da24c026

October 13, 2024

DA24C026 - Assignment 7

TASK 1

[391]:
```
import numpy as np
import pandas as pd
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 f1_score, make_scorer
from sklearn.model_selection import StratifiedKFold, cross_val_score
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA
from sklearn.impute import SimpleImputer
from imblearn.over_sampling import SMOTE
from imblearn.over_sampling import RandomOverSampler
from imblearn.pipeline import Pipeline as ImbPipeline
import matplotlib.pyplot as plt
```

[305]:
```
df = pd.read_csv("aps_failure_training_set.csv", skiprows = 20)
```

[306]:
```
df.head()
```

[306]:
```
  class  aa_000 ab_000      ac_000 ad_000 ae_000 af_000 ag_000 ag_001 ag_002  \
0   neg   76698     na  2130706438    280      0      0      0      0      0
1   neg   33058     na           0     na      0      0      0      0      0
2   neg   41040     na         228    100      0      0      0      0      0
3   neg      12      0          70     66      0     10      0      0      0
4   neg   60874     na        1368    458      0      0      0      0      0

   …   ee_002  ee_003  ee_004  ee_005  ee_006  ee_007  ee_008 ee_009 ef_000  \
0  …  1240520  493384  721044  469792  339156  157956   73224      0      0
1  …   421400  178064  293306  245416  133654   81140   97576   1500      0
2  …   277378  159812  423992  409564  320746  158022   95128    514      0
3  …      240      46      58      44      10       0       0      0      4
4  …   622012  229790  405298  347188  286954  311560  433954   1218      0

  eg_000
```

```
0      0
1      0
2      0
3     32
4      0

[5 rows x 171 columns]
```

[307]:
```
df['class'].describe()
```

[307]:
```
count     60000
unique        2
top         neg
freq      59000
Name: class, dtype: object
```

Data Preprocessing

[308]:
```
df.replace('na', np.nan, inplace=True)
```

[309]:
```
df.isna().sum()
```

[309]:
```
class         0
aa_000        0
ab_000    46329
ac_000     3335
ad_000    14861
          …
ee_007      671
ee_008      671
ee_009      671
ef_000     2724
eg_000     2723
Length: 171, dtype: int64
```

Dropping Columns which have more than 50% missing values as they don't add enough new info for training

[310]:
```
df = df.dropna(axis=1, thresh=0.5*len(df))
```

[311]:
```
df.shape
```

[311]:
```
(60000, 163)
```

[312]:
```
df.isna().sum()
```

[312]:
```
class         0
aa_000        0
ac_000     3335
ad_000    14861
ae_000     2500
          …
ee_007      671
ee_008      671
ee_009      671
ef_000     2724
eg_000     2723
Length: 163, dtype: int64
```

[313]:
```
X = df.iloc[:,1:]
y = df.iloc[:,0]
```

Handling missing values of remaining coloumns

[314]:
```
imputer = SimpleImputer(strategy='median')
X = pd.DataFrame(imputer.fit_transform(X), columns=X.columns)
```

[315]:
```
X.isna().sum()
```

[315]:
```
aa_000    0
ac_000    0
ad_000    0
ae_000    0
af_000    0
         ..
ee_007    0
ee_008    0
ee_009    0
ef_000    0
eg_000    0
Length: 162, dtype: int64
```

Scaling dow data for faster convergence

[316]:
```
std_scaler = StandardScaler()
x_scaled = std_scaler.fit_transform(X)
```

