# Nikshay Jain | MM21B044

## Assign 7

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

from sklearn.svm import SVC
from sklearn.impute import SimpleImputer
from sklearn.tree import DecisionTreeClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import f1_score, classification_report
from sklearn.preprocessing import StandardScaler
from sklearn.utils.class_weight import compute_sample_weight
from sklearn.model_selection import GridSearchCV, train_test_split, ParameterGrid

from imblearn.under_sampling import RandomUnderSampler
from imblearn.over_sampling import RandomOverSampler, SMOTE
```

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

data = pd.read_csv('/content/drive/MyDrive/aps_failure_training_set.csv')
```

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

```
# data = pd.read_csv('aps_failure_training_set.csv')
# data
```

```
len(data)/sum(data['class']=='pos')  # tot vs +ves in dataset
```

```
60.0
```

```
data[data.columns[0]].replace({'neg':'0','pos':'1'},inplace=True)
data = data.replace('na',np.NaN)
data[data.columns[0]] = data[data.columns[0]].astype(int)
data[data.columns[1:]] = data[data.columns[1:]].astype(float)
data.dropna(inplace=True)
```

```
<ipython-input-4-19ff5df2fc7f>:1: FutureWarning: A value is trying to be set on a copy of a DataFrame or Series through chained assignme
The behavior will change in pandas 3.0. This inplace method will never work because the intermediate object on which we are setting valu

For example, when doing 'df[col].method(value, inplace=True)', try using 'df.method({col: value}, inplace=True)' or df[col] = df[col].me


  data[data.columns[0]].replace({'neg':'0','pos':'1'},inplace=True)
```

```
data
```

| | class | aa_000 | ab_000 | ac_000 | ad_000 | ae_000 | af_000 | ag_000 | ag_001 | ag_002 | ... | ee_002 | ee_003 | ee_004 | ee_005 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **16** | 0 | 31300.0 | 0.0 | 784.0 | 740.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | ... | 798872.0 | 112724.0 | 51736.0 | 7054.0 |
| **179** | 0 | 97000.0 | 0.0 | 378.0 | 160.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | ... | 1078982.0 | 313334.0 | 511330.0 | 552328.0 |
| **225** | 0 | 124656.0 | 2.0 | 278.0 | 170.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | ... | 1205696.0 | 866148.0 | 697610.0 | 700400.0 |
| **394** | 1 | 281324.0 | 2.0 | 3762.0 | 2346.0 | 0.0 | 0.0 | 4808.0 | 215720.0 | 967572.0 | ... | 624606.0 | 269976.0 | 638838.0 | 1358354.0 |
| **413** | 1 | 43482.0 | 0.0 | 1534.0 | 1388.0 | 0.0 | 0.0 | 0.0 | 0.0 | 40024.0 | ... | 497196.0 | 121166.0 | 202272.0 | 232636.0 |
| **...** | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... |
| **59432** | 0 | 118028.0 | 0.0 | 740.0 | 714.0 | 618.0 | 690.0 | 0.0 | 0.0 | 0.0 | ... | 838952.0 | 631338.0 | 541036.0 | 1285274.0 |
| **59562** | 0 | 229916.0 | 0.0 | 616.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | ... | 0.0 | 0.0 | 0.0 | 0.0 |
| **59843** | 0 | 224084.0 | 0.0 | 912.0 | 766.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | ... | 413576.0 | 209524.0 | 469894.0 | 2233992.0 |
| **59870** | 0 | 197332.0 | 0.0 | 658.0 | 616.0 | 216.0 | 346.0 | 0.0 | 0.0 | 0.0 | ... | 73940.0 | 49896.0 | 90454.0 | 575264.0 |
| **59950** | 0 | 76812.0 | 0.0 | 376.0 | 340.0 | 0.0 | 0.0 | 0.0 | 0.0 | 5744.0 | ... | 1487016.0 | 60100.0 | 29754.0 | 4200.0 |

