Model Training and Evaluation Report

Introduction

This report is a summary of the model training and evaluation process for the `model.py` script.
The script is used to train a some basic models to predict the churn of customers in a Bank Company.

Setup

Import necessary libraries and modules.

In [ ]:
```
import pandas as pd
import matplotlib.pyplot as plt

# data lib
from data.data_scripts.data_preprocessing import *

# model lib
from evaluations.evaluate import get_best_model
from models.model_scripts.train import train
from models.models import *
```

Data Loading

Load the dataset and display the first few rows to understand its structure.

In [ ]:
```
data = load_data("../data/raw_data/data.csv")
data.head()
```

Out[ ]:

	CLIENTNUM	Attrition_Flag	Customer_Age	Gender	Dependent_count	Educatio
0	768805383	Existing Customer	45	M	3	Higl
1	818770008	Existing Customer	49	F	5	C
2	713982108	Existing Customer	51	M	3	C
3	769911858	Existing Customer	40	F	4	Higl
4	709106358	Existing Customer	40	M	3	Une

5 rows × 23 columns

Data Preprocessing

In this section, we preprocess the data to make it ready for training. as I used 3 types of preprocessing techniques:

- **One Hot Encoding**:

  where I convert the categorical data into numerical data by creating a new column for each category and assign 1 or 0 to the column based on the presence of the category in the row. we can't use the one hot encoding on the 'Attrition_Flag' column because it's the target column.

  - Categolical data:
    - Income_Category
    - Education_Level
    - Marital_Status
    - Card_Category
    - Gender
- **Ordinal Encoding**:

  where I convert the categorical data into numerical data by assigning a unique number to each category.

  - Categolical data:
    - Attrition_Flag
    - Income_Category
    - Education_Level
    - Marital_Status
    - Card_Category
    - Gender
- **Diff Encoding**:

  In this type of pre processing I tried to separate the data into two parts, the first part is

  - the data that is based on the order of the data like:
    - Attrition_Flag
    - Education_Level
    - Income_Category

  and the second part is

  - the data that is not based on the order of the data like:
    - Marital_Status
    - Card_Category
    - Gender

Categorical Data Preprocessing

- Ordinal Encoding

In [ ]:
```
data_ordinal = pre_processing_categorical_ordinal(data)
data_ordinal.head()
```

Out[ ]:

	CLIENTNUM	Attrition_Flag	Customer_Age	Gender	Dependent_count	Educatio
0	768805383	1	45	M	3	
1	818770008	1	49	F	5	
2	713982108	1	51	M	3	
3	769911858	1	40	F	4	
4	709106358	1	40	M	3	

5 rows × 23 columns

- One-Hot Encoding

In [ ]:
```
data_onehot = pre_processing_categorical_onehot(data)
data_onehot.head()
```

Out[ ]:

	CLIENTNUM	Attrition_Flag	Customer_Age	Dependent_count	Education_Level
0	768805383	1	45	3	3
1	818770008	1	49	5	2
2	713982108	1	51	3	2
3	769911858	1	40	4	3
4	709106358	1	40	3	5

5 rows × 30 columns

Combining All Preprocessing

- All Ordinal

In [ ]:
```
data_all_ordinal = pre_processing_categorical_all_ordinal(data)
data_all_ordinal.head()
```

Out[ ]:

	CLIENTNUM	Attrition_Flag	Customer_Age	Gender	Dependent_count	Educatio
0	768805383	1	45	1	3	
1	818770008	1	49	0	5	
2	713982108	1	51	1	3	
3	769911858	1	40	0	4	
4	709106358	1	40	1	3	

5 rows × 23 columns

- All One-Hot

In [ ]:
```
data_all_onehot = pre_processing_categorical_all_onehot(data)
data_all_onehot
```

Out[ ]:

	CLIENTNUM	Attrition_Flag	Customer_Age	Dependent_count	Months_on_b
0	768805383	1	45	3	
1	818770008	1	49	5	
2	713982108	1	51	3	
3	769911858	1	40	4	
4	709106358	1	40	3	
...	...	...	...	...	
10122	772366833	1	50	2	
10123	710638233	0	41	2	
10124	716506083	0	44	1	
10125	717406983	0	30	2	
10126	714337233	0	43	2	

10127 rows × 41 columns

Model Training and Evaluation

Train multiple models with different preprocessing techniques.
Evaluate the models using various metrics.