Data dimensionality reduction

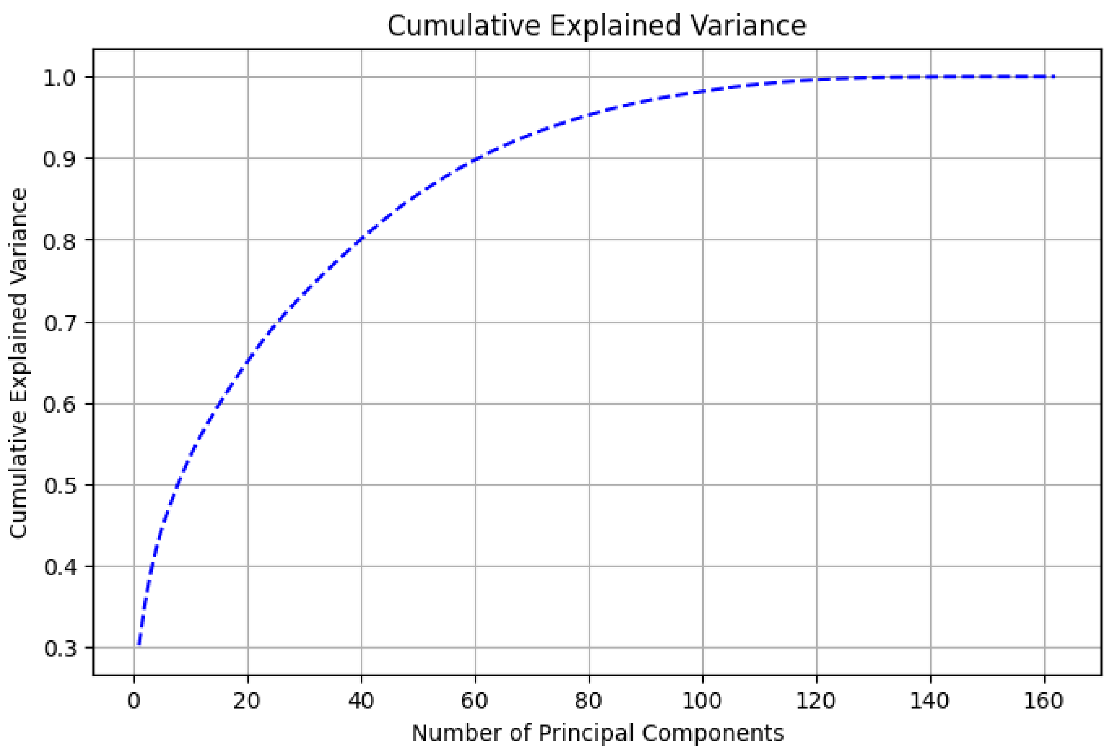
[317]:
```
pca = PCA()
X_pca = pca.fit_transform(x_scaled)

explained_variance_ratio = pca.explained_variance_ratio_
```

```
cumulative_variance = np.cumsum(explained_variance_ratio)

plt.figure(figsize=(8, 5))
plt.plot(np.arange(1, len(cumulative_variance)+1), cumulative_variance,␣
  ↪linestyle='--', color='b')
plt.title('Cumulative Explained Variance')
plt.xlabel('Number of Principal Components')
plt.ylabel('Cumulative Explained Variance')
plt.grid(True)
plt.show()

for i, cumulative_var in enumerate(cumulative_variance):
    print(f'Number of Components: {i+1}, Cumulative Explained Variance:␣
  ↪{cumulative_var:.4f}')
```




```
Number of Components: 1, Cumulative Explained Variance: 0.3017
Number of Components: 2, Cumulative Explained Variance: 0.3512
Number of Components: 3, Cumulative Explained Variance: 0.3894
Number of Components: 4, Cumulative Explained Variance: 0.4199
Number of Components: 5, Cumulative Explained Variance: 0.4449
Number of Components: 6, Cumulative Explained Variance: 0.4653
Number of Components: 7, Cumulative Explained Variance: 0.4840
Number of Components: 8, Cumulative Explained Variance: 0.5024
```

```
Number of Components: 9, Cumulative Explained Variance: 0.5188
Number of Components: 10, Cumulative Explained Variance: 0.5338
Number of Components: 11, Cumulative Explained Variance: 0.5483
Number of Components: 12, Cumulative Explained Variance: 0.5612
Number of Components: 13, Cumulative Explained Variance: 0.5737
Number of Components: 14, Cumulative Explained Variance: 0.5854
Number of Components: 15, Cumulative Explained Variance: 0.5968
Number of Components: 16, Cumulative Explained Variance: 0.6079
Number of Components: 17, Cumulative Explained Variance: 0.6186
Number of Components: 18, Cumulative Explained Variance: 0.6291
Number of Components: 19, Cumulative Explained Variance: 0.6394
Number of Components: 20, Cumulative Explained Variance: 0.6495
Number of Components: 21, Cumulative Explained Variance: 0.6594
Number of Components: 22, Cumulative Explained Variance: 0.6689
Number of Components: 23, Cumulative Explained Variance: 0.6783
Number of Components: 24, Cumulative Explained Variance: 0.6872
Number of Components: 25, Cumulative Explained Variance: 0.6955
Number of Components: 26, Cumulative Explained Variance: 0.7036
Number of Components: 27, Cumulative Explained Variance: 0.7116
Number of Components: 28, Cumulative Explained Variance: 0.7192
Number of Components: 29, Cumulative Explained Variance: 0.7266
Number of Components: 30, Cumulative Explained Variance: 0.7338
Number of Components: 31, Cumulative Explained Variance: 0.7410
Number of Components: 32, Cumulative Explained Variance: 0.7480
Number of Components: 33, Cumulative Explained Variance: 0.7549
Number of Components: 34, Cumulative Explained Variance: 0.7617
Number of Components: 35, Cumulative Explained Variance: 0.7684
Number of Components: 36, Cumulative Explained Variance: 0.7751
Number of Components: 37, Cumulative Explained Variance: 0.7815
Number of Components: 38, Cumulative Explained Variance: 0.7878
Number of Components: 39, Cumulative Explained Variance: 0.7940
Number of Components: 40, Cumulative Explained Variance: 0.8002
Number of Components: 41, Cumulative Explained Variance: 0.8064
Number of Components: 42, Cumulative Explained Variance: 0.8123
Number of Components: 43, Cumulative Explained Variance: 0.8181
Number of Components: 44, Cumulative Explained Variance: 0.8240
Number of Components: 45, Cumulative Explained Variance: 0.8297
Number of Components: 46, Cumulative Explained Variance: 0.8352
Number of Components: 47, Cumulative Explained Variance: 0.8406
Number of Components: 48, Cumulative Explained Variance: 0.8457
Number of Components: 49, Cumulative Explained Variance: 0.8506
Number of Components: 50, Cumulative Explained Variance: 0.8554
Number of Components: 51, Cumulative Explained Variance: 0.8601
Number of Components: 52, Cumulative Explained Variance: 0.8646
Number of Components: 53, Cumulative Explained Variance: 0.8691
Number of Components: 54, Cumulative Explained Variance: 0.8734
Number of Components: 55, Cumulative Explained Variance: 0.8777
Number of Components: 56, Cumulative Explained Variance: 0.8820
```

```
Number of Components: 57, Cumulative Explained Variance: 0.8861
Number of Components: 58, Cumulative Explained Variance: 0.8900
Number of Components: 59, Cumulative Explained Variance: 0.8939
Number of Components: 60, Cumulative Explained Variance: 0.8976
Number of Components: 61, Cumulative Explained Variance: 0.9013
Number of Components: 62, Cumulative Explained Variance: 0.9048
Number of Components: 63, Cumulative Explained Variance: 0.9083
Number of Components: 64, Cumulative Explained Variance: 0.9116
Number of Components: 65, Cumulative Explained Variance: 0.9149
Number of Components: 66, Cumulative Explained Variance: 0.9179
Number of Components: 67, Cumulative Explained Variance: 0.9209
Number of Components: 68, Cumulative Explained Variance: 0.9238
Number of Components: 69, Cumulative Explained Variance: 0.9266
Number of Components: 70, Cumulative Explained Variance: 0.9293
Number of Components: 71, Cumulative Explained Variance: 0.9319
Number of Components: 72, Cumulative Explained Variance: 0.9345
Number of Components: 73, Cumulative Explained Variance: 0.9370
Number of Components: 74, Cumulative Explained Variance: 0.9394
Number of Components: 75, Cumulative Explained Variance: 0.9417
Number of Components: 76, Cumulative Explained Variance: 0.9440
Number of Components: 77, Cumulative Explained Variance: 0.9462
Number of Components: 78, Cumulative Explained Variance: 0.9484
Number of Components: 79, Cumulative Explained Variance: 0.9505
Number of Components: 80, Cumulative Explained Variance: 0.9527
Number of Components: 81, Cumulative Explained Variance: 0.9547
Number of Components: 82, Cumulative Explained Variance: 0.9566
Number of Components: 83, Cumulative Explained Variance: 0.9585
Number of Components: 84, Cumulative Explained Variance: 0.9603
Number of Components: 85, Cumulative Explained Variance: 0.9621
Number of Components: 86, Cumulative Explained Variance: 0.9638
Number of Components: 87, Cumulative Explained Variance: 0.9654
Number of Components: 88, Cumulative Explained Variance: 0.9670
Number of Components: 89, Cumulative Explained Variance: 0.9685
Number of Components: 90, Cumulative Explained Variance: 0.9699
Number of Components: 91, Cumulative Explained Variance: 0.9714
Number of Components: 92, Cumulative Explained Variance: 0.9727
Number of Components: 93, Cumulative Explained Variance: 0.9739
Number of Components: 94, Cumulative Explained Variance: 0.9751
Number of Components: 95, Cumulative Explained Variance: 0.9763
Number of Components: 96, Cumulative Explained Variance: 0.9774
Number of Components: 97, Cumulative Explained Variance: 0.9786
Number of Components: 98, Cumulative Explained Variance: 0.9796
Number of Components: 99, Cumulative Explained Variance: 0.9807
Number of Components: 100, Cumulative Explained Variance: 0.9818
Number of Components: 101, Cumulative Explained Variance: 0.9828
Number of Components: 102, Cumulative Explained Variance: 0.9838
Number of Components: 103, Cumulative Explained Variance: 0.9848
Number of Components: 104, Cumulative Explained Variance: 0.9858
```

```
Number of Components: 105, Cumulative Explained Variance: 0.9867
Number of Components: 106, Cumulative Explained Variance: 0.9876
Number of Components: 107, Cumulative Explained Variance: 0.9884
Number of Components: 108, Cumulative Explained Variance: 0.9892
Number of Components: 109, Cumulative Explained Variance: 0.9899
Number of Components: 110, Cumulative Explained Variance: 0.9906
Number of Components: 111, Cumulative Explained Variance: 0.9913
Number of Components: 112, Cumulative Explained Variance: 0.9919
Number of Components: 113, Cumulative Explained Variance: 0.9925
Number of Components: 114, Cumulative Explained Variance: 0.9931
Number of Components: 115, Cumulative Explained Variance: 0.9937
Number of Components: 116, Cumulative Explained Variance: 0.9943
Number of Components: 117, Cumulative Explained Variance: 0.9948
Number of Components: 118, Cumulative Explained Variance: 0.9952
Number of Components: 119, Cumulative Explained Variance: 0.9956
Number of Components: 120, Cumulative Explained Variance: 0.9960
Number of Components: 121, Cumulative Explained Variance: 0.9964
Number of Components: 122, Cumulative Explained Variance: 0.9968
Number of Components: 123, Cumulative Explained Variance: 0.9971
Number of Components: 124, Cumulative Explained Variance: 0.9973
Number of Components: 125, Cumulative Explained Variance: 0.9976
Number of Components: 126, Cumulative Explained Variance: 0.9978
Number of Components: 127, Cumulative Explained Variance: 0.9980
Number of Components: 128, Cumulative Explained Variance: 0.9982
Number of Components: 129, Cumulative Explained Variance: 0.9984
Number of Components: 130, Cumulative Explained Variance: 0.9986
Number of Components: 131, Cumulative Explained Variance: 0.9987
Number of Components: 132, Cumulative Explained Variance: 0.9989
Number of Components: 133, Cumulative Explained Variance: 0.9990
Number of Components: 134, Cumulative Explained Variance: 0.9991
Number of Components: 135, Cumulative Explained Variance: 0.9992
Number of Components: 136, Cumulative Explained Variance: 0.9993
Number of Components: 137, Cumulative Explained Variance: 0.9994
Number of Components: 138, Cumulative Explained Variance: 0.9995
Number of Components: 139, Cumulative Explained Variance: 0.9996
Number of Components: 140, Cumulative Explained Variance: 0.9997
Number of Components: 141, Cumulative Explained Variance: 0.9997
Number of Components: 142, Cumulative Explained Variance: 0.9998
Number of Components: 143, Cumulative Explained Variance: 0.9999
Number of Components: 144, Cumulative Explained Variance: 0.9999
Number of Components: 145, Cumulative Explained Variance: 0.9999
Number of Components: 146, Cumulative Explained Variance: 0.9999
Number of Components: 147, Cumulative Explained Variance: 1.0000
Number of Components: 148, Cumulative Explained Variance: 1.0000
Number of Components: 149, Cumulative Explained Variance: 1.0000
Number of Components: 150, Cumulative Explained Variance: 1.0000
Number of Components: 151, Cumulative Explained Variance: 1.0000
Number of Components: 152, Cumulative Explained Variance: 1.0000
```

```
Number of Components: 153, Cumulative Explained Variance: 1.0000
Number of Components: 154, Cumulative Explained Variance: 1.0000
Number of Components: 155, Cumulative Explained Variance: 1.0000
Number of Components: 156, Cumulative Explained Variance: 1.0000
Number of Components: 157, Cumulative Explained Variance: 1.0000
Number of Components: 158, Cumulative Explained Variance: 1.0000
Number of Components: 159, Cumulative Explained Variance: 1.0000
Number of Components: 160, Cumulative Explained Variance: 1.0000
Number of Components: 161, Cumulative Explained Variance: 1.0000
Number of Components: 162, Cumulative Explained Variance: 1.0000
```

As it can be seen in the graph, if we have 60 components then more than 90% variance is captured

[318]:
```
pca = PCA(n_components=60)
X_pca = pca.fit_transform(x_scaled)
```

Splitting Data and training models

[319]:
```
X_train, X_test, y_train, y_test = train_test_split(X_pca, y, test_size = 0.25,␣
↪random_state = 42)
```

1.) SVC

[320]:
```
svc = SVC()
svc_param_grid = {
    'kernel': ['rbf', 'sigmoid'],
    'C': [0.1, 1, 10],
    'gamma': ['scale', 'auto']
}
```

[321]:
```
svc_grid = GridSearchCV(svc, svc_param_grid, cv=5, scoring = 'f1_macro')
svc_grid.fit(X_train, y_train)
```

[321]:
```
GridSearchCV(cv=5, estimator=SVC(),
             param_grid={'C': [0.1, 1, 10], 'gamma': ['scale', 'auto'],
                         'kernel': ['rbf', 'sigmoid']},
             scoring='f1_macro')
```

[322]:
```
svc_best = svc_grid.best_estimator_
print ("Best parameters for SVC: ", svc_grid.best_params_)
```

```
Best parameters for SVC:  {'C': 10, 'gamma': 'scale', 'kernel': 'rbf'}
```

[323]:
```
y_train_pred_svc = svc_best.predict(X_train)
y_test_pred_svc = svc_best.predict(X_test)

f1_macro_train_svc = f1_score(y_train, y_train_pred_svc, average='macro')
f1_macro_test_svc = f1_score(y_test, y_test_pred_svc, average='macro')
```

```
print("SVC Performance on Train Set:\n", f1_macro_train_svc)
print("SVC Performance on Test Set:\n", f1_macro_test_svc)
```

```
SVC Performance on Train Set:
 0.9790807359965954
SVC Performance on Test Set:
 0.7851316110119919
```

2.) Logistic Regression

[324]:
```
log_reg = LogisticRegression(solver='liblinear')
log_reg_param_grid = {
    'penalty': ['l1', 'l2'],
    'C': [0.1, 1, 10],
}
```

[325]:
```
log_reg = GridSearchCV(log_reg, log_reg_param_grid, cv=5, scoring = "f1_macro")
log_reg.fit(X_train, y_train)
```

[325]:
```
GridSearchCV(cv=5, estimator=LogisticRegression(solver='liblinear'),
             param_grid={'C': [0.1, 1, 10], 'penalty': ['l1', 'l2']},
             scoring='f1_macro')
```

[326]:
```
log_reg_best = log_reg.best_estimator_
print ("Best parameters for Logistic Regression: ", log_reg.best_params_)
```

```
Best parameters for Logistic Regression:  {'C': 10, 'penalty': 'l1'}
```

[327]:
```
y_train_pred_log_reg = log_reg_best.predict(X_train)
y_test_pred_log_reg = log_reg_best.predict(X_test)

f1_macro_train_log_reg = f1_score(y_train, y_train_pred_log_reg,␣
 ↪average='macro')
f1_macro_test_log_reg = f1_score(y_test, y_test_pred_log_reg, average='macro')

print("Logistic Regression Performance on Train Set:\n", f1_macro_train_log_reg)
print("Logistic Regression Performance on Test Set:\n", f1_macro_test_log_reg)
```

```
Logistic Regression Performance on Train Set:
 0.8243212359917116
Logistic Regression Performance on Test Set:
 0.835307388606881
```

3.) Decision Tree

[328]:
```
dt = DecisionTreeClassifier()
dt_param_grid = {
    'max_depth': [10, 15, 20],
    'min_samples_leaf': [1, 2, 4]
```

```
}
```

[329]:
```
dt_grid = GridSearchCV(dt, dt_param_grid, cv=5, scoring = "f1_macro")
dt_grid.fit(X_train, y_train)
```

[329]:
```
GridSearchCV(cv=5, estimator=DecisionTreeClassifier(),
             param_grid={'max_depth': [10, 15, 20],
                         'min_samples_leaf': [1, 2, 4]},
             scoring='f1_macro')
```

[330]:
```
dt_best = dt_grid.best_estimator_
print ("Best parameters for Decision Tree: ", dt_grid.best_params_)
```

```
Best parameters for Decision Tree:  {'max_depth': 10, 'min_samples_leaf': 1}
```

[331]:
```
y_train_pred_dt = dt_best.predict(X_train)
y_test_pred_dt = dt_best.predict(X_test)

f1_macro_train_dt = f1_score(y_train, y_train_pred_dt, average='macro')
f1_macro_test_dt = f1_score(y_test, y_test_pred_dt, average='macro')

print("Decision Tree Performance on Train Set:\n", f1_macro_train_dt)
print("Decision Tree Performance on Test Set:\n", f1_macro_test_dt)
```

```
Decision Tree Performance on Train Set:
 0.9574379762054479
Decision Tree Performance on Test Set:
 0.7885611433218309
```

Baseline Macro F1 Scores : SVC - 0.78, Logistic Reg - 0.83, Decision Tree - 0.79

TASK 2

Subtask 1 - Oversampling Minority Class

Using Smote for oversampling - Smote is used to oversample dataset based on no,of neighbors.

[332]:
```
sm = SMOTE(random_state=12, sampling_strategy = 0.25)
x_train_res, y_train_res = sm.fit_resample(X_train,y_train)
```

[333]:
```
y_train_res.describe()
```

[333]:
```
count     55317
unique        2
top         neg
freq      44254
Name: class, dtype: object
```

Training models for different oversampling strategies

1.) SVC

[366]:
```
clf = SVC(C=10, kernel='rbf', gamma='scale').fit(x_train_res,y_train_res)
```

[367]:
```
y_train_pred_svc_os = clf.predict(X_train)
f1_macro_train_os_svc = f1_score(y_train, y_train_pred_svc_os, average='macro')
y_test_pred_svc_os = clf.predict(X_test)
f1_macro_test_os_svc = f1_score(y_test, y_test_pred_svc_os, average = "macro")
print(" Upon oversampling, F1 Score for SVC on training set :",␣
↪f1_macro_train_os_svc)
print(" Upon oversampling, F1 Score for SVC on test set :",␣
↪f1_macro_test_os_svc)
```

```
 Upon oversampling, F1 Score for SVC on training set : 0.9790807359965954
 Upon oversampling, F1 Score for SVC on test set : 0.7942521597522247
```

2.) Logistic Regression

[368]:
```
sm = SMOTE(random_state=12, sampling_strategy = 0.06)
x_train_res, y_train_res = sm.fit_resample(X_train,y_train)
```

[369]:
```
clf = LogisticRegression(solver='liblinear', C = 10, penalty='l1', max_iter =␣
↪500).fit(x_train_res,y_train_res)
```

[370]:
```
y_train_pred_logreg_os = clf.predict(X_train)
f1_macro_train_os_logreg = f1_score(y_train, y_train_pred_logreg_os,␣
↪average='macro')
y_test_pred_logreg_os = clf.predict(X_test)
f1_macro_test_os_logreg = f1_score(y_test, y_test_pred_logreg_os, average =␣
↪"macro")
print(" Upon oversampling, F1 Score for Logistic Regression on training set :",␣
↪f1_macro_train_os_logreg)
print(" Upon oversampling, F1 Score for Logistic Regression on test set :",␣
↪f1_macro_test_os_logreg)
```

```
 Upon oversampling, F1 Score for Logistic Regression on training set :
0.8488898122715383
 Upon oversampling, F1 Score for Logistic Regression on test set :
0.8400754227724834
```

3.) Decision Tree

[371]:
```
sm = SMOTE(random_state=12, sampling_strategy = 0.02)
x_train_res, y_train_res = sm.fit_resample(X_train,y_train)
```

[372]:
```
clf = DecisionTreeClassifier()
```

[373]:
```
clf = DecisionTreeClassifier(max_depth=10, min_samples_leaf=1).fit(x_train_res,␣
↪y_train_res)
```

[374]:
```
y_train_pred_dtree_os = clf.predict(X_train)
f1_macro_train_os_dtree = f1_score(y_train, y_train_pred_dtree_os,␣
↪average='macro')
y_test_pred_dtree_os = clf.predict(X_test)
f1_macro_test_os_dtree = f1_score(y_test, y_test_pred_dtree_os, average =␣
↪"macro")
print(" Upon oversampling, F1 Score for Decision Tree on training set :",␣
↪f1_macro_train_os_dtree)
print(" Upon oversampling, F1 Score for Decision Tree on test set :",␣
↪f1_macro_test_os_dtree)
```

```
 Upon oversampling, F1 Score for Decision Tree on training set :
0.9626167990642506
 Upon oversampling, F1 Score for Decision Tree on test set : 0.8039748989858151
```

Subtask 2 - Assigning class weights

SVC

[375]:
```
clf = SVC(C=10, kernel='rbf', gamma='scale', class_weight= {'neg':1, 'pos':2}).
↪fit(X_train,y_train)
```

[376]:
```
y_train_pred_svc_cw = clf.predict(X_train)
f1_macro_train_cw_svc = f1_score(y_train, y_train_pred_svc_cw, average='macro')
y_test_pred_svc_cw = clf.predict(X_test)
f1_macro_test_cw_svc = f1_score(y_test, y_test_pred_svc_cw, average = "macro")
print(" Upon assigning Class weights, F1 Score for SVC on training set :",␣
↪f1_macro_train_cw_svc)
print(" Upon assigning Class weights, F1 Score for SVC on test set :",␣
↪f1_macro_test_cw_svc)
```

```
 Upon assigning Class weights, F1 Score for SVC on training set :
0.9856597439968171
 Upon assigning Class weights, F1 Score for SVC on test set : 0.7861070775130975
```

Logistic Regression

[377]:
```
clf = LogisticRegression(solver='liblinear', C = 10, penalty='l1', max_iter =␣
↪500, class_weight={'neg':4, 'pos':11}).fit(X_train,y_train)
```

[378]:
```
y_train_pred_logreg_cw = clf.predict(X_train)
f1_macro_train_cw_logreg = f1_score(y_train, y_train_pred_logreg_cw,␣
↪average='macro')
y_test_pred_logreg_cw = clf.predict(X_test)
f1_macro_test_cw_logreg = f1_score(y_test, y_test_pred_logreg_cw, average =␣
↪"macro")
print(" Upon assigning Class weights, F1 Score for Logistic Regresion on␣
↪training set :", f1_macro_train_cw_logreg)
```

```
print(" Upon assigning Class weights, F1 Score for Logistic Regresion on test␣
↪set :", f1_macro_test_cw_logreg)
```

```
 Upon assigning Class weights, F1 Score for Logistic Regresion on training set :
0.8457921283439009
 Upon assigning Class weights, F1 Score for Logistic Regresion on test set :
0.8446141718090743
```

Decision Tree

[379]:
```
clf = DecisionTreeClassifier(max_depth=10, min_samples_leaf=1,␣
↪class_weight={'neg':4, 'pos':11}).fit(X_train, y_train)
```

[410]:
```
y_train_pred_dtree_cw = clf.predict(X_train)
f1_macro_train_cw_dtree = f1_score(y_train, y_train_pred_dtree_cw,␣
↪average='macro')
y_test_pred_dtree_cw = clf.predict(X_test)
f1_macro_test_cw_dtree = f1_score(y_test, y_test_pred_dtree_cw, average =␣
↪"macro")
print(" Upon assigning Class weights, F1 Score for Decision on training set :",␣
↪f1_macro_train_cw_dtree)
print(" Upon assigning Class weights, F1 Score for Decision Regresion on test␣
↪set :", f1_macro_test_cw_dtree)
```

```
 Upon assigning Class weights, F1 Score for Decision on training set :
0.9590298235571053
 Upon assigning Class weights, F1 Score for Decision Regresion on test set :
0.816793893129771
```

Subtask - 3 Assigning sample weights instead of class weights, to penalize misclassifications

Main Difference between sample weights and class weights is Sample weights are assigned to each individual sample (data point) in the training dataset. Thus, it's parameter is specified while fitting the model. While, Class weights are assigned to each class (category) in a classification problem. Thus, it's parameter is specfied while making object of class.

SVC

[381]:
```
weights = {'neg': 0.475, 'pos': 1}
sw = np.array([weights[class_] for class_ in y_train])
```

[382]:
```
clf = SVC(C=10, kernel='rbf', gamma='scale').fit(X_train,y_train,␣
↪sample_weight= sw)
```

[383]:
```
y_train_pred_svc_sw = clf.predict(X_train)
f1_macro_train_sw_svc = f1_score(y_train, y_train_pred_svc_sw, average='macro')
y_test_pred_svc_sw = clf.predict(X_test)
f1_macro_test_sw_svc = f1_score(y_test, y_test_pred_svc_sw, average = "macro")
print(" Upon assigning Sample weights, F1 Score for SVC on training set :",␣
↪f1_macro_train_sw_svc)
```

```
print(" Upon assigning Sample weights, F1 Score for SVC on test set :",␣
↪f1_macro_test_sw_svc)
```

```
 Upon assigning Sample weights, F1 Score for SVC on training set :
0.9796171911702234
 Upon assigning Sample weights, F1 Score for SVC on test set :
0.7865366157429332
```

Logistic Regression

[384]:
```
weights = {'neg': 0.75, 'pos': 2.25}
sw = np.array([weights[class_] for class_ in y_train])
```

[385]:
```
clf = LogisticRegression(solver='liblinear', C = 10, penalty='l1', max_iter =␣
↪500).fit(X_train,y_train, sample_weight=sw)
```

[386]:
```
y_train_pred_logreg_sw = clf.predict(X_train)
f1_macro_train_sw_logreg = f1_score(y_train, y_train_pred_logreg_sw,␣
↪average='macro')
y_test_pred_logreg_sw = clf.predict(X_test)
f1_macro_test_sw_logreg = f1_score(y_test, y_test_pred_logreg_sw, average =␣
↪"macro")
print(" Upon assigning Sample weights, F1 Score for Logistic Regresion on␣
↪training set :", f1_macro_train_sw_logreg)
print(" Upon assigning Sample weights, F1 Score for Logistic Regresion on test␣
↪set :", f1_macro_test_sw_logreg)
```

```
 Upon assigning Sample weights, F1 Score for Logistic Regresion on training set
: 0.8431944395038975
 Upon assigning Sample weights, F1 Score for Logistic Regresion on test set :
0.8403600017355732
```

Decision Tree

[387]:
```
weights = {'neg': 0.75, 'pos': 2.15}
sw = np.array([weights[class_] for class_ in y_train])
```

[388]:
```
clf = DecisionTreeClassifier(max_depth=10, min_samples_leaf=1).fit(X_train,␣
↪y_train, sample_weight=sw)
```

[389]:
```
y_train_pred_dtree_sw = clf.predict(X_train)
f1_macro_train_sw_dtree = f1_score(y_train, y_train_pred_dtree_sw,␣
↪average='macro')
y_test_pred_dtree_sw = clf.predict(X_test)
f1_macro_test_sw_dtree = f1_score(y_test, y_test_pred_dtree_sw, average =␣
↪"macro")
print(" Upon assigning Sample weights, F1 Score for Decision Tree on training␣
↪set :", f1_macro_train_sw_dtree)
```

```
print(" Upon assigning Sample weights, F1 Score for Decision Tree on test set :
↪", f1_macro_test_sw_dtree)
```

```
 Upon assigning Sample weights, F1 Score for Decision Tree on training set :
0.9590298235571053
 Upon assigning Sample weights, F1 Score for Decision Tree on test set :
0.816793893129771
```

Subtask 4 - A different way to handle Imbalance

Stratified K-fold cross validation which takes enough samples from both classes

[443]:
```
skf = StratifiedKFold(n_splits=5, shuffle=True, random_state=42)

# Classifiers
svc = SVC()
logreg = LogisticRegression(solver='liblinear')
dt = DecisionTreeClassifier()

smote = SMOTE(sampling_strategy=0.06)

svc_pipeline = ImbPipeline([('smote', smote), ('svc', svc)])
logreg_pipeline = ImbPipeline([('smote', smote), ('logisticregression',␣
↪logreg)])
dt_pipeline = ImbPipeline([('smote', smote), ('decisiontreeclassifier', dt)])

svc_pipeline.fit(X_train, y_train)
logreg_pipeline.fit(X_train, y_train)
dt_pipeline.fit(X_train, y_train)

svc_test_pred = svc_pipeline.predict(X_test)
logreg_test_pred = logreg_pipeline.predict(X_test)
dt_test_pred = dt_pipeline.predict(X_test)

svc_kvc = f1_score(y_test, svc_test_pred, average='macro')
logreg_kcv = f1_score(y_test, logreg_test_pred, average='macro')
dtree_kcv = f1_score(y_test, dt_test_pred, average='macro')

print("SVC F1 Macro Score (Test):", svc_kvc)
print("Logistic Regression F1 Macro Score (Test):", logreg_kcv)
print("Decision Tree F1 Macro Score (Test):", dtree_kcv)
```

```
SVC F1 Macro Score (Test): 0.850736519594085
Logistic Regression F1 Macro Score (Test): 0.8394369441196272
Decision Tree F1 Macro Score (Test): 0.7771914047514632
```

[448]:
```
x, width = np.arange(3), 0.1
spacing = 0.01
```

```python
plt.bar(x - 2*(width + spacing), [f1_macro_test_svc, f1_macro_test_log_reg,␣
 ↪f1_macro_test_dt], width, label='Baseline')
plt.bar(x - (width + spacing), [f1_macro_test_os_svc, f1_macro_test_os_logreg,␣
 ↪f1_macro_test_os_dtree], width, label='Oversampling')
plt.bar(x, [f1_macro_test_cw_svc, f1_macro_test_cw_logreg,␣
 ↪f1_macro_test_cw_dtree], width, label='Class weights')
plt.bar(x + (width + spacing), [f1_macro_test_sw_svc, f1_macro_test_sw_logreg,␣
 ↪f1_macro_test_sw_dtree], width, label='Sample weights')
plt.bar(x + 2*(width + spacing), [svc_kvc, logreg_kcv, dtree_kcv], width,␣
 ↪label='Stratified K Fold CV')

plt.ylabel('Macro F1 Score')
plt.title('Baseline model vs Imbalance handling techniques (Testing set)')
plt.xticks(x,['SVC', 'Logistic Regression', 'Decision Tree'])
plt.ylim(0.75, 0.87)
plt.legend()
plt.tight_layout()
plt.show()
```