PREDICTING DIABETIC USING MACHINE LEARNING:

A COMPARATIVE STUDY OF ALGORITHMS

ISM 6359 Data Mining

FEBRUARY 10, 2023

YAYUAN ZHANG

Table of Contents

Abstract

An increasing number of people are being diagnosed with diabetes year after year, which represents a worldwide health problem. In recent years, machine learning algorithms have become a potential tool for predicting the onset and progression of diabetes.

In this article, my objective is to explore diabetes data more thoroughly following the CRISP-DM framework and obtain an understanding of the most indicative features of diabetes. The ultimate objective of the article is to utilize a variety of machine learning algorithms to develop predictive models with high accuracy that can predict whether a person has diabetes or not. The article would display an outline of each phase of the data-mining process, including the tool that was used, Business insight, data preparation, feature engineering, EDA, data analysis, train-test split approaches, the tweaking of parameters, modeling, evaluation and conclude with the 3 W's.

Business insight

1.1 Business Problems and Motivations

This article will examine and make predictions about the Diabetic based on CRISP-DM framework. Why predict Diabetes is important? According to the World Health Organization (WHO), 422 million people worldwide have diabetes. (WHO) Diabetes can not only be detected and managed early to delay or prevent serious complications but also to reduce the likelihood of complications and enhance overall health. Models can help people and their families quickly identify diabetes and prepare for treatment as early as possible.

1.2 State Data Mining Tool

In this process, the Python programming language that acts on the PyCharm and Jupyter platforms is my data mining tool and environment. The two main reasons I chose this tool and environment were. During my academic career at SPU, I took a course in data analysis using Python programming and completed some Bootcamp online courses last year. I realized that this was a professional way to implement deeper machine learning in a programming environment using one programming language. In addition, I was eager to hone my programming skills and add the results to my portfolio as a project to demonstrate my data mining skills.

1.3 Stakeholders

- □ Patients and their families: Patients with diabetes or at risk of developing diabetes will be the main beneficiaries for the prediction model. The model will help them and their families to understand whether the patient has diabetes or not.
- □ Insurance companies: The companies would better understand the health risks of their customers and might adjust their insurance rates.

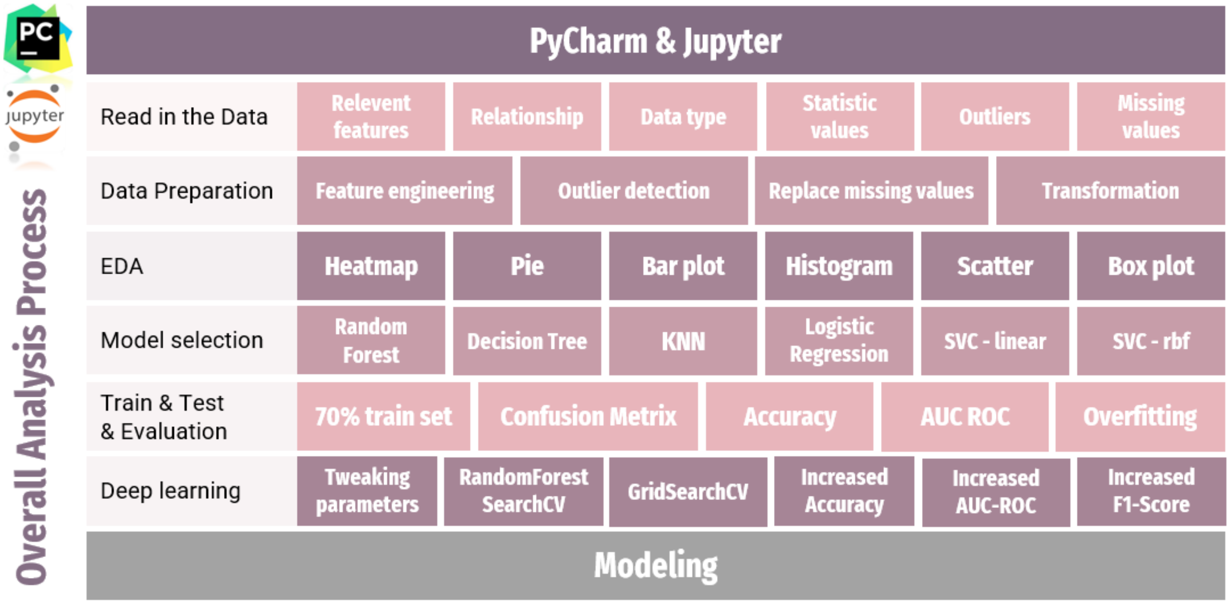
- □ Government department: The Government department can monitor the growing trend in the number of people with diabetes and control an increase through advocacy or policy development, for example, by supporting improved dietary choices.

1.4 Data & Source

The dataset is from the National Institute of Diabetes and Digestive and Kidney Diseases and is available on Kaggle. The dataset contains 768 data points and 9 features. Each data point represents basic information about each patient and their health data, such as BMI, insulin levels, age, etc.

2. Data Analysis

2.1 Workflow Analysis process



Overall Analysis Process

PyCharm & Jupyter

Read in the Data	Relevent features	Relationship	Data type	Statistic values	Outliers	Missing values
Data Preparation	Feature engineering	Outlier detection	Replace missing values	Transformation		
EDA	Heatmap	Pie	Bar plot	Histogram	Scatter	Box plot
Model selection	Random Forest	Decision Tree	KNN	Logistic Regression	SVC - linear	SVC - rbf
Train & Test & Evaluation	70% train set	Confusion Metrix	Accuracy	AUC ROC	Overfitting	
Deep learning	Tweaking parameters	RandomForest SearchCV	GridSearchCV	Increased Accuracy	Increased AUC-ROC	Increased F1-Score

Modeling

Throughout the workflow analysis process, I first import libraries and datasets to understand the data, including but not limited to understanding the meaning of features, defining target variables, selecting features, checking data types, checking for missing values, checking for outliers, checking statistical values of features, and checking relationships between features. Exploratory data analysis was performed after I had processed the necessary data preparation for the model, such as data cleaning and feature engineering.

During the modeling process, I divided the data into training and test sets and determined the model with the highest accuracy as the best model for the data. After fitting the model, I evaluated and considered the metrics of the model to ensure that the model looked good. In this phase, I performed one of the most important deep machine learning steps, which is tuning the model parameters. Finding the best configuration of the model (the right combination of parameters) can greatly improve the performance of the model and the overall effectiveness of the predictions. Finally, I built the "best" model using the "optimal" parameters and evaluated the model again. In this paper, I examine the best model and the next best model.

