ECON 425 HW5

February 7, 2024

Problem 2 (SMOTE-NC)

The file card_transdata.csv contains data on 1,000,000 credit card transactions. The goal is to fit and test a model predicting whether a given transaction is fraudulent. Let the first 500,000 observations be the training set and the remaining observations be the test set.

Since the dataset contains both continuous and categorical features, while SMOTE can only handle continuous features, we use SMOTE-NC, a version of SMOTE that can handle both. In Python, use imblearn.over_sampling.SMOTENC. Do not forget to provide information about the categorical features to SMOTENC.

(i) For each $\gamma \in \{0.1, 0.2, \dots, 1\}$, oversample the training set with the desired ratio γ of the number of samples in the minority class over the number of samples in the majority class.

(ii) For each of the resulting oversampled datasets, fit a decision tree (with information gain as the feature selection criterion) and calculate its training and testing recall, precision, and F_1 score.

(iii) Draw a plot of training and testing F_1 score as functions of γ. Does using SMOTE-NC improve the decision tree's performance?

0.1 Problem 2

0.1.1 i) + ii)

[4]:
```
df=pd.read_csv("/Users/sandinatatu/Desktop/card_transdata-1-2.csv")
```

Below I check for categorical features. It seems that apart from the dependent variable "fraud", other categorical variables are "repeat retailer", "used_chip", "used_pin_number", and "online_order".

[5]:
```
df.nunique()
```

[5]:
```
distance_from_home                999971
distance_from_last_transaction    999836
ratio_to_median_purchase_price    999808
repeat_retailer                        2
used_chip                              2
```

```
used_pin_number                   2
online_order                      2
fraud                             2
dtype: int64
```

I store all of my independent variables in "X" and the dependant variable "fraud" in "y". I then proceed to create my training and test sets.

I also store the indexes of all of the categorical variables from X in a list I call "catind". I will be using this list when createing my SMOTENC models.

Finally, I create a list containing all of the values for my ratio in an array I call "la".

[6]:
```
X=df.drop("fraud", axis=1).copy()
y=df["fraud"]

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = .5)

catcols = [col for col in X.columns if X[col].nunique()==2]
catind = [X.columns.get_loc(col) for col in catcols]
la =np.arange(0.1, 1.01, 0.1)
```

In the loop below, I iterate through each of the ratios in my "la" array.

For each ratio, I create a SMOTE model that accounts for the categorical independent variables and for my ratio .

For each ratio, the respective SMOTE model is used to resample my training set. After the resampling is done, I fit a decision tree with "entropy" (or information gain) as my criterion on the resampled training set.

After computing this decision tree for each ratio, I calculate 3 metrics (preicion, recall, and F1 score) for both my training and test sets. I store all of these results in a list called "metrics".

[7]:
```
metrics=[]

for no in la:
    sm = SMOTENC(categorical_features = catind, sampling_strategy = no,␣
 ↪random_state=1)
    X_train_new, y_train_new = sm.fit_resample(X_train, y_train)

    tr = tree.DecisionTreeClassifier(criterion = "entropy")
    tr.fit(X_train_new, y_train_new)
    y_train_pred = tr.predict(X_train_new)
    y_test_pred = tr.predict(X_test)

    ptrain = precision_score(y_train_new, y_train_pred)
    rtrain = recall_score(y_train_new, y_train_pred)
    f1train = f1_score(y_train_new, y_train_pred)
```

```
    ptest = precision_score(y_test, y_test_pred)
    rtest = recall_score(y_test, y_test_pred)
    f1test = f1_score(y_test, y_test_pred)

    metrics.append({
        'r': no,
        'Training Precision': ptrain,
        'Training Recall': rtrain,
        'Training F1': f1train,
        'Test Precision': ptest,
        'Test Recall': rtest,
        'Test F1': f1test
    })
```

[8]:
```
for a in metrics:

    print(f"For ratio = {round(a['r'],2)}:")
    print(f"Training set: Precision = {a['Training Precision']}, Recall =␣
↪{a['Training Recall']}, F1 = {a['Training F1']}")
    print(f"Test set: Precision = {a['Test Precision']}, Recall = {a['Test␣
↪Recall']}, F1 = {a['Test F1']}")
```

```
For ratio = 0.1:
Training set: Precision = 1.0, Recall = 1.0, F1 = 1.0
Test set: Precision = 0.9999541337002638, Recall = 0.9997477874077131, F1 =
0.9998509499077036
For ratio = 0.2:
Training set: Precision = 1.0, Recall = 1.0, F1 = 1.0
Test set: Precision = 0.9997019784048967, Recall = 0.9998624294951163, F1 =
0.9997821975124663
For ratio = 0.3:
Training set: Precision = 1.0, Recall = 1.0, F1 = 1.0
Test set: Precision = 0.9995644898801201, Recall = 0.9998624294951163, F1 =
0.9997134374892539
For ratio = 0.4:
Training set: Precision = 1.0, Recall = 1.0, F1 = 1.0
Test set: Precision = 0.9994041343906128, Recall = 0.9998624294951163, F1 =
0.9996332294150009
For ratio = 0.5:
Training set: Precision = 1.0, Recall = 1.0, F1 = 1.0
Test set: Precision = 0.9991980570066905, Recall = 0.9998853579125969, F1 =
0.9995415893098627
For ratio = 0.6:
Training set: Precision = 1.0, Recall = 1.0, F1 = 1.0
Test set: Precision = 0.9992896099729593, Recall = 0.9998395010776356, F1 =
0.999564479897309
For ratio = 0.7:
Training set: Precision = 1.0, Recall = 1.0, F1 = 1.0
```

