INFO6105 34274 Data Sci Eng Methods SEC 02
Spring 2021

Final Project Report

Invasive Ductal Carcinoma (IDC)-A Type of Breast Cancer Prediction

Instructor: **Dr. Liu Handan**

Team Number: **Team 8**

Team Members:

Jinning Yang 001029162 **Shuyao Zhang 001522915**

Introduction

Background:

- **What is Brest Cancer?**

  Cancer occurs when some of the body's cells behave abnormally changing, growing and reproducing more than usual. For breast cancer, the cells in the breast tissue are the ones with abnormal growth. Breast cancer is the kind of cancers that starts in different part of the breasts.

  Cancer starts when cells begin to grow out of control. Breast cancer cells usually form a tumor that can often be seen on an x-ray or felt as a lump. Breast cancer occurs almost entirely in women, but men can get breast cancer, too.

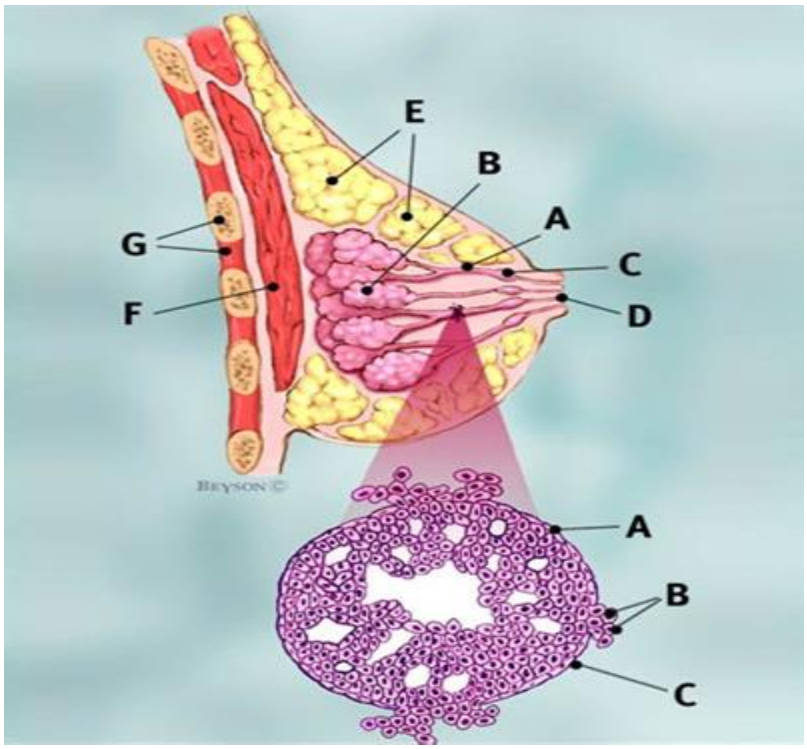
  A breast cancer diagnosis comes with a lot of information. It is normal for patient with breast cancer to feel overwhelmed. It is also one of the most common types of cancers for women, sometimes for men as well. There are more than 4700 women were diagnosed with breast cancer in 2016 and 710 of them were dead.

  It's important to understand that most breast lumps are benign and not cancer (malignant). Non-cancerous breast tumors are abnormal growths, but they do not spread outside of the breast. They are not life threatening, but some types of benign breast lumps can increase a woman's risk of getting breast cancer. Any breast lump or change needs to be checked by a health care professional to determine if it is benign or malignant (cancer) and if it might affect your future cancer risk.

  To minimize the number of people that is killed by breast cancer, it is very critical that we can predict whether the breast cancer of people is benign or malignant. It is necessary to improve the accuracy of prediction.

- **What is Invasive Ductal Carcinoma?**

  Invasive ductal carcinoma (IDC), also known as infiltrating ductal carcinoma, is cancer that began growing in a milk duct and has invaded the fibrous or fatty tissue of the breast outside of the duct. IDC is the most common form of breast cancer, representing 80 percent of all breast cancer diagnoses.



Invasive means that the cancer has "invaded" or spread to the surrounding breast tissues. ***Ductal*** means that the cancer began in the milk ducts, which are the "pipes" that carry milk from the milk-producing lobules to the nipple. ***Carcinoma*** refers to any cancer that begins in the skin or other tissues that cover internal organs — such as breast tissue. All together, "invasive ductal carcinoma" refers to cancer that has broken through the wall of the milk duct and begun to invade the tissues of the breast. Over time, invasive ductal carcinoma can spread to the lymph nodes and possibly to other areas of the body.

According to the American Cancer Society, more than 180,000 women in the United States find out they have invasive breast cancer each year. Most of them are diagnosed with invasive ductal carcinoma.

Although invasive ductal carcinoma can affect women at any age, it is more common as women grow older. According to the American Cancer Society, about two-thirds of women are 55 or older when they are diagnosed with an invasive breast cancer. Invasive ductal carcinoma also affects men.

Motivation:

In the past, clinicians have used basic software programs, such as Microsoft Excel, to analyze breast cancer. However, it is not adaptable in identifying new variables as well as generating creative and integrative visualizations. It is important to find a better solution for breast cancer prediction.

Machine Learning is one of the best solutions for this problem. Various machine learning approaches like decision tree, random forest, neural networks, extreme boost, logistic regression could be the answer of predicting breast cancer currently.

Apparently, identifying and categorizing breast cancer subtypes is an important task for clinical, which can highly reduce the error of prediction of cancer and increase the survival ratio for people with breast cancer.

Invasive ductal carcinoma (IDC) is one of the most common types of breast cancer, accounting for about 80%. It is malicious and can form a diversion, which makes it particularly dangerous. Biopsies are usually done to remove small tissue samples. Then, the pathologist has to decide whether the patient has IDC, another type of breast cancer, or is healthy. In addition, disease cells need to be located to find out the progress of the disease and which level should be assigned. This has to be done manually and is a time-consuming process. In addition, the decision depends on the pathologist's expertise and his or her equipment. Therefore, in-depth learning is helpful to the automatic detection and localization of tumor cells and accelerate the detection and localization process of tumor cells. In order to fully tap this potential, people can use a large number of tissue image data from different hospitals evaluated by different experts to construct pipelines. This can overcome the dependence on pathologists, which is especially useful in areas where there are no experts.

Goal:

Give a patient a slice of tissue to predict if it contains IDC - three possibilities: healthy tissue, IDC, another subtype of breast cancer.

So far, predictions have been made manually by pathologists, and the predictions vary from expert to expert. Our goal is to help automatically detect tumors (independent of experts).

We will use K-Fold this method to train and test a model, and we combine K-Fold with Decision Tree, Random Forest Classifier, SVM, Logistic Regression, K Neighbors, or Naive Bayesian. Then we compare the accuracy between different models and try to find the best one.

We also used data to train and test the CNN (Central Neural Networks) model. We spited the data into 3 parts-test data, train data, and dev data. CNN is expected to give us a precise result after training.

Key Words:

Invasive ductal carcinoma (IDC), Breast cancer, K-Fold, Decision Tree, Random Forest, SVM, Logistic Regression, K Neighbors, or Naive Bayesian, CNN (Central Neural Networks)

Jinning Yang Part

1.Methodology

1.1 Algorithms & Methodology:

- **K-Fold:**

  In *k*-fold cross-validation, the original sample is randomly partitioned into *k* equal sized subsamples. Of the *k* subsamples, a single subsample is retained as the validation data for testing the model, and the remaining $k - 1$ subsamples are used as training data. The cross-validation process is then repeated *k* times, with each of the *k* subsamples used exactly once as the validation data. The *k* results can then be averaged to produce a single estimation.

  Advantage:

  The advantage of this method over repeated random sub-sampling (see below) is that all observations are used for both training and validation, and each observation is used for validation exactly once.

  Disadvantage:

  10-fold cross-validation is commonly used,[16] but in general k remains an unfixed parameter.

- **Decision Tree:**

  A decision tree is a flowchart-like structure in which each internal node represents a "test" on an attribute, each branch represents the outcome of the test, and each leaf node represents a class label (decision taken after computing all attributes).

  Advantage:

  a. Are simple to understand and interpret. People are able to understand decision tree models after a brief explanation.
  b. Have value even with little hard data. Important insights can be generated based on experts describing a situation (its alternatives, probabilities, and costs) and their preferences for outcomes.
  c. Help determine worst, best and expected values for different scenarios.
  d. Use a white box model. If a given result is provided by a model.
  e. Can be combined with other decision techniques.

  Disadvantage:

  a. They are unstable, meaning that a small change in the data can lead to a large change in the structure of the optimal decision tree.

b. They are often relatively inaccurate. Many other predictors perform better with similar data. This can be remedied by replacing a single decision tree with a random forest of decision trees, but a random forest is not as easy to interpret as a single decision tree.
c. For data including categorical variables with different number of levels, information gain in decision trees is biased in favor of those attributes with more levels.
d. Calculations can get very complex, particularly if many values are uncertain and/or if many outcomes are linked.

- **Random Forest:**

  Random forests or random decision forests are an ensemble learning method for classification, regression and other tasks that operates by constructing a multitude of decision trees at training time and outputting the class that is the mode of the classes (classification) or mean/average prediction (regression) of the individual trees.

  Advantage:

  Random forests present estimates for variable importance, i.e., neural nets. They also offer a superior method for working with missing data. Missing values are substituted by the variable appearing the most in a particular node. Among all the available classification methods, random forests provide the highest accuracy.

  The random forest technique can also handle big data with numerous variables running into thousands. It can automatically balance data sets when a class is more infrequent than other classes in the data. The method also handles variables fast, making it suitable for complicated tasks.

  Disadvantage:

  While random forests often achieve higher accuracy than a single decision tree, they sacrifice the intrinsic interpretability present in decision trees. Decision trees are among a fairly small family of machine learning models that are easily interpretable along with linear models, rule-based models, and attention-based models. This interpretability is one of the most desirable qualities of decision trees. It allows developers to confirm that the model has learned realistic information from the data and allows end-users to have trust and confidence in the decisions made by the model.

- **SVM:**

  support-vector machines (SVMs, also support-vector networks) are supervised learning models with associated learning algorithms that analyze data for classification and regression analysis.

  Advantage:

  SVMs are helpful in text and hypertext categorization, as their application can significantly reduce the need for labeled training instances in both the standard inductive and

transudative settings. Some methods for shallow semantic parsing are based on support vector machines.

Disadvantage:

SVM algorithm is not suitable for large data sets. SVM does not perform very well when the data set has more noise i.e., target classes are overlapping. In cases where the number of features for each data point exceeds the number of training data samples, the SVM will underperform.

- **Logistic Regression:**

  Logistic regression is a statistical model that in its basic form uses a logistic function to model a binary dependent variable, although many more complex extensions exist. In regression analysis, logistic regression(or logit regression) is estimating the parameters of a logistic model (a form of binary regression).