2.2 Prep Tools & Coding

2.2.1 Import library

First, I import the required libraries, including the parts mainly for visualization, libraries and packages for model selection and processing feature selection, ignoring some warnings, and so on.

2.2.2 Set rules for notebooks

I use the following code to retrieve all the columns and all the rules, it will allow all the input codes in the display data box without restriction and ensure that I can see all the columns.

- pd.options.display.max_columns = None
- pd.options.display.max_rows = None

2.3 Data Understanding & Data Pre-processing

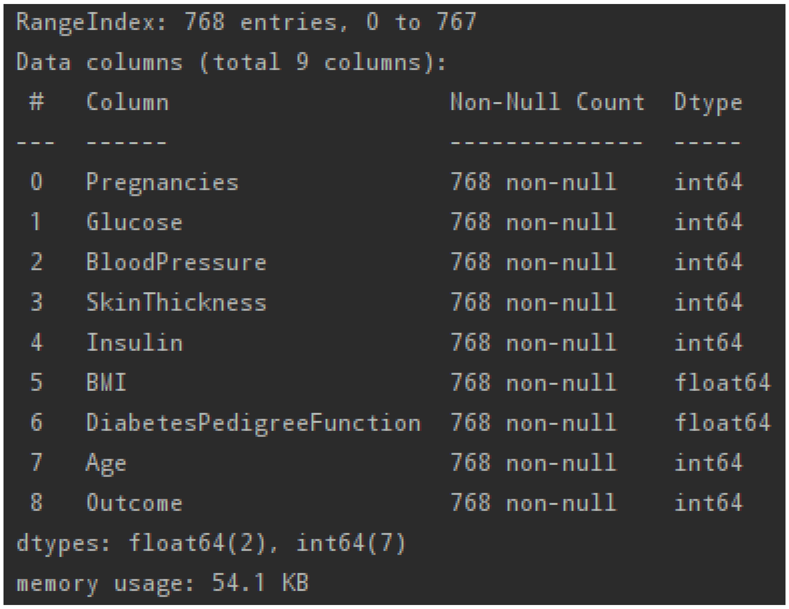
2.3.1 Basic information

The data has 768 observations with 9 features. The target variable is the outcome. Other features named: Pregnancies, Glucose, Blood Pressure, Skin Thickness, Insulin, BMI, Diabetes Pedigree Function, Age.

- # Checking the head of data and define the target variable
- data.head()
- # Checking the data frame size to learn what amount of data are dealing with
- data.shape

In the dataset, nine features are numeric fields, seven of which are integral and two of which are floating points.

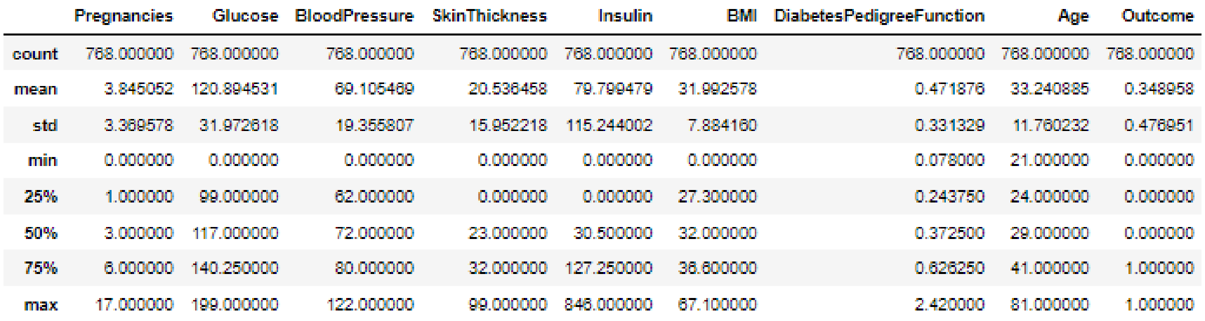
- # Checking the datatypes for all features/columns & whether have empty cell or not
- data.info()
- data.dtypes



```
RangeIndex: 768 entries, 0 to 767
Data columns (total 9 columns):
 #   Column                    Non-Null Count  Dtype
---  ------                    --------------  -----
 0   Pregnancies               768 non-null    int64
 1   Glucose                   768 non-null    int64
 2   BloodPressure             768 non-null    int64
 3   SkinThickness             768 non-null    int64
 4   Insulin                   768 non-null    int64
 5   BMI                       768 non-null    float64
 6   DiabetesPedigreeFunction  768 non-null    float64
 7   Age                       768 non-null    int64
 8   Outcome                   768 non-null    int64
dtypes: float64(2), int64(7)
memory usage: 54.1 KB
```

Next, I checked the distribution of the values of all the numeric fields. the describe function basically checks the distribution of the values of all the numeric fields, so it gets the counts, means, standard deviations, etc. for each feature.

- # Reading the data error describe to check spread of values across all numerical fields
- data.describe()



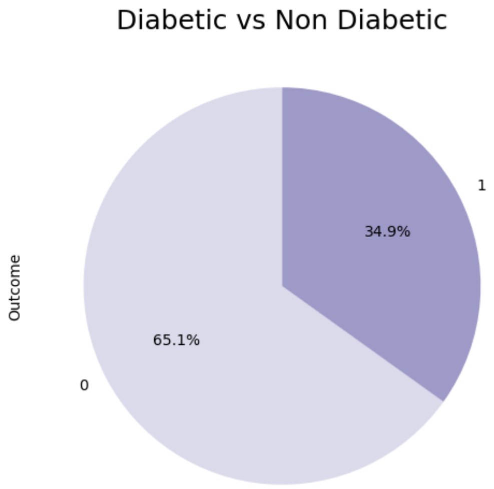
	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
count	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000
mean	3.845052	120.894531	69.105469	20.536458	79.799479	31.992578	0.471876	33.240885	0.348958
std	3.369578	31.972618	19.355807	15.952218	115.244002	7.884160	0.331329	11.760232	0.476951
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.078000	21.000000	0.000000
25%	1.000000	99.000000	62.000000	0.000000	0.000000	27.300000	0.243750	24.000000	0.000000
50%	3.000000	117.000000	72.000000	23.000000	30.500000	32.000000	0.372500	29.000000	0.000000
75%	6.000000	140.250000	80.000000	32.000000	127.250000	36.600000	0.626250	41.000000	1.000000
max	17.000000	199.000000	122.000000	99.000000	846.000000	67.100000	2.420000	81.000000	1.000000

2.3.2 EDA

Before building a model, I prefer to quickly view the data distribution through a data report to gain more insight and understand the relationship between each feature. Therefore, in this section, I use profile reports and simple charts to create a quick preview data report of the entire data.

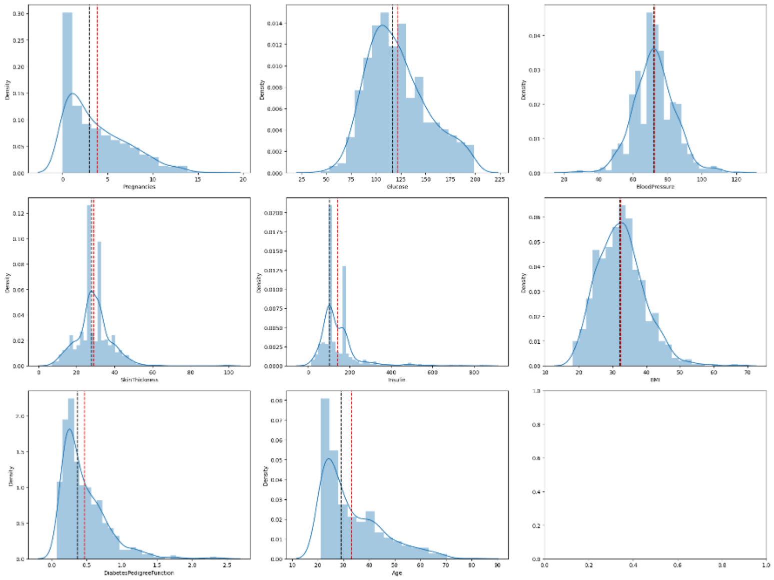
- ProfileReport(data)

The result of Profiling Report is attached the submit document, named Pandas Profiling Report.



According to the following pie chart of the percentage of people with and without diabetes, a total of 34.9% of people in this dataset have diabetes. 1 for diabetic patients and 0 for non-diabetic patients.

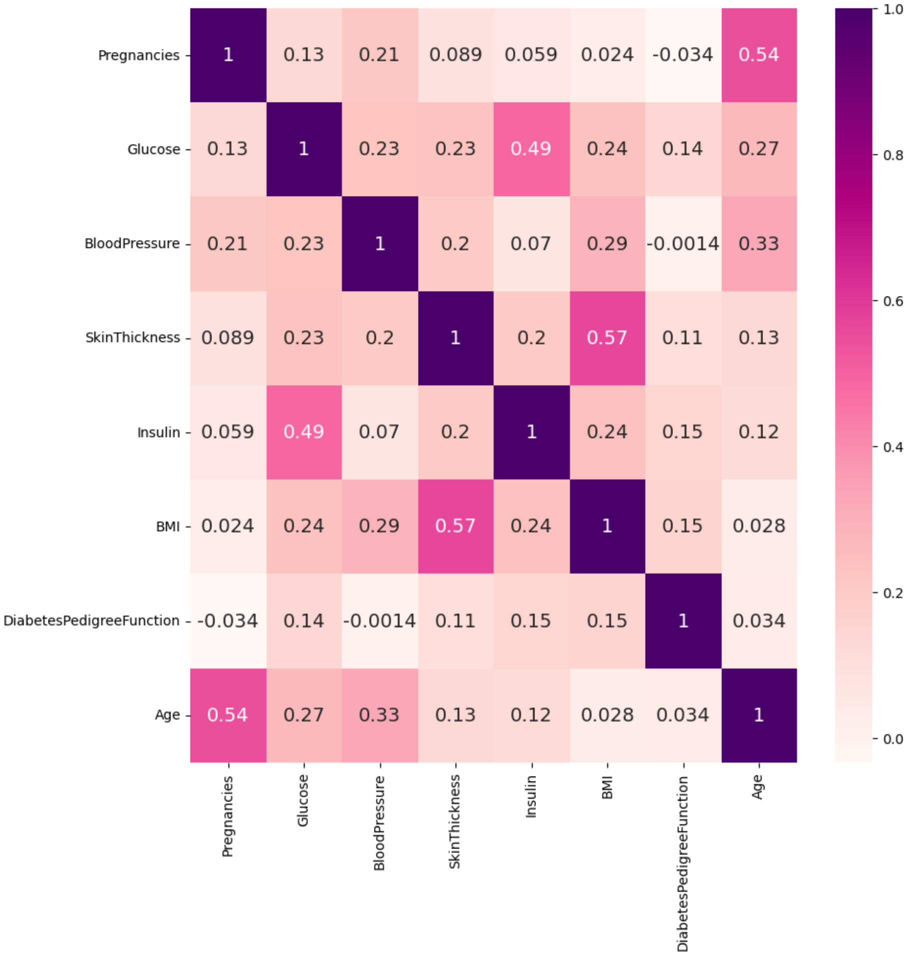
```
pie_colors = sns.color_palette("Purples",3)
plt.subplots(figsize=(6,6))
data['Outcome'].value_counts().plot.pie(colors=pie_colors, autopct='%1.1f%%', startangle = 90)
plt.title('Diabetic vs Non Diabetic', fontsize=18)
plt.show()
```



In this dataset, all variables display positive distribution.

```
def make_distplot(data, col, ax):
        sns.distplot(data[col], ax = ax)
        ax.axvline(data[col].mean(), linestyle = '--', color = "red")
        ax.axvline(data[col].median(), linestyle = '--', color = "black")
numeric_columns = list(data.select_dtypes(include=np.number).columns)
categorical_columns = list(data.select_dtypes(include="object").columns)
```

```
fig, ax = plt.subplots(3,3, figsize = (20,15))

ax = np.ravel(ax)

for i in range(len(numeric_columns)):

      make_distplot(data, numeric_columns[i], ax[i])

plt.tight_layout()
```



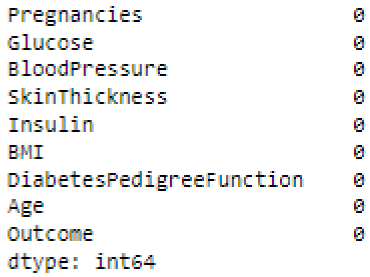
Normally, if we fit highly correlated data in model, it results in the overfitting problem. The good news is there is not highly correlated feature with Diabetes in this data set.

```
plt.figure(figsize=(10, 10))

sns.heatmap(data.corr(), annot=True, cmap="RdPu", annot_kws={"size":14})
```

2.3.3 Missing values Imputation & Analysis

After checking the missing values by the following codes. It seems that there are no missing values in the dataset.

