Assignment 7

Preethi MM21B051

Task 1:

We load the data set, convert pos to +1 and neg to 0. We covert all the data to np.float32 to speed up the model training and hyper parameter tuning process. Since the data set is huge and takes hours to train, we use a random sample of the dataset.

In [ ]:
```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split

# Load the dataset
file_path = 'aps_failure_training_set.csv'
data = pd.read_csv(file_path)

#convert 'pos'/'neg' to 1/0
data['class'] = data['class'].map({'pos': 1, 'neg': 0})

#replace 'na' values with NaN
data.replace('na', np.nan, inplace=True)
data = data.apply(pd.to_numeric, errors='coerce')

# fill missing values with the median of each column
data.fillna(data.median(), inplace=True)
data = data.astype(np.float32)
data = data.sample(3000)

X = data.drop(columns=['class'])
y = data['class']

# split the dataset into 80% training and 20% testing
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_sta

print(f"Training data shape: {X_train.shape}, Data type: {X_train.dtypes[0]}")
print(f"Testing data shape: {X_test.shape}, Data type: {X_test.dtypes[0]}")
```

```
Training data shape: (2400, 170), Data type: float32
Testing data shape: (600, 170), Data type: float32
```

```
C:\Users\preet\AppData\Local\Temp\ipykernel_11320\3570510493.py:39: FutureWarning: S
eries.__getitem__ treating keys as positions is deprecated. In a future version, int
eger keys will always be treated as labels (consistent with DataFrame behavior). To
access a value by position, use `ser.iloc[pos]`
  print(f"Training data shape: {X_train.shape}, Data type: {X_train.dtypes[0]}")
C:\Users\preet\AppData\Local\Temp\ipykernel_11320\3570510493.py:40: FutureWarning: S
eries.__getitem__ treating keys as positions is deprecated. In a future version, int
eger keys will always be treated as labels (consistent with DataFrame behavior). To
access a value by position, use `ser.iloc[pos]`
  print(f"Testing data shape: {X_test.shape}, Data type: {X_test.dtypes[0]}")
```

In [ ]:
```
from sklearn.svm import SVC
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import GridSearchCV

# Define hyperparameter grids
param_grid_svc = {
    'kernel': ['linear', 'rbf'],
    'C': [0.1, 1, 10],
    'gamma': ['scale', 'auto']
}

param_grid_logreg = {
    'penalty': ['l1', 'l2'],
    'C': [0.01, 0.1, 1, 10],
    'solver': ['liblinear']
}

param_grid_dt = {
    'max_depth': [5, 10, 20],
    'min_samples_leaf': [1, 5, 10]
}

# Initialize models
svc = SVC()
logreg = LogisticRegression()
dt = DecisionTreeClassifier()

# Initialize GridSearchCV
grid_svc = GridSearchCV(svc, param_grid_svc, cv=5, scoring='f1', n_jobs=-1)
grid_logreg = GridSearchCV(logreg, param_grid_logreg, cv=5, scoring='f1', n_jobs=-1
grid_dt = GridSearchCV(dt, param_grid_dt, cv=5, scoring='f1', n_jobs=-1)
```

In [ ]:
```
# Fit the models
print("Tuning SVC...")
grid_svc.fit(X_train, y_train)

print("Tuning Logistic Regression...")
grid_logreg.fit(X_train, y_train)

print("Tuning Decision Tree...")
grid_dt.fit(X_train, y_train)

print("Best parameters for SVC:", grid_svc.best_params_)
```

```
print("Best parameters for Logistic Regression:", grid_logreg.best_params_)
print("Best parameters for Decision Tree:", grid_dt.best_params_)
```

```
Tuning SVC...
Tuning Logistic Regression...
Tuning Decision Tree...
Best parameters for SVC: {'C': 0.1, 'gamma': 'scale', 'kernel': 'linear'}
Best parameters for Logistic Regression: {'C': 0.01, 'penalty': 'l1', 'solver': 'lib
linear'}
Best parameters for Decision Tree: {'max_depth': 20, 'min_samples_leaf': 5}
```

In [12]:
```
# Train models with the best parameters
svc_best = grid_svc.best_estimator_
logreg_best = grid_logreg.best_estimator_
dt_best = grid_dt.best_estimator_

svc_best.fit(X_train, y_train)
logreg_best.fit(X_train, y_train)
dt_best.fit(X_train, y_train)
```

```
c:\Users\preet\AppData\Local\Programs\Python\Python312\Lib\site-packages\sklearn\svm
\_base.py:1235: ConvergenceWarning: Liblinear failed to converge, increase the numbe
r of iterations.
  warnings.warn(
```

Out[12]:

DecisionTreeClassifier

```
DecisionTreeClassifier(max_depth=20, min_samples_leaf=5)
```

In [ ]:
```
from sklearn.metrics import classification_report, roc_auc_score, average_precision

# evaluate and print metrics
def evaluate_model(model, X_train, y_train, X_test, y_test):
    y_train_pred = model.predict(X_train)
    y_test_pred = model.predict(X_test)

    print("Training Metrics:")
    print(classification_report(y_train, y_train_pred))
    print("Train AUPRC:", average_precision_score(y_train, y_train_pred))
    print("Train AUROC:", roc_auc_score(y_train, y_train_pred))

    print("\nTesting Metrics:")
    print(classification_report(y_test, y_test_pred))
    print("Test AUPRC:", average_precision_score(y_test, y_test_pred))
    print("Test AUROC:", roc_auc_score(y_test, y_test_pred))

# results on all three models
print("Evaluating SVC:")
evaluate_model(svc_best, X_train, y_train, X_test, y_test)

print("\nEvaluating Logistic Regression:")
evaluate_model(logreg_best, X_train, y_train, X_test, y_test)

print("\nEvaluating Decision Tree:")
evaluate_model(dt_best, X_train, y_train, X_test, y_test)
```

```
Evaluating SVC:
Training Metrics:
              precision    recall  f1-score   support

         0.0       1.00      1.00      1.00      2366
         1.0       1.00      1.00      1.00        34

    accuracy                           1.00      2400
   macro avg       1.00      1.00      1.00      2400
weighted avg       1.00      1.00      1.00      2400

Train AUPRC: 1.0
Train AUROC: 1.0

Testing Metrics:
              precision    recall  f1-score   support

         0.0       0.99      0.98      0.99       592
         1.0       0.29      0.50      0.36         8

    accuracy                           0.98       600
   macro avg       0.64      0.74      0.68       600
weighted avg       0.98      0.98      0.98       600

Test AUPRC: 0.1495238095238095
Test AUROC: 0.7415540540540542

Evaluating Logistic Regression:
Training Metrics:
              precision    recall  f1-score   support

         0.0       1.00      1.00      1.00      2366
         1.0       1.00      1.00      1.00        34

    accuracy                           1.00      2400
   macro avg       1.00      1.00      1.00      2400
weighted avg       1.00      1.00      1.00      2400

Train AUPRC: 1.0
Train AUROC: 1.0

Testing Metrics:
              precision    recall  f1-score   support

         0.0       0.99      0.98      0.99       592
         1.0       0.36      0.62      0.45         8

    accuracy                           0.98       600
   macro avg       0.68      0.80      0.72       600
weighted avg       0.99      0.98      0.98       600

Test AUPRC: 0.22821428571428573
Test AUROC: 0.8048986486486487

Evaluating Decision Tree:
Training Metrics:
```

```
              precision    recall  f1-score   support

         0.0       1.00      1.00      1.00      2366
         1.0       0.85      0.85      0.85        34

    accuracy                           1.00      2400
   macro avg       0.93      0.93      0.93      2400
weighted avg       1.00      1.00      1.00      2400

Train AUPRC: 0.7295919838523645
Train AUROC: 0.9254139525632739

Testing Metrics:
              precision    recall  f1-score   support

         0.0       0.99      0.99      0.99       592
         1.0       0.40      0.50      0.44         8

    accuracy                           0.98       600
   macro avg       0.70      0.74      0.72       600
weighted avg       0.99      0.98      0.98       600

Test AUPRC: 0.20666666666666667
Test AUROC: 0.7449324324324325
```

Results:

Overfitting on Training Data: All models (SVC, Logistic Regression, Decision Tree) show perfect performance on the training set (precision, recall, F1-score ~1.0), indicating overfitting.

Poor Performance on Positive Class: Testing metrics for the positive class (1) are very low (precision, recall, F1-score < 0.5), suggesting difficulty in identifying the minority class.

Class Imbalance Impact: The severe class imbalance (1 positive for every 59 negatives) causes the models to bias predictions toward the negative class, leading to misleading accuracy (~98%) but poor performance on the positive class.

SVC and Decision Tree Struggles: SVC has the lowest AUPRC (0.15) and poor recall, while the Decision Tree has better precision but similar recall issues.

Macro Average F1-Score: The macro average F1-score is low, highlighting the models' poor performance on the minority class despite high overall accuracy.

Task 2

We the severe class imbalance (1:59 ratio) in the IDA2016 dataset using various techniques to improve classification performance, specifically targeting the macro-average F1-score. We apply data resampling (undersampling, oversampling with SMOTE, and SMOTEENN), use

class weight adjustments (class_weight='balanced'), and use sample weights inversely proportional to class frequency. Additionally, we use custom ensemble methods, such as Balanced Random Forest and AdaBoost with Decision Trees as base classifiers, to handle the imbalance. These strategies are tested on classifiers (SVC, Logistic Regression, Decision Tree), and the models are evaluated using macro F1-score on both training and test sets.

In [14]:
```
from imblearn.over_sampling import SMOTE
from imblearn.under_sampling import RandomUnderSampler
from imblearn.combine import SMOTEENN

# Initialize resampling techniques
undersampler = RandomUnderSampler(random_state=42)
smote = SMOTE(random_state=42)
smoteenn = SMOTEENN(random_state=42)

# Undersampling the majority class
X_train_undersampled, y_train_undersampled = undersampler.fit_resample(X_train, y_t
print("Undersampled training data shape:", X_train_undersampled.shape)

# Oversampling the minority class
X_train_oversampled, y_train_oversampled = smote.fit_resample(X_train, y_train)
print("Oversampled training data shape:", X_train_oversampled.shape)

# Combined SMOTE and ENN
X_train_smoteenn, y_train_smoteenn = smoteenn.fit_resample(X_train, y_train)
print("SMOTEENN training data shape:", X_train_smoteenn.shape)
```

```
Undersampled training data shape: (68, 170)
Oversampled training data shape: (4732, 170)
SMOTEENN training data shape: (4661, 170)
```

In [15]:
```
# Define the models with class weight adjustments
svc_weighted = SVC(class_weight='balanced')
logreg_weighted = LogisticRegression(class_weight='balanced', solver='liblinear')
dt_weighted = DecisionTreeClassifier(class_weight='balanced')

# Hyperparameter grids for GridSearchCV
param_grid_svc_weighted = {'kernel': ['linear', 'rbf'], 'C': [0.1, 1, 10], 'gamma':
param_grid_logreg_weighted = {'penalty': ['l1', 'l2'], 'C': [0.01, 0.1, 1, 10]}
param_grid_dt_weighted = {'max_depth': [5, 10, 20], 'min_samples_leaf': [1, 5, 10]}

# GridSearchCV with class-weighted models
grid_svc_weighted = GridSearchCV(svc_weighted, param_grid_svc_weighted, cv=5, scori
grid_logreg_weighted = GridSearchCV(logreg_weighted, param_grid_logreg_weighted, cv
grid_dt_weighted = GridSearchCV(dt_weighted, param_grid_dt_weighted, cv=5, scoring=
```

In [16]:
```
# Compute sample weights
class_counts = y_train.value_counts()
sample_weights = y_train.map({0: 1/class_counts[0], 1: 1/class_counts[1]})

# Fit models with sample weights
svc_sample_weighted = SVC()
logreg_sample_weighted = LogisticRegression(solver='liblinear')
dt_sample_weighted = DecisionTreeClassifier()
```

```
# Training with sample weights
svc_sample_weighted.fit(X_train, y_train, sample_weight=sample_weights)
logreg_sample_weighted.fit(X_train, y_train, sample_weight=sample_weights)
dt_sample_weighted.fit(X_train, y_train, sample_weight=sample_weights)
```

```
c:\Users\preet\AppData\Local\Programs\Python\Python312\Lib\site-packages\sklearn\svm
\_base.py:1235: ConvergenceWarning: Liblinear failed to converge, increase the numbe
r of iterations.
  warnings.warn(
```

Out[16]:

DecisionTreeClassifier

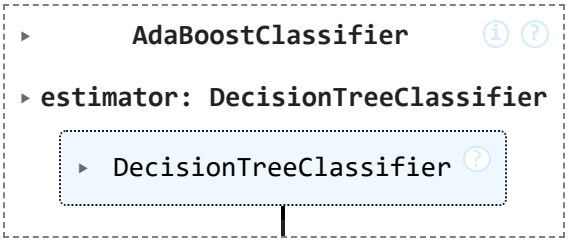
```
DecisionTreeClassifier()
```

In [18]:

```
from imblearn.ensemble import BalancedRandomForestClassifier
from sklearn.ensemble import AdaBoostClassifier

# Balanced Random Forest
brf = BalancedRandomForestClassifier(n_estimators=100, random_state=42)

# AdaBoost with Decision Tree base estimator
adaboost = AdaBoostClassifier(estimator=DecisionTreeClassifier(max_depth=1), n_esti

# Fit models
brf.fit(X_train, y_train)
adaboost.fit(X_train, y_train)
```

```
c:\Users\preet\AppData\Local\Programs\Python\Python312\Lib\site-packages\imblearn\en
semble\_forest.py:577: FutureWarning: The default of `sampling_strategy` will change
from ``'auto'`` to ``'all'`` in version 0.13. This change will follow the implementation
proposed in the original paper. Set to ``'all'`` to silence this warning and adopt the
future behaviour.
  warn(
c:\Users\preet\AppData\Local\Programs\Python\Python312\Lib\site-packages\imblearn\en
semble\_forest.py:589: FutureWarning: The default of `replacement` will change from
`False` to `True` in version 0.13. This change will follow the implementation propos
ed in the original paper. Set to `True` to silence this warning and adopt the future
behaviour.
  warn(
c:\Users\preet\AppData\Local\Programs\Python\Python312\Lib\site-packages\imblearn\en
semble\_forest.py:601: FutureWarning: The default of `bootstrap` will change from `T
rue` to `False` in version 0.13. This change will follow the implementation proposed
in the original paper. Set to `False` to silence this warning and adopt the future b
ehaviour.
  warn(
c:\Users\preet\AppData\Local\Programs\Python\Python312\Lib\site-packages\sklearn\ens
emble\_weight_boosting.py:527: FutureWarning: The SAMME.R algorithm (the default) is
deprecated and will be removed in 1.6. Use the SAMME algorithm to circumvent this wa
rning.
  warnings.warn(
```

Out[18]:



In [19]:

```python
from sklearn.metrics import f1_score

# Evaluation function
def evaluate_macro_f1(model, X_train, y_train, X_test, y_test):
    y_train_pred = model.predict(X_train)
    y_test_pred = model.predict(X_test)

    train_f1_macro = f1_score(y_train, y_train_pred, average='macro')
    test_f1_macro = f1_score(y_test, y_test_pred, average='macro')

    print(f"Macro F1-Score (Train): {train_f1_macro:.4f}")
    print(f"Macro F1-Score (Test): {test_f1_macro:.4f}")

# Evaluate all classifiers
print("\nEvaluating Weighted SVC:")
evaluate_macro_f1(svc_sample_weighted, X_train, y_train, X_test, y_test)

print("\nEvaluating Weighted Logistic Regression:")
evaluate_macro_f1(logreg_sample_weighted, X_train, y_train, X_test, y_test)

print("\nEvaluating Weighted Decision Tree:")
evaluate_macro_f1(dt_sample_weighted, X_train, y_train, X_test, y_test)

print("\nEvaluating Balanced Random Forest:")
evaluate_macro_f1(brf, X_train, y_train, X_test, y_test)

print("\nEvaluating AdaBoost:")
evaluate_macro_f1(adaboost, X_train, y_train, X_test, y_test)
```

```
Evaluating Weighted SVC:
Macro F1-Score (Train): 0.4964
Macro F1-Score (Test): 0.4966

Evaluating Weighted Logistic Regression:
Macro F1-Score (Train): 1.0000
Macro F1-Score (Test): 0.6850

Evaluating Weighted Decision Tree:
Macro F1-Score (Train): 1.0000
Macro F1-Score (Test): 0.7109

Evaluating Balanced Random Forest:
Macro F1-Score (Train): 0.6616
Macro F1-Score (Test): 0.6151

Evaluating AdaBoost:
Macro F1-Score (Train): 1.0000
Macro F1-Score (Test): 0.7637
```

Results

1. Both Logistic Regression and Decision Tree show perfect training performance (Macro F1 = 1.0) but significantly lower test performance (Logistic Regression: 0.6850, Decision Tree: 0.7109), indicating overfitting. AdaBoost also shows perfect training performance but outperforms both on the test set (0.7637), suggesting better generalization.

2. The Weighted SVC maintains consistent performance on both the training (0.4964) and test (0.4966) sets, indicating it doesn't overfit but struggles to achieve high performance on this imbalanced dataset.

3. The Balanced Random Forest has moderate training (0.6616) and test (0.6151) performance, showing that it handles class imbalance better than SVC but still faces challenges in generalizing to the test set.

4. AdaBoost's test performance (0.7637) is the highest among all models, demonstrating its ability to generalize well despite overfitting in the training phase.

5. Both models overfit with a perfect training F1 score but perform significantly worse on the test set, indicating that regularization or model adjustments are necessary.

6. AdaBoost is the best performing model, followed by Balanced Random Forest. Logistic Regression and Decision Tree need adjustments to prevent overfitting, and further tuning may help improve SVC’s performance.

Conclusion

In Task1, we built baseline classifiers and tuned their hyperparameters, with varying performance across models. In Task 2, we addressed class imbalance using resampling, class weights, and sample weights, along with ensemble methods like AdaBoost and Balanced Random Forest. AdaBoost showed the best performance, effectively handling class imbalance, while Logistic Regression and Decision Tree overfitted, and SVC performed moderately.