A MACHINE LEARNING APPROACH FOR DIABETIC RETINOPATHY

TABLE OF CONTENTS

Contents

1. Introduction

According to Wikipedia[1], **Diabetic retinopathy**, also known as **diabetic eye disease** (**DED**), is a medical condition in which damage occurs to the retina due to diabetes mellitus. It is a leading cause of blindness in developed countries. So, predicting the probability of it happening and taking actions beforehand based on the level of intensity (if possible) is the key to preventing it. Those with higher risk of DED, are to be taken into special care for possible treatment.

In the meantime, AI with its applications in fields such as medical diagnoses and health care, has not been actionless. Even though Artificial Intelligence and its applications are not perfect and totally reliable, they play a big role in many important decisions in health care mostly as an assistant beside a human expert. So, having an acceptable and high accuracy AI system can be very promising. With that being said, an example of how machine learning can help diagnose **Diabetic retinopathy** is presented in this paper.

The dataset in use for this paper is **Diabetic Retinopathy Debrecen Data Set**[2] from UCI[3]**.** This dataset contains features extracted from the Messidor image set to predict whether an image contains signs of diabetic retinopathy or not. All features represent either a detected lesion, a descriptive feature of an anatomical part or an image-level descriptor.

2. Data Analysis

2.1. Data Definition

The dataset **Diabetic Retinopathy Debrecen Data Set** from **UCI** contains 1151 instances that represent the features extracted from each image in Messidor image set. Each instance has consists of 20 features with column names of id,0,1,2,..,17,18,Class. These names show the following features:

id) Patient's id number.

0) The binary result of quality assessment. 0 = bad quality, 1 = sufficient quality.
1) The binary result of pre-screening, where 1 indicates severe retinal abnormality and 0 means its lack.
2-7) The results of MA detection. Each feature value stand for the number of MAs found at the confidence levels alpha = 0.5, . . . , 1, respectively.
8-15) contain the same information as 2-7) for exudates. However, as exudates are represented by a set of points rather than the number of pixels constructing the lesions, these features are normalized by

[1] https://en.wikipedia.org/wiki/Diabetic_retinopathy
[2] https://archive.ics.uci.edu/ml/datasets/Diabetic+Retinopathy+Debrecen+Data+Set
[3] https://archive.ics.uci.edu/ml/index.php

dividing the number of lesions with the diameter of the ROI to compensate different image sizes.
16) The Euclidean distance of the center of the macula and the center of the optic disc to provide important information regarding the patient's condition. This feature is also normalized with the diameter of the ROI.
17) The diameter of the optic disc.
18) The binary result of the AM/FM-based classification.
Class) Class label. 1 = contains signs of **Diabetic Retinopathy** (Accumulative label for the Messidor classes 1, 2, 3), 0 = no signs of DR.

The programming language used in this paper is *Python* and for the purpose of analyzing data, *Pandas*, a *Python* library specialized for various data processes, is used.

First, the dataset is loaded as a ***DataFrame*** instance of Pandas module for further analysis (figure 1).

```
1 import pandas as pd
2 data = pd.read_csv('csv_result-messidor_features.csv')
```

Figure 1. Load the dataset

To avoid complication, the index of values are set to the 'id' column. This will remove 'id' column from the features (figure 2).

```
1 data.set_index('id', inplace=True)
```

Figure 2. Set index

2.2. Exploratory Data Analysis

In this section of the paper, various methods for analysis will be applied to the dataset to derive insights from the data. These methods are checking the data types, checking for missing values (null), plotting frequency histograms of features and label, count plots for some feature to check how they are distributed between each class, and checking the median of each class.

In figure 3, the code and its output which tells the data types and missing values in features is shown.

```
1 data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1151 entries, 0 to 1150
Data columns (total 21 columns):
 #   Column  Non-Null Count  Dtype
---  ------  --------------  -----
 0   id      1151 non-null   int64
 1   0       1151 non-null   int64
 2   1       1151 non-null   int64
 3   2       1151 non-null   int64
 4   3       1151 non-null   int64
 5   4       1151 non-null   int64
 6   5       1151 non-null   int64
 7   6       1151 non-null   int64
 8   7       1151 non-null   int64
 9   8       1151 non-null   float64
 10  9       1151 non-null   float64
 11  10      1151 non-null   float64
 12  11      1151 non-null   float64
 13  12      1151 non-null   float64
 14  13      1151 non-null   float64
 15  14      1151 non-null   float64
 16  15      1151 non-null   float64
 17  16      1151 non-null   float64
 18  17      1151 non-null   float64
 19  18      1151 non-null   int64
 20  Class   1151 non-null   int64
dtypes: float64(10), int64(11)
memory usage: 189.0 KB
```

Figure 3. Data types

As it's shown in figure 3, there is no missing value in any column. Also, all the features are numerical values both integer and float.

Frequency histograms of features and label will indicate the distributions of them and based on it, the value introduced by each feature can be estimated. For instance, when a feature is distributed only on a fixed number, it means that the existence of the feature may not have much value for the algorithm. But all of these depend on what the feature is indicating.

Histograms of all the columns of dataset is shown in figure 4.

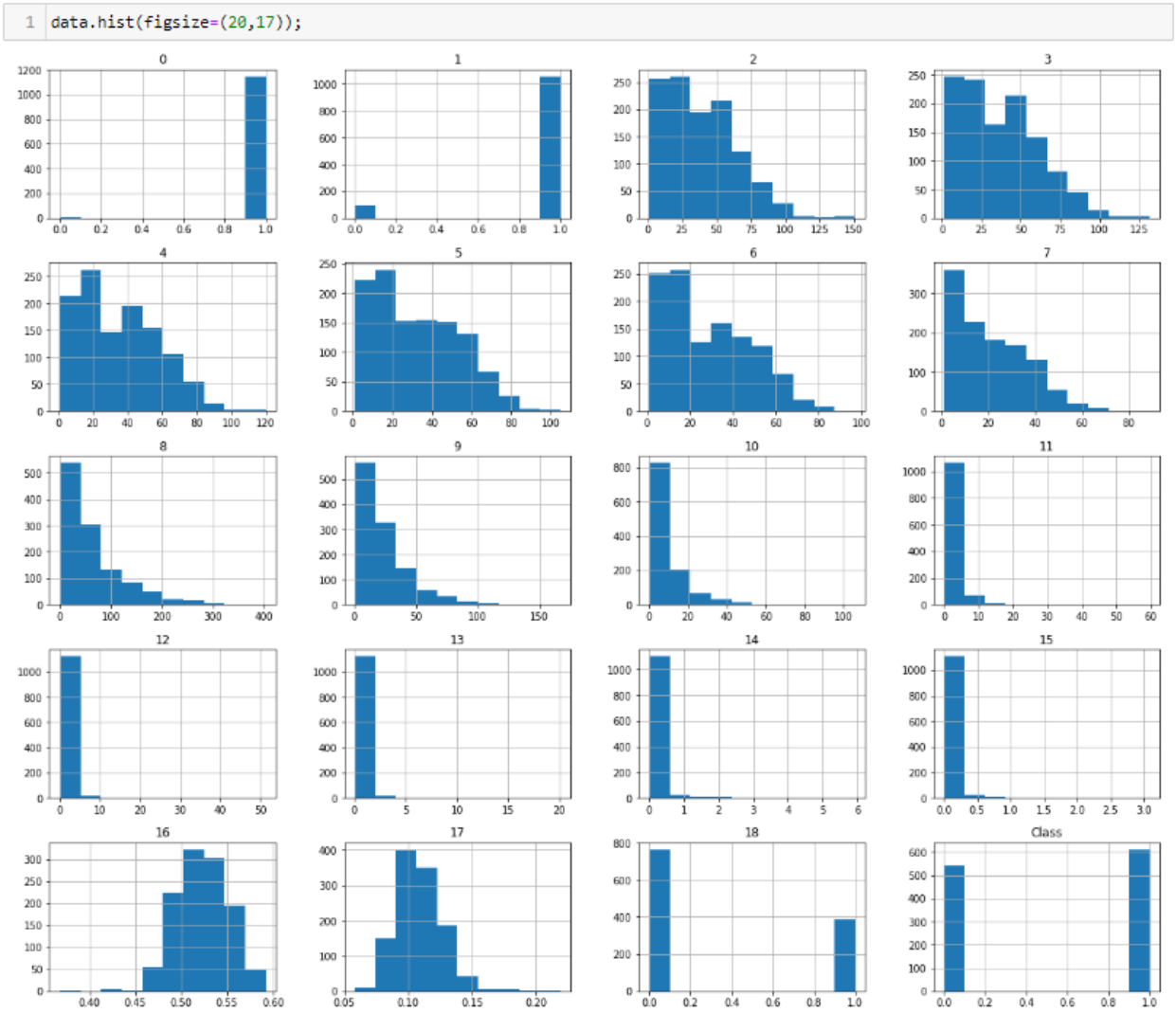
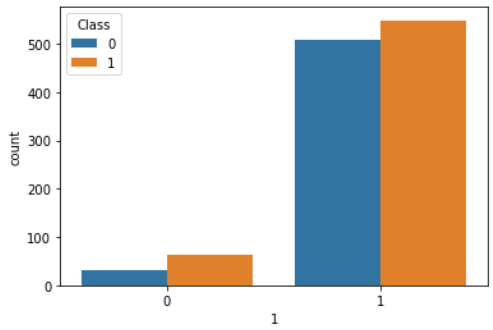


Figure 4. Frequency histograms

Features 2, 3, 4, 5, 6, 7, 8, 9, 10 are all skewed right distributions which means that the lower numbers are a lot more common and frequent. Meanwhile feature 0, which indicates the quality of the images, has a very skewed distribution in a way that almost no other value than 1 is there. This could mean feature 0 is not necessary.

Here features ‘1’ and ‘18’ are used to plot a count plot so in case they are not very predictive, they will be dropped. This is shown in figure 5.

```
sns.countplot(x='1',hue='Class',data=data);
```




```
sns.countplot(x='18',hue='Class',data=data);
```

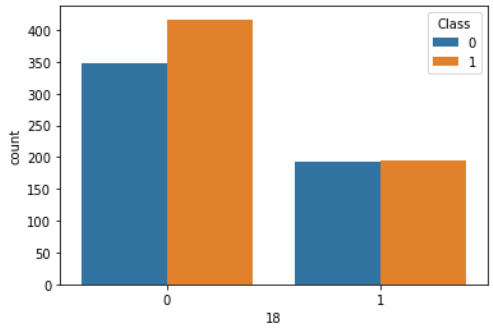


Figure 5. Count plots

The count plots show that for each class, the predictivity of features shown, is almost 50%. This means they are in a sense randomly influencing the output. So, they are not valuable features and will be dropped.

Finally, by checking the median of features in each class, and calculating their differences, the values of features for each class can be compared. In case there is low difference, it means that there is not much difference between their values, therefore can not be used for in the algorithm (Figure 6).

```
keys = [data.columns[i] for i in range(1, 17)]
values = [(data.groupby(['Class'])[data.columns[i]].median()[1] - data.groupby(['Class'])[data.columns[i]].median()[0])\
          / (data.groupby(['Class'])[data.columns[i]].median()[0] + 0.001) * 100  for i in range(1,17)]
median_diff = dict(zip(keys, values))
median_diff
```

```
{'1': 0.0,
 '2': 75.99696012159514,
 '3': 67.99728010879565,
 '4': 62.497395941835755,
 '5': 54.542975319303665,
 '6': 44.99775011249437,
 '7': 33.33111125924938,
 '8': -14.82120029423414,
 '9': -19.44002003471845,
 '10': -4.561559196780727,
 '11': 32.18926668739695,
 '12': 315.8679146748082,
 '13': 393.0,
 '14': 0.0,
 '15': 0.0,
 '16': 0.04320890504056551}
```

Figure 6. Difference of medians

According to this analysis, features '2', '3', '4', '5', '6', '7', '8', '9', '10', '11', '12', '13' have the highest predictivity level. So, they will be used in the classifier model.

2.3. Feature Selection

First the instances with low quality image will be dropped as shown in figure 7.

```
#dropping low quality images
df.drop(df[(df['0'] == 0)].index, axis=0, inplace=True)
```

Figure 7. Drop low quality images

Then the most predictive features will be kept (figure 8). This is done according to the analysis in section 2.2.

```
#keeping just probable predictive features
X = df.drop(columns=['14', '15', '16', '17', 'Class'], axis=1)
y = df.Class
```

Figure 8. Define X and y

Now, X and y will be split into two sets of train and test for evaluation purposes (figure 9).

```
X_train, X_test, y_train, y_test = train_test_split(X,y, test_size=0.4, random_state=0)
```

Figure 9. Train and Test set split

3. Model Creation

Machine learning models are made by training an algorithm iteratively. Training in machine learning literature means showing data to the algorithm and having it learn the patterns.

There are many different algorithms for classification tasks that perform well depending on the dataset and task type. In this paper, Support Vector Machine (SVM) is chosen as it is the best suited for the problem.

Support Vector Machine (SVM) tries to draw a line between all classes in the dataset. This way, it defines a boundary for each class and when a new example is given, the class which the point falls in is predicted as the new point's label.

3.1. Define Algorithm

The SVM algorithm, which is explained above, is used in this paper. Instead of implementing the algorithm from scratch, the *Sickit-Learn Python* library will be used. This module contains many different types of Machine learning algorithms which is very easy and simple to use.

Figure 10 shown the Python code for creating a SVM instance using *Sickit-Learn* library.

```
from sklearn.svm import SVC
from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
model = SVC()
```

Figure 10. Define algorithm

Notice that the inputs are scaled for SVM as it's a condition for it to work properly.

3.2. Train the Model

The code below (figure 11) shows how the model is trained and fit.

```
model.fit(X_train_scaled,y_train)
```

Figure 11. Train the model

4. Model Evaluation

Having a single numerical evaluation metric for any machine learning task is essential. This helps with defining the scope and goal of the project.

There are many different methods and formulas for evaluating a machine learning model's performance such as accuracy score, precision, recall, f1-score, and many more. Each one of the metrics have their own advantages and disadvantages relative to the type of algorithm and model. In this section, a brief definition of each with their pros and cons are presented and then the appropriate metric will be chosen.

4.1. Accuracy Score

Accuracy score, as the name suggests, calculates how many examples out of all examples are predicted accurately and correctly. Accuracy score, is best used when two conditions are met in a dataset and project objectives:

- Dataset should not be skewed and unbalanced. This is because in case one class makes up the majority, then predicting only that class can give a relatively high accuracy score.
- The project's objectives should be to predict all the classes correctly and there is no preferred outcome over any other. In simple terms, predicting one class correctly, should not have higher importance than any other class and they should all be equal in terms of weight.

When these conditions are met, accuracy score can be a good choice for evaluation metric.

4.2. Confusion Matrix, Precision, and Recall

For classification problems, confusion matrix and other related metrics such as precision and recall are a better choice. The reason behind this claim is presented below.

Confusion matrix looks not only at what examples are correctly predicted like accuracy score, it takes the predicted classes and the true labels into account too. Thus, for a classifier model, the predicted values are divided into 4 groups:

- True Positives (TP). Refers to the examples that their true label is the Positive class (usually: $y=1$) and the predicted class is also the Positive class ($\hat{y}=1$).
- True Negatives (TN): Refers to the examples that their true label is the Negative class (usually: $y=0$) and the predicted class is also the Negative class ($\hat{y}=0$).
- False Positives (FP, also known as Type 1 Error): Refers to the examples that their true label is the Negative class ($y=0$) and the predicted class is the Positive class ($\hat{y}=1$).
- False Negatives (FN, also known as Type 2 Error): Refers to the examples that their true label is the Positive class ($y=1$) and the predicted class is the Negative class ($\hat{y}=0$).

It can interpreted that confusion matrix counts for all possible outcomes of a classifier model. This way it is possible to value one outcome over the other. But the problem with confusion matrix is that it does not give us a single numerical evaluation. So, it is harder to decide if the model is improving after applying some changes to it. That's where Precision and Recall come in.

Precision wants to answer the following question:

- Of all the examples that the algorithm predicted true ($\hat{y}=1$), What percent was actually true ($y=1$)?

This is represented mathematically as:

- Number of True Positives (TP) divided by the sum of True Positives (TP) and False Positives (FP). So: Precision = TP / (TP + FP)

Recall wants to answer the following question:

- Of all the examples that were actually true ($y=1$), What percent did the algorithm predict true ($\hat{y}=1$)?

This is represented mathematically as:

- Number of True Positives (TP) divided by the sum of True Positives (TP) and False Negatives (FN). So: Recall = TP / (TP + FN)

Depending on the type of classification problem and the importance of each possible outcome, either of these two metrics is chosen. While precision insists on getting the positive predicted values correct, recall weighs getting all true examples correctly predicted. So, for medical diagnoses where patients with probability of having a disease should not be miss labeled, recall plays a more important and effective role. In a way that it pushed the algorithm to not miss any true (y=1) examples.

That is why Recall is used as the key evaluation metric here.

Figure 12 shows different metrics evaluated for the trained model.

```
SVC_preds = model.predict(X_test_scaled)
print(classification_report(SVC_preds, y_test))
```

```
              precision    recall  f1-score   support

           0       0.86      0.65      0.74       284
           1       0.59      0.83      0.69       175

    accuracy                           0.72       459
   macro avg       0.73      0.74      0.71       459
weighted avg       0.76      0.72      0.72       459
```

As discussed before, only recall is considered here for its general benefits for this case. The model has been able to get a recall score of 83% which indicates that of all the patients with **Diabetic retinopathy**, 83% of them were predicted correctly.

5. Conclusion

6. Appendices