CSCE 633 Homework 5

Designing and disseminating ML for a real-world problem

Team 10: Sanjeevani Choudhery, Samantha Ray, Sournav Bhattacharya, Rohan Singh Wilkho, and Mounika Kunduru

a.(1 point) Image pre-processing.

The images that are provided in the data are of different sizes. Crop the images to a square shape (i.e., image length equals image width) and resize the cropped image into a fixed size.

We read the images and the label CSV and ensure that the images are stored in the same order as the CSV. Our preprocessing pipeline is as follows: read file as grayscale, center crop, resize to 200x200, morph holes closed (dilation followed by erosion), and finally sharpening the image.

```python
import cv2
from google.colab.patches import cv2_imshow

def center_crop(img):
  # Crop the image to a square based on its smaller dimension
  # Params: image file
  # Returns: square image file
  height, width  = img.shape[0], img.shape[1]
  mid_h = height//2
  mid_w = width//2
  s = height//2 if height < width else width//2 # half of the size of the smaller dimension
  return img[mid_h-s:mid_h+s,mid_w-s:mid_w+s]

def preprocess(img):
  return sharpen(morph_close(img))

def morph_close(img):
  kernel = np.ones((3,3),np.uint8)
  return cv2.morphologyEx(img, cv2.MORPH_CLOSE, kernel)

def sharpen(img):
  # Note: the 5 was determined empirically. Making it a 2 or 10 turned the image black
  kernel = np.array([[0, -1, 0],
                     [-1, 5,-1],
                     [0, -1, 0]])
  return cv2.filter2D(src=img,ddepth=-1,kernel=kernel)

##############################
# Part (a) - Data Preprocessing
##############################
#THIS IS THE TARGET WIDHT/HEIGHT AFTER RESIZING
tgt_h = 200
tgt_w = 200

# We crop the images to be a square, and resize them to a target width/height
# Test images are stored in imgs_test, and train in imgs_test as numpy arrays
imgs_train = [cv2.resize(center_crop(cv2.imread(train_data_path+'/'+i,cv2.IMREAD_GRAYSCALE)),(tgt_h,tgt_w)) for i in tqdm(os.listdir(train_data_path))]
imgs_test = [cv2.resize(center_crop(cv2.imread(test_data_path+'/'+i,cv2.IMREAD_GRAYSCALE)),(tgt_h,tgt_w)) for i in tqdm(os.listdir(test_data_path))]

# Sort the image files by filename
imgs_train_filenames = os.listdir(train_data_path)
imgs_train_sync = sorted(zip(imgs_train_filenames, imgs_train), key=lambda x:x[0])
imgs_train = [x[1] for x in imgs_train_sync]

# Sort the labels by filename
lbls_train = pd.read_csv(train_labels)
lbls_filenames = lbls_train["filename"].values
lbls_train = lbls_train["covid(label)"].values
lbls_sync = sorted(zip(lbls_filenames, lbls_train), key=lambda x:x[0])
lbls_train = np.asarray([x[1] for x in lbls_sync])

# Further preprocess the images
imgs_train = [preprocess(img) for img in imgs_train]
imgs_test = [preprocess(img) for img in imgs_test]
```

Figure 1. Image Preprocessing Code.

(b.i) (2 points) Visual feature extraction.

Extract image features, which will be used to predict the COVID-19 diagnosis. Please describe the features and provide a brief justification on how these features might work well for the outcome of interest.

Healthy lungs will appear dark and clear in x-rays. Abnormalities in the lungs will show up white and can have clump-like, spotty, or vein-y appearances. Other body parts such as bones, the heart, and the intestines will also appear white in the x-ray. As such, we need to extract features that can recognize what a healthy x-ray looks like and detect any abnormalities without getting false positives on other body parts.

We used standard computer vision feature extraction methods to understand the content of the image, e.g., the location of the various body parts and edges present in the image. These features include: Histogram of Gradients, Gabor Filter, Canny Edge Detection, Laplace Filter, Merijering Filter, Hessian Filter, and Sobel Filter (Vertical and Horizontal). Knowing that symptoms of COVID appear white and healthy lungs appear black, we also engineered some simple features to detect light and dark patches of pixels. Additionally, we included the demographic features supplied with the labels, encoding the qualitative data as one-hot vectors.

The computer vision features produce 200x200 matrices (the size of our input images) which results in too many features to create a generalizable model. To address this, we train a PCA dimensionality reduction model for each type of feature to compress the features into a more reasonable number. We tested different values for the number of components, i.e., 5, 10, and 20, and settled on 20.

To generate all of the features, we developed a modular feature generator that produces a dataframe of features. Each of the feature functions and the class code are included in Figures 2-11.

```
##########################################################################
# Filter features
##########################################################################

def calculate_canny(img):
    # Calculuates Canny edge detection for img
    # Params: img - the preprocessed image (2D)
    # Returns: feature vector (1D), edge map (2D)
    canny_img = cv2.Canny(img,100,200)
    return canny_img.flatten(), canny_img
```

Figure 2. Canny Edge Detection.

```python
def calculate_gabor(img,
                    theta = np.pi/2,
                    k_size = (8,8),
                    sigma = 10.0,
                    gamma = 4,
                    psi = 0.5):
    # Applies Gabor to an image
    # Params:
    #   Req: img - the preprocessed image (2D)
    #   Opt: theta, k_size,sigma,gamma,psi
    # Returns: image with gabor applied to it
  g_kernel = cv2.getGaborKernel(k_size, 1, theta, sigma, gamma, psi, ktype=cv2.CV_32F)
  filtered_img = cv2.filter2D(img, cv2.CV_8UC3, g_kernel)
  return filtered_img.flatten(), filtered_img

def calculate_gabor15(img):
  return calculate_gabor(img,theta=np.pi/8)

def calculate_gabor45(img):
  return calculate_gabor(img,theta=np.pi/4)

def calculate_gabor75(img):
  return calculate_gabor(img,theta=3*np.pi/8)

def calculate_gabor90(img):
  return calculate_gabor(img,theta=np.pi/2)

def calculate_gabor105(img):
  return calculate_gabor(img,theta=5*np.pi/8)

def calculate_gabor135(img):
  return calculate_gabor(img,theta=3*np.pi/4)

def calculate_gabor165(img):
  return calculate_gabor(img,theta=7*np.pi/8)

def calculate_gabor180(img):
  return calculate_gabor(img,theta=2*np.pi)
```

