

```
In [1]: import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline
import warnings, re, joblib
warnings.filterwarnings("ignore")
from sklearn.model_selection import train_test_split, RandomizedSearchCV, GridSearchCV
from sklearn.linear_model import LogisticRegression, PassiveAggressiveClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier, ExtraTreesClassifier
from xgboost import XGBClassifier, XGBRFClassifier
from catboost import CatBoostClassifier
from lightgbm import LGBMClassifier
from sklearn.svm import LinearSVC
from sklearn.naive_bayes import GaussianNB, BernoulliNB
from sklearn.neural_network import MLPClassifier
from scipy.stats import probplot
from sklearn.experimental import enable_iterative_imputer
from sklearn.impute import IterativeImputer, SimpleImputer
from sklearn.feature_selection import SelectKBest, SelectPercentile, SelectFpr, SelectFdr
from sklearn.preprocessing import StandardScaler, OrdinalEncoder, PowerTransformer
from sklearn.metrics import ConfusionMatrixDisplay, classification_report, accuracy_score
from imblearn.over_sampling import SMOTE
```

Can we predict whether a customer will have a "Good" or "Bad" credit score next month based on their financial and credit-related attributes?

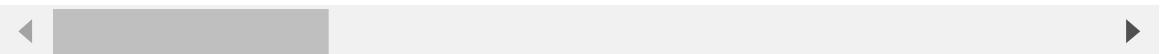
```
In [2]: train = pd.read_csv('train.csv')
```

```
In [3]: train.head()
```

```
Out[3]:
```

	ID	Customer_ID	Month	Name	Age	SSN	Occupation	Annual_Income	Month
0	0x1602	CUS_0xd40	January	Aaron Maashoh	23	821-00-0265	Scientist	19114.12	
1	0x1603	CUS_0xd40	February	Aaron Maashoh	23	821-00-0265	Scientist	19114.12	
2	0x1604	CUS_0xd40	March	Aaron Maashoh	-500	821-00-0265	Scientist	19114.12	
3	0x1605	CUS_0xd40	April	Aaron Maashoh	23	821-00-0265	Scientist	19114.12	
4	0x1606	CUS_0xd40	May	Aaron Maashoh	23	821-00-0265	Scientist	19114.12	

5 rows × 28 columns



```
In [4]: train.shape
```

```
Out[4]: (100000, 28)
```

```
In [5]: train.duplicated().sum()
```

```
Out[5]: 0
```

```
In [6]: train.isnull().sum()
```

```
Out[6]: ID          0  
Customer_ID      0  
Month           0  
Name            9985  
Age             0  
SSN             0  
Occupation       0  
Annual_Income    0  
Monthly_Inhand_Salary 15002  
Num_Bank_Accounts 0  
Num_Credit_Card   0  
Interest_Rate     0  
Num_of_Loan       0  
Type_of_Loan      11408  
Delay_from_due_date 0  
Num_of_Delayed_Payment 7002  
Changed_Credit_Limit 0  
Num_Credit_Inquiries 1965  
Credit_Mix        0  
Outstanding_Debt  0  
Credit_Utilization_Ratio 0  
Credit_History_Age 9030  
Payment_of_Min_Amount 0  
Total_EMI_per_month 0  
Amount_invested_monthly 4479  
Payment_Behaviour 0  
Monthly_Balance    1200  
Credit_Score       0  
dtype: int64
```

```
In [7]: train.drop(['Month', 'ID', "Customer_ID", "Name", "SSN", "Credit_History_Age", 'I
```

```
In [8]: train.shape
```

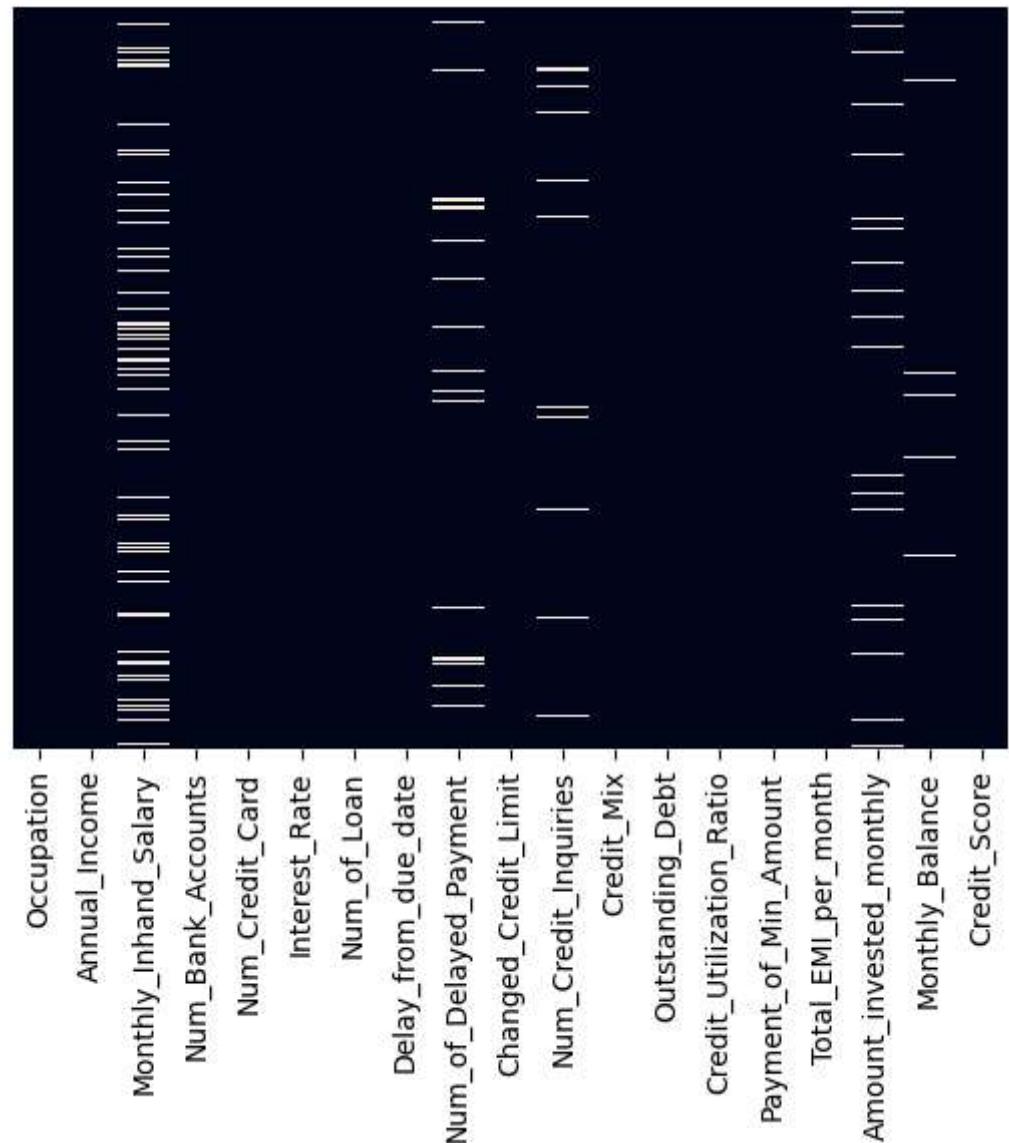
```
Out[8]: (100000, 19)
```

```
In [9]: def get_missing_data_details(hrdata):
    sns.heatmap(train.isnull(), yticklabels=False, cbar=False)
    total = train.isnull().sum().sort_values(ascending=False)
    percent = ((train.isnull().sum() / train.isnull().count()) * 100).sort_
    missing = pd.concat([total, percent], axis=1, keys=['Total', 'Percent'])
    missing = missing[missing["Percent"] > 0]

    plt.figure(figsize=(5, 5))
    sns.barplot(x=missing["Percent"], y=missing.index)
    plt.xlabel('Percentage', fontsize=15)
    plt.ylabel('Features', fontsize=15)
    plt.title('Missing Data %', fontsize=15)

    plt.show()

# Call the function with your hrdata
get_missing_data_details(train)
```





```
In [10]: train.isnull().sum()
```

```
Out[10]: Occupation          0
Annual_Income           0
Monthly_Inhand_Salary  15002
Num_Bank_Accounts       0
Num_Credit_Card         0
Interest_Rate           0
Num_of_Loan              0
Delay_from_due_date     0
Num_of_Delayed_Payment  7002
Changed_Credit_Limit    0
Num_Credit_Inquiries    1965
Credit_Mix               0
Outstanding_Debt        0
Credit_Utilization_Ratio 0
Payment_of_Min_Amount   0
Total_EMI_per_month      0
Amount_invested_monthly 4479
Monthly_Balance          1200
Credit_Score              0
dtype: int64
```

In [11]: `train.info()`

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 100000 entries, 0 to 99999
Data columns (total 19 columns):
 #   Column           Non-Null Count  Dtype  
--- 
 0   Occupation       100000 non-null   object  
 1   Annual_Income    100000 non-null   object  
 2   Monthly_Inhand_Salary  84998 non-null   float64 
 3   Num_Bank_Accounts 100000 non-null   int64  
 4   Num_Credit_Card   100000 non-null   int64  
 5   Interest_Rate    100000 non-null   int64  
 6   Num_of_Loan      100000 non-null   object  
 7   Delay_from_due_date 100000 non-null   int64  
 8   Num_of_Delayed_Payment 92998 non-null   object  
 9   Changed_Credit_Limit 100000 non-null   object  
 10  Num_Credit_Inquiries 98035 non-null   float64 
 11  Credit_Mix      100000 non-null   object  
 12  Outstanding_Debt 100000 non-null   object  
 13  Credit_Utilization_Ratio 100000 non-null   float64 
 14  Payment_of_Min_Amount 100000 non-null   object  
 15  Total_EMI_per_month 100000 non-null   float64 
 16  Amount_invested_monthly 95521 non-null   object  
 17  Monthly_Balance  98800 non-null   object  
 18  Credit_Score     100000 non-null   object  
dtypes: float64(4), int64(4), object(11)
memory usage: 14.5+ MB
```

In [12]: `train.head()`

Out[12]:

	Occupation	Annual_Income	Monthly_Inhand_Salary	Num_Bank_Accounts	Num_Credit_Card
0	Scientist	19114.12	1824.843333	3	
1	Scientist	19114.12	NaN	3	
2	Scientist	19114.12	NaN	3	
3	Scientist	19114.12	NaN	3	
4	Scientist	19114.12	1824.843333	3	

Treating anomaly and Feature engineering

In [13]: `train['Annual_Income'].unique()`

Out[13]:

```
array(['19114.12', '34847.84', '34847.84_',
       ..., '20002.88', '39628.99',
       '39628.99_'], dtype=object)
```

```
In [14]: train['Annual_Income'] = train['Annual_Income'].apply(lambda x:x.replace("_", ""))
```

```
In [15]: train['Annual_Income'] = train['Annual_Income'].astype(float)
```

```
In [16]: train['Num_of_Loan'] = train['Num_of_Loan'].apply(lambda x:x.replace('_', ''))
```

```
In [17]: train['Num_of_Loan'] = train['Num_of_Loan'].astype(float)
```

```
In [18]: import pandas as pd  
  
# Assuming test is your DataFrame  
train["Num_of_Delayed_Payment"] = train["Num_of_Delayed_Payment"].apply(lambda x:x.replace(" ", 0))
```

```
In [19]: train['Num_of_Delayed_Payment'] = train['Num_of_Delayed_Payment'].astype(float)
```

```
In [20]: train['Changed_Credit_Limit'] = train['Changed_Credit_Limit'].apply(lambda x:x.replace(" ", 0))
```

```
In [21]: train['Changed_Credit_Limit'] = train['Changed_Credit_Limit'].replace(' ', 0)
```

```
In [22]: train['Changed_Credit_Limit'] = train['Changed_Credit_Limit'].astype(float)
```

```
In [23]: train['Outstanding_Debt'] = train['Outstanding_Debt'].apply(lambda x:x.replace(" ", 0))
```

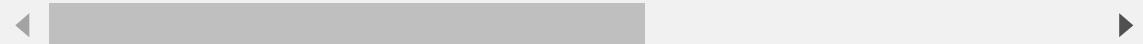
```
In [24]: train['Outstanding_Debt'] = train['Outstanding_Debt'].astype(float)
```

```
In [25]:  
# Assuming test is your DataFrame  
train["Amount_invested_monthly"] = train["Amount_invested_monthly"].apply(lambda x:x.replace(" ", 0))
```

```
In [26]: train['Amount_invested_monthly'] = train['Amount_invested_monthly'].astype(float)
```

In [27]:

```
# Assuming test is your DataFrame
train["Monthly_Balance"] = train["Monthly_Balance"].apply(lambda x: x.replace
```

In [28]: `train['Monthly_Balance'] = train['Monthly_Balance'].astype(float)`In [29]: `train.info()`

```
1 Annual_Income      100000 non-null   float64
2 Monthly_Inhand_Salary 84998 non-null   float64
3 Num_Bank_Accounts    100000 non-null   int64
4 Num_Credit_Card      100000 non-null   int64
5 Interest_Rate        100000 non-null   int64
6 Num_of_Loan          100000 non-null   float64
7 Delay_from_due_date 100000 non-null   int64
8 Num_of_Delayed_Payment 92998 non-null   float64
9 Changed_Credit_Limit 100000 non-null   float64
10 Num_Credit_Inquiries 98035 non-null   float64
11 Credit_Mix          100000 non-null   object
12 Outstanding_Debt    100000 non-null   float64
13 Credit_Utilization_Ratio 100000 non-null   float64
14 Payment_of_Min_Amount 100000 non-null   object
15 Total_EMI_per_month 100000 non-null   float64
16 Amount_invested_monthly 95521 non-null   float64
17 Monthly_Balance     98800 non-null   float64
18 Credit_Score        100000 non-null   object
dtypes: float64(11), int64(4), object(4)
memory usage: 14.5+ MB
```

In [30]: `train.isnull().sum()`

```
Occupation                  0
Annual_Income                0
Monthly_Inhand_Salary       15002
Num_Bank_Accounts           0
Num_Credit_Card              0
Interest_Rate                0
Num_of_Loan                  0
Delay_from_due_date         0
Num_of_Delayed_Payment      7002
Changed_Credit_Limit        0
Num_Credit_Inquiries        1965
Credit_Mix                  0
Outstanding_Debt            0
Credit_Utilization_Ratio    0
Payment_of_Min_Amount       0
Total_EMI_per_month         0
Amount_invested_monthly    4479
Monthly_Balance              1200
Credit_Score                 0
dtype: int64
```

```
In [31]: train2 = train.dropna()
```

```
In [32]: train2.isnull().sum()
```

```
Out[32]: Occupation          0  
Annual_Income           0  
Monthly_Inhand_Salary    0  
Num_Bank_Accounts        0  
Num_Credit_Card          0  
Interest_Rate            0  
Num_of_Loan              0  
Delay_from_due_date      0  
Num_of_Delayed_Payment   0  
Changed_Credit_Limit     0  
Num_Credit_Inquiries     0  
Credit_Mix                0  
Outstanding_Debt         0  
Credit_Utilization_Ratio  0  
Payment_of_Min_Amount     0  
Total_EMI_per_month       0  
Amount_invested_monthly  0  
Monthly_Balance           0  
Credit_Score              0  
dtype: int64
```

```
In [33]: train2.head()
```

```
Out[33]:
```

	Occupation	Annual_Income	Monthly_Inhand_Salary	Num_Bank_Accounts	Num_Credit_Ca
0	Scientist	19114.12	1824.843333	3	
6	Scientist	19114.12	1824.843333	3	
7	Scientist	19114.12	1824.843333	3	
8	_____	34847.84	3037.986667	2	
9	Teacher	34847.84	3037.986667	2	

```
In [34]: train2 = train2[train2['Occupation'] != '_____']
```

```
In [35]: train2['Occupation'].value_counts()
```

```
Out[35]: Occupation
Lawyer           4867
Engineer         4650
Mechanic         4627
Architect        4605
Accountant       4571
Developer        4563
Media_Manager    4553
Teacher          4545
Scientist         4524
Doctor           4523
Entrepreneur     4494
Journalist        4442
Musician          4373
Manager           4350
Writer            4311
Name: count, dtype: int64
```

```
In [36]: train2.shape
```

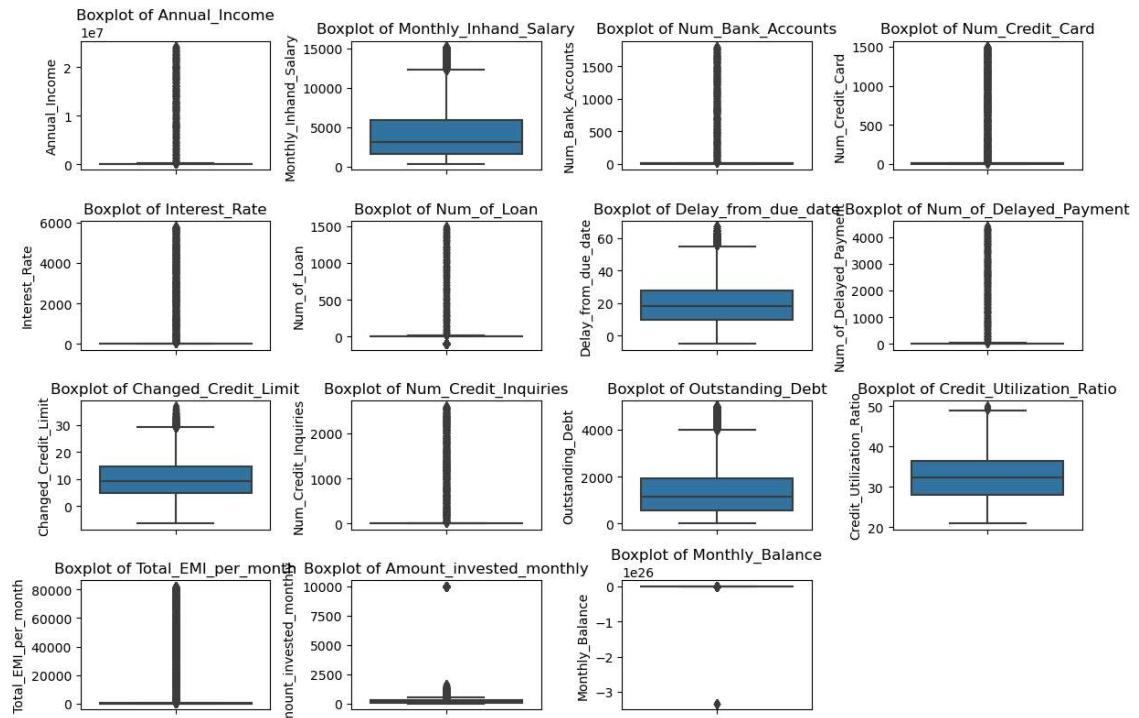
```
Out[36]: (67998, 19)
```

Treating and checking outliers`

```
In [37]: # Select only the numeric columns
numeric_columns = train2.select_dtypes(include=['int', 'float'])

# Plot boxplots for each numeric feature
plt.figure(figsize=(12, 8))
for i, feature in enumerate(numeric_columns.columns):
    plt.subplot(4, 4, i + 1)
    sns.boxplot(y=numeric_columns[feature], data=train2)
    plt.title('Boxplot of {}'.format(feature))
    plt.tight_layout()

plt.show()
```



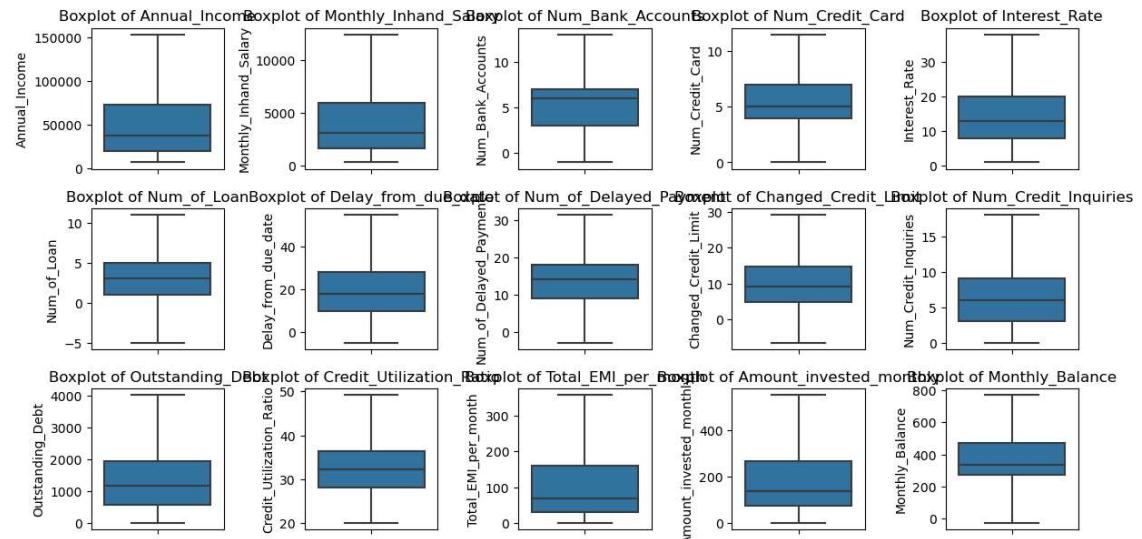
```
In [38]: def remove_outlier(col):
    col = pd.to_numeric(col, errors='coerce')
    Q1, Q3 = col.quantile([0.25, 0.75])
    IQR = Q3 - Q1
    lower_range = Q1 - (1.5 * IQR)
    upper_range = Q3 + (1.5 * IQR)
    return lower_range, upper_range

feature_list = train.columns

for feature in feature_list:
    LL, UL = remove_outlier(train[feature])
    train[feature] = np.where(train[feature] > UL, UL, train[feature])
    train[feature] = np.where(train[feature] < LL, LL, train[feature])
```

```
In [39]: plt.figure(figsize=(12, 8))
numeric_features = train.select_dtypes(include=[np.number]).columns

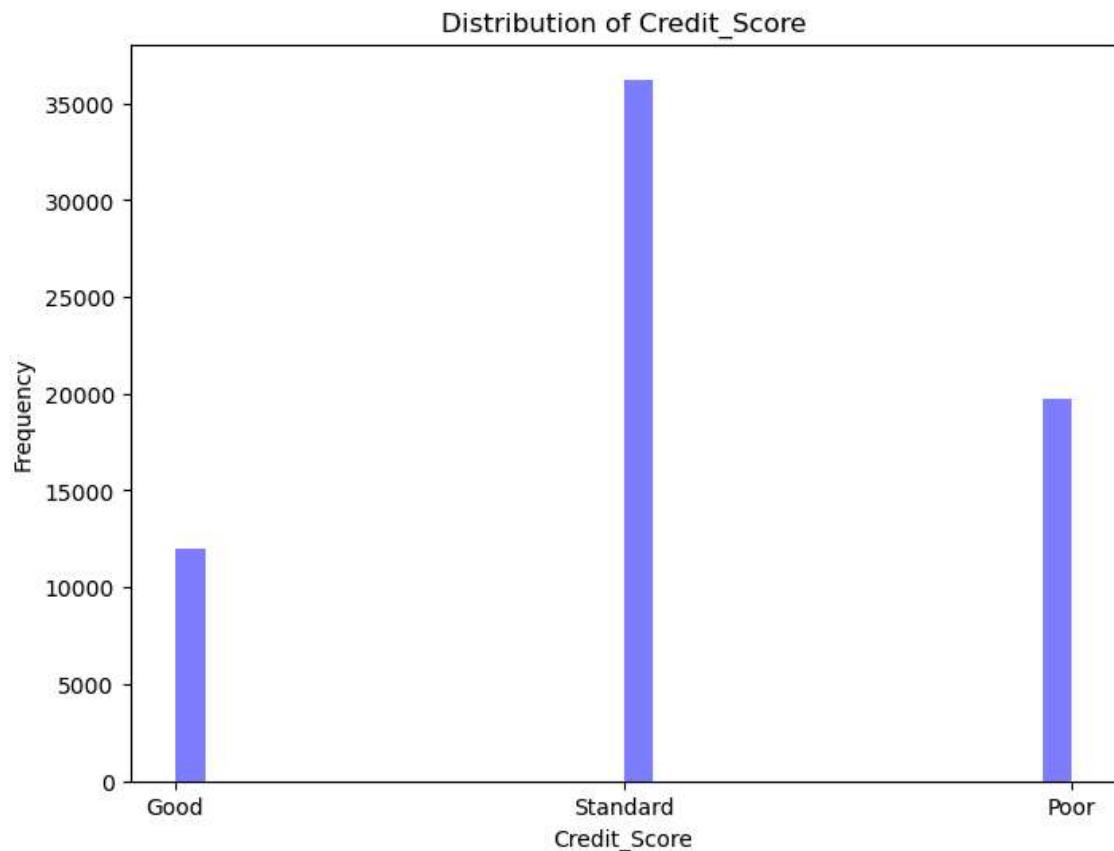
for i, feature in enumerate(numeric_features):
    plt.subplot(4, 5, i + 1)
    sns.boxplot(y=train[feature], data=train)
    plt.title('Boxplot of {}'.format(feature))
    plt.tight_layout()
```



Chekking Biasness in the data

```
In [40]: import matplotlib.pyplot as plt

plt.figure(figsize=(8, 6))
plt.hist(train2['Credit_Score'], bins=30, color='blue', alpha=0.5)
plt.xlabel('Credit_Score')
plt.ylabel('Frequency')
plt.title('Distribution of Credit_Score')
plt.show()
```



In [41]: `train2.info()`

```
<class 'pandas.core.frame.DataFrame'>
Index: 67998 entries, 0 to 99999
Data columns (total 19 columns):
 #   Column           Non-Null Count  Dtype  
--- 
 0   Occupation       67998 non-null   object  
 1   Annual_Income    67998 non-null   float64 
 2   Monthly_Inhand_Salary 67998 non-null   float64 
 3   Num_Bank_Accounts 67998 non-null   int64  
 4   Num_Credit_Card   67998 non-null   int64  
 5   Interest_Rate    67998 non-null   int64  
 6   Num_of_Loan       67998 non-null   float64 
 7   Delay_from_due_date 67998 non-null   int64  
 8   Num_of_Delayed_Payment 67998 non-null   float64 
 9   Changed_Credit_Limit 67998 non-null   float64 
 10  Num_Credit_Inquiries 67998 non-null   float64 
 11  Credit_Mix        67998 non-null   object  
 12  Outstanding_Debt  67998 non-null   float64 
 13  Credit_Utilization_Ratio 67998 non-null   float64 
 14  Payment_of_Min_Amount 67998 non-null   object  
 15  Total_EMI_per_month 67998 non-null   float64 
 16  Amount_invested_monthly 67998 non-null   float64 
 17  Monthly_Balance   67998 non-null   float64 
 18  Credit_Score       67998 non-null   object  
dtypes: float64(11), int64(4), object(4)
memory usage: 10.4+ MB
```

Convert Object Feature types for Linear Discriminant Analysis

Doing label and hot encoding

In [73]: `import pandas as pd
from sklearn.preprocessing import LabelEncoder`

In [74]: `categorical_cols = ['Credit_Score', 'Payment_of_Min_Amount', 'Credit_Mix',
label_encoder = LabelEncoder()
for col in categorical_cols:
 if col in train2.columns:

 train2[col] = label_encoder.fit_transform(train2[col].astype(str))`

In [75]:

```
# Display the mapping of original labels to encoded values
print(f"Mapping for {col}:")
for original_label, encoded_value in zip(label_encoder.classes_, la:
    print(f"  {original_label} is encoded as {encoded_value}")

# Display the updated DataFrame
print("\nUpdated DataFrame:")
print(train2)
```

Mapping for Occupation:

```

0 is encoded as 0
1 is encoded as 1
10 is encoded as 2
11 is encoded as 3
12 is encoded as 4
13 is encoded as 5
14 is encoded as 6
2 is encoded as 7
3 is encoded as 8
4 is encoded as 9
5 is encoded as 10
6 is encoded as 11
7 is encoded as 12
8 is encoded as 13
9 is encoded as 14

```

Updated DataFrame:

	Occupation	Annual_Income	Monthly_Inhand_Salary	Num_Bank_Account
0	4	19114.12	1824.843333	
3				
6	4	19114.12	1824.843333	
3				
7	4	19114.12	1824.843333	
3				
9	5	34847.84	3037.986667	
2				
10	5	34847.84	3037.986667	
2				
...
...				
99994	14	39628.99	3359.415833	
4				
99995	14	39628.99	3359.415833	
4				
99996	14	39628.99	3359.415833	
4				
99997	14	39628.99	3359.415833	
4				
99999	14	39628.99	3359.415833	
4				
	Num_Credit_Card	Interest_Rate	Num_of_Loan	Delay_from_due_date
\				
0	4	3	4.0	3
6	4	3	4.0	3
7	4	3	4.0	3
9	4	6	1.0	7
10	1385	6	1.0	3
...
99994	6	7	2.0	20
99995	6	7	2.0	23
99996	6	7	2.0	18
99997	6	5729	2.0	27
99999	6	7	2.0	18
	Num_of_Delayed_Payment	Changed_Credit_Limit	Num_Credit_Inquiries	
\				
0	7.0	11.27	4.0	
6	8.0	11.27	4.0	

7	6.0	11.27	4.0
9	1.0	7.42	2.0
10	-1.0	5.42	2.0
...
99994	6.0	9.50	3.0
99995	7.0	11.50	3.0
99996	7.0	11.50	3.0
99997	6.0	11.50	3.0
99999	6.0	11.50	3.0

	Credit_Mix	Outstanding_Debt	Credit_Utilization_Ratio	\
0	3	809.98	26.822620	
6	1	809.98	22.537593	
7	1	809.98	23.933795	
9	1	605.03	38.550848	
10	3	605.03	33.224951	
...
99994	3	502.38	39.323569	
99995	3	502.38	34.663572	
99996	3	502.38	40.565631	
99997	1	502.38	41.255522	
99999	1	502.38	34.192463	

y \	Payment_of_Min_Amount	Total_EMI_per_month	Amount_invested_monthly
0	1	49.574949	80.41529
5			
6	1	49.574949	178.34406
7			
7	1	49.574949	24.78521
7			
9	1	18.816215	40.39123
8			
10	1	18.816215	58.51597
6			
...
...			
99994	1	35.104023	140.58140
3			
99995	1	35.104023	60.97133
3			
99996	1	35.104023	54.18595
0			
99997	1	35.104023	24.02847
7			
99999	1	35.104023	167.16386
5			

	Monthly_Balance	Credit_Score
0	312.494089	2
6	244.565317	2
7	358.124168	1
9	484.591214	2
10	466.466476	1
...
99994	410.256158	0
99995	479.866228	0
99996	496.651610	0
99997	516.809083	0
99999	393.673696	0

```
[67998 rows x 19 columns]
```

ordinal encoding

```
In [76]: #train2['Credit_Score']=np.where(train2['Credit_Score'] =='Poor', '0', tra  
#train2['Credit_Score']=np.where(train2['Credit_Score'] =='Standard', '1',  
#train2['Credit_Score']=np.where(train2['Credit_Score'] =='Good', '2', tra
```

```
In [ ]:
```

```
In [77]: train2.head()
```

Out[77]:

	Occupation	Annual_Income	Monthly_Inhand_Salary	Num_Bank_Accounts	Num_Credit_Cards
0	4	19114.12	1824.843333	3	
6	4	19114.12	1824.843333	3	
7	4	19114.12	1824.843333	3	
9	5	34847.84	3037.986667	2	
10	5	34847.84	3037.986667	2	1

◀ ▶

```
In [78]: train3 =train2
```

spliting the data in train test split

```
In [79]: X= train3.drop("Credit_Score",axis=1)  
y=train3["Credit_Score"]
```

In [80]: X.value_counts()

Out[80]:

	Occupation	Annual_Income	Monthly_Inhand_Salary	Num_Bank_Accounts	Num_Credit_Card	Interest_Rate	Num_of_Loan	Delay_from_due_date	Num_of_Delayed_Payment	Changed_Credit_Limit	Num_Credit_Inquiries	Credit_Mix	Outstanding_Debt	Credit_Utilization_Ratio	Payment_of_Min_Amount	Total_EMI_per_month	Amount_invested_monthly	Monthly_Balance
0		7.021910e+03	507.159167			3											9	
20		7.0	27			18.0												
12.75			13.0			3										1919.27		
36.665508			2													27.590627	29.	
830585			273.294705			1												
10		8.637875e+03	614.822917			9										10		
20		5.0	25			22.0												
12.63			8.0			0										1549.11		
22.891886			0													19.940708	31.	
104887			300.436696			1												
		8.294385e+03	946.198750			7										7		
26		7.0	6			11.0												
7.59			4.0			2										1756.74		
25.216641			0													48.032063	72.	
876540			263.711272			1												
12.0			7.59			4.0										2		
1756.74			35.554515			2										48.032		
063			112.355868			224.231944										1		
			8.308130e+03	916.344167		10										9		
20		7.0	27			10.0												
15.96			11.0			2										2683.29		
33.593707			0													27.600324	51.	
617315			292.416777			1												
..																		
5		7.708690e+03	379.390833			8										6		
25		5.0	57			13.0												
28.96			11.0			0										4147.58		
35.687940			2													26.588314	0.0	
00000			251.768000			1												
15.0			28.96			11.0										3		
4147.58			26.663496			2										26.588		
314			21.471077			269.879692										1		
16.0			28.96			11.0										0		
4147.58			35.136649			2										26.588		
314			15.370843			245.979926										1		
17.0			28.96			11.0										0		
4147.58			38.792202			2										26.588		
314			18.161497			273.189272										1		
14		2.417715e+07	2373.828333			4											3	
6		0.0	6			4.0												
2.56			3.0			1										1443.42		
39.408584			1													0.000000	13	
6.175542			381.207291			1												
Name: count, Length: 67998, dtype: int64																		

```
In [81]: y.value_counts()
```

```
Out[81]: Credit_Score
1    36239
0    19737
2    12022
Name: count, dtype: int64
```

```
In [82]: smote = SMOTE() # Synthetic Minority Oversampling TEchnique
X, y = smote.fit_resample(X,y)
```

```
In [83]: y.value_counts()
```

```
Out[83]: Credit_Score
2    36239
1    36239
0    36239
Name: count, dtype: int64
```

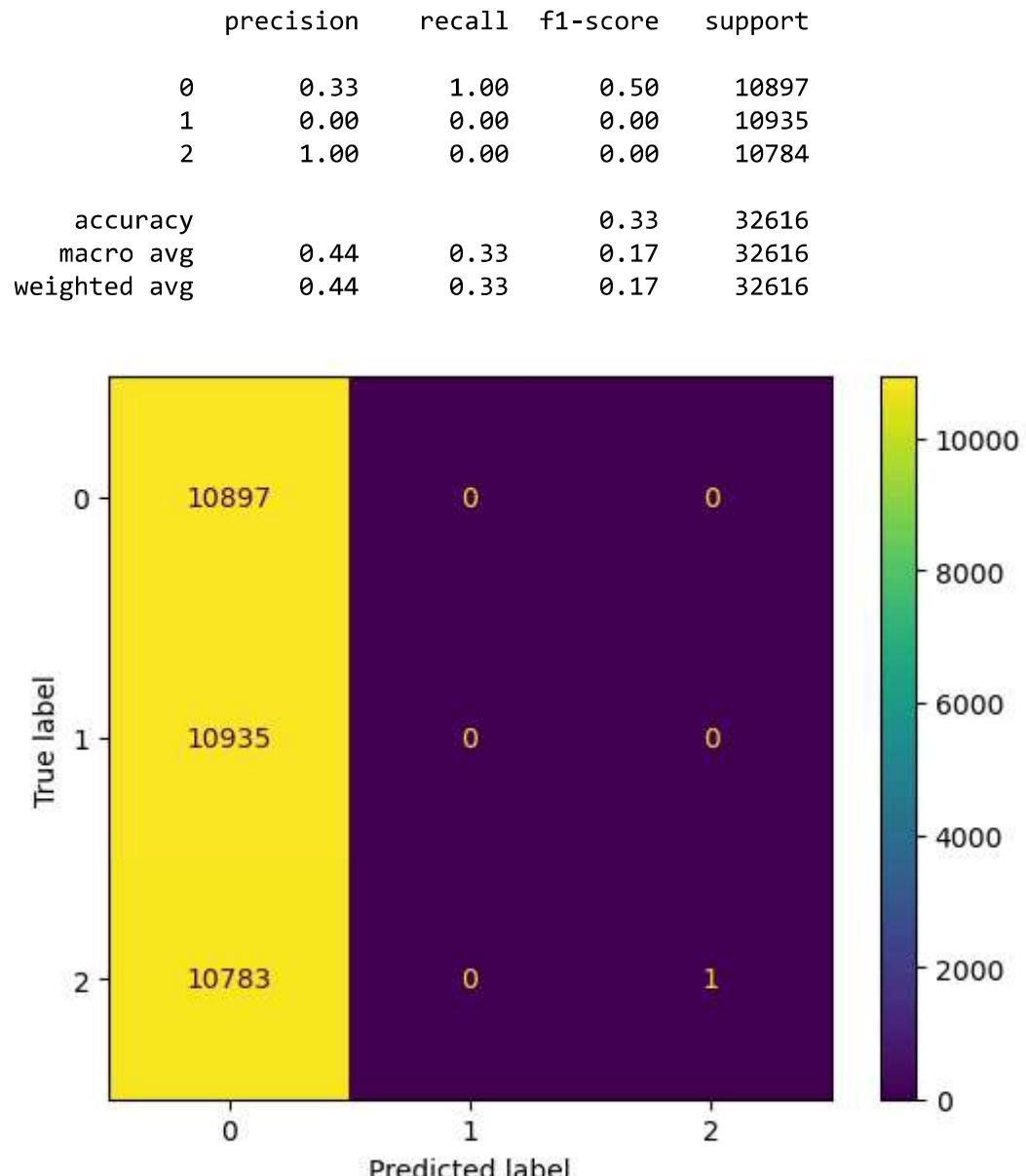
```
In [84]: X_train, X_test, y_train, y_test = train_test_split(X,y,test_size=0.3,shuf-
```

Model Training & Evaluation

```
In [85]: model_names = []
accuracy_scores = []
precision_scores = []
recall_scores = []
f1_scores = []
```

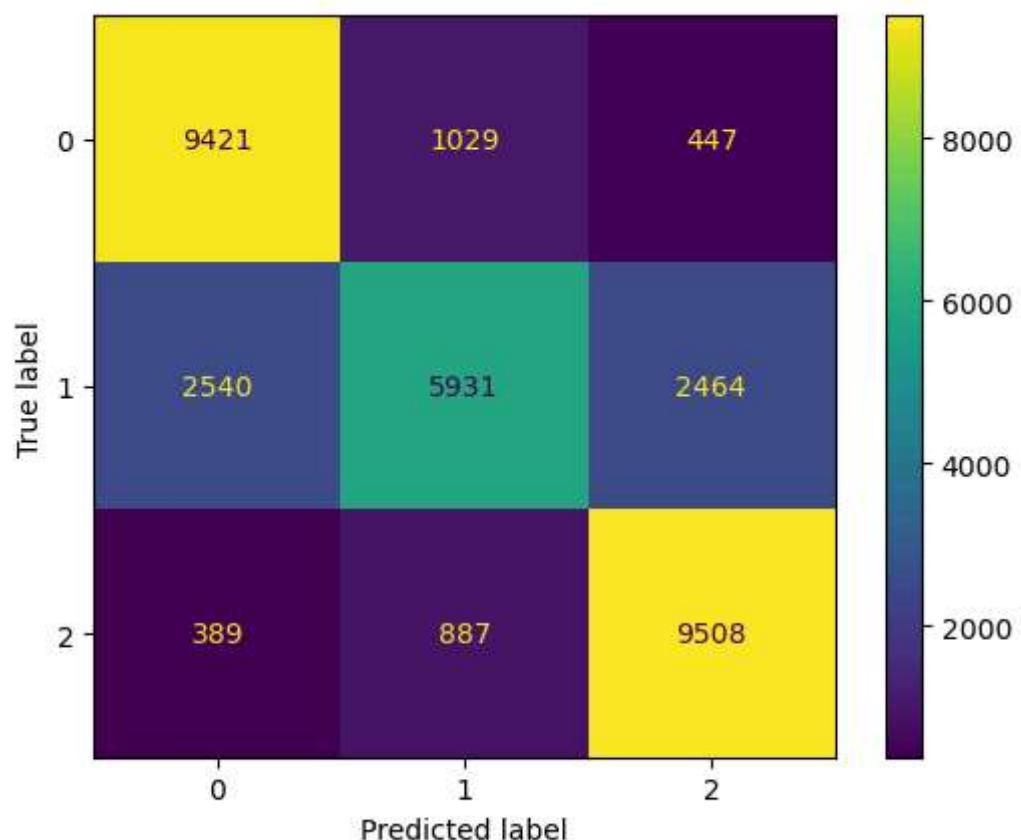
```
In [86]: def train_and_evaluate_model(model):
    model.fit(X_train,y_train)
    y_pred = model.predict(X_test)
    print(classification_report(y_test,y_pred))
    acc = accuracy_score(y_test,y_pred)
    precision = precision_score(y_test,y_pred,average='micro')
    recall = recall_score(y_test,y_pred,average='micro')
    f1 = f1_score(y_test,y_pred,average='micro')
    ConfusionMatrixDisplay.from_predictions(y_test,y_pred)
    plt.show();
    model_names.append(model)
    accuracy_scores.append(acc)
    precision_scores.append(precision)
    recall_scores.append(recall)
    f1_scores.append(f1)
```

```
In [87]: train_and_evaluate_model(LogisticRegression())
```



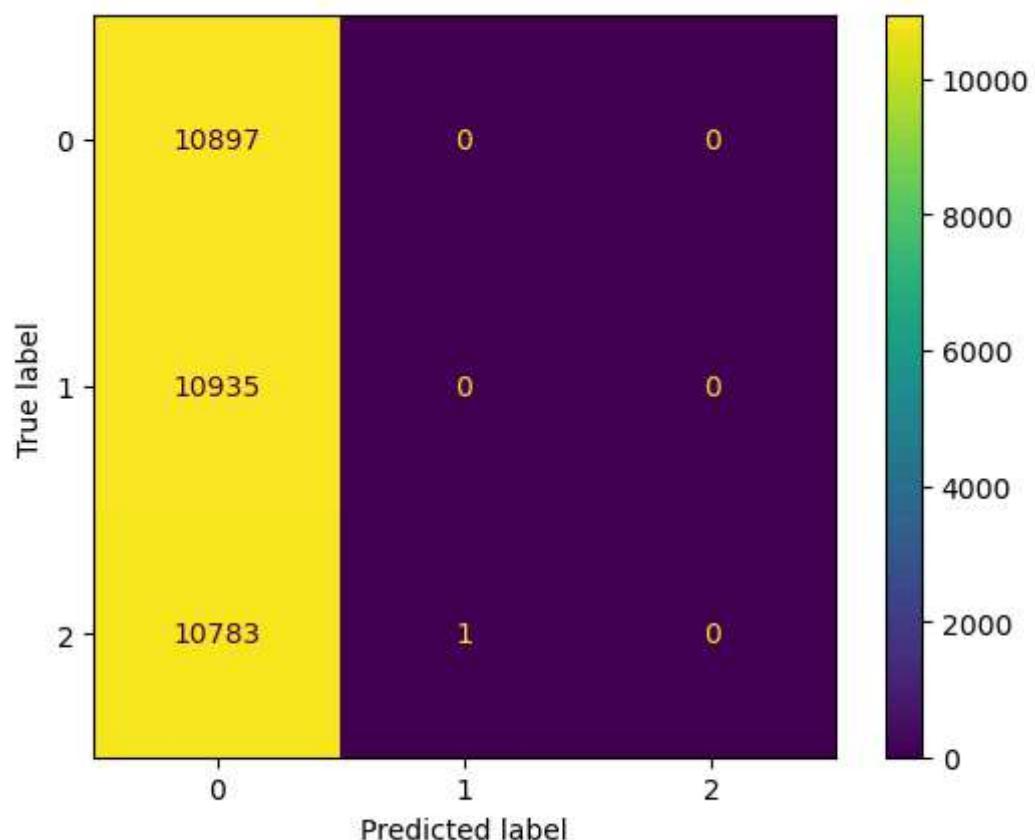
```
In [88]: train_and_evaluate_model(KNeighborsClassifier())
```

	precision	recall	f1-score	support
0	0.76	0.86	0.81	10897
1	0.76	0.54	0.63	10935
2	0.77	0.88	0.82	10784
accuracy			0.76	32616
macro avg	0.76	0.76	0.75	32616
weighted avg	0.76	0.76	0.75	32616



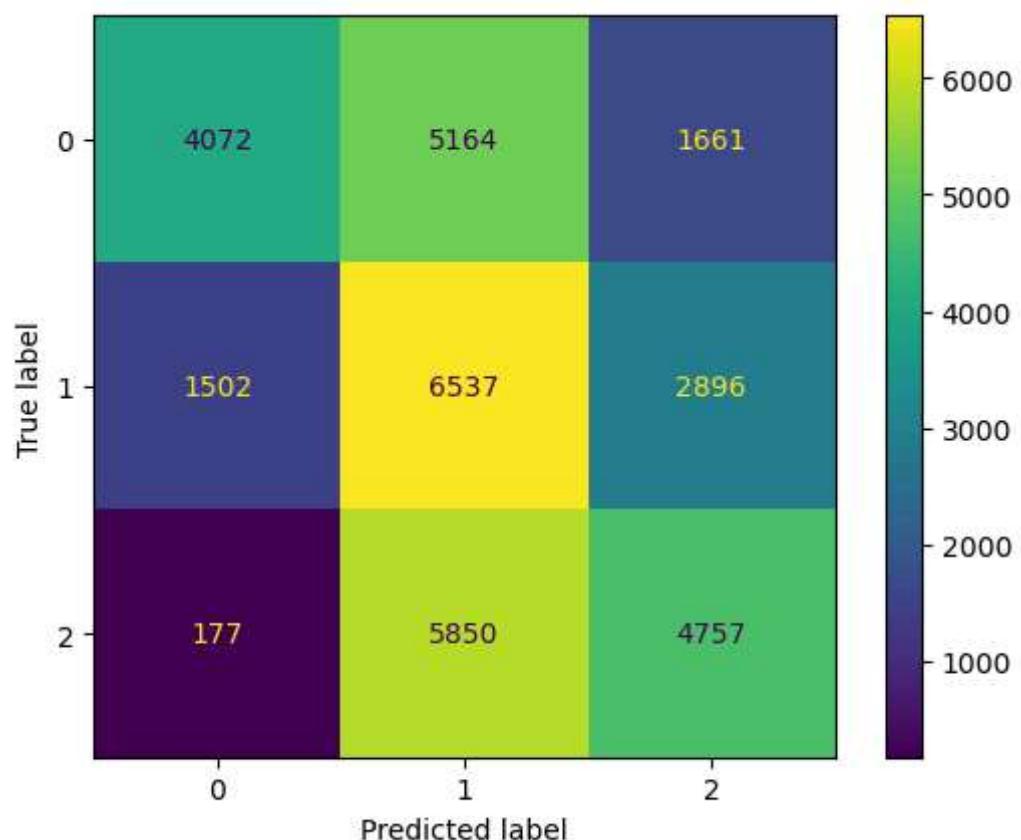
```
In [89]: train_and_evaluate_model(GaussianNB())
```

	precision	recall	f1-score	support
0	0.33	1.00	0.50	10897
1	0.00	0.00	0.00	10935
2	0.00	0.00	0.00	10784
accuracy			0.33	32616
macro avg	0.11	0.33	0.17	32616
weighted avg	0.11	0.33	0.17	32616

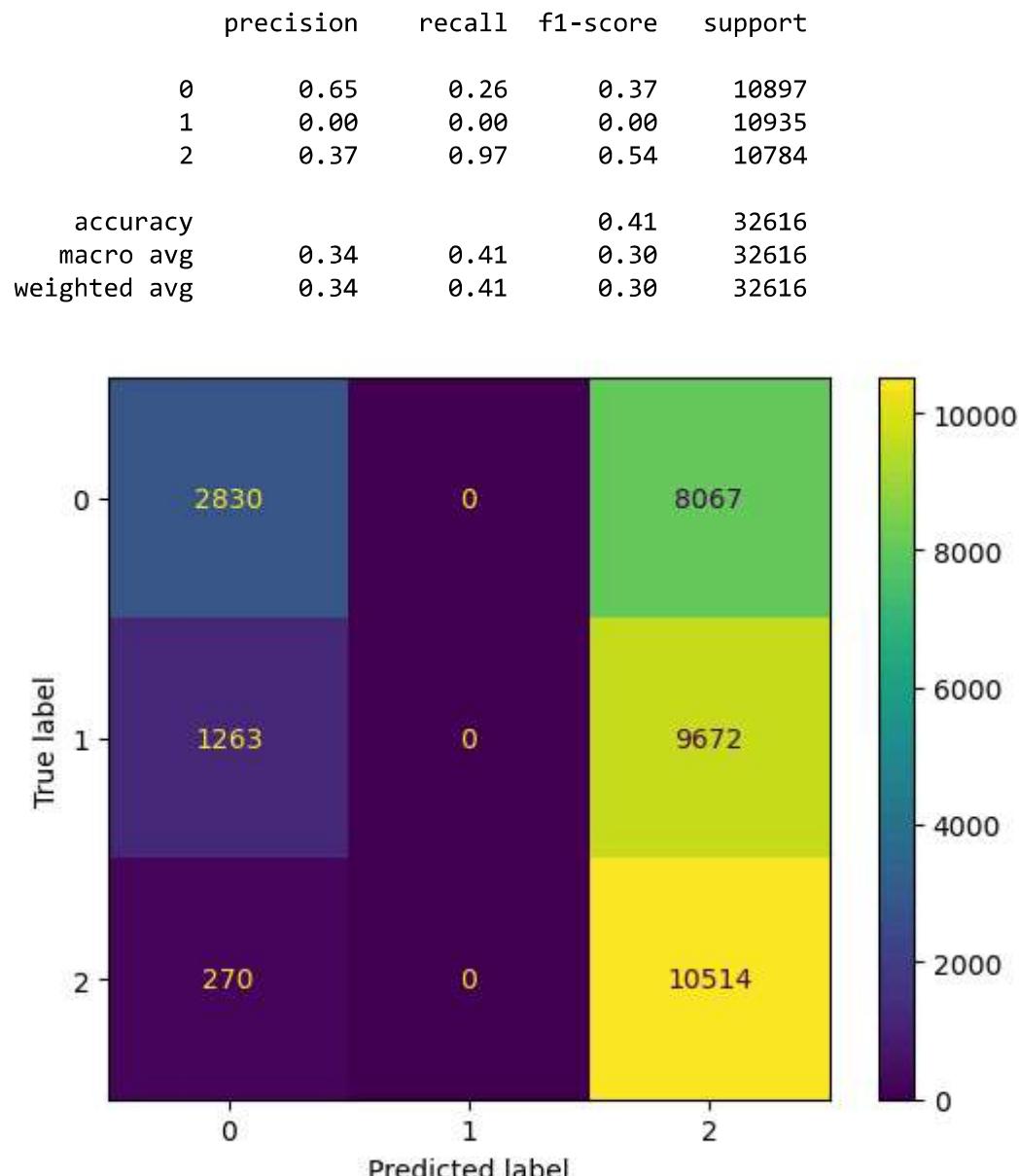


```
In [90]: train_and_evaluate_model(BernoulliNB())
```

	precision	recall	f1-score	support
0	0.71	0.37	0.49	10897
1	0.37	0.60	0.46	10935
2	0.51	0.44	0.47	10784
accuracy			0.47	32616
macro avg	0.53	0.47	0.47	32616
weighted avg	0.53	0.47	0.47	32616

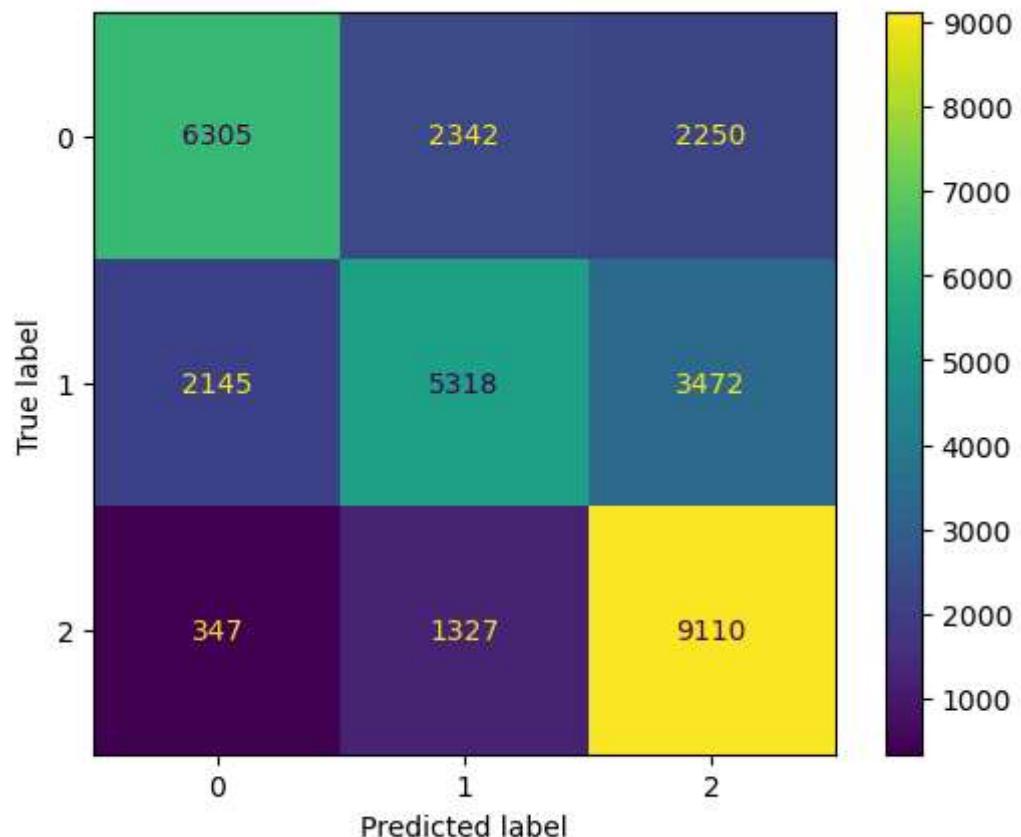


```
In [91]: train_and_evaluate_model(PassiveAggressiveClassifier())
```



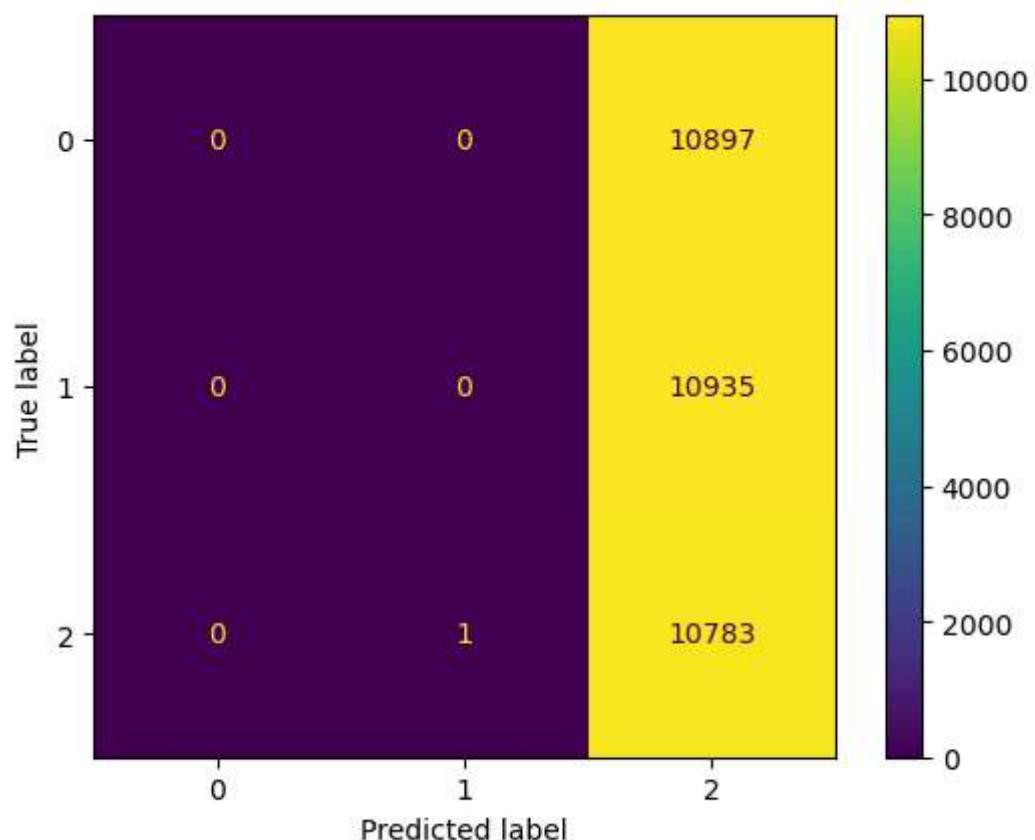
```
In [92]: train_and_evaluate_model(RidgeClassifier())
```

	precision	recall	f1-score	support
0	0.72	0.58	0.64	10897
1	0.59	0.49	0.53	10935
2	0.61	0.84	0.71	10784
accuracy			0.64	32616
macro avg	0.64	0.64	0.63	32616
weighted avg	0.64	0.64	0.63	32616



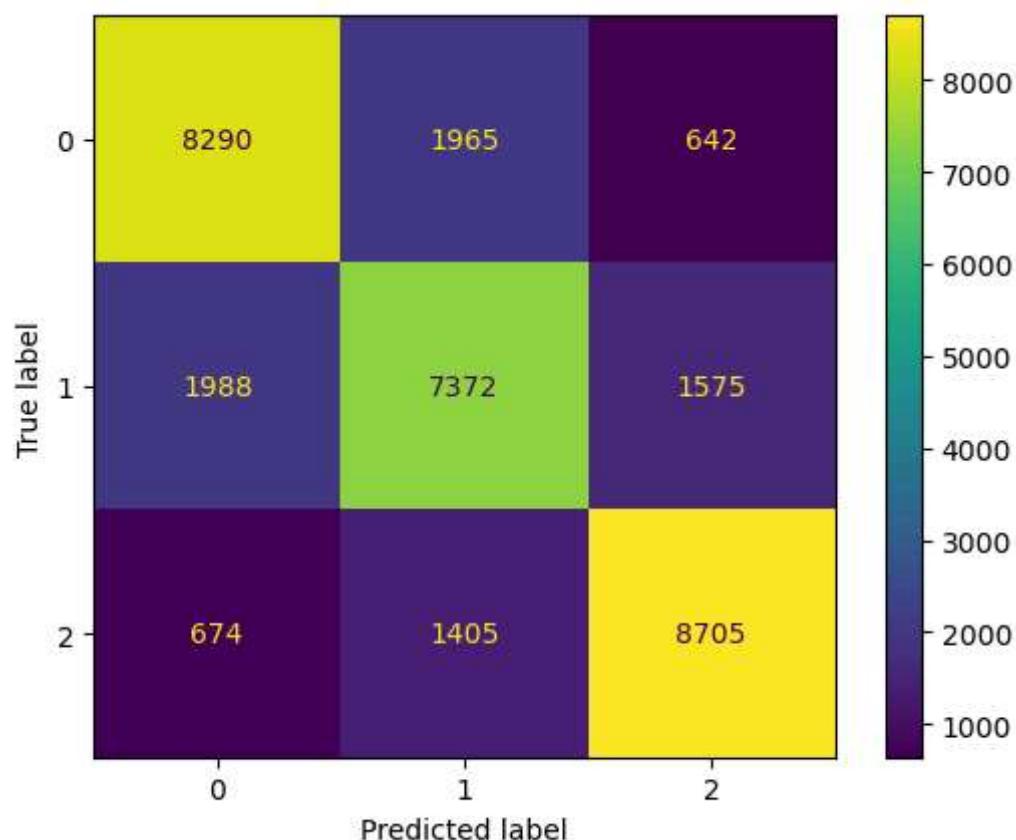
```
In [93]: train_and_evaluate_model(SGDClassifier())
```

	precision	recall	f1-score	support
0	0.00	0.00	0.00	10897
1	0.00	0.00	0.00	10935
2	0.33	1.00	0.50	10784
accuracy			0.33	32616
macro avg	0.11	0.33	0.17	32616
weighted avg	0.11	0.33	0.16	32616



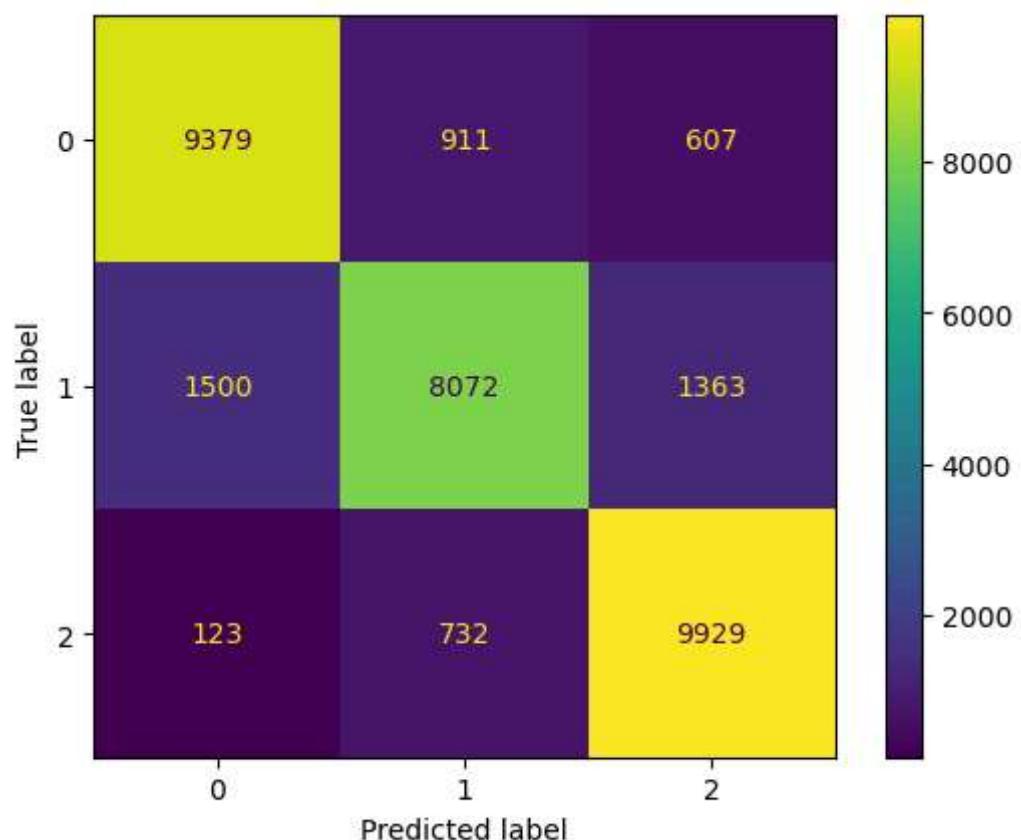
```
In [94]: train_and_evaluate_model(DecisionTreeClassifier())
```

	precision	recall	f1-score	support
0	0.76	0.76	0.76	10897
1	0.69	0.67	0.68	10935
2	0.80	0.81	0.80	10784
accuracy			0.75	32616
macro avg	0.75	0.75	0.75	32616
weighted avg	0.75	0.75	0.75	32616

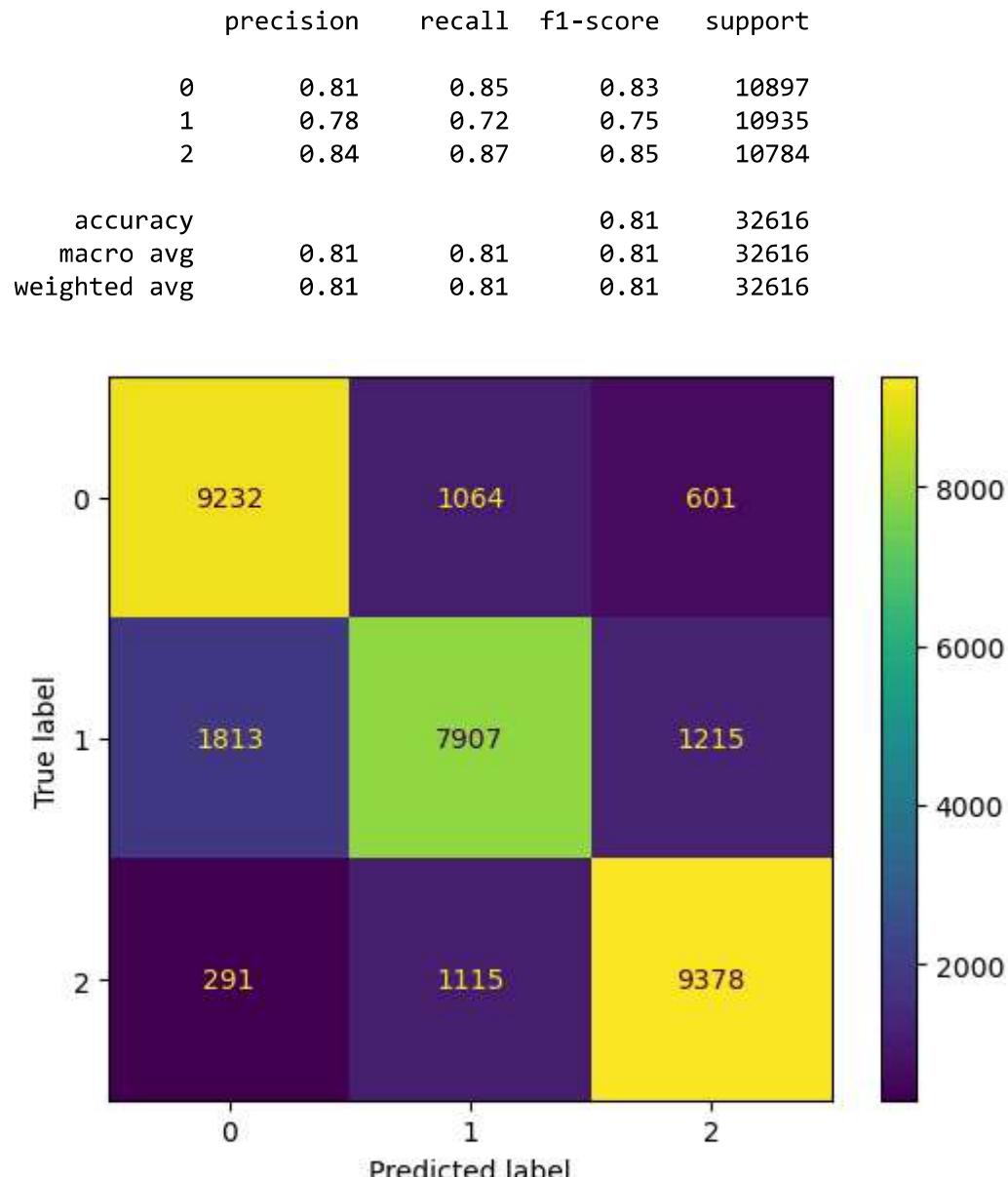


```
In [95]: train_and_evaluate_model(RandomForestClassifier())
```

	precision	recall	f1-score	support
0	0.85	0.86	0.86	10897
1	0.83	0.74	0.78	10935
2	0.83	0.92	0.88	10784
accuracy			0.84	32616
macro avg	0.84	0.84	0.84	32616
weighted avg	0.84	0.84	0.84	32616

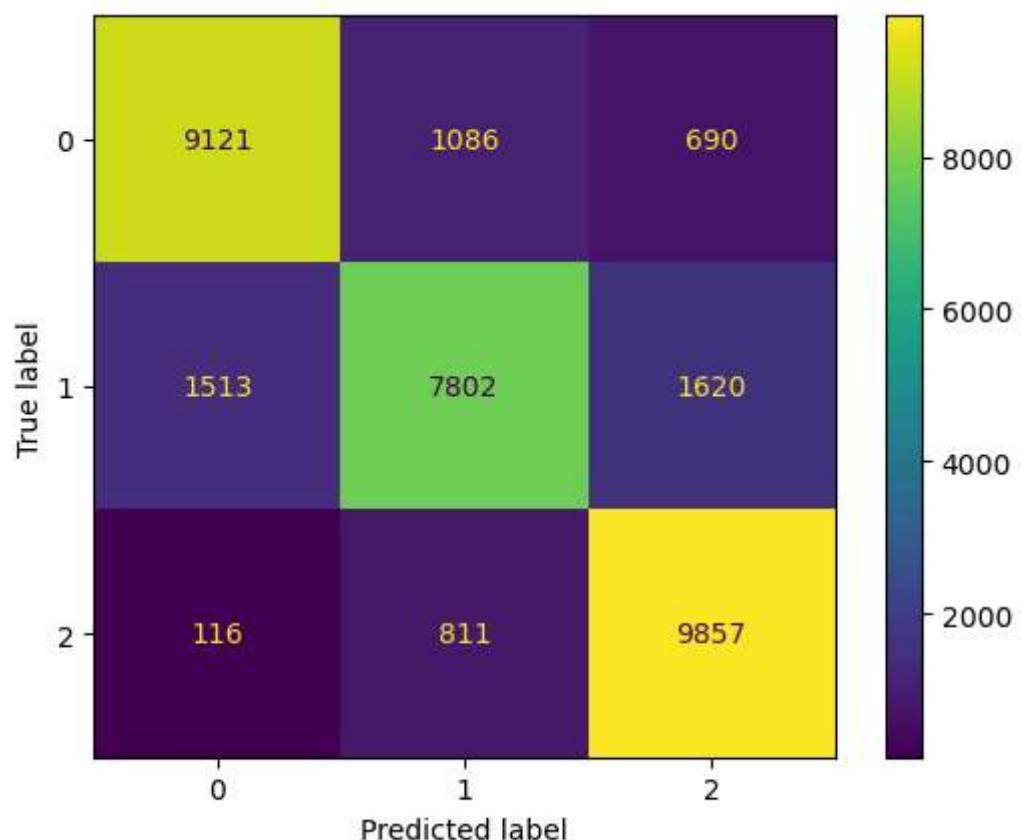


```
In [96]: train_and_evaluate_model(BaggingClassifier())
```



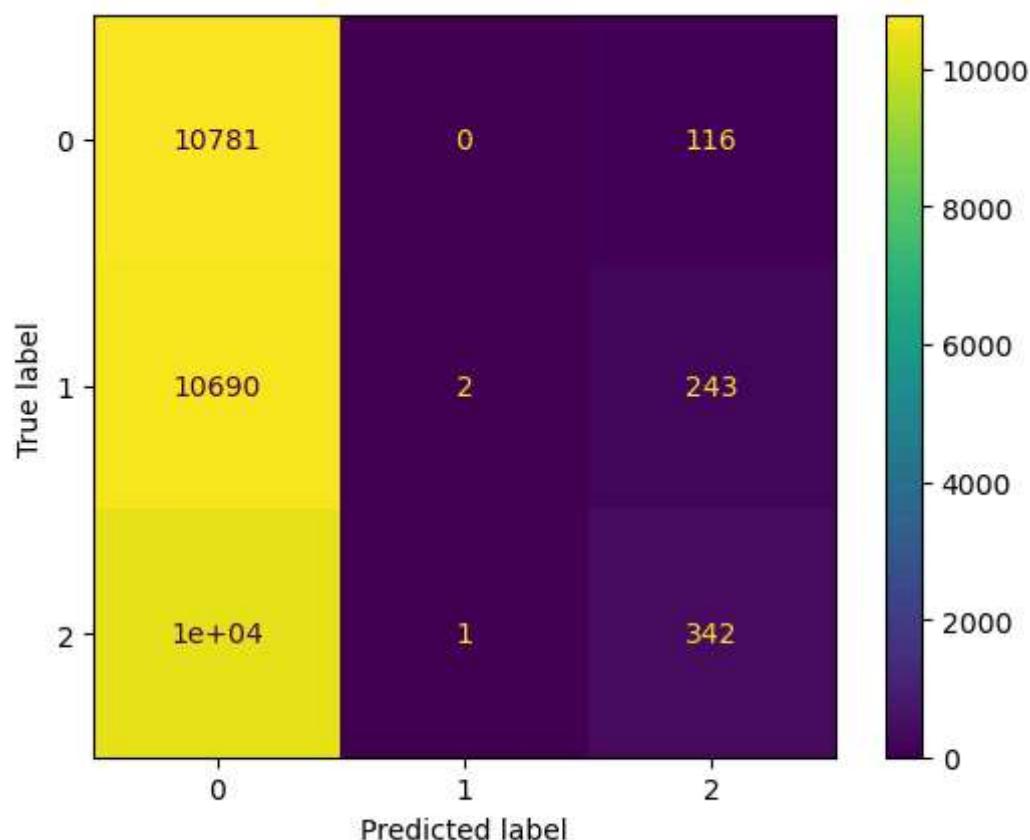
```
In [97]: train_and_evaluate_model(ExtraTreesClassifier())
```

	precision	recall	f1-score	support
0	0.85	0.84	0.84	10897
1	0.80	0.71	0.76	10935
2	0.81	0.91	0.86	10784
accuracy			0.82	32616
macro avg	0.82	0.82	0.82	32616
weighted avg	0.82	0.82	0.82	32616



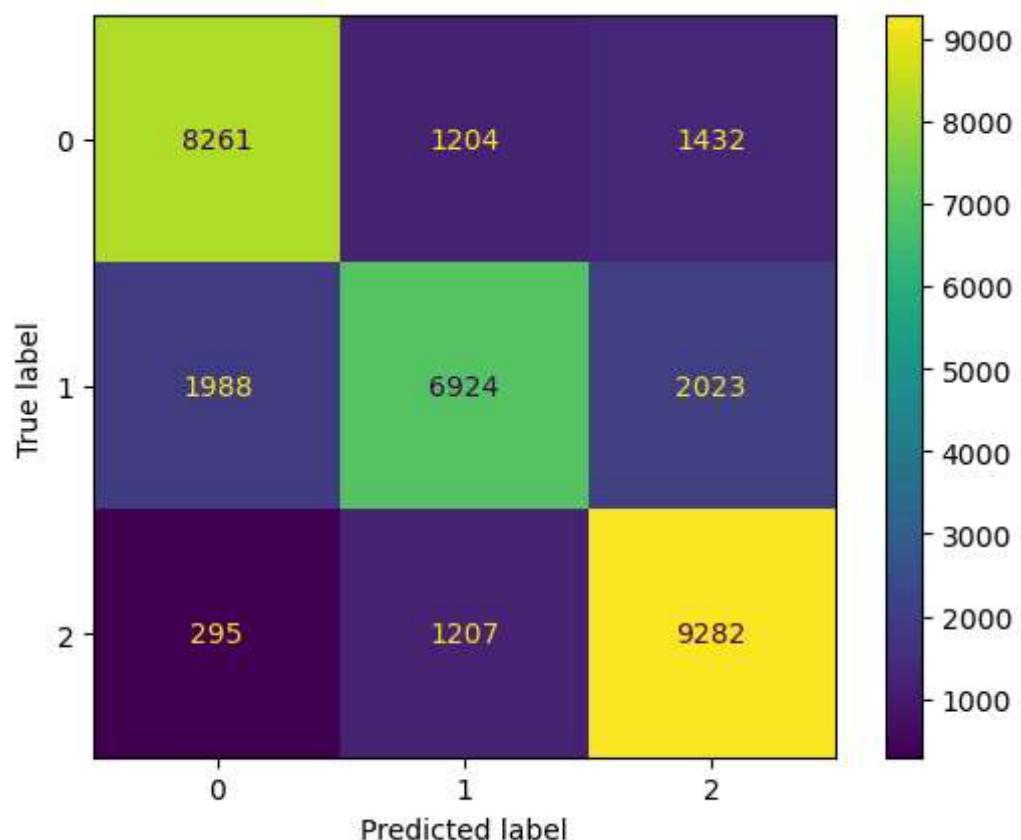
```
In [98]: train_and_evaluate_model(LinearSVC())
```

	precision	recall	f1-score	support
0	0.34	0.99	0.50	10897
1	0.67	0.00	0.00	10935
2	0.49	0.03	0.06	10784
accuracy			0.34	32616
macro avg	0.50	0.34	0.19	32616
weighted avg	0.50	0.34	0.19	32616



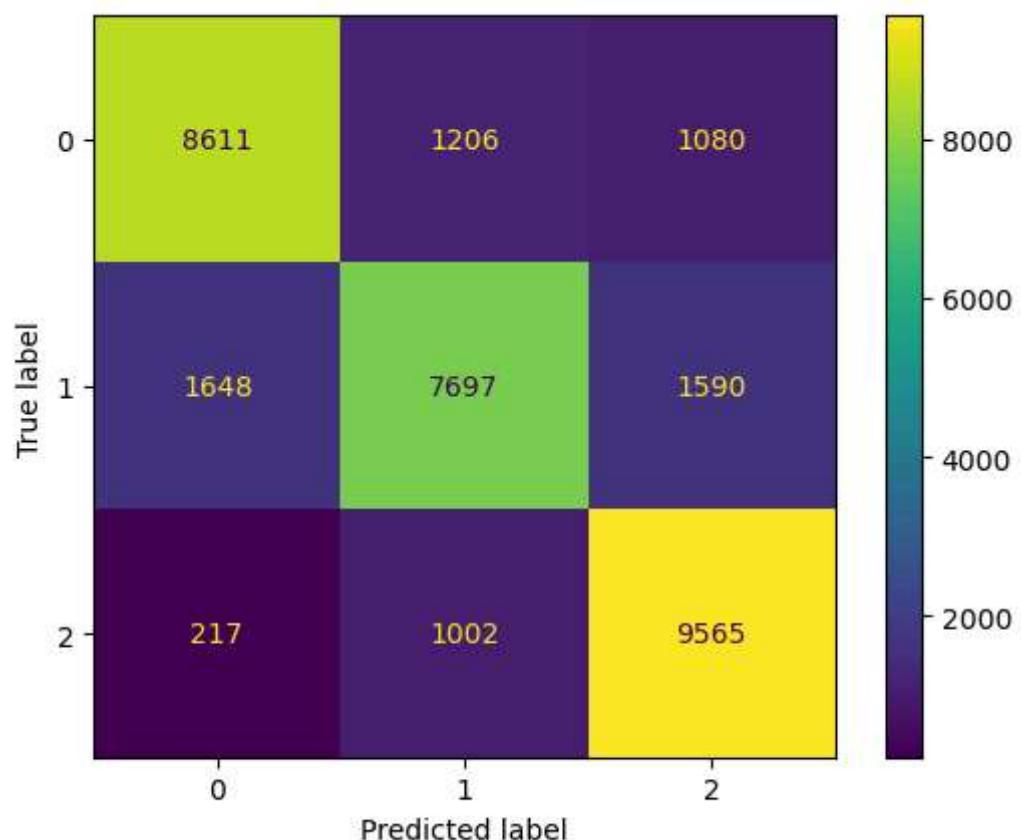
```
In [99]: train_and_evaluate_model(GradientBoostingClassifier())
```

	precision	recall	f1-score	support
0	0.78	0.76	0.77	10897
1	0.74	0.63	0.68	10935
2	0.73	0.86	0.79	10784
accuracy			0.75	32616
macro avg	0.75	0.75	0.75	32616
weighted avg	0.75	0.75	0.75	32616



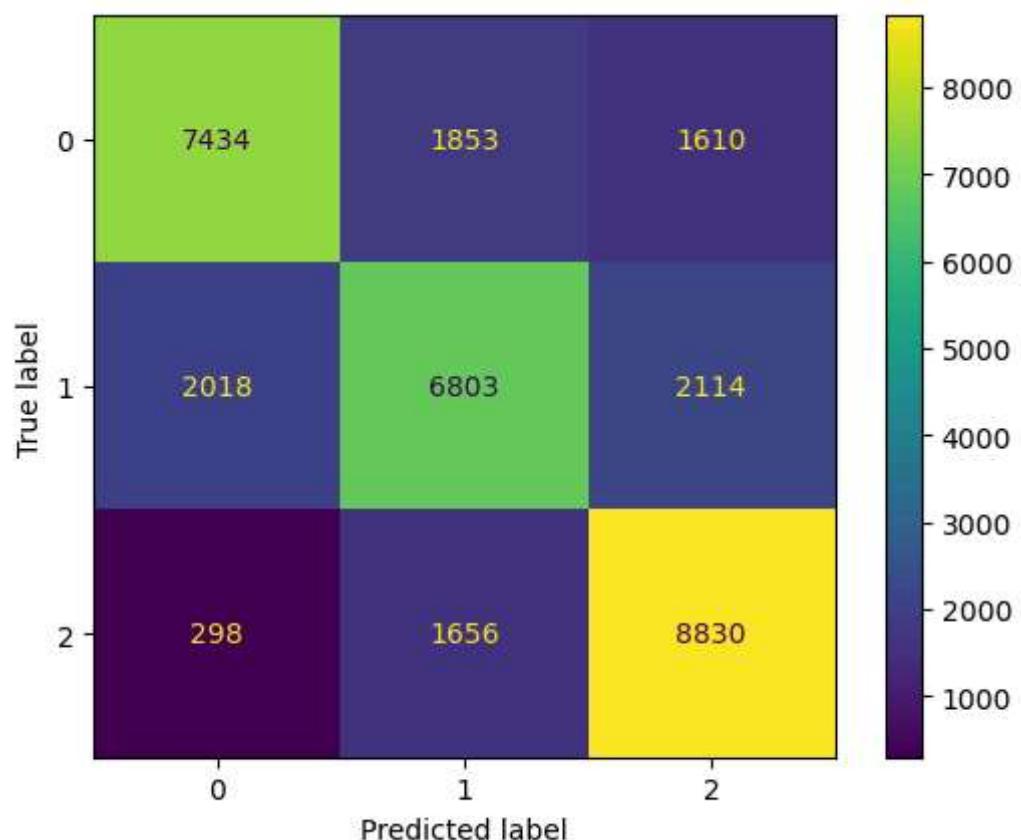
```
In [100]: train_and_evaluate_model(HistGradientBoostingClassifier())
```

	precision	recall	f1-score	support
0	0.82	0.79	0.81	10897
1	0.78	0.70	0.74	10935
2	0.78	0.89	0.83	10784
accuracy			0.79	32616
macro avg	0.79	0.79	0.79	32616
weighted avg	0.79	0.79	0.79	32616



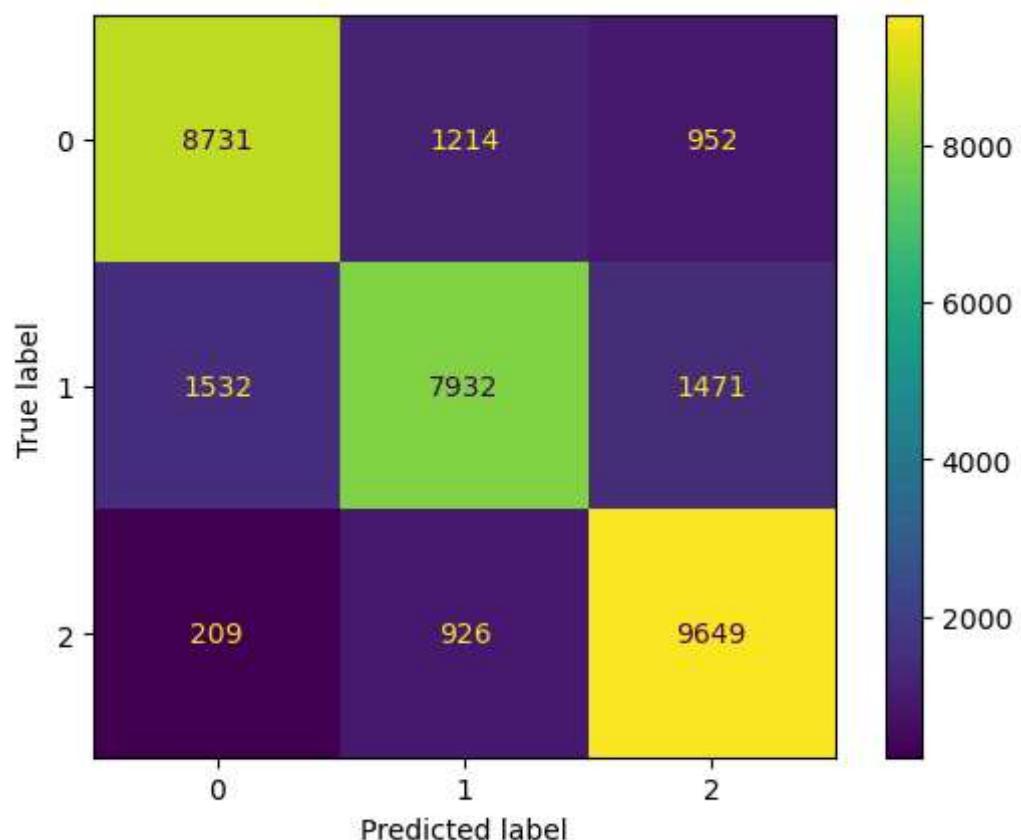
```
In [101]: train_and_evaluate_model(AdaBoostClassifier())
```

	precision	recall	f1-score	support
0	0.76	0.68	0.72	10897
1	0.66	0.62	0.64	10935
2	0.70	0.82	0.76	10784
accuracy			0.71	32616
macro avg	0.71	0.71	0.71	32616
weighted avg	0.71	0.71	0.71	32616



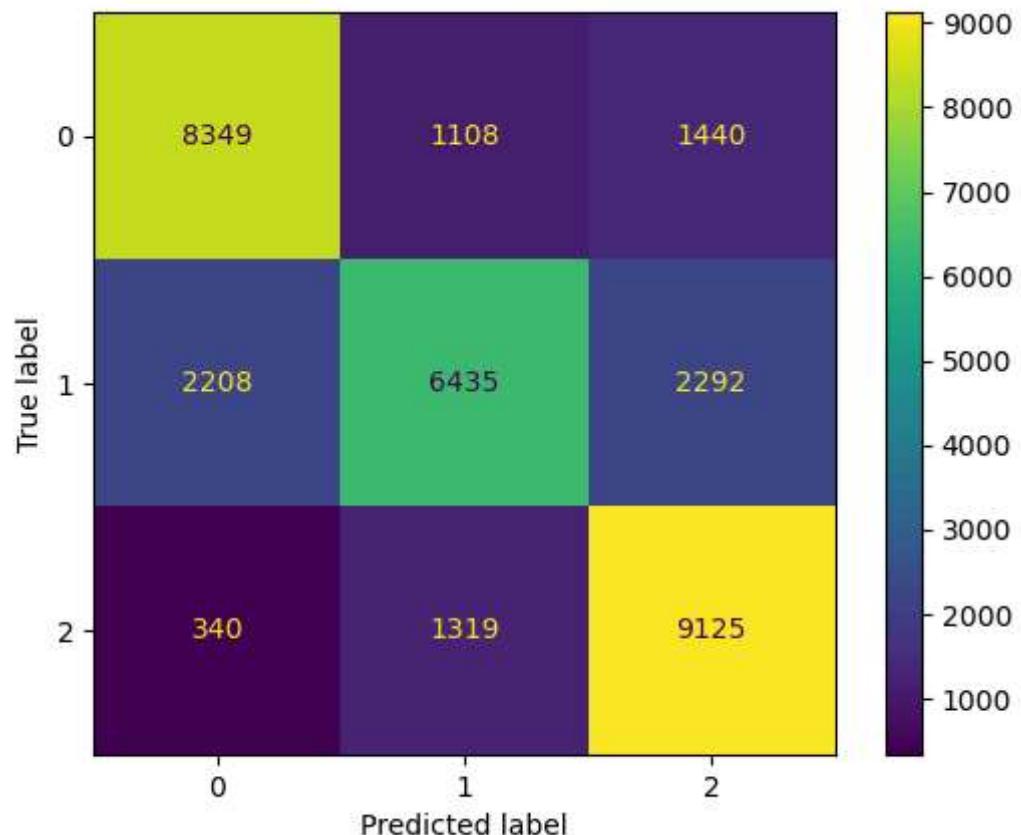
```
In [102]: train_and_evaluate_model(XGBClassifier())
```

	precision	recall	f1-score	support
0	0.83	0.80	0.82	10897
1	0.79	0.73	0.76	10935
2	0.80	0.89	0.84	10784
accuracy			0.81	32616
macro avg	0.81	0.81	0.81	32616
weighted avg	0.81	0.81	0.81	32616



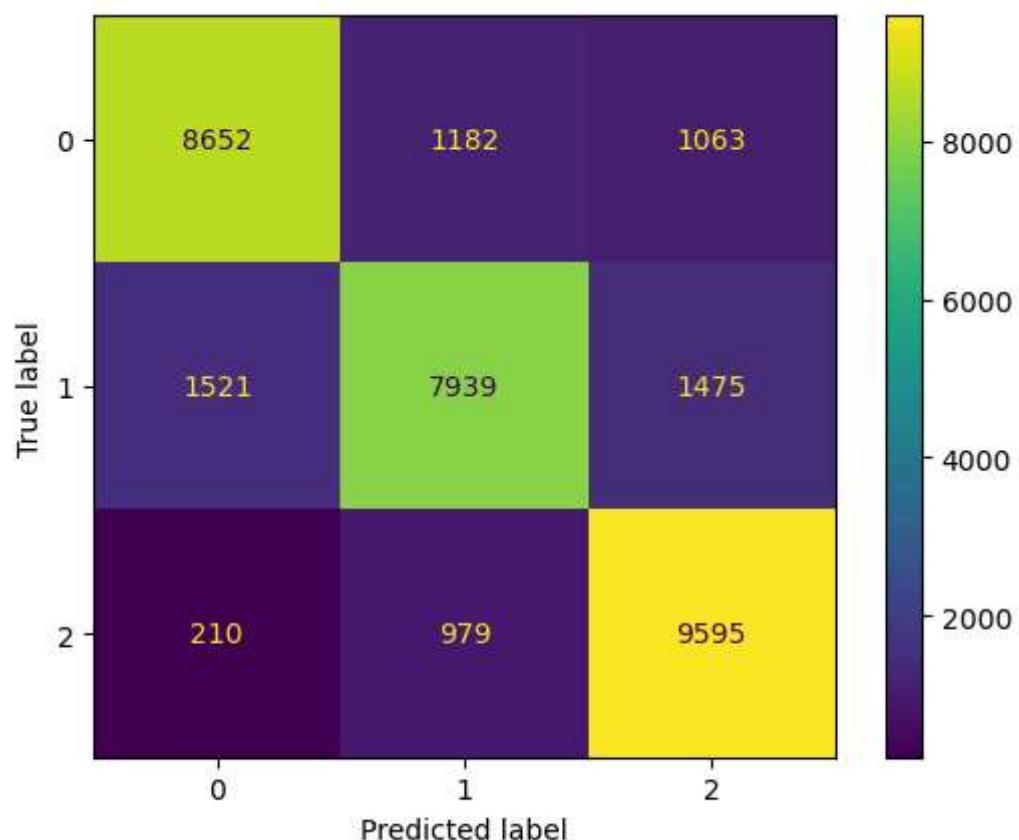
```
In [103]: train_and_evaluate_model(XGBRFClassifier())
```

	precision	recall	f1-score	support
0	0.77	0.77	0.77	10897
1	0.73	0.59	0.65	10935
2	0.71	0.85	0.77	10784
accuracy			0.73	32616
macro avg	0.73	0.73	0.73	32616
weighted avg	0.73	0.73	0.73	32616



```
In [104]: train_and_evaluate_model(CatBoostClassifier(silent=True))
```

	precision	recall	f1-score	support
0	0.83	0.79	0.81	10897
1	0.79	0.73	0.75	10935
2	0.79	0.89	0.84	10784
accuracy			0.80	32616
macro avg	0.80	0.80	0.80	32616
weighted avg	0.80	0.80	0.80	32616



```
In [105]: train_and_evaluate_model(LGBMClassifier())
```

```
[LightGBM] [Info] Auto-choosing col-wise multi-threading, the overhead of testing was 0.007383 seconds.
```

```
You can set `force_col_wise=true` to remove the overhead.
```

```
[LightGBM] [Info] Total Bins 3661
```

```
[LightGBM] [Info] Number of data points in the train set: 76101, number of used features: 18
```

```
[LightGBM] [Info] Start training from score -1.099598
```

```
[LightGBM] [Info] Start training from score -1.101099
```

```
[LightGBM] [Info] Start training from score -1.095149
```

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

0	0.82	0.77	0.79	10897
---	------	------	------	-------

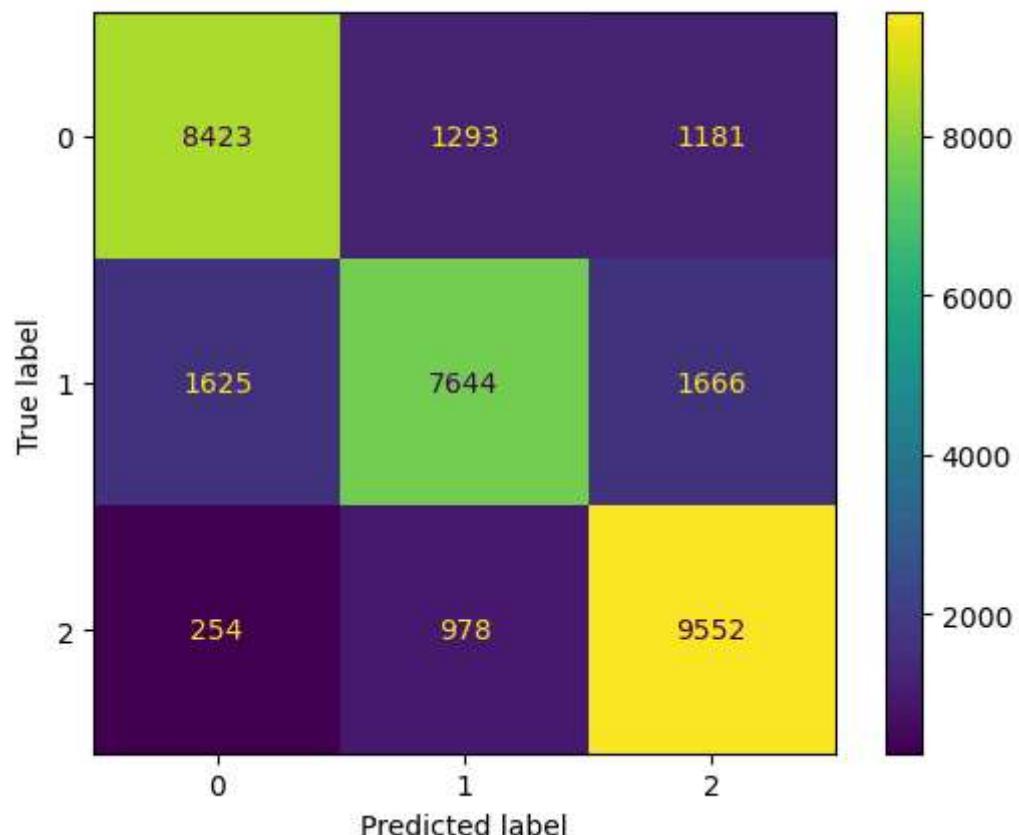
1	0.77	0.70	0.73	10935
---	------	------	------	-------

2	0.77	0.89	0.82	10784
---	------	------	------	-------

accuracy			0.79	32616
----------	--	--	------	-------

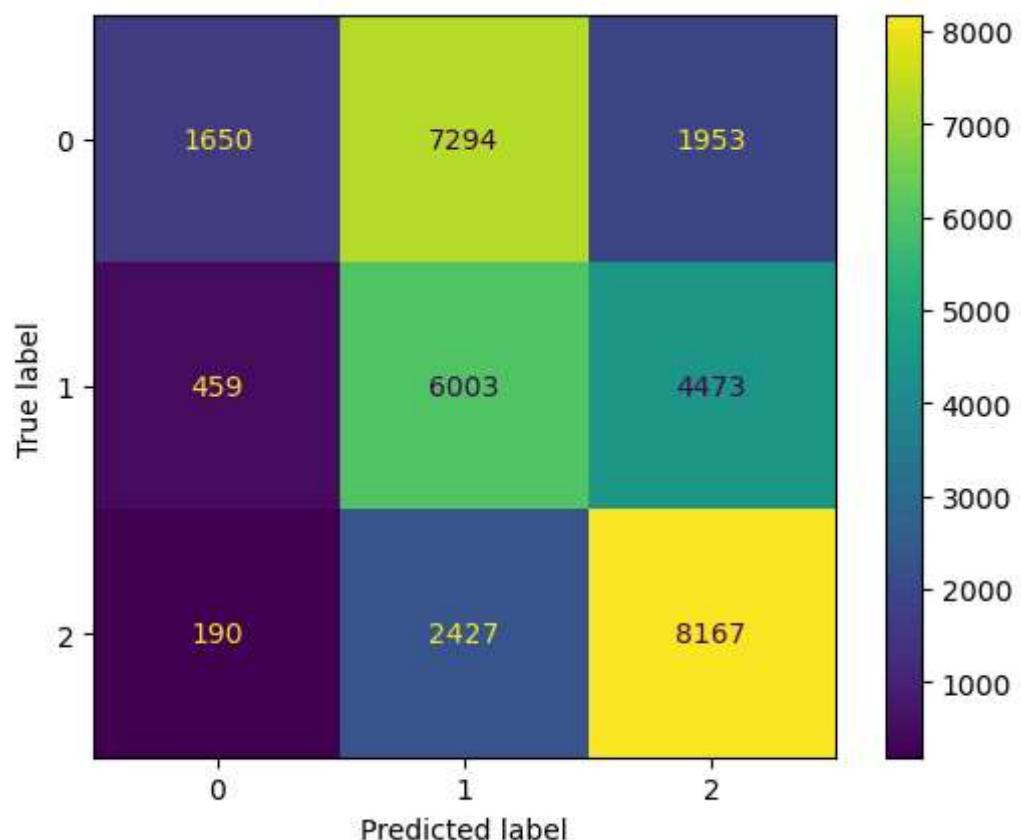
macro avg	0.79	0.79	0.78	32616
-----------	------	------	------	-------

weighted avg	0.79	0.79	0.78	32616
--------------	------	------	------	-------



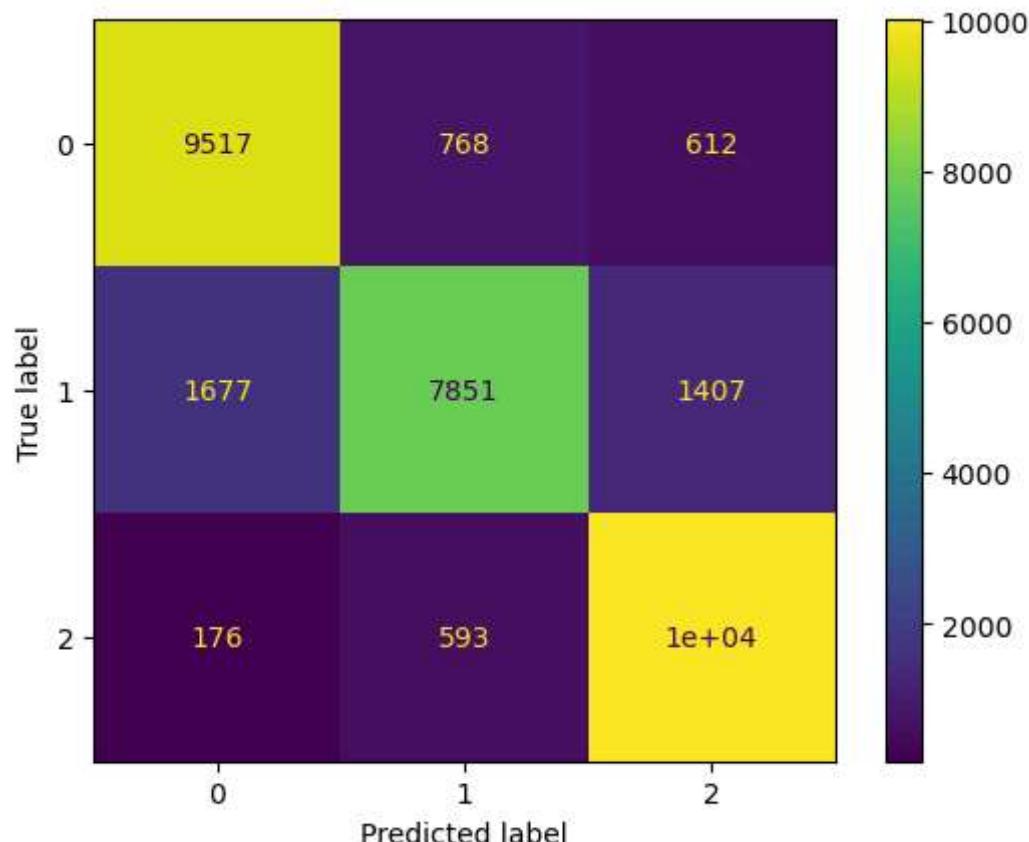
```
In [106]: train_and_evaluate_model(MLPClassifier())
```

	precision	recall	f1-score	support
0	0.72	0.15	0.25	10897
1	0.38	0.55	0.45	10935
2	0.56	0.76	0.64	10784
accuracy			0.49	32616
macro avg	0.55	0.49	0.45	32616
weighted avg	0.55	0.49	0.45	32616



```
In [107]: train_and_evaluate_model(VotingClassifier(estimators=[  
    ('RF', RandomForestClassifier()),  
    ('KNN', KNeighborsClassifier()),  
    ('HIST', HistGradientBoostingClassifier())  
], voting='hard'))
```

	precision	recall	f1-score	support
0	0.84	0.87	0.85	10897
1	0.85	0.72	0.78	10935
2	0.83	0.93	0.88	10784
accuracy			0.84	32616
macro avg	0.84	0.84	0.84	32616
weighted avg	0.84	0.84	0.84	32616



Checking score of models

Baseline Models Performance Comparison

```
In [108]: model_perfs = pd.DataFrame({'Model': model_names, 'Accuracy': accuracy_scores})
model_perfs
```

Out[108]:

		Model	Accuracy	Precision	Recall	F1
0	VotingClassifier(estimators=[('RF', RandomForest...)	0.839557	0.839557	0.839557	0.839557	0.839557
1	(DecisionTreeClassifier(max_features='sqrt', r...	0.839465	0.839465	0.839465	0.839465	0.839465
2	(ExtraTreeClassifier(random_state=1036040382),...	0.821069	0.821069	0.821069	0.821069	0.821069
3	(DecisionTreeClassifier(random_state=136957480...	0.813006	0.813006	0.813006	0.813006	0.813006
4	XGBClassifier(base_score=None, booster=None, c...	0.806721	0.806721	0.806721	0.806721	0.806721
5	<catboost.core.CatBoostClassifier object at 0x...	0.802857	0.802857	0.802857	0.802857	0.802857
6	HistGradientBoostingClassifier()	0.793261	0.793261	0.793261	0.793261	0.793261
7	LGBMClassifier()	0.785473	0.785473	0.785473	0.785473	0.785473
8	KNeighborsClassifier()	0.762203	0.762203	0.762203	0.762203	0.762203
9	(DecisionTreeRegressor(criterion='friedman_ms...	0.750153	0.750153	0.750153	0.750153	0.750153
10	DecisionTreeClassifier()	0.747087	0.747087	0.747087	0.747087	0.747087
11	XGBRFClassifier(base_score=None, booster=None,...	0.733045	0.733045	0.733045	0.733045	0.733045
12	(DecisionTreeClassifier(max_depth=1, random_st...	0.707230	0.707230	0.707230	0.707230	0.707230
13	RidgeClassifier()	0.635670	0.635670	0.635670	0.635670	0.635670
14	MLPClassifier()	0.485038	0.485038	0.485038	0.485038	0.485038
15	BernoulliNB()	0.471118	0.471118	0.471118	0.471118	0.471118
16	PassiveAggressiveClassifier()	0.409124	0.409124	0.409124	0.409124	0.409124
17	LinearSVC()	0.341090	0.341090	0.341090	0.341090	0.341090
18	LogisticRegression()	0.334130	0.334130	0.334130	0.334130	0.334130
19	GaussianNB()	0.334100	0.334100	0.334100	0.334100	0.334100
20	SGDClassifier()	0.330605	0.330605	0.330605	0.330605	0.330605

Hyperparameter Tuning using RandomizedSearchCV

```
In [109]: param_grid = {'learning_rate': [0.2,0.4,0.5,0.8,1.0],  
                      'loss': ['auto', 'binary_crossentropy', 'categorical_crossentropy']}  
grid_hgb = RandomizedSearchCV(HistGradientBoostingClassifier(),param_grid, cv=5)  
train_and_evaluate_model(grid_hgb)
```

```
Fitting 5 folds for each of 10 candidates, totalling 50 fits
[CV] END .....learning_rate=0.4, loss=auto; total time= 0.0s
[CV] END .....learning_rate=0.5, loss=auto; total time= 0.0s
```

```
In [110]: param_grid = {'penalty': ['l1', 'l2', 'elasticnet'],
                      'C': [0.001, 0.01, 0.1, 0.5],
                      'solver': ['newton-cg', 'lbfgs', 'liblinear', 'sag', 'saga'],
                      'multi_class': ['ovr', 'multinomial'],
                      'l1_ratio': [0.2, 0.5, 0.8]}
}

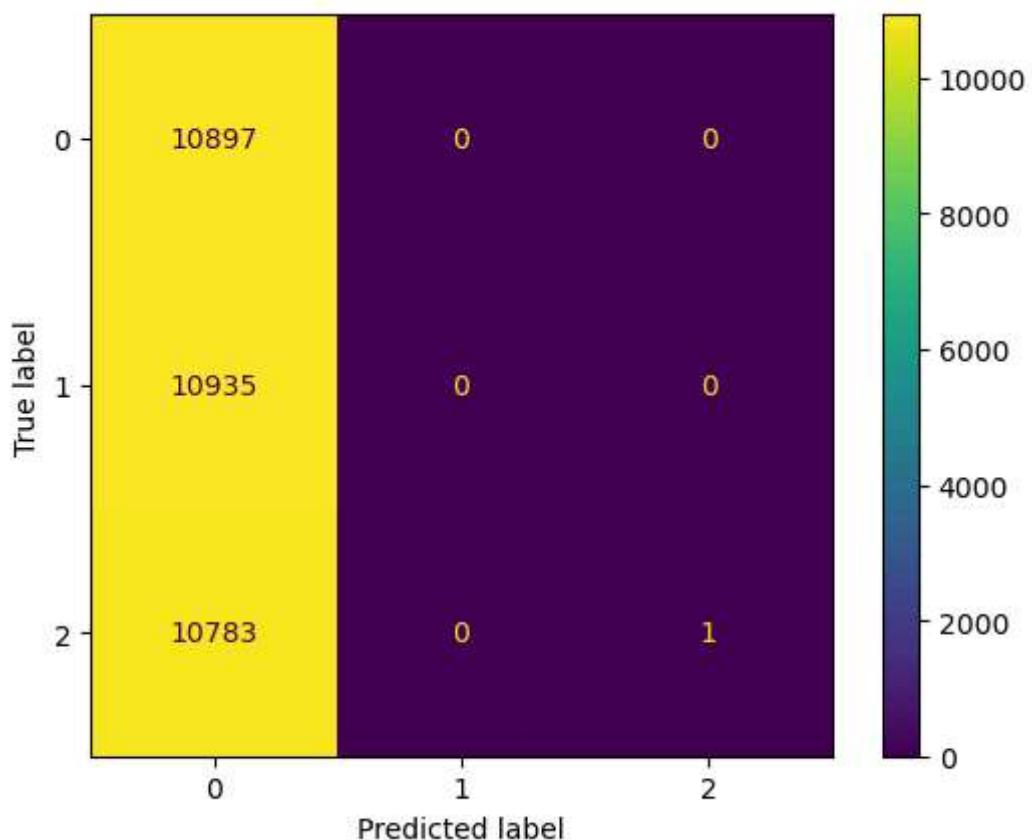
grid_lr = RandomizedSearchCV(LogisticRegression(), param_grid, verbose=2, cv=5)
train_and_evaluate_model(grid_lr)
```

Fitting 5 folds for each of 10 candidates, totalling 50 fits

```
[CV] END C=0.1, l1_ratio=0.5, multi_class=multinomial, penalty=l2, solver=liblinear; total time= 0.0s
[CV] END C=0.1, l1_ratio=0.5, multi_class=multinomial, penalty=l2, solver=liblinear; total time= 0.0s
[CV] END C=0.1, l1_ratio=0.5, multi_class=multinomial, penalty=l2, solver=liblinear; total time= 0.0s
[CV] END C=0.1, l1_ratio=0.5, multi_class=multinomial, penalty=l2, solver=liblinear; total time= 0.0s
[CV] END C=0.1, l1_ratio=0.5, multi_class=multinomial, penalty=l2, solver=liblinear; total time= 0.0s
[CV] END C=0.5, l1_ratio=0.8, multi_class=ovr, penalty=l2, solver=lbfgs; total time= 0.4s
[CV] END C=0.5, l1_ratio=0.8, multi_class=ovr, penalty=l2, solver=lbfgs; total time= 0.4s
[CV] END C=0.5, l1_ratio=0.8, multi_class=ovr, penalty=l2, solver=lbfgs; total time= 0.5s
[CV] END C=0.5, l1_ratio=0.8, multi_class=ovr, penalty=l2, solver=lbfgs; total time= 0.4s
[CV] END C=0.5, l1_ratio=0.8, multi_class=ovr, penalty=l2, solver=lbfgs; total time= 0.4s
[CV] END C=0.5, l1_ratio=0.8, multi_class=multinomial, penalty=l1, solver=liblinear; total time= 0.0s
[CV] END C=0.5, l1_ratio=0.8, multi_class=multinomial, penalty=l1, solver=liblinear; total time= 0.0s
[CV] END C=0.5, l1_ratio=0.8, multi_class=multinomial, penalty=l1, solver=liblinear; total time= 0.0s
[CV] END C=0.5, l1_ratio=0.8, multi_class=multinomial, penalty=l1, solver=liblinear; total time= 0.0s
[CV] END C=0.5, l1_ratio=0.8, multi_class=multinomial, penalty=l1, solver=liblinear; total time= 0.0s
[CV] END C=0.01, l1_ratio=0.8, multi_class=ovr, penalty=elasticnet, solve_r=sag; total time= 0.0s
[CV] END C=0.01, l1_ratio=0.8, multi_class=ovr, penalty=elasticnet, solve_r=sag; total time= 0.0s
[CV] END C=0.01, l1_ratio=0.8, multi_class=ovr, penalty=elasticnet, solve_r=sag; total time= 0.0s
[CV] END C=0.01, l1_ratio=0.8, multi_class=ovr, penalty=elasticnet, solve_r=sag; total time= 0.0s
[CV] END C=0.01, l1_ratio=0.8, multi_class=ovr, penalty=elasticnet, solve_r=sag; total time= 0.0s
[CV] END C=0.01, l1_ratio=0.8, multi_class=ovr, penalty=elasticnet, solve_r=sag; total time= 0.0s
[CV] END C=0.01, l1_ratio=0.8, multi_class=ovr, penalty=elasticnet, solve_r=sag; total time= 0.0s
[CV] END C=0.01, l1_ratio=0.8, multi_class=ovr, penalty=elasticnet, solve_r=sag; total time= 0.0s
[CV] END C=0.5, l1_ratio=0.8, multi_class=ovr, penalty=elasticnet, solver=liblinear; total time= 0.0s
[CV] END C=0.5, l1_ratio=0.8, multi_class=ovr, penalty=elasticnet, solver=liblinear; total time= 0.0s
[CV] END C=0.5, l1_ratio=0.8, multi_class=ovr, penalty=elasticnet, solver=liblinear; total time= 0.0s
[CV] END C=0.5, l1_ratio=0.8, multi_class=ovr, penalty=elasticnet, solver=liblinear; total time= 0.0s
[CV] END C=0.5, l1_ratio=0.8, multi_class=ovr, penalty=elasticnet, solver=liblinear; total time= 0.0s
```

```
[CV] END C=0.5, l1_ratio=0.5, multi_class=ovr, penalty=elasticnet, solver=liblinear; total time= 0.0s
[CV] END C=0.5, l1_ratio=0.5, multi_class=ovr, penalty=elasticnet, solver=liblinear; total time= 0.0s
[CV] END C=0.5, l1_ratio=0.5, multi_class=ovr, penalty=elasticnet, solver=liblinear; total time= 0.0s
[CV] END C=0.5, l1_ratio=0.5, multi_class=ovr, penalty=elasticnet, solver=liblinear; total time= 0.0s
[CV] END C=0.5, l1_ratio=0.5, multi_class=ovr, penalty=elasticnet, solver=liblinear; total time= 0.0s
[CV] END C=0.01, l1_ratio=0.8, multi_class=ovr, penalty=elasticnet, solve
r=newton-cg; total time= 0.0s
[CV] END C=0.01, l1_ratio=0.8, multi_class=ovr, penalty=elasticnet, solve
r=newton-cg; total time= 0.0s
[CV] END C=0.01, l1_ratio=0.8, multi_class=ovr, penalty=elasticnet, solve
r=newton-cg; total time= 0.0s
[CV] END C=0.01, l1_ratio=0.8, multi_class=ovr, penalty=elasticnet, solve
r=newton-cg; total time= 0.0s
[CV] END C=0.01, l1_ratio=0.8, multi_class=ovr, penalty=elasticnet, solve
r=newton-cg; total time= 0.0s
[CV] END C=0.001, l1_ratio=0.2, multi_class=multinomial, penalty=l1, solv
er=sag; total time= 0.0s
[CV] END C=0.001, l1_ratio=0.2, multi_class=multinomial, penalty=l1, solv
er=sag; total time= 0.0s
[CV] END C=0.001, l1_ratio=0.2, multi_class=multinomial, penalty=l1, solv
er=sag; total time= 0.0s
[CV] END C=0.001, l1_ratio=0.2, multi_class=multinomial, penalty=l1, solv
er=sag; total time= 0.0s
[CV] END C=0.001, l1_ratio=0.2, multi_class=multinomial, penalty=l1, solv
er=sag; total time= 0.0s
[CV] END C=0.001, l1_ratio=0.8, multi_class=multinomial, penalty=elasticn
et, solver=lbfgs; total time= 0.0s
[CV] END C=0.001, l1_ratio=0.8, multi_class=multinomial, penalty=elasticn
et, solver=lbfgs; total time= 0.0s
[CV] END C=0.001, l1_ratio=0.8, multi_class=multinomial, penalty=elasticn
et, solver=lbfgs; total time= 0.0s
[CV] END C=0.001, l1_ratio=0.8, multi_class=multinomial, penalty=elasticn
et, solver=lbfgs; total time= 0.0s
[CV] END C=0.001, l1_ratio=0.8, multi_class=multinomial, penalty=elasticn
et, solver=lbfgs; total time= 0.0s
```

	precision	recall	f1-score	support
0	0.33	1.00	0.50	10897
1	0.00	0.00	0.00	10935
2	1.00	0.00	0.00	10784
accuracy			0.33	32616
macro avg	0.44	0.33	0.17	32616
weighted avg	0.44	0.33	0.17	32616



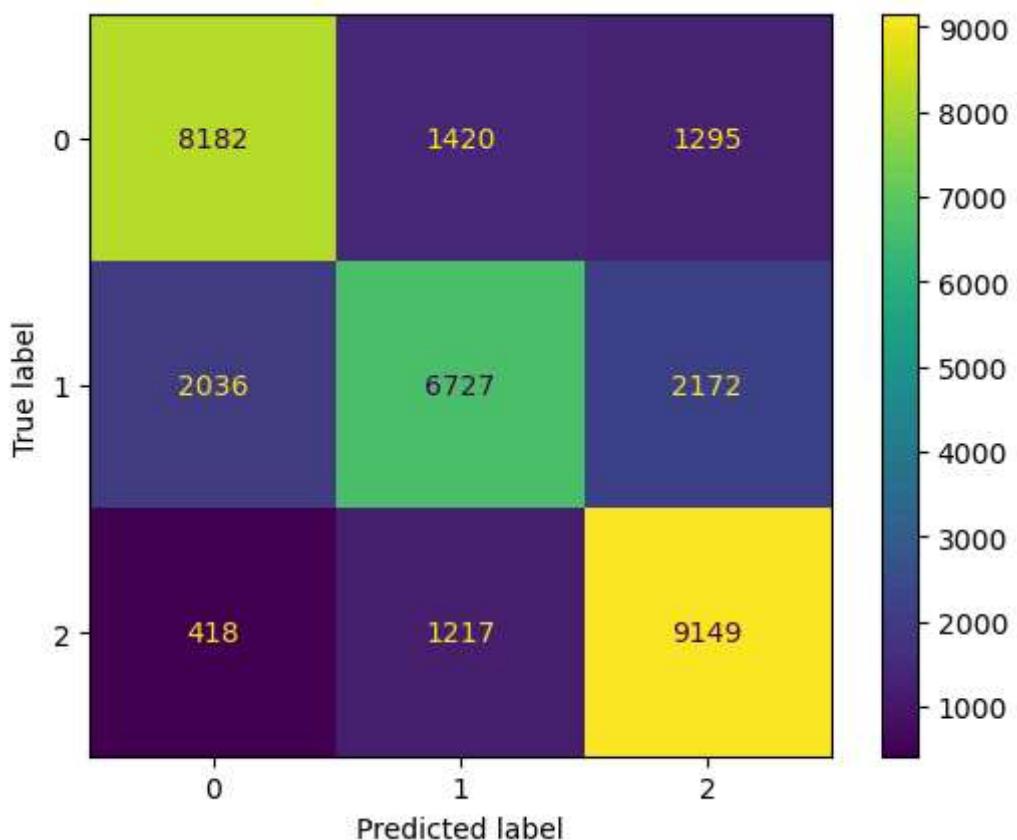
```
In [111]: param_grid = {'criterion': ['gini', 'entropy'],
                      'max_features': ['sqrt', 'log2'],
                      'max_depth': [2, 12, 38, 68, 98, 128, 201]}
                    }

grid_dt = RandomizedSearchCV(DecisionTreeClassifier(), param_grid, verbose=4)
train_and_evaluate_model(grid_dt)
```

```
Fitting 5 folds for each of 10 candidates, totalling 50 fits
[CV 1/5] END criterion=entropy, max_depth=12, max_features=sqrt;, score=0.737 total time= 0.4s
[CV 2/5] END criterion=entropy, max_depth=12, max_features=sqrt;, score=0.731 total time= 0.5s
[CV 3/5] END criterion=entropy, max_depth=12, max_features=sqrt;, score=0.735 total time= 0.4s
[CV 4/5] END criterion=entropy, max_depth=12, max_features=sqrt;, score=0.731 total time= 0.4s
[CV 5/5] END criterion=entropy, max_depth=12, max_features=sqrt;, score=0.718 total time= 0.4s
[CV 1/5] END criterion=gini, max_depth=2, max_features=sqrt;, score=0.616 total time= 0.0s
[CV 2/5] END criterion=gini, max_depth=2, max_features=sqrt;, score=0.625 total time= 0.0s
[CV 3/5] END criterion=gini, max_depth=2, max_features=sqrt;, score=0.617 total time= 0.0s
[CV 4/5] END criterion=gini, max_depth=2, max_features=sqrt;, score=0.653 total time= 0.0s
[CV 5/5] END criterion=gini, max_depth=2, max_features=sqrt;, score=0.559 total time= 0.0s
[CV 1/5] END criterion=entropy, max_depth=12, max_features=log2;, score=0.728 total time= 0.4s
[CV 2/5] END criterion=entropy, max_depth=12, max_features=log2;, score=0.725 total time= 0.4s
[CV 3/5] END criterion=entropy, max_depth=12, max_features=log2;, score=0.735 total time= 0.4s
[CV 4/5] END criterion=entropy, max_depth=12, max_features=log2;, score=0.733 total time= 0.4s
[CV 5/5] END criterion=entropy, max_depth=12, max_features=log2;, score=0.724 total time= 0.4s
[CV 1/5] END criterion=entropy, max_depth=201, max_features=log2;, score=0.730 total time= 0.8s
[CV 2/5] END criterion=entropy, max_depth=201, max_features=log2;, score=0.735 total time= 0.7s
[CV 3/5] END criterion=entropy, max_depth=201, max_features=log2;, score=0.734 total time= 0.7s
[CV 4/5] END criterion=entropy, max_depth=201, max_features=log2;, score=0.737 total time= 0.7s
[CV 5/5] END criterion=entropy, max_depth=201, max_features=log2;, score=0.729 total time= 0.7s
[CV 1/5] END criterion=entropy, max_depth=2, max_features=sqrt;, score=0.629 total time= 0.0s
[CV 2/5] END criterion=entropy, max_depth=2, max_features=sqrt;, score=0.592 total time= 0.0s
[CV 3/5] END criterion=entropy, max_depth=2, max_features=sqrt;, score=0.593 total time= 0.0s
[CV 4/5] END criterion=entropy, max_depth=2, max_features=sqrt;, score=0.655 total time= 0.1s
[CV 5/5] END criterion=entropy, max_depth=2, max_features=sqrt;, score=0.609 total time= 0.0s
[CV 1/5] END criterion=entropy, max_depth=68, max_features=sqrt;, score=0.739 total time= 0.7s
[CV 2/5] END criterion=entropy, max_depth=68, max_features=sqrt;, score=0.731 total time= 0.7s
[CV 3/5] END criterion=entropy, max_depth=68, max_features=sqrt;, score=0.735 total time= 0.7s
[CV 4/5] END criterion=entropy, max_depth=68, max_features=sqrt;, score=0.725 total time= 0.8s
[CV 5/5] END criterion=entropy, max_depth=68, max_features=sqrt;, score=0.732 total time= 0.7s
```

```
[CV 1/5] END criterion=entropy, max_depth=128, max_features=log2;, score=0.730 total time= 0.7s
[CV 2/5] END criterion=entropy, max_depth=128, max_features=log2;, score=0.725 total time= 0.7s
[CV 3/5] END criterion=entropy, max_depth=128, max_features=log2;, score=0.737 total time= 0.8s
[CV 4/5] END criterion=entropy, max_depth=128, max_features=log2;, score=0.732 total time= 0.7s
[CV 5/5] END criterion=entropy, max_depth=128, max_features=log2;, score=0.723 total time= 0.8s
[CV 1/5] END criterion=gini, max_depth=98, max_features=sqrt;, score=0.741 total time= 0.5s
[CV 2/5] END criterion=gini, max_depth=98, max_features=sqrt;, score=0.735 total time= 0.5s
[CV 3/5] END criterion=gini, max_depth=98, max_features=sqrt;, score=0.739 total time= 0.5s
[CV 4/5] END criterion=gini, max_depth=98, max_features=sqrt;, score=0.726 total time= 0.5s
[CV 5/5] END criterion=gini, max_depth=98, max_features=sqrt;, score=0.725 total time= 0.6s
[CV 1/5] END criterion=entropy, max_depth=38, max_features=sqrt;, score=0.729 total time= 0.8s
[CV 2/5] END criterion=entropy, max_depth=38, max_features=sqrt;, score=0.731 total time= 0.7s
[CV 3/5] END criterion=entropy, max_depth=38, max_features=sqrt;, score=0.733 total time= 0.8s
[CV 4/5] END criterion=entropy, max_depth=38, max_features=sqrt;, score=0.734 total time= 0.7s
[CV 5/5] END criterion=entropy, max_depth=38, max_features=sqrt;, score=0.718 total time= 0.8s
[CV 1/5] END criterion=gini, max_depth=12, max_features=log2;, score=0.735 total time= 0.3s
[CV 2/5] END criterion=gini, max_depth=12, max_features=log2;, score=0.734 total time= 0.3s
[CV 3/5] END criterion=gini, max_depth=12, max_features=log2;, score=0.738 total time= 0.3s
[CV 4/5] END criterion=gini, max_depth=12, max_features=log2;, score=0.732 total time= 0.3s
[CV 5/5] END criterion=gini, max_depth=12, max_features=log2;, score=0.728 total time= 0.3s
```

	precision	recall	f1-score	support
0	0.77	0.75	0.76	10897
1	0.72	0.62	0.66	10935
2	0.73	0.85	0.78	10784
accuracy			0.74	32616
macro avg	0.74	0.74	0.73	32616
weighted avg	0.74	0.74	0.73	32616



```
In [112]: param_grid = {'boosting_type': ['gbdt', 'dart', 'goss', 'rf'],
                     'learning_rate': np.linspace(0, 1, 6)[1:],
                     'n_estimators': [200, 500, 600, 1000],
                     'importance_type': ['split', 'gain'],
                     'min_split_gain': [0.68, 0.79, 0.87, 1]}

grid_lgbm = RandomizedSearchCV(LGBMClassifier(), param_grid, verbose=3)
train_and_evaluate_model(grid_lgbm)
```

Fitting 5 folds for each of 10 candidates, totalling 50 fits
[LightGBM] [Info] Auto-choosing col-wise multi-threading, the overhead of testing was 0.012778 seconds.
You can set `force_col_wise=true` to remove the overhead.
[LightGBM] [Info] Total Bins 3660
[LightGBM] [Info] Number of data points in the train set: 60880, number of used features: 18
[LightGBM] [Info] Start training from score -1.099615
[LightGBM] [Info] Start training from score -1.101096
[LightGBM] [Info] Start training from score -1.095136
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf

```
In [113]: param_grid = {'alpha': np.linspace(0,1,6),
                      'binarize': np.linspace(0,1,5),
                      'fit_prior': [True, False]}
                    }

grid_bnb = RandomizedSearchCV(BernoulliNB(), param_grid, verbose=3, cv=5)
train_and_evaluate_model(grid_bnb)
```

Fitting 5 folds for each of 10 candidates, totalling 50 fits

[CV 1/5] END alpha=1.0, binarize=0.5, fit_prior=False;, score=0.480 total time= 0.0s

[CV 2/5] END alpha=1.0, binarize=0.5, fit_prior=False;, score=0.481 total time= 0.0s

[CV 3/5] END alpha=1.0, binarize=0.5, fit_prior=False;, score=0.477 total time= 0.0s

[CV 4/5] END alpha=1.0, binarize=0.5, fit_prior=False;, score=0.480 total time= 0.0s

[CV 5/5] END alpha=1.0, binarize=0.5, fit_prior=False;, score=0.479 total time= 0.0s

[CV 1/5] END alpha=0.8, binarize=1.0, fit_prior=False;, score=0.599 total time= 0.0s

[CV 2/5] END alpha=0.8, binarize=1.0, fit_prior=False;, score=0.592 total time= 0.0s

[CV 3/5] END alpha=0.8, binarize=1.0, fit_prior=False;, score=0.600 total time= 0.0s

[CV 4/5] END alpha=0.8, binarize=1.0, fit_prior=False;, score=0.600 total time= 0.0s

[CV 5/5] END alpha=0.8, binarize=1.0, fit_prior=False;, score=0.589 total time= 0.0s

[CV 1/5] END alpha=0.4, binarize=0.5, fit_prior=False;, score=0.480 total time= 0.0s

[CV 2/5] END alpha=0.4, binarize=0.5, fit_prior=False;, score=0.481 total time= 0.0s

[CV 3/5] END alpha=0.4, binarize=0.5, fit_prior=False;, score=0.477 total time= 0.0s

[CV 4/5] END alpha=0.4, binarize=0.5, fit_prior=False;, score=0.480 total time= 0.0s

[CV 5/5] END alpha=0.4, binarize=0.5, fit_prior=False;, score=0.479 total time= 0.0s

[CV 1/5] END alpha=0.6000000000000001, binarize=0.0, fit_prior=True;, score=0.473 total time= 0.0s

[CV 2/5] END alpha=0.6000000000000001, binarize=0.0, fit_prior=True;, score=0.475 total time= 0.0s

[CV 3/5] END alpha=0.6000000000000001, binarize=0.0, fit_prior=True;, score=0.470 total time= 0.0s

[CV 4/5] END alpha=0.6000000000000001, binarize=0.0, fit_prior=True;, score=0.472 total time= 0.0s

[CV 5/5] END alpha=0.6000000000000001, binarize=0.0, fit_prior=True;, score=0.471 total time= 0.0s

[CV 1/5] END alpha=0.0, binarize=0.25, fit_prior=False;, score=0.333 total time= 0.0s

[CV 2/5] END alpha=0.0, binarize=0.25, fit_prior=False;, score=0.333 total time= 0.0s

[CV 3/5] END alpha=0.0, binarize=0.25, fit_prior=False;, score=0.333 total time= 0.0s

[CV 4/5] END alpha=0.0, binarize=0.25, fit_prior=False;, score=0.333 total time= 0.0s

[CV 5/5] END alpha=0.0, binarize=0.25, fit_prior=False;, score=0.333 total time= 0.0s

[CV 1/5] END alpha=0.6000000000000001, binarize=0.5, fit_prior=False;, score=0.480 total time= 0.0s

[CV 2/5] END alpha=0.6000000000000001, binarize=0.5, fit_prior=False;, score=0.481 total time= 0.0s

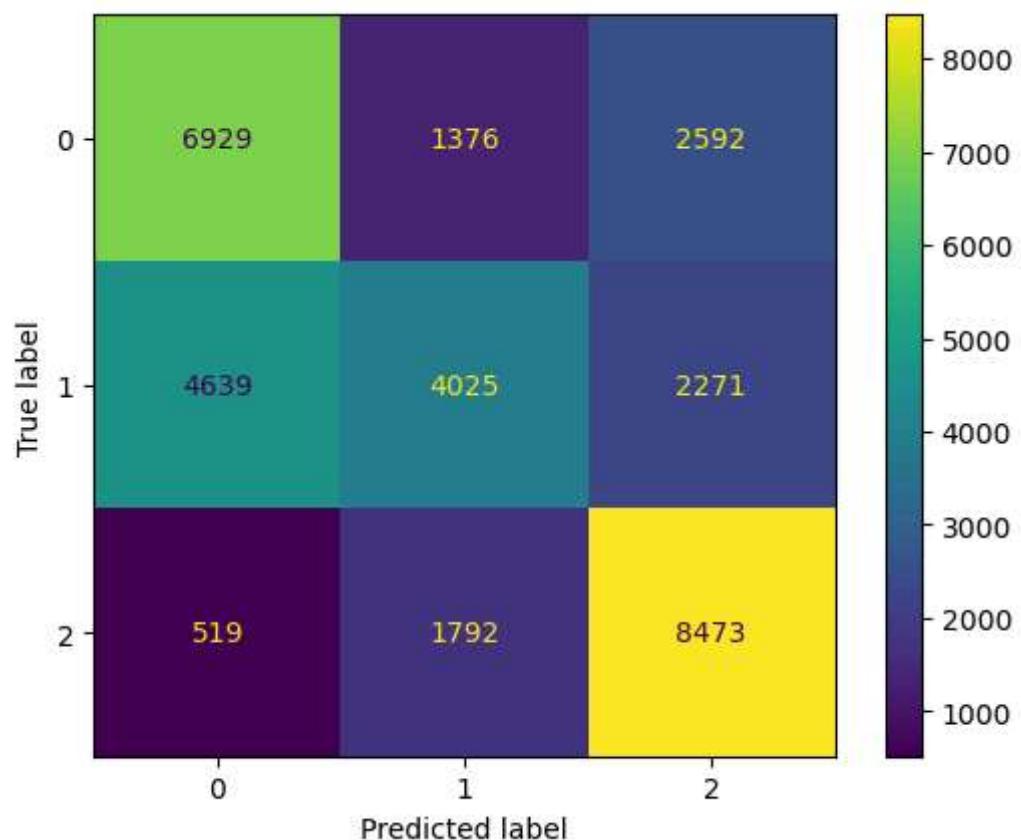
[CV 3/5] END alpha=0.6000000000000001, binarize=0.5, fit_prior=False;, score=0.477 total time= 0.0s

[CV 4/5] END alpha=0.6000000000000001, binarize=0.5, fit_prior=False;, score=0.480 total time= 0.0s

[CV 5/5] END alpha=0.6000000000000001, binarize=0.5, fit_prior=False;, score=0.479 total time= 0.0s

```
[CV 1/5] END alpha=0.8, binarize=0.75, fit_prior=False;, score=0.485 total time= 0.0s
[CV 2/5] END alpha=0.8, binarize=0.75, fit_prior=False;, score=0.486 total time= 0.0s
[CV 3/5] END alpha=0.8, binarize=0.75, fit_prior=False;, score=0.483 total time= 0.0s
[CV 4/5] END alpha=0.8, binarize=0.75, fit_prior=False;, score=0.487 total time= 0.0s
[CV 5/5] END alpha=0.8, binarize=0.75, fit_prior=False;, score=0.484 total time= 0.0s
[CV 1/5] END alpha=0.8, binarize=0.25, fit_prior=True;, score=0.475 total time= 0.0s
[CV 2/5] END alpha=0.8, binarize=0.25, fit_prior=True;, score=0.476 total time= 0.0s
[CV 3/5] END alpha=0.8, binarize=0.25, fit_prior=True;, score=0.472 total time= 0.0s
[CV 4/5] END alpha=0.8, binarize=0.25, fit_prior=True;, score=0.476 total time= 0.0s
[CV 5/5] END alpha=0.8, binarize=0.25, fit_prior=True;, score=0.474 total time= 0.0s
[CV 1/5] END alpha=0.0, binarize=0.75, fit_prior=True;, score=0.333 total time= 0.0s
[CV 2/5] END alpha=0.0, binarize=0.75, fit_prior=True;, score=0.333 total time= 0.0s
[CV 3/5] END alpha=0.0, binarize=0.75, fit_prior=True;, score=0.333 total time= 0.0s
[CV 4/5] END alpha=0.0, binarize=0.75, fit_prior=True;, score=0.333 total time= 0.0s
[CV 5/5] END alpha=0.0, binarize=0.75, fit_prior=True;, score=0.333 total time= 0.0s
[CV 1/5] END alpha=0.2, binarize=0.25, fit_prior=True;, score=0.475 total time= 0.0s
[CV 2/5] END alpha=0.2, binarize=0.25, fit_prior=True;, score=0.476 total time= 0.0s
[CV 3/5] END alpha=0.2, binarize=0.25, fit_prior=True;, score=0.472 total time= 0.0s
[CV 4/5] END alpha=0.2, binarize=0.25, fit_prior=True;, score=0.476 total time= 0.0s
[CV 5/5] END alpha=0.2, binarize=0.25, fit_prior=True;, score=0.474 total time= 0.0s
```

	precision	recall	f1-score	support
0	0.57	0.64	0.60	10897
1	0.56	0.37	0.44	10935
2	0.64	0.79	0.70	10784
accuracy			0.60	32616
macro avg	0.59	0.60	0.58	32616
weighted avg	0.59	0.60	0.58	32616

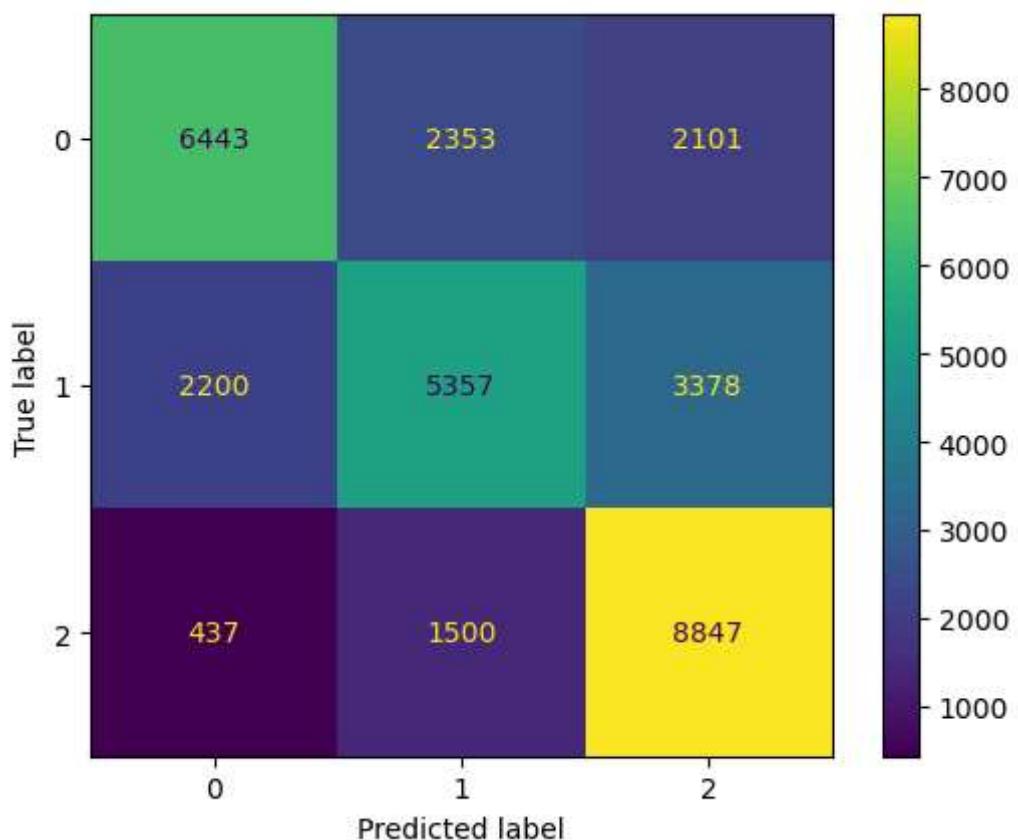


```
In [114]: param_grid = {'alpha': np.linspace(0,1,5),
                      'fit_intercept': [True, False],
                      'positive': [True, False],
                      'solver': ['svd', 'cholesky', 'lsqr', 'sparse_cg', 'sag', 'sas']}
grid_ridge = RandomizedSearchCV(RidgeClassifier(), param_grid, verbose=2, cv=cv)
train_and_evaluate_model(grid_ridge)
```

```
Fitting 5 folds for each of 10 candidates, totalling 50 fits
[CV] END alpha=1.0, fit_intercept=False, positive=True, solver=cholesky;
total time= 0.0s
[CV] END alpha=1.0, fit_intercept=False, positive=True, solver=cholesky;
total time= 0.0s
[CV] END alpha=1.0, fit_intercept=False, positive=True, solver=cholesky;
total time= 0.0s
[CV] END alpha=1.0, fit_intercept=False, positive=True, solver=cholesky;
total time= 0.0s
[CV] END alpha=1.0, fit_intercept=False, positive=True, solver=cholesky;
total time= 0.0s
[CV] END alpha=0.0, fit_intercept=False, positive=True, solver=svd; total
time= 0.0s
[CV] END alpha=0.0, fit_intercept=False, positive=True, solver=svd; total
time= 0.0s
[CV] END alpha=0.0, fit_intercept=False, positive=True, solver=svd; total
time= 0.0s
[CV] END alpha=0.0, fit_intercept=False, positive=True, solver=svd; total
time= 0.0s
[CV] END alpha=0.0, fit_intercept=False, positive=True, solver=svd; total
time= 0.0s
[CV] END alpha=0.5, fit_intercept=True, positive=True, solver=lsqr; total
time= 0.0s
[CV] END alpha=0.5, fit_intercept=True, positive=True, solver=lsqr; total
time= 0.0s
[CV] END alpha=0.5, fit_intercept=True, positive=True, solver=lsqr; total
time= 0.0s
[CV] END alpha=0.5, fit_intercept=True, positive=True, solver=lsqr; total
time= 0.0s
[CV] END alpha=0.25, fit_intercept=True, positive=True, solver=cholesky;
total time= 0.0s
[CV] END alpha=0.25, fit_intercept=True, positive=True, solver=cholesky;
total time= 0.0s
[CV] END alpha=0.25, fit_intercept=True, positive=True, solver=cholesky;
total time= 0.0s
[CV] END alpha=0.25, fit_intercept=True, positive=True, solver=cholesky;
total time= 0.0s
[CV] END alpha=0.25, fit_intercept=True, positive=True, solver=cholesky;
total time= 0.0s
[CV] END alpha=0.75, fit_intercept=False, positive=False, solver=cholesky;
total time= 0.0s
[CV] END alpha=0.75, fit_intercept=False, positive=False, solver=cholesky;
total time= 0.0s
[CV] END alpha=0.75, fit_intercept=False, positive=False, solver=cholesky;
total time= 0.0s
[CV] END alpha=0.75, fit_intercept=False, positive=False, solver=cholesky;
total time= 0.0s
[CV] END alpha=0.75, fit_intercept=False, positive=False, solver=cholesky;
total time= 0.0s
[CV] END alpha=0.75, fit_intercept=False, positive=True, solver=cholesky;
total time= 0.0s
[CV] END alpha=0.75, fit_intercept=False, positive=True, solver=cholesky;
total time= 0.0s
[CV] END alpha=0.75, fit_intercept=False, positive=True, solver=cholesky;
total time= 0.0s
[CV] END alpha=0.75, fit_intercept=False, positive=True, solver=cholesky;
```

```
[CV] END alpha=0.25, fit_intercept=True, positive=True, solver=sparse_cg;
total time= 0.0s
[CV] END alpha=0.25, fit_intercept=True, positive=True, solver=sparse_cg;
total time= 0.0s
[CV] END alpha=0.25, fit_intercept=True, positive=True, solver=sparse_cg;
total time= 0.0s
[CV] END alpha=0.25, fit_intercept=True, positive=True, solver=sparse_cg;
total time= 0.0s
[CV] END alpha=0.25, fit_intercept=True, positive=True, solver=sparse_cg;
total time= 0.0s
[CV] END alpha=0.75, fit_intercept=True, positive=True, solver=svd; total
time= 0.0s
[CV] END alpha=0.75, fit_intercept=True, positive=True, solver=svd; total
time= 0.0s
[CV] END alpha=0.75, fit_intercept=True, positive=True, solver=svd; total
time= 0.0s
[CV] END alpha=0.75, fit_intercept=True, positive=True, solver=svd; total
time= 0.0s
[CV] END alpha=0.75, fit_intercept=True, positive=True, solver=svd; total
time= 0.0s
[CV] END alpha=0.25, fit_intercept=True, positive=False, solver=saga; tot
al time= 3.2s
[CV] END alpha=0.25, fit_intercept=True, positive=False, solver=saga; tot
al time= 3.2s
[CV] END alpha=0.25, fit_intercept=True, positive=False, solver=saga; tot
al time= 2.4s
[CV] END alpha=0.25, fit_intercept=True, positive=False, solver=saga; tot
al time= 3.2s
[CV] END alpha=0.25, fit_intercept=True, positive=False, solver=saga; tot
al time= 2.8s
[CV] END alpha=0.25, fit_intercept=False, positive=False, solver=svd; tot
al time= 0.1s
[CV] END alpha=0.25, fit_intercept=False, positive=False, solver=svd; tot
al time= 0.0s
[CV] END alpha=0.25, fit_intercept=False, positive=False, solver=svd; tot
al time= 0.0s
[CV] END alpha=0.25, fit_intercept=False, positive=False, solver=svd; tot
al time= 0.0s
```

	precision	recall	f1-score	support
0	0.71	0.59	0.65	10897
1	0.58	0.49	0.53	10935
2	0.62	0.82	0.70	10784
accuracy			0.63	32616
macro avg	0.64	0.63	0.63	32616
weighted avg	0.64	0.63	0.63	32616



Optimized Models Performance Comparison

In [115]: model_perfs = pd.DataFrame({'Model': model_names, 'Accuracy': accuracy_scores})
model_perfs

Out[115]:

		Model	Accuracy	Precision	Recall	F1
0	VotingClassifier(estimators=[('RF', RandomForestClassifi...]	0.839557	0.839557	0.839557	0.839557	0.839557
1	(DecisionTreeClassifier(max_features='sqrt', random_...]	0.839465	0.839465	0.839465	0.839465	0.839465
2	RandomizedSearchCV(estimator=LGBMClassifier(), cv=5,...]	0.828888	0.828888	0.828888	0.828888	0.828888
3	(ExtraTreeClassifier(random_state=1036040382), ...]	0.821069	0.821069	0.821069	0.821069	0.821069
4	(DecisionTreeClassifier(random_state=136957480), ...]	0.813006	0.813006	0.813006	0.813006	0.813006
5	XGBClassifier(base_score=None, booster=None, c...]	0.806721	0.806721	0.806721	0.806721	0.806721
6	<catboost.core.CatBoostClassifier object at 0x...]	0.802857	0.802857	0.802857	0.802857	0.802857
7	HistGradientBoostingClassifier()	0.793261	0.793261	0.793261	0.793261	0.793261
8	LGBMClassifier()	0.785473	0.785473	0.785473	0.785473	0.785473
9	KNeighborsClassifier()	0.762203	0.762203	0.762203	0.762203	0.762203
10	([DecisionTreeRegressor(criterion='friedman_ms...]	0.750153	0.750153	0.750153	0.750153	0.750153
11	DecisionTreeClassifier()	0.747087	0.747087	0.747087	0.747087	0.747087
12	RandomizedSearchCV(cv=5, estimator=DecisionTre...]	0.737613	0.737613	0.737613	0.737613	0.737613
13	XGBRFClassifier(base_score=None, booster=None,...]	0.733045	0.733045	0.733045	0.733045	0.733045
14	(DecisionTreeClassifier(max_depth=1, random_st...]	0.707230	0.707230	0.707230	0.707230	0.707230
15	RidgeClassifier()	0.635670	0.635670	0.635670	0.635670	0.635670
16	RandomizedSearchCV(cv=5, estimator=RidgeClassi...]	0.633033	0.633033	0.633033	0.633033	0.633033
17	RandomizedSearchCV(cv=5, estimator=BernoulliNB...]	0.595628	0.595628	0.595628	0.595628	0.595628
18	MLPClassifier()	0.485038	0.485038	0.485038	0.485038	0.485038
19	BernoulliNB()	0.471118	0.471118	0.471118	0.471118	0.471118
20	PassiveAggressiveClassifier()	0.409124	0.409124	0.409124	0.409124	0.409124
21	LinearSVC()	0.341090	0.341090	0.341090	0.341090	0.341090
22	RandomizedSearchCV(cv=5, estimator=LogisticReg...]	0.334130	0.334130	0.334130	0.334130	0.334130
23	LogisticRegression()	0.334130	0.334130	0.334130	0.334130	0.334130
24	GaussianNB()	0.334100	0.334100	0.334100	0.334100	0.334100
25	SGDClassifier()	0.330605	0.330605	0.330605	0.330605	0.330605

Before evaluating the best model after hyperparameter tuning, let's first understand the initial performance of the models. Looking at the accuracy, precision, recall, and F1-score metrics, the best model initially seems to be the VotingClassifier and the DecisionTreeClassifier. Both models achieved the highest scores across all the mentioned metrics.

Now, after hyperparameter tuning, the scores remain the same for the VotingClassifier and DecisionTreeClassifier. The other models show similar performance as well.

Evaluation of the Best Model (VotingClassifier):

Accuracy: 83.96% Precision: 83.96% Recall: 83.96% F1 Score: 83.96% Why is it the Best Model?

Consistency: The VotingClassifier demonstrates consistency in performance across all metrics, indicating a balanced and robust model.

Ensemble Advantage: The VotingClassifier is an ensemble model, combining the strengths of multiple base models. This often leads to better generalization and performance.

Hyperparameter Tuning: Even after hyperparameter tuning, the VotingClassifier maintains its high scores, suggesting that the initial configuration was already effective.

DecisionTreeClassifier Contribution: Given that the DecisionTreeClassifier is one of the base models in the ensemble, its individual performance contributes significantly to the overall success of the VotingClassifier.

Practical Considerations: Depending on the nature of your problem and the importance of precision, recall, or overall accuracy, you might prioritize different models. In this case, the

In []:

In []: