CO461 DATA WAREHOUSE AND DATA MINING

A REPORT ON THE PROJECT ENTITLED

Microaneurysm Detection Using Principal Component Analysis and Machine Learning Methods

SUBMITTED BY
HARDIK RANA (16CO138)
AMAL BYJU (16CO205)
ROSHAN LAL SAHU (16CO242)

VII SEMESTER B.Tech (CSE)

SUBMITTED TO
M. VENKATESAN

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

NATIONAL INSTITUTE OF TECHNOLOGY KARNATAKA

SURATHKAL

2019-2020

Table of Contents

Title

Microaneurysm Detection Using Principal Component Analysis and Machine Learning Methods

Introduction

DIABETIC Retinopathy (DR) is a progressive disease with almost no early symptoms of vision impairment, which is the leading cause of blindness prior to the age of 50. The first detectable sign of DR is the presence of microaneurysms (MAs), which result from leakage of tiny blood vessels in the retina and manifest themselves as small red circular spots on the surface of retinas. Early detection of MAs is critical for diagnosis and treatment of DR, which has led to a great deal of research towards automatic detection of MAs.

Progressive changes in the field of science and technology are making human life healthier, secure, more comfortable and livable. Automated diagnosis systems (ADSs) are providing services for the ease of humanity. ADSs perform a vital role in the early diagnosis of serious diseases. Diabetic retinopathy (DR) is a serious and widespread disease worldwide. Recently, the World Health Organization (WHO) has reported that diabetes will become the seventh highest death causing disease in the world by 2030. In this context, it is a big challenge to save the lives of patients affected by diabetes. Diabetic retinopathy is a common disease that exists in diabetic patients. In diabetic retinopathy, some lesions are produced in the eyes, which become a cause of non-reversible blindness with the passage of time. These types of lesions include abnormal retinal blood vessels, microaneurysm (MA), cotton wool spots, exudates, and hemorrhages

According to the disease severity scale [, diabetic retinopathy can be classified into five stages: non-DR, mild severe, moderately severe, severe, and proliferative DR. In this regard, many researchers have introduced different types of methods, architectures, models, and frameworks, and have performed a vital role in detecting lesions in the early stage of DR. Haloi et al. [1] introduced a method to detect cotton wool spots and exudates. Furthermore, Haloi [2] used deep neural networks to find microaneurysms in color retinal fundus images. Van Grinsven et al. [3] introduced a novel technique to detect rapid hemorrhages. Moreover, Srivastava et al. [4] obtained significant experimental results when finding hemorrhages and microaneurysms using multiple kernel methods.

On the other hand, the detection of the severity levels of DR is also an important task when curing an affected eye. Seoud et al. [5] developed a computer-aided system for the classification of fundus images using random forests [6]. On the basis of deep learning methods, Kuen et al. [7] promoted DCNN for diabetic retinopathy detection. Gulshan et al. [8] classified the fundus images into non-DR and moderate severe with the help of 54 American ophthalmologists and other medical researchers on more than 128 thousand DR images. Furthermore, Sankar et al. [9] classified the fundus images into non-DR, mild DR, and severe DR. Pratt et al. [10] proposed a robust technique to classify the severity levels of DR followed by the standards of the severity scale [11].

Background

Many existing MA detection methods rely on hand-crafted features, which are often based on low-level information. Low Level information is easily susceptible to signal drift artifacts and thus prevent reliable generalization among different research sites. A recent method [12] leveraged the use of deep learning for MA detection using a Stacked Sparse Autoencoder (SSAE). Deep learning approaches often learn high-level and robust attributes directly from the raw signal input, and have been successfully applied to various classification and recognition tasks [13]–[14]. In [12], small image patches were generated from the original fundus images and used by the SSAE to learn high-level features from pixel intensities. These patches were then classified as either MA or non-MA using the high-level features learned by SSAE

Several methods have been proposed for MA detection. Quellec et al. [15] explored the use of template matching in the wavelet domain. This method was further substantiated following the University of Iowa's release of the Retinopathy Online Challenge (ROC) database and subsequent competition for MA detection [16], in which the competition winner extended the wavelet domain template matching method. Ram et al. [17] created a clutter rejection strategy, in which successive stages of the algorithm eliminated more and more clutter, while passing most target MAs. One recent work [18] proposed a comprehensive grading system for DR based on classification of 16 features that captured shape, color, and intensity information and the features were extracted from candidate regions.

Feature representation is a common and critical step for MA detection methods. Most of the existing MA detection methods rely on hand-crafted features. Deep learning (DL) approaches, famous for the strong learning ability to automatically extract high-level features, have been applied to medical image processing in recent years. Xu et al. [17] utilized a Stacked Sparse Autoencoder (SSAE), to extract features for nuclei detection. Our work is inspired by [17] since MA detection shares many similarities with nuclei detection. In this work, we employed SSAE as the automatic feature extractor to learn from the image patches with and without MAs. The learned features are fed into a Softmax Classifier (SMC) to categorize a patch into lesion present or non-lesion-present classes.

Yang. et al. [19] developed an automatic DR analysis approach using two stage DCNN to localize and detect red lesions in the retinal fundus images. Additionally, the automatic system also classified the retinal fundus images on the basis of severity levels.

Gurudath et al. [20] proposed an automatic approach for the identification of DR from color retinal fundus images. Classification of the retinal fundus images was performed among three classes including normal, mild severe, and moderate severe. A Gaussian filtering model was used for retinal blood vessel segmentation on the input fundus images.

Cao et al. [21] analyzed the microaneurysm detection ability based on 25 × 25 image patches collected from retinal fundus images. Support vector machine (SVM), random forest (RF), and neural network (NN) were used for the classification of DR into five stages of severity. Principle component analysis (PCA) and random forest feature importance (RFFI) were applied for dimensionality reduction.

Nanni et al. [22] designed a computer vision system based on non-handcrafted and handcrafted features. For the non-handcrafted features, three approaches were applied including the compact binary descriptor (CBD), CNNs, and PCA. On the other hand, local phase quantization, completed local binary patterns, rotated local binary patterns, and some other algorithms were used for the handcrafted features. The proposed technique was applied to various datasets and obtained outstanding image classification results.

Algorithm

Input: Two datasets

1.DIARETDB1 - Calibration level 1

2.ROC (retinopathy online challenge)

Output:

The trained model (RF, NN, SVM - with and without PCA) applied to testing data will predict the accuracy and will return f_score and auc score of particular algorithm

Method:

(1) Patches of 300 x 300 will be generated from 50 ROC dataset's fundus images/ Patches of 300 x 300 will be generated from 89 DIARETDB1-1's fundus images (numpy array will be created that will contain all this patch-images - X)

(2) Each patch will be converted to only red plane (in red planes the MA's will be clearly visible)

(3) Yes/No Label will be assigned to each patch manually by checking whether that patch contains (numpy array will be created that will contain all this labels - y)

(4) After that each of the machine learning will be applied without-PCA and with PCA

(5) For random forest algorithm- without PCA:

- First random forest classifier with default parameters like number of trees=10,splitting criteria = Gini index, max_depth etc.. will be created
- After that with the use of function train_test_split 10% of the original data is given to testing data and rest 90% to training data.
- After that the created training set will be used to train the classifier (using fit function) we created initially.
- At last the accuracy,f_score,precision,recall,roc-auc score will be calculated for trained classifier by running it for testing data

For random forest algorithm - with PCA

- First the X_train and X_test data will be normalized using StandardScaler() function in scikit-learn. (This is a prerequisite step for applying PCA to any dataset)
- After that PCA() function of scikit-learn library will be applied to X_train and X_test to reduce the dimensionality of the dataset.
- After applying PCA all the steps of random forest algorithm-without PCA will be repeated and it's accuracy,f_score etc.. will be calculated by using the classifier on testing data

(6) For neural network algorithm- without PCA:

- First neural network classifier with default parameters like max_iter = 200,activation_funciton = 'relu', hidden_layer_size = (10,10,10) etc.. will be created
- After that with the use of function train_test_split 10% of the original data is given to testing data and rest to training data.
- After that the created training set will be used to train the classifier (using fit function) we created initially.
- At last the accuracy,f_score,precision,recall,roc-auc score will be calculated for trained classifier by running it for testing data

For neural network algorithm - with PCA

- First the X_train and X_test data will be normalized using StandardScaler() function in scikit-learn. (This is a prerequisite step for applying PCA to any dataset)
- After that PCA() function of scikit-learn library will be applied to X_train and X_test to reduce the dimensionality of the dataset.
- After applying PCA all the steps of neural network algorithm-without PCA will be repeated and it's accuracy,f_score etc.. will be calculated by using the classifier on testing data

(7) For support vector machine- without PCA:

- First support vector machine classifier with default parameters like kernel = rbf(radial basis function), degree=3, gamma = ‘auto_deprecated’ etc.. will be created
- After that with the use of function train_test_split 10% of the original data is given to testing data and rest to training data.
- After that the created training set will be used to train the classifier (using fit function) we created initially.
- At last the accuracy,f_score,precision,recall,roc-auc score will be calculated for trained classifier by running it for testing data

For support vector machine - with PCA

- First the X_train and X_test data will be normalized using StandardScaler() function in scikit-learn. (This is a prerequisite step for applying PCA to any dataset)
- After that PCA() function of scikit-learn library will be applied to X_train and X_test to reduce the dimensionality of the dataset.
- After applying PCA all the steps of support vector machine algorithm-without PCA will be repeated and it’s accuracy,f_score etc.. will be calculated by using the classifier on testing data

Dataset Description/Input

Dataset 1: DIARETDB1 - Diabetic retinopathy database - Calibration level 1

- This dataset contains a total of 89 fundus images,some of which have Microaneurysm and some will not have it.
- Each fundus image is of size 1500 x 1152 pixels
- From each fundus image we will generate patch of 300x300
- As some of the patches will not be of a size equal to others (they can be of shape rectangle), we will remove them.
- We will also remove patches which contains corner's black part of original fundus image.
- So by taking the above things into consideration, we generated a total of 1335 patches of 300x300 size from total 89 fundus images.
- So our one np_array input to all the classifier will be these images (X)
- For each of patches generated (1335),we will create another list which will contain labels (Yes/No) for whether the patch contains MA or not. (y)
- So if we use this dataset than these two (X,y) will be the inputs to our classifiers.

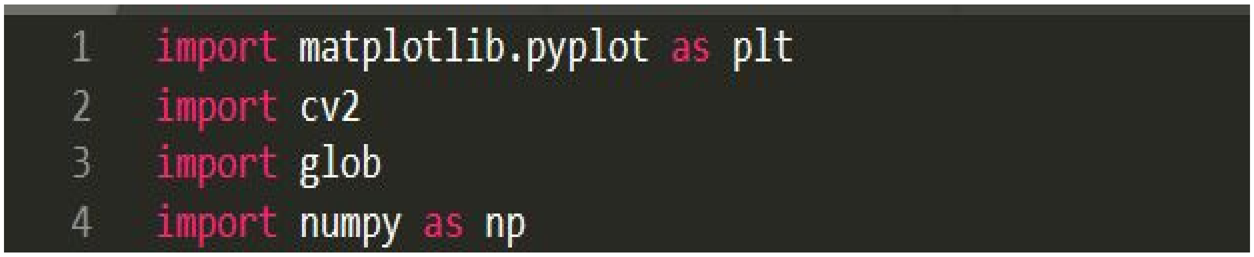
ROC: Retinopathy Online Challenge

- This dataset contains a total of 50 fundus images,some of which have
- Microaneurysm and some will not have it.
- Each fundus image is of size 1389 x 1383 pixels
- From each fundus image we will generate patch of 300x300
- As some of the patches will not be of a size equal to others (they can be of shape rectangle), we will remove them.
- We will also remove patches which contains corner's black part of original fundus image.
- So by taking the above things into consideration, we generated a total of 471 patches of 300x300 size from total 50 fundus images.
- So our one np_array input to all the classifier will be these images (X)
- For each of patches generated (471),we will create another list which will contain labels (Yes/No) for whether the patch contains MA or not. (y)
- So if we use this dataset than these two (X,y) will be the inputs to our classifiers.