Figure 3. Gabor Filter.

```python
def calculate_hog(img, _orientation=8, _pixels_per_cell=16, _cells_per_block=3,multi=False):
    # Calculuates HoG features for img
    # Params:
    #   Req: img - the preprocessed image (2D)
    #   Opt: _orientation, _pixels_per_cell, _cells_per_block
    # Returns: feature vector (1D), gradient image (2D)

    X, hog_img = hog(img,
                     orientations=_orientation,
                     pixels_per_cell=(_pixels_per_cell, _pixels_per_cell),
                     cells_per_block=(_cells_per_block, _cells_per_block),
                     feature_vector=True,
                     visualize=True,
                     multichannel=multi)
    hog_img *= 255.0/np.max(hog_img)
    return X, hog_img

def calculate_hog16(img):
  return calculate_hog(img,_pixels_per_cell=16)
def calculate_hog8(img):
  return calculate_hog(img,_pixels_per_cell=8)
def calculate_hog4(img):
  return calculate_hog(img,_pixels_per_cell=4)
```

Figure 4. Histogram of Gradients.

```python
################################################################################
# Filter Functions - Stub functions call calculate_filter
################################################################################

def calculate_filter(img, func):
  # Processes the image using the filter passed in func
  # Params: img - the preprocessed image (2D)
  # Returns: feature vector (1D), edge map (2D)
  edges = func(img)
  edges *= 255.0/np.max(edges)
  return edges.flatten(), edges

def calculate_hessian(img):
  return calculate_filter(img,hessian)

def calculate_meijering(img):
  return calculate_filter(img,meijering)

def calculate_laplace(img):
  return calculate_filter(img,laplace)

def calculate_sobelh(img):
  return calculate_filter(img,sobel_h)

def calculate_sobelv(img):
  return calculate_filter(img,sobel_v)
```

Figure 5. Filter Features Including Hessian, Laplace, Meijering, and Sobel.

```
##########################################################################
# Feature Engineering - Context from Filters
##########################################################################

white_threshold = 220
dark_threshold = 50

def canny_white(img):
  # Returns the percentage of the image is white after applying canny filter
  _, _img = calculate_canny(img)
  return np.asarray([np.sum(np.where(_img >= white_threshold, 1, 0))/_img.size])

def laplace_white(img):
  # Returns the percentage of the image is white after applying laplace filter
  _, _img = calculate_laplace(img)
  return np.asarray([np.sum(np.where(_img >= white_threshold, 1, 0))/_img.size])

def hog_white(img):
  # Returns the percentage of the image is white after applying HoG filter
  _, _img = calculate_hog(img)
  return np.asarray([np.sum(np.where(_img >= white_threshold, 1, 0))/_img.size])

def hessian_white(img):
  _, _img = calculate_hessian(img)
  return np.asarray([np.sum(np.where(_img >= white_threshold, 1, 0))/_img.size])

def meijering_white(img):
  _, _img = calculate_meijering(img)
  return np.asarray([np.sum(np.where(_img >= white_threshold, 1, 0))/_img.size])

def blobdog(img):
  return np.array([blob_dog(img).shape[0]],ndmin=1)
def blobdoh(img):
  return np.array([blob_doh(img).shape[0]],ndmin=1)
def bloblog(img):
  return np.array([blob_log(img).shape[0]],ndmin=1)
```

Figure 6. Domain-Specific Features (Pt. 1). These features further quantify how many edges and blobs the other computer vision features detected.

```
def raw_white(img):
  return np.asarray([np.sum(np.where(img >= white_threshold, 1, 0))/img.size])
def raw_dark(img):
  return np.asarray([np.sum(np.where(img <= 100, 1, 0))/img.size])
def raw_black(img):
  return np.asarray([np.sum(np.where(img <= 50, 1, 0))/img.size])

def raw_mean(img):
  return np.asarray([np.mean(img)])
def center_mean(img):
  _img = img[img.shape[0]//4:3*img.shape[0]//4,:img.shape[1]//4:3*img.shape[1]//4]
  return np.asarray([np.mean(_img)])

def left_mean(img):
  _img = img[:img.shape[0]//2,:]
  return np.asarray([np.mean(_img)])
def right_mean(img):
  _img = img[img.shape[0]//2:,:]
  return np.asarray([np.mean(_img)])
def left_first_quartile(img):
  _img = img[:img.shape[0]//2,:]
  return np.asarray([np.quantile(_img,0.25)])
def right_first_quartile(img):
  _img = img[img.shape[0]//2:,:]
  return np.asarray([np.quantile(_img,0.25)])
def left_third_quartile(img):
  _img = img[:img.shape[0]//2,:]
  return np.asarray([np.quantile(_img,0.75)])
def right_third_quartile(img):
  _img = img[img.shape[0]//2:,:]
  return np.asarray([np.quantile(_img,0.75)])

def light_count(img):
  # How many window x window squares have a mean value above the white_threshold?
  num = 0
  window = 3
  for i in range(window//2,img.shape[0]-window//2):
    for j in range(window//2,img.shape[1]-window//2):
      subset = img[i-window//2:i+window//2+1,j-window//2:j+window//2+1]
      if np.mean(subset) >= white_threshold:
        num += 1
  return np.asarray([num])

def dark_count(img):
  # How many window x window squares have a mean value below the dark_threshold?
  num = 0
  window = 3
  for i in range(window//2,img.shape[0]-window//2):
    for j in range(window//2,img.shape[1]-window//2):
      subset = img[i-window//2:i+window//2+1,j-window//2:j+window//2+1]
      if np.mean(subset) <= dark_threshold:
        num += 1
  return np.asarray([num])
```

