon tissue image from [Gultekin et al., 2014].

Is it possible to use Topological methods to determine whether a tissue sample is cancerous or not?

The Dataset

The Colorectal Histology MNIST

A dataset containing 5000 (150x150 pixel) image tiles of colon/rectal tissue, each belonging to 1 of 8 different categories of tissue.

Figure: Full Tissue Images

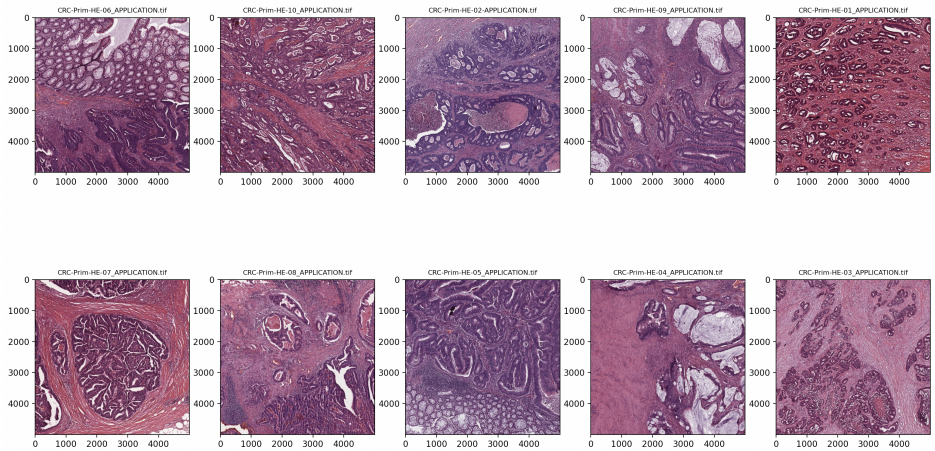


Figure generated from The Colorectal Histology MNIST dataset [K Scott Mader,].

Narrowing the Scope of the Problem

Classifying the images into 1 of 8 different categories is very complex. For this reason, we will be restricting ourselves to the following 2 settings (as recommended by Dr. Coskunuzer).

The Binary Setting:

- Tissue containing tumors
- Complex Stroma tissue

The 3-Label Setting:

- Tissue containing tumors
- Complex Stroma tissue
- Tissue containing immune cell conglomerates

Complex Stroma? Immune Cell Conglomerates?

Definition 1.1: Complex Stroma

The cells and tissues that support and give structure to organs, glands, or other tissues in the body. The stroma is mostly made up of connective tissue, blood vessels, lymphatic vessels, and nerves.

Definition 1.2: Conglomerate

Composed of several parts aggregated into one mass.

Definition 1.3: Immune Cell Conglomerate

An aggregated mass of immune cells.

Technical Outline

When beginning the project, I needed to figure out how the program would look at a high level.

The following is how I decided data would flow through the program:

1. Import all images from dataset into my program, probably as numpy arrays.
2. Transform the images in some way to make them easier to process.
3. Generate complexes of the data based on a filtration function.
4. Create persistence diagrams and obtain betti-0/betti-1 lifetimes of the topological objects in my cancer images.
5. Vectorize those lifetimes to make this important topological data discrete.
6. Do a 80:20 split of these vectors for training and testing, respectively.
7. Train both a Random Forest Classifier and a Gradient Boosting Classifier on the vectorized lifetimes and see which performed better.