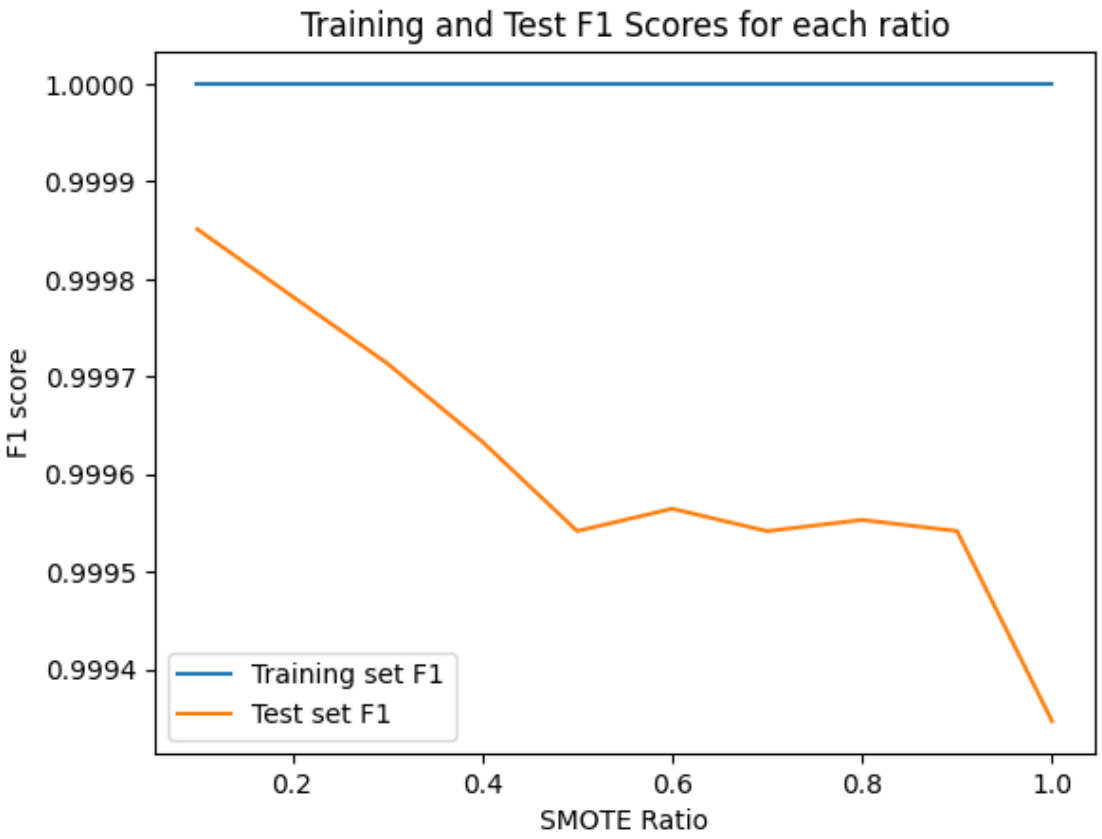
```
Test set: Precision = 0.9992438130155821, Recall = 0.9998395010776356, F1 =
0.9995415682948632
For ratio = 0.8:
Training set: Precision = 1.0, Recall = 1.0, F1 = 1.0
Test set: Precision = 0.9992667109695456, Recall = 0.9998395010776356, F1 =
0.9995530239647921
For ratio = 0.9:
Training set: Precision = 1.0, Recall = 1.0, F1 = 1.0
Test set: Precision = 0.9991751821472759, Recall = 0.9999082863300774, F1 =
0.9995415998166399
For ratio = 1.0:
Training set: Precision = 1.0, Recall = 1.0, F1 = 1.0
Test set: Precision = 0.9988547082942025, Recall = 0.9998395010776356, F1 =
0.9993468620733119
```

0.1.2 iii)

Below I draw a plot of training and testing F1 score as functions of my ratio. It seems that SMOTENC does not improve the decision tree's performance, as the F1 score generally decreases with an increase in the ratio.

[11]:
```
rs = [a['r'] for a in metrics]
trf1s = [a['Training F1'] for a in metrics]
tef1s = [a['Test F1'] for a in metrics]

plt.plot(rs, trf1s, label='Training set F1')
plt.plot(rs, tef1s, label='Test set F1')
plt.title('Training and Test F1 Scores for each ratio')
plt.xlabel('SMOTE Ratio')
plt.ylabel('F1 score')
plt.legend()
plt.show()
```


Training and Test F1 Scores for each ratio
1.0000
0.9999
0.9998
0.9997
0.9996
0.9995
0.9994
F1 score
Training set F1
Test set F1
0.2
0.4
0.6
0.8
1.0
SMOTE Ratio

[ ]: