Milestone#1

Task 1: Study on the characteristic of the data and identify the quality issues found

Data Characteristics :

- Categorical features recorded with same format (AXXX)
- There are outliers and class imbalance
- Dataset is out of date
- Categorical feature has many types but several are occur
- Including both categorical and numerical (multivariate) data make it difficult to select features to train the model

Quality issues :

- Bias
    - Imbalanced class could caused the prediction bias
- Performance
    - Overfitting, high accuracy, but low precision and recall
- Fairness
    - Prediction rely on specific people group (specific features)
- Reliable
    - Data's bias and unfairness caused the prediction unreliable

Task 2: Define the goals and a suitable measure for the quality issues

1st goal -> Improve the prediction performance

Measures:

- accuracy, precision, recall, and f1-score
- confusion matrix

Processes :

- Create a baseline prediction model using default parameter and features
- Create improved version of prediction models
- Compare those 4 measure with the baseline

2nd goal -> Reduce the prediction bias (in term of imbalance class)

Measures:

- precision, recall, and f1-score

Processes :

- EDA on dataset
- Apply data sampling techniques (over, under, combined)
- Create models using balanced dataset and compare with baseline model

Task 3: Explore different kinds of machine learning models developed with different modelling techniques. Then,choose the machine learning techniques, implement the models using scikit-learn,
and train the models.

Implementation steps :

- Data exploration and analysis (EDA)
- Data preprocessing
- Training models
- Testing models
- Summarise model results

In this task we have explored, trained and tested a total of 6 machine learning models using scikit-learn, each model assumption is described below and the complete implementation of models are presented in “Milestone1_materials” directory.]

Noted : The model's result may vary depending on individual data preprocessing steps.

Gaussian Naives Bayes Classifier (developer: Wendy)

A short Introduction Naive Bayes Algorithm

Naive Bayes - a supervised learning algorithm, which is based on Bayes theorem and used for solving classification problems.

One of the reasons that I chose this algorithm is that according to the assumption made by Naive Bayes , each feature in the dataset is given the same weight. In other words, every feature is important. None of the features is irrelevant and assumed to be contributing equally to the predicted result. Additionally,as features contain both numerical and categorical values, I chose Gaussian Naïve Bayes Classifier in which each feature is assumed to be distributed to a Gussian distribution whose density function is defined by two parameters called mean and standard deviation. [1-3]

This figure is shown about the testing result of the training model

GaussianNaiveBayes CLassification

```
In [39]: # Fitting Naive Bayes to the Training set
         from sklearn.naive_bayes import GaussianNB
         from sklearn.metrics import mean_squared_error, r2_score, accuracy_score, confusion_matrix, classification_report, fbeta_score
         classifier = GaussianNB()
         classifier.fit(x_train, y_train)

Out[39]: GaussianNB()

In [40]: # Predicting the Test set results
         y_pred = classifier.predict(x_test)

In [41]:
         print("Test predict accuracy score: ", accuracy_score(y_test,y_pred),"\n")

         print("Confussion Matrix: \n", confusion_matrix(y_test, y_pred),"\n")

         print("Classification report according to Test prediction: \n", classification_report(y_test, y_pred))

         Test predict accuracy score:  0.7266666666666667

         Confussion Matrix:
          [[163  46]
          [ 36  55]]

         Classification report according to Test prediction:
                       precision    recall  f1-score   support

                    1       0.82      0.78      0.80       209
                    2       0.54      0.60      0.57        91

             accuracy                           0.73       300
            macro avg       0.68      0.69      0.69       300
         weighted avg       0.74      0.73      0.73       300
```

