Algorithmics for Data Mining

Deliverable 4: Comparing the results of Stacking methods of two distinct datasets

Ali Arabyarmohammadi June 2022

Contents

List of Figures

List of Tables

1 Introduction

We can think of stacking as the process of combining various machine learning models. For ensemble models, there are several different techniques that can be used, including **stacking**, **boosting**, and **bagging**.
Because stacking focuses on examining the space of several models used to solve the same problem, it differs significantly from these ensemble methods.
The basic idea behind this approach is to solve a machine learning problem utilizing various model types that can learn to a certain amount rather than the entire problem field.
As we mentioned, an ensemble machine learning algorithm is called stacking or stacked generalization. It learns the most effective way to combine the predictions from two or more base machine learning models using a meta-learning method.
To clarify more and in order to compare **bagging** and **boosting** methods with **stacking**, we should mention that, first, stacking is often considered heterogeneous weak learners (various learning methods are mixed), whereas bagging and boosting assume mainly homogeneous weak learners. Second, bagging and boosting combine weak learners using deterministic algorithms, whereas stacking learns to combine the basic models using a meta-model.
A concept diagram of stacking (From Graham Harrison's website [1]) illustrated in Figure 1.

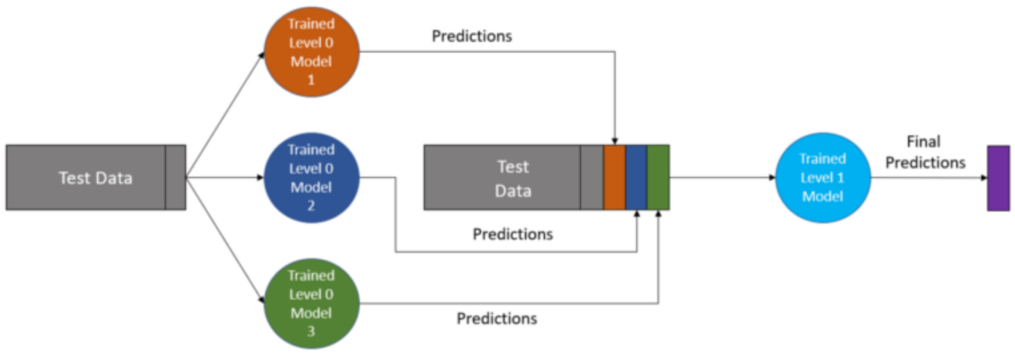


Figure 1: Concept diagram of stacking: Make predictions for the Testing/Validation of Data [1]

In this work, among common stacking-based ensemble learning methods, we train a learning algorithm to integrate the predictions of these basic classifications using decision tree (DT), K-Nearest Neighbor (k-NN), and Random Forest (RF) as the basic classifications.

- **k-NN(k-nearest neighbor)**
  The supervised machine learning technique known as the k-nearest neighbors (k-NN) can be used to tackle classification and regression issues. It is simple to use and comprehend, but it has the important problem of becoming noticeably slower as the amount of data in use increases.

- **RF(random forest)**
  Random forests, also known as random choice forests, are an ensemble learning method for classification, regression, and other tasks that works by building a large number of decision trees during training.

- **DT(decision tree)**
  In statistics, data mining, and machine learning, decision tree learning is a supervised learning approach. In this formalization, inferences about a set of data are made using a classification or regression decision tree as a predictive model.

In this work, we study two different datasets using mentioned methods:

- **Project#1: Diagnosis of diabetes using stacking data mining algorithms.**

  In modern healthcare systems, disease detection automation is pervasive. A serious issue that has gained worldwide traction is the diabetic condition. Different algorithms are utilized in the heterogeneous model. To determine if a person has diabetes positively or negatively, this study uses a heterogeneous ensemble model called the stacked ensemble model. The prediction is beneficial with this stacked ensemble model.

- **Project#2: Credit Scoring using stacking data mining algorithms.**

  Lenders and financial organizations use credit scoring, which is a statistical analysis, to determine a person's creditworthiness. Credit scores are used by lenders to determine whether to grant or deny credit. An individual's credit score ranges from 300 to 850, with 850 representing the best possible rating. Credit card applications, mortgage approvals, auto loans, and personal loans are all influenced by credit scores.
  Usually, some factors—payment history, categories of credit previously obtained, new credit, outstanding debt, and length of credit held—have an impact on a credit score. The lender must pay close attention to the borrower's payment history and current debt. In this project, we also use a stacked ensemble model.

After each project is done and analyzed, we will compare their results together.
Although comparing the results between these two projects may not lead to helpful conclusions, because we use the same techniques in both projects, it is ultimately possible to display and compare the output parameters.