591 rows × 171 columns

## Task 1

```
y = data['class']
X = data.drop('class',axis=1)
```

```
y = data[data.columns[0]]
X = data[data.columns[1:]]
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

Train baseline models

SVC - 0.91 f1 score

```
svc_param = {
    'kernel': ['linear', 'rbf', 'sigmoid'],
    'C': [1e2, 1e3, 1e4, 1e5],
    'gamma': ['scale', 'auto']
}

svc_raw = GridSearchCV(SVC(), param_grid=svc_param, cv=5, n_jobs=-1, scoring='f1_macro')
svc_raw.fit(X_train, y_train)
svc_best = svc_raw.best_estimator_

print("Best Params for SVC:", svc_raw.best_params_)

svc_pred = svc_best.predict(X_test)
macro_f1_svc = f1_score(y_test, svc_pred, average='macro')
print("Macro-average F1-score for SVC:", macro_f1_svc)
print(classification_report(y_test, svc_pred))
```

```
Best Params for SVC: {'C': 10000.0, 'gamma': 'scale', 'kernel': 'rbf'}
Macro-average F1-score for SVC: 0.9073208722741433
              precision    recall  f1-score   support

           0       0.97      0.99      0.98       106
           1       0.91      0.77      0.83        13

    accuracy                           0.97       119
   macro avg       0.94      0.88      0.91       119
weighted avg       0.97      0.97      0.97       119
```

Logistic Regression - 0.89 f1 score

```
logreg_params = {
    'penalty': ['l2', 'l1'],
    'C': [1e-4, 1e-3, 1e-2, 1],
    'solver': ['liblinear', 'saga']
```

```
}

logreg_raw = GridSearchCV(LogisticRegression(), param_grid=logreg_params, cv=5, n_jobs=-1, scoring='f1_macro')
logreg_raw.fit(X_train, y_train)
logreg_best = logreg_raw.best_estimator_

print("Best Params for Logistic Regression:", logreg_raw.best_params_)

logreg_pred = logreg_best.predict(X_test)
macro_f1_logreg = f1_score(y_test, logreg_pred, average='macro')
print("Macro-average F1-score for Logistic regression:", macro_f1_logreg)
print(classification_report(y_test, logreg_pred))
```

```
Best Params for Logistic Regression: {'C': 0.001, 'penalty': 'l1', 'solver': 'liblinear'}
Macro-average F1-score for Logistic regression: 0.8882629107981221
              precision    recall  f1-score   support

           0       0.97      0.98      0.98       106
           1       0.83      0.77      0.80        13

    accuracy                           0.96       119
   macro avg       0.90      0.88      0.89       119
weighted avg       0.96      0.96      0.96       119
```

Decision Tree - 0.86 f1 score

```
dt_params = {
    'max_depth': [2, 3, 5, 10, 15],
    'min_samples_leaf': [1, 2, 4, 6, 8],
    'min_samples_split': [2, 3, 4, 6]
}

dt_raw = GridSearchCV(DecisionTreeClassifier(), param_grid=dt_params, cv=5, n_jobs=-1, scoring='f1_macro')
dt_raw.fit(X_train, y_train)
dt_best = dt_raw.best_estimator_

print("Best Params for Decision Tree:", dt_raw.best_params_)

dt_pred = dt_best.predict(X_test)
macro_f1_dt = f1_score(y_test, dt_pred, average='macro')
print("Macro-average F1-score for decision tree :", macro_f1_dt)
print(classification_report(y_test, dt_pred))
```

```
Best Params for Decision Tree: {'max_depth': 5, 'min_samples_leaf': 1, 'min_samples_split': 4}
Macro-average F1-score for decision tree : 0.860981308411215
              precision    recall  f1-score   support

           0       0.96      0.98      0.97       106
           1       0.82      0.69      0.75        13

    accuracy                           0.95       119
   macro avg       0.89      0.84      0.86       119
weighted avg       0.95      0.95      0.95       119
```

## Task 2

Task 2a - oversampling