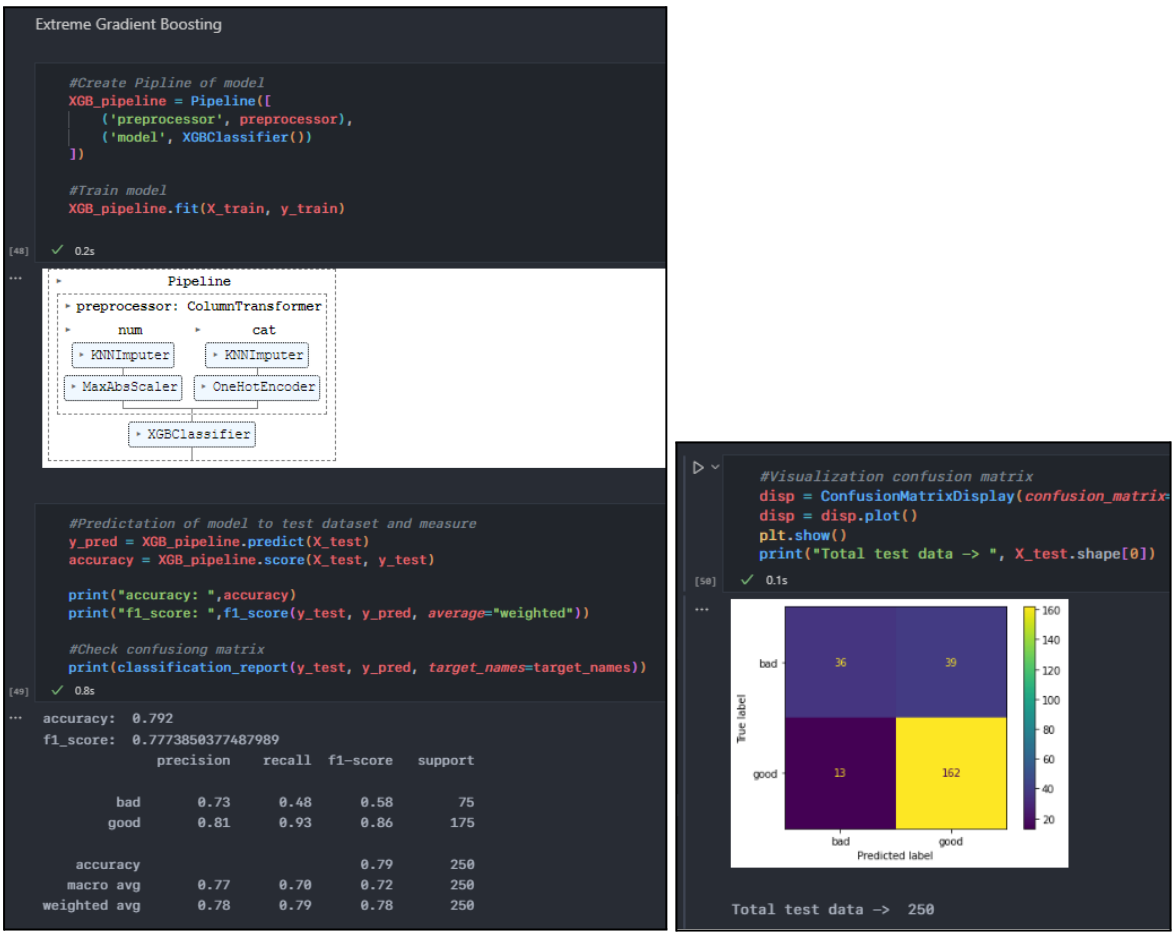
To interpret the prediction performance metrics for the confusion matrix,there are 36+46= 82 incorrect predictions, and 163+55=218 correct predictions for 300 testing data.

Extreme Gradient Boosting Classifier (developer:Frog)

A short Introduction Extreme Gradient Boosting Algorithm

Extreme Gradient Boosting is an efficient open-source implementation of the stochastic gradient boosting ensemble algorithm which is used for classification or regression predictive modelling problems. Therefore, ensembles are constructed from decision tree models which are added one at a time to the ensemble and fit to correct the prediction errors made by prior models. This is a type of ensemble machine learning model referred to as boosting. Models are fit using any arbitrary differentiable loss function and gradient descent optimization algorithm accordingly we call this technique “gradient boosting” as the loss gradient is minimised as the model is fit, much like a neural network. [4]