In order to implement these mentioned data mining methods, python was used, and the also libraries **Sklearn** and **pandas** libraries were used in both projects. The source code file for each project is located in a separate folder, and the tables and charts are located in the related subfolder.

```python
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.preprocessing import StandardScaler
from sklearn.pipeline import make_pipeline
from sklearn.ensemble import StackingClassifier
from sklearn.neighbors import KNeighborsClassifier
import matplotlib.pyplot as plt
from sklearn.metrics import plot_confusion_matrix
from sklearn.metrics import classification_report
```

Listing 1: Python libraries were used

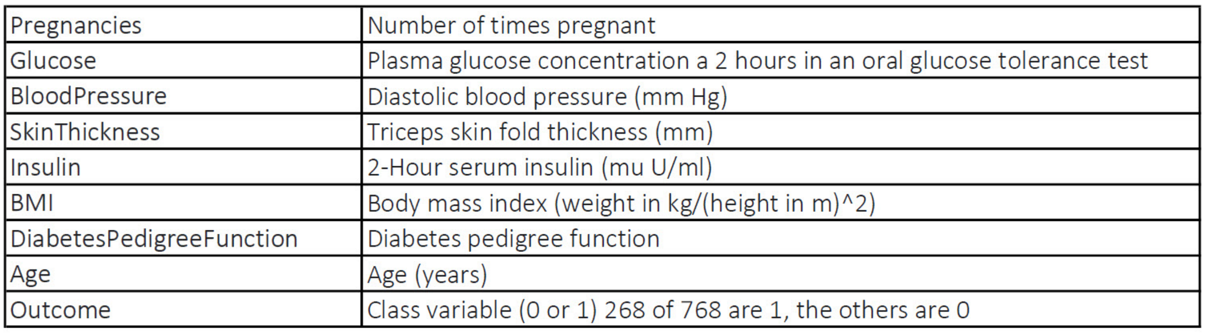
2 Diagnosis of diabetes using stacking data mining algorithms.

2.1 Data Understanding

The National Institute of Diabetes and Digestive and Kidney Diseases is the original source of this dataset. Using **pandas** library, the dataset file **diabetes.csv** was loaded. The database is used adopted from the website **kaggle** and the web page **https://www.kaggle.com/datasets/uciml/pima-indians-diabetes-database** which Predicts the onset of diabetes based on diagnostic measures.

Based on specific diagnostic parameters present in the dataset, the dataset's goal is to diagnostically predict whether a patient has diabetes or not. The selection of these examples from a bigger database was subject to a number of restrictions. Particularly, all patients at this facility are Pima Indian women who are at least 21 years old.
Using these two lines of code below, we are reading the datasets.

Table 1: Description of Pima Indians Diabetes Database



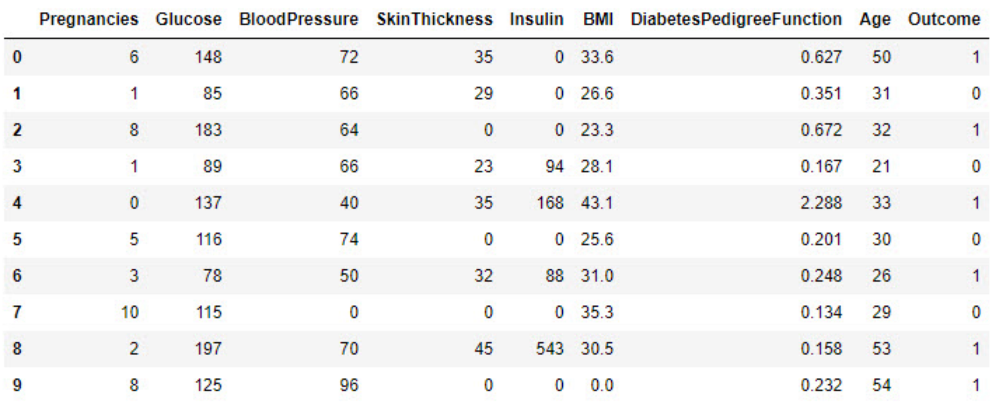
Pregnancies	Number of times pregnant
Glucose	Plasma glucose concentration a 2 hours in an oral glucose tolerance test
BloodPressure	Diastolic blood pressure (mm Hg)
SkinThickness	Triceps skin fold thickness (mm)
Insulin	2-Hour serum insulin (mu U/ml)
BMI	Body mass index (weight in kg/(height in m)^2)
DiabetesPedigreeFunction	Diabetes pedigree function
Age	Age (years)
Outcome	Class variable (0 or 1) 268 of 768 are 1, the others are 0

```
df = pd.read_csv("diabetes.csv")
df.head (10)
```

Listing 2: Loading the datasets

The top ten rows of the tables are shown below.

Table 2: Training dataset (Diabetes)



	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
0	6	148	72	35	0	33.6	0.627	50	1
1	1	85	66	29	0	26.6	0.351	31	0
2	8	183	64	0	0	23.3	0.672	32	1
3	1	89	66	23	94	28.1	0.167	21	0
4	0	137	40	35	168	43.1	2.288	33	1
5	5	116	74	0	0	25.6	0.201	30	0
6	3	78	50	32	88	31.0	0.248	26	1
7	10	115	0	0	0	35.3	0.134	29	0
8	2	197	70	45	543	30.5	0.158	53	1
9	8	125	96	0	0	0.0	0.232	54	1

The datasets consist of one target variable, Outcome, and a number of medical predictor variables. The patient's BMI, insulin level, age, number of previous pregnancies, and other factors are predictor variables.