In [ ]:
```
evaluations = train("../data/raw_data/data.csv")
```

```
/Users/bssayla/Documents/Projects/Level_1/Churn_Prediction/venv/lib/python
3.9/site-packages/sklearn/ensemble/_weight_boosting.py:527: FutureWarning:
The SAMME.R algorithm (the default) is deprecated and will be removed in
1.6. Use the SAMME algorithm to circumvent this warning.
  warnings.warn(
/Users/bssayla/Documents/Projects/Level_1/Churn_Prediction/venv/lib/python
3.9/site-packages/sklearn/ensemble/_weight_boosting.py:527: FutureWarning:
The SAMME.R algorithm (the default) is deprecated and will be removed in
1.6. Use the SAMME algorithm to circumvent this warning.
  warnings.warn(
/Users/bssayla/Documents/Projects/Level_1/Churn_Prediction/venv/lib/python
3.9/site-packages/sklearn/ensemble/_weight_boosting.py:527: FutureWarning:
The SAMME.R algorithm (the default) is deprecated and will be removed in
1.6. Use the SAMME algorithm to circumvent this warning.
  warnings.warn(
```

In [ ]:

```
import json
import numpy as np
def json_serialize(obj):
    if isinstance(obj, np.ndarray):
        return obj.tolist()
    return obj

with open('../evaluations/evaluations.json', 'w') as f:
    json.dump(evaluations, f, default=json_serialize)
```

Compare Models

Compare the performance of the models using various metrics.

first we separate the model's metrics into arrays to be able to plot them.

In [ ]:

```
# get the accuracy of the models
accuracy = []
f1 = []
precision = []
recall = []
roc_auc = []
for pre_processing_type in evaluations:
    for model in evaluations[pre_processing_type]:
        accuracy.append(evaluations[pre_processing_type][model]["accuracy
        f1.append(evaluations[pre_processing_type][model]["f1"]*100)
        precision.append(evaluations[pre_processing_type][model]["precisi
        recall.append(evaluations[pre_processing_type][model]["recall"]*1
        roc_auc.append(evaluations[pre_processing_type][model]["roc_auc"]
accuracy = np.round(np.array(accuracy),2)
f1 = np.round(np.array(f1),2)
precision = np.round(np.array(precision),2)
recall = np.round(np.array(recall),2)
roc_auc = np.round(np.array(roc_auc),2)

# create a dataframe
data = pd.DataFrame(
    {
        "accuracy": accuracy,
        "f1": f1,
        "precision": precision,
        "recall": recall,
```

```
        "roc_auc": roc_auc
    }
)
data.to_csv("../evaluations/evaluations.csv", index=False)
```

Accuracy Table

Model	One hot	Ordinal	Diff
1st	95.46	95.66	95.76
2nd	95.16	95.06	95.16
3rd	91.12	92.50	92.05

Precision Table

Model	One hot	Ordinal	Diff
1st	95.84	96.37	96.36
2nd	96.41	96.46	96.41
3rd	91.98	93.43	92.77

recall Table

Model	One hot	Ordinal	Diff
1st	98.88	98.53	98.65
2nd	97.88	97.70	97.88
3rd	97.94	97.94	98.18

F1 Score Table

Model	One hot	Ordinal	Diff
1st	97.33	97.44	97.50
2nd	97.14	97.08	97.14
3rd	94.87	95.63	95.40

ROC AUC Table

Model	One hot	Ordinal	Diff
1st	88.28	89.63	89.69
2nd	89.46	89.53	89.46
3rd	76.80	81.08	79.21

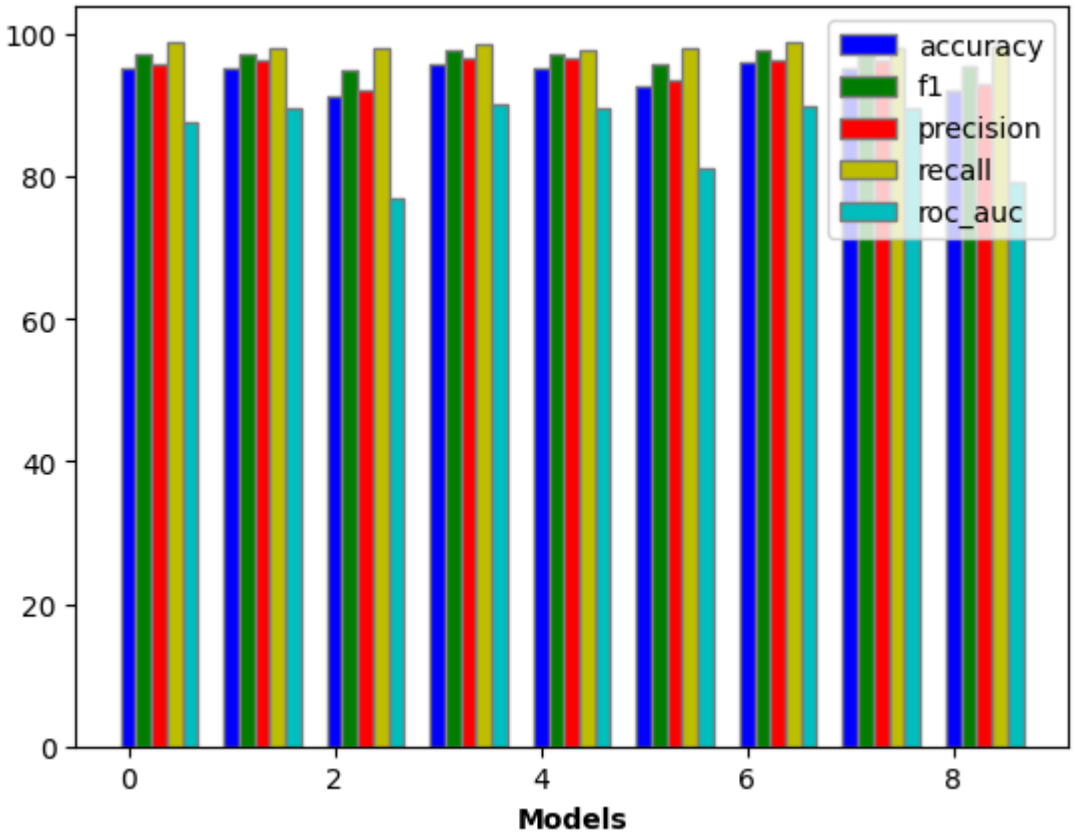
Plot the Evaluations

In [ ]:
```
fig, ax = plt.subplots()
barWidth = 0.15
r1 = np.arange(len(accuracy))
r2 = [x + barWidth for x in r1]
r3 = [x + barWidth for x in r2]
r4 = [x + barWidth for x in r3]
r5 = [x + barWidth for x in r4]

plt.bar(r1, accuracy, color='b', width=barWidth, edgecolor='grey', label=
plt.bar(r2, f1, color='g', width=barWidth, edgecolor='grey', label='f1')
plt.bar(r3, precision, color='r', width=barWidth, edgecolor='grey', label
plt.bar(r4, recall, color='y', width=barWidth, edgecolor='grey', label='r
plt.bar(r5, roc_auc, color='c', width=barWidth, edgecolor='grey', label='

plt.xlabel('Models', fontweight='bold')
plt.legend()
plt.show()

fig.savefig("../evaluations/evaluations.png")
```



Best Model Selection

Select the best model based on evaluation metrics.

- The best model based on the accuracy

In [ ]:
```
# get the best model
best_model = get_best_model(evaluations, "accuracy")
best_model
```

Out[ ]:
```
('DIFF', 'first_model', 0.9595261599210266)
```

- The best model based on the f1 score

In [ ]:
```
# get the best model
best_model = get_best_model(evaluations, "f1")
best_model
```

Out[ ]:
```
('DIFF', 'first_model', np.float64(0.9761766414875073))
```

- The best model based on precision

In [ ]:
```
# get the best model
best_model = get_best_model(evaluations, "precision")
best_model
```

Out[ ]:
```
('ORDINAL', 'first_model', np.float64(0.9654178674351584))
```

- The best model based on recall

In [ ]:
```
# get the best model
best_model = get_best_model(evaluations, "recall")
best_model
```

Out[ ]:
```
('DIFF', 'first_model', np.float64(0.9888169511477339))
```

- The best model based on roc auc

In [ ]:
```
# get the best model
best_model = get_best_model(evaluations, "roc_auc")
best_model
```

Out[ ]:
```
('ORDINAL', 'first_model', np.float64(0.9011939025114611))
```

Conclusion

Metric	Best pre-Processing	Best Model	Score
Accuracy	DIFF	first_model	0.959526159921026
F1_Score	DIFF	first_model	0.9761766414875073
Precision	ORDINAL	first_model	0.9654178674351584
Recall	DIFF	first_model	0.9888169511477339
ROC_AUC	ORDINAL	first_model	0.9011939025114611

At the end, we can conclude that the best model is the first model with the diff pre-processing technique.