```python
from google.colab import drive
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

```python
!ls "/content/drive/My Drive"
```

```
 all_language_files.zip                            'M.Tech DA'
'Resume1 (3).pdf'
 aps_failure_training_set.csv                      'October mess fees.pdf'
Resume1.pdf
 Bachelor_Degree_Certificate_and_Marksheet.pdf     Report.gdoc
        test_dataset.zip
'Colab Notebooks'                                  Resume
validation_dataset.zip
'Fixed Resume.pdf'                                 'Resume1 (1).pdf'
 language_files_massive.zip                        'Resume1 (2).pdf'
```

```python
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.svm import SVC
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import classification_report
from sklearn.preprocessing import StandardScaler
```

```python
data=pd.read_csv("/content/drive/My
Drive/aps_failure_training_set.csv")
data
```

{"type":"dataframe","variable_name":"data"}

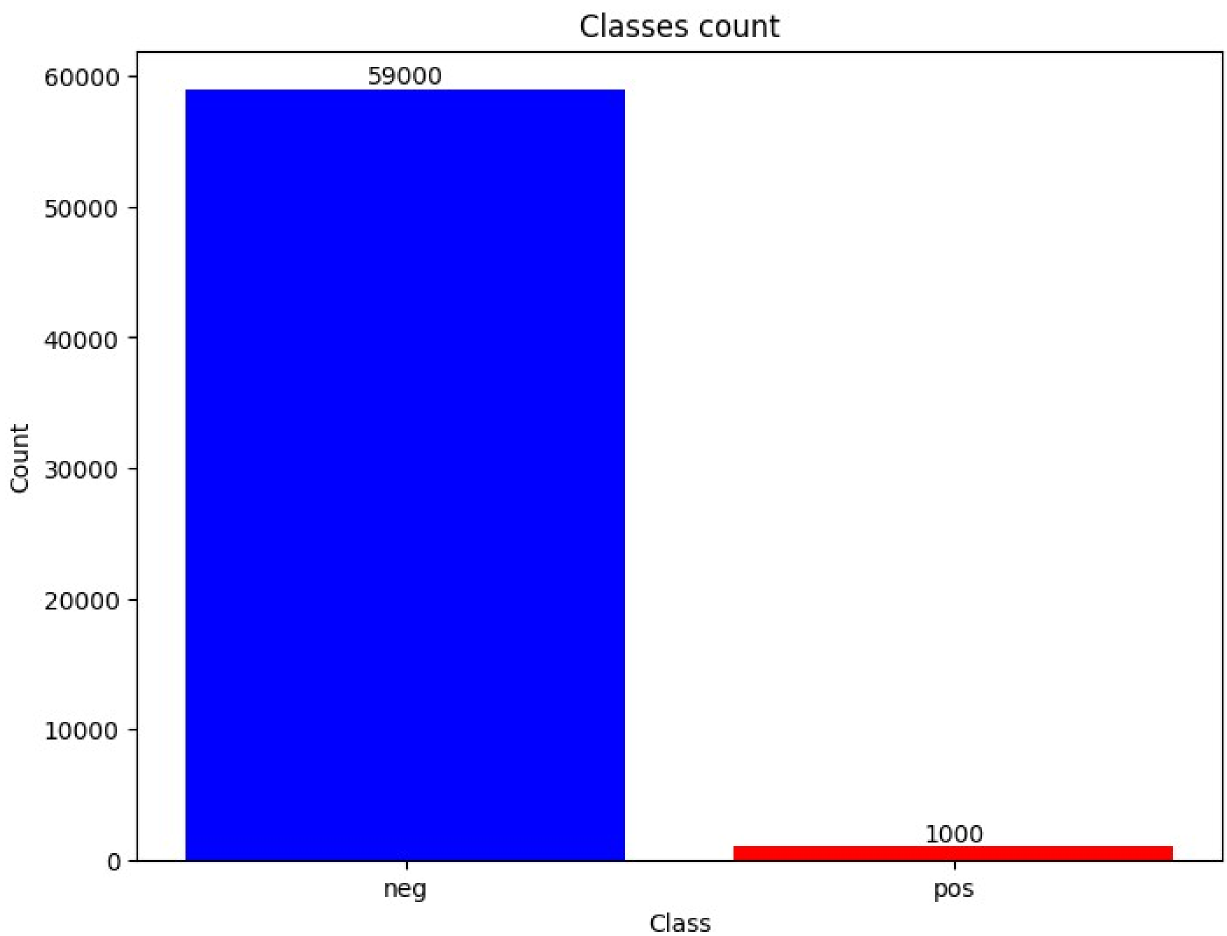
Exploratory Data Analysis

```python
data.head()
```

{"type":"dataframe","variable_name":"data"}

```python
counts = data['class'].value_counts()

plt.figure(figsize=(8, 6))
bars = plt.bar(counts.index, counts.values, color=['blue', 'red'])

for bar, count in zip(bars, counts.values):
    plt.text(bar.get_x() + bar.get_width() / 2, bar.get_height() +
0.05, count,
             ha='center', va='bottom', fontsize=10)
```

```
plt.xlabel('Class')
plt.ylabel('Count')
plt.title('Classes count')
plt.show()
```



Based on this , we can observe that there are many more data points classified as 'neg', which corresponds to air systems without defects, than data points classified as 'pos', which are anamolous air systems.

```
missing_counts=data.apply(lambda x:x[x=='na'].count())

total_counts=len(data)

misssing_percentage=(missing_counts/total_counts)*100

missing_data=pd.DataFrame({'Columns':missing_counts.index,
                           'Missing Values':missing_counts.values,
                           'Percentage of Missing
Values':misssing_percentage.values
                           })
```

```
missing_data = missing_data[missing_data['Missing Values'] > 0]
missing_data = missing_data.sort_values(by='Percentage of Missing
Values', ascending=False)

print(missing_data)
```

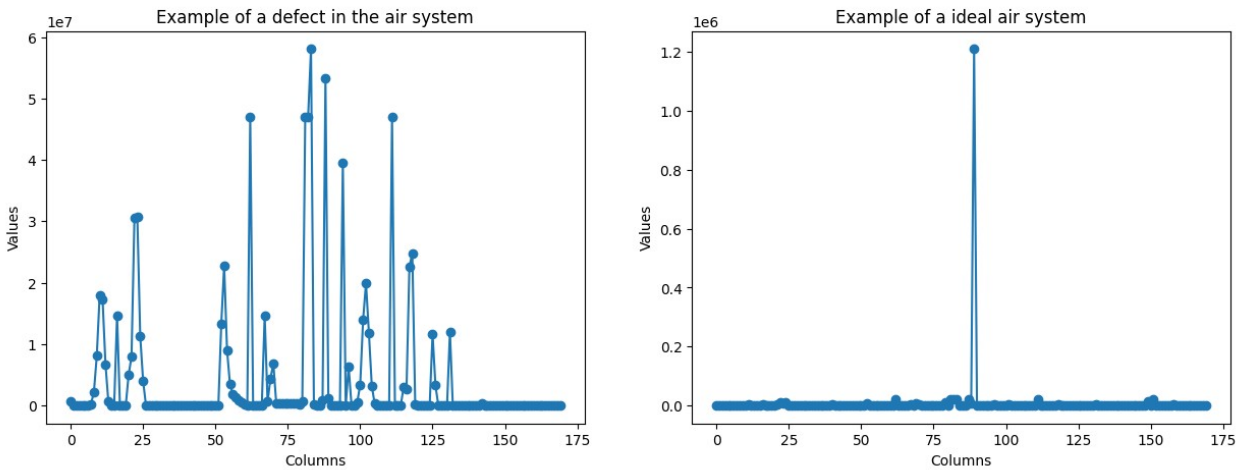
```
    Columns  Missing Values  Percentage of Missing Values
79   br_000           49264                     82.106667
78   bq_000           48722                     81.203333
77   bp_000           47740                     79.566667
76   bo_000           46333                     77.221667
113  cr_000           46329                     77.215000
..      ...             ...                           ...
85   by_000             473                      0.788333
97   ck_000             338                      0.563333
96   cj_000             338                      0.563333
95   ci_000             338                      0.563333
81   bt_000             167                      0.278333

[169 rows x 3 columns]
```

```
# Get columns with more than 5% missing values
coloumns_with_missing_values=missing_data[missing_data['Percentage of
Missing Values']>5]['Columns'].tolist()
print(len(coloumns_with_missing_values))
print(coloumns_with_missing_values)
```

```
42
['br_000', 'bq_000', 'bp_000', 'bo_000', 'cr_000', 'ab_000', 'bn_000',
'bm_000', 'bl_000', 'bk_000', 'co_000', 'cg_000', 'ad_000', 'cf_000',
'ch_000', 'ct_000', 'cy_000', 'cz_000', 'cu_000', 'cv_000', 'dc_000',
'da_000', 'cx_000', 'db_000', 'ec_00', 'cm_000', 'ed_000', 'cl_000',
'ak_000', 'ca_000', 'dm_000', 'dg_000', 'df_000', 'dl_000', 'dh_000',
'dj_000', 'eb_000', 'dk_000', 'di_000', 'ac_000', 'bx_000', 'cc_000']
```

```
def plot_examples(index):
    data_copy=data.copy()
    data_copy.replace('na',0,inplace=True)


positive=data_copy[data_copy['class']=='pos'].iloc[index].drop('class'
,axis=0)

negative=data_copy[data_copy['class']=='neg'].iloc[index].drop('class'
,axis=0)

    plt.figure(figsize=(15,5))
    plt.subplot(1,2,1)
    plt.plot(positive.values,marker='o',linestyle='-')
    plt.xlabel('Columns')
    plt.ylabel('Values')
```

```python
    plt.title('Example of a defect in the air system')

    plt.subplot(1,2,2)
    plt.plot(negative.values,marker='o',linestyle='-')
    plt.xlabel('Columns')
    plt.ylabel('Values')
    plt.title('Example of a ideal air system')

    plt.show()

plot_examples(3)
```



We can see from the plot that an ideal air system and an anomalous air system to start understanding the dataset we are working with. One idea that arises from this is to analyze the quartiles of an anomalous data point and an ideal data point to try to indetify any interesting differences that could be useful for training a future machine learning model

```python
data.dtypes
```

```
class      object
aa_000      int64
ab_000     object
ac_000     object
ad_000     object
            ...
ee_007     object
ee_008     object
ee_009     object
ef_000     object
eg_000     object
Length: 171, dtype: object
```

From above we can see that dtypes is object. We have to convert it into numerical.