Figure 7. Domain-Specific Features (Pt. 2). These features include statistical analysis of the light and dark pixels present in the images.

```
##########################################################################
# Special: context-based feature engineering stuff
##########################################################################

def generate_special_df(self, imgs,
                    filter_funcs=[blobdog, blobdoh, bloblog,
                                  canny_white, laplace_white],
                    ):
  # Feature engineered features
  # Params:
  #   imgs -            array of images (images are 3D)
  #   filter_funcs -    1D feature generator
  #   train -           bool for whether generating features for train or test (for PCA)):

  dfs = []
  for func in tqdm(filter_funcs):
    features = np.concatenate([func(cv2.split(img)[0]) for img in imgs], axis=0)
    df = pd.DataFrame.from_dict({func.__name__:features})
    dfs.append(df)

  df = pd.concat(dfs,axis=1)
  return df
```

Figure 8. Feature Generator Function for Domain-Specific Features.

```
##########################################################################
# PCA: dimensionality reduction of calculate_features
##########################################################################

def calculate_PCA(self, imgs, func, train=False):
  # Calculates features specified in params
  # Params:
  #   imgs -  all preprocessed images (images are 3D)
  #   func -  filter func to calculate PCA for
  #   train - bool for whether to call fit_tranform or just transform
  # Returns: feature vector (1D)

  features = np.concatenate([func(cv2.split(img)[0])[0].reshape(1, -1) for img in imgs],axis=0)
  if train:
    pca = PCA(n_components=self.pca_components)
    components = pca.fit_transform(features)
    self.pca_objects[func.__name__] = pca
  else:
    components = self.pca_objects[func.__name__].transform(features)
  return components

def generate_PCA_df(self, imgs,
                    filter_funcs=[calculate_hog, calculate_gabor, calculate_canny,
                                  calculate_hessian, calculate_meijering, calculate_laplace],
                    train=False):
  # Top level feature generation that calls the feature generator for each image and stores them in DataFrame w/ labels
  # Params:
  #   imgs -            array of images (images are 3D)
  #   filter_funcs -    "raw" feature generators (produce 2D matrices the same size as input image)
  #   train -           bool for whether generating features for train or test (for PCA)):

  dfs = []
  for f in tqdm(filter_funcs):
    components = self.calculate_PCA(imgs,f,train)
    df = pd.DataFrame.from_records(components, columns=[f.__name__.split("_")[1]+" PCA "+str(i) for i in range(self.pca_components)])
    dfs.append(df)

  df = pd.concat(dfs,axis=1)

  return df
```

Figure 9. Feature Generator Function for Computer Vision Features.

```
filter_funcs = [calculate_hog16, calculate_hog8, calculate_hog4,
                calculate_canny,
                calculate_gabor15, calculate_gabor45, calculate_gabor75, calculate_gabor90,
                calculate_hessian, calculate_meijering, calculate_laplace,
                calculate_sobelh, calculate_sobelv]

special_funcs = [blobdog, blobdoh, bloblog,
                 canny_white, laplace_white, hog_white, meijering_white, hessian_white,
                 raw_white, raw_dark, raw_black,
                 raw_mean, center_mean, left_mean, right_mean,
                 left_first_quartile, right_first_quartile,
                 left_third_quartile, right_third_quartile,
                 light_count, dark_count
                 ]

########################################

FG = FeatureGenerator(pca_components=20)


special_df = FG.generate_special_df(imgs=imgs_train,
                                    filter_funcs=special_funcs)


pca_df = FG.generate_PCA_df(imgs=imgs_train,
                            filter_funcs=filter_funcs,
                            train=True)
```

Figure 10. Feature Generator Instantiation.

```
train_df = pd.concat([pca_df, special_df],axis=1)

scaler = MinMaxScaler()
train_df = normalize_features(train_df, scaler, train=True)

######################## Adding in the External Features #################
external_features = pd.read_csv(train_labels)
gender = []
age = []
location = []
for lbl in lbls_filenames:
    image_info_row = external_features[external_features["filename"]==lbl].iloc[0]
    gender.append(image_info_row["gender"])
    age.append(image_info_row["age"])
    location.append(image_info_row["location"])

train_df["gender"] = gender
train_df["gender"]=train_df["gender"].fillna(train_df['gender'].value_counts().index[0])
train_df["age"] = age
train_df["age"]=train_df["age"].fillna(train_df['age'].mean())
train_df["location"]=location
train_df["location"]=train_df["location"].fillna("Null Island")
train_df = pd.get_dummies(train_df, prefix=['gender', 'location'], columns=['gender', 'location'])
```

Figure 11. Concatenating Feature DataFrames and Adding Demographic Features.

(b.ii) (2 points) Feature exploration.

Provide visualizations of the features with respect to the outcome (e.g., overlaying histograms, scatter plots), and quantify associations between the features and the outcome. metric that indicates associations between features and categorical outcome.

We explored conditional entropy, χ^2, mutual information between each feature and the outcome, correlation with the outcome, and the Fisher Score to determine which features provided the most information about the outcome.

We find that most features will share a similar range between the two classes. This pattern makes sense when you consider that all of the images are of x-rays and will only have relatively minor variations between positive and negative samples. As we generate 354 features, we will list the top-5 features from each metric and the corresponding histograms for the top feature for each metric (see Table 1). “Location Melbourne, Australia” is not included in the histograms because it’s a binary feature. In Figure 12, we show the conditional entropies of each feature to the outcome. Most features provide a significant amount of information about the outcome as evidenced by the number of features with 0 entropy.

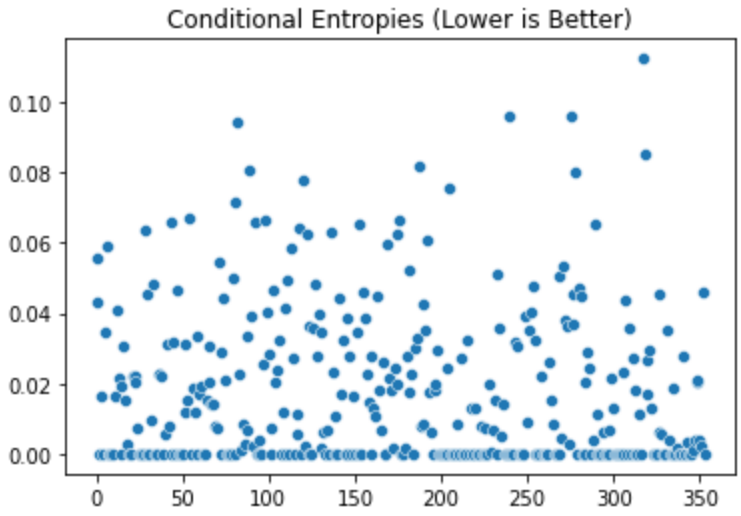


Figure 12. Conditional Entropy of the Generated Features to the Outcomes.

