**Diagnosis of COVID-19 from Chest CT Using Decision Trees**

IES 5194 – Human-Centered Machine Learning

Spring 2021

|  |  |
| --- | --- |
| **Katie Albert** | **Matt Boraske** |

Abstract

The progression of the COVID-19 pandemic has prompted advancements in machine learning applications in medicine, particularly for disease detection and diagnosis. While many state-of-the-art approaches achieve high levels of accuracy, they fall short in their interpretability. Because model interpretability is critical for effective human-automation interaction, we sought to demonstrate that comparable levels of accuracy could be achieved by a more interpretable approach. To do this, we designed a decision tree model that classified chest CT images for the presence of COVID-19 related infection. Our model achieved almost 80% accuracy, demonstrating that moderate accuracy levels in COVID-19 diagnosis are feasible with a relatively simple and highly interpretable model.

# Introduction

## Objective

The objective of this project is to develop an explainable supervised learning agent that classifies chest CT images based on whether that scan came from a patient infected with COVID-19. Due to the ongoing pandemic there has been extensive effort in investigating machine learning applications for diagnosis of COVID-19 and COVID-19 related infections. While point-of-care diagnostic tools such as reverse transcription polymerase chain reaction (RT-PCR) are becoming more widely applied, such tests often present low sensitivities and high false negative rates. Diagnosis via computed tomography (CT) offers an increased sensitivity and ability to detect the presence of disease in asymptomatic patients or patients with a negative RT-PCR (Harmon et al., 2020).

## Current Practices

Current approaches mainly apply deep learning algorithms, and many have so far achieved accuracies over 90%. However, the challenge with many of the approaches used is that they lack interpretability. Even if the algorithms or decision processes can be described in mathematical or other terms, this does not always imply that the users of these machine learning agents can make sense of the decisions it makes.

## Our Approach

With this project we seek to develop a more interpretable machine learning agent that can achieve comparable accuracy in classifying COVID-19 from chest CT images. To do this we design a learning agent that uses a decision tree classifier. We believe this is an effective approach to making a more trustworthy machine learning agent because the mechanics of decision trees are easily visualized, making them highly interpretable for humans.

## Implications

High accuracy is critical in domains such as disease diagnosis, but being able to trust the agent that is making those decisions is necessary for that accuracy to matter. Using a machine learning agent that is easily interpreted by humans can be an effective way to create that trust.

## Measures of Success

We expect that with this approach we may not immediately achieve the accuracy level of some of the more complex deep learning methods especially within the timeline we had for this project, but for the model to be successful it must demonstrate better accuracy than what could be achieved by random classification. Our goal is to demonstrate that it is possible to create a moderately successful classifier with a relatively interpretable algorithm. We consider an accuracy level of above 75% to indicate a successful initial implementation. This is below what many of the deep learning methods have achieved but still demonstrates intelligence in the image classification. We also believe that with more development time, more sophisticated image preprocessing could be explored to achieve levels of accuracy close to those more complex algorithms.

# Literature Review

Here we present a survey of literature regarding some of the state-of-the-art machine learning applications for COVID-19 diagnosis as well as the significance of interpretable AI. The data for our study comes from a set made publicly available by Soares et al. (2020). The dataset includes 1252 CT scans from 60 patients infected with COVID-19 and 1230 scans from 60 patients not infected with COVID-19. All CT scans were sourced from hospitals in Sao Paulo, Brazil. The authors who sourced the data apply an explainable deep neural network and note that this approach could achieve an accuracy of 97% which was higher than many other state-of-the-art methods they also tested. They also tested a decision tree with this dataset and achieved an accuracy level of 79.44%. As this was not a peer-reviewed study, we use their results as a frame of reference rather than a benchmark for our own success. We also recognize that this dataset does not fully represent the demographics that may be infected by COVID-19 and that by training an agent on data from one geographic location there is potential for bias in that agent.