```
over_sampler = RandomOverSampler(random_state=42)
X_ros, y_ros = over_sampler.fit_resample(X_train, y_train)


svc_param_new = {
    'kernel': ['linear', 'rbf', 'sigmoid'],
    'C': [1e-12, 1e-10, 1e-9],
    'gamma': ['scale', 'auto']
}

svc_new1 = GridSearchCV(SVC(tol=0.03), param_grid=svc_param_new, cv=5, n_jobs=-1, scoring='f1_macro')
svc_new1.fit(X_ros, y_ros)
svc_new1_best = svc_new1.best_estimator_
```

```
print("Best Params for SVC:", svc_new1.best_params_)

svc_pred_new1 = svc_new1_best.predict(X_test)
macro_f1_svc_new1 = f1_score(y_test, svc_pred_new1, average='macro')
print("Macro-average F1-score for SVC:", macro_f1_svc_new1)
print(classification_report(y_test, svc_pred_new1))
```

```
Best Params for SVC: {'C': 1e-10, 'gamma': 'scale', 'kernel': 'linear'}
Macro-average F1-score for SVC: 0.9018313809602376
              precision    recall  f1-score   support

           0       0.99      0.96      0.98       106
           1       0.75      0.92      0.83        13

    accuracy                           0.96       119
   macro avg       0.87      0.94      0.90       119
weighted avg       0.96      0.96      0.96       119
```

```
logreg_params_new = {
    'penalty': ['l2', 'l1'],
    'C': [1e-11, 1e-10, 1e-9, 1e-8]
}

logreg_new1 = GridSearchCV(LogisticRegression(solver='saga', max_iter=1000, tol=1e-3), param_grid=logreg_params_new, cv=5, n_jobs=-1, scorir
logreg_new1.fit(X_ros, y_ros)
logreg_new1_best = logreg_new1.best_estimator_

print("Best Params for Logistic Regression:", logreg_new1.best_params_)

logreg_pred_new1 = logreg_new1_best.predict(X_test)
macro_f1_logreg_new1 = f1_score(y_test, logreg_pred_new1, average='macro')
print("Macro-average F1-score for Logistic regression:", macro_f1_logreg_new1)
print(classification_report(y_test, logreg_pred_new1))
```

```
Best Params for Logistic Regression: {'C': 1e-10, 'penalty': 'l2'}
Macro-average F1-score for Logistic regression: 0.7700038654812524
              precision    recall  f1-score   support

           0       0.99      0.87      0.92       106
           1       0.46      0.92      0.62        13

    accuracy                           0.87       119
   macro avg       0.73      0.90      0.77       119
weighted avg       0.93      0.87      0.89       119
```

```
dt_params_new = {
    'max_depth': [5, 10, 15, 20],
    'min_samples_leaf': [1, 3, 5],
    'min_samples_split': [2, 3, 4, 5, 6]
}

dt_new1 = GridSearchCV(DecisionTreeClassifier(), param_grid=dt_params_new, cv=5, n_jobs=-1, scoring='f1_macro')
dt_new1.fit(X_ros, y_ros)
dt_new1_best = dt_new1.best_estimator_

print("Best Params for Decision Tree:", dt_new1.best_params_)

dt_pred_new1 = dt_new1_best.predict(X_test)
macro_f1_dt_new1 = f1_score(y_test, dt_pred_new1, average='macro')
print("Macro-average F1-score for decision tree :", macro_f1_dt_new1)
print(classification_report(y_test, dt_pred_new1))
```

```
Best Params for Decision Tree: {'max_depth': 15, 'min_samples_leaf': 3, 'min_samples_split': 2}
Macro-average F1-score for decision tree : 0.7988732394366197
              precision    recall  f1-score   support

           0       0.95      0.96      0.96       106
           1       0.67      0.62      0.64        13

    accuracy                           0.92       119
   macro avg       0.81      0.79      0.80       119
weighted avg       0.92      0.92      0.92       119
```

Task 2b - class weights

```
wt = np.where(y_train == 0,(1/np.sum(y_train == 0)),(1/np.sum(y_train == 1)))
wt = wt / np.sum(wt)

samples = np.random.choice(np.arange(0, len(X_train)), p=wt, size=len(X_train))

X_class_weight = X_train.iloc[samples]
y_class_weight = y_train.iloc[samples]


svc_param_new2 = {
    'kernel': ['linear', 'rbf', 'sigmoid'],
    'C': [1e-3, 1e-2, 1, 10],
    'gamma': ['scale', 'auto']
}

svc_new2 = GridSearchCV(SVC(max_iter=1000, class_weight='balanced'), param_grid=svc_param_new2, cv=5, n_jobs=-1, scoring='f1_macro')
svc_new2.fit(X_class_weight, y_class_weight)
svc_new2_best = svc_new2.best_estimator_

print("Best Params for SVC:", svc_new2.best_params_)

svc_pred_new2 = svc_new2_best.predict(X_test)
macro_f1_svc_new2 = f1_score(y_test, svc_pred_new2, average='macro')
print("Macro-average F1-score for SVC:", macro_f1_svc_new2)
print(classification_report(y_test, svc_pred_new2, zero_division=1))
```

```
Best Params for SVC: {'C': 1, 'gamma': 'auto', 'kernel': 'rbf'}
Macro-average F1-score for SVC: 0.4711111111111111
              precision    recall  f1-score   support

           0       0.89      1.00      0.94       106
           1       1.00      0.00      0.00        13

    accuracy                           0.89       119
   macro avg       0.95      0.50      0.47       119
weighted avg       0.90      0.89      0.84       119
```

```
logreg_params_new2 = {
    'penalty': ['l2', 'l1'],
    'C': [1e-12, 1e-10, 1e-8, 1e-6, 1e-4, 1e-2],
    'solver': ['liblinear', 'saga']
}

logreg_new2 = GridSearchCV(LogisticRegression(max_iter=1000, tol=1e-3), param_grid=logreg_params_new2, cv=5, n_jobs=-1, scoring='f1_macro')
logreg_new2.fit(X_class_weight, y_class_weight)
logreg_new2_best = logreg_new2.best_estimator_

print("Best Params for Logistic Regression:", logreg_new2.best_params_)

logreg_pred_new2 = logreg_new2_best.predict(X_test)
macro_f1_logreg_new2 = f1_score(y_test, logreg_pred_new2, average='macro')
print("Macro-average F1-score for Logistic regression:", macro_f1_logreg_new2)
print(classification_report(y_test, logreg_pred_new2))
```

```
Best Params for Logistic Regression: {'C': 0.0001, 'penalty': 'l1', 'solver': 'liblinear'}
Macro-average F1-score for Logistic regression: 0.9373354397051079
              precision    recall  f1-score   support

           0       0.99      0.98      0.99       106
           1       0.86      0.92      0.89        13

    accuracy                           0.97       119
   macro avg       0.92      0.95      0.94       119
weighted avg       0.98      0.97      0.98       119
```

```
dt_params_new2 = {
    'max_depth': [5, 10, 15, 20],
    'min_samples_leaf': [1, 3, 5],
    'min_samples_split': [2, 3, 4, 5, 6]
}

dt_new2 = GridSearchCV(DecisionTreeClassifier(), param_grid=dt_params_new2, cv=5, n_jobs=-1, scoring='f1_macro')
dt_new2.fit(X_class_weight, y_class_weight)
dt_new2_best = dt_new2.best_estimator_
```

```
print("Best Params for Decision Tree:", dt_new2.best_params_)

dt_pred_new2 = dt_new2_best.predict(X_test)
macro_f1_dt_new2 = f1_score(y_test, dt_pred_new2, average='macro')
print("Macro-average F1-score for decision tree :", macro_f1_dt_new2)
print(classification_report(y_test, dt_pred_new2))
```

```
Best Params for Decision Tree: {'max_depth': 20, 'min_samples_leaf': 1, 'min_samples_split': 6}
Macro-average F1-score for decision tree : 0.8315470171890799
              precision    recall  f1-score   support

           0       0.95      0.98      0.97       106
           1       0.80      0.62      0.70        13

    accuracy                           0.94       119
   macro avg       0.88      0.80      0.83       119
weighted avg       0.94      0.94      0.94       119
```

Task 2c - sample weights

```
svc_param_new3 = {
    'kernel': ['linear', 'rbf', 'sigmoid'],
    'C': [1e-12, 1e-10, 1e-8, 1e-6, 1e-4, 1e-2],
    'gamma': ['scale', 'auto']
}

svc_new3 = GridSearchCV(SVC(tol=0.03), param_grid=svc_param_new3, cv=5, n_jobs=-1, scoring='f1_macro')
svc_new3.fit(X_train, y_train, sample_weight=wt)
svc_new3_best = svc_new3.best_estimator_

print("Best Params for SVC:", svc_new3.best_params_)

svc_pred_new3 = svc_new3_best.predict(X_test)
macro_f1_svc_new3 = f1_score(y_test, svc_pred_new3, average='macro')
print("Macro-average F1-score for SVC:", macro_f1_svc_new3)
print(classification_report(y_test, svc_pred_new3))
```

```
Best Params for SVC: {'C': 1e-10, 'gamma': 'scale', 'kernel': 'linear'}
Macro-average F1-score for SVC: 0.9373354397051079
              precision    recall  f1-score   support

           0       0.99      0.98      0.99       106
           1       0.86      0.92      0.89        13

    accuracy                           0.97       119
   macro avg       0.92      0.95      0.94       119
weighted avg       0.98      0.97      0.98       119
```

```
logreg_params_new3 = {
    'penalty': ['l2', 'l1'],
    'C': [1e-3, 1e-2, 1e-1, 1, 10],
    'solver': ['liblinear', 'saga']
}

logreg_new3 = GridSearchCV(LogisticRegression(max_iter=1000, tol=1e-3), param_grid=logreg_params_new3, cv=5, n_jobs=-1, scoring='f1_macro')
logreg_new3.fit(X_train, y_train, sample_weight=wt)
logreg_new3_best = logreg_new3.best_estimator_

print("Best Params for Logistic Regression:", logreg_new3.best_params_)

logreg_pred_new3 = logreg_new3_best.predict(X_test)
macro_f1_logreg_new3 = f1_score(y_test, logreg_pred_new3, average='macro')
print("Macro-average F1-score for Logistic regression:", macro_f1_logreg_new3)
print(classification_report(y_test, logreg_pred_new3))
```

```
Best Params for Logistic Regression: {'C': 0.1, 'penalty': 'l1', 'solver': 'liblinear'}
Macro-average F1-score for Logistic regression: 0.8882629107981221
              precision    recall  f1-score   support

           0       0.97      0.98      0.98       106
           1       0.83      0.77      0.80        13

    accuracy                           0.96       119
   macro avg       0.90      0.88      0.89       119
weighted avg       0.96      0.96      0.96       119
```

```
dt_params_new3 = {
    'max_depth': [2, 3, 4, 5, 10],
    'min_samples_leaf': [1, 3, 5],
    'min_samples_split': [2, 4, 6, 8, 10]
}

dt_new3 = GridSearchCV(DecisionTreeClassifier(), param_grid=dt_params_new3, cv=5, n_jobs=-1, scoring='f1_macro')
dt_new3.fit(X_train, y_train, sample_weight=wt)
dt_new3_best = dt_new3.best_estimator_

print("Best Params for Decision Tree:", dt_new3.best_params_)

dt_pred_new3 = dt_new3_best.predict(X_test)
macro_f1_dt_new3 = f1_score(y_test, dt_pred_new3, average='macro')
print("Macro-average F1-score for decision tree :", macro_f1_dt_new3)
print(classification_report(y_test, dt_pred_new3))
```

```
Best Params for Decision Tree: {'max_depth': 5, 'min_samples_leaf': 3, 'min_samples_split': 2}
Macro-average F1-score for decision tree : 0.8537826926452519
              precision    recall  f1-score   support

           0       0.97      0.96      0.97       106
           1       0.71      0.77      0.74        13

    accuracy                           0.94       119
   macro avg       0.84      0.87      0.85       119
weighted avg       0.94      0.94      0.94       119
```

Task 2d - generate synthetic data

```
sm = SMOTE(random_state=42)
X_smote, y_smote = sm.fit_resample(X_train, y_train)
```

```
svc_param_new4 = {
    'kernel': ['linear', 'rbf'],
    'C': [0.1, 1, 10],
    'gamma': ['scale', 'auto']
}

svc_new4 = GridSearchCV(SVC(tol=0.03), param_grid=svc_param_new4, cv=5, n_jobs=-1, scoring='f1_macro')
svc_new4.fit(X_smote,y_smote)
svc_new4_best = svc_new4.best_estimator_

print("Best Params for SVC:", svc_new4.best_params_)

svc_pred_new4 = svc_new4_best.predict(X_test)
macro_f1_svc_new4 = f1_score(y_test, svc_pred_new4, average='macro')
print("Macro-average F1-score for SVC:", macro_f1_svc_new4)
print(classification_report(y_test, svc_pred_new4))
```

...

```
logreg_params_new4 = {
    'penalty': ['l2', 'l1'],
    'C': [1e-4, 1e-2, 1],
```

```
    'solver': ['liblinear', 'saga']
}

logreg_new4 = GridSearchCV(LogisticRegression(max_iter=1000, tol=1e-3), param_grid=logreg_params_new4, cv=5, n_jobs=-1, scoring='f1_macro'
logreg_new4.fit(X_smote, y_smote)
logreg_new4_best = logreg_new4.best_estimator_

print("Best Params for Logistic Regression:", logreg_new4.best_params_)

logreg_pred_new4 = logreg_new4_best.predict(X_test)
macro_f1_logreg_new4 = f1_score(y_test, logreg_pred_new4, average='macro')
print("Macro-average F1-score for Logistic regression:", macro_f1_logreg_new4)
```

```
Best Params for Logistic Regression: {'C': 0.01, 'penalty': 'l1', 'solver': 'liblinear'}
Macro-average F1-score for Logistic regression: 0.8172043010752688
              precision    recall  f1-score   support

           0       0.95      0.99      0.97       106
           1       0.88      0.54      0.67        13

    accuracy                           0.94       119
   macro avg       0.91      0.76      0.82       119
weighted avg       0.94      0.94      0.93       119
```

```
dt_params_new4 = {
    'max_depth': [5, 10, 15],
    'min_samples_leaf': [1, 3, 5],
    'min_samples_split': [2, 4, 6]
}

dt_new4 = GridSearchCV(DecisionTreeClassifier(), param_grid=dt_params_new4, cv=5, n_jobs=-1, scoring='f1_macro')
dt_new4.fit(X_smote, y_smote)
dt_new4_best = dt_new4.best_estimator_

print("Best Params for Decision Tree:", dt_new4.best_params_)

dt_pred_new4 = dt_new4_best.predict(X_test)
macro_f1_dt_new4 = f1_score(y_test, dt_pred_new4, average='macro')
print("Macro-average F1-score for decision tree :", macro_f1_dt_new4)
print(classification_report(y_test, dt_pred_new4))
```

```
Best Params for Decision Tree: {'max_depth': 10, 'min_samples_leaf': 1, 'min_samples_split': 4}
Macro-average F1-score for decision tree : 0.860981308411215
              precision    recall  f1-score   support

           0       0.96      0.98      0.97       106
           1       0.82      0.69      0.75        13

    accuracy                           0.95       119
   macro avg       0.89      0.84      0.86       119
weighted avg       0.95      0.95      0.95       119
```

## Final result:

The macro - F1 score shows that dealing with the class imbalance problem, the performance of best classification model has increased form 89% to 94% for Support Vector Classifier and Logistic Regression using class weights and sample weights respectively.

Start coding or generate with AI.