Table 1. Top 5 Features Per Metric

Rank	χ^2	Correlation	Mutual Information	Fisher Score
1	Age	Location Melbourne, Australia	Sobel_V PCA 12	Meijering PCA 7
2	Location Melbourne, Australia	HoG (16,16) PCA 3	Sobel_V PCA 0	HoG (8,8) PCA 3
3	Location Italy	Location Italy	Location Doha, Qatar	Location Hospital Universitario Doctor Peset, Valencia, Spain
4	Location Milan, Italy	HoG (4,4) PCA 1	Gabor (θ=15) PCA 2	HoG (16,16) PCA 1
5	Location Edinburgh, United Kingdom	Gabor (θ=75) PCA 3	HoG (4,4) PCA 11	Gabor (θ=75) PCA 4

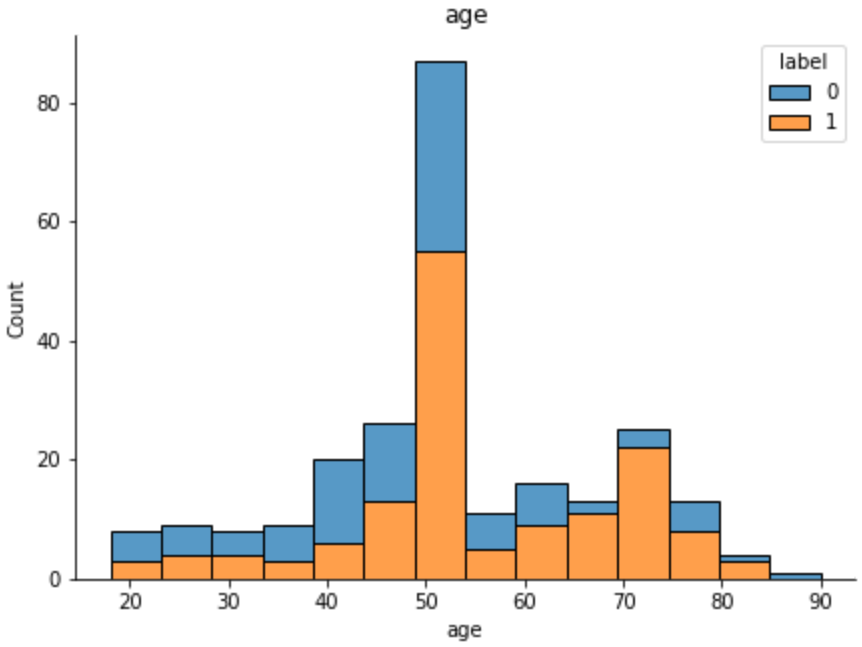


Figure 13. Histogram for the Age Demographic Feature

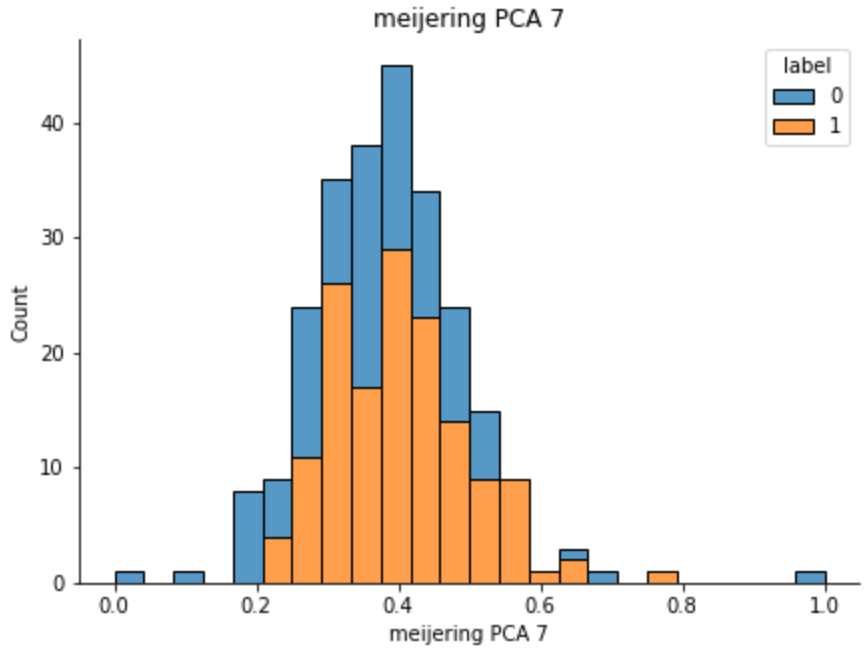


Figure 14. Histogram for the 7th PCA Feature for Meijering Filter

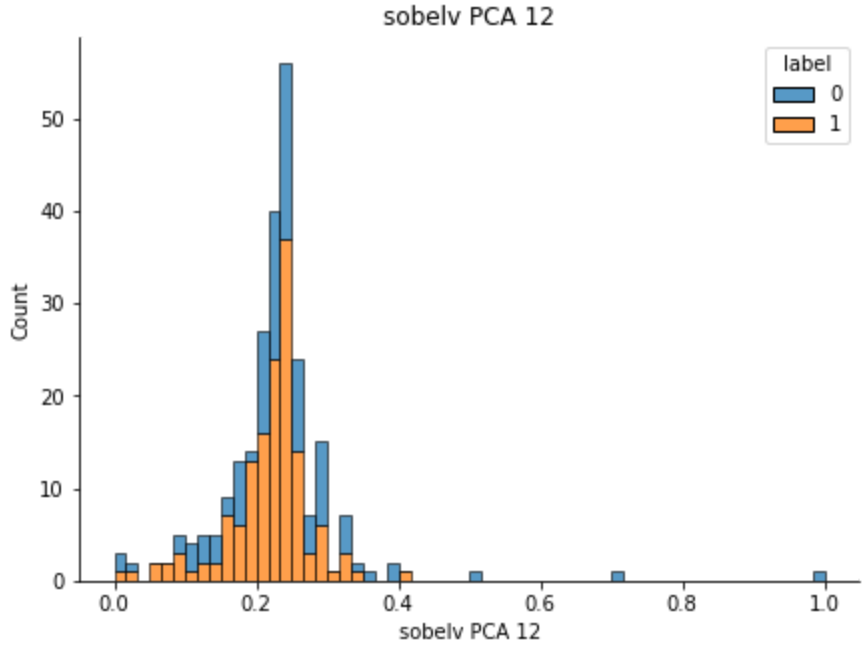


Figure 15. Histogram for the 12th PCA Feature for the Vertical Sobel Filter

(b.iii) (2 points) Feature selection.

*Using the features that you have designed, explore two different feature selection methods of your choice. One method should be part of the **Filter** category and the other should be part of the **Wrapper** category. Using a simple classifier (e.g., SVM, logistic regression), plot the classification performance using a 5-fold cross-validation on the training data against the number of features for both feature selection methods. Compare and contrast between the two (e.g., in terms of performance and computation time).*

For the Filter methods, we explored the four metrics detailed in Table 1 with three classifiers, Logistic Regression, SVM, and Decision Tree. For the Wrapper method, we explored sklearn's Recursive Feature Elimination with Cross Validation (RFECV) with an SVM as the base classifier. The filter methods use greedy selection, choosing the features with the best scores for the current metric in steps of 5. The RFECV uses a step of 1.

Depending on the classifier used in the Filter methods, exploring all of the subsets took between 49s and 1 minute and 41s. The highest accuracy was 86.4% from Logistic Regression. The Wrapper method took 8s and got 98.8% accuracy. As such, the Wrapper method outperformed the Filter methods with respect to both time and performance.

The inner performance of the feature selection methods are given in Figures 16-19.

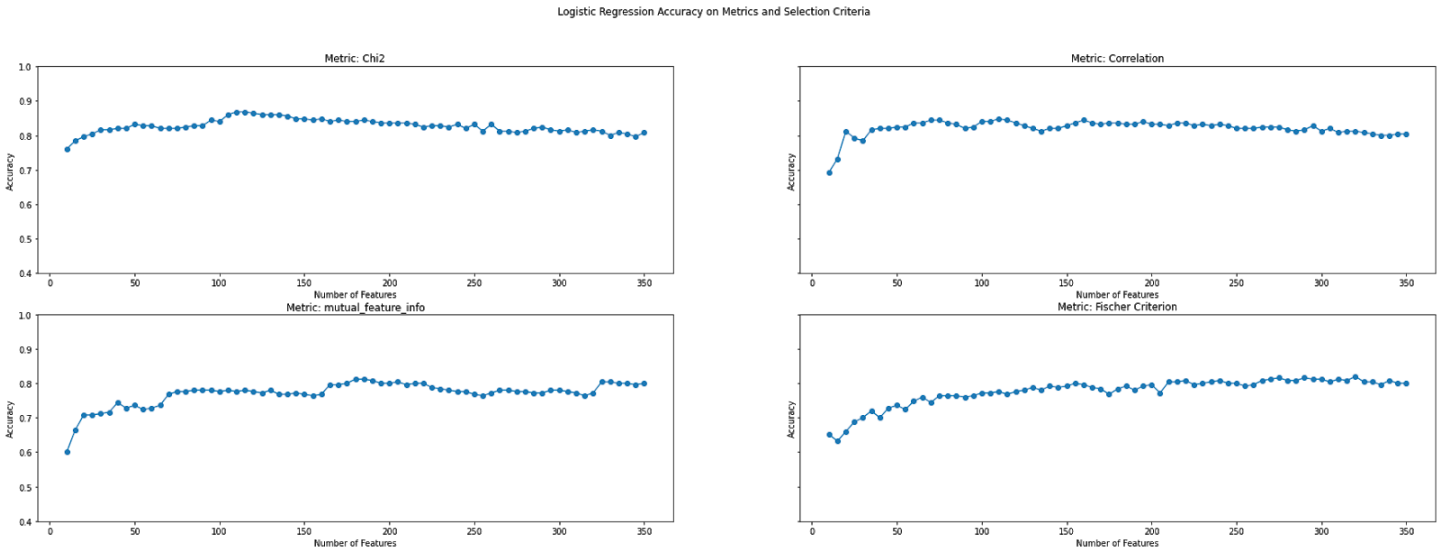


Figure 16. Logistic Regression Filter Method.