The study by Harmon et al. (2020) offers an example of desirable design characteristics for this type of model. Their study uses a multinational dataset in their application of a series of deep neural networks to segment and classify chest CT scans based on whether they presented COVID-19 related pneumonia. This study includes data from Europe, Asia, and the U.S., which the researchers note was done intentionally to increase the model’s applicability to a diverse array of populations. The array of conditions presented in this dataset is also diverse, including CT scans from patients with COVID-19 related pneumonia, with non covid-related pneumonia, undergoing cancer screenings, or undergoing general evaluations. The model is able to distinguish COVID-19 pneumonia from these other conditions with up to 90.8% accuracy. Because the model is trained on data from a variety of demographics and on diseases that can have similar presentations in the lungs, we felt this accuracy to be a reliable benchmark for what state-of-the-art models are capable of achieving. The design decisions for this model ensure that it can be more sensitive to characteristics of COVID-19 pneumonia and widely applicable across populations affected by COVID-19.

In order to create a machine learning model that is interpretable, we must define interpretability and identify its significance in machine learning applications. Our definition of interpretability follows that of Roscher et al., (2020), who explicitly distinguishes *interpretability*, *transparency*, and *explainability* as separate but all important concepts in the design of machine learning agents. They define interpretability as the model’s ability to be understood by a human, whereas transparency is the degree to which the model structure and learning algorithm can be described, e.g. as mathematical formulas. The authors note that explainability is being discussed in the machine learning community more frequently, but how explainability is defined tends to vary. Their definition of explainability is the agent’s ability to reveal its goals and justification for its decisions to the user. While all three concepts are core to human-centered design of machine learning agents, our approach described in this paper focuses only on investigating the interpretability of the agent.

Interpretability of machine learning agents is becoming increasingly important as the applications of machine learning in healthcare expand. Decision making in medicine is critical because it directly affects the health and well being of human patients. Ahmad et al. (2018) notes that many healthcare providers view interpretability of the model as a priority for a machine learning agent to be implemented in their practice. The users of decision making models in medicine may already have significant knowledge of the domain and will therefore understand the implications of their decisions. For users to trust the model that aids them in making these decisions, they need to understand the model’s structure and how it maps to the domain they are working in. Model interpretability also impacts medicine beyond individual patient care. Holzinger et al. (2017) states that black-box approaches to machine learning make it difficult to ensure privacy of patient data. When users understand how the model uses patient data, they can better facilitate the ethical management and distribution of that information.

# Connection to HCI

In safety-critical domains such as medicine it is important that automated agents interact effectively with humans. A healthcare provider’s trust in the decisions made by a machine learning agent is critical in medicine because these decisions can directly impact patient outcomes. Because clinicians are domain experts, they need to understand how the agent makes its decisions and recognize when the decision making process may not be appropriate for the domain. Machine learning agents can enhance much of the work done by clinicians by speeding up processing time and reducing error, but for these tools to be useful they need to be trusted by the human experts they are aiding. While these are not the only characteristics of effective human-machine interaction, interpretability, transparency, and explainability may be used to evaluate a machine learning agent’s ability to establish trust with its users (Roscher et al., 2020).

Of the three characteristics previously listed, this project focuses on interpretability. Interpretability refers to how well the machine learning agent’s algorithm, decision process, or model can be understood by human users (Roscher et al., 2020). We explore an alternative to the deep learning approaches that have thus far been applied in classifying chest CT for COVID-19. While many of those approaches could achieve high levels of accuracy, they possess a relatively low level of interpretability. Our approach uses decision trees which are highly interpretable classification models. Implementing this decision tree model in a healthcare domain could effectively aid in diagnosis of COVID-19 related infection because users can understand how the model functions and know when to trust in the decisions it makes.

# Methodology

The dataset from Soares et al. (2020) used to train the decision tree model consisted of 2,924 CT scan images from 120 patients, 80 of which had covid-19 and 40 were healthy. Each patient had a folder that contained their CT scan images.

The process of developing the decision tree model can be broken into the following three sections: image preprocessing, training of the decision tree model, and then comparing it to a random forest model to analyze how well this simple model compares to a more sophisticated ensemble one. The Python code developed to implement all of this can be found in Appendix A.

# 4.1 Image Preprocessing

In order for the data to be used to train a decision tree model, each picture had to be transformed into an array of RGB values for each pixel. This was accomplished through the following steps:

1. All file paths to folders of images for all patients were inserted into an array.
2. To ensure that all of a patient’s images are included in either the training or testing set, the folders were split into 80% for training and 20% for testing.
3. This array was iterated through to convert each image in each folder into an array of pixel RGB values by completing the following substeps
   1. To standardize the image resolution, each was resized to 280x200
   2. Converted the images into RGB tensors
   3. Flattened these tensors into matrices
   4. For each picture arrays of the RGB values of each pixel were created by extracting only the red value for each pixel. This is valid because the images are grayscale and therefore all RGB values are equivalent.

Following this, each of these arrays were added as rows to a matrix so that the image data was now a matrix with 2924 rows and 56,000 columns since this was the total number of images and number of pixels per picture, respectively. At this point, the data was ready to be used to train the decision tree model.

# 4.2 Training of Decision Tree Model

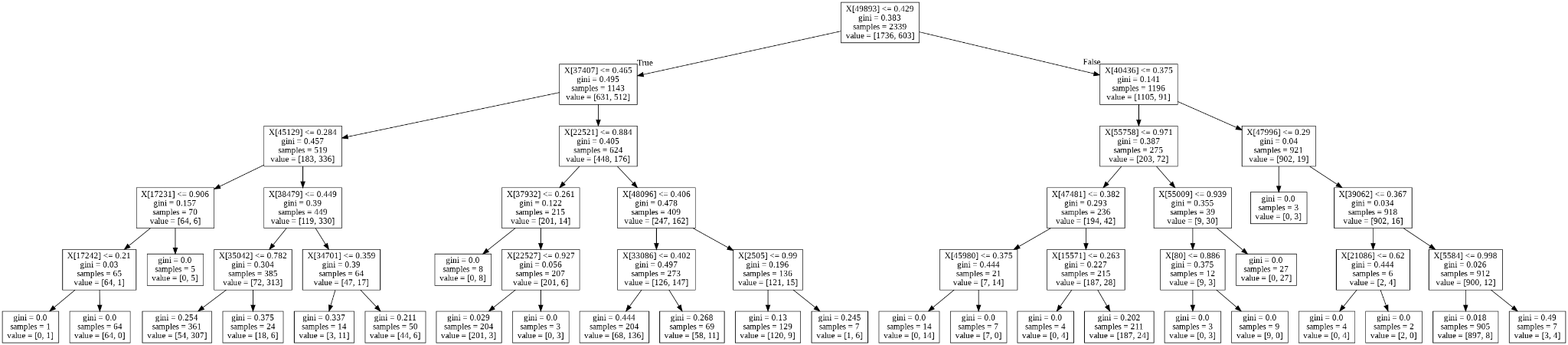
The first step to training the decision tree model was to conduct parameter tuning in order to identify the optimal max depth of the tree and max number of features to consider when creating a new split in the tree. Depths of three, five, seven, and nine were considered as well as a maximum of 10, 50, 100, 500, 1000, and 2,000 features. For each set of parameters, a decision tree was created and trained using the DecisionTreeClassifier class provided in the sklearn.tree Python module and the one that returned the best accuracy was determined to be the optimal. A confusion matrix for the optimal tree was created to analyze its performance and frequency of producing either false positives or negatives. The optimal tree was then visualized by saving it as a PNG file.

# 4.3 Comparison to Random Forest Model

A random forest consisting of 100 trees was trained to analyze the performance of our decision tree against more sophisticated machine learning models. To determine the optimal max depth of trees in the random forest, ones with depths ranging one to nine were created and trained using the RandomForestClassifier class provided in the sklearn.ensemble Python module and their accuracies was compared against one another. The optimal random forest was selected as the one with the greatest accuracy. In order to visualize the change in accuracy as the max depth of the trees in the random increased, the accuracies of the aforementioned random forests were plotted.

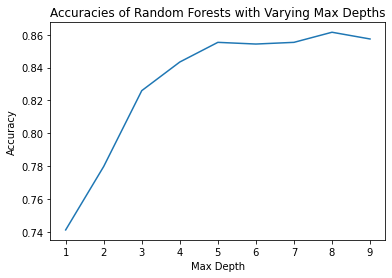
# Results & Discussion

The optimal decision tree was determined to have a max depth of five and max number of features to consider when creating a new split in the tree of 2,000. It achieved an accuracy of 79.86% and it is shown in Figure 1. The accuracies for all the decision trees for various maximum depths and features are displayed in Table 1. The optimal random forest was determined to have a maximum depth of eight. It achieved an accuracy of 86.15% and Figure 2 visualizes the change in accuracy with respect to the maximum depth.