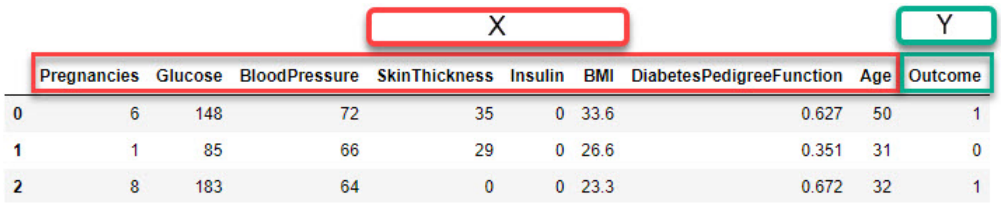
2.2 Data Preparation

First, among the factors in the data set, we assign the "outcome" value to the variable Y and the rest of the factors to the variable X.

```
X = df.iloc[:,:-1]
y = df.iloc[:,-1]
```

Listing 3: Assigning the factors to the variables X and Y

Table 3: Assigning the factors to the variables X and Y (Diabetes)



	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
0	6	148	72	35	0	33.6	0.627	50	1
1	1	85	66	29	0	26.6	0.351	31	0
2	8	183	64	0	0	23.3	0.672	32	1

Here we will split the data into the Test and Train. We will assign 33 percent to the Test and the rest to the train data.

```
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.33, random_state=42)
```

Listing 4: Splitting data into the Test and Train

The following code is an auxiliary function that helps to show the result. It will show the "accuracy," and then we have the "confusion matrix" and then a comprehensive report using the function "classification_report."

```
def showResult(clf, X_train, y_train, X_test, y_test):
    clf.fit(X_train, y_train)
    plot_confusion_matrix(clf, X_test, y_test)
    plt.show()
    y_pred = clf.predict(X_test)
    print(classification_report(y_test, y_pred))
```

Listing 5: An auxiliary function to show the results

2.3 Modeling

To implement our stacking method, we combined k-NN, DT, and RF in the code below. We considered the following combinations:

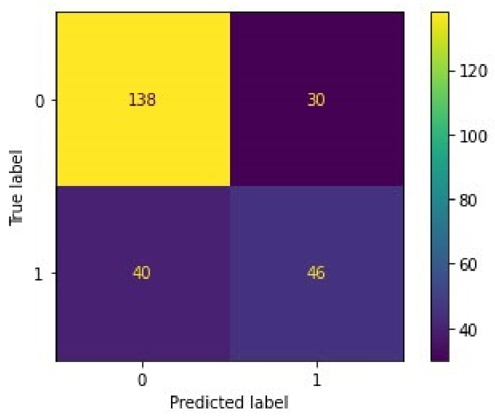
- **k-NN - DT - RF**
- **k-NN - RF - DT**
- **DT - RF - k-NN**

2.4 k-NN - DT - RF combination

```
estimators = [
    ('knn', KNeighborsClassifier(n_neighbors=3)),
    ('dt', DecisionTreeClassifier(random_state=0))
]
clf = StackingClassifier(
    estimators=estimators, final_estimator= RandomForestClassifier(n_estimators
   =10, random_state=42)
)
showResult(clf, X_train, y_train, X_test, y_test)
```

Listing 6: Combining kNN-DT-RF

The confusion matrix and the results (Parameters *precision*, *recall*, *f1-score*, and *support*) for the diabetes dataset and for classes 0 and 1 are shown below.



```
              precision    recall  f1-score   support

           0       0.78      0.82      0.80       168
           1       0.61      0.53      0.57        86

    accuracy                           0.72       254
   macro avg       0.69      0.68      0.68       254
weighted avg       0.72      0.72      0.72       254
```

Figure 2: Confusion matrix and the results for (kNN - DT - RF) (Diabetes)

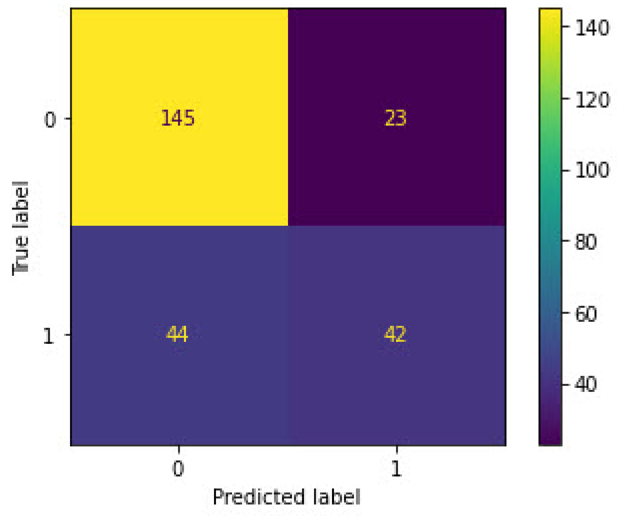
From the confusion matrix above and for the combination kNN-DT-RF (Figure 2), we can see that 138 items labeled 0 are accurately predicted (yellow square on the top left). The same for label 1; the predicted number is correctly reported (46 items - blue square at bottom right). The main diagonal is the values that are predicted accurately, and the other diagonal, which contains the numbers 30 and 40, are the records that are not predicted correctly.
The parameter "accuracy" for this combination is 72%.