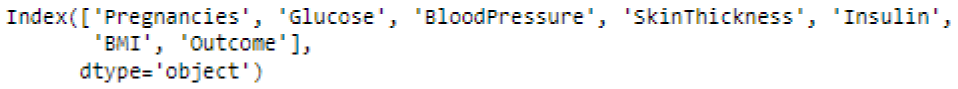
- 🗎 data.isnull().sum()



```
Pregnancies                 0
Glucose                     0
BloodPressure               0
SkinThickness               0
Insulin                     0
BMI                         0
DiabetesPedigreeFunction    0
Age                         0
Outcome                     0
dtype: int64
```

However, I found that some variables have a minimum value of 0 in the descriptive statistics table, which is illogical. Therefore, I will continue exploring these variables and treating them accordingly.

- 🗎 # checking all columnists which equal to critical zero in any one of its rows
- 🗎 data.loc[:, (data == 0). any(axis=0)].columns



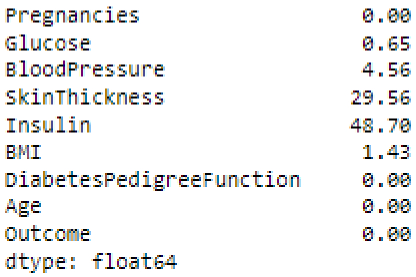
```
Index(['Pregnancies', 'Glucose', 'BloodPressure', 'SkinThickness', 'Insulin',
       'BMI', 'Outcome'],
      dtype='object')
```

The zero-value analysis across columns is basically understood and attempted from the following business perspective and is reasonable:

- □ Pregnancy can be 0.
- □ Outcome is targeting variable with value 0 or 1.
- □ Glucose, Blood Pressure, Skin Thickness, Insulin, BMI cannot be zero in the real world. A value of zero for these columns means that the data is not available and needs to be replaced.

In reality, there are many ways to deal with missing values. However, this dataset only has 768 samples, and we cannot go to remove a large amount of missing value data. Therefore, I decided to fill the missing value 0 with NaN and check the percentage of missing values in each column before deciding whether to ignore or fill the data.

- # Replacing 0 value with null values in values with NaN -
- cols = ['Glucose', 'BloodPressure', 'SkinThickness', 'Insulin','BMI']
- data[cols] = data[cols].replace(0,np.NaN)
- # Checking the percentage of missing values across all the columns
- round(100*(data[data.columns].isnull().sum()/len(data.index)),2)



```
Pregnancies                  0.00
Glucose                      0.65
BloodPressure                4.56
SkinThickness               29.56
Insulin                     48.70
BMI                          1.43
DiabetesPedigreeFunction     0.00
Age                          0.00
Outcome                      0.00
dtype: float64
```

According to the results above it was found that these missing values are huge, close to half. I decided to try to replace them. According to the data report and visualization, it is clear that these variables follow a normal distribution (click here). Therefore, I decided to fill the missing value 0 with statistical evidence by following the steps below, i.e., filling the median of the feature.

- # Printing to check median value of Glucose across different outcomes
- print(data.groupby('Outcome')['Glucose'].agg(['median']))

It was obvious that all the null values I knew of resulted in zero. My solution was to create a function with a for loop in the list to calculate the median of each column, so that the null value of each feature was replaced by its median, and the medians were grouped by result.

- def impute_median(dataFrame,cols):
- for col in cols:
- dataFrame[col] = dataFrame.groupby('Outcome')[col].transform (lambda x: x.fillna(x.median()))
- # Calling the function defined above to perform imputation with median

- impute_median(data,cols)

Before moving on to the next step, an important step is to check for missing values and make sure that all missing values in all columns are processed.

- round(100*(data[data.columns].isnull().sum()/len(data.index)),2)

2.3.4 Defining target variable

In the last step, my target feature is each column, but what pops out is the outcome of diabetes and the predictor variable is diabetes. Therefore, I split the entire data into X and Y by using the following code.

- X=data
- y=data.pop('Outcome')
- Also, I am trying to split into train set and test set and set the train size is 70%.
- X_train, X_test, y_train, y_test = train_test_split(X, y, train_size=0.7, random_state=42,stratify=y)

2.3.5 Outlier

Also, I see in the pregnancy feature that the maximum month of pregnancy is 17, which doesn't make any sense in the real world. My solution was to leave them unchanged and to scale the data. It is because I take into account the need to be careful with outliers. Some outliers may help us to observe important results. Moreover, the variance between different predictor variables is largely variable and scaling the data will help in predictive modeling.

- # Fitting the standard scaler on train set and transforming across all the sets of X
- scaler=StandardScaler()
- scaler.fit(X_train)

- X_train = scaler.transform(X_train)
- X_test = scaler.transform(X_test)

3. Modeling Selection

3.1 Multiple Model

For model selection, I tested 7 classification algorithms and finally selected the model with the highest accuracy value as the best model. Linear SVM and non-linear SVM are both classification algorithms that use Support Vector Machines (SVMs) First, I created a list of models and initialized all model objects.