**Figure 1: Optimal Decision Trees**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Max Depth** | **Max Features** | | | | | |
| **10** | **50** | **100** | **500** | **1,000** | **2,000** |
| **3** | 74.83% | 75.24% | 76.67% | 77.29% | 76.06% | 77.60% |
| **5** | 75.79% | 77.83% | 78.15% | 79.14% | 79.11% | 79.86% |
| **7** | 76.10% | 77.46% | 78.52% | 78.25% | 78.70% | 78.79% |
| **9** | 76.95% | 77.40% | 78.59% | 77.46% | 78.86% | 78.80% |

**Table 1: Accuracies of Decision Tree for Various Maximum Depths and Features**



**Figure 1: Accuracies of Random Forests for Varying Max Depths**

The implementation of the decision tree model was deemed successful because it achieved the goal of 75% accuracy. When comparing the accuracies of the decision tree and random forest models, it was found that the latter only provided an improvement of 6.29%, which was less than expected for the significantly increased complexity. Due to this, it was determined that the decision tree was successfully able to sacrifice a relatively small amount of accuracy for a more significant improvement in its interpretability. While decision trees can be visualized like in Figure 1 which makes it easy to understand its artificial thought process, it is challenging to create a visual of the entire random forest that also elucidates its decision-making. Overall, the objective of developing an interpretable AI that is also able to diagnose Covid-19 from CT scan images to a reasonably accurate level was accomplished.

# Conclusion

In this project we constructed a decision tree classifier that could identify COVID-19 patients from chest CT scans with about 80% accuracy. Our goal in the design of this machine learning agent was to achieve at least 75% accuracy with a highly interpretable model. It was important for us to demonstrate that moderate accuracy could still be achieved with a relatively simple model because many of the existing approaches to this problem have low interpretability, and interpretability is key for the user to be able to trust the decisions made by a machine learning agent.

We consider this initial implementation of a decision tree classifier to be successful, but we also believe that there is potential for even higher levels of accuracy while maintaining the same level of interpretability. Better accuracy might be achieved with more sophisticated image preprocessing such as image segmentation.

We also recognize that due to the lack of publicly available data our model has not been trained on a dataset that is fully representative of the domain. Because of this, our model may not be applicable to a wide range of demographics. An improvement for the future would be to train our model on a multinational dataset to reduce the risk of bias and ensure the model is applicable to any person who can be infected by COVID-19.

# Acknowledgments & Team Contributions

The authors would like to thank Dr. Samantha Krening for her knowledge and guidance on this project and throughout the semester.

As the team only consisted of two individuals, both worked closely together on the written and programming portions of the project. Katie Albert led the completion of the literature review as well as the development of connections to human-computer interaction (HCI). Matt Boraske led the development of the Python code to implement the decision tree and random forest models.

# References

Ahmad, M. A., Eckert, C., & Teredesai, A. (2018). Interpretable Machine Learning in Healthcare. *ACM-BCB ’18: 9th ACM Int’l Conf. on Bioinformatics, Computational Biology, and Health Informatics*.

Harmon, S. A., Sanford, T. H., Xu, S., Turkbey, E. B., Roth, H., Xu, Z., Yang, D., Myronenko, A., Anderson, V., Amalou, A., Blain, M., Kassin, M., Long, D., Varble, N., Walker, S. M., Bagci, U., Ierardi, A. M., Stellato, E., Plensich, G. G., … Turkbey, B. (2020). Artificial intelligence for the detection of COVID-19 pneumonia on chest CT using multinational datasets. *Nature Communications*, *11*(1), 1–7. https://doi.org/10.1038/s41467-020-17971-2

Holzinger, A., Biemann, C., Pattichis, C. S., & Kell, D. B. (2017). What do we need to build explainable AI systems for the medical domain? *ArXiv*, *Ml*, 1–28.