2.5 k-NN - RF - DT combination

```python
estimators = [
    ('knn', KNeighborsClassifier(n_neighbors=3)),
    ('rf', RandomForestClassifier(n_estimators=10, random_state=42))
]
clf = StackingClassifier(
    estimators=estimators, final_estimator= DecisionTreeClassifier(random_state=0)
)

showResult(clf, X_train, y_train, X_test, y_test)
```

Listing 7: Combining kNN-RF-DT



```
              precision    recall  f1-score   support

           0       0.77      0.86      0.81       168
           1       0.65      0.49      0.56        86

    accuracy                           0.74       254
   macro avg       0.71      0.68      0.68       254
weighted avg       0.73      0.74      0.73       254
```

Figure 3: Confusion matrix and the results for (kNN - RF - DT) (Diabetes)

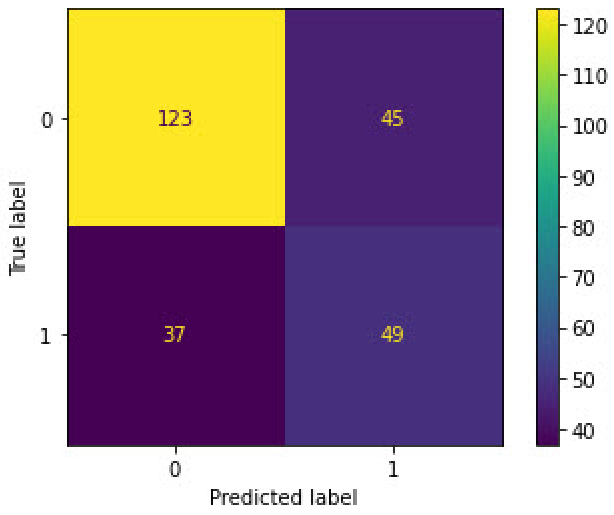
From the confusion matrix above and for the combination kNN-RF-DT (Figure 3), we can see that 145 items labeled 0 are accurately predicted (yellow square on the top left). The same for label 1; the predicted number is correctly reported (42 items - blue square at bottom right). The main diagonal is the values that are predicted accurately, and the other diagonal, which contains the numbers 23 and 44, are the records that are not predicted correctly.
The parameter "accuracy" for this combination is 74%.

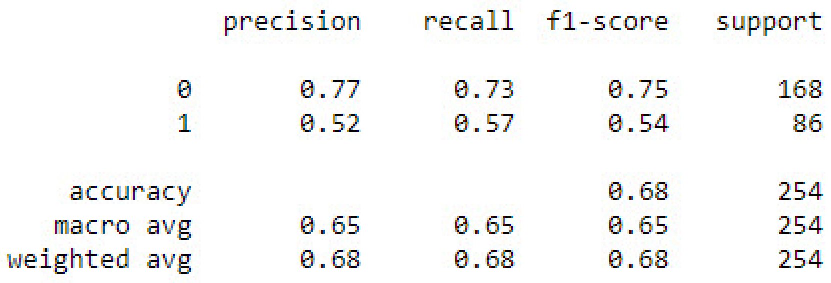
2.6 DT - RF - k-NN combination

```
estimators = [
    ('dt', DecisionTreeClassifier(random_state=0)),
    ('rf', RandomForestClassifier(n_estimators=10, random_state=42))
]
clf = StackingClassifier(
    estimators=estimators, final_estimator= KNeighborsClassifier(n_neighbors=3)
)

showResult(clf, X_train, y_train, X_test, y_test)
```

Listing 8: Combining DT-RF-kNN





```
              precision    recall  f1-score   support

           0       0.77      0.73      0.75       168
           1       0.52      0.57      0.54        86

    accuracy                           0.68       254
   macro avg       0.65      0.65      0.65       254
weighted avg       0.68      0.68      0.68       254
```

Figure 4: Confusion matrix and the results for (DT - RF - kNN) (Diabetes)

From the confusion matrix above and for the combination DT-RF-kNN (Figure 4), we can see that 123 items labeled 0 are accurately predicted (yellow square on the top left). The same for label 1; the predicted number is correctly reported (49 items - blue square at bottom right). The main diagonal is the values that are predicted accurately, and the other diagonal, which contains the numbers 45 and 37, are the records that are not predicted correctly.
The parameter "accuracy" for this combination is 68%.

We can see from the results that the combination **kNN-RF-DT** shows the best value in terms of "accuracy" with 74%.

