Machine Learning

Coursework 1

The Diabetic Retinopathy Debrecen Data Set

Süleyman Kalkan
University of Bradford
UBN: 21042430

05/04/2022

Introduction

Diabetic retinopathy is a common and potentially disabling long-term complication of diabetes. Retinopathy can lead to glaucoma and increased pressure within the eye that can further threaten vision. Untreated, retinopathy can lead to progressive and irreversible vision loss. This condition is the leading cause of blindness in people between the ages of 20 and 60. (Webmd, 2021)

In this study, the main aim is to design and develop a machine learning model that can detect if a patient has signs of Diabetic Retinopathy or not. So that this machine learning model can help professionals to detect Diabetic Retinopathy early as possible.

Early detection of retinopathy increases the chances of treatment being effective and stopping it from getting worse. To minimise the risk of causing blindness or vision loss, people with diabetes should attend annual diabetic eye screening appointments. (NHS, 2021)

Background

Diabetic retinopathy affects up to 80% of those who have had diabetes for 20 years or more. At least 90% of new cases could be reduced with proper treatment and monitoring of the eyes. The longer a person has diabetes, the higher his or her chances of developing diabetic retinopathy. Each year in the United States, diabetic retinopathy accounts for 12% of all new cases of blindness. It is also the leading cause of blindness in people aged 20 to 64.

Diabetic retinopathy is diagnosed entirely by recognizing abnormalities on retinal images taken by fundoscopy.

Because fundoscopic images are the main sources for diagnosis of diabetic retinopathy, manually analyzing those images can be time-consuming and unreliable, as the ability to detect abnormalities varies by years of experience. Therefore, scientists have explored developing computer-aided diagnosis approaches to automating the process, which involves extracting information about the blood vessels and any abnormal patterns from the rest of the fundoscopic image and analyzing them. (Wikipedia, 2022)

Some technologies in the same area of this study: EyeArt (Eyenuk), RetinaLyze (RetinaLyze System A/S) and Retmarker (Retmarker) use artificial intelligence (AI) technology to analyse retinal images to help diagnose diabetic retinopathy. The aim is to speed up and improve diagnosis. They could be used in national screening programmes and settings with limited expertise.

Methodology and Data

One of the crucial parts of developing strong machine learning models is deciding on a machine learning algorithm that performs with high accuracy. The main focus of this work is developing and training a machine learning model that can decide if a patient has signs of Diabetic Retinopathy or not, and as we check through the dataset we can see that column 19 is giving us this information. We need to aim for choosing a machine learning algorithm that gives good performance on binary classification. Our dataset is pre-labelled so it is a supervised learning problem. But first, we need to prepare our data to perform better in training.

Pre-processing the Dataset

As we look through the dataset we can see some values of columns varies from 0 to above 100 with no specific range that we can see. Many machine learning algorithms perform better when numerical input variables are scaled to a standard range. Normalization scales each input variable separately to the range 0-1, which is the range for floating-point values where we have the most precision. So first, we are going to normalize the values from columns 3 to 19 (excluding the 1,2,20 and 21 which are binary expressions). (Machinelearningmastery, 2020

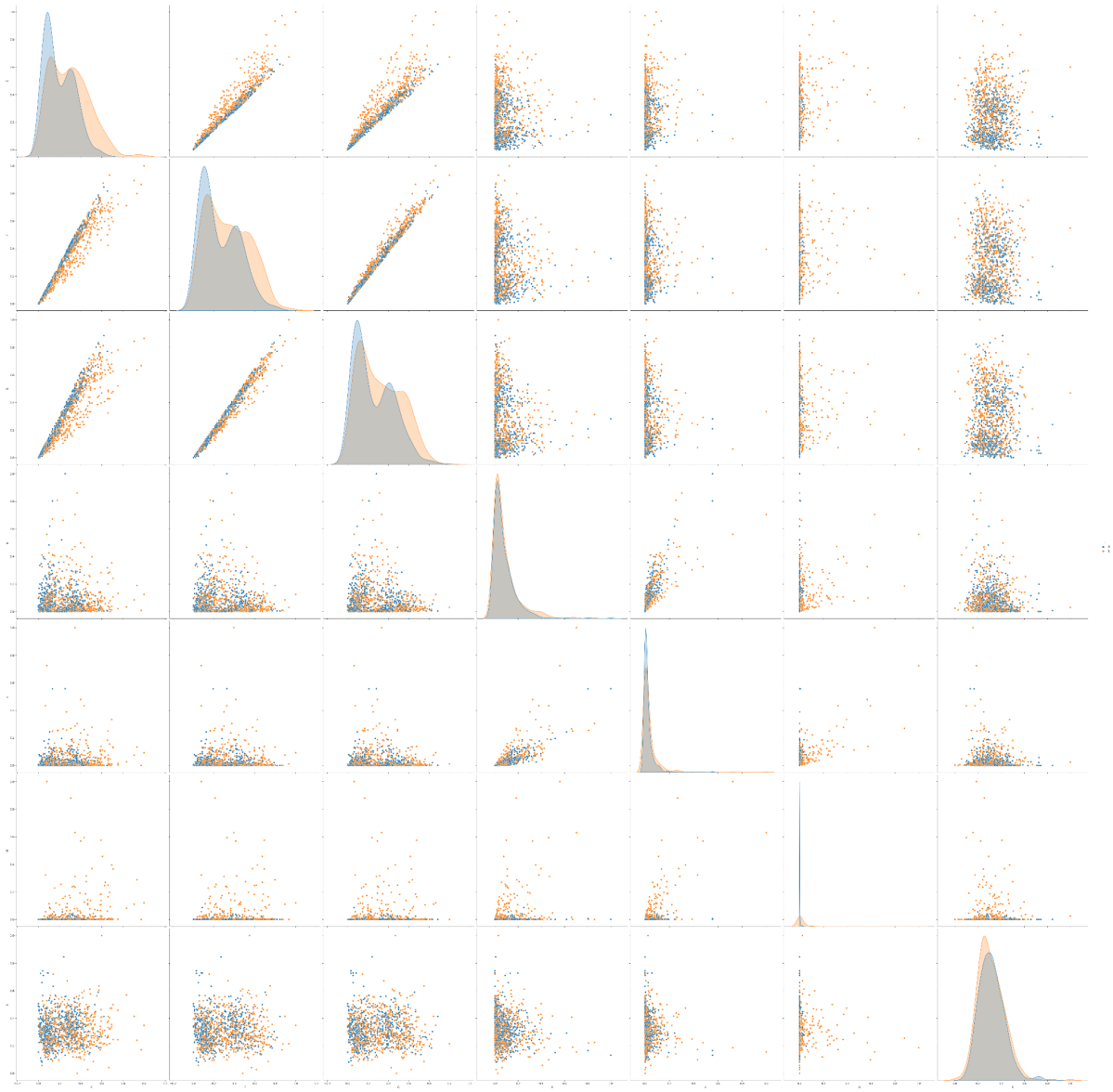
Normalization requires that you know or are able to accurately estimate the minimum and maximum observable values. We can extract these values from our data with sklearn MinMaxScaler.

```
# Import dataset
dataset = pandas.read_csv("diabeticret_dataset.csv")



# Preprocessing the dataset: Normalizing Values
normalizer = MinMaxScaler()
dataset[['C', 'D', 'E', 'F', 'G', 'H', 'I', 'J', 'K', 'L', 'M', 'N', 'O',
'P', 'Q', 'R']] = normalizer.fit_transform(dataset[['C', 'D', 'E',  'F', 'G',
'H', 'I', 'J', 'K', 'L', 'M', 'N', 'O', 'P', 'Q', 'R']])
```

To figure out which algorithm to choose, we can run pairplot.

```
# Pair plot
sb.pairplot(dataset[['C', 'F', 'G', 'K', 'L', 'O', 'R', 'T']], hue='T',
dropna=True, height=8)
plt.show()
```

Output:



We can see from the plot that there is a lot of overlap between data points. We can use the K Nearest Neighbour algorithm since its based on the principle of Euclidean distance. We can use Decision Tree as well in case of KNN doesn't perform to our expectations.

We can also think about using linear classifiers such as Logistic Regression or Linear Support Vector Machine.

Trying all these algorithms and comparing their results would be the best approach at this point.

Final implementation:

```
import time

import pandas
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.preprocessing import MinMaxScaler

# Import dataset
from sklearn.svm import SVC
from sklearn.tree import DecisionTreeClassifier

dataset = pandas.read_csv("diabeticret_dataset.csv")

# Preprocessing the dataset: Normalizing Values
normalizer = MinMaxScaler()
dataset[['C', 'D', 'E', 'F', 'G', 'H', 'I', 'J', 'K', 'L', 'M', 'N', 'O',
'P', 'Q', 'R']] = normalizer.fit_transform(dataset[['C', 'D', 'E',  'F', 'G',
'H', 'I', 'J', 'K', 'L', 'M', 'N', 'O', 'P', 'Q', 'R']])
X = dataset.drop(columns=['T'])
Y = dataset['T']

# Split test data from dataset
X, X_test, Y, Y_test = train_test_split(X, Y, test_size= 20/100)

# @exec_model: Execute and print out the results of the given model
def exec_model(model, modelname):
```

```
    start = time.time()
    model.fit(X, Y);
    accuracy = round(accuracy_score(Y_test, model.predict(X_test))* 100, 2)
    stop = time.time()
    training_time = round((stop - start), 4)
    return accuracy, training_time

# returns the average of a list
def Average(lst):
    return sum(lst) / len(lst)

# List of algorithms to test
MODELS = {
    "Decision Tree": DecisionTreeClassifier(),
    "KNN": KNeighborsClassifier(),
    "Logistic Regression": LogisticRegression(),
    "SVC" : SVC(kernel='linear')
}

# Run the algorithms for 500 times and print the averages
for modelname, model in MODELS.items():
    accuracy_list = []
    training_time_list = []
    for index in range(500):
        accuracy , training_time = exec_model(model, modelname)
        accuracy_list.append(accuracy)
        training_time_list.append(training_time)

    accuracy_average = Average(accuracy_list)
    time_average = Average(training_time_list)

    print(f"#####-{modelname}-#####")
    print("Average Accuracy: ", round(accuracy_average, 2))
    print(f"Average Time:", round(time_average, 4), "s")
    print("---------")
```

Github repository; https://github.com/suleymankalkan/ml-coursework1

Analysis and Discussions

After executing the implementation above, here is the result of our analysis;

PS: Each algorithm is executed 500 times.

```
#####-Decision Tree-#####
Average Accuracy:  67.67
Average Time: 0.0089 s
---------
#####-KNN-#####
Average Accuracy:  60.43
Average Time: 0.0206 s
---------
#####-Logistic Regression-#####
Average Accuracy:  70.43
Average Time: 0.022 s
---------
#####-SVC-#####
Average Accuracy:  71.3
Average Time: 0.0197 s
---------

Process finished with exit code 0
```

Decision tree is the fastest and has impressive accuracy based on its training speed. Decision tree can be a go-to option for large datasets.

KNN didn't really meet the expectations with its lowest accuracy and similar time range.

For small to medium size datasets, SVC can be used as we see how accurate it is.

Conclusions and suggestions for future work

As a result of the implementation, the Decision Tree Algorithm is the best for accuracy/training time value. It can train big datasets that may contain millions of records with a rapid speed and good accuracy.

For future work, I would like to go more deeply into the data, especially for the pre-proccessing phase where can be done more research and comparison of different pre-processing models. I think that if we make some feature extractions on the dataset, we get better results as well. Also, when choosing algorithms there can be more than one way to decide on algorithms but requires more deep research.

Bibliography and Citations

1. https://www.webmd.com/diabetes/closer-look-diabetic-retinopathy
2. https://medium.com/thinkport/top-10-binary-classification-algorithms-a-beginners-guide-feeacbd7a3e2
3. https://machinelearningmastery.com/standardscaler-and-minmaxscaler-transforms-in-python/
4. https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.MinMaxScaler.html
5. An ensemble-based system for automatic screening of diabetic Retinopathy by Bálint Antal and András Hajdu
6. https://www.nhs.uk/conditions/diabetic-retinopathy/
7. https://www.nei.nih.gov/learn-about-eye-health/eye-conditions-and-diseases/diabetic-retinopathy
8. https://en.wikipedia.org/wiki/Diabetic_retinopathy
9. https://www.nice.org.uk/advice/mib265/resources/ai-technologies-for-detecting-diabetic-retinopathy-pdf-2285965755558853