Roscher, R., Bohn, B., Duarte, M. F., & Garcke, J. (2020). Explainable Machine Learning for Scientific Insights and Discoveries. *IEEE Access*, *8*, 42200–42216. https://doi.org/10.1109/ACCESS.2020.2976199

Soares, E., Angelov, P., Biaso, S., Froes, M. H., & Abe, D. K. (2020). SARS-CoV-2 CT-scan dataset: A large dataset of real patients CT scans for SARS-CoV-2 identification. *MedRxiv*, 1–8. https://doi.org/10.1101/2020.04.24.20078584

**Appendix A**

Python Code

"""

Covid\_Diagnosis\_Using\_CT\_Scans.ipynb

Original file is located at

https://colab.research.google.com/drive/1SNVmBfae\_qTZrQ70GZ0dyfv4TSDyzVNT

"""

### Classification of CT scan images as either covid or healthy ###

#installation of pillow for image resizing

!pip install --upgrade pip

!pip install --upgrade Pillow

#Required libraries

import numpy as np

import scipy

import numpy as np

import matplotlib.pyplot as plt

from matplotlib import image

from sklearn.model\_selection import train\_test\_split

from sklearn.datasets import load\_digits

from sklearn.ensemble import RandomForestClassifier

import glob

import os

from PIL import Image

import zipfile

##Mounting of google drive to get data

from google.colab import drive

drive.mount('/content/drive')

##Unzipping of data

zip\_ref = zipfile.ZipFile("/content/drive/MyDrive/HCML\_project/archive.zip",'r')

zip\_ref.extractall("/content/dataset")

zip\_ref.close()

##Converts PNG to RGB matrix

def image\_to\_matrix(image\_file, grays=False):

img = image.imread(image\_file)

if(len(img.shape) == 3 and img.shape[2] > 3):

height, width, depth = img.shape

new\_img = np.zeros([height, width, 3])

for r in range(height):

for c in range(width):

new\_img[r,c,:] = img[r,c,0:3]

img = np.copy(new\_img)

if(grays and len(img.shape) == 3):

height, width = img.shape[0:2]

new\_img = np.zeros([height, width])

for r in range(height):

for c in range(width):

new\_img[r,c] = img[r,c,0]

img = new\_img

if(len(img.shape) == 2):

zeros = np.where(img == 0)[0]

img[zeros] += 1e-7

return img

##flatten RGB matrix

def flatten(image\_matrix):

if(len(image\_matrix.shape) == 3):

height, width, depth = image\_matrix.shape

else:

height, width = image\_matrix.shape

depth = 1

flattened\_values = np.zeros([height\*width,depth])

for i, r in enumerate(image\_matrix):

for j, c in enumerate(r):

flattened\_values[i\*width+j,:] = c

oneDim = []

for pixel in range(len(flattened\_values)):

for RGB\_value in range(3):

if RGB\_value == 0:

oneDim.append(flattened\_values[pixel][RGB\_value])

return np.array(oneDim)

##Resize image function

def resize\_image(image\_file, width, height):

image = Image.open(image\_file)

new\_image = image.resize((width, height))

new\_image.save(image\_file)

return image\_file

##reading in CT images and converting them to RGB matrices

from PIL import Image

#getting patient folders (both covid and healthy)

patient\_folders = []

target=[]

for filepath in glob.glob(os.path.join('/content/dataset/New\_Data\_CoV2/Covid', '\*')):

patient\_folders.append(filepath)

target.append("Covid")

healthy\_patient\_folders = []

for filepath in glob.glob(os.path.join('/content/dataset/New\_Data\_CoV2/Healthy', '\*')):

patient\_folders.append(filepath)

target.append("Healthy")

#convert folder lists to arrays

patient\_folders = np.array(patient\_folders)

#split the folders in training/testing sets

folder\_train, folder\_test, y\_train, y\_test = train\_test\_split(patient\_folders, target, test\_size = 0.2, random\_state = 0)

#Convert training images to matrices and add them to a list of them

print("Beginning conversion of training images to matrices")

train\_image\_matrix = []

train\_target = []

for i in range(len(folder\_train)): ##len(folder\_train)

folder\_filepath = folder\_train[i]

category = y\_train[i]

for png\_filepath in glob.glob(os.path.join(folder\_filepath, '\*.png')):

img\_filepath = resize\_image(png\_filepath,280,200)

train\_image\_matrix.append(flatten(image\_to\_matrix(img\_filepath)))

train\_target.append(category)

print("\tAdding images folder:", folder\_filepath)

#Convert testing images to matrices and add them to a list of them

print("Beginning conversion of testing images to matrices")

test\_image\_matrix = []

test\_target = []

for i in range(len(folder\_test)):

folder\_filepath = folder\_test[i]

category = y\_test[i]

for png\_filepath in glob.glob(os.path.join(folder\_filepath, '\*.png')):

img\_filepath = resize\_image(png\_filepath,280,200)

test\_image\_matrix.append(flatten(image\_to\_matrix(img\_filepath)))

test\_target.append(category)

print("\tAdding images folder:", folder\_filepath)

\_

#convert image\_matrix lists to arrays

train\_image\_matrix = np.array(train\_image\_matrix)

test\_image\_matrix = np.array(test\_image\_matrix)

train\_target = np.array(train\_target)

test\_target = np.array(test\_target)

print("Conversion of images to matrices is complete")

#print that conversion process is complete and size of the training and testing image matrices

print("Training images matrix dimensions:", train\_image\_matrix.shape)

print("Testing images matrix dimensions:", test\_image\_matrix.shape)

print("Training images target matrix dimensions:", train\_target.shape)

print("Testing image target matrix dimensions:", test\_target.shape)

###ML MODELS###

##Single Decision Tree Classifier

from sklearn.tree import DecisionTreeClassifier

from sklearn.metrics import confusion\_matrix

from sklearn import tree

from sklearn.metrics import accuracy\_score

from sklearn.tree import export\_graphviz

import graphviz

from sklearn.model\_selection import cross\_val\_score

from sklearn.model\_selection import cross\_val\_predict

#tuning parameters of decision tree by trying different combos of max\_depth and max\_features

depths = [3,5,7,9]

feature\_amts = [10,50,100,500,1000,2000]

best\_accuracy = 0

best\_depth = 0

best\_feature\_amt = 0

for depth in depths:

for feature\_amt in feature\_amts:

#create decision tree

print("Decision tree where max depth =", depth, "and max features =", feature\_amt)

dt = DecisionTreeClassifier(max\_depth=depth, max\_features=feature\_amt)

#train decision tree

dt.fit(train\_image\_matrix, train\_target)

#create predictions using decision tree

y\_pred = dt.predict(test\_image\_matrix)

#accuracy

print("\tAccuracy =", accuracy\_score(test\_target, y\_pred)

if accuracy\_score(test\_target, y\_pred) > best\_accuracy:

best\_accuracy = accuracy\_score(test\_target, y\_pred)

best\_depth = depth

best\_feature\_amt = feature\_amt

print("\nBest Accuracy =", best\_accuracy)

print("\tMax Depth =", best\_depth)

print("\tMax features =", best\_feature\_amt)

##recreate decision tree using best found configuration of parameters (aka the optimal decision tree)

dt = DecisionTreeClassifier(max\_depth=5, max\_features=2000)

#train decision tree

dt.fit(x\_train, y\_train)

#create predictions using decision tree

y\_pred = dt.predict(x\_test)

#confusion matrix

print(confusion\_matrix(y\_test, y\_pred))

#accuracy

acc = accuracy\_score(y\_test, y\_pred)

print("Accuracy of optimal decision tree =", acc)

#save optimal decision tree to a .png file

export\_graphviz(dt, out\_file="mytree.dot")

with open("mytree.dot") as f:

dot\_graph = f.read()

graphviz.Source(dot\_graph)

#convert .dot to .png and save it

!dot mytree.dot -Tpng -o NewDecisionTree.png

##Random Forest Classifier

from sklearn.ensemble import RandomForestClassifier

##determine what max depth for RF trees achieves best accuracy

best\_acc = 0

best\_depth = 0

depths = [1,2,3,4,5,6,7,8,9]

accuracies = []

for depth in depths:

#create and train RF

print("\nTraining random forest with max depth of",depth)

rf = RandomForestClassifier(max\_depth=depth, random\_state=0)

#train decision tree

scores = cross\_val\_score(rf, image\_matrices, image\_targets, cv=5)

#accuracy

print("\tAccuracy =", scores.mean())

accuracies.append(scores.mean())

if scores.mean() > best\_acc:

best\_acc = scores.mean()

best\_depth = depth

print("\nBest accuracy achieved is", best\_acc, "using a max depth of", best\_depth)

##plot accuracies over max depth

x\_vals = range(1,10)

plt.plot(x\_vals,accuracies)

plt.title("Accuracies of Random Forests with Varying Max Depths")

plt.xlabel("Max Depth")

plt.ylabel("Accuracy")