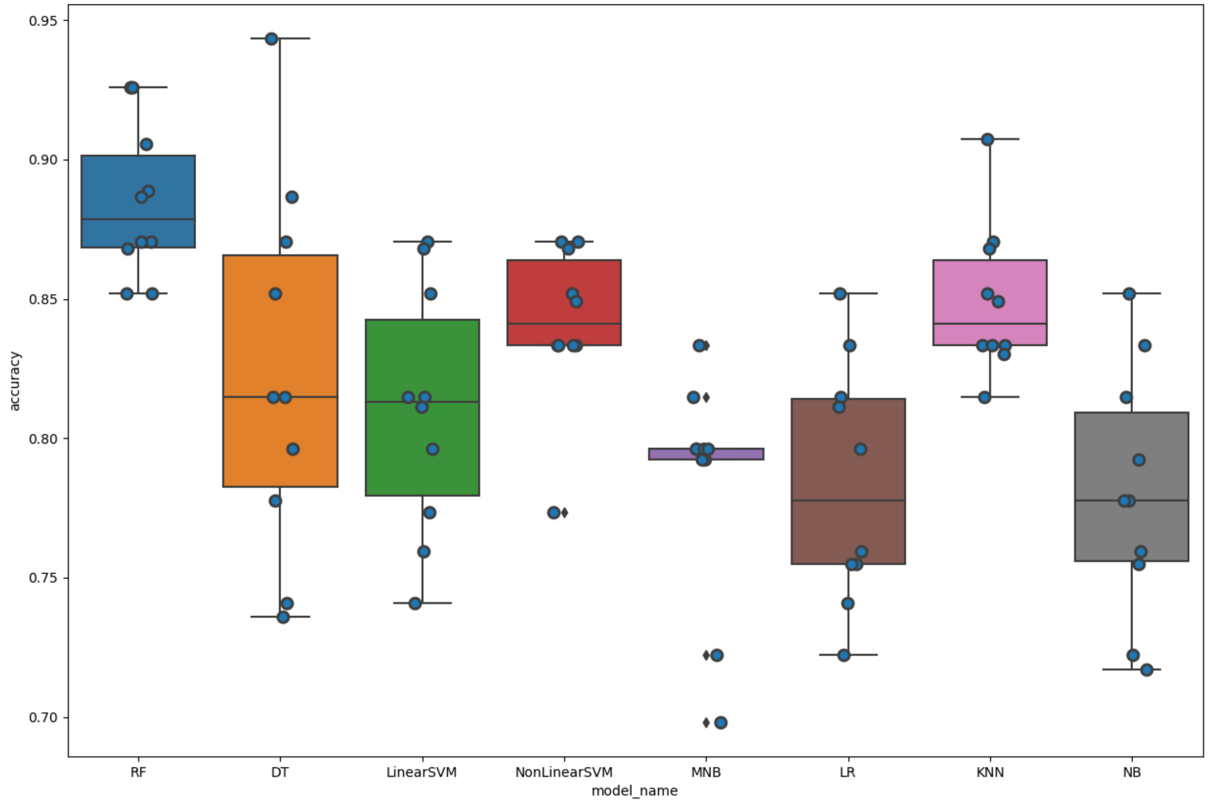
- models = []
- models.append(('RF',RandomForestClassifier()))
- models.append(('DT',DecisionTreeClassifier()))
- models.append(('LinearSVM',SVC(kernel='linear')))
- models.append(('NonLinearSVM',SVC(kernel='rbf')))
- models.append(('MNB',MultinomialNB()))
- models.append(('LR',LogisticRegression()))
- models.append(('KNN',KNeighborsClassifier()))
- models.append(('NB', GaussianNB()))

3.2 K-Fold & Average accuracy

In this step, I use the k-fold cross-validation method, collapsing the dataset 10 times and finding the highest accuracy of the mean. I chose 10 as the k-value because 10 provides a good balance between computational cost and evaluation quality. Too many folds would result in a low bias in MSE but a higher variance. 10 times provides a large enough validation set to obtain an accurate assessment of the model performance while also ensuring that most of the data is used to train the model.

- folds = 10
- kfold=KFold(n_splits=folds,random_state=45,shuffle=True)
- cv_df = pd.DataFrame(index=range(folds * len(models)))

Next, I will show the average accuracy of various models after ten folds by Bexplot.



According to the box plot, it shows that the random forest model performs better than the other models because it has a higher mean than the other models, and I have also printed out the accuracy of each model below.

RF	0.864116
NonLinearSVM	0.836164
DT	0.815863
KNN	0.815514
LinearSVM	0.811985
LR	0.783892

NB	0.780119
MNB	NaN

Therefore, I consider the random forest model as a "best model" and will learn the top1 and top2 models in depth below.

4. Best Model

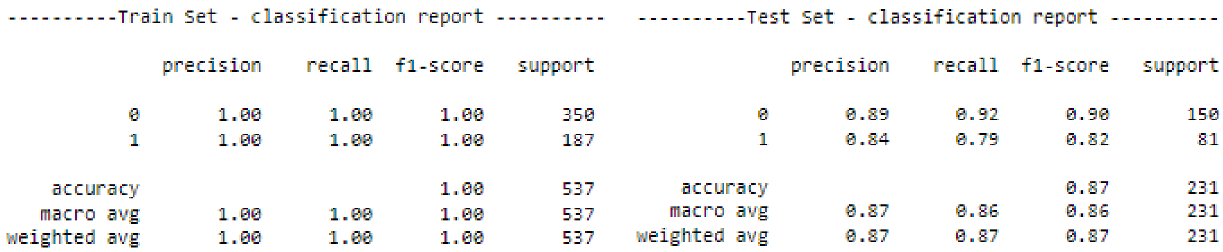
4.1 Random Forest Model – Fit on Train & Test Set

After determining the best model for the dataset, I started fitting the model to the training set.

- rfc = RandomForestClassifier()
- rfc.fit(X_train,y_train)

4.2 Random Forest Model – Evaluation Train & Test set

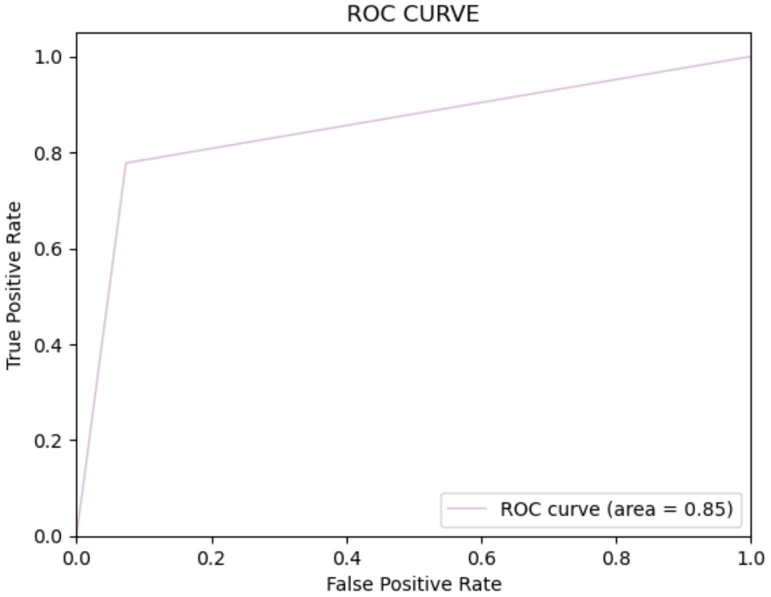
Based on the predictions for the training and test sets, I got the classification report and found a big red flag in my model - overfitting. In the classification report, the training set scored high in accuracy, precision, recall and F1. However, the model did not generalize well from the training data to unseen data, and it had a slightly lower accuracy of 0.87 on the test set.



```
----------Train Set - classification report ----------   ----------Test Set - classification report ----------

              precision    recall  f1-score   support                  precision    recall  f1-score   support

           0       1.00      1.00      1.00       350              0       0.89      0.92      0.90       150
           1       1.00      1.00      1.00       187              1       0.84      0.79      0.82        81

    accuracy                           1.00       537       accuracy                           0.87       231
   macro avg       1.00      1.00      1.00       537      macro avg       0.87      0.86      0.86       231
weighted avg       1.00      1.00      1.00       537   weighted avg       0.87      0.87      0.87       231
```

Next, I created a graph to view the AUC-ROC values. The ROC value for the random forest model is 85.22%. We know that a high AUC value equates to a model with better predictive power and also means that the model is better at distinguishing between positive and negative categories. In addition, the confusion matrix shows that I get a good number of true positives and true negatives.

```
[[138  12]

[ 17  64]]
```

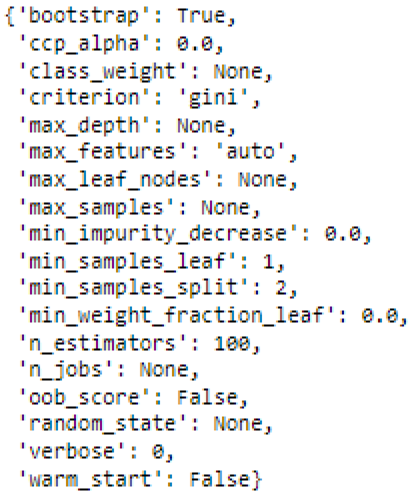


```
RF roc_value: 0.8460493827160493
```

4.3 Random Forest Model – Tweaking parameters

However, I still need to address the overfitting issue and try to improve the performance of the model by tweaking the parameters. This is because a key part of making the most of the model is the tweaking of the hyperparameters. The good thing is that the one of solution for the overfitting problem is exactly to use the K-fold cross-validation method to tune the parameters. Thus, my ultimate goal is to have a robust, accurate, and non-overfitting model. First, I needed to record the default parameters of the current model.

🗎 rfc.get_params()



```
{'bootstrap': True,
 'ccp_alpha': 0.0,
 'class_weight': None,
 'criterion': 'gini',
 'max_depth': None,
 'max_features': 'auto',
 'max_leaf_nodes': None,
 'max_samples': None,
 'min_impurity_decrease': 0.0,
 'min_samples_leaf': 1,
 'min_samples_split': 2,
 'min_weight_fraction_leaf': 0.0,
 'n_estimators': 100,
 'n_jobs': None,
 'oob_score': False,
 'random_state': None,
 'verbose': 0,
 'warm_start': False}
```

I created a model based and searched for the best combination of parameters using a RandomizedSearchCV(). In that formula, I need to list the parameters to be searched and set the parameter range, then bring all the available parameters into the formula and finally find the parameter combination with the highest accuracy. In this step, I set the cross-validation to 10 times because 10 times provides a good balance between computational cost and evaluation quality.

- folds = KFold(n_splits = 10, shuffle=True, random_state=42)
- param_grid = {

  'criterion': ["entropy", "gini"],

  'max_depth': range(0,10,1),

  'min_samples_leaf': range(0,10,1),

  'min_samples_split': range(0,10,1),

  'n_estimators': [40,50,60,70,80,100,200,300],

  'max_features': ["auto", "sqrt", "log2"]}
- rf_grid_search = RandomizedSearchCV(estimator = rf,

  param_distributions = param_grid,

  scoring = 'accuracy',

  cv = folds,

```
                    n_jobs = -1,

                    verbose = 1,

                    n_iter = 500,

                    random_state = 42)
```

🗎 rf_grid_search.fit(X_train, y_train)

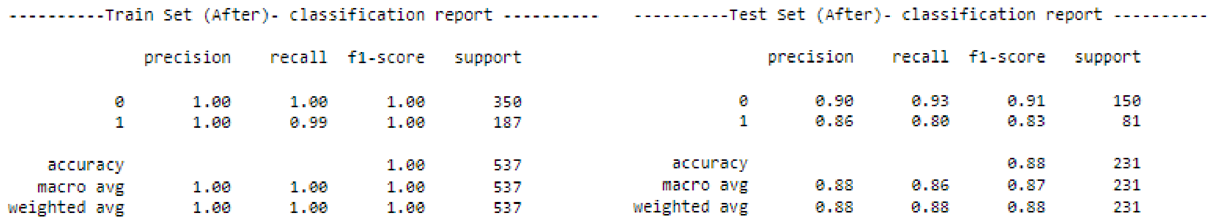
I can get score of: 0.8864779874213836 using {'n_estimators': 40, 'min_samples_split': 7, 'min_samples_leaf': 1, 'max_features': 'log2', 'max_depth': 9, 'criterion': 'entropy'}

After checking everything and returning what are the best parameters, it has reached the perfect large accuracy. I used the following code to load the best combination of parameters into the model.

🗎 rfc_h.fit(X_train, y_train)

4.4 Random Forest Model – Prediction with best parameters

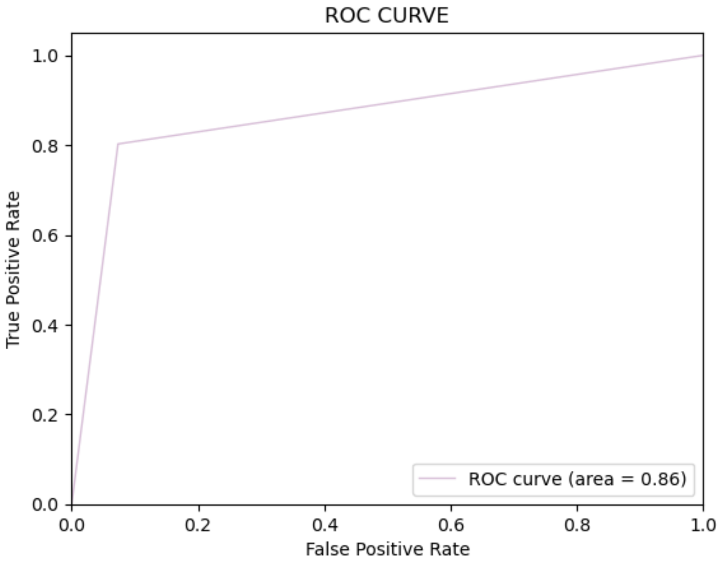
Now, I use the best combination of parameters to make predictions on the training and test sets and check the classification report again. We can see that the accuracy of the test set has improved from 0.87 to 0.88.



```
----------Train Set (After)- classification report ----------    ----------Test Set (After)- classification report ----------

              precision    recall  f1-score   support                          precision    recall  f1-score   support

           0       1.00      1.00      1.00       350                       0       0.90      0.93      0.91       150
           1       1.00      0.99      1.00       187                       1       0.86      0.80      0.83        81

    accuracy                           1.00       537                accuracy                           0.88       231
   macro avg       1.00      1.00      1.00       537               macro avg       0.88      0.86      0.87       231
weighted avg       1.00      1.00      1.00       537            weighted avg       0.88      0.88      0.88       231
```

The AUC-ROC value of the test set also improved from 84.60% to 86.46%. The number of true positives and true negatives of the confusion matrix was also improved.

```
[[139  11]

 [ 16  65]]
```



RF roc_value: 0.864567901234568

4.5 Random Forest Model – Summary

The random forest model with default values has an AUC-ROC value of 84.60% and an accuracy of 0.87, with overfitting on the training set. This is because the model performed well on the training data but underperformed on the evaluation data. However, in the process of adjusting the parameters, the model is no longer overfitted and has a higher AUC-ROC value of 86.46% and a better F1 score and accuracy with an accuracy of 0.88. Therefore, I define the hyperparameter-adjusted model as the best model with the best parameters.

5. Top 2 Model – SVC(Kernel = ‘rbf’)

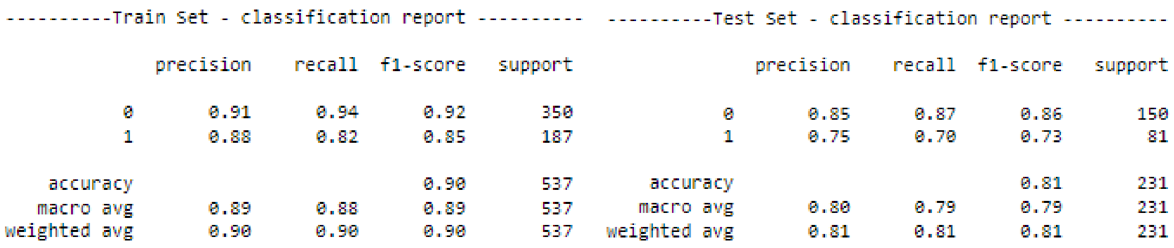
5.1 SVC – Fit on Train & Test Set

Although I got the best model, I will continue to explore the model with the second highest accuracy value - SVC.

- svc = SVC(kernel='linear')
- svc.fit(X_train,y_train)

5.2 SVC – Evaluation Train & Test set

For the SVC model, I used the same method as the random forest model to split the training and test sets and get the classification report. It seems little overfitting occur here. The accuracy of the training set was 0.90 and the accuracy of the test set was 0.81.

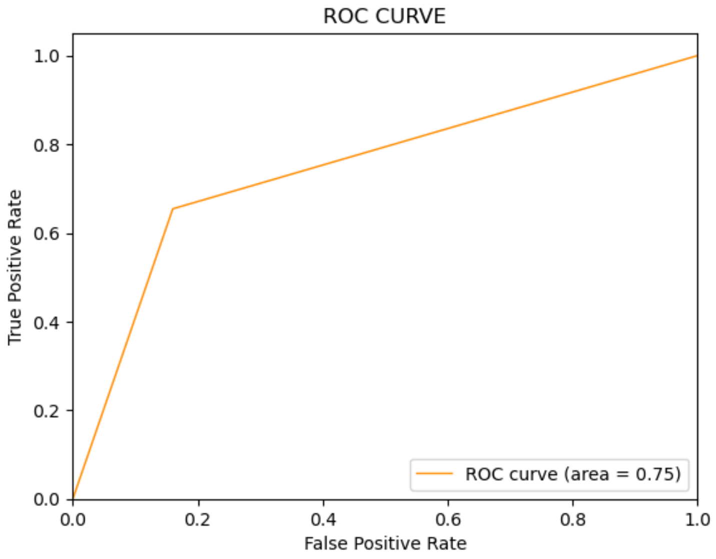


```
----------Train Set - classification report ----------  ----------Test Set - classification report ----------

              precision    recall  f1-score   support                 precision    recall  f1-score   support

           0       0.91      0.94      0.92       350              0       0.85      0.87      0.86       150
           1       0.88      0.82      0.85       187              1       0.75      0.70      0.73        81

    accuracy                           0.90       537       accuracy                           0.81       231
   macro avg       0.89      0.88      0.89       537      macro avg       0.80      0.79      0.79       231
weighted avg       0.90      0.90      0.90       537   weighted avg       0.81      0.81      0.81       231
```

The AUC-ROC value of the SVC model with default parameters is 74.72%. It does not seem to be higher than the random forest model. Also, the number of true positives and true negatives in the confusion matrix are seem not bad.

[[126 24]

[28 53]]



svc roc_value: 0.7471604938271604

5.3 SVC – Tweaking Parameters

Tweaking parameters at SVC is the same as finding the best combination of parameters at RF. I use the GridSearchCV formula to search for the highest precision parameter combinations.

- param_grid = {'C': [0.1, 0.99, 1, 10,100],

  'gamma': [0.0001, 0.001, 0.01, 0.1, 'scale', 'auto'],

  'degree': [1, 2, 3, 4],

  'max_iter': [-1, 10, 100],

  'decision_function_shape': ['ovr', 'ovo'],

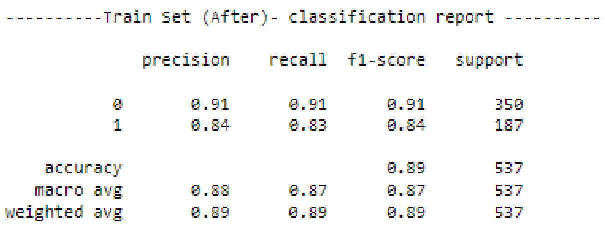
  'cache_size': [100, 1000]}

  base_model = SVC(kernel='rbf', C=1)
- grid_search = GridSearchCV(base_model, param_grid, cv=folds, scoring='accuracy', refit=True, verbose=3, n_jobs=-1)
- grid_search.fit(X_train,y_train)

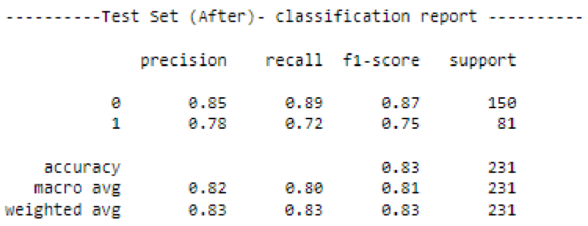
We can get score of : 0.8474842767295598 using {'C': 100, 'cache_size': 100, 'decision_function_shape': 'ovr', 'degree': 1, 'gamma': 0.01, 'kernel': 'rbf', 'max_iter': -1}

5.4 SVC – Prediction with best parameters

To my surprise, adjusting the hyperparameters of the SVC resulted in a significant improvement in accuracy. I got a score of 84.75% with the best combination of parameters. Next, I fitted the best parameters to the training set and the test set and checked the results.



```
----------Train Set (After)- classification report ----------

              precision    recall  f1-score   support

           0       0.91      0.91      0.91       350
           1       0.84      0.83      0.84       187

    accuracy                           0.89       537
   macro avg       0.88      0.87      0.87       537
weighted avg       0.89      0.89      0.89       537
```

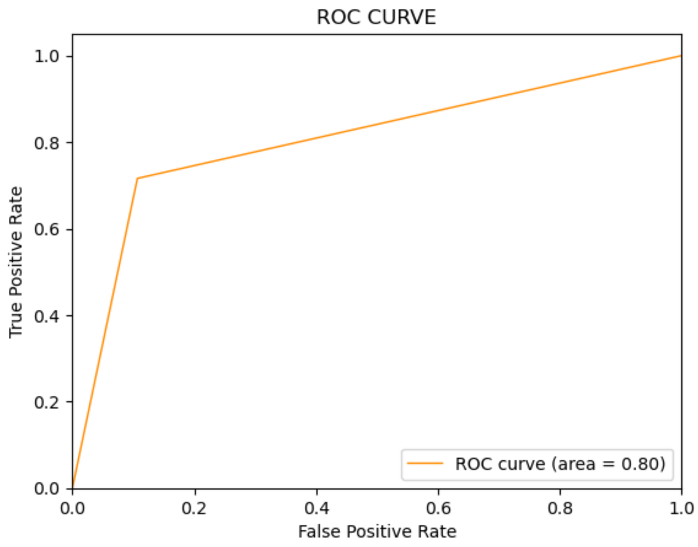


```
----------Test Set (After)- classification report ----------

              precision    recall  f1-score   support

           0       0.85      0.89      0.87       150
           1       0.78      0.72      0.75        81

    accuracy                           0.83       231
   macro avg       0.82      0.80      0.81       231
weighted avg       0.83      0.83      0.83       231
```

Based on the classification reports of the training and test sets, it can be seen that the accuracy of both sets is above 0.80. And the overfitting problem is resolved. The AUC-ROC value of the test set improved from 74.72% to 80.47%, and the number of false positives and false negatives on the confusion matrix was less than that of the model with default parameters.

[[134 16]

[23 58]]



SVC roc_value: 0.8046913580246913

5.5 SVC – Summary

The SVC model with default parameters on the test set had an AUC-ROC value of 74.72% and an accuracy of 0.81, with overfitting on the training set. After adjusting the parameters by deep learning, the model had an even higher AUC-ROC value of 80.47% and an accuracy of 0.83. Although the accuracy of this model is not as high as that of RF, in my opinion, an accuracy of over 80% is already a good performing model. What is most rewarding is that the span of accuracy improvement is so large. All in all, tuning the hyperparameters of the SVC model was a very interesting deep learning experience, and it surprised me.

3W

What went well?

My experience with this data mining project has been generally positive. Previous experience modeling and creating small projects using Python machine learning was particularly helpful for the project. This familiarity with the language allowed me to smoothly go through the entire process of creating models. Another thing that went well was that I was able to analyze missing values in the dataset through real-life common sense and smoothly decide what to do with them. In addition, the following tasks all went well:

- □ finding available datasets
- □ replacing missing values with median values
- □ splitting the training and test sets
- □ building and evaluating the model
- □ tuning the parameters of the random forest model

Overall, I am pleased with the results of my data mining project and am excited to continue developing my skills in this area.

What did not go well?

There were a couple of things that didn't go well in my data mining project. First, I initially ignored outliers when processing the data, which led to some incorrect data being used to build the model. This caused me to go back and revisit the data and pre-processing steps, causing delays and extra work.

Second, I encountered significant challenges in optimizing the parameters of the SVC model. I spent several days trying to tweak the parameters but did not resolve the issue until the day before the deadline. I tried to use GridSearchCV to find the best combination of parameters. After three days of running on standby, I started troubleshooting and debugging and I recognized perhaps my laptop device system cannot support such a large algorithm. Therefore, at the first I looked at the algorithm of the code to find a solution, and even consulted many sources and made the necessary optimizations to the code. Then, I decided to switch to PyCharm environment and use a gaming desktop computer (with a better system and equipment) to speed up the results. In the end, I found that I only needed to modify a few parameter ranges or run a handful of parameters at a time to successfully solve the problem, also get good enough accuracy. Despite the challenges, I learned a lot from this experience and will apply this knowledge to future data mining projects.

What I would do differently next time?

Reflecting on my data mining project, there are a few things I would do differently next time. First, I would prioritize spending more time and effort on data pre-processing and understanding preparation. I now recognize the importance of preparing the data thoroughly, as it will save me a lot of time in my next data mining project, as well as lead to better performance for my models.

Second, I learned a valuable lesson about using code to tune parameters. Next time, I will be more patient and avoid rushing through the process. I now realize that trying to tweak all the parameters at once or running many wide ranges of parameters can cause my computer to get stuck, which can be time consuming and frustrating. I will aim to select a narrower range of parameter choices and test one or more parameters at a time to avoid overwhelming my system. In addition, I would allow some time for this part of the work to avoid a similar situation from happening again. Overall, these changes will help me to handle data mining projects more efficiently in the future.

References

World Health Organization. (n.d.). Diabetes. World Health Organization. Retrieved February 10, 2023, from https://www.who.int/health-topics/diabetes#tab=tab_1