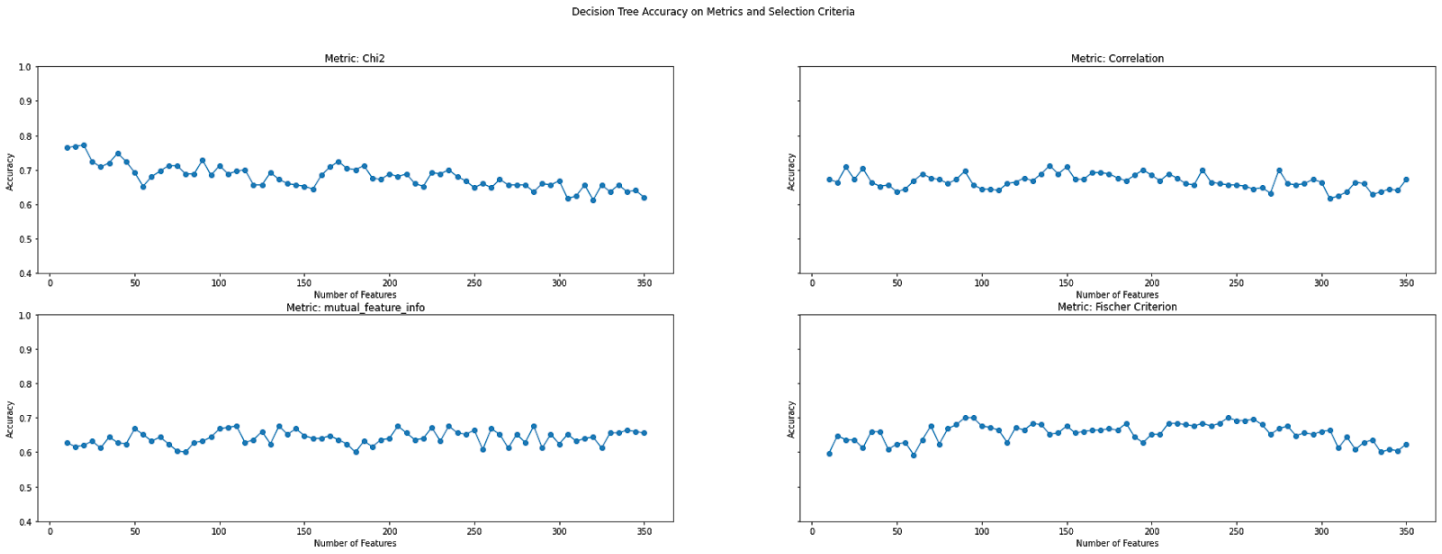


Figure 17. Decision Tree Filter Method.

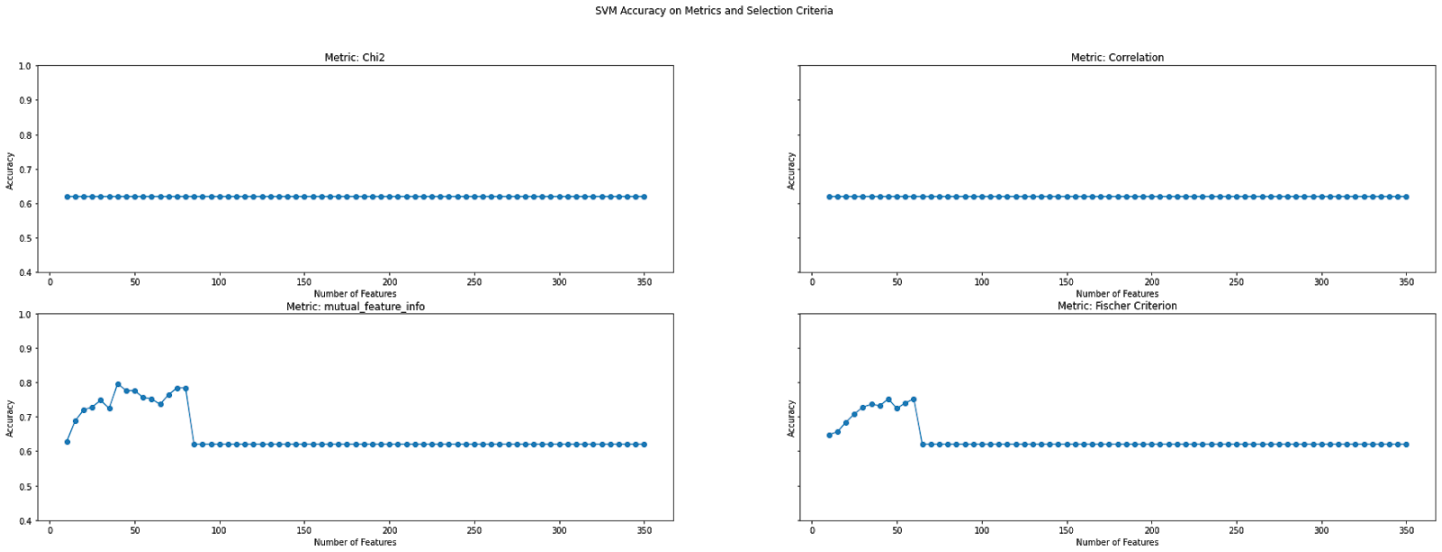


Figure 18. SVM Filter Method.

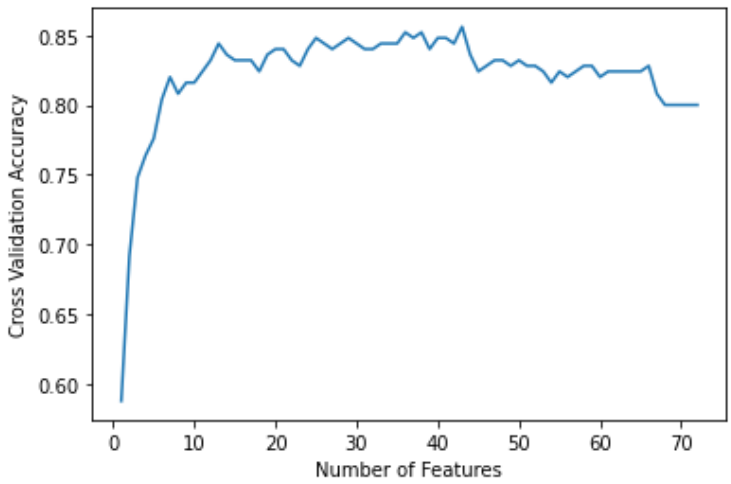


Figure 19. RFECV Feature Selection with SVM as Base Classifier

(b.iv) (1 point) Ensemble learning.

Using the features that you have designed, employ the Adaboost method to estimate the COVID-19 diagnosis. Report results using a 5-fold cross validation on the training data. Compare the result from Adaboost with the ones from feature selection.

To explore the data using ensemble learning, we used the AdaBoost function pre-defined in the sklearn.ensemble library. We used sklearn's GridSearchCV to find the best values for two parameters, the learning rate and number of estimators, through a 5-fold cross validation hyperparameter tuning. After the best hyperparameters were found, we evaluated the AdaBoost model with no feature selection, the best Filter feature set, and the best Wrapper feature set. These results are given in Table 2.

```
#5-fold CV using best estimators
def cv5(clf, X, y):
  # Runs the classifier with 5-fold stratified CV and prints the average accuracy and confusion matrix
  # Params:
  #   clf - sklearn classifier constructor
  #   X   - 2D feature matrix (get from DataFrame using .values)
  #   y   - 1D label vector
  acc = 0
  y_true = [] # keep track of the true labels
  y_pred = [] # keep track of the predictions
  skf = SKFold(n_splits=5)
  for train_index, test_index in tqdm(skf.split(X,y)):
    clf.fit(X[train_index],y[train_index])
    preds = clf.predict(X[test_index])

    acc += metrics.accuracy_score(y_true=y[test_index],y_pred=preds)
    y_true.extend(y[test_index])
    y_pred.extend(preds)
  print("Average Accuracy =",acc/5)
  print(metrics.confusion_matrix(y_true=y_true,y_pred=y_pred))
  pass
```

Figure 20. Our Cross-Fold Validation Function

```
# Run GridSearchCV on AdaBoost
param_grid = [{'n_estimators': [50,100,200], 'learning_rate':[0.01,0.1,1, 0.05]}]
gb = GridSearchCV(AdaBoostClassifier(), param_grid=param_grid, cv=5, n_jobs=-1, scoring='accuracy')
gb.fit(train_df[selected_feats],lbls_train)

print("Best Hyperparameters: {}".format(gb.best_params_))
print("Best score on the training set is {}".format(gb.best_score_))
model = gb.best_estimator_

cv5(model, train_df[all_features].values, lbls_train, None, False)
cv5(model, train_df[selected_feats].values, lbls_train, None, False)
cv5(model, train_df[selected_feats2].values, lbls_train, None, False)
```

Figure 21. Finding the Ideal Hyperparameters for AdaBoost and running the resulting best model with the different feature selection options

In general, AdaBooost performed worse than the models we have previously tested. The accuracy regardless of feature selection method was much lower than basic Logistic Regression with selected features. However, within the three tested options tested, the most accurate AdaBoost model version used the Wrapper (RFECV) feature set which had also had the highest performance during the feature selection phase.

Table 2. Ensemble Learning Results

Model	Feature Selection	Accuracy
AdaBoost	None	74%
	Filter (Log. Reg.)	74.4%
	Wrapper (RFECV)	78%

(c) (2 points) Improving performance:

Use any type of machine learning algorithm or feature design to improve the performance of your system. Describe your proposed approach and report the performance using a 5-fold cross-validation on the training data.

We tested various combinations of the two feature selection methods and different models we developed (see above for details) to find the model-feature combination with the best performance. We looked at three models, Random Forests, Linear SVM, and a Multi-Layer Perceptron. Each of these models uses the sklearn implementation. We looked at the each model's performance with no feature selection (all 354 features-both extracted and provided), the features selected through a filter using the χ^2 criterion judged on Logistic Regression and features selected through an RFECV Wrapper.

Table 3. Ensemble Learning Results

Model	Feature Selection	Accuracy
Random Forest (# trees = 100)	None	70.8%
	Filter (Log. Reg.)	73.6%
	Wrapper (RFECV)	73.6%
Linear SVM	None	82.8%
	Filter (Log. Reg.)	87.7%
	Wrapper (RFECV)	**95.6%**
Multilayer Perceptron (1 hidden layer, 200 nodes)	None	83.6%
	Filter (Log. Reg.)	82%
	Wrapper (RFECV)	90.4%

(d) (2 Bonus points)

*Select the **two** models that you believe that work best and are the most generalizable. Apply these models on the testing data. Provide the decisions in a csv file that includes three columns (in the provided order): test filename, model 1 decision, model 2 decision. The three teams that achieve the best performance will get 2 bonus points. Note: Please do not provide more than two models.*

While our model in Table 3 achieved high accuracy on the training set, the feature set has room for improvement with respect to generalizability. Some of our engineered features could cause the data to overfit, e.g., the location where the x-ray was taken.

Model 1: Linear SVM with only computer vision features (i.e., no engineered features or demographic information). This model had an accuracy of 76% on the training set with 65 features selected by a filter method.

Model 2: Linear SVM with computer vision features with lower dimensionality PCA (pca_components=10) and a reduced subset of our engineered features that were more likely to generalize. This model achieved 79.2% accuracy on the training set with 46 features selected by RFECV.

Our predictions are given in the file **test_predictions.csv** (attached). The columns are:

- filename - the name of the image in the test set
- predict1 - negative/positive prediction (0/1) for Model 1
- predict2 - negative/positive prediction (0/1) for Model 2

(e) (2 points + 1 Bonus point) Elevator pitch video.

Prepare a 45 sec video to describe your work and results. In your elevator pitch, you have to attract the interest of your audience, so that they want to listen to more details on your presentation. You can use any type of visuals that you would like. Please upload your video on any type of social media (e.g., YouTube, TikTok) and provide the url in your final report. You will be also sending us the url of the video before the day that you are presenting in class, so that we can show it everyone! We will send you a reminder regarding this. Note: The elevator pitch will be presented in beginning of the class on 11/20 and 11/23. The best videos, as voted by the rest of the class, will get 1 bonus point. Note: You can use a private link if you would like.

Our video is available on YouTube at : https://www.youtube.com/watch?v=NXduXrBpuEI

(f) (2 points) E-poster.

Create an e-poster presentation of your work. The e-poster will give the main gist of your work, including the problem statement, your methodology, and the main results from your experiments. Add visuals to your poster so that people understand the main concepts. Do not make your e-poster too crowded, since you want other people to be able to see through the screen projection. You can find here the link to prepare your poster presentation https://www.youtube.com/watch?v=1RwJbhkCA58&feature=youtu.be. Note: Each team will be assigned to a Zoom link. Zoom links will be available to everyone in the class. The teams that are not presenting in the current day will log in to discuss your e-poster presentation. All members of the team need to be present during the presentation.

Our poster is included in this report in Figure 22 and attached as its own file, **Team10_Poster.pdf**.