Implementation using Tools

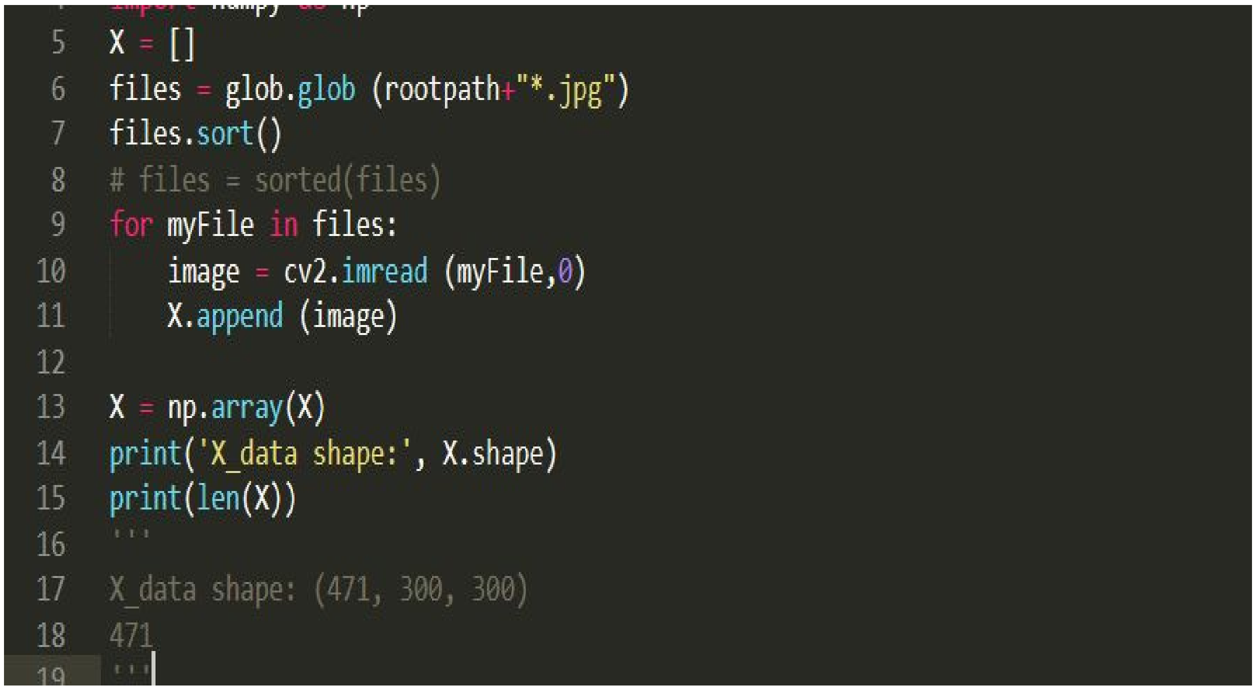
Without PCA

Step 1: importing the modules



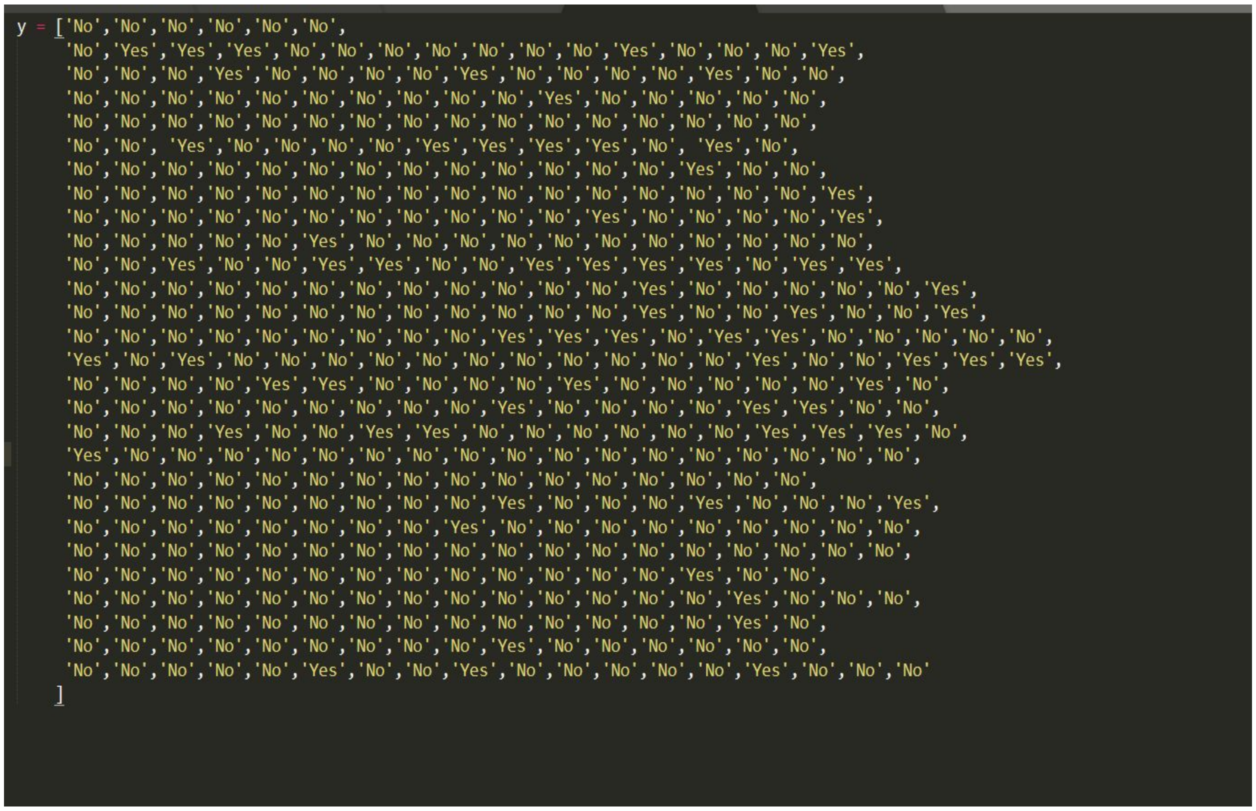
```python
import matplotlib.pyplot as plt
import cv2
import glob
import numpy as np
```

Step 2: Getting the generated dataset (patches of 300x300 size)



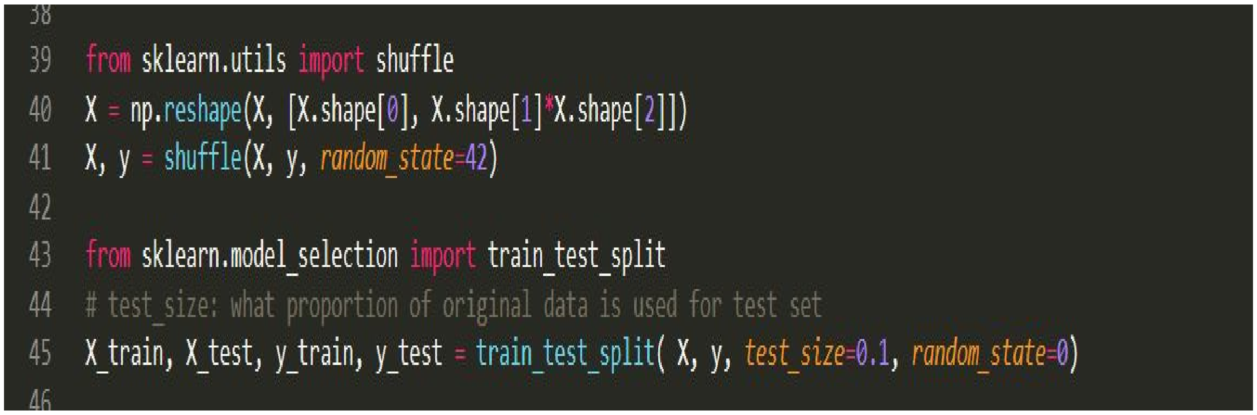
```python
X = []
files = glob.glob (rootpath+"*.jpg")
files.sort()
# files = sorted(files)
for myFile in files:
    image = cv2.imread (myFile,0)
    X.append (image)

X = np.array(X)
print('X_data shape:', X.shape)
print(len(X))
'''
X_data shape: (471, 300, 300)
471
'''
```

Step 3: labelling the dataset



```
y = ['No','No','No','No','No','No',
     'No','Yes','Yes','Yes','No','No','No','No','No','No','No','Yes','No','No','No','Yes',
     'No','No','No','Yes','No','No','No','No','Yes','No','No','No','No','Yes','No','No',
     'No','No','No','No','No','No','No','No','No','No','Yes','No','No','No','No','No',
     'No','No','No','No','No','No','No','No','No','No','No','No','No','No','No','No',
     'No','No', 'Yes','No','No','No','No','Yes','Yes','Yes','Yes','No', 'Yes','No',
     'No','No','No','No','No','No','No','No','No','No','No','No','No','Yes','No','No',
     'No','No','No','No','No','No','No','No','No','No','No','No','No','No','No','No','Yes',
     'No','No','No','No','No','No','No','No','No','No','No','Yes','No','No','No','No','Yes',
     'No','No','No','No','No','Yes','No','No','No','No','No','No','No','No','No','No','No',
     'No','No','Yes','No','No','Yes','Yes','No','No','Yes','Yes','Yes','Yes','No','Yes','Yes',
     'No','No','No','No','No','No','No','No','No','No','No','No','Yes','No','No','No','No','No','Yes',
     'No','No','No','No','No','No','No','No','No','No','No','No','Yes','No','No','Yes','No','No','Yes',
     'No','No','No','No','No','No','No','No','No','Yes','Yes','Yes','No','Yes','Yes','No','No','No','No','No',
     'Yes','No','Yes','No','No','No','No','No','No','No','No','No','No','No','Yes','No','No','Yes','Yes','Yes',
     'No','No','No','No','Yes','Yes','No','No','No','No','Yes','No','No','No','No','No','Yes','No',
     'No','No','No','No','No','No','No','No','No','Yes','No','No','No','No','Yes','Yes','No','No',
     'No','No','No','Yes','No','No','Yes','Yes','No','No','No','No','No','No','Yes','Yes','Yes','No',
     'Yes','No','No','No','No','No','No','No','No','No','No','No','No','No','No','No','No','No',
     'No','No','No','No','No','No','No','No','No','No','No','No','No','No','No','No',
     'No','No','No','No','No','No','No','No','No','Yes','No','No','No','Yes','No','No','No','Yes',
     'No','No','No','No','No','No','No','No','Yes','No','No','No','No','No','No','No','No','No',
     'No','No','No','No','No','No','No','No','No','No','No','No','No','No','No','No','No','No',
     'No','No','No','No','No','No','No','No','No','No','No','No','No','Yes','No','No',
     'No','No','No','No','No','No','No','No','No','No','No','No','No','No','Yes','No','No','No',
     'No','No','No','No','No','No','No','No','No','No','No','No','No','No','Yes','No',
     'No','No','No','No','No','No','No','No','No','Yes','No','No','No','No','No','No',
     'No','No','No','No','No','Yes','No','No','Yes','No','No','No','No','No','Yes','No','No','No'
   ]
```

