FIT1043 Assignment 2-34550720

A1. Data Wrangling

A1.1:

```
import pandas as pd

student_data = pd.read_csv('Student_List_A2.csv')

print("Columns name: ", student_data.columns)
```

```
Columns name:  Index(['StudentID', 'Age', 'StudyTimeWeekly', 'Absences', 'ParentalSupport',
       'GPA', 'GradeClass'],
      dtype='object')
```

A1.2:

```
import pandas as pd

file = 'Student_List_A2.csv'

data = pd.read_csv('Student_List_A2.csv')

grade_mapping = {0: 'A', 1: 'B', 2: 'C', 3: 'D', 4: 'F'}   #Create dictionary to change letter from number

data['GradeClass'] = data['GradeClass'].map(grade_mapping).astype(str)  #use map function change 'gradeclass'column

data.to_csv(file, index=False)
```

A1.3:

```
import pandas as pd       #cannot be run alone. need to run the code of A1.2 before running it

file = 'Student_List_A2.csv'

data = pd.read_csv('Student_List_A2.csv')

grade_start = {'A': 0, 'B': 1, 'C': 2, 'D': 3, 'F': 4}     #change gradeclass column to number again

data['GradeClass'] = data['GradeClass'].map(grade_start)   #if this column doesn't change back to the number then wi

column = data.select_dtypes(include=[np.number]).columns   #use the select_dtypes to filter out columns and store in

data[numeric_cols] = data[column].fillna(data[column].median())  #use median to fill in

grade_mapping = {0: 'A', 1: 'B', 2: 'C', 3: 'D', 4: 'F'}   #change back to letter

data['GradeClass'] = data['GradeClass'].map(grade_mapping).astype(str)

data.to_csv(file, index=False)
```

(PS: This code does not run alone. need to finish running A1.2. Need run one by one.)

Filling with a median:

Robustness: The median is not sensitive to extreme values and better reflects the central trend of the data.

Applicability: Applies to numeric data, especially when the data has outliers.

A1.4:

```
import pandas as pd

file = 'Student_List_A2.csv'

data = pd.read_csv('Student_List_A2.csv')

data = data[data['Absences'] <= 50]     #give a range so that the data is not outrageous

data.to_csv(file, index=False)
```

Remove outliers:

Improve model performance: Outliers can interfere with machine learning models, resulting in overfitting or increased bias.

Data consistency: Removing outliers can make data more consistent and reduce analysis biases caused by abnormal data.

Enhanced data quality: Outliers are usually errors or anomalies in the data collection process and removing them can improve the reliability of the data.

A1.5:

```
import pandas as pd

file = 'Student_List_A2.csv'

data = pd.read_csv('Student_List_A2.csv')

def choose_data(row):       #set a function that uses if-loop to select data that does not satisfy the condition
    if row['GradeClass'] == 'A' and row['GPA'] >= 3.5:
        return True
    elif row['GradeClass'] == 'B' and 3.0 <= row['GPA'] < 3.5:
        return True
    elif row['GradeClass'] == 'C' and 2.5 <= row['GPA'] < 3.0:
        return True
    elif row['GradeClass'] == 'D' and 2.0 <= row['GPA'] < 2.5:
        return True
    elif row['GradeClass'] == 'F' and row['GPA'] < 2.0:
        return True
    else:
        return False

new_data = data[data.apply(choose_data, axis=1)]  #use apply to apply the selection function to each row of data

new_data.to_csv(file, index=False)
```

Filter out all rows that do not meet the criteria. This means that rows whose GPA does not match GradeClass will be deleted. For example, if A row has a GPA of 3.8 but GradeClass is 'B', the row is deleted because it does not meet the condition that GPA >= 3.5 should correspond to 'A'.

A2. Supervised Learning

A2.1:

Supervised machine learning

Supervised learning concept Supervised learning is a basic type of machine learning where the goal is to learn models from labeled data to make predictions about new data. Supervised learning has the following key elements:

Features: Input data used for prediction. It is usually expressed in matrix form, with each row representing a sample and each column representing a feature.

