HW2 Report

Miner2 username: hkhurana
Mason userID: hkhurana (G01229319)
Best Public Score: 0.66

I approached the assignment in the following three steps:

1. Feature Selection and Data Analysis
2. Locally experimenting to find the best classifier

Feature Selection and Data Analysis:

First I imported all the essential libraries I would be needing for the program. Then I read the train dataset and test dataset into train_data and test_data.

Then I did some **checking for null values** which turned out to be zero. Post this, I **dropped the 'id' column** since it is an unnecessary and irrelevant attribute and would not contribute to the prediction. Then I **renamed all columns to meaningful names** based on their description given on Miner.

After closely assessing the dataset, I realized that for a model to run properly on such a tabular dataset, I would need all values to be numeric. So, in order to ensure this, I **encoded the gender and race attributes** using numerical categorical encoding.

```
print("Distribution of race attribute")
print(train_data['race'].value_counts())
train_data['race'] = train_data['race'].astype('category')
train_data['race_encoded'] = train_data['race'].cat.codes
```

Next, in order to run the model efficiently, and to reduce the calculation complexity for models behind the scenes, we perform **normalization** of continuous data attributes which was my next step in my approach.

```
def normalize(df):
    result = df.copy()
    for feature_name in df.columns:
        max_value = df[feature_name].max()
        min_value = df[feature_name].min()
        result[feature_name] = (df[feature_name] - min_value) / (max_value - min_value)
    return result
```

Before normalization

yearsSinceLastDegree	hoursWorkedPerWeek
13	40
13	13
9	40
7	40
13	40
...	...
12	38
9	40
9	40
9	20

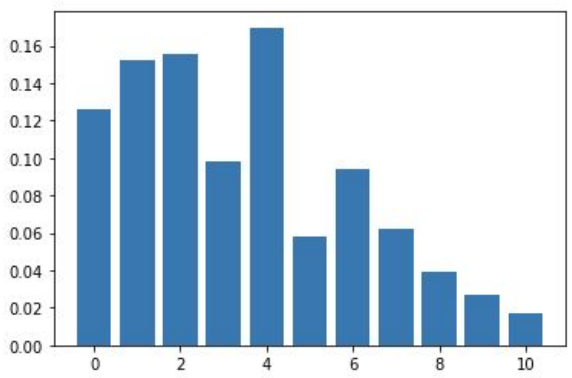
After normalization

yearsSinceLastDegree	hoursWorkedPerWeek
0.800000	0.397959
0.800000	0.122449
0.533333	0.397959
0.400000	0.397959
0.800000	0.397959
...	...
0.733333	0.377551
0.533333	0.397959
0.533333	0.397959
0.533333	0.193878

For further precision in the calculations, I converted all attributes to the float data type.

Next, in order to remove irrelevant and unimportant attributes, I calculated the **feature importance** for all the features in the dataset.

```
# random forest for feature importance on a classification problem
from sklearn.ensemble import RandomForestClassifier, RandomForestRegressor
# define the model
model = RandomForestClassifier()
# fit the model
model.fit(train_data[feature_cols], train_data['credit'])
# get importance
importance = model.feature_importances_
# summarize feature importance
for i,v in enumerate(importance):
    print('Feature: %s, Score: %.5f' % (i,v))
# plot feature importance
plt.bar([x for x in range(len(importance))], importance)
plt.show()
```



Looking at the feature importance graph, it was clear that the last two features - race and gender were the least important. However, testing on Miner gave a lower F1 score than before after removing those attributes. Hence I chose to keep them as is.

One of the major issues with the dataset given was the imbalance in the classes.

```
from collections import Counter
a = Counter(train_data['credit'])
a
```

```
Counter({0: 24720, 1: 7841})
```

From the image above, it is clear that there are thrice as many data points labelled 0 as there are for 1. This sometimes creates an issue of biasing the model towards predicting 0 most of the time and thus giving inaccurate results. In order to combat this, we have to resample the dataset using **upsampling or downsampling**. Upsampling simply means increasing the number of the minority class data points to match the number of majority data points. Whereas, downsampling means decreasing the number of majority class data points to match the number of minority data points. I performed both upsampling and downsampling and tested my models on all three kinds of datasets - normal, upsampled and downsampled.

Downsampling:

```
train_data_majority = train_data[train_data['credit']==0]
train_data_minority = train_data[train_data['credit']==1]

train_data_majority_downsampled = resample(train_data_majority,
                                 replace=False,                    # sample without replacement
                                 n_samples=len(train_data_minority),     # to match minority class size
                                 random_state=123)                 # reproducible results


# Combine minority class with downsampled majority class
train_data_downsampled = pd.concat([train_data_majority_downsampled, train_data_minority])

# Display new class counts
train_data_downsampled['credit'].value_counts()
```

Upsampling:

```
train_data_majority = train_data[train_data['credit']==0]
train_data_minority = train_data[train_data['credit']==1]

train_data_minority_upsampled = resample(train_data_minority,
                                 replace=True,                 # sample without replacement
                                 n_samples=len(train_data_majority),     # to match minority class
                                 random_state=123)              # reproducible results


# Combine minority class with downsampled majority class
train_data_upsampled = pd.concat([train_data_minority_upsampled, train_data_majority])

# Display new class counts
train_data_upsampled['credit'].value_counts()
```

Locally experimenting to find the best classifier

Step 2 of my approach was to experiment with different classification algorithms and find the one which gave the highest F1 score for the dataset. I tested my dataset on the following 5 models:

1. Logistic Regression
2. KNN
3. Decision Tree
4. Random Forest Ensemble Learning
5. Multi Layer Perceptron Classifier (Neural Network)

The hyperparameters used in the models were determined experimentally for best results.

```
from sklearn.model_selection import KFold, cross_val_score
cv = KFold(n_splits=5, random_state=101, shuffle=True)

knn = KNeighborsClassifier(n_neighbors = 3, n_jobs=-1)
clf = DecisionTreeClassifier(criterion='gini', max_depth=30)
rfm = RandomForestClassifier(n_estimators=80, oob_score=False, n_jobs=-1,
                             min_samples_leaf = 10, max_leaf_nodes=100, max_depth=20)
mlp = MLPClassifier(alpha=1, max_iter=1000, hidden_layer_sizes=(30,30,30))
lr  = LogisticRegression(max_iter=40000)

models = [knn, clf, rfm, mlp, lr]

scores_normal, scores_downsampled, scores_upsampled = dict(), dict(), dict()

for model in models:
    scores_upsampled[model] = np.mean(cross_val_score(model, train_data_upsampled[feature_cols],
                                                      train_data_upsampled['credit'], scoring='f1',
                                                      cv=cv, n_jobs=-1))
    scores_downsampled[model] = np.mean(cross_val_score(model, train_data_downsampled[feature_cols],
                                                        train_data_downsampled['credit'], scoring='f1',
                                                        cv=cv, n_jobs=-1))
    scores_normal[model] = np.mean(cross_val_score(model, train_data[feature_cols],
                                                   train_data['credit'], scoring='f1',
                                                   cv=cv, n_jobs=-1))
```

Conclusion:

After experimenting and performing cross validation for the aforementioned models, I chose the **Random Forest classifier** since it gives better F1 score on the downsampled and the normal datasets. It however loses against the Decision Tree classifier for the upsampled dataset. Now, after having chosen the Random Forest CLassifier, I tried training it with all three types of datasets - upsampled, downsampled, and normal, and tested it on Miner. And the **upsampled** variant gave me the best F1 score i.e. **0.66.**