3 Credit Scoring using stacking data mining algorithms.

3.1 Data Understanding

Center for Machine Learning and Intelligent Systems is the original source of this dataset. Using **pandas** library, the dataset file **german.csv** (Statlog (German Credit Data) Data Set) was loaded. The database is used adopted from the website **UCI - Machine learning repository** and the web page **https://archive.ics.uci.edu/ml/datasets/statlog+(german+credit+data)** which classifies people described by a set of attributes as good or bad credit risks.
This dataset is also can be found on the website "Kaggle" and the link:
https://www.kaggle.com/datasets/uciml/german-credit)

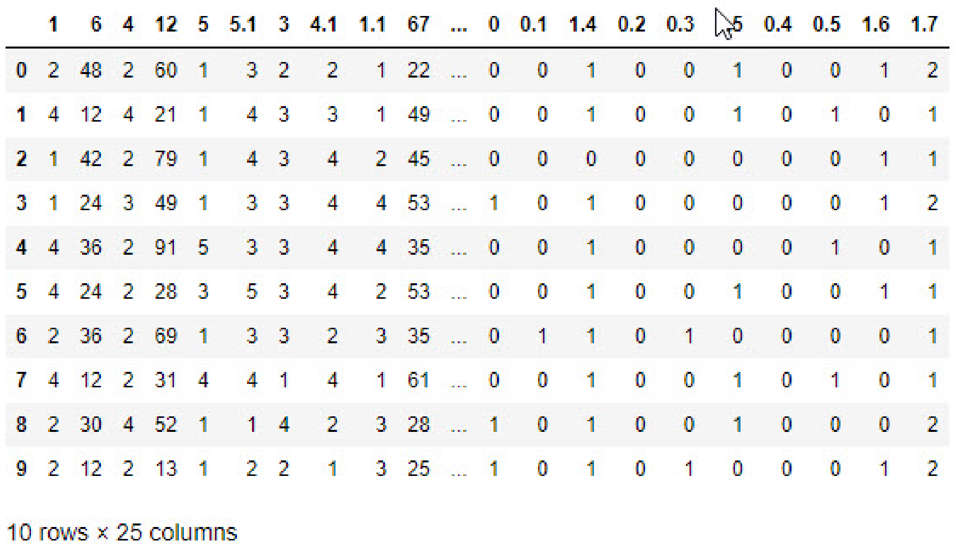
The original dataset contains 1000 entries with 20 categorial/symbolic attributes prepared by Prof. Hofmann. Each entry in this collection reflects a person who accepts a bank credit. Depending on a set of factors, each person is categorized as either a good or bad credit risk.

```
df = pd.read_csv("german.csv")
df.head (10)
```

Listing 9: Loading the datasets

The top ten rows of the tables are shown below.

Table 4: Training dataset (German Credit Data)



	1	6	4	12	5	5.1	3	4.1	1.1	67	...	0	0.1	1.4	0.2	0.3	5	0.4	0.5	1.6	1.7
0	2	48	2	60	1	3	2	2	1	22	...	0	0	1	0	0	1	0	0	1	2
1	4	12	4	21	1	4	3	3	1	49	...	0	0	1	0	0	1	0	1	0	1
2	1	42	2	79	1	4	3	4	2	45	...	0	0	0	0	0	0	0	0	1	1
3	1	24	3	49	1	3	3	4	4	53	...	1	0	1	0	0	0	0	0	1	2
4	4	36	2	91	5	3	3	4	4	35	...	0	0	1	0	0	0	0	1	0	1
5	4	24	2	28	3	5	3	4	2	53	...	0	0	1	0	0	1	0	0	1	1
6	2	36	2	69	1	3	3	2	3	35	...	0	1	1	0	1	0	0	0	0	1
7	4	12	2	31	4	4	1	4	1	61	...	0	0	1	0	0	1	0	1	0	1
8	2	30	4	52	1	1	4	2	3	28	...	1	0	1	0	0	1	0	0	0	2
9	2	12	2	13	1	2	2	1	3	25	...	1	0	1	0	1	0	0	0	1	2

10 rows × 25 columns

Attribute description for German Credit Data:

It is almost impossible to understand the original dataset due to its complicated system of categories and symbols. Thus, I wrote a small Python script to convert it into a readable CSV file. Several columns are simply ignored, because in my opinion either they are not important or their descriptions are obscure. The selected attributes are:

- **Age** (numeric)
- **Sex** (text: male, female)

- **Job** (numeric: 0 - unskilled and non-resident, 1 - unskilled and resident, 2 - skilled, 3 - highly skilled)
- **Housing** (text: own, rent, or free)
- **Saving accounts** (text - little, moderate, quite rich, rich)
- **Checking account** (numeric, in DM - Deutsch Mark)
- **Credit amount** (numeric, in DM)
- **Duration** (numeric, in month)
- **Purpose** (text: car, furniture/equipment, radio/TV, domestic appliances, repairs, education, business, vacation/others)

3.2 Data Preparation

First, among the factors in the data set, we assign the last column's value to the variable Y and the rest of the factors to the variable X.

```
X = df.iloc[:,:-1]
y = df.iloc[:,-1]

y= y -1
y.head()
```

Listing 10: Assigning the factors to the variables X abd Y

Because the last column in the dataset is in **1 and 2**. We converted then to the value **0 and 1**, like what we had in diabetes dataset.

```
0 1
1 0
2 0
3 1
4 0
Name: 1.7, dtype: int64
```

Here we will split the data into the Test and Train. We will assign 33 percent to the Test and the rest to the train data.

```
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.33, random_state=42)
```

Listing 11: Splitting data into the Test and Train

The following code is an auxiliary function that helps to show the result. It will show the "accuracy," and then we have the "confusion matrix" and then a comprehensive report using the function "classification_report."

```
def showResult(clf, X_train, y_train, X_test, y_test):
    clf.fit(X_train, y_train)
    plot_confusion_matrix(clf, X_test, y_test)
    plt.show()
    y_pred = clf.predict(X_test)
    print(classification_report(y_test, y_pred))
```

Listing 12: An auxiliary function to show the results

3.3 Modeling

To implement our stacking method, we combined k-NN, DT, and RF in the code below. We considered the following combinations:

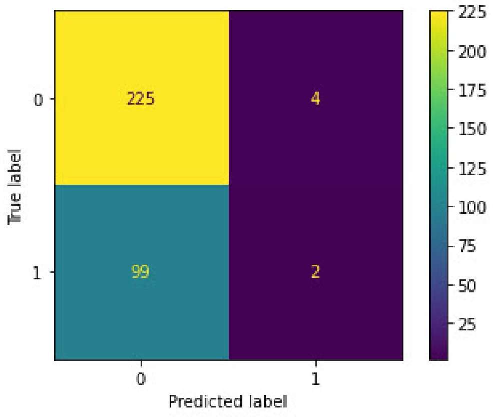
- **k-NN - DT - RF**
- **k-NN - RF - DT**
- **DT - RF - k-NN**

3.4 k-NN - DT - RF combination

```
estimators = [
    ('knn', KNeighborsClassifier(n_neighbors=3)),
    ('dt', DecisionTreeClassifier(random_state=0))
]
clf = StackingClassifier(
    estimators=estimators, final_estimator= RandomForestClassifier(n_estimators
   =10, random_state=42)
)
showResult(clf, X_train, y_train, X_test, y_test)
```

Listing 13: Combining kNN-DT-RF

The confusion matrix and the results (Parameters *precision*, *recall*, *f1-score*, and *support*) for the german dataset and for classes 0 and 1 are shown below.



```
              precision    recall  f1-score   support

           0       0.69      0.98      0.81       229
           1       0.33      0.02      0.04       101

    accuracy                           0.69       330
   macro avg       0.51      0.50      0.43       330
weighted avg       0.58      0.69      0.58       330
```

Figure 5: Confusion matrix and the results for (kNN - DT - RF) (german)

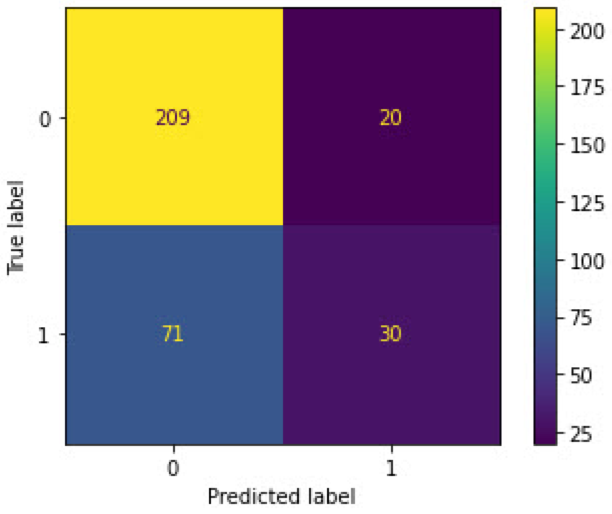
From the confusion matrix above and for the combination kNN-DT-RF (Figure 5), we can see that 225 items labeled 0 are accurately predicted (yellow square on the top left). The same for label 1; the predicted number is correctly reported (2 items - blue square at bottom right). The main diagonal is the values that are predicted accurately, and the other diagonal, which contains the numbers 4 and 99, are the records that are not predicted correctly.
The parameter "accuracy" for this combination is 69%.

3.5 k-NN - RF - DT combination

```python
estimators = [
    ('knn', KNeighborsClassifier(n_neighbors=3)),
    ('rf', RandomForestClassifier(n_estimators=10, random_state=42))
]
clf = StackingClassifier(
    estimators=estimators, final_estimator= DecisionTreeClassifier(random_state=0)
)

showResult(clf, X_train, y_train, X_test, y_test)
```

Listing 14: Combining kNN-RF-DT



```
              precision    recall  f1-score   support

           0       0.75      0.91      0.82       229
           1       0.60      0.30      0.40       101

    accuracy                           0.72       330
   macro avg       0.67      0.60      0.61       330
weighted avg       0.70      0.72      0.69       330
```

Figure 6: Confusion matrix and the results for (kNN - RF - DT) (german)