Labels: Indicates the class or target value to which the sample belongs. Supervised learning models make predictions by learning the relationship between input features and labels.

Training Set: A data set of known features and labels for the training of the model.

Test Set: A data set that is used to verify the performance of a model and contains known labels but is not used in training.

Partitioning of data sets

In supervised learning, dividing a data set into a training set and a test set is a very important step. A common division is to use 80% of the data for training and 20% for testing.

Training Set: Used to train a model that learns the mapping between features and labels in the training set.

Test Set: Used to evaluate the generalization ability of the model, the test set is not involved in model training, and is used to verify the performance of the model on unknown data.

Common algorithms for supervised learning

Classification: used to predict discrete class labels, such as support vector machines (SVMS), decision trees, random forests, etc.

Regression: Used to predict continuous numerical labels, such as linear regression, decision tree regression, etc.

Feature selection and data preparation

In supervised learning, it is very important to select the right features. Feature selection can affect the accuracy and training speed of the model. In the data preparation stage, it is necessary to remove irrelevant features, process missing values, and encode features.

A2.2 & A2.3:

```
import pandas as pd
from sklearn.model_selection import train_test_split

data = pd.read_csv('Student_List_A2.csv')

X = data.drop(columns=['StudentID', 'GradeClass', 'GPA'])    #delete useless columns
Y = data['GradeClass']      #extract the target variable Y

X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.2, random_state=42)
# use the train_test_split function to divide the data set into training and test sets

scaler = StandardScaler()    #data standardization
X_train_scaled = scaler.fit_transform(X_train)   #the mean and standard deviation of the training set are calculated
X_test_scaled = scaler.transform(X_test)         #standardize the test set
```

X: X is the feature of the data set. With the drop method, we delete the columns ['StudentID', 'GradeClass', 'GPA'], leaving the remaining columns as features.

- A StudentID is a unique identifier for a student that has no practical meaning for the classification task and is usually removed.
- GradeClass is the target variable, which is what we're predicting, so it won't be a feature either.
- GPA represents the Grade Point Average of students. According to the previous task description, this column has been used to generate GradeClass, so the two columns are repeated (GradeClass is generated according to GPA) and do not need to be used as a feature.

Y: Y is the target variable, also called the label, which is what we want the model to predict. The GradeClass column is selected, which represents the student's grade classification (such as A, B, C, etc.).

test size=0.2: indicates that 20% of the samples in the data set are divided into test sets, and the remaining 80% are divided into training sets.

A3. Classification (training)

A3.1.a:

Data Normalization and Standardization:

Scaling Data Standardization refers to the Data, make it have the same scale, to better deal with machine learning model. Normalization (also called feature scaling) is crucial when working with machine learning algorithms that are sensitive to the scale of the input data. The common methods of Standardization are:

Z-score standardization: The mean of a feature is reduced to zero, and the standard

deviation is normalized. It is suitable for most machine learning algorithms, such as SVM and linear regression.

A3.1.b:

In sklearn.preprocessing, the most common function used for normalization is StandardScaler(). This scaler standardizes features by removing the mean and scaling to unit variance. This ensures that all features are treated equally by the model.

A3.2.a:

SVM is a supervised learning model for classification and regression, often used for classification tasks. The core idea is to find a hyperplane (or decision boundary) that maximizes the Margin between two types of data points. It looks for the optimal hyperplane in a high-dimensional space to classify data points as clearly as possible. SVM include:

Hyperplanes: decision boundaries used to separate different classes of data.

Support Vectors: The sample points closest to the hyperplane that determine the direction and position of the hyperplane.

Kernel Function: When the data cannot be segmented linearly, SVM maps the data to the high-dimensional space through the kernel function, so that the data can be segmented linearly in the high-dimensional space. Common kernel functions are linear kernel, polynomial kernel, and RBF kernel.

A3.2.b:

Kernel Function: When the data cannot be segmented linearly, SVM maps the data to the high-dimensional space through the kernel function, so that the data can be segmented linearly in the high-dimensional space. Common kernel functions are linear kernel, polynomial kernel, and RBF kernel.

Linear Kernel: Used when the data is linearly separable.

Polynomial Kernel: Useful when the data is polynomially separable.

Radial Basis Function (RBF) Kernel: A popular kernel for non-linear classification tasks. It maps the data into infinite-dimensional space.

A3.2.c

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.svm import SVC
from sklearn.metrics import confusion_matrix, accuracy_score

data = pd.read_csv('Student_List_A2.csv')

X = data.drop(columns=['StudentID', 'GradeClass', 'GPA'])
Y = data['GradeClass']

X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.2, random_state=100)

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

svm_model = SVC(kernel='linear', random_state=42)     #choose the SVM model and choose the linear kernel
svm_model.fit(X_train_scaled, Y_train)

Y_predict = svm_model.predict(X_test_scaled)          #use the trained model to make predict on the test

matrix = confusion_matrix(Y_test, Y_predict, labels=['A', 'B', 'C', 'D', 'F'])   #calculate the confusion matrix and

print("Confusion matrix of 5x5:")
print(matrix)       #print matrix

accuracy = accuracy_score(Y_test, Y_predict)     #computational accuracy
print(f"Accuracy of SVM model: {accuracy:.2f}")
```

First, the students' data is read from the CSV file and then stored.

X is the feature and Y is the target variable.

The data set is then divided into a training set and a test set, with the test set being 20% as required.

After many tests and changes, I found that when random_state=100, the prediction accuracy is the highest. So, I use random_state=100.

The training set is standardized using StandardScaler, the mean and standard deviation are calculated and applied to the training set.

The same normalization is performed on the test set, but the mean and standard deviation are not recalculated.

Create a linear kernel support vector machine classifier. The SVM model was trained using the training set data.

A3.3

I used the decision tree in task A3.3.

A decision tree is a flowchart-like structure where internal nodes represent features, branches represent decision rules, and leaf nodes represent results. The model learns to

divide the data according to the most important features, at the nodes where the information gain is maximized.

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import classification_report, confusion_matrix, accuracy_score

file = 'Student_List_A2.csv'
data = pd.read_csv('Student_List_A2.csv')

X = data.drop(columns=['StudentID', 'GradeClass', 'GPA'])
Y = data['GradeClass']

X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.2, random_state=44)

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

DecisionTree = DecisionTreeClassifier(random_state=45)   #create a decision tree model

DecisionTree.fit(X_train_scaled, Y_train)       #using decision trees mod on a training set

Y_predict = dt_model.predict(X_test_scaled)     #use trained model to make predict on the test

print("Confusion matrix of 5x5:\n", confusion_matrix(Y_test, Y_predict))      #print matrix
print("Accuracy of decision tree model:", accuracy_score(Y_test, Y_predict))     #computational accuracy
```

The overall flow of the code is similar to SVM. However, I need to change the model used, replacing the SVM model with a decision tree model.

After my test, in the decision tree model, the prediction accuracy is relatively high when random_state=45, so I changed 100 to 45.

A4. Classification (prediction)

A4.1 & A4.2

```python
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.svm import SVC
from sklearn.metrics import confusion_matrix, accuracy_score

data = pd.read_csv('Student_List_A2.csv')

X = data.drop(columns=['StudentID', 'GradeClass', 'GPA'])
Y = data['GradeClass']

X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.2, random_state=100)

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

svm_model = SVC(kernel='linear', random_state=42)     #choose the SVM model and choose the linear kernel
svm_model.fit(X_train_scaled, Y_train)

Y_predict = svm_model.predict(X_test_scaled)          #use the trained model to make predict on the test

matrix = confusion_matrix(Y_test, Y_predict, labels=['A', 'B', 'C', 'D', 'F'])   #calculate the confusion matrix and

print("Confusion matrix of 5x5:")
print(matrix)       #print matrix

accuracy = accuracy_score(Y_test, Y_predict)     #computational accuracy
print(f"Accuracy of SVM model: {accuracy:.2f}")
```

```
Confusion matrix of 5x5:
[[  0  11   0   0   0]
 [  0  26  15   0   0]
 [  0  11  32  10   1]
 [  0   0  15  43  12]
 [  0   0   0  15 209]]
Accuracy of SVM model: 0.78
```

```python
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import classification_report, confusion_matrix, accuracy_score

file = 'Student_List_A2.csv'
data = pd.read_csv('Student_List_A2.csv')

X = data.drop(columns=['StudentID', 'GradeClass', 'GPA'])
Y = data['GradeClass']

X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.2, random_state=44)

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

DecisionTree = DecisionTreeClassifier(random_state=45)   #create a decision tree model

DecisionTree.fit(X_train_scaled, Y_train)       #using decision trees mod on a training set

Y_predict = dt_model.predict(X_test_scaled)     #use trained model to make predict on the test

print("Confusion matrix of 5x5:\n", confusion_matrix(Y_test, Y_predict))      #print matrix
print("Accuracy of decision tree model:", accuracy_score(Y_test, Y_predict))     #computational accuracy
```

```
Confusion matrix of 5x5:
 [[  7   4   0   0   0]
 [  6  10  14   0   0]
 [  1  20  25  14   0]
 [  0   1  20  38  17]
 [  0   0   1  14 208]]
Accuracy of decision tree model: 0.72
```

The first image shows the 5x5 matrix of the SVM model with an accuracy of 78%

The second image shows a 5x5 matrix of the decision tree with an accuracy of 72%.

A4.3

Analyze decision tree model:

Advantages:

Easy to understand and explain: The structure of the decision tree is similar to the human decision-making process and is easy to visualize and understand.

Multiple types of data can be processed: both numerical and categorical data can be processed.

There are fewer data preprocessing requirements: there is no need to standardize or normalize the data, and missing values can be processed.

Ability to handle multiple output problems: decision trees can predict multiple target variables simultaneously.

Disadvantages:

Easy to overfit: Decision trees tend to overfit training data, especially when the depth of the tree is large.

Sensitive to small changes in the data: Small changes in the data can result in a completely different tree structure, which can affect the stability of the model.

Multi-category features: When there are more features and more categories, the decision tree is more inclined to choose features with more values for splitting.

Choose the SVM model and explain why:

Compared with SVM model, decision tree model is less complex, because the basic structure of SVM model is hyperplane and support vector. Although the training and prediction speed of SVM model are lower than that of decision tree model, the prediction accuracy is higher. Because there are four data to be processed, the multi-dimensional SVM model is dominant. Compared with decision tree model, SVM model has lower overfitting risk and stronger anti-noise ability. So, the final choice is the SVM model.

A5. Independent evaluation

```python
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.svm import SVC
from sklearn.metrics import confusion_matrix, accuracy_score

data = pd.read_csv('Student_List_A2.csv')

X = data.drop(columns=['StudentID', 'GradeClass', 'GPA'])
Y = data['GradeClass']

scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X)

svm_model = SVC(kernel='linear', random_state=42)
svm_model.fit(X_train_scaled, Y)

new_data = pd.read_csv('Student_List_A2_Submission.csv')

X_new = new_data.drop(columns=['StudentID'])     #remove useless columns

X_new_scaled = scaler.transform(X_new)    #standardize the new data, using the parameters previously trained

new_predict = svm_model.predict(X_new_scaled)   #make predict about new data

new_print = pd.DataFrame({                 #columns to save
    'StudentID': new_data['StudentID'],
    'GradeClass': new_predict
})

new_print.to_csv('Student_List_A2_Predictions.csv', index=False)   #save the result to a new csv file
```

```
Confusion matrix of 5x5:
[[  0  11   0   0   0]  A
 [  0  26  15   0   0]  B
 [  0  11  32  10   1]  C
 [  0   0  15  43  12]  D
 [  0   0   0  15 209]] F
Accuracy of SVM model: 0.78
```

There are 11 'A' and none of them are predict correct

	A	B
1	StudentID	GradeClass
2	5000	F
3	5001	C
4	5002	F
5	5003	D
6	5004	C
7	5005	C
8	5006	C
9	5007	F
10	5008	C
11	5009	F
12	5010	F
13	5011	F
14	5012	F
15	5013	C
16	5014	F
17	5015	C
18	5016	F
19	5017	D
20	5018	D
21	5019	C
22	5020	F
23	5021	F
24	5022	C
25	5023	C
26	5024	F
27	5025	F
28	5026	B
29	5027	F
135	5133	F
136	5134	C
137	5135	B
138	5136	C
139	5137	F
140	5138	F
141	5139	F
142	5140	F
143	5141	D
144	5142	D
145	5143	B
146	5144	F
147	5145	F
148	5146	F
149	5147	F
150	5148	F
151	5149	F
152	5150	D
153	5151	F
154	5152	B
155	5153	D
156	5154	F
157	5155	F
158	5156	D
159	5157	F
160	5158	F
161	5159	C
162	5160	F

I used SVM model for prediction, but I found that there was no 'A' in the predicted student grades. Through the 5x5 confusion matrix before the SVM model, it is obvious that during training and testing, the prediction accuracy of 'A' option is 0. So there is no 'A' option for this prediction.

Task B: Selection of Dataset, Clustering and Video Preparation

```
In [150]: import pandas as pd
          from sklearn.cluster import KMeans
          from sklearn.preprocessing import StandardScaler
          import matplotlib.pyplot as plt
          import seaborn as sns

          data = pd.read_csv('Financial Statements.csv')    #read csv file

          print("\nCheck for missing values: ")    #checks for missing values and outputs the missing number for each row
          print(data.isnull().sum())

          data.dropna(inplace=True)        #delete all rows that contain missing values

          data.to_csv('Financial Statements_new.csv', index=False)    #save the processed data to a new csv file

          print("\nMissing value after processing: ")
          print(data.isnull().sum())       #check for missing values again, make sure no missing value

          X = data[['Revenue', 'Net Income']]     #select two columns

          scaler = StandardScaler()        #standardized data
          X_scaled = scaler.fit_transform(X)

          kmeans = KMeans(n_clusters=3, random_state=42)     #create the KMeans clustering model and set the number to 3
          kmeans.fit(X_scaled)

          data['Cluster'] = kmeans.labels_      #adds cluster labels to the raw data set

          plt.figure(figsize=(10, 6))
          sns.scatterplot(x='Revenue', y='Net Income', hue='Cluster', data=data, palette='Set1', s=100)
          plt.title('K-Means center:Revenue vs Net Income')
          plt.xlabel('Revenue')
          plt.ylabel('Net Income')
          plt.show()
```

```
Check for missing values:
Year                                  0
Company                               0
Category                              0
Market Cap(in B USD)                  1
Revenue                               0
Gross Profit                          0
Net Income                            0
Earning Per Share                     0
EBITDA                                0
Share Holder Equity                   0
Cash Flow from Operating              0
Cash Flow from Investing              0
Cash Flow from Financial Activities   0
Current Ratio                         0
Debt/Equity Ratio                     0
ROE                                   0
ROA                                   0
ROI                                   0
Net Profit Margin                     0
Free Cash Flow per Share              0
Return on Tangible Equity             0
Number of Employees                   0
Inflation Rate(in US)                 0
dtype: int64
```

```
Missing value after processing:
Year                                    0
Company                                 0
Category                                0
Market Cap(in B USD)                    0
Revenue                                 0
Gross Profit                            0
Net Income                              0
Earning Per Share                       0
EBITDA                                  0
Share Holder Equity                     0
Cash Flow from Operating                0
Cash Flow from Investing                0
Cash Flow from Financial Activities     0
Current Ratio                           0
Debt/Equity Ratio                       0
ROE                                     0
ROA                                     0
ROI                                     0
Net Profit Margin                       0
Free Cash Flow per Share                0
Return on Tangible Equity               0
Number of Employees                     0
Inflation Rate(in US)                   0
dtype: int64
```