Level-Up Lungs: Identifying COVID-19 Infected Lungs Using Machine Learning

Sanjeevani Choudhery, Samantha Ray, Sournav Bhattacharya, Rohan Singh Wilkho, and Mounika Kunduru

Is It COVID-19?

Note: These are inferences made by computer science students, not medical professionals

COVID-19 Patient X-Ray [1]

White dense residue in lungs makes X-ray "blurry".

Healthy Patient X-Ray [1]

Dark background comes through with **clear** lungs.

Motivation: Given lung X-rays and patient demographics (gender, location and age), we can predict with **95.6%** accuracy in 5-fold cross validation whether the patient has COVID-19.

Making it Black and White

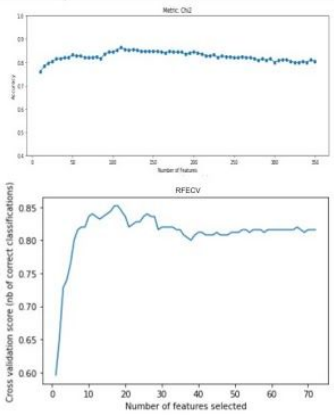
Dense matter shows up as white in x-rays, i.e., bones, certain organs such as the heart, and abnormalities in the lungs.

Standard Computer Vision Features	Domain-Knowledge Features
• Edge Detection: Canny, Hessian, Meijering, Laplace, Sobel (Horizontal and Vertical) • Histogram of Gradients (HoG) • Gabor Filter with various values for theta	• Blob Counts • Light/Dark Patch Counts • Edge Detection Ratio • Pixel Value Statistics (Mean, 1st and 3rd Quartile, Light/Dark Percentage)

Selecting the Best Features

Filter Method	Wrapper Method
• Metrics Tested: correlation with with the target variable, chi-squared, mutual information gain, and Fisher's criterion. • Accuracy : 86% with ~200 features based on Chi-Square • Time: ~ 3 minutes to run each metrics with 5-Fold CV	• RFECV with SVC linear kernel • Accuracy : 0.98 with 79 selected features using SVC • Time : <1 minute total with GPU acceleration

Important Feature Categories Selected
• HoG • Canny • Gabor • Hessian • Laplace • Sobel • Blob Count - Difference of Gaussian • Pixel Value Percentages • Light/Dark Patch Counts • Location



5-Fold Cross Validation Accuracy

Model	w/o Feature Selection	w/ Feature Selection
AdaBoost (n_estimators = 200)	0.744	0.816
Random Forest (n_estimators = 100)	0.736	0.776
SVM (kernel = linear)	0.820	**0.956**
Multilayer Perceptron (hidden layer size = 200)	0.824	0.948

Best Performing Model: SVM

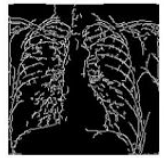
Feature Subset	w/o Feature Selection	w/ Feature Selection
Standard Computer Vision Features Only	0.708	0.800
Standard Computer Vision Features + Domain-Knowledge Features	0.688	0.828
Standard Computer Vision Features + Domain-Knowledge Features + Demographic Features	0.820	**0.956**

Processing Pipeline

Original Image Cropped and Resized to 200x200

Preprocessing: Dilate, Erode, and Sharpen

These preprocessing steps standardize the inputs and make important features clearer



A. Canny

B. Hessian

C. Gabor

D. HoG

E. Laplace

And all the other features generated

PCA

Feature Selection

References

[1] Chandra, T. B., Verma, K., Singh, B. K., Jain, D., & Netam, S. S. (2021). Coronavirus disease (COVID-19) detection in Chest X-Ray images using majority voting based classifier ensemble. Expert systems with applications, 165, 113909. https://doi.org/10.1016/j.eswa.2020.113909

[2] Parekh, M., Donuru, A., Balasubramanya, R., & Kapur, S. (2020). Review of the Chest CT Differential Diagnosis of Ground-Glass Opacities in the COVID Era. Radiology, 297(3), E289–E302. https://doi.org/10.1148/radiol.2020202504

[3] Mayo Clinic. Chest X-ray. Last accessed Nov. 21, 2020. Available: https://www.mayoclinic.org/tests-procedures/chest-x-rays/multimedia/chest-x-ray/img-20006961

Figure 22. Team 10's Poster.

(g) (1 point) Reporting other teams' work.

During the day that your team is not presenting, your will go around the posters of the teams that are presenting and report the main findings of the other teams. In the final report, provide a brief description of the work and main results from 8 other teams.

We attended the poster sessions of the 10 teams presenting on Friday. Our notes are compiled in Table 4.

Table 4. Even Number Teams' Posters

Team #	Best Model	Accuracy	Technique Notes
1	ResNet-18	72%	Used HoG, Gabor, Horizontal/Vertical Edge Detection, Pixel Value, Mean Pixel Value
3	Linear SVM	92.5%	Feature selection: RFECV. They explored several ML models and used standard computer vision features, e.g., HoG
5	VGG19	~85%	Pipeline: Preprocess images, VGG19 feature extraction, then Logistic Regression
7	SVM	82.04%	SVM Feature Extraction, VGG16 features extraction with ImageNet pretraining, followed by their own final classifier trained using 200 epochs
9	CNN	71.2%	They used HoG, ORB, SIFT, and SURF features. Their poster didn't contain a lot of detail.
11	SVM	82.4%	Features: HoG. Feature Selection: Fischer Score. Then they passed these features into an RFECV SVM.
13	Random Forest	77.5%	They preprocessed their images to 128x128 and used HoG features. They tried transfer learning with VGG16, but Random Forest with a Wrapper feature selection method.
15	CNN	91.6%	Features: Grayscale, Mean Pixel, Horizontal/Vertical Edge Detection, HoG. Feature Dimensionality Reduction: PCA. They tried AdaBoost and CNN
17	CNN	85%	Features: Gabor, HoG, Horizontal/Vertical Edge Detection. Feature Selection: SVM Wrapper
19	Pretrained DenseNet on ChexNet	77%	Features: Gabor, HoG, Sobel Edge Detection, Prewitt Edge Detection. The main models they tested are AdaBoost and CNN