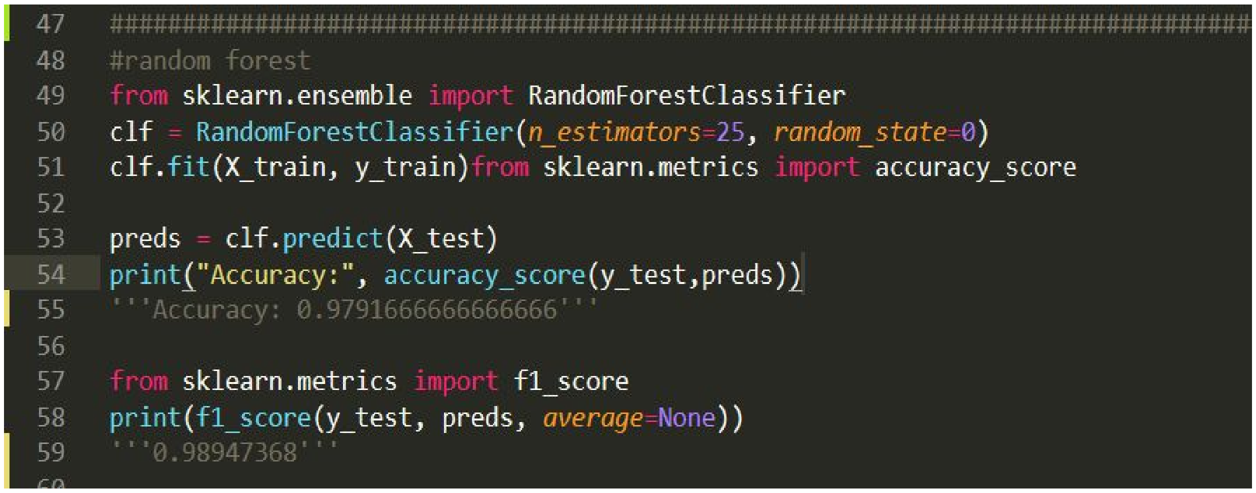
Step 4: Splitting the dataset into training and testing of the model



```
from sklearn.utils import shuffle
X = np.reshape(X, [X.shape[0], X.shape[1]*X.shape[2]])
X, y = shuffle(X, y, random_state=42)

from sklearn.model_selection import train_test_split
# test_size: what proportion of original data is used for test set
X_train, X_test, y_train, y_test = train_test_split( X, y, test_size=0.1, random_state=0)
```

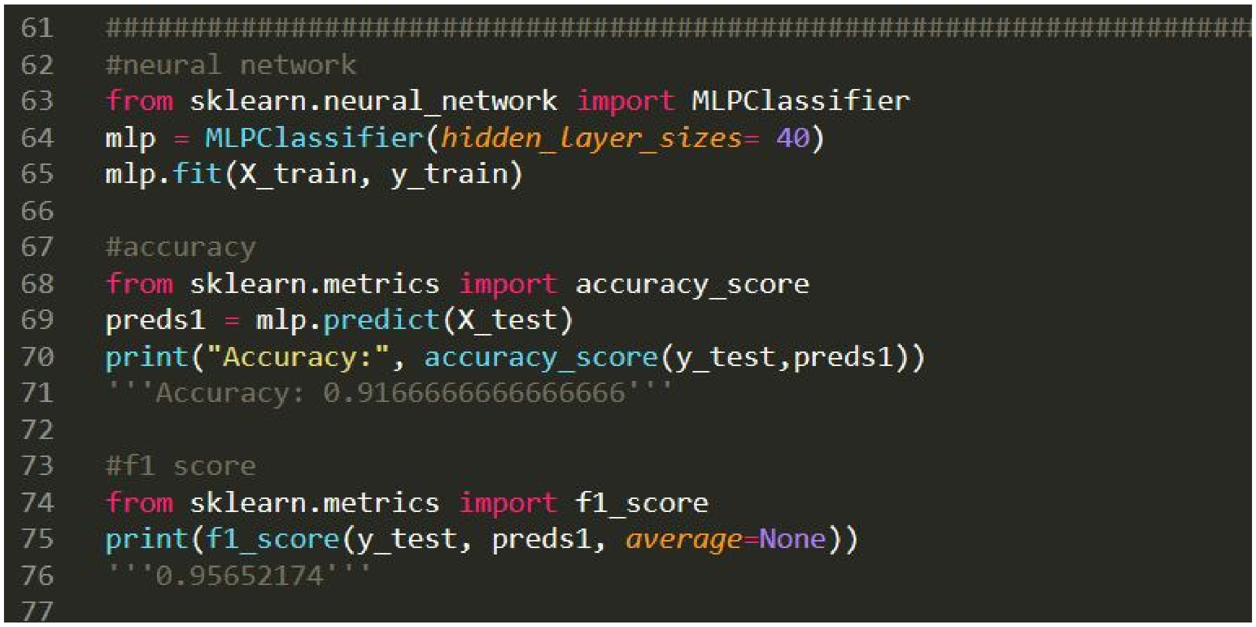
Step 5: applying the models to our dataset

- Random forest



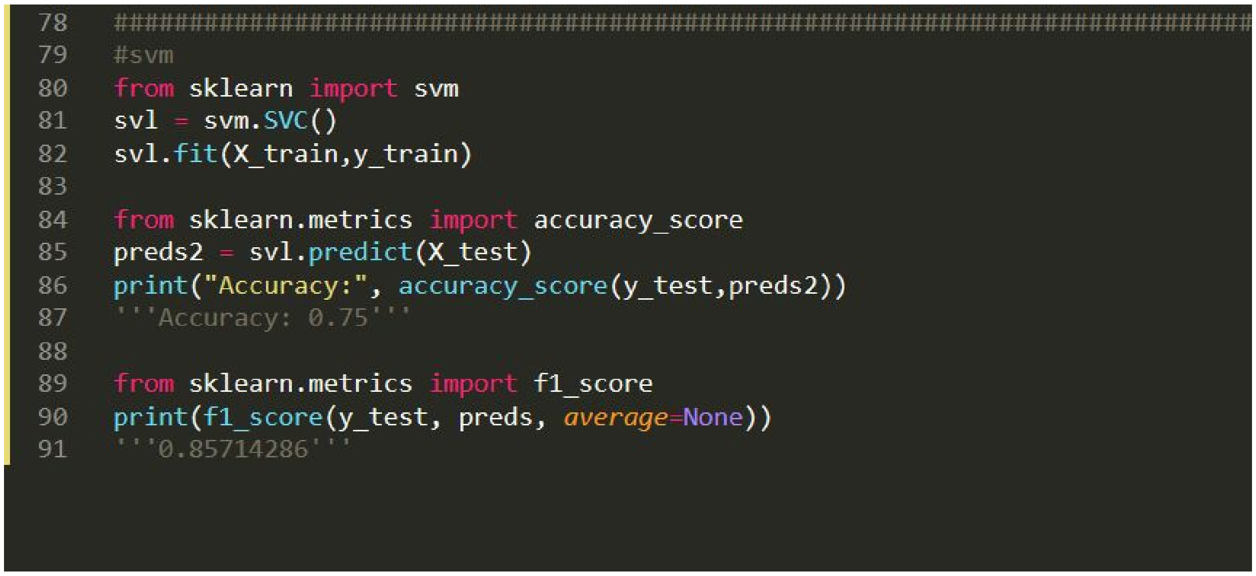
```
##########################################################################
#random forest
from sklearn.ensemble import RandomForestClassifier
clf = RandomForestClassifier(n_estimators=25, random_state=0)
clf.fit(X_train, y_train)from sklearn.metrics import accuracy_score

preds = clf.predict(X_test)
print("Accuracy:", accuracy_score(y_test,preds))
'''Accuracy: 0.9791666666666666'''

from sklearn.metrics import f1_score
print(f1_score(y_test, preds, average=None))
'''0.98947368'''
```

- Neural Network



```
##########################################################################
#neural network
from sklearn.neural_network import MLPClassifier
mlp = MLPClassifier(hidden_layer_sizes= 40)
mlp.fit(X_train, y_train)

#accuracy
from sklearn.metrics import accuracy_score
preds1 = mlp.predict(X_test)
print("Accuracy:", accuracy_score(y_test,preds1))
'''Accuracy: 0.9166666666666666'''

#f1 score
from sklearn.metrics import f1_score
print(f1_score(y_test, preds1, average=None))
'''0.95652174'''
```

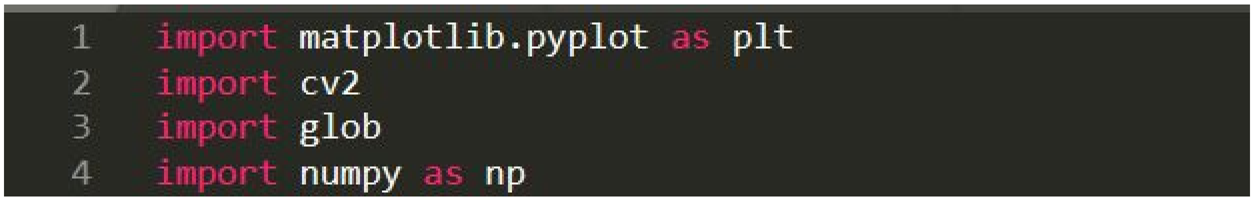
- Support vector machine



```
############################################################################
#svm
from sklearn import svm
svl = svm.SVC()
svl.fit(X_train,y_train)

from sklearn.metrics import accuracy_score
preds2 = svl.predict(X_test)
print("Accuracy:", accuracy_score(y_test,preds2))
'''Accuracy: 0.75'''

from sklearn.metrics import f1_score
print(f1_score(y_test, preds, average=None))
'''0.85714286'''
```

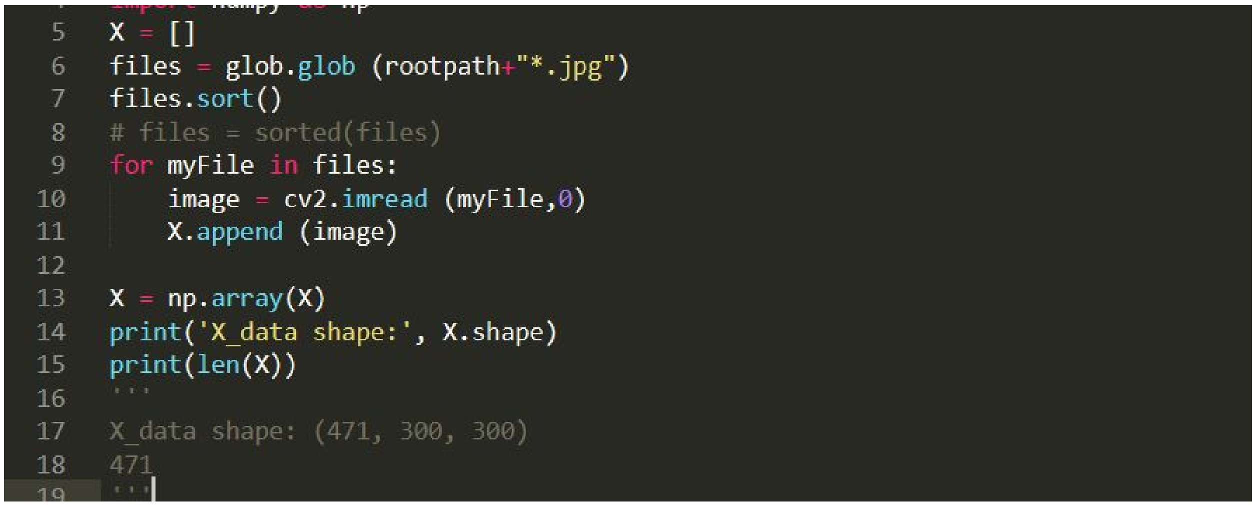
With PCA

Step 1: importing the modules



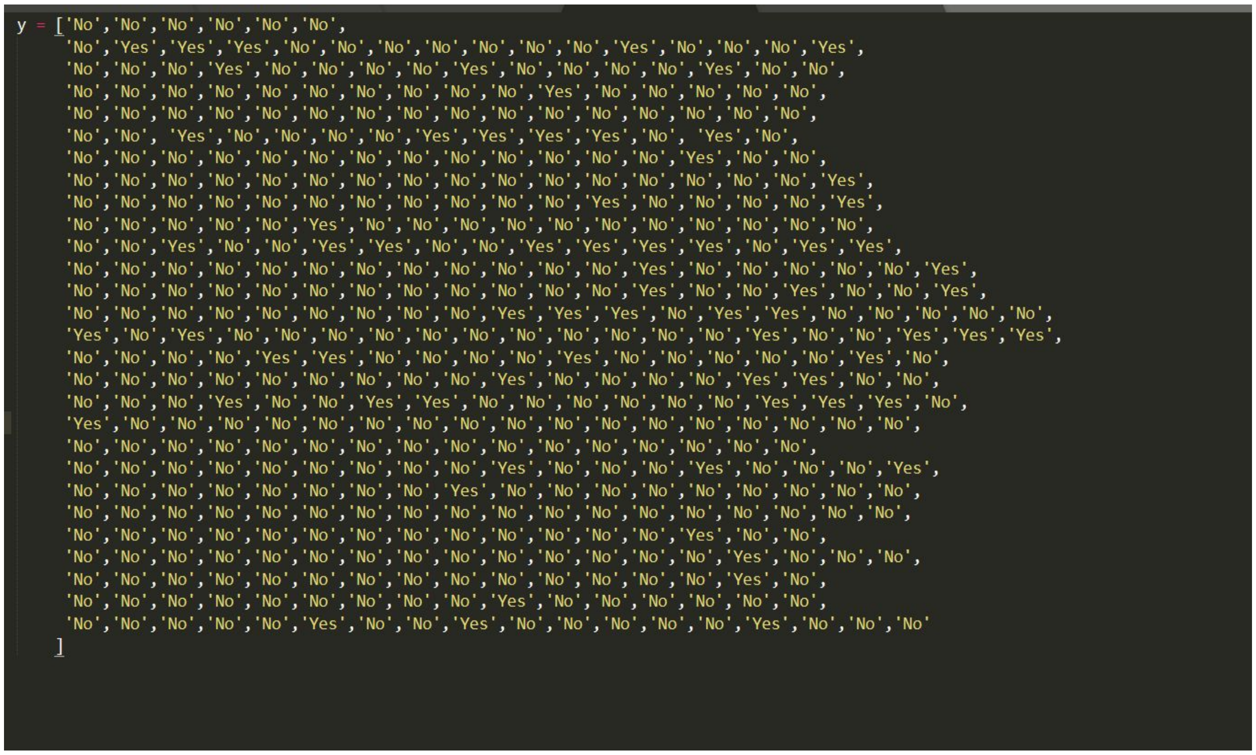
```
import matplotlib.pyplot as plt
import cv2
import glob
import numpy as np
```

Step 2: Getting the generated dataset (patches of 300x300 size)



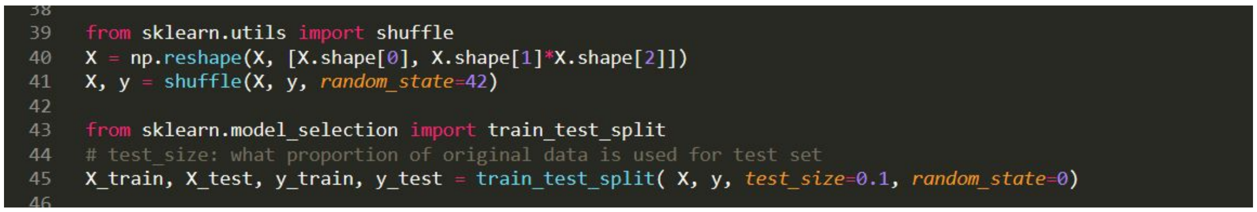
```
X = []
files = glob.glob (rootpath+"*.jpg")
files.sort()
# files = sorted(files)
for myFile in files:
    image = cv2.imread (myFile,0)
    X.append (image)

X = np.array(X)
print('X_data shape:', X.shape)
print(len(X))
'''
X_data shape: (471, 300, 300)
471
'''
```

Step 3: labelling the dataset



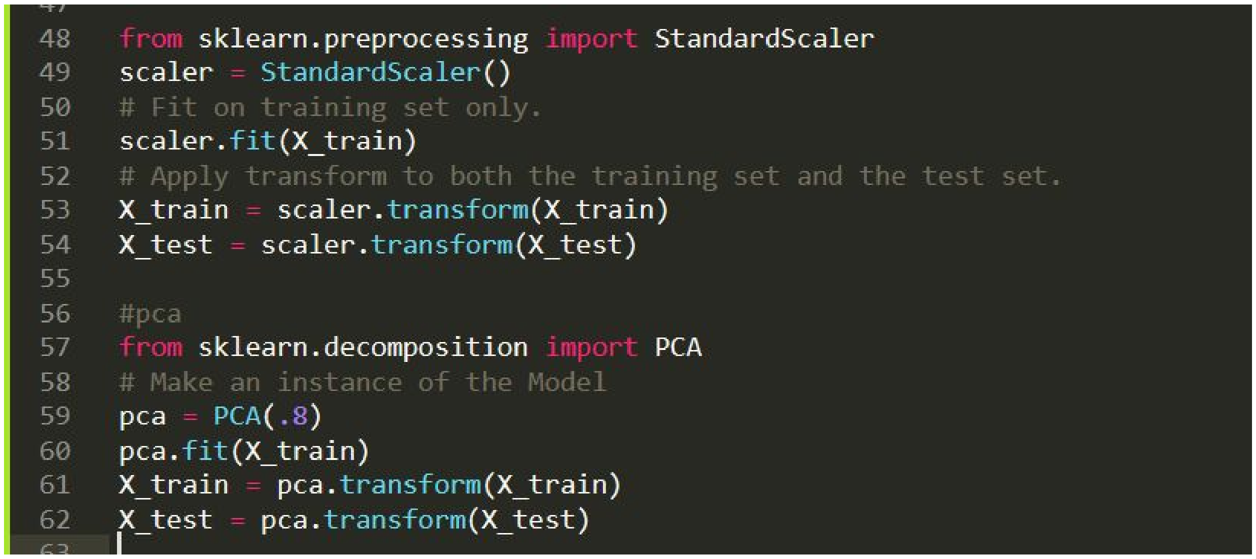
```
y = ['No','No','No','No','No','No',
     'No','Yes','Yes','Yes','No','No','No','No','No','No','No','Yes','No','No','No','Yes',
     'No','No','No','Yes','No','No','No','No','Yes','No','No','No','No','Yes','No','No',
     'No','No','No','No','No','No','No','No','No','No','Yes','No','No','No','No','No',
     'No','No','No','No','No','No','No','No','No','No','No','No','No','No','No','No',
     'No','No', 'Yes','No','No','No','No','Yes','Yes','Yes','Yes','No', 'Yes','No',
     'No','No','No','No','No','No','No','No','No','No','No','No','No','Yes','No','No',
     'No','No','No','No','No','No','No','No','No','No','No','No','No','No','No','No','Yes',
     'No','No','No','No','No','No','No','No','No','No','No','Yes','No','No','No','No','Yes',
     'No','No','No','No','No','Yes','No','No','No','No','No','No','No','No','No','No','No',
     'No','No','Yes','No','No','Yes','Yes','No','No','Yes','Yes','Yes','Yes','No','Yes','Yes',
     'No','No','No','No','No','No','No','No','No','No','No','No','Yes','No','No','No','No','No','Yes',
     'No','No','No','No','No','No','No','No','No','No','No','No','Yes','No','No','Yes','No','No','Yes',
     'No','No','No','No','No','No','No','No','No','Yes','Yes','Yes','No','Yes','Yes','No','No','No','No','No',
     'Yes','No','Yes','No','No','No','No','No','No','No','No','No','No','No','Yes','No','No','Yes','Yes','Yes',
     'No','No','No','No','Yes','Yes','No','No','No','No','Yes','No','No','No','No','No','Yes','No',
     'No','No','No','No','No','No','No','No','No','Yes','No','No','No','No','Yes','Yes','No','No',
     'No','No','No','Yes','No','No','Yes','Yes','No','No','No','No','No','No','Yes','Yes','Yes','No',
     'Yes','No','No','No','No','No','No','No','No','No','No','No','No','No','No','No','No','No',
     'No','No','No','No','No','No','No','No','No','No','No','No','No','No','No','No',
     'No','No','No','No','No','No','No','No','No','Yes','No','No','No','Yes','No','No','No','Yes',
     'No','No','No','No','No','No','No','No','Yes','No','No','No','No','No','No','No','No','No',
     'No','No','No','No','No','No','No','No','No','No','No','No','No','No','No','No','No','No',
     'No','No','No','No','No','No','No','No','No','No','No','No','No','Yes','No','No',
     'No','No','No','No','No','No','No','No','No','No','No','No','No','No','Yes','No','No','No',
     'No','No','No','No','No','No','No','No','No','No','No','No','No','No','Yes','No',
     'No','No','No','No','No','No','No','No','No','Yes','No','No','No','No','No','No',
     'No','No','No','No','No','Yes','No','No','Yes','No','No','No','No','No','Yes','No','No','No'
    ]
```

Step 4: Splitting the dataset into training and testing of the model



```
from sklearn.utils import shuffle
X = np.reshape(X, [X.shape[0], X.shape[1]*X.shape[2]])
X, y = shuffle(X, y, random_state=42)

from sklearn.model_selection import train_test_split
# test_size: what proportion of original data is used for test set
X_train, X_test, y_train, y_test = train_test_split( X, y, test_size=0.1, random_state=0)
```