The Structure of Our Dataset

Figure: Directory Structure of Colorectal Histology MNIST dataset

```
→  TDA-Project git:(master) ✗ tree -L 3 Colorectal-Histology-MNIST
Colorectal-Histology-MNIST
├── Kather_texture_2016_image_tiles_5000
│   └── Kather_texture_2016_image_tiles_5000
│       ├── 01_TUMOR
│       ├── 02_STROMA
│       ├── 03_COMPLEX
│       ├── 04_LYMPHO
│       ├── 05_DEBRIS
│       ├── 06_MUCOSA
│       ├── 07_ADIPOSE
│       └── 08_EMPTY
├── Kather_texture_2016_larger_images_10
│   └── Kather_texture_2016_larger_images_10
│       ├── CRC-Prim-HE-01_APPLICATION.tif
│       ├── CRC-Prim-HE-02_APPLICATION.tif
│       ├── CRC-Prim-HE-03_APPLICATION.tif
│       ├── CRC-Prim-HE-04_APPLICATION.tif
│       ├── CRC-Prim-HE-05_APPLICATION.tif
│       ├── CRC-Prim-HE-06_APPLICATION.tif
│       ├── CRC-Prim-HE-07_APPLICATION.tif
│       ├── CRC-Prim-HE-08_APPLICATION.tif
│       ├── CRC-Prim-HE-09_APPLICATION.tif
│       └── CRC-Prim-HE-10_APPLICATION.tif
├── hmnist_28_28_L.csv
├── hmnist_28_28_RGB.csv
├── hmnist_64_64_L.csv
├── hmnist_8_8_L.csv
└── hmnist_8_8_RGB.csv

12 directories, 15 files
```

Note: there are 300-800 tissue tif images under each numbered directory

Getting Our Dataset into Python!

My strategy for importing this dataset into my program is:

1. First I pulled all directory names from the dataset into python.
2. Next I checked to see if the directory name matched the 3 labels I was going to be training the models on.
3. Finally I read all the images belonging to these selected directories into python using the PIL Image library.
4. As they were being read into python, they would be placed into the appropriate "setting" array (i.e. binary or 3-label)

I wanted the process of importing data to be general enough such that one could easily change which labels are important to the program. This sacrifices some efficiency, but this generally isn't an issue as this process never takes more than a few seconds.

Image Transformations Applied

The first thing I did was gray-scale the images.

Then, I reduced their size to 28x28 pixels since originally they were 150x150 pixels.

Finally, the PIL images were converted to NumPy arrays since they are easier to work with, in a data science setting.

Note

Later on, I would come back and change this resize to 75x75 pixels, because after extensive testing I found I couldn't reduce the images further without significantly reducing the accuracy of my models. I had originally thought that 28x28 pixels would be sufficient since other machine learning models that don't utilize topological methods were able to get good results with these small images, but this was not the case when training the models on the topological data of the images.

First Approach to the Vectorization Pipeline

Finally, it was time to convert the possibly cancerous tissue images into vectors that could be consumed by my models.

The first pipeline I considered was the following:

1. Convert image into point-cloud data
2. Obtain Persistence Diagrams from Vietoris-Rips filtrations
3. Vectorize betti-0 and betti-1 lifetimes

However, no matter what vectorization methods were used (diagram.Amplitude, diagram.PersistenceEntropy, diagram.NumberOfPoints), both the Random Forest Classifier and the Gradient Boosting Classifier performed poorly.

It seemed like the conversion from image data to point-cloud data wouldn't work for my purposes. I would have to try something else.

A Better Approach

The vectorization pipeline I landed on after extensive testing is:

1. Create Persistence diagrams of the lifetimes of cubical complexes of homology dimension-0 (holes) and dimesion-1 (loops).
2. Apply Linear Scaler to diagrams (this doesn't matter as much without the density filtration).
3. Create a vector of the amplitudes of our generated lifetimes using a "persistence image" metric.

Figure: Code to create our Sklearn pipeline

```
steps = [
    ('diagram', CubicalPersistence()),
    ('rescaling', Scaler()),
    ('amplitude', Amplitude(metric='persistence_image'))
]
pipeline = Pipeline(steps)
```

Notes on This Approach

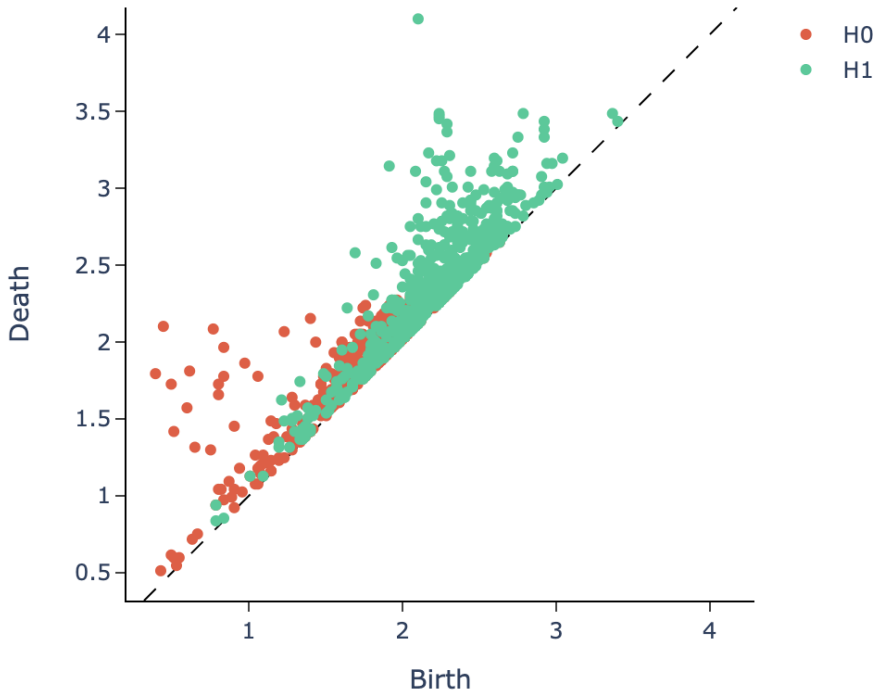
Sidenote

At first I tried applying a Density Filtration to my images before generating persistence diagrams from cubical complexes, but these performed much worse than directly generating persistence diagrams from the gray-scaled image.

Intuitively, this makes sense, since cancerous cells nearly always look "deformed" from a geometric point of view, whereas cancerous tissue is only higher density after a lot of time and when confined in a membrane preventing it from replicating outwards.

Linearly Scaled Persistence Diagrams

Figure: Linearly Scaled Persistence Diagrams



The Vectorization Process

Finally, we want to make our continuous persistence diagram discrete without losing too much information.

In the end, I decided to use an Amplitude transformer from the Giotto-TDA library. This transformer:

1. Takes in persistence diagrams
2. For each point in each diagram, calculate the amplitude of the point using a predetermined metric.
3. Collect each of these points' amplitude into a vector
4. Collect each of these vectors into a NumPy array.

I used a built-in metric called the *persistence image metric*, which is a distance between Gaussian-smoothed diagrams represented on birth-persistence axes [giotto tda, b]. My rationale for using it instead of the Bottleneck or Wasserstein metrics was purely empirical, namely, this metric far outperformed all other built-in metrics by more than 10 percent after training the models on the produced vectors.

Why Random Forest? Why Gradient Boosting?

First, it's important to understand that while we informally refer to the RandomForestClassifier and GradientBoostingClassifier as "models", they are actually a collection of models called **ensembles**.

Definition 3.1: Ensemble

The goal of [ensembles] is to combine the predictions of several base estimators built with a given learning algorithm in order to improve generalizability/robustness over a single estimator.

There are two types of ensembles, namely:

1. *averaging ensembles* - independent estimators with averaged results
2. *boosting ensembles* - sequential estimators each tasked with reducing the bias of the previous estimator

Information from Sklearn [giotto tda, a]

Hyperparameter Selection

What is a hyperparameter?

In machine learning, a hyperparameter is a parameter whose value is used to control the learning process. By contrast, the values of other parameters (typically node weights) are derived via training. [contributors, 2022]

In other words, a hyperparameter is a variable that affects the process by which the model learns.

For example: When constructing a Sklearn RandomForestClassifier object, you can pass it an argument of "max_depth" and "n_estimators" which changes the maximum depth of every tree as well as the number of trees in our forest, respectively.

Without the ability to tune hyperparameters, we can't be sure our model wouldn't have performed better than with shallower trees.

Important Hyperparameters

For the Random Forest Model, we'd like to tune:

- *max_depth* - Maximum depth of each tree.
- *n_estimators* - Number of trees in our decision forest.

For the Gradient Boosting Model, we'd like to tune all the same parameters with the addition of:

- *learning_rate* - Changes the amount each tree "contributes" to the overall classification.

Note

The higher the *learning_rate*, the lower the contribution each tree has to the final classification. There is a trade-off between *learning_rate* and *n_estimators* [giotto tda, c]

Hyperparameter Tuning In Context

Now obviously these parameters can be manually tuned, but in order to ensure the best results we want the program to find the best parameters from a list of reasonably selected parameters.