```
columns=data.columns[1:]
data[columns]=data[columns].apply(pd.to_numeric,errors='coerce')
data.dtypes
```

```
class       object
aa_000       int64
ab_000     float64
ac_000     float64
ad_000     float64
            ...
ee_007     float64
ee_008     float64
ee_009     float64
ef_000     float64
eg_000     float64
Length: 171, dtype: object
```

```
def describe(df):
  df_pos=df[df['class']=='pos'].drop('class',axis=1)
  df_neg=df[df['class']=='neg'].drop('class',axis=1)

  print('POS')
  print(df_pos.describe())
  print('Neg')
  print(df_neg.describe())

describe(data)
```

```
POS
             aa_000      ab_000        ac_000        ad_000
ae_000  \
count  1.000000e+03  229.000000  5.380000e+02    355.000000
659.000000
mean   6.591737e+05    1.563319  5.940968e+07   2262.214085
20.130501
std    4.343839e+05    4.466411  3.511083e+08   5463.415364
234.564023
min    0.000000e+00    0.000000  0.000000e+00      0.000000
0.000000
25%    3.181575e+05    0.000000  4.500000e+00    191.000000
0.000000
50%    5.849940e+05    0.000000  8.650000e+02    648.000000
0.000000
75%    9.305015e+05    2.000000  2.729500e+03   2014.000000
0.000000
max    2.746564e+06   48.000000  2.130706e+09  60466.000000
5386.000000

             af_000         ag_000        ag_001        ag_002
ag_003  \
```

```
count    659.000000     996.000000  9.960000e+02  9.960000e+02
9.960000e+02
mean      54.907436    2159.925703  5.108507e+04  4.004426e+05
2.617797e+06
std      621.432189   22245.119346  2.560935e+05  9.998827e+05
4.247642e+06
min        0.000000       0.000000  0.000000e+00  0.000000e+00
0.000000e+00
25%        0.000000       0.000000  0.000000e+00  0.000000e+00
2.060000e+02
50%        0.000000       0.000000  0.000000e+00  2.208000e+03
4.024190e+05
75%        0.000000       0.000000  7.777500e+03  3.763340e+05
3.857914e+06
max    11284.000000  544866.000000  4.109372e+06  1.055286e+07
2.904730e+07

       ...        ee_002        ee_003        ee_004        ee_005  \
count  ...  9.950000e+02  9.950000e+02  9.950000e+02  9.950000e+02
mean   ...  4.384086e+06  2.000143e+06  4.177729e+06  4.574311e+06
std    ...  4.243864e+06  2.132602e+06  4.511349e+06  5.569835e+06
min    ...  0.000000e+00  0.000000e+00  0.000000e+00  0.000000e+00
25%    ...  1.147822e+06  4.754150e+05  9.149670e+05  8.472080e+05
50%    ...  2.949986e+06  1.295384e+06  2.591636e+06  2.717748e+06
75%    ...  6.612008e+06  2.938731e+06  5.941204e+06  6.488371e+06
max    ...  3.123272e+07  1.454922e+07  2.700915e+07  5.743524e+07

             ee_006        ee_007        ee_008        ee_009
ef_000  \
count  9.950000e+02  9.950000e+02  9.950000e+02  9.950000e+02
623.000000
mean   3.741013e+06  2.470956e+06  9.505801e+05  5.058748e+04
0.857143
std    4.602131e+06  3.992694e+06  2.179326e+06  2.208744e+05
14.982642
min    0.000000e+00  0.000000e+00  0.000000e+00  0.000000e+00
0.000000
25%    5.307930e+05  1.415090e+05  7.380000e+03  0.000000e+00
0.000000
50%    2.185156e+06  9.115520e+05  6.424800e+04  8.000000e+00
0.000000
75%    5.433839e+06  3.104029e+06  4.816260e+05  1.946000e+03
0.000000
max    3.160781e+07  2.605551e+07  1.926740e+07  3.810078e+06
362.000000

           eg_000
count  623.000000
mean     1.688604
std     25.773175
```

```
min      0.000000
25%      0.000000
50%      0.000000
75%      0.000000
max    606.000000

[8 rows x 170 columns]
Neg
             aa_000        ab_000        ac_000        ad_000
ae_000  \
count  5.900000e+04  13442.000000  5.612700e+04  4.478400e+04
56841.000000
mean   4.916977e+04      0.698706  3.588573e+08  1.921137e+05
6.664802
std    1.100525e+05      3.458102  7.974031e+08  4.056424e+07
160.498622
min    0.000000e+00      0.000000  0.000000e+00  0.000000e+00
0.000000
25%    7.840000e+02      0.000000  1.600000e+01  2.400000e+01
0.000000
50%    3.041600e+04      0.000000  1.500000e+02  1.240000e+02
0.000000
75%    4.549300e+04      0.000000  9.420000e+02  4.260000e+02
0.000000
max    2.434708e+06    204.000000  2.130707e+09  8.584298e+09
21050.000000

             af_000        ag_000         ag_001        ag_002
ag_003  \
count  56841.000000  5.833300e+04   58333.000000  5.833300e+04
5.833300e+04
mean      10.497845  1.885413e+02     120.136115  1.915646e+03
4.540666e+04
std      200.075183  2.044559e+04    5229.047509  5.711983e+04
4.138746e+05
min        0.000000  0.000000e+00       0.000000  0.000000e+00
0.000000e+00
25%        0.000000  0.000000e+00       0.000000  0.000000e+00
0.000000e+00
50%        0.000000  0.000000e+00       0.000000  0.000000e+00
0.000000e+00
75%        0.000000  0.000000e+00       0.000000  0.000000e+00
0.000000e+00
max    20070.000000  3.376892e+06  618212.000000  7.771682e+06
6.340207e+07

       ...        ee_002        ee_003        ee_004        ee_005  \
count  ...  5.833400e+04  5.833400e+04  5.833400e+04  5.833400e+04
mean   ...  3.783093e+05  1.806113e+05  3.820779e+05  3.226419e+05
std    ...  8.843276e+05  4.089108e+05  8.943229e+05  6.681121e+05
```

```
min    ...  0.000000e+00  0.000000e+00  0.000000e+00  0.000000e+00
25%    ...  2.838500e+03  1.106000e+03  2.534500e+03  3.326500e+03
50%    ...  2.267640e+05  1.082340e+05  2.140620e+05  1.832650e+05
75%    ...  4.273700e+05  2.125540e+05  4.523675e+05  3.915910e+05
max    ...  7.793393e+07  3.775839e+07  9.715238e+07  3.411102e+07

             ee_006        ee_007        ee_008        ee_009
ef_000  \
count  5.833400e+04  5.833400e+04  5.833400e+04  5.833400e+04
56653.000000
mean   2.749289e+05  3.100308e+05  1.248823e+05  7.669136e+03
0.082149
std    7.747345e+05  1.639254e+06  3.363635e+05  3.781107e+04
4.101906
min    0.000000e+00  0.000000e+00  0.000000e+00  0.000000e+00
0.000000
25%    4.625000e+02  1.020000e+02  0.000000e+00  0.000000e+00
0.000000
50%    8.423600e+04  3.846400e+04  3.294000e+03  0.000000e+00
0.000000
75%    2.668760e+05  1.630720e+05  1.370910e+05  2.030000e+03
0.000000
max    2.811407e+07  1.195801e+08  1.404564e+07  2.708070e+06
482.000000

             eg_000
count  56654.000000
mean       0.196526
std        8.456985
min        0.000000
25%        0.000000
50%        0.000000
75%        0.000000
max     1146.000000

[8 rows x 170 columns]
```

#Data Preprocessing

```
import numpy as np
from sklearn.preprocessing import MinMaxScaler
def preprocess(df: pd.DataFrame) -> pd.DataFrame:
    df = df.replace('na', np.nan)
    df = df.drop(columns=coloumns_with_missing_values,
errors='ignore')
    df['class'] = pd.Categorical(df['class'])
    label_map = {'neg': 0, 'pos': 1}
    df['class'] = df['class'].map(label_map)
    numeric_cols = df.select_dtypes(include=np.number).columns
```

```
    df[numeric_cols] = df[numeric_cols].apply(lambda x:
x.fillna(x.median()))
    scaler = MinMaxScaler()
    df.iloc[:, 1:] = scaler.fit_transform(df.iloc[:, 1:])

    return df


data = preprocess(data)
data.head()
```

```
<ipython-input-13-ae3c08ae6b5c>:12: FutureWarning: Setting an item of
incompatible dtype is deprecated and will raise in a future error of
pandas. Value '[2.79250729e-02 1.20361295e-02 1.49423061e-02 ...
4.07782233e-05
 2.92336170e-02 1.46444794e-02]' has dtype incompatible with int64,
please explicitly cast to a compatible dtype first.
  df.iloc[:, 1:] = scaler.fit_transform(df.iloc[:, 1:])
```

{"type":"dataframe","variable_name":"data"}

Let’s learn to deal with class-imbalance this time! We will consider the IDA2016 Challenge dataset for our experimentation. The dataset is a binary classification y = {‘pos’, ‘neg’} problem with 170 features and 60,000 data points. The craziness here is that the class ratio is 1:59, that is, for every positive data point, there are 59 negative data points in the training data. The challenge dataset has a training file (aps_failure_training_set.csv) and a testing file (aps_failure_test_set.csv). We will consider only the training file for our experimentation.

Task 1

- Split the data file (aps_failure_training_set.csv) into train and test partitions.
- Build baseline classifiers {SVC, LogReg and DecisionTree} by cross-validating the best hyper-parameters of the respective models. For SVC, the hyperparametes are {kernel, kernel-params}; for LogReg {regularization choice L1/L2, regularization params}; and for DT {depth, leaf size}. Upon using GridSearchCV, the best parameters are to be found.
- Note that, GridSearchCV does 5-fold CV by default, which is sufficient for us. Once the parameters are fixed, you will learn the models on the train partition and report the performance metrics on the train and test partitions.

```
X=data.drop(['class'], axis=1)
y=data['class']
X_train,X_test,y_train,y_test=train_test_split(X,y,test_size=0.2,rando
m_state=355)

X_train
```

{"type":"dataframe","variable_name":"X_train"}

```
X_test
```

{"type":"dataframe","variable_name":"X_test"}

```
y_train
```

```
32318    0
879      0
35059    0
57467    0
4017     0
        ..
26675    0
7459     0
24053    0
19074    0
56062    0
Name: class, Length: 48000, dtype: category
Categories (2, int64): [0, 1]
```

```
y_test
```

```
42237    0
24143    0
20925    0
12715    0
14940    0
        ..
53820    0
49495    0
38980    0
38789    0
2014     0
Name: class, Length: 12000, dtype: category
Categories (2, int64): [0, 1]
```

```
classifiers = {
    'SVC': SVC(),
    'LogisticRegression': LogisticRegression(solver='liblinear'),
    'DecisionTree': DecisionTreeClassifier()
}

param_grids = {
    'SVC': {
        'kernel': ['linear', 'rbf'],
        'C': [0.1, 1, 10],
    },
    'LogisticRegression': {
        'penalty': ['l1', 'l2'],
        'C': [0.1, 1, 10]
    },
    'DecisionTree': {
```

```
        'max_depth': [10, 20, 30],
        'min_samples_leaf': [1, 2, 4]
    }
}

for model_name, model in classifiers.items():
    print(f"\nTraining {model_name}...")

    grid_search = GridSearchCV(model, param_grids[model_name], cv=5,
scoring='f1_macro')
    grid_search.fit(X_train, y_train)
    print(f"Best Parameters for {model_name}:
{grid_search.best_params_}")

    print(f"{model_name} - Training Performance:")
    y_train_pred = grid_search.predict(X_train)
    print(classification_report(y_train, y_train_pred))

    print(f"{model_name} - Test Performance:")
    y_test_pred = grid_search.predict(X_test)
    print(classification_report(y_test, y_test_pred))
```