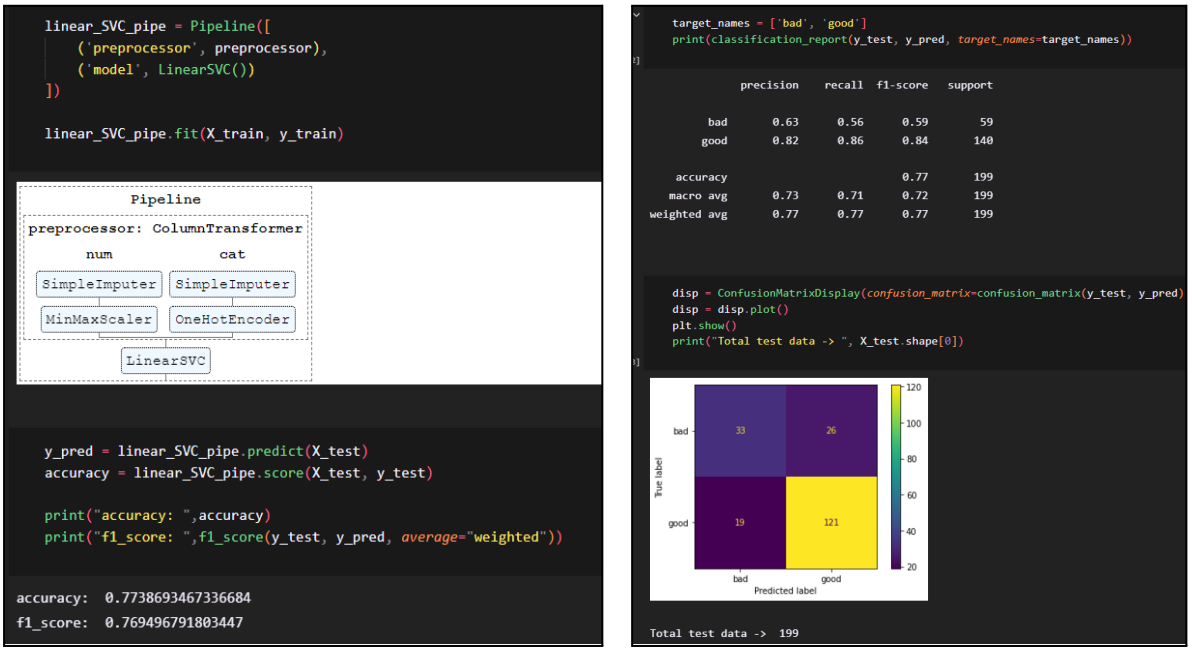
This figure is shown about the testing result of the training model



Support Vector Machine Classifier (developer: Punch)

Support vector machine classifier (SVC) is a classifier that classifies by drawing a line within data points to separate between the classes, which can be used as both linear and nonlinear based on kernels. This algorithm is chosen because of its ability to handle high dimension data and its simpliness, which can be considered as a baseline mode in later milestones. However, because of the imbalanced dataset could decrease the model's performance. [5, 6]

The figure below shows the training process and result of the model.

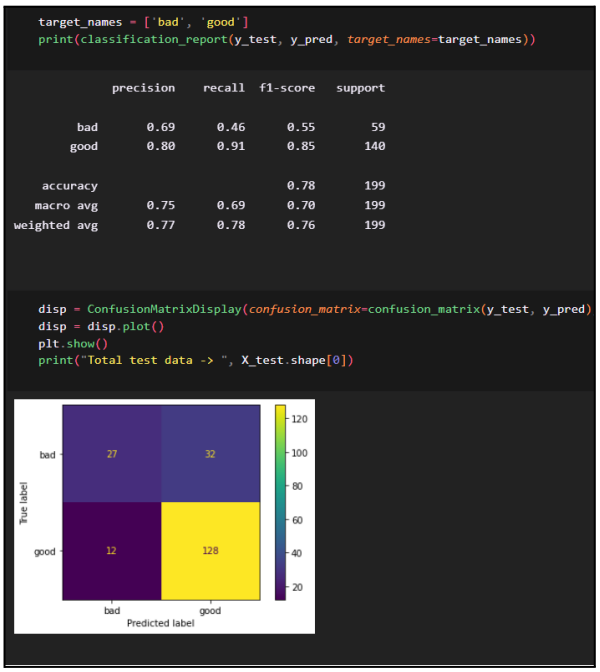


According to the figues, this model using simple preprocessing steps, i.e., simple impute, scale, and encode. The test set is 20% hold-out from the dataset before training and the rest is used as training set. The final result indicates that the model has accuracy and f1-score of 77% and 76% respectively. However, when considering precision and recall, for the 'bad' , the model obtains less value on both measures (63% and 56% respectively). This could be caused by the imbalance dataset which contains more 'good' instances. Moreover, the confusion matrix can be used to confirm the imbalance which presents more false positive (26) than false negative (19).

Random Forest Classifier (developer: Punch)

Random forest is an ensemble learning for classification problem working by constructing multiple decision trees and each individual tree has their own prediction which the most votes among all predictions becomes model prediction. Random forest has the advantage of performance when dealing with nonlinear problems, large amounts of data, missing data or outliers, and imbalanced dataset. Moreover, this algorithm not only can be used to extract feature importances from the dataset, but also provides less overfitting. In contrast, the model is less interpreted and the features used need to have some predictive power. [5, 6]

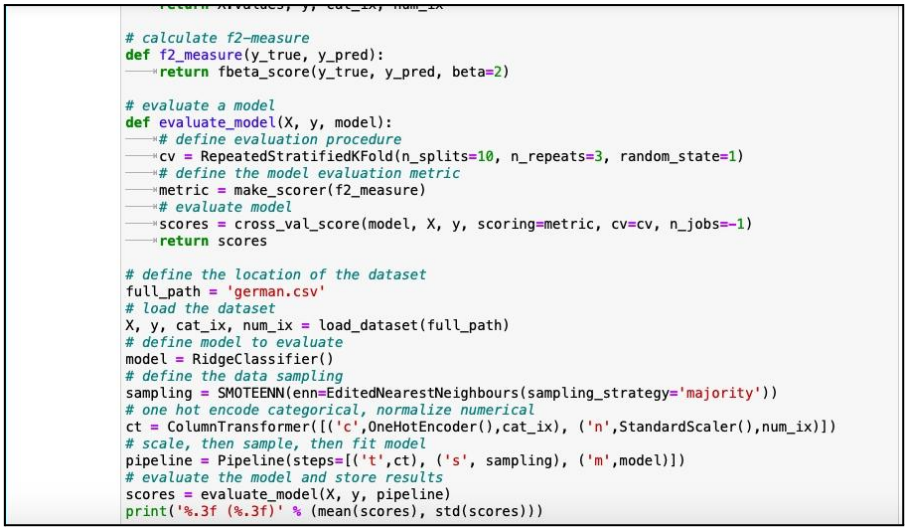
The figure below shows the training process and result of the model.



According to the figues, this model also used the simple preprocessing step as SVM, including training and testing data ratio (80% and 20%). The final result indicates that the model has accuracy and f1-score of 77% and 76% respectively. Moreover, the precision and recall are higher compared to SVC. In addition, this model presents less true negative, but more false positive which could be caused from an imbalanced dataset.

SMOTEENN with StandardScaler and RidgeClassifier (developer: Jincheng Zhang)

An F2-measure of about 0.741 can be achieved with further improvements to the SMOTEENN using a RidgeClassifier instead of LDA and using a StandardScaler for the numeric inputs instead of a MinMaxScaler.



```
# calculate f2-measure
def f2_measure(y_true, y_pred):
    return fbeta_score(y_true, y_pred, beta=2)

# evaluate a model
def evaluate_model(X, y, model):
    # define evaluation procedure
    cv = RepeatedStratifiedKFold(n_splits=10, n_repeats=3, random_state=1)
    # define the model evaluation metric
    metric = make_scorer(f2_measure)
    # evaluate model
    scores = cross_val_score(model, X, y, scoring=metric, cv=cv, n_jobs=-1)
    return scores

# define the location of the dataset
full_path = 'german.csv'
# load the dataset
X, y, cat_ix, num_ix = load_dataset(full_path)
# define model to evaluate
model = RidgeClassifier()
# define the data sampling
sampling = SMOTEENN(enn=EditedNearestNeighbours(sampling_strategy='majority'))
# one hot encode categorical, normalize numerical
ct = ColumnTransformer([('c',OneHotEncoder(),cat_ix), ('n',StandardScaler(),num_ix)])
# scale, then sample, then fit model
pipeline = Pipeline(steps=[('t',ct), ('s', sampling), ('m',model)])
# evaluate the model and store results
scores = evaluate_model(X, y, pipeline)
print('%.3f (%.3f)' % (mean(scores), std(scores)))
```

References:

1. https://www.kdnuggets.com/2020/06/naive-bayes-algorithm-everything.html
2. https://www.geeksforgeeks.org/naive-bayes-classifiers/
3. https://www.javatpoint.com/machine-learning-naive-bayes-classifier
4. Extreme Gradient Boosting (XGBoost) Ensemble in Python (machinelearningmastery.com)
5. Pros and cons of various Machine Learning algorithms | by Shailaja Gupta | Towards Data Science
6. Advantages and Disadvantages of different Classification Models - GeeksforGeeks