Classwork_Sec_1_Week_5_ML

February 7, 2024

1 0.) Import the Credit Card Fraud Data From CCLE

[166]:
```
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
import random
```

[122]:
```
df = pd.read_csv("fraudTest.csv")
```

[123]:
```
df.head()
```

[123]:
```
   Unnamed: 0 trans_date_trans_time            cc_num  \
0           0   2020-06-21 12:14:25  2291163933867244
1           1   2020-06-21 12:14:33  3573030041201292
2           2   2020-06-21 12:14:53  3598215285024754
3           3   2020-06-21 12:15:15  3591919803438423
4           4   2020-06-21 12:15:17  3526826139003047

                              merchant        category    amt   first  \
0                fraud_Kirlin and Sons   personal_care   2.86    Jeff
1                 fraud_Sporer-Keebler   personal_care  29.84  Joanne
2  fraud_Swaniawski, Nitzsche and Welch  health_fitness  41.28  Ashley
3                    fraud_Haley Group        misc_pos  60.05   Brian
4                fraud_Johnston-Casper          travel   3.19  Nathan

       last gender                       street  …      lat      long  \
0   Elliott      M            351 Darlene Green  …  33.9659  -80.9355
1  Williams      F             3638 Marsh Union  …  40.3207 -110.4360
2     Lopez      F         9333 Valentine Point  …  40.6729  -73.5365
3  Williams      M  32941 Krystal Mill Apt. 552  …  28.5697  -80.8191
4    Massey      M     5783 Evan Roads Apt. 465  …  44.2529  -85.0170

   city_pop                        job         dob  \
0    333497        Mechanical engineer  1968-03-19
1       302  Sales professional, IT  1990-01-17
2     34496         Librarian, public  1970-10-21
3     54767              Set designer  1987-07-25
```

```
4      1126      Furniture designer  1955-07-06

                          trans_num   unix_time  merch_lat  merch_long  \
0  2da90c7d74bd46a0caf3777415b3ebd3  1371816865  33.986391  -81.200714
1  324cc204407e99f51b0d6ca0055005e7  1371816873  39.450498 -109.960431
2  c81755dbbbea9d5c77f094348a7579be  1371816893  40.495810  -74.196111
3  2159175b9efe66dc301f149d3d5abf8c  1371816915  28.812398  -80.883061
4  57ff021bd3f328f8738bb535c302a31b  1371816917  44.959148  -85.884734

   is_fraud
0         0
1         0
2         0
3         0
4         0

[5 rows x 23 columns]
```

[124]:
```
df_select = df[["trans_date_trans_time", "category", "amt", "city_pop",␣
↪"is_fraud"]]

df_select["trans_date_trans_time"] = pd.
↪to_datetime(df_select["trans_date_trans_time"])
df_select["time_var"] = [i.second for i in df_select["trans_date_trans_time"]]

X = pd.get_dummies(df_select, ["category"]).drop(["trans_date_trans_time",␣
↪"is_fraud"], axis = 1)
y = df["is_fraud"]
```

```
/var/folders/vn/pldcj5450lbfrdv8wcxb0mlm0000gn/T/ipykernel_59229/2282180580.py:3
: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  df_select["trans_date_trans_time"] =
pd.to_datetime(df_select["trans_date_trans_time"])
/var/folders/vn/pldcj5450lbfrdv8wcxb0mlm0000gn/T/ipykernel_59229/2282180580.py:4
: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  df_select["time_var"] = [i.second for i in df_select["trans_date_trans_time"]]
```

2 1.) Use scikit learn preprocessing to split the data into 70/30 in out of sample

[125]:
```
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
```

[ ]:
```

```

[126]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = .3)
```

[127]:
```
X_test, X_holdout, y_test, y_holdout = train_test_split(X_test, y_test,␣
↪test_size = .5)
```

[128]:
```
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
X_holdout = scaler.transform(X_holdout)
```

3 2.) Make three sets of training data (Oversample, Undersample and SMOTE)

[129]:
```
from imblearn.over_sampling import RandomOverSampler
from imblearn.under_sampling import RandomUnderSampler
from imblearn.over_sampling import SMOTE
```

[130]:
```
ros = RandomOverSampler()
over_X, over_y = ros.fit_resample(X_train, y_train)

rus = RandomUnderSampler()
under_X, under_y = rus.fit_resample(X_train, y_train)

smote = SMOTE()
smote_X, smote_y = smote.fit_resample(X_train, y_train)
```

[78]:
```
y_train.value_counts()
```

[78]:
```
is_fraud
0    387488
1      1515
Name: count, dtype: int64
```

[79]:
```
over_y.value_counts()
```

[79]:
```
is_fraud
0    387488
1    387488
```

```
Name: count, dtype: int64
```

[80]:
```
under_y.value_counts()
```

[80]:
```
is_fraud
0    1515
1    1515
Name: count, dtype: int64
```

[81]:
```
smote_y.value_counts()
```

[81]:
```
is_fraud
0    387488
1    387488
Name: count, dtype: int64
```

4 3.) Train three logistic regression models

[82]:
```
from sklearn.linear_model import LogisticRegression
```

[83]:
```
over_log = LogisticRegression().fit(over_X, over_y)

under_log = LogisticRegression().fit(under_X, under_y)

smote_log = LogisticRegression().fit(smote_X, smote_y)
```

[ ]:

[ ]:

[ ]:

[ ]:

[ ]:

[ ]:

5 4.) Test the three models

[84]:
```
over_log.score(X_test, y_test)
```

[84]:
```
0.9272295400561433
```

[85]:
```
under_log.score(X_test, y_test)
```

[85]: 0.9312843398354087

[86]:
```
smote_log.score(X_test, y_test)
```

[86]: 0.9233426905635932

[87]:
```
# We see SMOTE performing with higher accuracy but is ACCURACY really the best␣
↪measure?
```

[ ]:
```

```

6 5.) Which performed best in Out of Sample metrics?

[88]:
```
# Sensitivity here in credit fraud is more important as seen from last class
```

[89]:
```
from sklearn.metrics import confusion_matrix
```

[90]:
```
y_true = y_test
```

[91]:
```
y_pred = over_log.predict(X_test)
cm = confusion_matrix(y_true, y_pred)
cm
```

[91]:
```
array([[77081,  5982],
       [   84,   211]])
```

[92]:
```
print("Over Sample Sensitivity : ", cm[1,1] /( cm[1,0] + cm[1,1]))
```

```
Over Sample Sensitivity :  0.7152542372881356
```

[93]:
```
y_pred = under_log.predict(X_test)
cm = confusion_matrix(y_true, y_pred)
cm
```

[93]:
```
array([[77419,  5644],
       [   84,   211]])
```

[94]:
```
print("Under Sample Sensitivity : ", cm[1,1] /( cm[1,0] + cm[1,1]))
```

```
Under Sample Sensitivity :  0.7152542372881356
```

[95]:
```
y_pred = smote_log.predict(X_test)
cm = confusion_matrix(y_true, y_pred)
cm
```

[95]:
```
array([[76757,  6306],
       [   84,   211]])
```

[96]:
```
print("SMOTE Sample Sensitivity : ", cm[1,1] /( cm[1,0] + cm[1,1]))
```

```
SMOTE Sample Sensitivity :  0.7152542372881356
```

[97]:
```
X.columns
```

[97]:
```
Index(['amt', 'city_pop', 'time_var', 'category_entertainment',
       'category_food_dining', 'category_gas_transport',
       'category_grocery_net', 'category_grocery_pos',
       'category_health_fitness', 'category_home', 'category_kids_pets',
       'category_misc_net', 'category_misc_pos', 'category_personal_care',
       'category_shopping_net', 'category_shopping_pos', 'category_travel'],
      dtype='object')
```

[99]:
```
X_train.shape
```

[99]:
```
(389003, 17)
```

[57]:
```
len(y_train)
```

[57]:
```
389003
```

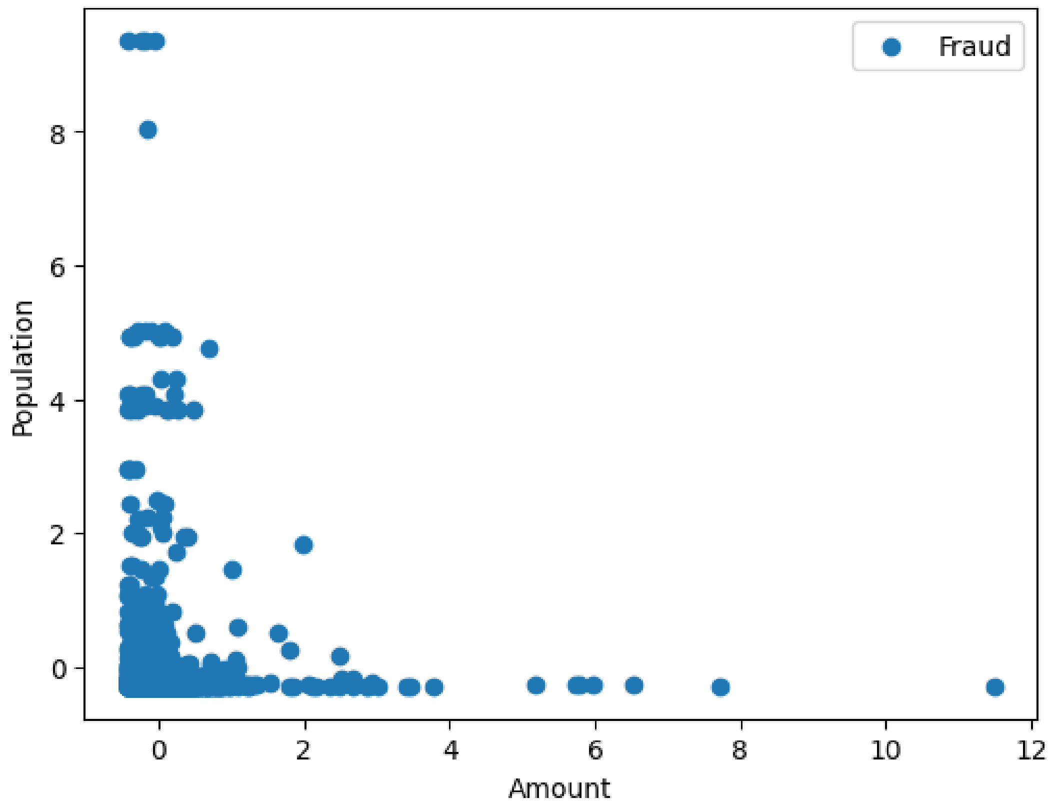
7 6.) Pick two features and plot the two classes before and after SMOTE.

[101]:
```
raw_temp = pd.concat([pd.DataFrame(X_train,columns = X.columns), y_train], axis␣
 ↪=1)
```

[102]:
```
#plt.scatter(raw_temp[raw_temp["is_fraud"] == 0]["amt"],␣
 ↪raw_temp[raw_temp["is_fraud"] == 0]["city_pop"])

plt.scatter(raw_temp[raw_temp["is_fraud"] == 1]["amt"],␣
 ↪raw_temp[raw_temp["is_fraud"] == 1]["city_pop"])
plt.legend(["Fraud", "Not Fraud"])
plt.xlabel("Amount")
plt.ylabel("Population")

plt.show()
```

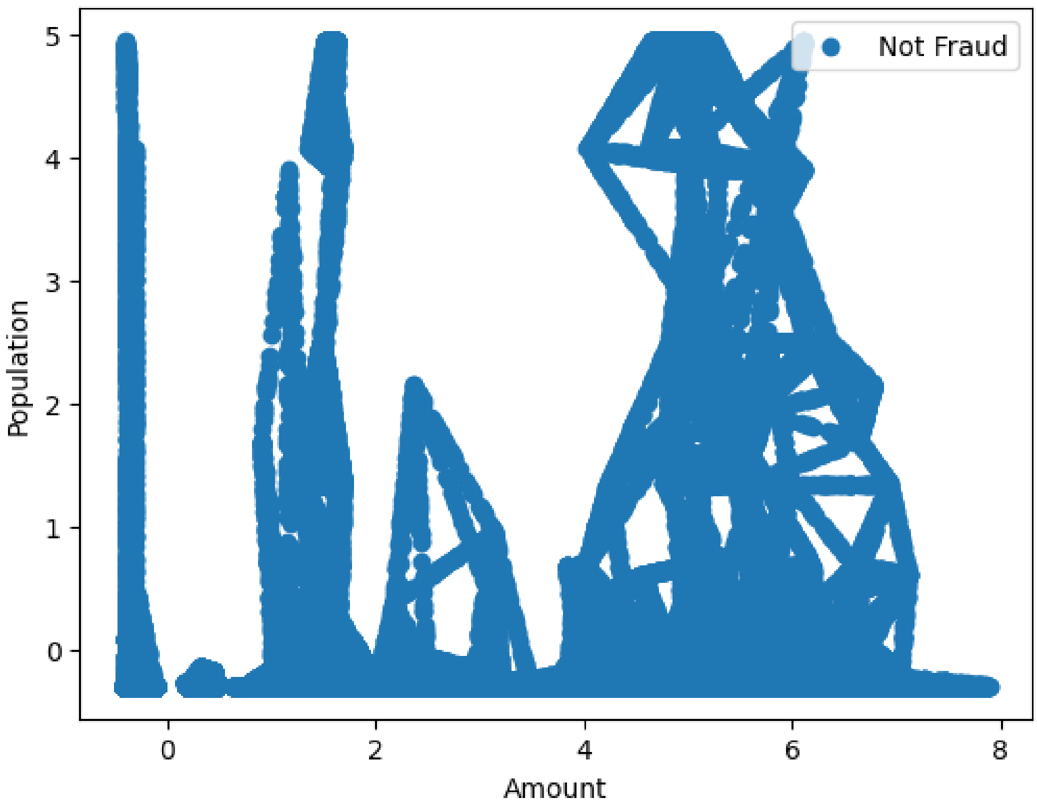



```
[104]: raw_temp = pd.concat([pd.DataFrame(smote_X,columns = X.columns), smote_y], axis␣
        ↪=1)
```