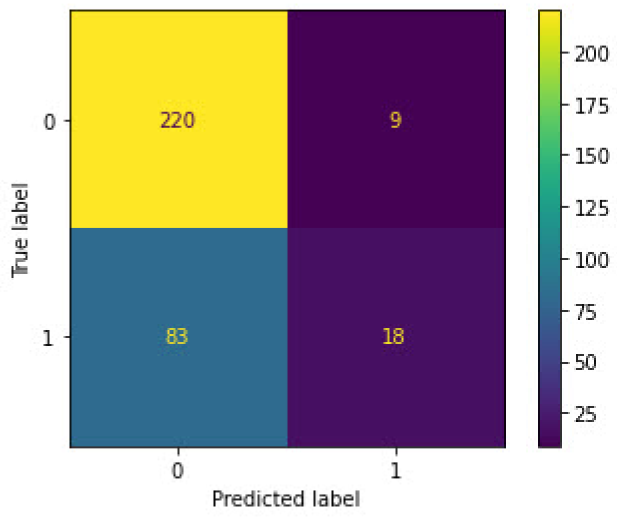
From the confusion matrix above and for the combination kNN-RF-DT (Figure 6), we can see that 209 items labeled 0 are accurately predicted (yellow square on the top left). The same for label 1; the predicted number is correctly reported (30 items - blue square at bottom right). The main diagonal is the values that are predicted accurately, and the other diagonal, which contains the numbers 20 and 71, are the records that are not predicted correctly.
The parameter "accuracy" for this combination is 72%.

3.6 DT - RF - k-NN combination

```
estimators = [
    ('dt', DecisionTreeClassifier(random_state=0)),
    ('rf', RandomForestClassifier(n_estimators=10, random_state=42))
]
clf = StackingClassifier(
    estimators=estimators, final_estimator= KNeighborsClassifier(n_neighbors=3)
)

showResult(clf, X_train, y_train, X_test, y_test)
```

Listing 15: Combining DT-RF-kNN



```
              precision    recall  f1-score   support

           0       0.73      0.96      0.83       229
           1       0.67      0.18      0.28       101

    accuracy                           0.72       330
   macro avg       0.70      0.57      0.55       330
weighted avg       0.71      0.72      0.66       330
```

Figure 7: Confusion matrix and the results for (DT - RF - kNN) (german)

From the confusion matrix above and for the combination DT-RF-kNN (Figure 7), we can see that 220 items labeled 0 are accurately predicted (yellow square on the top left). The same for label 1; the predicted number is correctly reported (18 items - blue square at bottom right). The main diagonal is the values that are predicted accurately, and the other diagonal, which contains the numbers 9 and 83, are the records that are not predicted correctly.
The parameter "accuracy" for this combination is 72%.

We can see from the results that the combination **kNN-RF-DT** and **DT-RF-kNN** both show 74% accuracy, and both combinations are acceptable and can be chosen.

4 Comparison of results

In performing the stacking method, we used three combinations: *kNN-DT-RF*, *kNN-RF-DT*, and *DT-RF-kNN*.
Figure 8 and Table 5 show the values of the *precision, recall, f1-score, and support* parameters for the **diabetes** project, and Figure 9 and Table 6 show the same parameters for the **credit scoring** project.

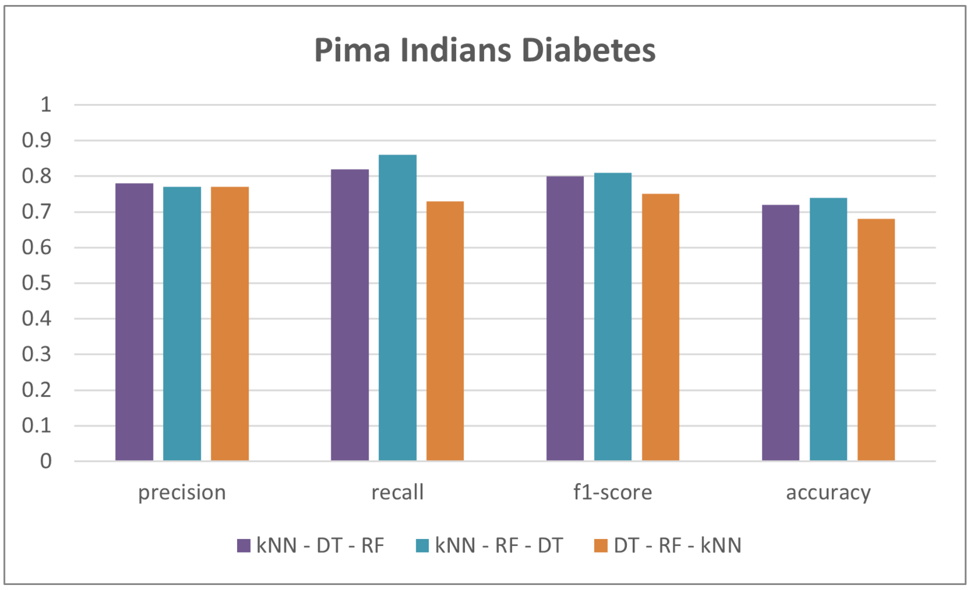


Figure 8: Comparison of results for three combinations (Diabetes)

Table 5: Comparison of results for three combinations (Diabetes)

Pima Indians Diabetes				
	precision	**recall**	**f1-score**	**accuracy**
kNN - DT - RF	0.78	0.82	0.8	0.72
kNN - RF - DT	0.77	0.86	0.81	0.74
DT - RF - kNN	0.77	0.73	0.75	0.68

Interestingly, the results for both diabetes and credit scoring projects are very close. In the diabetes project, the best accuracy is related to the kNN-RF-DT combination with 74 percent, and for the Credit scoring project, the two combinations DT-RF-kNN and kNN-RF-DT have the same accuracy (74 percent).
It can be seen that the maximum accuracy obtained is the same for both projects.
For the diabetes project, the precision values are very close to each other and have the least variation between the three selected combinations. These values are more different for the Credit scoring project.
For the diabetes project, the lowest precision is for the DT-RF-kNN combination with 68 percent, and for the Credit scoring project, the lowest is for the kNN-DT-RF combination with 69 percent.

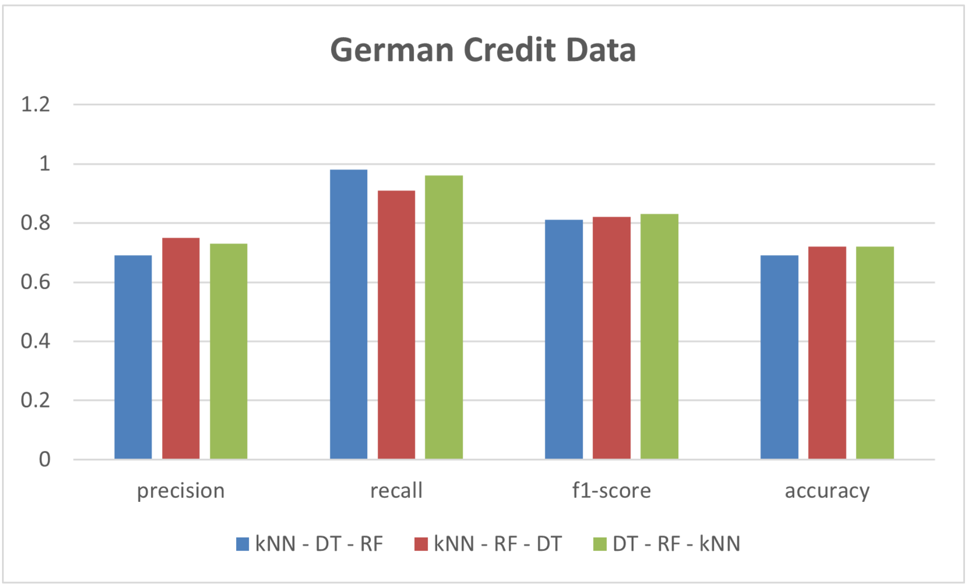


Figure 9: Comparison of results for three combinations (German Credit Data)

Table 6: Comparison of results for three combinations (German Credit Data)

German Credit Data				
	precision	**recall**	**f1-score**	**accuracy**
kNN - DT - RF	0.69	0.98	0.81	0.69
kNN - RF - DT	0.75	0.91	0.82	0.72
DT - RF - kNN	0.73	0.96	0.83	0.72

By comparing the available details and closely observing the numbers obtained for all three combinations, it is possible to ensure the accuracy of the stacking method for both projects.
The results show the efficiency of the stacking method for both projects. In this work, only three methods, kNN, DT, and RF, have been used. For future work and future study, it is recommended to use other methods such as logistic regression Naïve Bayes, and Latent Dirichlet Allocation (LDA). Of course, increasing the number of methods eventually leads to the production of high number of combinations, which of course, is possible with more programming works and automated procedure of comparison.

References

[1] Graham Harrison, *A Deep Dive into Stacking Ensemble Machine Learning — Part I* `https://towardsdatascience.com/a-deep-dive-into-stacking-ensemble-machine-learning-part-i-10476b2ade3.`

[2] Zhongyuan Han, Jiaming Gao, Huilin Sun, Ruifeng Liu, *An Ensemble Learning-based model for classification of Insincere Question* Forum for Information Retrieval Evaluation FIRE 2019 - Computer Science, 2019.

[3] Yiheng Li, Weidong Chen, *A Comparative Performance Assessment of Ensemble Learning for Credit Scoring* Mathematical Analysis in Economics and Management, September 2020.

[4] Sivashankari R, Sudha M, Mohammad Kamrul Hasan, Rashid A. Saeed, Suliman A. Alsuhibany and Sayed Abdel-Khalek, *An Empirical Model to Predict the Diabetic Positive Using Stacked Ensemble Approach* , Frontiers in Public Health, January 2022.

[5] Mohammed Gollapalli, Aisha Alansari, Heba Alkhorasani, Meelaf Alsubaii, Rasha Sakloua, Reem Alzahrani, Mohammed Taha Al-Hariri, Maiadah Nasser Alfares, Dania AlKhafaji, Reem Jaafar Al Argan, Waleed Albaker, *A novel stacking ensemble for detecting three types of diabetes mellitus using a Saudi Arabian dataset* , Computers in Biology and Medicine, June 2022.