- **K Neighbors:**

  k-NN is a type of classification where the function is only approximated locally and all computation is deferred until function evaluation. Since this algorithm relies on distance for classification, if the features represent different physical units or come in vastly different scales then normalizing the training data can improve its accuracy dramatically

- **Naive Bayesian:**

  Naïve Bayes classifiers are highly scalable, requiring a number of parameters linear in the number of variables (features/predictors) in a learning problem. Maximum-likelihood training can be done by evaluating a closed-form expression, 718 which takes linear time, rather than by expensive iterative approximation as used for many other types of classifiers.

1.2 Problems and Solutions

1.2.1 Load packages & import Datas:

In this project, I

- use pandas to process the data
- use NumPy to do linear algebra
- use the metrics from the Sklearn to generate the Confusion Metric, and the seaborn to draw some diagram to make the data visualized.
- use the os to make sure to path of the images and use json to link the different paths.
- use the pyplot to print some picture about the analyzing results.
- use the random to print some random images as test.
- use the IncrementalPCA to apply Principle Component Analysis (PCA)

```
import warnings
warnings.filterwarnings("ignore", category=DeprecationWarning)
warnings.filterwarnings("ignore", category=UserWarning)
warnings.filterwarnings("ignore", category=FutureWarning)

import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)
import matplotlib.pyplot as plt
%matplotlib inline
import seaborn as sns
sns.set()
from PIL import Image

import torch
import torch.nn as nn
import torch.optim as optim
from torch.optim.lr_scheduler import ReduceLROnPlateau, StepLR, CyclicLR
import torchvision
from torchvision import datasets, models, transforms
from torch.utils.data import Dataset, DataLoader
import torch.nn.functional as F
```

```
from sklearn.model_selection import train_test_split, StratifiedKFold
from sklearn.utils.class_weight import compute_class_weight
from sklearn import metrics

from glob import glob
from skimage.io import imread
from os import listdir
from skimage import io
from sklearn.svm import SVC
from sklearn.metrics import classification_report

import time
import copy
from tqdm import tqdm_notebook as tqdm
from sklearn import svm
import gc
from sklearn.decomposition import IncrementalPCA
```

1.2.2 Datas Description

Read datas

```
#read datas
folder = '../input/breast-histopathology-images/IDC_regular_ps50_idx5/'
patient_ids = os.listdir(folder)
p_id = listdir(folder)
```

After reading the datas, we need to review the datas to have a clear understanding of the data information.

In order to train our model, we provide each patch separately for our model, so each patch will be used as an input. The following code snippet traverses the entire file structure and extracts the total number of clippings for each class.

```
class_0_total = 0
class_1_total = 0

for patient_id in patient_ids:
    class_0_files = os.listdir(folder + patient_id + '/0')
    class_1_files = os.listdir(folder + patient_id + '/1')

    class_0_total += len(class_0_files)
    class_1_total += len(class_1_files)
```

We have lots of images, and we shouldn't store the individual pixel values of all images. In order to provide image patches for the algorithm, we will store the paths of each image and create a data frame containing all paths. Thus, we can load a batch of images one by one.

Next, we first construct a dictionary list, and then create a dataframe based on the list. This is much more efficient than using the ".append" or ".loc" method.

For example,

```
>>> import pandas as pd>>> from numpy.random import randint>>> df =
pd.DataFrame(columns=['lib', 'qty1', 'qty2'])>>> for i in range(5):>>>
df.loc[i] = ['name' + str(i)] + list(randint(10, size=2))>>> df
```

```
columns = ["patient_id","diagnosis","path"]
data_rows = []

for patient_id in patient_ids:
    for outcome in [0,1]:
        class_path = folder + patient_id + '/' + str(outcome) + '/'
        imgs = os.listdir(class_path)

        # Extracting Image Paths
        img_paths = [class_path + img + '/' for img in imgs]

        for path in img_paths:
            values = [patient_id,outcome,path]
            data_rows.append({k:v for (k,v) in zip(columns,values)})
```

```
data = pd.DataFrame(data_rows)
#data = data.sample(frac = 0.5)
print(data.shape)
display(data)
```

(277524, 3)

	patient_id	diagnosis	path
0	10295	0	../input/breast-histopathology-images/IDC_regu...
1	10295	0	../input/breast-histopathology-images/IDC_regu...
2	10295	0	../input/breast-histopathology-images/IDC_regu...
3	10295	0	../input/breast-histopathology-images/IDC_regu...
4	10295	0	../input/breast-histopathology-images/IDC_regu...
...	...	...	...
277519	12873	1	../input/breast-histopathology-images/IDC_regu...
277520	12873	1	../input/breast-histopathology-images/IDC_regu...
277521	12873	1	../input/breast-histopathology-images/IDC_regu...
277522	12873	1	../input/breast-histopathology-images/IDC_regu...
277523	12873	1	../input/breast-histopathology-images/IDC_regu...

277524 rows × 3 columns

This table shows the basic information of our datas. We can learn from the table that we have 277524 pieces of data, and there are 3 columns, representing *patient_id*, *diagnosis*, and *path*. *Diagnosis* '**1**' is positive for IDC, and **'0'** is negative for IDC.

This is the detailed data information. It also shows datatype of each attribute.

```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 277524 entries, 0 to 277523
Data columns (total 3 columns):
 #   Column      Non-Null Count   Dtype 
---  ------      --------------   ----- 
 0   patient_id  277524 non-null  object
 1   diagnosis   277524 non-null  int64 
 2   path        277524 non-null  object
dtypes: int64(1), object(2)
memory usage: 6.4+ MB
```

This table show basic statistics—average is 0.28, standard deviation is 0.45 and so on.

|:
```
data.describe()
```

]:

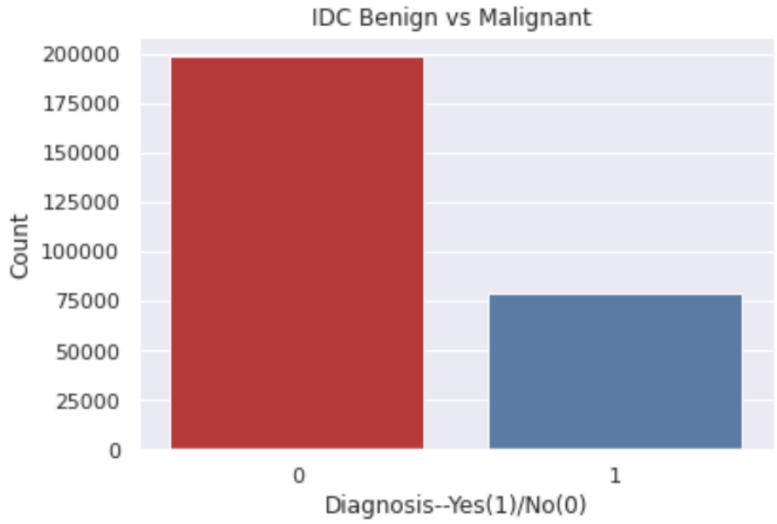
	diagnosis
count	277524.000000
mean	0.283889
std	0.450884
min	0.000000
25%	0.000000
50%	0.000000
75%	1.000000
max	1.000000

Data Analysis:

```
data_copy_1=data.iloc[:277524,:]
from PIL.Image import open
data_count=data_copy_1['diagnosis'].value_counts()
print(data_count)
sns.countplot(data_copy_1['diagnosis'],palette="Set1")
plt.title('IDC Benign vs Malignant ')
plt.xlabel("Diagnosis--Yes(1)/No(0)")
plt.ylabel("Count")
plt.show()
print('The difference between benign and malignant :'+" "
      +str(data_count[0]-data_count[1]))
```

We can learn from the below chart--the classification of IDC and Non-IDC is unbalanced. After setting the validation strategy and finding the strategy to handle class weights, we have to check this again.

```
0    198738
1     78786
Name: diagnosis, dtype: int64
```




```
The difference between benign and malignant : 119952
```

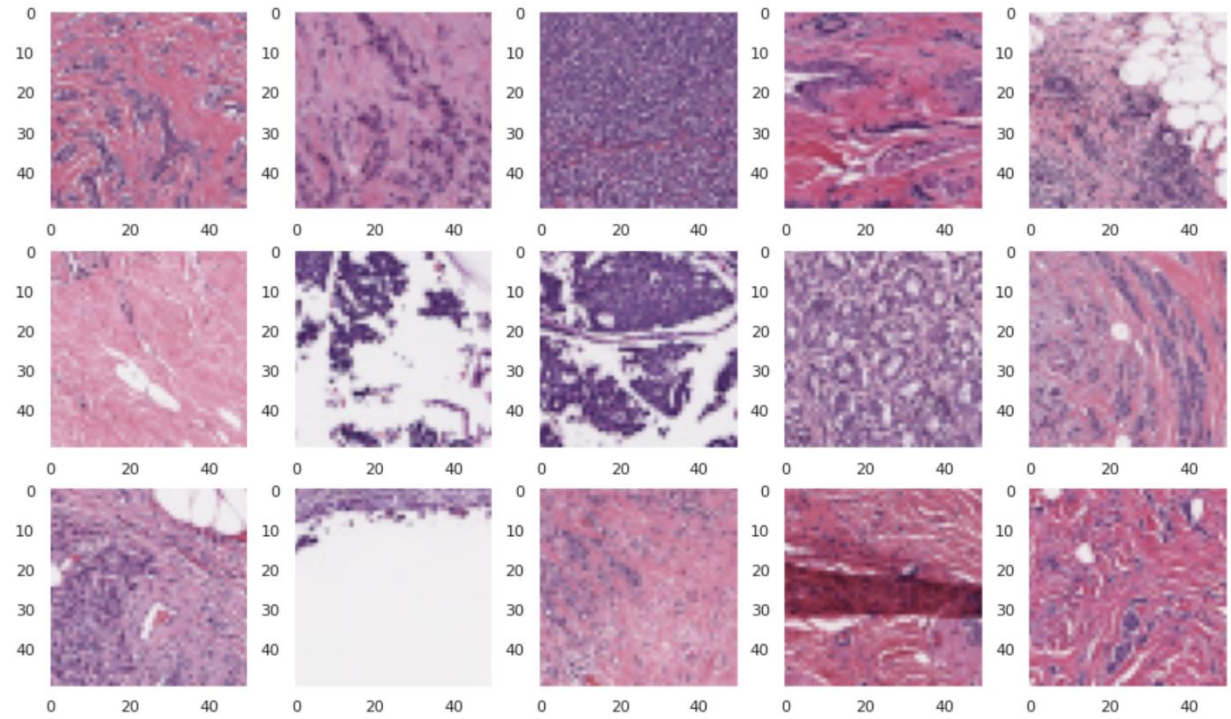
Healthy Tissue Samples Vs Cancerous Tissue Samples

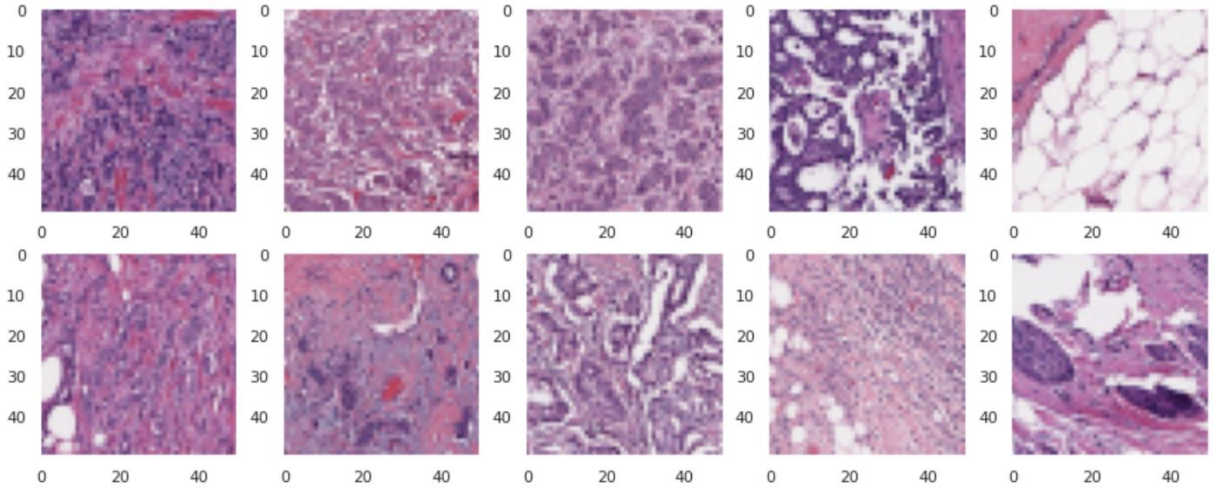
Now let's explore the visual differences between cancer tissue patches and healthy tissue patches.

```
positive_sample = np.random.choice(data[data.diagnosis==1].index.values, size=100, replace=False)
negative_sample = np.random.choice(data[data.diagnosis==0].index.values, size=100, replace=False)

rows = 5
cols = 5
```

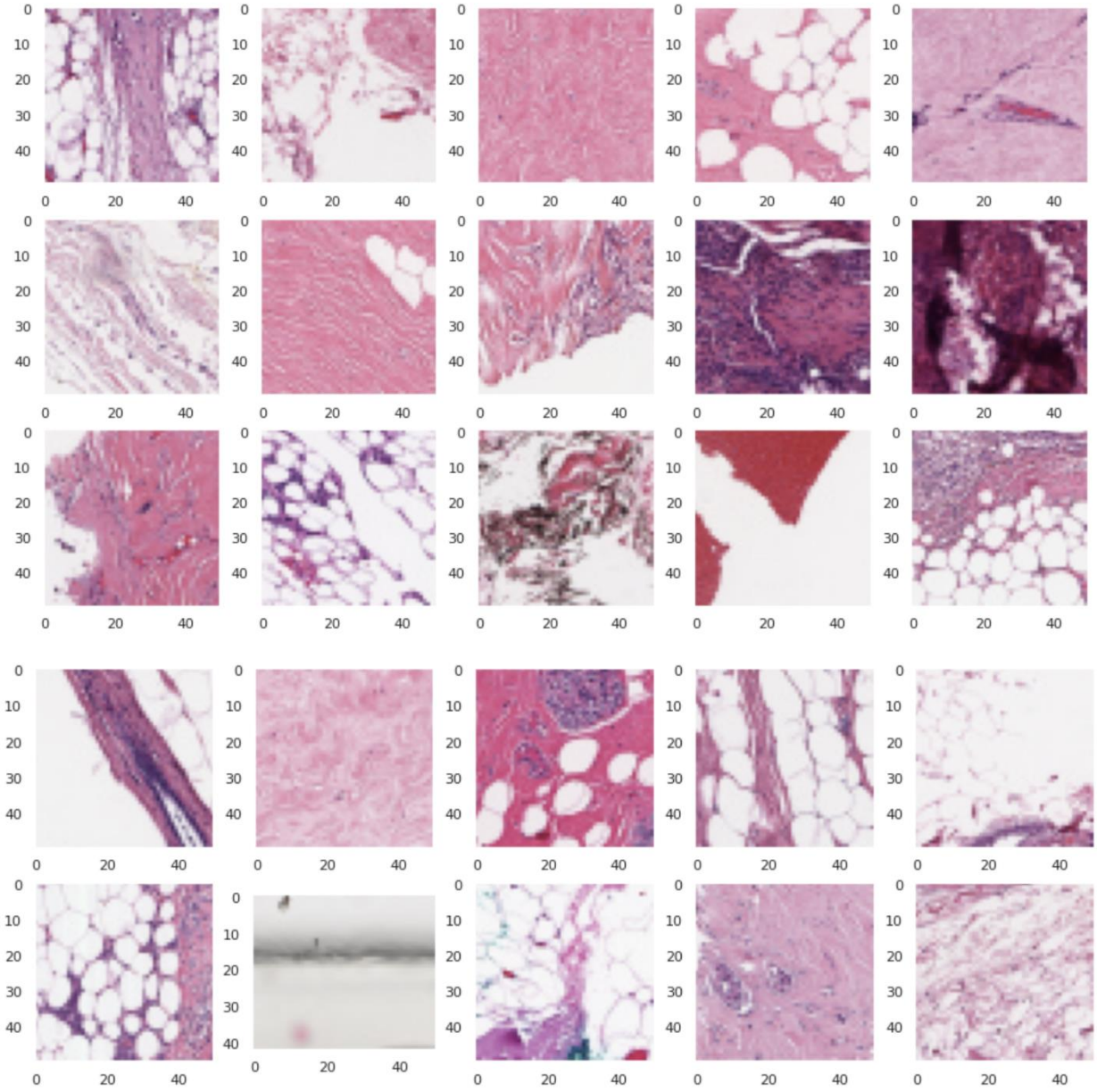
```
fig,ax = plt.subplots(rows,cols,figsize = (15,15))

for row in range(rows):
    for col in range(cols):

        idx = positive_sample[col + cols*row]
        image = io.imread(data.loc[idx, "path"])
        ax[row,col].imshow(image[:,:,:])
        ax[row,col].grid(False)
```

Cancerous Patches





Healthy Patches



1.2.3 Image Pre-Processing:

1.2.3.1 PCA Introduction:

Basic mathematical knowledge-Linear algebra refresher

a. Matrix transpose: reindex a 2-D matrix A to switch the row and column indices, effectively replacing all of its elements a_{ij} with a_{ji}. The notation for

transpose is a superscripted ⊤ or' on the matrix. In numpy, you can call the .T or .transpose() method of the np.ndarray object to transpose a matrix.

b. Dot product and matrix multiplication: the product C=AB of two matrices A (n×m) and B (m×p) should have a shape of n×p. Two matrices can be multiplied only when the second dimension of the former matches the first dimension of the latter. The element c_{ij} in the resultant matrix C is computed as:

$$c_{ij} = \sum_{k=1}^{m} a_{ik} b_{kj} = \mathbf{a}_{i,*} \cdot \mathbf{b}_{*,j}$$

c. Matrix inverse: only square matrices can be inverted, the product of a matrix A (n×n) with its inverse A^(-1) is an identity matrix I, where elements on the diagonal are 1's everywhere else are 0's. In numpy, a matrix can be inverted by np.linalg.inv function.

d. Conjugate transpose: defined as the transpose of a conjugate matrix. Typically denoted with a * or H (Hermitian) as superscript. A conjugate matrix is a matrix obtained from taking the complex conjugate of all the elements in the original matrix:

$$\mathbf{A}^* \equiv (\overline{\mathbf{A}})^\top = \overline{\mathbf{A}^\top}$$

e. Unitary matrix: defined as a square matrix whose conjugate transpose is also its inverse. For a unitary matrix, we have its transpose equals its inverse:

$$\mathbf{U}\mathbf{U}^* = \mathbf{U}\mathbf{U}^{-1} = \mathbf{I}$$
$$\mathbf{U}^* = \mathbf{U}^\top = \mathbf{U}^{-1}, \text{ if } \mathbf{U} \text{ is real matrix}$$

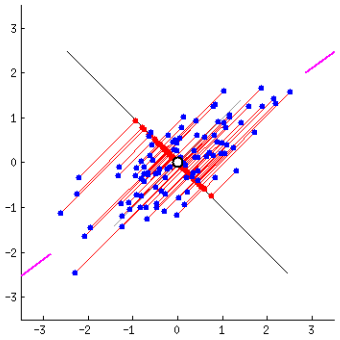
f. Covariance matrix: covariance quantifies the joint variability between two random variables X and Y and is calculated as:

$$cov(X, Y) = \mathbb{E}[(X - \mathbb{E}[X])(Y - \mathbb{E}[Y])]$$

$$\mathbf{C} = \frac{\mathbf{X}^\top \mathbf{X}}{n-1}$$

PCA

PCA aims to find linearly uncorrelated orthogonal axes, which are also known as principal components (PCs) in the m dimensional space to project the data points onto those PCs. The first PC captures the largest variance in the data. Let's intuitively understand PCA by fitting it on a 2-D data matrix, which can be conveniently represented by a 2-D scatter plot:



Since all the PCs are orthogonal to each other, we can use a pair of perpendicular lines in the 2-D space as the two PCs. To make the first PC capture the largest variance, we rotate our pair of PCs to make one of them optimally align with the spread of the data points. Next, all the data points can be projected onto the PCs, and their projections (red dots on PC1) are essentially the resultant dimensionality-reduced representation of the dataset. Viola, we just reduced the matrix from 2-D to 1-D while retaining the largest variance!

The PCs can be determined via eigen decomposition of the covariance matrix C. After all, the geometrical meaning of eigen decomposition is to find a new coordinate system of the eigenvectors for C through rotations.

The PCs can be determined via eigen decomposition of the covariance matrix C. After all, the geometrical meaning of eigen decomposition is to find a new coordinate system of the eigenvectors for C through rotations.

$$\mathbf{C} = \mathbf{W}\mathbf{\Lambda}\mathbf{W}^{-1}$$

In the equation above, the covariance matrix C(m×m) is decomposed to a matrix of eigenvectors W(m×m) and a diagonal matrix of m eigenvalues Λ. The eigenvectors, which are the column vectors in W, are in fact the PCs we are seeking. We can then use matrix multiplication to project the data onto the PC space. For the purpose of dimensionality reduction, we can project the data points onto the first k PCs as the representation of the data:

$$\mathbf{X}_k = \mathbf{X}\mathbf{W}_k$$

```
def pca(X):
  # Data matrix X, assumes 0-centered
  n, m = X.shape
  assert np.allclose(X.mean(axis=0), np.zeros(m))
  # Compute covariance matrix
  C = np.dot(X.T, X) / (n-1)
  # Eigen decomposition
  eigen_vals, eigen_vecs = np.linalg.eig(C)
  # Project X onto PC space
  X_pca = np.dot(X, eigen_vecs)
  return X_pca
```

How to Apply PCA on Satellite Images

1. Data Standardization: Before applying PCA, we have to bring our data to a common format through standardization. The purpose of doing this is to make sure that variables are internally consistent with each other regardless of their type.
2. PCA Transformation
3. Eigenvalues and Vectors Computation: In this step, data compression and dimensionality reduction come into the picture.

1.2.3.2 Apply PCA into our application:

As we all know, we have a large amount of data, so we need to reduce the number of dimensions we are working on. One way is to convert our image from RGB to grayscale image. By taking the weighted sum of R, G and B values and getting a one-dimensional value, the image is converted from RGB image to grayscale.

After applying PCA to color dimension, we can reduce the total dimension of each flattened image from (17500) to (12500). This will help us reduce memory consumption in the future.

RGB->Grayscale

```
#all
all_image_paths=data['path']
all_rgb_pca=rgb_to_grayscale(all_image_paths)
```

Reshape image so that PCA can be applied on the RGB dimession

```
def apply_rgb_to_pca(img,pca_object):

    img = img.reshape(-1, 3)

    img = pca_object.transform(img)

    img = img.flatten()
    return img
```

```
#Transorming RGB image paths to single scale image array using PCA
#all
all_image_paths = data['path']
all_image_targets = data['diagnosis']
```

initialize an array containing zeros to place the images in it

```
all_img_array = np.zeros((all_image_paths.shape[0],2500))

for i,all_image_paths in tqdm(enumerate(all_image_paths),total = all_image_paths.shape[0]):
    img = io.imread(all_image_paths)
    try:
        all_img_array[i:img.shape[0]] = apply_rgb_to_pca(img,all_rgb_pca)

    except:
        pass
```

fit the pixel PCA object

```
#all
all_pca=fit_pixel_pca(all_img_array)
print(all_pca)
```

1.2.4 Algorithms & Methodology:

1.2.4.1 Cross validation (more accurate in calculating accuracy)--K-Fold

Never mix training and test data!

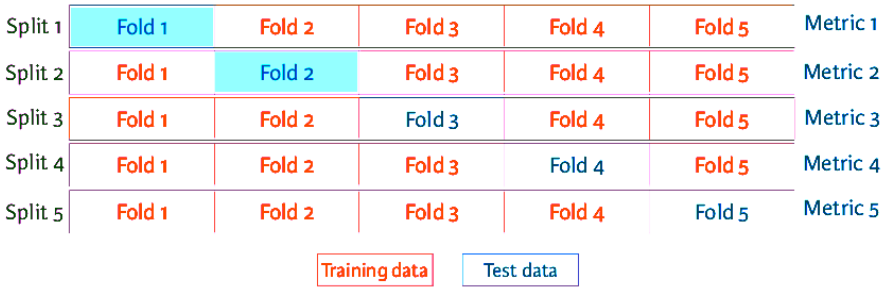
First, we need to isolate the test data-set and use it only for final evaluation.

Define K-Fold Cross Model

- **Why ues K-Fold Cross Model?**

Initially, the whole training data set is divided into k equal parts. The first part is kept as the test set, and the rest k-1 part is used to train the model. Then the trained model is tested on the test set. The above process is repeated K times. In each case, we constantly change the preserving set. Therefore, each data point has an equal opportunity to be included in the test set.

In the application, k is equal to 5, that means the whole data set is divided into 5 equal parts.

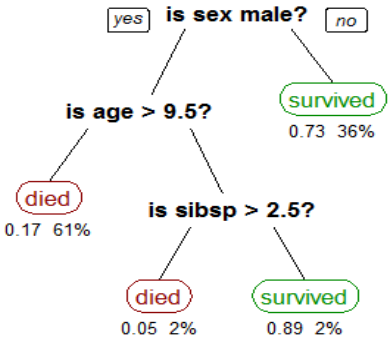


We pass the training model, prediction data and target value for K-Fold cross-validation. The method will return a list of k(5) accuracy values for each iteration. In general, we take the average of them and use it as a consolidated cross-validation score.

```
def classification_model(model,prediction_input,output):
    kf = KFold(n_splits=5,shuffle=True, random_state=1)
    error = []

    for train, test in kf.split(data):
        train_X = (all_data_train_X[train,:])
        train_y = data['diagnosis'].iloc[train]
        x = train_X[~np.all(train_X == 0, axis=1)]
        y = train_y[~np.all(train_X == 0, axis=1)]
        model.fit(x, y)
        test_X=all_data_train_X[test,:]
        test_y=data['diagnosis'].iloc[test]
        x = test_X[~np.all(test_X == 0, axis=1)]
        y = test_y[~np.all(test_X == 0, axis=1)]
        error.append(model.score(x,y))
        print("Cross-Validation Score : %s" % "{0:.3%}".format(np.mean(error)))
    fig, ax = plt.subplots(figsize=(5,5))
    conf_matrix = confusion_matrix(model.predict(x),y)
    conf_matrix = conf_matrix/np.sum(conf_matrix)
    sns.heatmap(conf_matrix, annot=True, fmt=".2f", square=True, cbar=False, cmap=plt.cm.jet,ax=ax)
    ax.set_ylabel('Actual')
    ax.set_xlabel('Predicted')
    ax.set_title('Confusion Matrix')
    plt.show(block=False)
    print("The Accuracy Is : %s" % "{0:.3%}".format(np.mean(error)))
    print(error)
    return error
```

1.2.4.2 Training and Prediction

1.2.4.2.1 Decision Tree

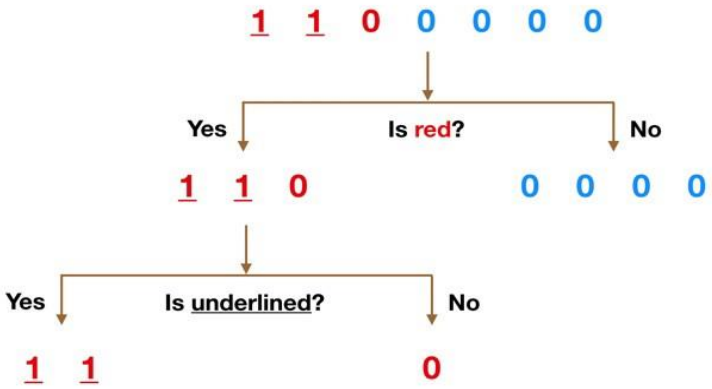


Decision Trees (DTs) are probably one of the most useful supervised learning algorithms out there. In a way, supervised learning is like learning with a teacher, and then apply that knowledge to new data. DTs algorithms are perfect to solve classification (where machines sort data into classes, like whether an email is spam or not) and regression (where machines predict values, like a property price) problems.

```
acc_results=[[],[]]
from sklearn.tree import DecisionTreeClassifier
dt_model = DecisionTreeClassifier()
#classification_model(dt_model,all_data_train_X,data['diagnosis'])
acc_results[0] += ["Decision Tree"]
acc_results[1] += [classification_model(dt_model,all_data_train_X,data['diagnosis'])]
```

```
Cross-Validation Score : 60.000%
Cross-Validation Score : 59.167%
Cross-Validation Score : 69.074%
Cross-Validation Score : 65.442%
Cross-Validation Score : 62.354%
```

1.2.4.2.2 Random Forest Classifier



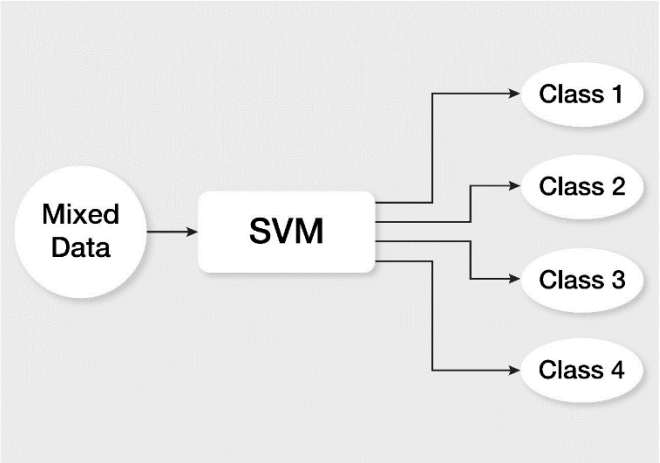
Random forest, like its name implies, consists of a large number of individual decision trees that operate as an ensemble. Each individual tree in the random forest spits out a class prediction and the class with the most votes becomes our model's prediction.

A large number of relatively uncorrelated models (trees) operating as a committee will outperform any of the individual constituent models.

```
from sklearn.ensemble import RandomForestClassifier

rfc_model=RandomForestClassifier(n_estimators=100)
#classification_model(rfc_model,all_data_train_X,data['diagnosis'])
acc_results[0] += ["Random Forest"]
acc_results[1] += [classification_model(rfc_model,all_data_train_X,data['diagnosis'])]
```

```
Cross-Validation Score : 80.000%
Cross-Validation Score : 81.667%
Cross-Validation Score : 84.074%
Cross-Validation Score : 81.237%
Cross-Validation Score : 79.990%
```

1.2.4.2.3 SVM



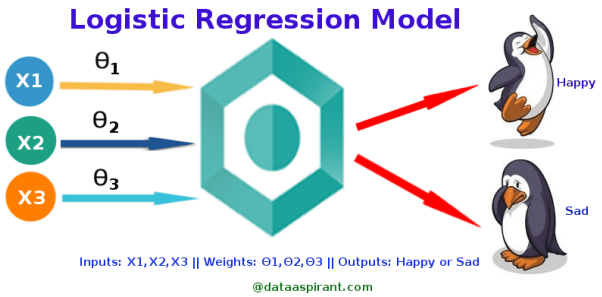
SVM is a linear model for classification and regression problems. It can solve linear and non-linear problems and work well for many practical problems. The idea of SVM is simple: The algorithm creates a line or a hyperplane which separates the data into classes.

```
from sklearn.svm import SVC
svc = SVC(kernel = 'rbf',gamma = 'auto' )
svc_n = SVC(gamma = 'auto')
```

```
svm_model = svm.SVC()
#classification_model(svm_model,all_data_train_X,data['diagnosis'])
acc_results[0] += ["SVM"]
acc_results[1] += [classification_model(svm_model,all_data_train_X,data['diagnosis'])]
```

```
Cross-Validation Score : 80.000%
Cross-Validation Score : 81.667%
Cross-Validation Score : 84.074%
Cross-Validation Score : 81.237%
Cross-Validation Score : 79.990%
```

1.2.4.2.4 Logistic Regression



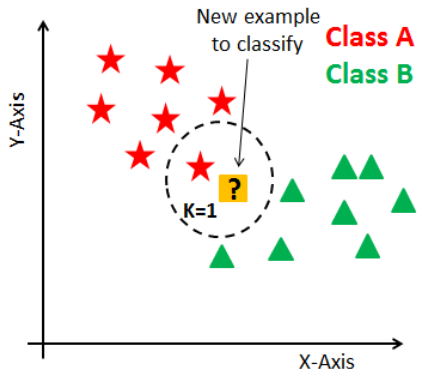
Logistic Regression is used when the dependent variable(target) is categorical.

In our model, we can use Logistic Regression to predict whether the IDC is positive(1) or negative(0).

```
from sklearn.linear_model import LogisticRegression
lr_model=LogisticRegression()
#classification_model(lr_model,all_data_train_X,data['diagnosis'])
acc_results[0] += ['Logistic Regression']
acc_results[1] += [classification_model(lr_model,all_data_train_X,data['diagnosis'])]
```

```
Cross-Validation Score : 80.000%
Cross-Validation Score : 77.500%
Cross-Validation Score : 81.296%
Cross-Validation Score : 79.154%
Cross-Validation Score : 80.823%
```

1.2.4.2.5 K Neighbors



KNN is a non-parametric and lazy learning algorithm. Non-parametric means there is no assumption for underlying data distribution. In other words, the model structure determined from the dataset. This will be very helpful in practice where most of the real world datasets do not follow mathematical theoretical assumptions. Lazy algorithm means it does not need any training data points for model generation. All training data used in the testing phase. This makes training faster and testing phase slower and costlier.

```
from sklearn.neighbors import KNeighborsClassifier
kn_model = KNeighborsClassifier()
#classification_model(kn_model,all_data_train_X,data['diagnosis'])
acc_results[0] += ["K Neighbors"]
acc_results[1] += [classification_model(kn_model,all_data_train_X,data['diagnosis'])]
```

```
Cross-Validation Score : 70.000%
Cross-Validation Score : 76.667%
Cross-Validation Score : 80.741%
Cross-Validation Score : 76.465%
Cross-Validation Score : 73.672%
```

1.2.4.2.6 Naïve Bayesian

A Naive Bayes classifier is a probabilistic machine learning model that's used for classification task. The crux of the classifier is based on the Bayes theorem.

$$P(A|B) = \frac{P(B|A)P(A)}{P(B)}$$

```
from sklearn.naive_bayes import GaussianNB
nb_model=GaussianNB()
#classification_model(nb_model,all_data_train_X,data['diagnosis'])
acc_results[0] += ["Naive Bayesian"]
acc_results[1] += [classification_model(nb_model,all_data_train_X,data['diagnosis'])]
```

```
Cross-Validation Score : 80.000%
Cross-Validation Score : 85.833%
Cross-Validation Score : 86.852%
Cross-Validation Score : 78.775%
Cross-Validation Score : 80.520%
```

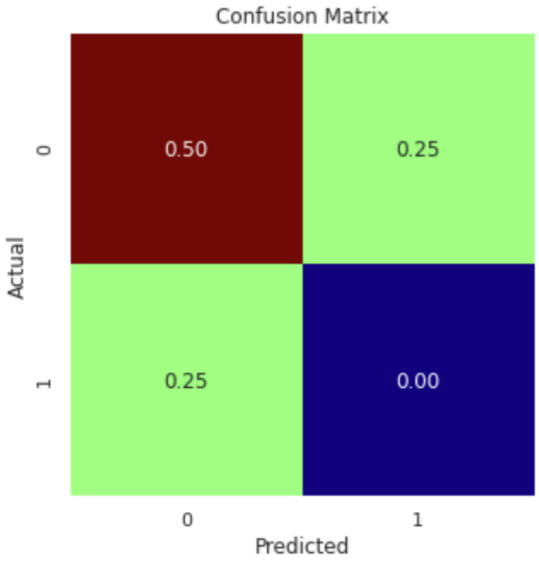
2. Results and Analysis:

Let's now define the most basic terms:

- **true positives (TP):** These are cases in which we predicted 1 (Positive), and they do have the IDC.
- **true negatives (TN):** We predicted 0 (Negative), and they don't have the IDC.
- **false positives (FP):** We predicted 1(Positive), but they don't actually have the IDC.
- **false negatives (FN):** We predicted 0(Negative), but they actually do have the IDC.

2.1 Decision Tree:

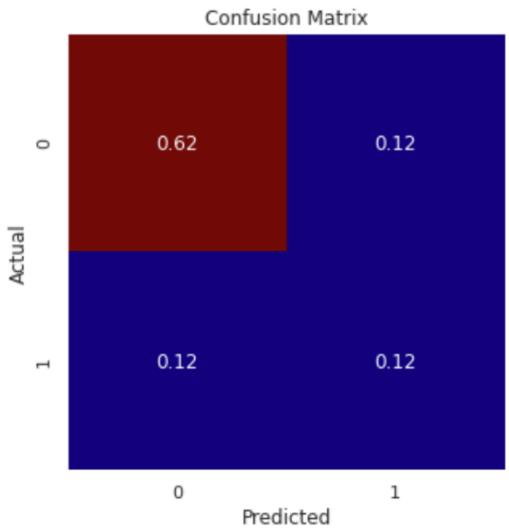
- There are two possible predicted classes: "1" and "0". If we were predicting the presence of a IDC, for example, "1" would mean they are positive, and "0" would mean they don't are negative.
- The classifier made a total of 100%
- Out of those 100% , the classifier predicted "1" 0.25, and "0" 0.75.
- In reality, 25% samples in the sample have the disease, and 75% samples do not.
- Confusion Matrix Accuracy=0.50+0=0.5
- K-Fold-DT Accuracy=62.354%




```
The Accuracy Is : 62.354%
[0.6, 0.5833333333333334, 0.8888888888888888, 0.5454545454545454, 0.5]
```

2.2 Random Forest Classifier:

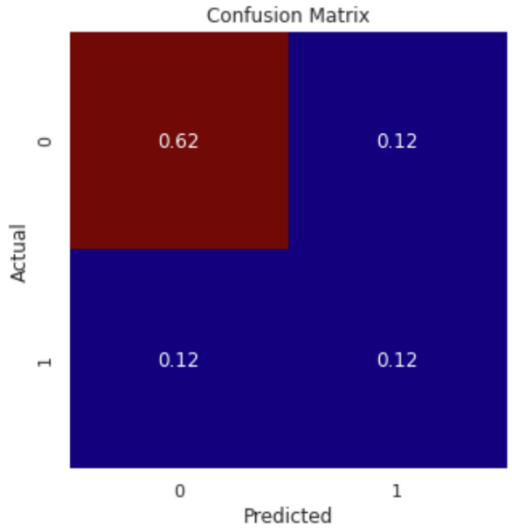
- There are two possible predicted classes: "1" and "0". If we were predicting the presence of a IDC, for example, "1" would mean they are positive, and "0" would mean they don't are negative.
- The classifier made a total of 100%
- Out of those 100% , the classifier predicted "1" 0.24, and "0" 0.76.
- In reality, 24% samples in the sample have the disease, and 76% samples do not.
- Confusion Matrix Accuracy= 0.62+0.12=0.74
- K-Fold-RF Accuracy=80%




```
The Accuracy Is : 79.990%
[0.8, 0.8333333333333334, 0.8888888888888888, 0.7272727272727273, 0.75]
```

2.3 SVM:

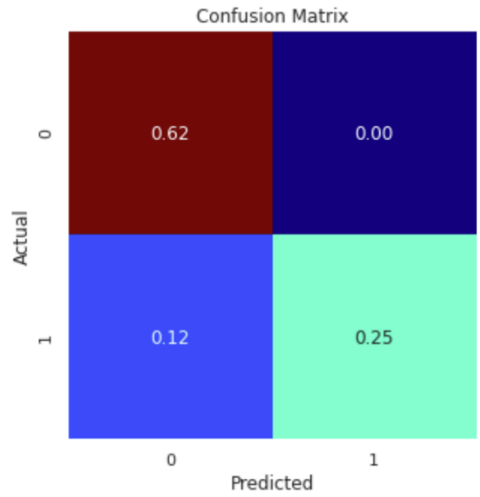
- There are two possible predicted classes: "1" and "0". If we were predicting the presence of a IDC, for example, "1" would mean they are positive, and "0" would mean they don't are negative.
- The classifier made a total of 100%
- Out of those 100% , the classifier predicted "1" 0.24, and "0" 0.76.
- In reality, 24% samples in the sample have the disease, and 76% samples do not.
- Confusion Matrix Accuracy= 0.62+0.12=0.74
- K-Fold-SVM Accuracy=80%



The Accuracy Is : 79.990%
[0.8, 0.8333333333333334, 0.8888888888888888, 0.7272727272727273, 0.75]

2.4 Logistic Regression:

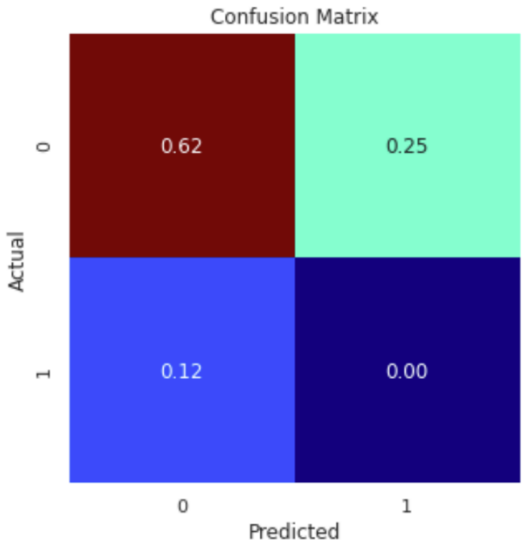
- There are two possible predicted classes: "1" and "0". If we were predicting the presence of a IDC, for example, "1" would mean they are positive, and "0" would mean they don't are negative.
- The classifier made a total of 100%
- Out of those 100% , the classifier predicted "1" 0.25, and "0" 0.75.
- In reality, 37% samples in the sample have the disease, and 63% samples do not.
- Confusion Matrix Accuracy= 0.62+0.25=0.87
- K-Fold-LR Accuracy=81%



The Accuracy Is : 80.823%
[0.8, 0.75, 0.8888888888888888, 0.7272727272727273, 0.875]

2.5 K-Neighbors:

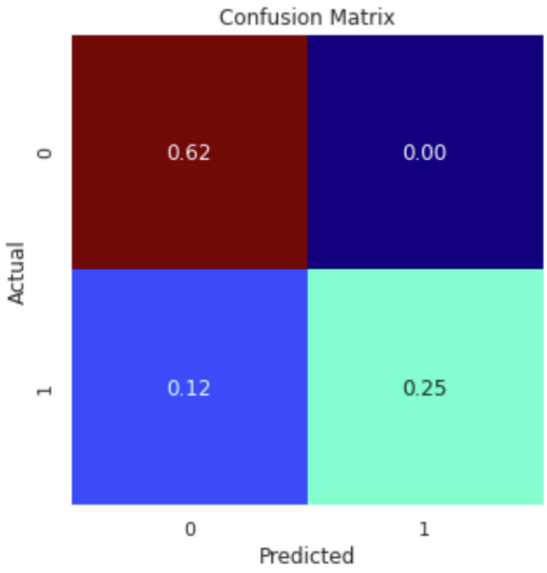
- There are two possible predicted classes: "1" and "0". If we were predicting the presence of a IDC, for example, "1" would mean they are positive, and "0" would mean they don't are negative.
- The classifier made a total of 100%
- Out of those 100% , the classifier predicted "1" 0.25, and "0" 0.75.
- In reality, 10% samples in the sample have the disease, and 90% samples do not.
- Confusion Matrix Accuracy= 0.62+0=0.62
- K-Fold-KNN Accuracy=74%




```
The Accuracy Is : 73.672%
[0.7, 0.8333333333333334, 0.8888888888888888, 0.6363636363636364, 0.625]
```

2.6 Naïve Bayesian:

- There are two possible predicted classes: "1" and "0". If we were predicting the presence of a IDC, for example, "1" would mean they are positive, and "0" would mean they don't are negative.
- The classifier made a total of 100%
- Out of those 100% , the classifier predicted "1" 0.25, and "0" 0.75.
- In reality, 37% samples in the sample have the disease, and 63% samples do not.
- Confusion Matrix Accuracy= 0.62+0.25=0.87
- K-Fold-NB Accuracy=81%
-



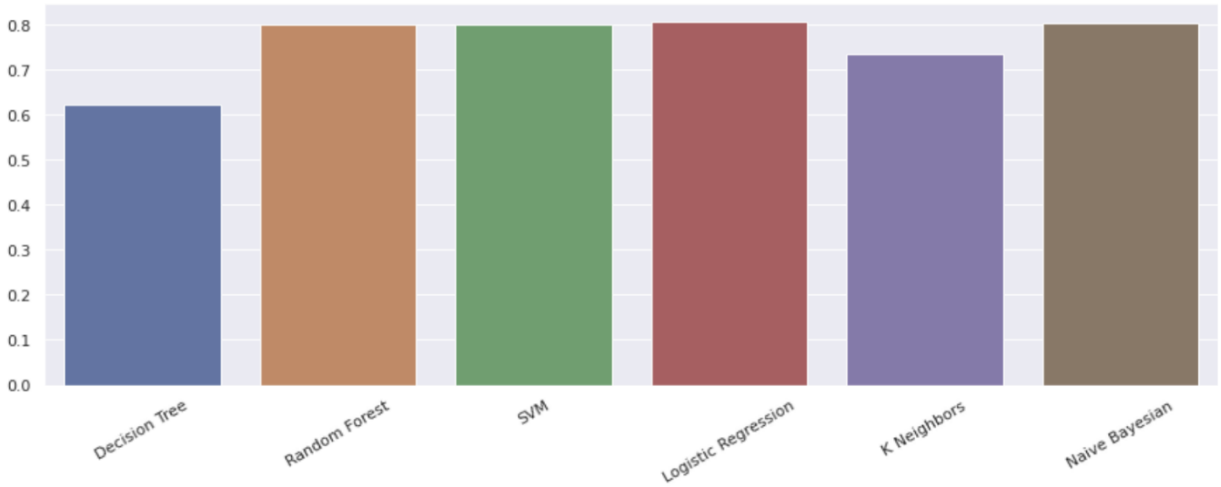

```
The Accuracy Is : 80.520%
[0.8, 0.9166666666666666, 0.8888888888888888, 0.5454545454545454, 0.875]
```

2.8 Accuracy Comparison

Then we counted the Accuracy of each model and made a chart.

As we can see that Logistic Regression in this data has the most high accuracy and Decision Tree has the lowest accuracy.

Besides, Random Forest, SVM and Naïve Bayesian also perform very well.



:
```
import matplotlib.pyplot as plt
import seaborn as sns
f, ax = plt.subplots(figsize=(15,5))
figsize=(15,5)
ticksize = 14
titlesize = ticksize + 8
labelsize = ticksize + 5
params = {'figure.figsize' : figsize,
          'axes.labelsize' : labelsize,
          'axes.titlesize' : titlesize,
          'xtick.labelsize': ticksize,
          'ytick.labelsize': ticksize}

plt.rcParams.update(params)

plt.xticks(rotation=30)
sns.barplot(acc_results[0], [np.mean(e) for e in acc_results[1]])
```

Conclusion

With the development of the times, huge technical and scientific made great advances, and the living standard of human beings has been improved enormously. However, woman's rights have not been promoted. Big mental pressure, big labor intensity and lack of feeling safety are easy to cause breast cancer's occurrence. Our project is going to help the expertise of the pathologist automatically detect and locate tumor tissue cells and to speed up the detecting process.

Deep learning is helpful to the automatic detection and localization of tumor cells, and accelerate the detection and localization process of tumor cells. In order to fully tap this potential, people can use a large number of tissue image data from different hospitals evaluated by different experts to construct pipelines. This can overcome the dependence on pathologists, which is especially useful in areas where there are no experts.

We can make some improvements in the future：

- Because of the storage limitation of the kaggle cloud, we can't train more data sets. As we all know, the more training data sets, the more accurate the training results.
- Because the classes of IDC versus no IDC are imbalanced, when training data, we need to shuffle and reorganize the data.
- Before building the model, we should carefully analyze the data and select the appropriate training model according to the characteristics of the data.

Description of Dataset (Link):

https://www.kaggle.com/paultimothymooney/breast-histopathology-images/download

The original dataset consisted of 162 whole mount slide images of Breast Cancer (BCa) specimens scanned at 40x. From that, 277,524 patches of size 50 x 50 were extracted (198,738 IDC negative and 78,786 IDC positive). Each patch's file name is of the format: uxXyYclassC.png — > example 10253idx5x1351y1101class0.png . Where u is the patient ID (10253idx5), X is the x-coordinate of where this patch was cropped from, Y is the y-coordinate of where this patch was cropped from, and C indicates the class where 0 is non-IDC and 1 is IDC.

Reference:

[1] https://www.cancer.org/cancer/breast-cancer.html

[2] https://www.breastcancer.org/symptoms/types/idc

[3] McLachlan, Geoffrey J.; Do, Kim-Anh; Ambroise, Christophe (2004). Analyzing microarray gene expression data. Wiley.

[4] Deng,H.; Runger, G.; Tuv, E. (2011). Bias of importance measures for multi-valued attributes and solutions. Proceedings of the 21st International Conference on Artificial Neural Networks (ICANN).

[5] https://en.wikipedia.org/wiki/Decision_tree#cite_note-7

[6] Ho, Tin Kam (1995). Random Decision Forests (PDF). Proceedings of the 3rd International Conference on Document Analysis and Recognition, Montreal, QC, 14–16 August 1995. pp. 278–282. Archived from the original (PDF) on 17 April 2016. Retrieved 5 June 2016.

[7] Ho TK (1998). "The Random Subspace Method for Constructing Decision Forests" (PDF). IEEE Transactions on Pattern Analysis and Machine Intelligence. 20 (8): 832–844. doi:10.1109/34.709601.

[8] Cortes, Corinna; Vapnik, Vladimir N. (1995). "Support-vector networks" (PDF). Machine Learning. 20 (3): 273–297. CiteSeerX 10.1.1.15.9362. doi:10.1007/BF00994018. S2CID 206787478.

[9]https://en.wikipedia.org/wiki/Principal_component_analysis#:~:text=Principal%20component%20analysis%20(PCA)%20is,components%20and%20ignoring%20the%20rest.

[10]https://towardsdatascience.com/principal-component-analysis-in-depth-understanding-through-image-visualization-892922f77d9f

[11]https://towardsdatascience.com/pca-and-svd-explained-with-numpy-5d13b0d2a4d8

[12]https://towardsdatascience.com/cross-validation-430d9a5fee22

[13]https://towardsdatascience.com/understanding-random-forest-58381e0602d2

[14]https://towardsdatascience.com/https-medium-com-pupalerushikesh-svm-f4b42800e989

[15]https://towardsdatascience.com/logistic-regression-detailed-overview-46c4da4303bc

[16]https://www.datacamp.com/community/tutorials/k-nearest-neighbor-classification-scikit-learn

[17] Piryonesi, S. Madeh; El-Diraby, Tamer E. (2021-02-01). "Using Machine Learning to Examine Impact of Type of Performance Indicator on Flexible Pavement Deterioration Modeling". Journal of Infrastructure Systems. 27 (2): 04021005. doi:10.1061/(ASCE)IS.1943-555X.0000602. ISSN 1076-0342.

[18] Hastie, Trevor; Tibshirani, Robert; Friedman, Jerome (2008). The Elements of Statistical Learning (2nd ed.). Springer. ISBN 0-387-95284-5.

[19]Tolles, Juliana; Meurer, William J (2016). "Logistic Regression Relating Patient Characteristics to Outcomes". JAMA. 316 (5): 533–4. doi:10.1001/jama.2016.7653. ISSN 0098-7484. OCLC 6823603312. PMID 27483067

[20] Piryonesi S. Madeh; El-Diraby Tamer E. (2020-06-01). "Role of Data Analytics in Infrastructure Asset Management: Overcoming Data Size and Quality Problems". Journal of Transportation Engineering, Part B: Pavements. 146 (2): 04020022. doi:10.1061/JPEODX.0000175

[21] Hastie, Trevor. (2001). The elements of statistical learning : data mining, inference, and prediction : with 200 full-color illustrations. Tibshirani, Robert., Friedman, J. H. (Jerome H.). New York: Springer. ISBN 0-387-95284-5. OCLC 46809224

Russell, Stuart; Norvig, Peter (2003) [1995]. Artificial Intelligence: A Modern Approach (2nd ed.). Prentice Hall. ISBN 978-0137903955.

Shuyao Zhang Part:

Methodology

Algorithms & Methodology:

- **CNN**

  Convolutional neural network (CNN) is one of the neural network models that is used to train large number of datasets. It extracts features of object by certain model and categorizes, identifies, or predicts such object.

  Advantages:

  It has shared convolution sum, which would increase the efficiency of dealing with high dimensional datasets.

  Also, we need to select characteristics manually to get the weight of different characteristic. This would make the classification of characteristics more precise.

  Disadvantages:

  Since we need to handle the characteristics manually, we must do a lot of work to investigate the parameters.

  Moreover, big datasets are necessary for training. The training process requires a good GPU.

Problems and Solutions

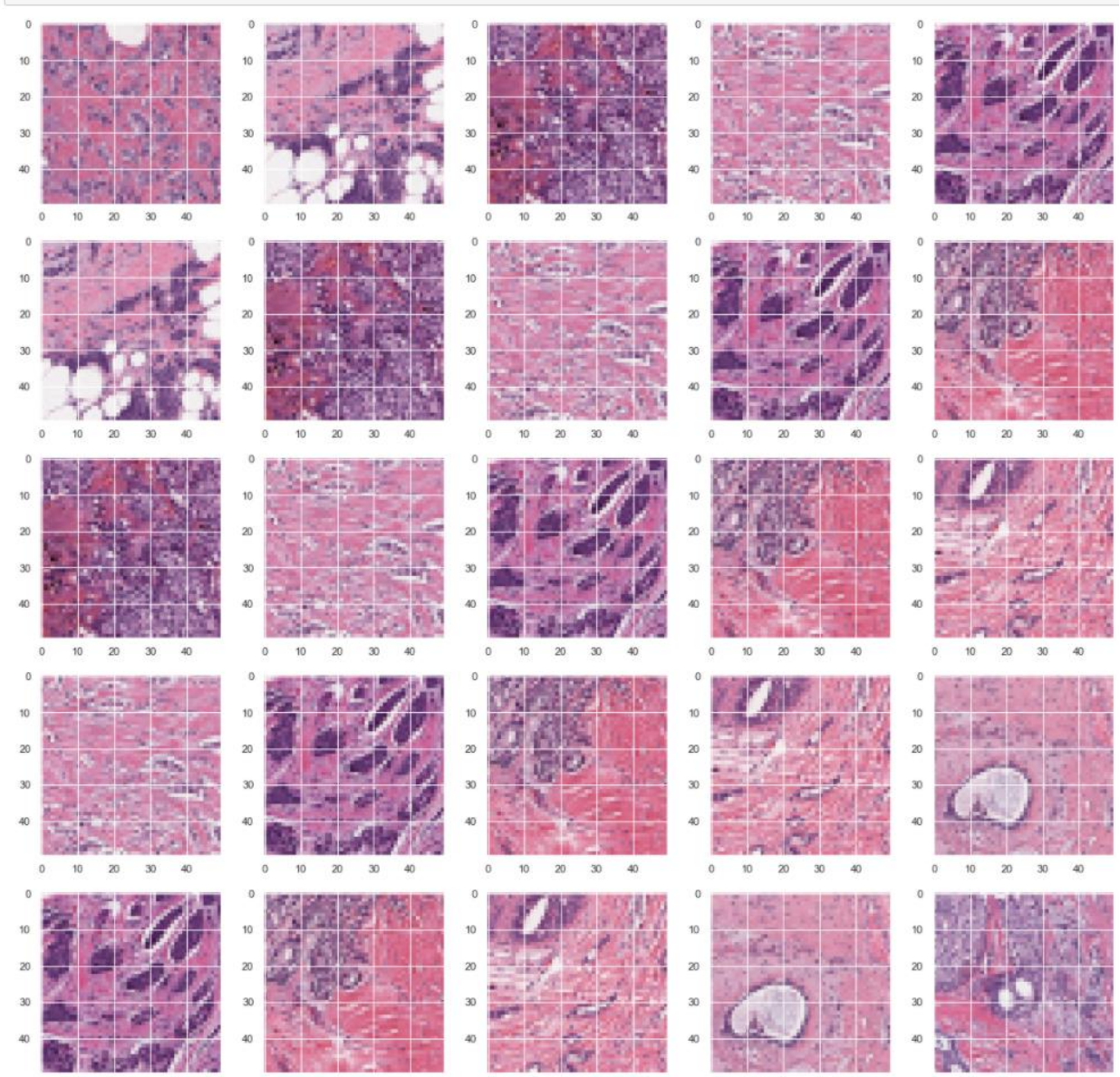
Import Packages:

```
import warnings
warnings.filterwarnings("ignore", category=DeprecationWarning)
warnings.filterwarnings("ignore", category=UserWarning)
warnings.filterwarnings("ignore", category=FutureWarning)

import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)
import matplotlib.pyplot as plt
%matplotlib inline
import seaborn as sns
sns.set()
from PIL import Image

import torch
import torch.nn as nn
import torch.optim as optim
from torch.optim.lr_scheduler import ReduceLROnPlateau, StepLR, CyclicLR
import torchvision
from torchvision import datasets, models, transforms
from torch.utils.data import Dataset, DataLoader
import torch.nn.functional as F

from sklearn.model_selection import train_test_split, StratifiedKFold
from sklearn.utils.class_weight import compute_class_weight
from sklearn import metrics

from glob import glob
from skimage.io import imread
from os import listdir
from skimage import io
from sklearn.svm import SVC
from sklearn.metrics import classification_report

import time
import copy
from tqdm import tqdm_notebook as tqdm
from sklearn import svm
import gc
from sklearn.decomposition import IncrementalPCA
```

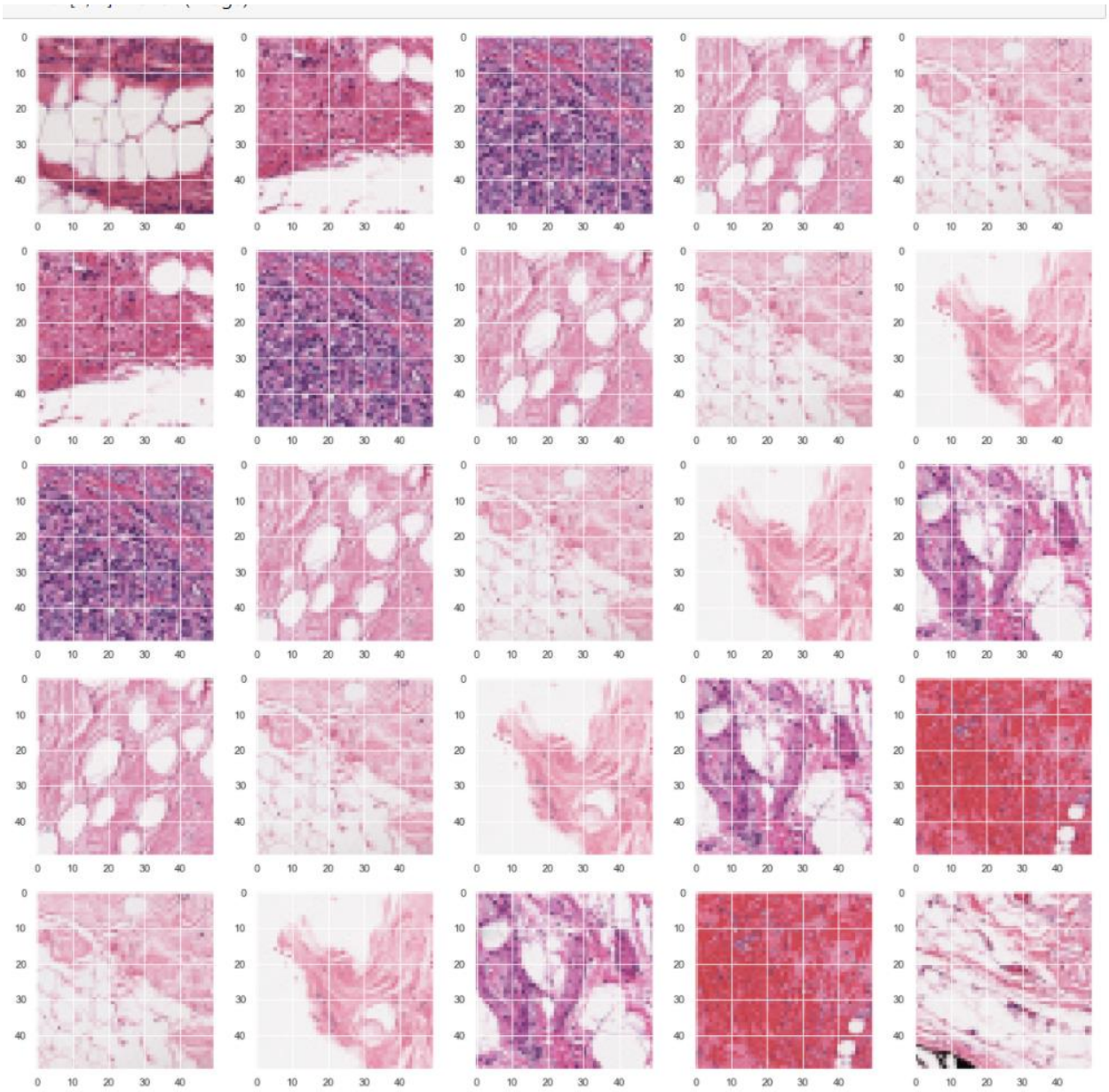
Datas Information:

We use images as our dataset. There are two types of images. One is the tissue with IDC, and anther is the tissue without IDC, as known as NIDC.

This is the few examples of IDC:

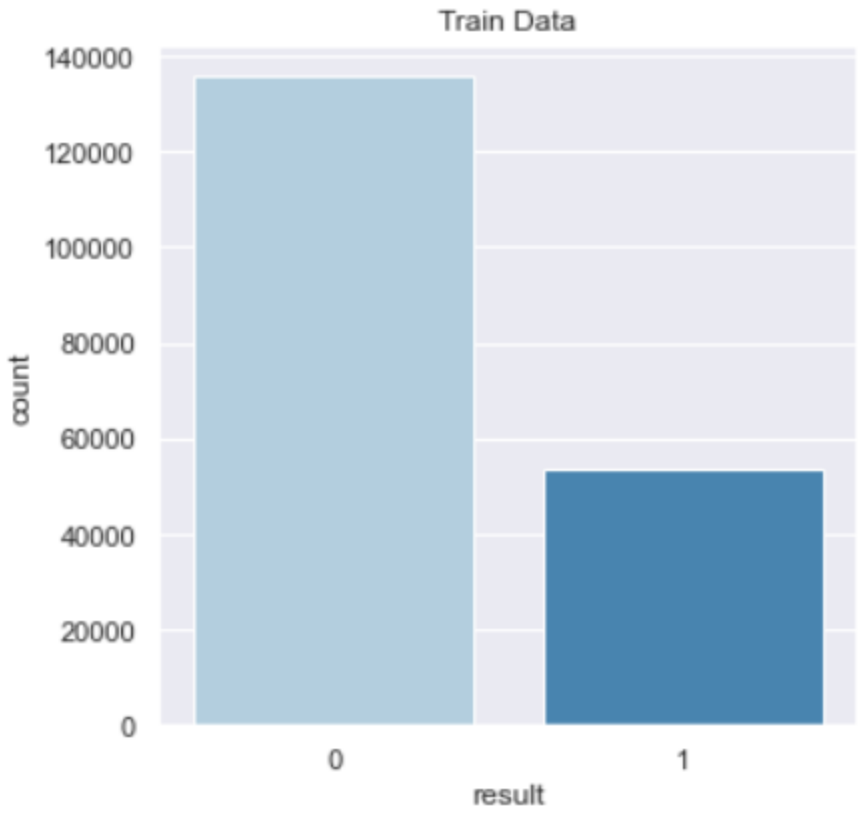


This is the few examples of NIDC:

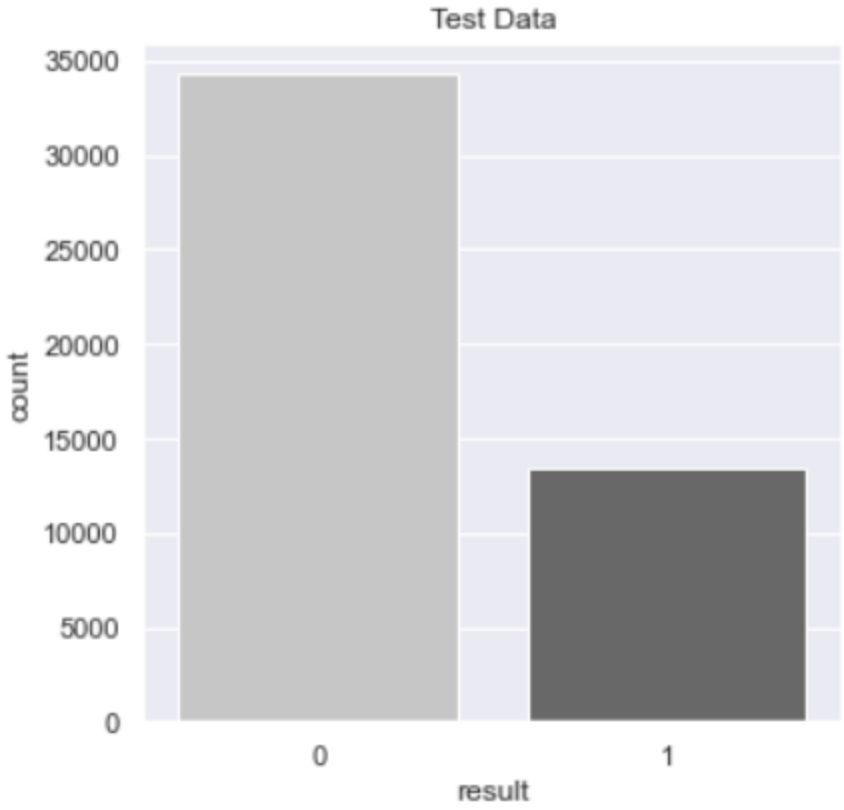


The difference between two groups above is visible by human eyes. Tissue with IDC is more intensive than NIDC tissue. Also, the color of IDC tissue seems deeper.

We also need to split the dataset. For CNN part, I split the dataset into three parts. The first one is training part, takes about 70% of the dataset.



The second part is test data. It takes 15% of the dataset.



The last part takes 15% of the dataset as well. It is the Dev data (Validation data)

The next step for us is to extract the characteristics of those images and use them to train the model.

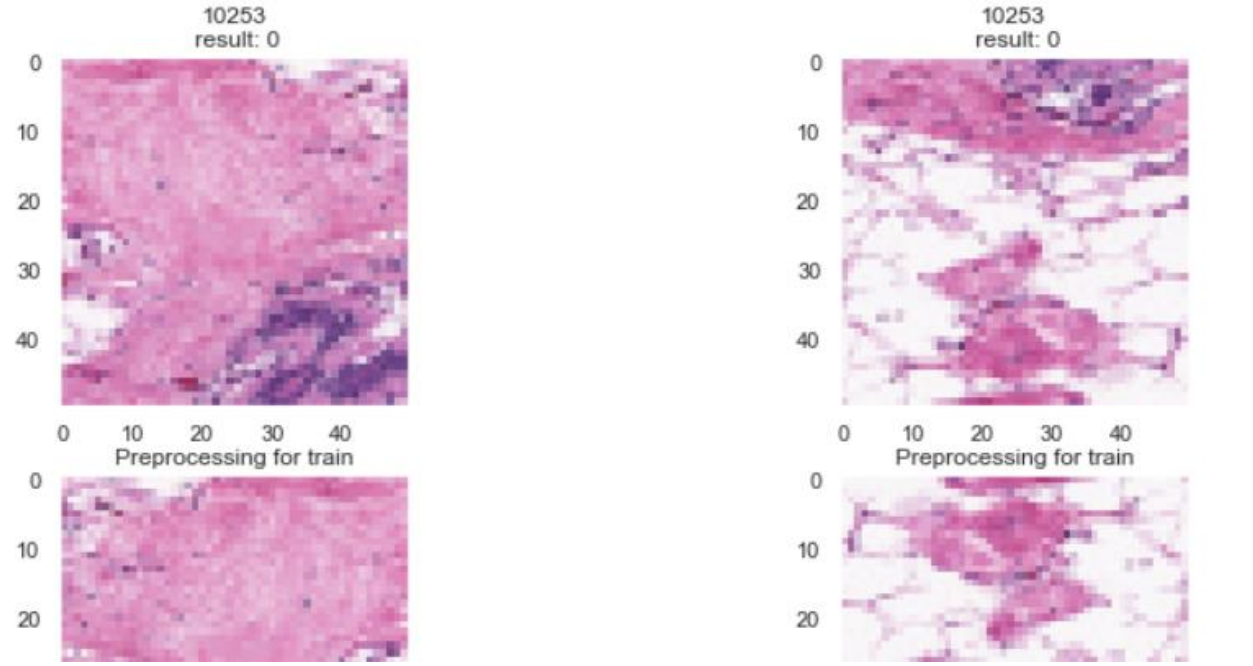
Image Pre-Processing:

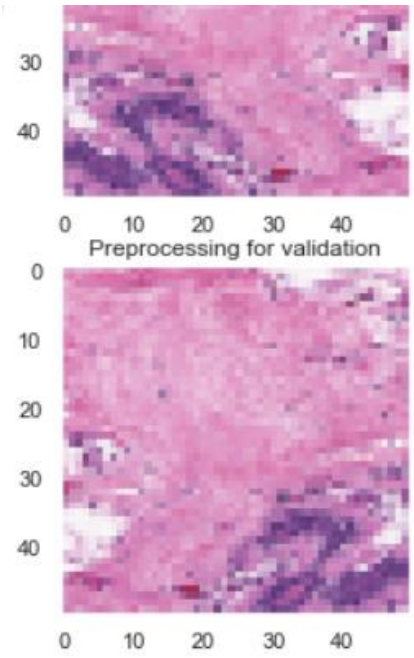
Instead of splitting data into two parts-30% test data and 70% training data, we categorized the data into 3 parts-15% test data, 15% validation data and 70% training data. The reason we did it is because we wanted to make the test data covering a broad range of variations.

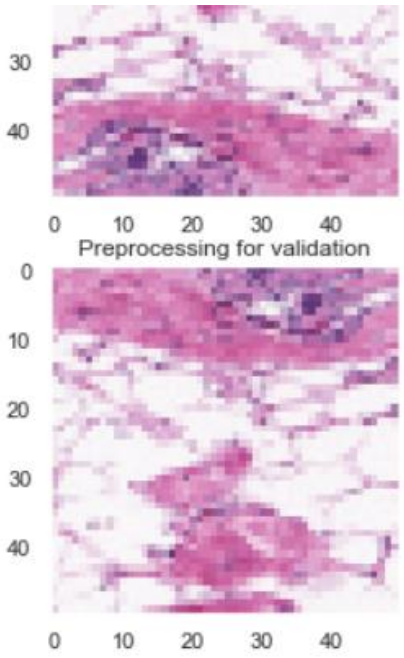
The way we process the data is that we use transform from pytorch to generate image datasets. First, we resize the images to the desired input shape. Since CNN is translational but not rotational invariant, performing horizontal and vertical flips is a good idea during training.

```
]: def resize_image(key="train", plot=False):
    train_sequence = [transforms.Resize((50,50)),
                      transforms.RandomHorizontalFlip(),
                      transforms.RandomVerticalFlip()]
    val_sequence = [transforms.Resize((50,50))]
    if plot==False:
        train_sequence.extend([
            transforms.ToTensor(),
            transforms.Normalize([0.485, 0.456, 0.406], [0.229, 0.224, 0.225])])
        val_sequence.extend([
            transforms.ToTensor(),
            transforms.Normalize([0.485, 0.456, 0.406], [0.229, 0.224, 0.225])])

    data_transforms = {'train': transforms.Compose(train_sequence),'val': transforms.Compose(val_sequence)}
    return data_transforms[key]
```

After flipping, the images looked like this:







Moreover, we need to convert the dataset into RGB:

```
]: class createDataset(Dataset):

    def __init__(self, df, transform=None):
        self.states = df
        self.transform=transform

    def __len__(self):
        return len(self.states)

    def __getitem__(self, idx):
        patient_id = self.states.patient_id.values[idx]
        x_coord = self.states.x.values[idx]
        y_coord = self.states.y.values[idx]
        image_path = self.states.path.values[idx]
        image = Image.open(image_path)
        image = image.convert('RGB')

        if self.transform:
            image = self.transform(image)

        if "result" in self.states.columns.values:
            result = np.int(self.states.target.values[idx])
        else:
            result = None

        return {"image": image,
                "label": target,
                "patient_id": patient_id,
                "x": x_coord,
                "y": y_coord}
```

The first layer of CNN extracts edges, and only the last layer contains the specific features.

Description of Dataset (Link):

https://www.kaggle.com/paultimothymooney/breast-histopathology-images/download

The link above is the download link of the dataset we chose. The name of the file we download is “archive.zip”. After downloading, extract the zip file and we only kept the file named

"**IDC_regular_ps50_idx5**". Before using the data, we renamed the folder into "data" for future use. Store the "data" folder with the ipynb file.

Results and Analysis:

CNN:

```
|: dev_predictions
```

|:

	true	predicted	proba	x	y	patient_id
0	0	0	0.003411	501.0	2001.0	9135
1	0	0	0.111403	1251.0	501.0	9135
2	0	0	0.234970	351.0	351.0	9135
3	0	0	0.599228	351.0	1051.0	9135
4	0	0	0.096663	1401.0	151.0	9135
...	...	...	...	...	...	...
40827	1	1	0.958453	2901.0	501.0	9255
40828	1	1	0.965054	1351.0	1101.0	9255
40829	1	1	0.845481	2751.0	1101.0	9255
40830	1	1	0.988100	2801.0	501.0	9255
40831	1	1	0.820634	2851.0	751.0	9255

40832 rows × 6 columns

In [63]:
```
test_predictions
```

Out[63]:

	true	predicted	proba	x	y	patient_id
0	0	0	0.128713	1201.0	2401.0	12906
1	0	0	0.347586	151.0	1001.0	12906
2	0	0	0.175564	1301.0	1251.0	12906
3	0	0	0.103502	1351.0	551.0	12906
4	0	0	0.224504	601.0	1551.0	12906
...	...	...	...	...	...	...
39600	1	1	0.680131	1651.0	701.0	14304
39601	1	1	0.814124	1201.0	851.0	14304
39602	1	0	0.551968	1201.0	901.0	14304
39603	1	1	0.894965	851.0	101.0	14304
39604	1	1	0.948164	1401.0	1151.0	14304

39605 rows × 6 columns

In [64]:
```
answer = test_predictions
length = answer.shape[0]
correct_predicted = 0

for i in range(length):
    if answer.true[i] == answer.predicted[i]:
        correct_predicted += 1

print("Accuracy of test set: ", correct_predicted/length)
```

Accuracy of test set: 0.83355636914531

In [65]:
```
answer = dev_predictions
length = answer.shape[0]
correct_predicted = 0

for i in range(length):
    if answer.true[i] == answer.predicted[i]:
        correct_predicted += 1

print("Accuracy of validation set: ", correct_predicted/length)
```

Accuracy of validation set: 0.8304516065830722

Conclusion:

The result seems acceptable. Since we did not use the whole package of dataset because of running time limitations, the accuracy of this model could be improved if we keep feeding data into the model.