```
[105]: #plt.scatter(raw_temp[raw_temp["is_fraud"] == 0]["amt"],␣
        ↪raw_temp[raw_temp["is_fraud"] == 0]["city_pop"])

       plt.scatter(raw_temp[raw_temp["is_fraud"] == 1]["amt"],␣
        ↪raw_temp[raw_temp["is_fraud"] == 1]["city_pop"])
       plt.legend([ "Not Fraud", "Fraud"])
       plt.xlabel("Amount")
       plt.ylabel("Population")

       plt.show()
```


Not Fraud
Population
Amount
5
4
3
2
1
0
0
2
4
6
8

[ ]:

8 7.) We want to compare oversampling, Undersampling and SMOTE across our 3 models (Logistic Regression, Logistic Regression Lasso and Decision Trees).

9 Make a dataframe that has a dual index and 9 Rows.

10 Calculate: Sensitivity, Specificity, Precision, Recall and F1 score. for out of sample data.

11 Notice any patterns across perfomance for this model. Does one totally out perform the others IE. over/under/smote or does a model perform better DT, Lasso, LR?

12 Choose what you think is the best model and why. test on Holdout

[210]:
```
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import confusion_matrix, precision_score, recall_score,␣
 ↪f1_score
import pandas as pd
```

[211]:
```
resampling_methods = {
    'over': RandomOverSampler(),
    'under': RandomUnderSampler(),
    'smote': SMOTE()
}

model_configs = {
    'LOG': LogisticRegression(),
    "LASSO": LogisticRegression(penalty = "l1",
                                solver = 'liblinear', C=0.5),
    "DecisionTree": DecisionTreeClassifier()
}
```

[214]:
```
def calc_perf_metrics(y_true,y_pred):
    cm_1 = confusion_matrix(y_true, y_pred)
    tn, fp, fn, tp = cm_1[0][0], cm_1[0][1], cm_1[1][0], cm_1[1][1]

    sensitivity = tp / (tp + fn)
    specificity = tn / (tn + fp)
    precision = tp / (tp + fp)
    recall = sensitivity
    f1_score = 2 * (precision * recall) / (precision + recall)

    return sensitivity, specificity, precision, recall, f1_score
```

[220]:
```
random.seed(42)
for model_name, model in model_configs.items():
    for resample_key, resampler in resampling_methods.items():
        resample_X, resample_y = resampler.fit_resample(X_train, y_train)


        combined_key = f"{resample_key}_{model_name}"

        trained_models[combined_key] = model.fit(resample_X, resample_y)

        Y_pred = trained_models[combined_key].predict(X_holdout)

        perf_metrics = calc_perf_metrics(y_true, Y_pred)

        results[combined_key] = perf_metrics
```

[221]:
```
out = pd.DataFrame(results, index=['sensitivity', 'specificity', 'precision',␣
↪'recall', 'f1_score']).transpose()
out
```

[221]:

	sensitivity	specificity	precision	recall	f1_score
over_LOG	0.084746	0.926935	0.004102	0.084746	0.007826
over_LASSO	0.091525	0.926538	0.004405	0.091525	0.008406
over_DecisionTree	0.010169	0.996593	0.010490	0.010169	0.010327
under_LOG	0.081356	0.933015	0.004295	0.081356	0.008159
under_LASSO	0.094915	0.924359	0.004437	0.094915	0.008477
under_DecisionTree	0.044068	0.945198	0.002848	0.044068	0.005350
smote_LOG	0.091525	0.925201	0.004327	0.091525	0.008263
smote_LASSO	0.094915	0.924082	0.004421	0.094915	0.008448
smote_DecisionTree	0.010169	0.990465	0.003774	0.010169	0.005505

For each of the measures we observe very similar values for logistic regression and Lasso, across all resampling methods. Decision tree on the other hand seems to under perform, except when considering specificity. More specifically for each measure we see the following.

Sensitivity: The sensitivity values are generally low across all models and resampling methods, indicating that the models have difficulty in correctly identifying positive instances. The decision tree models (across resampling methods) generally have higher sensitivity compared to logistic and lasso regression models.

Specificity: Specificity values are relatively high across all models and resampling methods, indicating a good ability to correctly identify negative instances. This is the only case where Decision Trees outperform the other two. There isn't much variation in specificity across different models and resampling methods.

Precision: Precision values are very low across all models and resampling methods, indicating a high number of false positives compared to true positives. Logistic and lasso regression models generally have slightly higher precision compared to decision tree models.

Recall: Recall values are consistent with sensitivity values, as they represent the same metric.

Similar to sensitivity, recall values are generally low across all models and resampling methods.

F1 Score: F1 scores are calculated based on precision and recall and provide a balanced measure of a model's performance. F1 scores are low across all models and resampling methods, indicating poor overall performance in terms of both precision and recall.