Machine Learning – Project Assignment

Alvin Tran

1. INTRODUCTION

Machine learning is about taking data from everyday life and using algorithms to learn more about the data that is not so obvious at the surface. It is important to note that some algorithms will perform better on the dataset than others. To add on to that, performance can be improved by preprocessing the data beforehand and picking parameters that makes sense for the algorithms with regard to the dataset.

For this project, two datasets were provided in which we were to apply three machine learning algorithms per dataset. With each algorithm, we were to tune and optimize the algorithms we used so that the algorithms can perform the best it can on the dataset with high and meaningful metric scores.

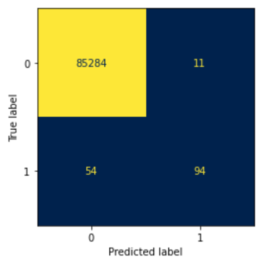
One of the datasets is the Credit Card Fraud Detection dataset. The dataset requires machine learning algorithms for classification problems. Classification problems are problems where a certain value of the dataset can be predicted and categorized into two options, 0 or 1, or in this case with the credit card dataset, not fraud or fraud. I have not finished using all three of my chosen algorithms for this dataset, so the results remain inconclusive. The algorithms I have chosen and completed are logistic regression with imbalanced learn and without, and support vector machines without imbalanced learning. Based off the best model for logistic regression, the best accuracy score, precision score, recall score, and f1 score I have obtained is 0.9768, 0.0627, 0.8885, and 0.1171 respectively. Based off of the support vector machines, the scores are 0.9992, 0.9659, 0.5743, and 0.7203 respectively.

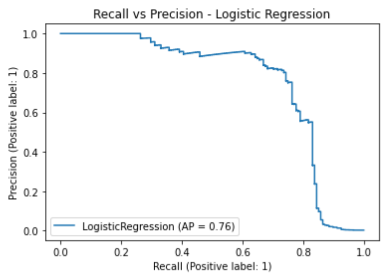
The other dataset is the Energy Efficiency dataset. Unlike the credit card dataset, this dataset requires machine learning algorithms for regression problems. Regression problems are problems where the values of the dataset can be predicted as any value using the existing data and, in this case, with the energy efficiency dataset, values of heating load and cooling load can be predicted based off of data in the dataset. I have not finished using all three of my chosen algorithms for this dataset, so the results remain inconclusive. The algorithms I have chosen and the only one I completed is linear regression. Just based on linear regression, the mean squared error and r2 score for heating load predictions is 7.4127 and 0.9289 respectively while the mean squared error and r2 score for cooling load predictions is 11.0915 and 0.8787 respectively.

2. Credit Card Fraud Detection Dataset

The Credit Card Fraud Detection dataset contains data such as the time in seconds between each transaction and the first one, numerical values from PCA transformations, the amount of each transaction, and if that transaction was fraudulent or not. This dataset is considered extremely unbalanced because the fraudulent category only consists of 492 instances out of 284,807 instances having a percentage of 0.172%. In the case of unbalanced datasets, we have to use imbalanced learning methods so that we can tell if the metric data from the machine learning algorithms on the dataset is truly accurate.

One of the machine learning algorithms I have chosen is logistic regression. Without any imbalanced learning methods, the algorithm returns an accuracy of 0.9992. Although it may sound good, we will need to look at the precision score, recall score, and f1 score to determine if the algorithm's performance was good because the dataset is imbalanced. The meaning of each of the scores is that the precision score tells how good the algorithm is at predicting true positives out of all the instances that were labeled positive while the recall score tells how good the algorithm is at predicting true positives out of the instances that were labeled true positive and the instances that were labeled false negative, and the f1 score is the weighted average of both the precision and recall score. The precision score came out to be 0.8952 which is pretty decent while the recall score is only 0.6351 which is a bad result. The f1 score came out to be 0.7430.





In the case of this dataset, we would want a higher recall score because we do not want false negatives because we are working credit card fraud and we do not want the case where a credit card is fraud but label it not fraud.

In [ ]:

```python
# Importing the libraries
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
```

In [ ]:

```python
df = pd.read_csv("creditcard.csv")
df
```

Out[ ]:

	Time	V1	V2	V3	V4	V5	V6	V7	V8	V9
0	0.0	-1.359807	-0.072781	2.536347	1.378155	-0.338321	0.462388	0.239599	0.098698	0.363787
1	0.0	1.191857	0.266151	0.166480	0.448154	0.060018	-0.082361	-0.078803	0.085102	-0.255425
2	1.0	-1.358354	-1.340163	1.773209	0.379780	-0.503198	1.800499	0.791461	0.247676	-1.514654
3	1.0	-0.966272	-0.185226	1.792993	-0.863291	-0.010309	1.247203	0.237609	0.377436	-1.387024
4	2.0	-1.158233	0.877737	1.548718	0.403034	-0.407193	0.095921	0.592941	-0.270533	0.817739
...	...	...	...	...	...	...	...	...	...	...
284802	172786.0	-11.881118	10.071785	-9.834783	-2.066656	-5.364473	-2.606837	-4.918215	7.305334	1.914428
284803	172787.0	-0.732789	-0.055080	2.035030	-0.738589	0.868229	1.058415	0.024330	0.294869	0.584800
284804	172788.0	1.919565	-0.301254	-3.249640	-0.557828	2.630515	3.031260	-0.296827	0.708417	0.432454
284805	172788.0	-0.240440	0.530483	0.702510	0.689799	-0.377961	0.623708	-0.686180	0.679145	0.392087
284806	172792.0	-0.533413	-0.189733	0.703337	-0.506271	-0.012546	-0.649617	1.577006	-0.414650	0.486180

284807 rows × 31 columns

In [ ]:

```python
df = df.drop(["Time", "Amount"], axis=1)
df
```

Out[ ]:

	V1	V2	V3	V4	V5	V6	V7	V8	V9	V10
0	-1.359807	-0.072781	2.536347	1.378155	-0.338321	0.462388	0.239599	0.098698	0.363787	0.090794
1	1.191857	0.266151	0.166480	0.448154	0.060018	-0.082361	-0.078803	0.085102	-0.255425	-0.166974
2	-1.358354	-1.340163	1.773209	0.379780	-0.503198	1.800499	0.791461	0.247676	-1.514654	0.207643
3	-0.966272	-0.185226	1.792993	-0.863291	-0.010309	1.247203	0.237609	0.377436	-1.387024	-0.054952
4	-1.158233	0.877737	1.548718	0.403034	-0.407193	0.095921	0.592941	-0.270533	0.817739	0.753074
...	...	...	...	...	...	...	...	...	...	...
284802	-11.881118	10.071785	-9.834783	-2.066656	-5.364473	-2.606837	-4.918215	7.305334	1.914428	4.356170
284803	-0.732789	-0.055080	2.035030	-0.738589	0.868229	1.058415	0.024330	0.294869	0.584800	-0.975926
284804	1.919565	-0.301254	-3.249640	-0.557828	2.630515	3.031260	-0.296827	0.708417	0.432454	-0.484782
284805	-0.240440	0.530483	0.702510	0.689799	-0.377961	0.623708	-0.686180	0.679145	0.392087	-0.399126
284806	-0.533413	-0.189733	0.703337	-0.506271	-0.012546	-0.649617	1.577006	-0.414650	0.486180	-0.915427

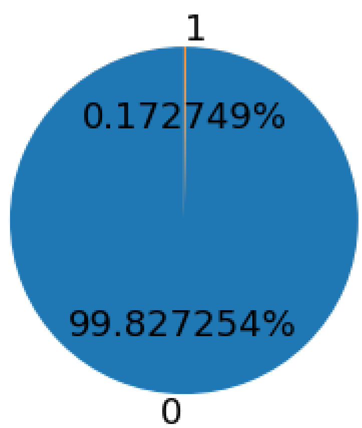
284807 rows × 29 columns

In [ ]:

```
X = df.iloc[:, 0:28].values # features
y = df.iloc[:, 28].values # target
```

In [ ]:

```
unique_elements, counts_elements = np.unique(y, return_counts = True)
plt.pie(counts_elements, labels=unique_elements, autopct='%1f%%',
        startangle=90, textprops={'fontsize': 18})
plt.show()
```



In [ ]:

```
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.30)

unique_elements, counts_elements = np.unique(y_test, return_counts = True)
print(unique_elements, counts_elements)
```

```
[0 1] [85295   148]
```

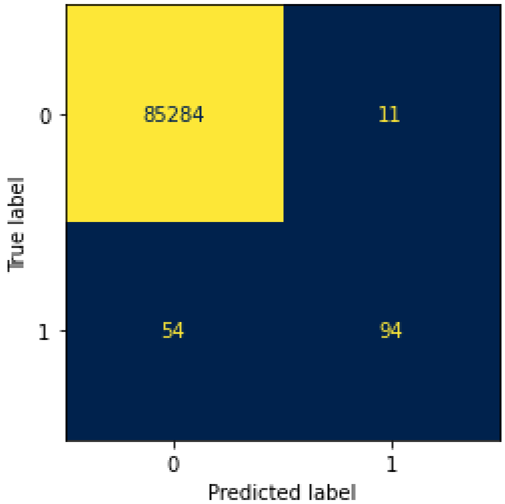
Logistic Regression (without Imbalanced Learning)

In [ ]:

```
from sklearn.linear_model import LogisticRegression

log_reg = LogisticRegression()
log_reg.fit(X_train, y_train)
y_pred = log_reg.predict(X_test)
```

In [ ]:

```
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay

display = ConfusionMatrixDisplay.from_predictions(y_test, y_pred, cmap ="cividis", colorbar=False)
```



In [ ]:

```
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score

accuracy = accuracy_score(y_test, y_pred)
precision = precision_score(y_test, y_pred)
recall = recall_score(y_test, y_pred)
f1 = f1_score(y_test, y_pred)

print("Accuracy score: ", accuracy)
print("Precision score: ", precision)
print("Recall score: ", recall)
print("f1 score: ", f1)

# Recall score is very low, bad results
```

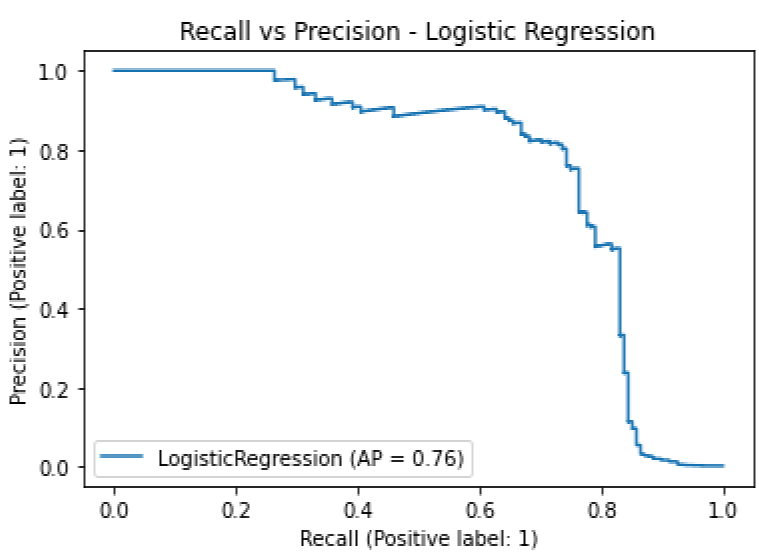
```
Accuracy score:  0.9992392589211521
Precision score:  0.8952380952380953
Recall score:  0.6351351351351351
f1 score:  0.7430830039525692
```

In [ ]:

```
from sklearn.metrics import PrecisionRecallDisplay

display = PrecisionRecallDisplay.from_estimator(log_reg, X_test, y_test)
display.ax_.set_title("Recall vs Precision - Logistic Regression")
```

Out[ ]:

```
Text(0.5, 1.0, 'Recall vs Precision - Logistic Regression')
```



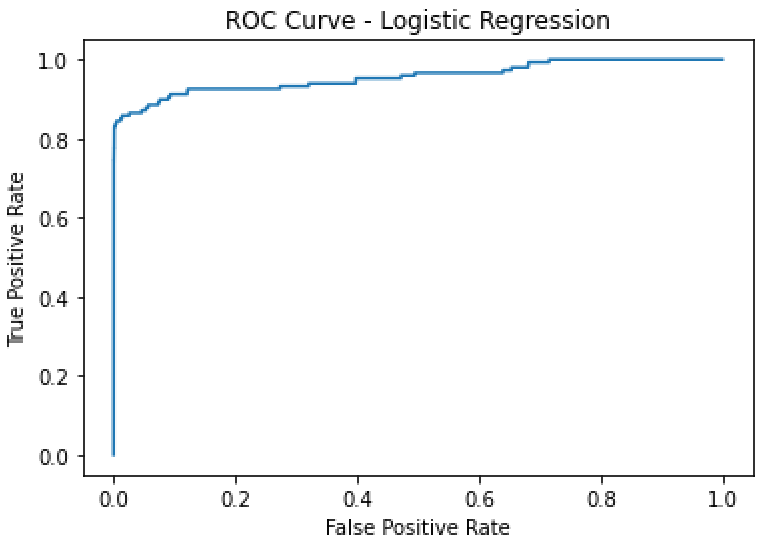
In [ ]:

```
from sklearn.metrics import roc_curve
```

```
y_scores = log_reg.decision_function(X_test)
fpr, tpr, thresholds = roc_curve(y_test, y_scores)

plt.plot(fpr, tpr)

plt.title("ROC Curve - Logistic Regression")
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
```

Out[ ]:

```
Text(0, 0.5, 'True Positive Rate')
```



Logisitic Regression (with ImBalanced Learning -> Data-level Preprocessing by Oversampling)

In [ ]:

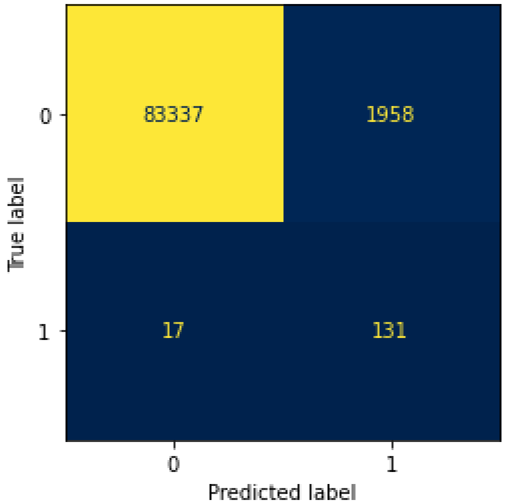
```
# Oversampling by uniform sampling with replacement to balance the input dataset before the traini
from imblearn.over_sampling import RandomOverSampler

ros = RandomOverSampler()
X_train_resampled, y_train_resampled = ros.fit_resample(X_train, y_train)
```

In [ ]:

```
log_reg = LogisticRegression()
log_reg.fit(X_train_resampled, y_train_resampled)
y_pred = log_reg.predict(X_test)
```

In [ ]:

```
display = ConfusionMatrixDisplay.from_predictions(y_test, y_pred, cmap ="cividis", colorbar=False)
```



In [ ]:

```
accuracy = accuracy_score(y_test, y_pred)
precision = precision_score(y_test, y_pred)
recall = recall_score(y_test, y_pred)
f1 = f1_score(y_test, y_pred)

print("Accuracy score: ", accuracy)
print("Precision score: ", precision)
print("Recall score: ", recall)
print("f1 score: ", f1)
```

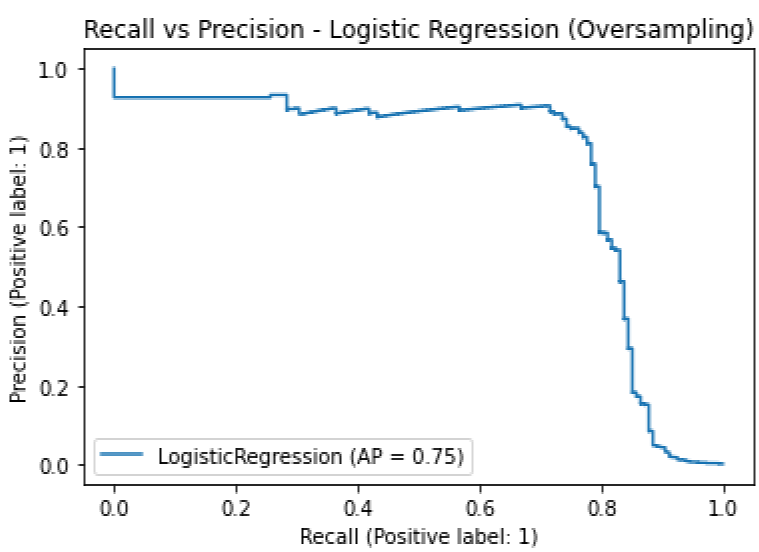
```
Accuracy score:  0.9768851749119296
Precision score:  0.0627094303494495
Recall score:  0.8851351351351351
f1 score:  0.11712114438980777
```

In [ ]:

```
display = PrecisionRecallDisplay.from_estimator(log_reg, X_test, y_test)
display.ax_.set_title("Recall vs Precision - Logistic Regression (Oversampling)")
```

Out[ ]:

```
Text(0.5, 1.0, 'Recall vs Precision - Logistic Regression (Oversampling)')
```



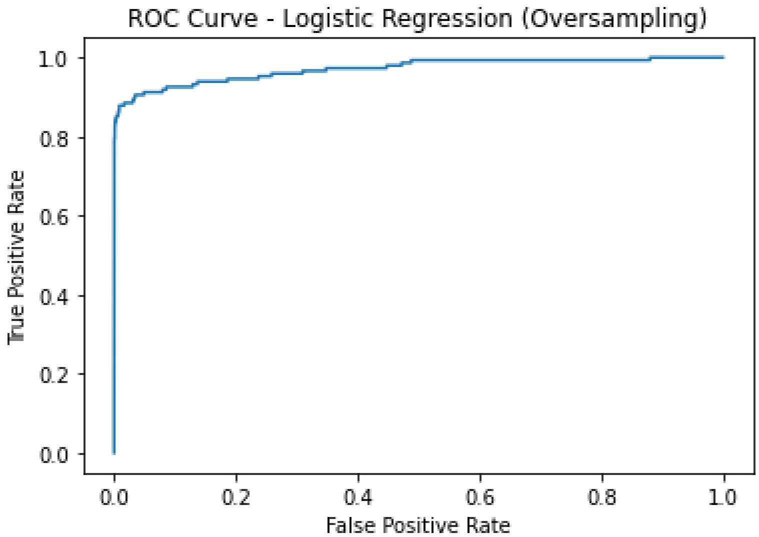
In [ ]:

```
from sklearn.metrics import roc_curve

y_scores = log_reg.decision_function(X_test)
fpr, tpr, thresholds = roc_curve(y_test, y_scores)

plt.plot(fpr, tpr)
```

```
plt.title("ROC Curve - Logistic Regression (Oversampling)")
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
```

Out[ ]:

```
Text(0, 0.5, 'True Positive Rate')
```



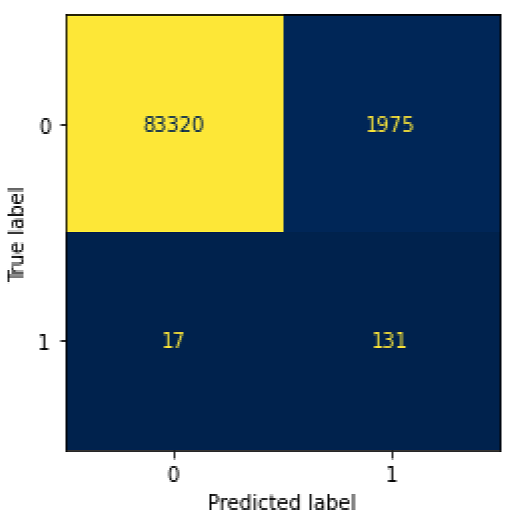
Logistic Regression (with Imbalanced Learning -> Cost-sensitive Learning)

In [ ]:

```
log_reg = LogisticRegression(class_weight="balanced")
log_reg.fit(X_train, y_train)
y_pred = log_reg.predict(X_test)
```

In [ ]:

```
display = ConfusionMatrixDisplay.from_predictions(y_test, y_pred, cmap ="cividis", colorbar=False)
```



In [ ]:

```
accuracy = accuracy_score(y_test, y_pred)
precision = precision_score(y_test, y_pred)
recall = recall_score(y_test, y_pred)
f1 = f1_score(y_test, y_pred)

print("Accuracy score: ", accuracy)
print("Precision score: ", precision)
print("Recall score: ", recall)
print("f1 score: ", f1)
```

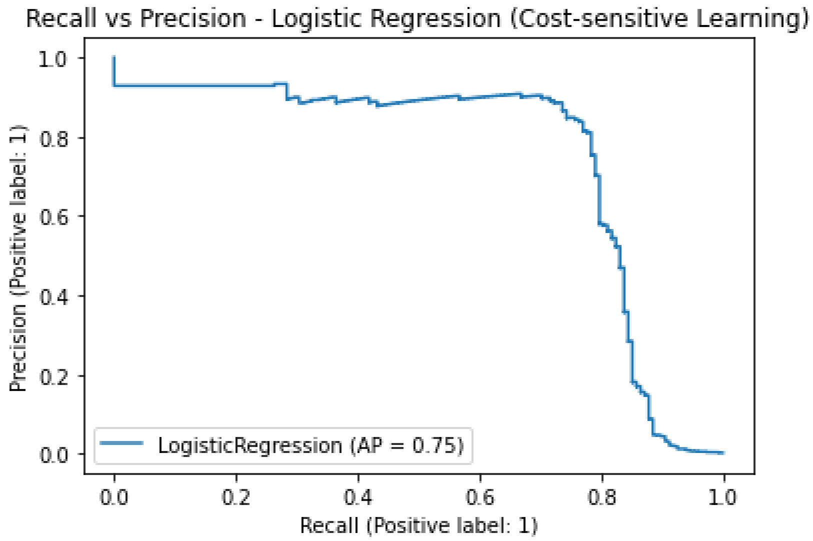
```
Accuracy score:  0.9766862118605386
Precision score:  0.06220322886989554
Recall score:  0.8851351351351351
f1 score:  0.11623779946761315
```

In [ ]:

```python
display = PrecisionRecallDisplay.from_estimator(log_reg, X_test, y_test)
display.ax_.set_title("Recall vs Precision - Logistic Regression (Cost-sensitive Learning)")
```

Out[ ]:

```
Text(0.5, 1.0, 'Recall vs Precision - Logistic Regression (Cost-sensitive Learning)')
```

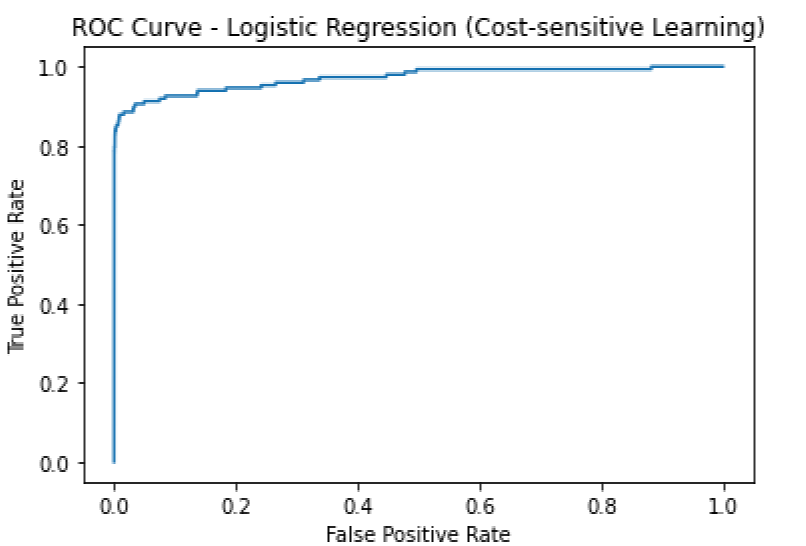


In [ ]:

```python
from sklearn.metrics import roc_curve

y_scores = log_reg.decision_function(X_test)
fpr, tpr, thresholds = roc_curve(y_test, y_scores)

plt.plot(fpr, tpr)

plt.title("ROC Curve - Logistic Regression (Cost-sensitive Learning)")
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
```

Out[ ]:

```
Text(0, 0.5, 'True Positive Rate')
```



Logistic Regression (with Imbalanced Learning ->

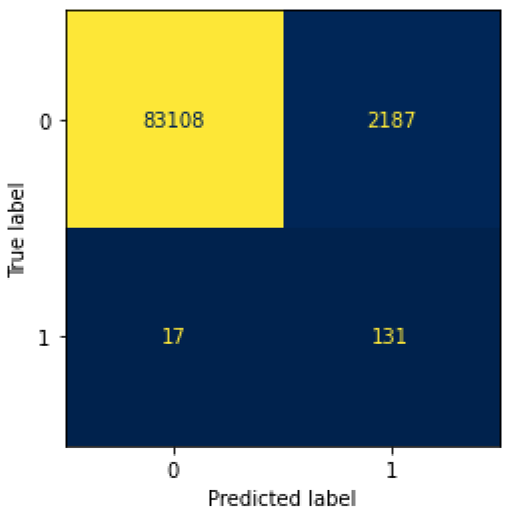
Emsemble Learning)

In [ ]:

```
from imblearn.over_sampling import SMOTE
from sklearn.ensemble import BaggingClassifier

smote = SMOTE()
X_train_resampled, y_train_resampled = smote.fit_resample(X_train, y_train)

log_reg = BaggingClassifier(base_estimator=LogisticRegression(), n_estimators=10)
log_reg.fit(X_train_resampled, y_train_resampled)
y_pred = log_reg.predict(X_test)
```

In [ ]:

```
display = ConfusionMatrixDisplay.from_predictions(y_test, y_pred, cmap ="cividis", colorbar=False)
```



In [ ]:

```
accuracy = accuracy_score(y_test, y_pred)
precision = precision_score(y_test, y_pred)
recall = recall_score(y_test, y_pred)
f1 = f1_score(y_test, y_pred)

print("Accuracy score: ", accuracy)
print("Precision score: ", precision)
print("Recall score: ", recall)
print("f1 score: ", f1)
```

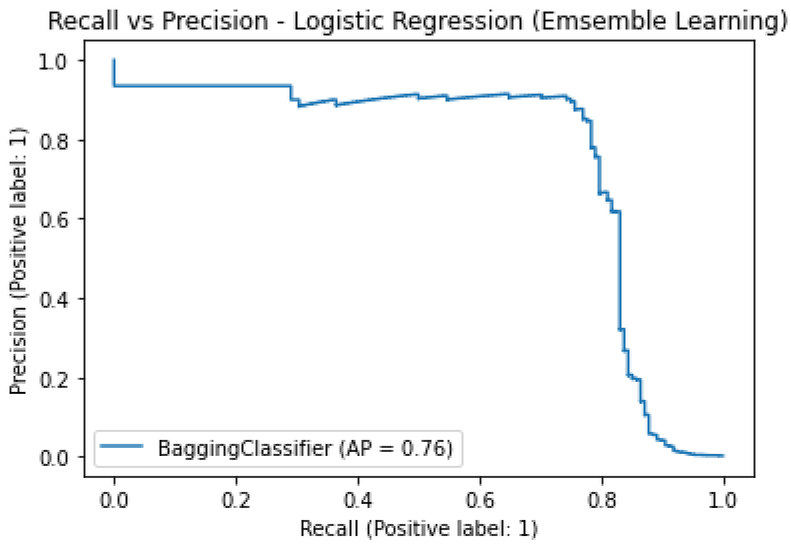
```
Accuracy score:  0.9742050255726039
Precision score:  0.05651423641069888
Recall score:  0.8851351351351351
f1 score:  0.10624493106244932
```

In [ ]:

```
display = PrecisionRecallDisplay.from_estimator(log_reg, X_test, y_test)
display.ax_.set_title("Recall vs Precision - Logistic Regression (Emsemble Learning)")
```

Out[ ]:

```
Text(0.5, 1.0, 'Recall vs Precision - Logistic Regression (Emsemble Learning)')
```

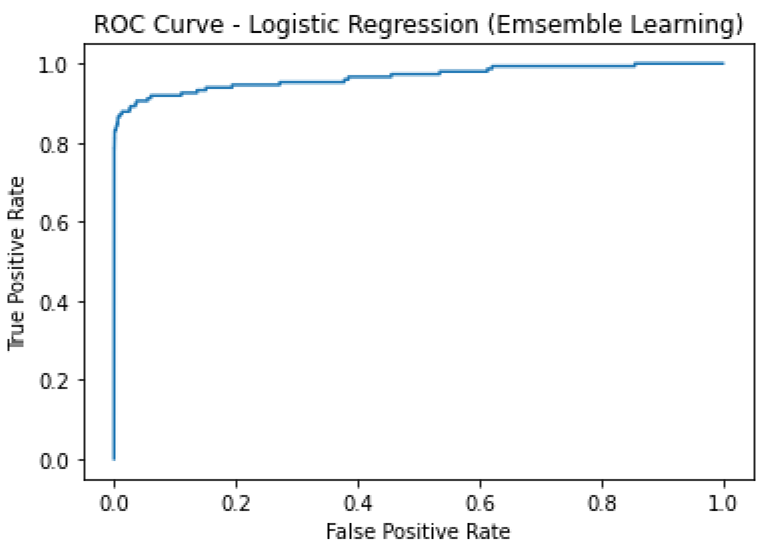


In [ ]:

```
from sklearn.metrics import roc_curve

y_scores = log_reg.decision_function(X_test)
fpr, tpr, thresholds = roc_curve(y_test, y_scores)

plt.plot(fpr, tpr)

plt.title("ROC Curve - Logistic Regression (Emsemble Learning)")
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
```

Out[ ]:

```
Text(0, 0.5, 'True Positive Rate')
```



Support Vector Machine (Without Imbalanced Learning)

In [ ]:

```
from sklearn.preprocessing import StandardScaler

X_scalar = StandardScaler()
X_train = X_scalar.fit_transform(X_train)
X_test = X_scalar.transform(X_test)
```

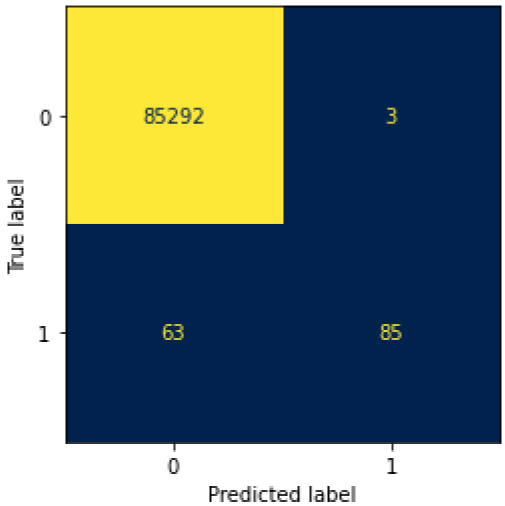
In [ ]:

```
from sklearn.svm import SVC
```

```python
svm_clf = SVC(probability=True)
svm_clf.fit(X_train, y_train)
y_pred = svm_clf.predict(X_test)
```

In [ ]:

```python
display = ConfusionMatrixDisplay.from_predictions(y_test, y_pred, cmap ="cividis", colorbar=False)
```



In [ ]:

```python
accuracy = accuracy_score(y_test, y_pred)
precision = precision_score(y_test, y_pred)
recall = recall_score(y_test, y_pred)
f1 = f1_score(y_test, y_pred)

print("Accuracy score: ", accuracy)
print("Precision score: ", precision)
print("Recall score: ", recall)
print("f1 score: ", f1)
```

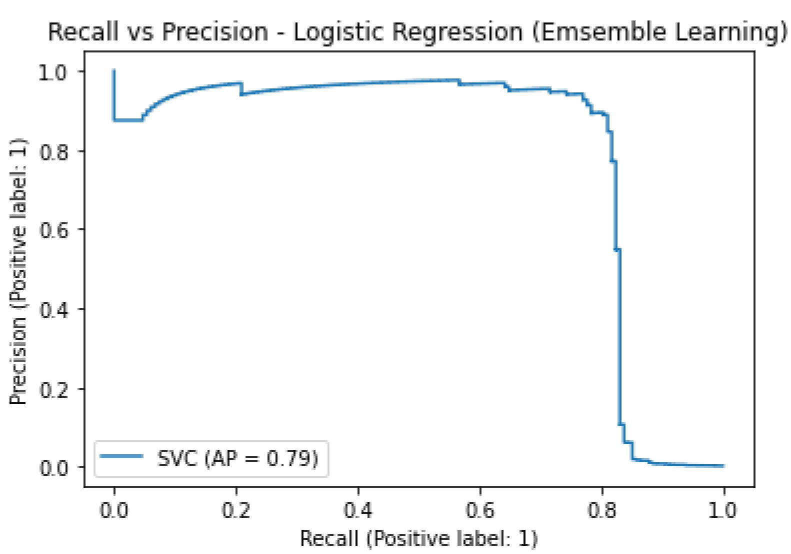
```
Accuracy score:  0.9992275552122467
Precision score:  0.9659090909090909
Recall score:  0.5743243243243243
f1 score:  0.7203389830508473
```

In [ ]:

```python
display = PrecisionRecallDisplay.from_estimator(svm_clf, X_test, y_test)
display.ax_.set_title("Recall vs Precision - Logistic Regression (Emsemble Learning)")
```

Out[ ]:

```
Text(0.5, 1.0, 'Recall vs Precision - Logistic Regression (Emsemble Learning)')
```

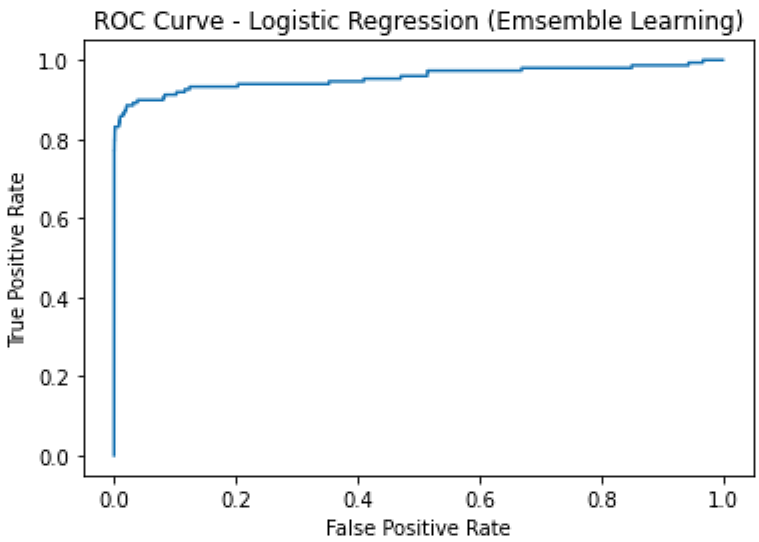


In [ ]:

```
from sklearn.metrics import roc_curve

y_scores = log_reg.decision_function(X_test)
fpr, tpr, thresholds = roc_curve(y_test, y_scores)

plt.plot(fpr, tpr)

plt.title("ROC Curve - Logistic Regression (Emsemble Learning)")
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
```

Out[ ]:

```
Text(0, 0.5, 'True Positive Rate')
```



In [ ]:

```
# Importing the libraries
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
```

In [ ]:

```
df = pd.read_excel("ENB2012_data.xlsx")
df
```

Out[ ]:

	X1	X2	X3	X4	X5	X6	X7	X8	Y1	Y2
0	0.98	514.5	294.0	110.25	7.0	2	0.0	0	15.55	21.33
1	0.98	514.5	294.0	110.25	7.0	3	0.0	0	15.55	21.33
2	0.98	514.5	294.0	110.25	7.0	4	0.0	0	15.55	21.33
3	0.98	514.5	294.0	110.25	7.0	5	0.0	0	15.55	21.33
4	0.90	563.5	318.5	122.50	7.0	2	0.0	0	20.84	28.28
...	...	...	...	...	...	...	...	...	...	...
763	0.64	784.0	343.0	220.50	3.5	5	0.4	5	17.88	21.40
764	0.62	808.5	367.5	220.50	3.5	2	0.4	5	16.54	16.88
765	0.62	808.5	367.5	220.50	3.5	3	0.4	5	16.44	17.11
766	0.62	808.5	367.5	220.50	3.5	4	0.4	5	16.48	16.61
767	0.62	808.5	367.5	220.50	3.5	5	0.4	5	16.64	16.03

768 rows × 10 columns

In [ ]:

```
X = df.iloc[:, 0:8].values # features
y1 = df.iloc[:, 8].values # target: Heating Load
y2 = df.iloc[:, 9].values # target: Cooling Load

from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
```

In [ ]:

```
from sklearn.model_selection import train_test_split

# Splits the featires and the heating/cooling load into
X_train_y1, X_test_y1, y1_train, y1_test = train_test_split(X, y1, test_size = 0.30)
X_train_y2, X_test_y2, y2_train, y2_test = train_test_split(X, y2, test_size = 0.30)
```

Linear Regression

In [ ]:

```
from sklearn.linear_model import LinearRegression

# fits the training and test sets into a linear regression model and predict the values for heatin
lin_reg_Xy1 = LinearRegression()
lin_reg_Xy1.fit(X_train_y1, y1_train)
y1_pred = lin_reg_Xy1.predict(X_test_y1)

lin_reg_Xy2 = LinearRegression()
```

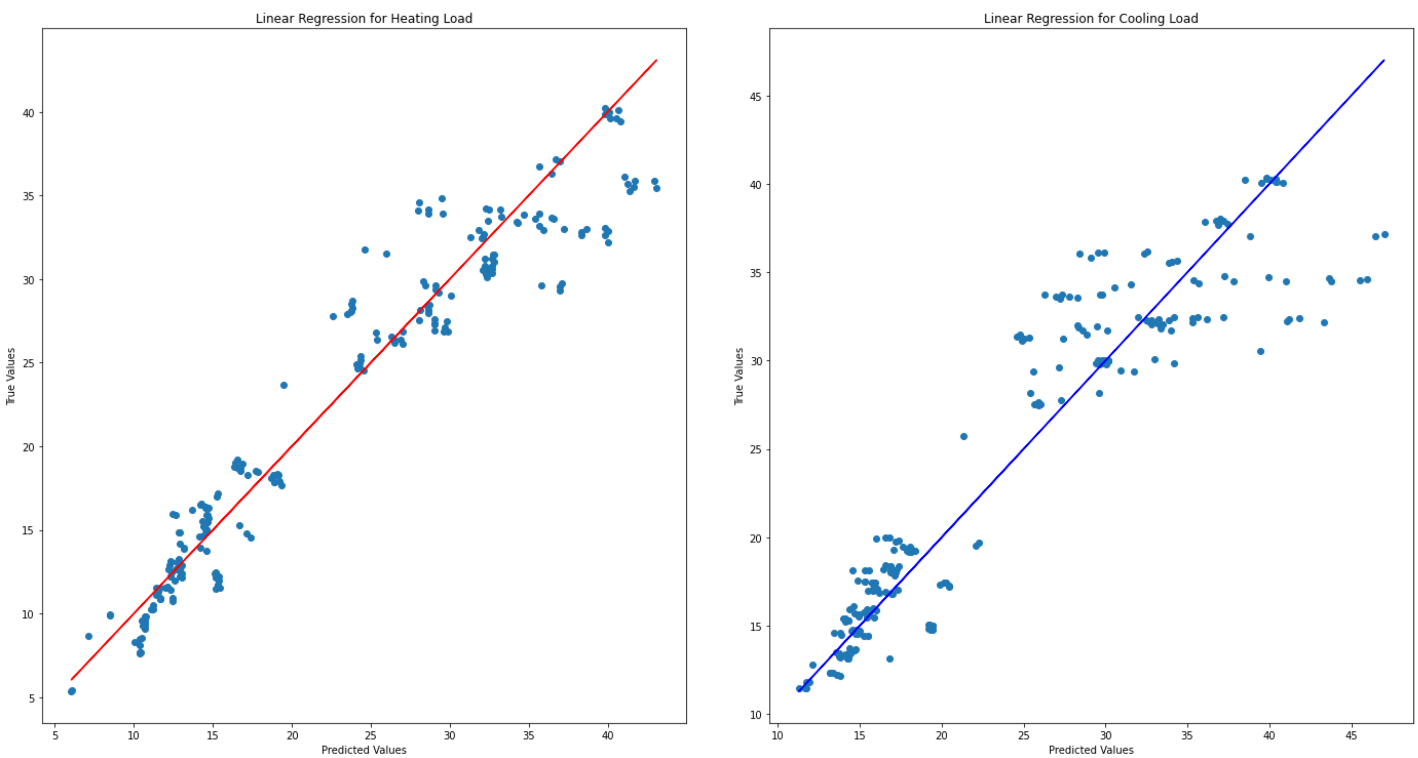
```
lin_reg_Xy2.fit(X_train_y2, y2_train)
y2_pred = lin_reg_Xy2.predict(X_test_y2)
```

In [ ]:

```
f, (ax1, ax2) = plt.subplots(1, 2, figsize = (10, 10))
f.tight_layout(w_pad=1.0)
f.set_figwidth(20)

# plotting predicted values for heating/cooling for the features
ax1.scatter(y1_test, y1_pred)
ax1.plot(y1_test, y1_test, 'red')
ax1.set_title("Linear Regression for Heating Load")
ax1.set_xlabel("Predicted Values")
ax1.set_ylabel("True Values")

ax2.scatter(y2_test, y2_pred)
ax2.plot(y2_test, y2_test, 'blue')
ax2.set_title("Linear Regression for Cooling Load")
ax2.set_xlabel("Predicted Values")
ax2.set_ylabel("True Values")
```

Out[ ]:

```
Text(367.48863636363626, 0.5, 'True Values')
```



In [ ]:

```
from sklearn.metrics import accuracy_score, mean_squared_error, r2_score

y1_mse = mean_squared_error(y1_test, y1_pred)
y1_r2 = r2_score(y1_test, y1_pred)

y2_mse = mean_squared_error(y2_test, y2_pred)
y2_r2 = r2_score(y2_test, y2_pred)

print("Mean squared error for heating load predictions: ", y1_mse)
print("R2 Score for heating load predictions: ", y1_r2)
print("\n")
print("Mean squared error for cooling load predictions: ", y2_mse)
print("R2 Score for cooling load predictions: ", y2_r2)
```

```
Mean squared error for heating load predictions:  7.412723406804143
R2 Score for heating load predictions:  0.9289089216619242
```

```
Mean squared error for cooling load predictions:  11.091531913504145
R2 Score for cooling load predictions:  0.8787393918931318
```