```
Training SVC...
Best Parameters for SVC: {'C': 10, 'kernel': 'rbf'}
SVC - Training Performance:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00     47193
           1       1.00      0.90      0.95       807

    accuracy                           1.00     48000
   macro avg       1.00      0.95      0.97     48000
weighted avg       1.00      1.00      1.00     48000

SVC - Test Performance:
              precision    recall  f1-score   support

           0       0.99      1.00      1.00     11807
           1       0.82      0.64      0.72       193

    accuracy                           0.99     12000
   macro avg       0.91      0.82      0.86     12000
weighted avg       0.99      0.99      0.99     12000


Training LogisticRegression...
Best Parameters for LogisticRegression: {'C': 10, 'penalty': 'l1'}
LogisticRegression - Training Performance:
              precision    recall  f1-score   support
```

```
           0       0.99      1.00      1.00     47193
           1       0.83      0.64      0.72       807

    accuracy                           0.99     48000
   macro avg       0.91      0.82      0.86     48000
weighted avg       0.99      0.99      0.99     48000

LogisticRegression - Test Performance:
              precision    recall  f1-score   support

           0       0.99      1.00      1.00     11807
           1       0.81      0.67      0.73       193

    accuracy                           0.99     12000
   macro avg       0.90      0.83      0.86     12000
weighted avg       0.99      0.99      0.99     12000


Training DecisionTree...
Best Parameters for DecisionTree: {'max_depth': 10,
'min_samples_leaf': 1}
DecisionTree - Training Performance:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00     47193
           1       0.99      0.82      0.90       807

    accuracy                           1.00     48000
   macro avg       0.99      0.91      0.95     48000
weighted avg       1.00      1.00      1.00     48000

DecisionTree - Test Performance:
              precision    recall  f1-score   support

           0       0.99      1.00      1.00     11807
           1       0.83      0.65      0.73       193

    accuracy                           0.99     12000
   macro avg       0.91      0.82      0.86     12000
weighted avg       0.99      0.99      0.99     12000
```

Task 2

Now, we want to address the class imbalance via multiple approaches. You are expected to apply the following in all the three families of classifiers.

a) Consider undersampling the majority class and/or oversampling the minority class.

b) Consider using class_weight which is inversely proportional to the class population.

c) Consider using sample_weights, where you may assign a penalty for misclassifying every data point depending on the class it falls in.

d) Consider any other creative ideas to address the class imbalance.

The goal here is the classification performance metric (macro average F_1) of the hacked classifiers should be better than the baseline classifiers.

```
from sklearn.utils import resample
from imblearn.over_sampling import SMOTE
from imblearn.under_sampling import RandomUnderSampler
```

a) Consider undersampling the majority class and/or oversampling the minority class.

```
# a) Undersample the majority class
undersampler = RandomUnderSampler(sampling_strategy='majority',
random_state=42)
X_train, y_train = undersampler.fit_resample(X_train, y_train)

classifiers = {
    'SVC': SVC(),
    'LogisticRegression': LogisticRegression(solver='liblinear'),
    'DecisionTree': DecisionTreeClassifier()
}

param_grids = {
    'SVC': {
        'kernel': ['linear', 'rbf'],
        'C': [0.1, 1, 10],
    },
    'LogisticRegression': {
        'penalty': ['l1', 'l2'],
        'C': [0.1, 1, 10]
    },
    'DecisionTree': {
        'max_depth': [10, 20, 30],
        'min_samples_leaf': [1, 2, 4]
    }
}

for model_name, model in classifiers.items():
    print(f"\nTraining {model_name}...")

    grid_search = GridSearchCV(model, param_grids[model_name], cv=5,
scoring='f1_macro')
    grid_search.fit(X_train, y_train)
    print(f"Best Parameters for {model_name}:
{grid_search.best_params_}")
```

```
    print(f"{model_name} - Training Performance:")
    y_train_pred = grid_search.predict(X_train)
    print(classification_report(y_train, y_train_pred))

    print(f"{model_name} - Test Performance:")
    y_test_pred = grid_search.predict(X_test)
    print(classification_report(y_test, y_test_pred))
    print(f"Macro F1 Score: {f1_score(y_test, y_test_pred,
average='macro')}")
```

```
Training SVC...
Best Parameters for SVC: {'C': 10, 'kernel': 'linear'}
SVC - Training Performance:
              precision    recall  f1-score   support

           0       0.94      0.98      0.96       193
           1       0.98      0.93      0.96       193

    accuracy                           0.96       386
   macro avg       0.96      0.96      0.96       386
weighted avg       0.96      0.96      0.96       386

SVC - Test Performance:
              precision    recall  f1-score   support

           0       0.94      0.98      0.96       193
           1       0.98      0.93      0.96       193

    accuracy                           0.96       386
   macro avg       0.96      0.96      0.96       386
weighted avg       0.96      0.96      0.96       386

Macro F1 Score: 0.9585213840532989

Training LogisticRegression...
Best Parameters for LogisticRegression: {'C': 10, 'penalty': 'l1'}
LogisticRegression - Training Performance:
              precision    recall  f1-score   support

           0       0.95      0.98      0.97       193
           1       0.98      0.95      0.97       193

    accuracy                           0.97       386
   macro avg       0.97      0.97      0.97       386
weighted avg       0.97      0.97      0.97       386

LogisticRegression - Test Performance:
              precision    recall  f1-score   support
```

```
           0       0.95      0.98      0.97       193
           1       0.98      0.95      0.97       193

    accuracy                           0.97       386
   macro avg       0.97      0.97      0.97       386
weighted avg       0.97      0.97      0.97       386

Macro F1 Score: 0.9663101640180736

Training DecisionTree...
Best Parameters for DecisionTree: {'max_depth': 30,
'min_samples_leaf': 4}
DecisionTree - Training Performance:
              precision    recall  f1-score   support

           0       0.97      0.97      0.97       193
           1       0.97      0.97      0.97       193

    accuracy                           0.97       386
   macro avg       0.97      0.97      0.97       386
weighted avg       0.97      0.97      0.97       386

DecisionTree - Test Performance:
              precision    recall  f1-score   support

           0       0.97      0.97      0.97       193
           1       0.97      0.97      0.97       193

    accuracy                           0.97       386
   macro avg       0.97      0.97      0.97       386
weighted avg       0.97      0.97      0.97       386

Macro F1 Score: 0.9740932642487047
```

```python
# b) Oversample the minority class using SMOTE
smote = SMOTE(sampling_strategy='minority', random_state=42)
X_train, y_train = smote.fit_resample(X_train, y_train)

classifiers = {
    'SVC': SVC(),
    'LogisticRegression': LogisticRegression(solver='liblinear'),
    'DecisionTree': DecisionTreeClassifier()
}

param_grids = {
    'SVC': {
        'kernel': ['linear', 'rbf'],
        'C': [0.1, 1, 10],
    },
    'LogisticRegression': {
        'penalty': ['l1', 'l2'],
```

```python
        'C': [0.1, 1, 10]
    },
    'DecisionTree': {
        'max_depth': [10, 20, 30],
        'min_samples_leaf': [1, 2, 4]
    }
}

for model_name, model in classifiers.items():
    print(f"\nTraining {model_name}...")

    grid_search = GridSearchCV(model, param_grids[model_name], cv=5,
scoring='f1_macro')
    grid_search.fit(X_train, y_train)
    print(f"Best Parameters for {model_name}:
{grid_search.best_params_}")

    print(f"{model_name} - Training Performance:")
    y_train_pred = grid_search.predict(X_train)
    print(classification_report(y_train, y_train_pred))

    print(f"{model_name} - Test Performance:")
    y_test_pred = grid_search.predict(X_test)
    print(classification_report(y_test, y_test_pred))
    print(f"Macro F1 Score: {f1_score(y_test, y_test_pred,
average='macro')}")
```

```
Training SVC...
Best Parameters for SVC: {'C': 1, 'kernel': 'rbf'}
SVC - Training Performance:
              precision    recall  f1-score   support

           0       0.92      0.96      0.94       193
           1       0.96      0.92      0.94       193

    accuracy                           0.94       386
   macro avg       0.94      0.94      0.94       386
weighted avg       0.94      0.94      0.94       386

SVC - Test Performance:
              precision    recall  f1-score   support

           0       0.92      0.96      0.94       193
           1       0.96      0.92      0.94       193

    accuracy                           0.94       386
   macro avg       0.94      0.94      0.94       386
weighted avg       0.94      0.94      0.94       386

Macro F1 Score: 0.9377971154620901
```

```
Training LogisticRegression...
Best Parameters for LogisticRegression: {'C': 10, 'penalty': 'l1'}
LogisticRegression - Training Performance:
              precision    recall  f1-score   support

           0       0.95      0.98      0.97       193
           1       0.98      0.95      0.97       193

    accuracy                           0.97       386
   macro avg       0.97      0.97      0.97       386
weighted avg       0.97      0.97      0.97       386

LogisticRegression - Test Performance:
              precision    recall  f1-score   support

           0       0.95      0.98      0.97       193
           1       0.98      0.95      0.97       193

    accuracy                           0.97       386
   macro avg       0.97      0.97      0.97       386
weighted avg       0.97      0.97      0.97       386

Macro F1 Score: 0.9663101640180736

Training DecisionTree...
Best Parameters for DecisionTree: {'max_depth': 30,
'min_samples_leaf': 1}
DecisionTree - Training Performance:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00       193
           1       1.00      1.00      1.00       193

    accuracy                           1.00       386
   macro avg       1.00      1.00      1.00       386
weighted avg       1.00      1.00      1.00       386

DecisionTree - Test Performance:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00       193
           1       1.00      1.00      1.00       193

    accuracy                           1.00       386
   macro avg       1.00      1.00      1.00       386
weighted avg       1.00      1.00      1.00       386

Macro F1 Score: 1.0
```

b) Consider using class_weight which is inversely proportional to the class population.

```python
from sklearn.metrics import classification_report, f1_score

classifiers = {
    'SVC': SVC(class_weight='balanced'),
    'LogisticRegression': LogisticRegression(class_weight='balanced',
solver='liblinear'),
    'DecisionTree': DecisionTreeClassifier(class_weight='balanced')
}

param_grids = {
    'SVC': {'kernel': ['linear', 'rbf'], 'C': [0.1, 1, 10], 'gamma':
['scale', 'auto']},
    'LogisticRegression': {'penalty': ['l1', 'l2'], 'C': [0.1, 1,
10]},
    'DecisionTree': {'max_depth': [10, 20, 30], 'min_samples_leaf':
[1, 2, 4]}
}

# Train classifiers using GridSearchCV with class_weight='balanced'
for model_name, model in classifiers.items():
    print(f"\nTraining {model_name} with class_weight...")

    grid_search = GridSearchCV(model, param_grids[model_name], cv=5,
scoring='f1_macro')
    grid_search.fit(X_train, y_train)

    # Evaluate performance on the test set
    y_test_pred = grid_search.predict(X_test)
    print(f"Test Performance for {model_name}:")
    print(classification_report(y_test, y_test_pred))
    print(f"Macro F1 Score: {f1_score(y_test, y_test_pred,
average='macro')}")
```

```
Training SVC with class_weight...
Test Performance for SVC:
              precision    recall  f1-score   support

           0       0.92      0.96      0.94       193
           1       0.96      0.92      0.94       193

    accuracy                           0.94       386
   macro avg       0.94      0.94      0.94       386
weighted avg       0.94      0.94      0.94       386

Macro F1 Score: 0.9377971154620901

Training LogisticRegression with class_weight...
Test Performance for LogisticRegression:
              precision    recall  f1-score   support
```

```
           0       0.95      0.98      0.97       193
           1       0.98      0.95      0.97       193

    accuracy                           0.97       386
   macro avg       0.97      0.97      0.97       386
weighted avg       0.97      0.97      0.97       386

Macro F1 Score: 0.9663101640180736

Training DecisionTree with class_weight...
Test Performance for DecisionTree:
              precision    recall  f1-score   support

           0       0.97      0.97      0.97       193
           1       0.97      0.97      0.97       193

    accuracy                           0.97       386
   macro avg       0.97      0.97      0.97       386
weighted avg       0.97      0.97      0.97       386

Macro F1 Score: 0.9740932642487047
```

c) Consider using sample_weights, where you may assign a penalty for misclassifying every data point depending on the class it falls in.

```
sample_weights = y_train.apply(lambda x: 1 if x == 0 else 59)  # Assign higher weight to the minority class

classifiers = {
    'SVC': SVC(),
    'LogisticRegression': LogisticRegression(solver='liblinear'),
    'DecisionTree': DecisionTreeClassifier()
}

param_grids = {
    'SVC': {
        'kernel': ['linear', 'rbf'],
        'C': [0.1, 1, 10],
    },
    'LogisticRegression': {
        'penalty': ['l1', 'l2'],
        'C': [0.1, 1, 10]
    },
    'DecisionTree': {
        'max_depth': [10, 20, 30],
        'min_samples_leaf': [1, 2, 4]
    }
}

for model_name, model in classifiers.items():
```

```
    print(f"\nTraining {model_name}...")

    grid_search = GridSearchCV(model, param_grids[model_name], cv=5,
scoring='f1_macro')
    grid_search.fit(X_train, y_train,sample_weight=sample_weights)
    print(f"Best Parameters for {model_name}:
{grid_search.best_params_}")

    print(f"{model_name} - Training Performance:")
    y_train_pred = grid_search.predict(X_train)
    print(classification_report(y_train, y_train_pred))

    print(f"{model_name} - Test Performance:")
    y_test_pred = grid_search.predict(X_test)
    print(classification_report(y_test, y_test_pred))
    print(f"Macro F1 Score: {f1_score(y_test, y_test_pred,
average='macro')}")
```

```
Training SVC...
Best Parameters for SVC: {'C': 10, 'kernel': 'rbf'}
SVC - Training Performance:
              precision    recall  f1-score   support

           0       1.00      0.52      0.68       193
           1       0.67      1.00      0.81       193

    accuracy                           0.76       386
   macro avg       0.84      0.76      0.74       386
weighted avg       0.84      0.76      0.74       386

SVC - Test Performance:
              precision    recall  f1-score   support

           0       1.00      0.52      0.68       193
           1       0.67      1.00      0.81       193

    accuracy                           0.76       386
   macro avg       0.84      0.76      0.74       386
weighted avg       0.84      0.76      0.74       386

Macro F1 Score: 0.7442196840687725

Training LogisticRegression...
Best Parameters for LogisticRegression: {'C': 10, 'penalty': 'l1'}
LogisticRegression - Training Performance:
              precision    recall  f1-score   support

           0       1.00      0.53      0.69       193
           1       0.68      1.00      0.81       193
```

```
    accuracy                           0.76       386
   macro avg       0.84      0.76      0.75       386
weighted avg       0.84      0.76      0.75       386

LogisticRegression - Test Performance:
              precision    recall  f1-score   support

           0       1.00      0.53      0.69       193
           1       0.68      1.00      0.81       193

    accuracy                           0.76       386
   macro avg       0.84      0.76      0.75       386
weighted avg       0.84      0.76      0.75       386

Macro F1 Score: 0.7503748711935472

Training DecisionTree...
Best Parameters for DecisionTree: {'max_depth': 20,
'min_samples_leaf': 2}
DecisionTree - Training Performance:
              precision    recall  f1-score   support

           0       1.00      0.97      0.99       193
           1       0.97      1.00      0.99       193

    accuracy                           0.99       386
   macro avg       0.99      0.99      0.99       386
weighted avg       0.99      0.99      0.99       386

DecisionTree - Test Performance:
              precision    recall  f1-score   support

           0       1.00      0.97      0.99       193
           1       0.97      1.00      0.99       193

    accuracy                           0.99       386
   macro avg       0.99      0.99      0.99       386
weighted avg       0.99      0.99      0.99       386

Macro F1 Score: 0.9870444583173906
```

d) Consider any other creative ideas to address the class imbalance.

```
# Combine SMOTE (oversampling) with class_weight='balanced' for better
results
smote = SMOTE(sampling_strategy='minority', random_state=42)
X_train, y_train = smote.fit_resample(X_train, y_train)

classifiers = {
    'SVC': SVC(class_weight='balanced'),
```

```
    'LogisticRegression': LogisticRegression(class_weight='balanced',
solver='liblinear'),
    'DecisionTree': DecisionTreeClassifier(class_weight='balanced')
}

param_grids = {
    'SVC': {'kernel': ['linear', 'rbf'], 'C': [0.1, 1, 10], 'gamma':
['scale', 'auto']},
    'LogisticRegression': {'penalty': ['l1', 'l2'], 'C': [0.1, 1,
10]},
    'DecisionTree': {'max_depth': [10, 20, 30], 'min_samples_leaf':
[1, 2, 4]}
}

# Train classifiers using GridSearchCV with class_weight='balanced'
for model_name, model in classifiers.items():
    print(f"\nTraining {model_name} with class_weight...")

    grid_search = GridSearchCV(model, param_grids[model_name], cv=5,
scoring='f1_macro')
    grid_search.fit(X_train, y_train)

    # Evaluate performance on the test set
    y_test_pred = grid_search.predict(X_test)
    print(f"Test Performance for {model_name}:")
    print(classification_report(y_test, y_test_pred))
    print(f"Macro F1 Score: {f1_score(y_test, y_test_pred,
average='macro')}")
```

```
Training SVC with class_weight...
Test Performance for SVC:
              precision    recall  f1-score   support

           0       0.92      0.96      0.94       193
           1       0.96      0.92      0.94       193

    accuracy                           0.94       386
   macro avg       0.94      0.94      0.94       386
weighted avg       0.94      0.94      0.94       386

Macro F1 Score: 0.9377971154620901

Training LogisticRegression with class_weight...
Test Performance for LogisticRegression:
              precision    recall  f1-score   support

           0       0.95      0.98      0.97       193
           1       0.98      0.95      0.97       193

    accuracy                           0.97       386
```

```
   macro avg       0.97      0.97      0.97       386
weighted avg       0.97      0.97      0.97       386

Macro F1 Score: 0.9663101640180736

Training DecisionTree with class_weight...
Test Performance for DecisionTree:
              precision    recall  f1-score   support

           0       0.97      0.97      0.97       193
           1       0.97      0.97      0.97       193

    accuracy                           0.97       386
   macro avg       0.97      0.97      0.97       386
weighted avg       0.97      0.97      0.97       386

Macro F1 Score: 0.9740932642487047
```

Test performance from task 1(imbalanced dataset) for all the model.

- We can see that in all the models for class 1(positive) , precision,recall and f1-score performance is very low as compared to class 0(negative).
- As we know that in imbalanced class, the majority class have high performance as compared to the minoriuty class. This trend we can see from this.

```
DecisionTree - Test Performance:
              precision    recall  f1-score   support

           0       0.99      1.00      1.00     11807
           1       0.83      0.65      0.73       193

LogisticRegression - Test Performance:
              precision    recall  f1-score   support

           0       0.99      1.00      1.00     11807
           1       0.81      0.67      0.73       193


SVC - Test Performance:
              precision    recall  f1-score   support

           0       0.99      1.00      1.00     11807
           1       0.82      0.64      0.72       193
```

Test performance from task 2(balanced dataset) for all the models

a) Consider undersampling the majority class and/or oversampling the minority class.

Undersampling

```
DecisionTree - Test Performance:
              precision    recall  f1-score   support

           0       0.97      0.97      0.97       193
           1       0.97      0.97      0.97       193

LogisticRegression - Test Performance:
              precision    recall  f1-score   support

           0       0.95      0.98      0.97       193
           1       0.98      0.95      0.97       193

SVC - Test Performance:
              precision    recall  f1-score   support

           0       0.94      0.98      0.96       193
           1       0.98      0.93      0.96       193
```

Oversampling

```
SVC - Test Performance:
              precision    recall  f1-score   support

           0       0.92      0.96      0.94       193
           1       0.96      0.92      0.94       193

DecisionTree - Test Performance:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00       193
           1       1.00      1.00      1.00       193

LogisticRegression - Test Performance:
              precision    recall  f1-score   support

           0       0.95      0.98      0.97       193
           1       0.98      0.95      0.97       193
```

b) Consider using class_weight which is inversely proportional to the class population.

```
Test Performance for DecisionTree:
              precision    recall  f1-score   support

           0       0.97      0.97      0.97       193
           1       0.97      0.97      0.97       193
Test Performance for LogisticRegression:
              precision    recall  f1-score   support

           0       0.95      0.98      0.97       193
           1       0.98      0.95      0.97       193
Test Performance for SVC:
              precision    recall  f1-score   support

           0       0.92      0.96      0.94       193
           1       0.96      0.92      0.94       193
```

c) Consider using sample_weights, where you may assign a penalty for misclassifying every data point depending on the class it falls in.

```
DecisionTree - Test Performance:
              precision    recall  f1-score   support

           0       1.00      0.97      0.99       193
           1       0.97      1.00      0.99       193
LogisticRegression - Test Performance:
              precision    recall  f1-score   support

           0       1.00      0.53      0.69       193
           1       0.68      1.00      0.81       193
SVC - Test Performance:
              precision    recall  f1-score   support

           0       1.00      0.52      0.68       193
           1       0.67      1.00      0.81       193
```

d) Consider any other creative ideas to address the class imbalance.

```
Test Performance for DecisionTree:
              precision    recall  f1-score   support

           0       0.97      0.97      0.97       193
           1       0.97      0.97      0.97       193
Test Performance for LogisticRegression:
              precision    recall  f1-score   support

           0       0.95      0.98      0.97       193
           1       0.98      0.95      0.97       193
Test Performance for SVC:
              precision    recall  f1-score   support

           0       0.92      0.96      0.94       193
           1       0.96      0.92      0.94       193
```

```
"The End"
```