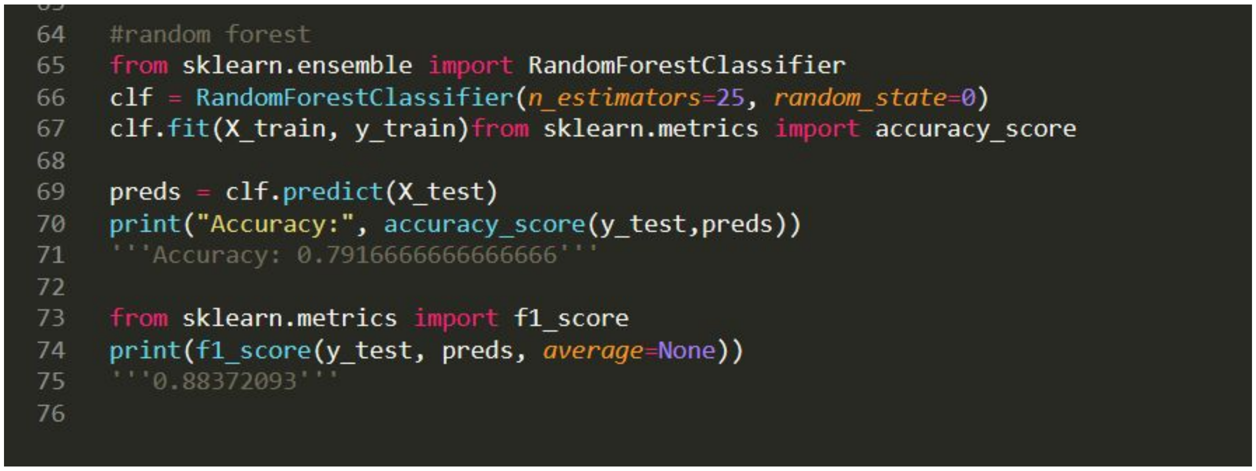
Step 5: Applying PCA



```
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
# Fit on training set only.
scaler.fit(X_train)
# Apply transform to both the training set and the test set.
X_train = scaler.transform(X_train)
X_test = scaler.transform(X_test)

#pca
from sklearn.decomposition import PCA
# Make an instance of the Model
pca = PCA(.8)
pca.fit(X_train)
X_train = pca.transform(X_train)
X_test = pca.transform(X_test)
```

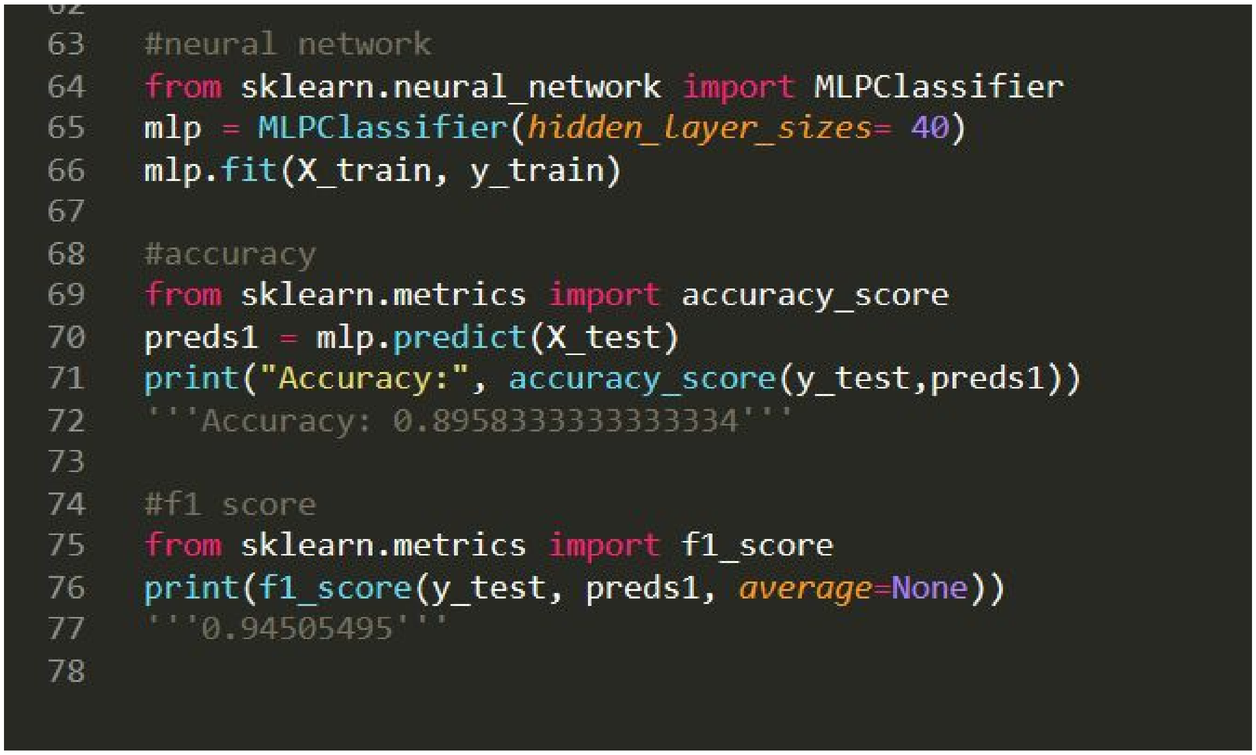
Step 5: applying the models to our dataset after applying PCA

- Random forest



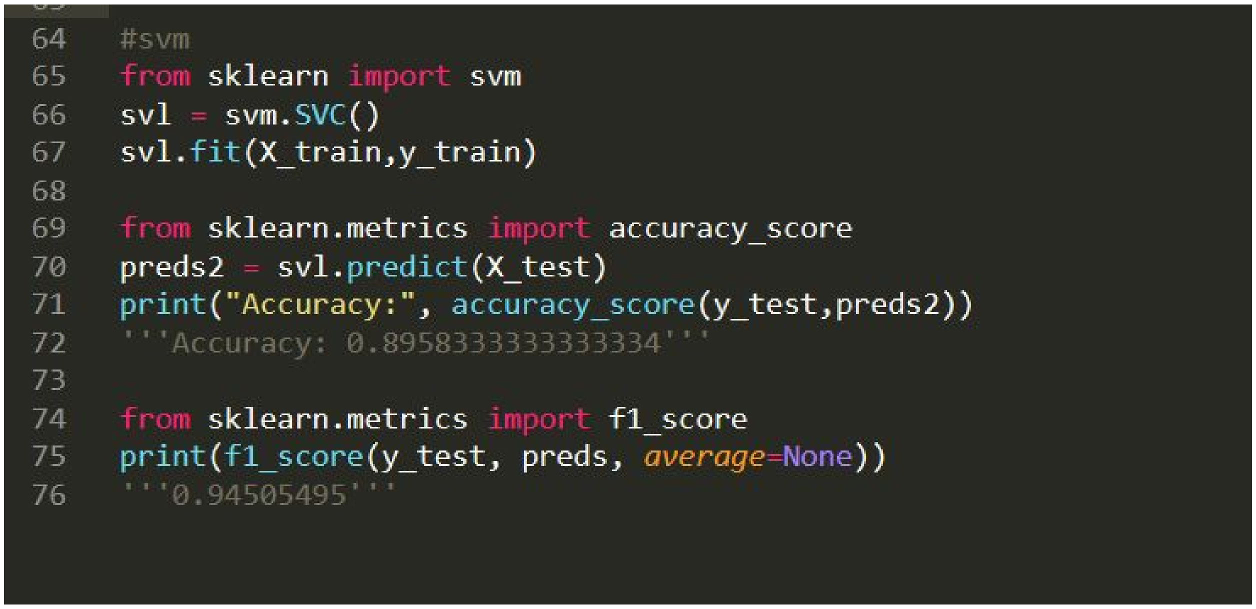
```
#random forest
from sklearn.ensemble import RandomForestClassifier
clf = RandomForestClassifier(n_estimators=25, random_state=0)
clf.fit(X_train, y_train)from sklearn.metrics import accuracy_score

preds = clf.predict(X_test)
print("Accuracy:", accuracy_score(y_test,preds))
'''Accuracy: 0.7916666666666666'''

from sklearn.metrics import f1_score
print(f1_score(y_test, preds, average=None))
'''0.88372093'''

```

- Neural Network



```
#neural network
from sklearn.neural_network import MLPClassifier
mlp = MLPClassifier(hidden_layer_sizes= 40)
mlp.fit(X_train, y_train)

#accuracy
from sklearn.metrics import accuracy_score
preds1 = mlp.predict(X_test)
print("Accuracy:", accuracy_score(y_test,preds1))
'''Accuracy: 0.8958333333333334'''

#f1 score
from sklearn.metrics import f1_score
print(f1_score(y_test, preds1, average=None))
'''0.94505495'''

```

- Support vector machine



```
#svm
from sklearn import svm
svl = svm.SVC()
svl.fit(X_train,y_train)

from sklearn.metrics import accuracy_score
preds2 = svl.predict(X_test)
print("Accuracy:", accuracy_score(y_test,preds2))
'''Accuracy: 0.8958333333333334'''

from sklearn.metrics import f1_score
print(f1_score(y_test, preds, average=None))
'''0.94505495'''
```

References

1. Haloi, M.; Dandapat, S.; Sinha, R. A Gaussian scale space approach for exudates detection, classification and severity prediction. *arXiv*, 2015; arXiv:1505.00737.
2. Haloi, M. Improved microaneurysm detection using deep neural networks. *arXiv*, 2015; arXiv:1505.04424.
3. van Grinsven, M.J.; van Ginneken, B.; Hoyng, C.B.; Theelen, T.; Sánchez, C.I. Fast convolutional neural network training using selective data sampling: Application to hemorrhage detection in color fundus images. *IEEE Trans. Med. Imaging* 2016, *35*, 1273–1284.
4. Srivastava, R.; Duan, L.; Wong, D.W.; Liu, J.; Wong, T.Y. Detecting retinal microaneurysms and hemorrhages with robustness to the presence of blood vessels. *Comput. Methods Programs Biomed.* 2017, *138*, 83–91.
5. Seoud, L.; Chelbi, J.; Cheriet, F. Automatic grading of diabetic retinopathy on a public database. In Proceedings of the Ophthalmic Medical Image Analysis Second International Workshop, Munich, Germany, 9 October 2015.
6. Barandiaran, I. The random subspace method for constructing decision forests. *IEEE Trans. Pattern Anal. Mach. Intell.* 1998, *20*, 832–844.
7. Gu, J.; Wang, Z.; Kuen, J.; Ma, L.; Shahroudy, A.; Shuai, B.; Liu, T.; Wang, X.; Wang, L.; Wang, G. Recent advances in convolutional neural networks. *arXiv*, 2015; arXiv:1512.07108.
8. Gulshan, V.; Peng, L.; Coram, M.; Stumpe, M.C.; Wu, D.; Narayanaswamy, A.; Venugopalan, S.; Widner, K.; Madams, T.; Cuadros, J. Development and validation of a deep learning algorithm for detection of diabetic retinopathy in retinal fundus photographs. *JAMA* **2016**, *316*, 2402–2410.
9. Sankar, M.; Batri, K.; Parvathi, R. Earliest diabetic retinopathy classification using deep convolution neural networks. *Int. J. Adv. Eng. Technol.* **2016**.
10. Pratt, H.; Coenen, F.; Broadbent, D.M.; Harding, S.P.; Zheng, Y. Convolutional neural networks for diabetic retinopathy. *Procedia Comput. Sci.* **2016**, *90*, 200–205.
11. Haneda, S.; Yamashita, H. International clinical diabetic retinopathy disease severity scale. *Nihon Rinsho. Jpn. J. Clin. Med.* **2010**, *68*, 228–235.

12. J. Shan and L. Li, “A deep learning method for microaneurysm detection in fundus images,” in Proc. IEEE 1st Int. Conf. Connected Health, Appl., Syst. Eng. Technol. (CHASE), Jun. 2016, pp. 357–358.

13. H. Lee, R. Grosse, R. Ranganath, and A. Y. Ng, “Convolutional deep belief networks for scalable unsupervised learning of hierarchical representations,” in Proc. 26th Annu. Int. Conf. Mach. Learn., 2009, pp. 609–616

14. Q. V. Le, W. Y. Zou, S. Y. Yeung, and A. Y. Ng, “Learning hierarchical invariant spatio-temporal features for action recognition with independent subspace analysis,” in Proc. IEEE Conf. Comput. Vis. Pattern Recognit., Jun. 2011, pp. 3361–3368.

15. G. Quellec, M. Lamard, P. M. Josselin, G. Cazuguel, B. Cochener, and C. Roux, “Optimal wavelet transform for the detection of microaneurysms in retina photographs,” IEEE Trans. Med. Imag., vol. 27, no. 9, pp. 1230–1241, Sep. 2008.

16. M. Niemeijer et al., “Retinopathy online challenge: Automatic detection of microaneurysms in digital color fundus photographs,” IEEE Trans. Med. Imag., vol. 29, no. 1, pp. 185–195, Jan. 2010.

17. K. Ram, G. D. Joshi, and J. Sivaswamy, “A successive clutter-rejection based approach for early detection of diabetic retinopathy,” IEEE Trans. Biomed. Eng., vol. 58, no. 3, pp. 664–673, Mar. 2011.

18. M. U. Akram, S. Khalid, A. Tariq, S. A. Khan, and F. Azam, “Detection and classification of retinal lesions for grading of diabetic retinopathy,” Comput. Biol. Med., vol. 45, no. 1, pp. 161–171, 2014.

19. Yang, Y.; Li, T.; Li, W.; Wu, H.; Fan, W.; Zhang, W. Lesion detection and grading of diabetic retinopathy via two-stages deep convolutional neural networks. In Proceedings of the International Conference on Medical Image Computing and Computer-Assisted Intervention, Honolulu, HI, USA, 21–26 July 2017; pp. 533–540.

20. Gurudath, N.; Celenk, M.; Riley, H.B. Machine learning identification of diabetic retinopathy from fundus images. In Proceedings of the 2014 IEEE Signal Processing in Medicine and Biology Symposium (SPMB), Philadelphia, PA, USA, 13 December 2014; pp. 1–7

21. Cao, W.; Shan, J.; Czarnek, N.; Li, L. Microaneurysm detection in fundus images using small image patches and machine learning methods. In

Proceedings of the IEEE International Conference on Bioinformatics and Biomedicine (BIBM), Kansas City, MO, USA, 13–16 November 2017; pp. 325–331

22. Nanni, L.; Ghidoni, S.; Brahnam, S. Handcrafted vs. non-handcrafted features for computer vision classification. Pattern Recognit. 2017, 71, 158–172