Luckily, Sklearn has a built-in feature called GridSearchCV (CV stands for cross-validation), which takes in a list of reasonable parameters and finds the parameters that maximize the accuracy of your model.

In my program, models are only trained after the ideal hyperparameters are found for each setting/model.

Figure: Simple function to apply GridSearchCV to a given model

```
14 # DESC   : Calculates the best parameters for my models
15 # Ex ret : {'max_depth' : 10, 'n_estimators': 250}
16 def calculate_best_parameters(model, parameters, X_train, y_train):
17     cv = GridSearchCV(model, parameters, cv=10, n_jobs=-1)
18     cv.fit(X_train, y_train)
19     return cv.best_params_
20
```

Other Cool Features Included in Program

Some nice features I included:

- The ability to save trained models to disk.
- The ability to import previously trained models from disk into the program.
- The ability to generate/display the diagrams and images used in this presentation.

One thing I kept finding myself needing to do was function composition, something rather annoying to do in languages without function currying. I couldn't find a standard library implementation of this basic feature, but luckily I found this nifty implementation online.

Figure: Python Function Composition

```
def compose(*fns) -> Callable[[Any], Any]:
    def inner(arg):
        for f in reversed(fns):
            arg = f(arg)
        return arg
    return inner
```

Example Program Output

Finally, after many adjustments, our program works!

Figure: Output of Model Training

```
[INFO] Printing Model Results
===== Results from Random Forest Binary Setting model ====
built-in model score: 0.8583333333333333
Area under ROC Curve: 0.8980555555555555
============================================================
===== Results from Random Forest 3-Label Setting model ====
built-in model score: 0.7111111111111111
Confusion Matrix
TP: 25 FP: 2
FN: 11 TN: 17
              precision    recall  f1-score   support

     complex       0.60      0.83      0.69        30
      lympho       0.81      0.57      0.67        30
       tumor       0.81      0.73      0.77        30

    accuracy                           0.71        90
   macro avg       0.74      0.71      0.71        90
weighted avg       0.74      0.71      0.71        90

============================================================
===== Results from Gradient Boost Binary Setting model ====
built-in model score: 0.8666666666666667
Area under ROC Curve: 0.9108333333333333
============================================================
===== Results from Gradient Boost 3-Label Setting model ====
built-in model score: 0.7
Confusion Matrix
TP: 25 FP: 2
FN: 10 TN: 17
              precision    recall  f1-score   support

     complex       0.60      0.83      0.69        30
      lympho       0.81      0.57      0.67        30
       tumor       0.78      0.70      0.74        30

    accuracy                           0.70        90
   macro avg       0.73      0.70      0.70        90
weighted avg       0.73      0.70      0.70        90

============================================================
Would you like to save these models? [y/n]: y
[INFO] Saving Models to Disk
→  TDA-Project git:(master) ✗
```

Results

Table: Binary Setting

Method	**Model Score**	**AUC ROC**
Random Forest	0.858333	0.8980555
Gradient Boosting	0.8666	0.9108333

Table: 3-Label Setting

Method	**Model Score**	**TP, TN, FN, FP**
Random Forest	0.7111	24, 17, 11, 2
Gradient Boosting	0.7	25, 17, 10, 2

Known Results

Figure: Known Results

Table 6.1: Accuracy and weighted F1 score for each dataset.

		ResNet 1D	Decision Tree	Gradient Boost	LightGBM	Random Forest	SVM	XGBoost	kNN	Related works
concrete	Accuracy	0.994	0.989	0.991	**0.9945**	0.993	0.956	0.9935	0.890	**0.999** with CNN [55]
	Weighted F1	0.994	0.988	0.989	0.994	0.992	0.955	0.993	0.884	
	run time	465.87	9.08	252.15	11.25	7.63	214.05	59.15	1.93	
mangopest	Accuracy	**0.931**	0.764	0.681	0.898	0.869	0.474	0.889	0.666	0.76 with CNN[44]
	Weighted F1	0.931	0.764	0.676	0.898	0.869	0.439	0.889	0.663	
	run time	760.94	17.17	5562.09	260.62	13.94	662.45	2041.22	2.33	
Indian fruits	Accuracy	**1.000**	0.9608	0.9608	0.9608	0.9608	0.7313	0.9608	0.676	0.999 SVM with deep features [5]
	Weighted F1 score	1.000	0.9608	0.9608	0.9608	0.9608	0.7236	0.9608	0.656	
	run time	271.21	4.44	4265.09	82.55	4.13	72.73	451.65	1.18	
Fashion MNIST	Accuracy	0.7427	0.567	0.696	**0.749**	0.693	0.535	0.746	0.397	**0.99** with CNN [38]
	Weighted F1	0.7414	0.569	0.694	0.749	0.692	0.529	0.746	0.390	
	run time	467.12	8.36	1808.07	89.37	7.66	935.21	1108.20	3.38	
APTOS	Accuracy	0.7326	0.698	0.760	**0.787**	0.782	0.674	0.775	0.655	**0.971** with CNN [66]
	Weighted F1	0.667	0.695	0.737	0.771	0.757	0.591	0.764	0.637	
	run time	61.81	0.63	86.02	13.16	0.70	3.49	42.34	0.08	
colorectal histology	Accuracy	**0.892**	0.75	0.842	0.869	0.855	0.679	0.874	0.759	0.874 with SVM[37]
	Weighted F1	0.89	0.727	0.832	0.850	0.834	0.686	0.843	0.743	
	run time	86.23	1.18	255.08	12.52	1.10	4.06	44.63	0.14	
	Accuracy	**0.882±0.109**	0.789±0.147	0.822±0.121	0.876±0.087	0.856±0.102	0.675±0.154	0.873±0.90	0.674±0.148	
	Weighted F1	**0.871±0.125**	0.784±0.148	0.815±0.124	0.870±0.091	0.851±0.105	0.654±0.164	0.866±0.092	0.662±0.147	

Data provided by [Choe and Ramanna, 2022]

Thoughts on Results

The areas under the binary setting ROC curve are around 0.9, which leads me to believe it's highly likely the displayed scores accurately represent the prediction power of my models in general.

Also, after running a few times it seems like the Random Forest model is better in the binary setting whereas the gradient-boosted decision tree model works better in the 3-label setting.

Something I don't like about my 3-label results is how my models tend towards false negatives in the case that they fail. This is a big issue in a clinical setting as we'd rather spend the resources to perform tests on a patient that doesn't have cancer than have patients with cancer slip through the cracks. However, there could be contexts where we want to be very certain we only get images with cancer, in these contexts we'd prefer false negatives to false positives.

Final Thoughts and Conclusion

One area I could try to explore to increase the accuracy of classification is creating my own ensembles. Currently, the program trains 2 ensembles that are provided by Sklearn, but I'm sure some data exploration could lead to some insights as to how these ensembles could be redesigned.

So to conclude, I'd like my 3-Label setting accuracy to be a bit better. However, I'm fairly happy with these results overall, especially with my accuracy in the binary setting.

References

Choe, S. and Ramanna, S. (2022).
Cubical homology-based machine learning: An application in image classification.
Axioms, 11(3).

contributors, W. (2022).
Hyperparameter (machine learning).
https://en.wikipedia.org/w/index.php?title=Hyperparameter_(machine_learning)&oldid=1111121452.

giotto tda.
Ensemble methods.
https://scikit-learn.org/stable/modules/ensemble.html.

giotto tda.
Persistence image.
https://giotto-ai.github.io/gtda-docs/0.5.1/modules/generated/diagrams/representations/gtda.diagrams.PersistenceImage.html.

giotto tda.
sklearn.ensemble.gradientboostingclassifier.
https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.GradientBoostingClassifier.html.

Gultekin, T., Koyuncu, C., Sokmensuer, C., and Gunduz-Demir, C. (2014).
Two-tier tissue decomposition for histopathological image representation and classification.
IEEE transactions on medical imaging, 34.

K Scott Mader, U. M.
Colorectal histology mnist.
https://kaggle.com/datasets/kmader/colorectal-histology-mnist.

Thanks!

grayson.croom@gmail.com