Data Mining Competition Report

You can run this notebook in jupyter notebook!
You may need to install pandas, sci-kit learn, and XGBoost

1. Data Processing

Problem: Numerical & Categorical Data | Empty Cells

In this specific problem the challenge of having a mix of numerical and categorical data was present. In the process of looking for the best way to process the model a variety of steps were taken in order to format the data for machine learning classifiers. In specific the letters that existed in certain columns needed to be converted to integers.

Solution: Replace String Labels | Address Empty Cells

In my final implementation for my model I replaced every letter with the next following integer in each corresponding column. This was done using the following function that I created.

```
"""
Function passes through a pandas dataframe and can be configured to
(drop rows with empty cells or fill empty cells with -1).
It also replaces the corresponding letter to the next unique integer in their respective categ
ory.
"""

def cleanHH(df):



    # Remove rows with empty cells
    #df.dropna(inplace=True)

    # Replace empty cells with -1
    df.fillna(-1,inplace=True)

    # Encode letters to ints in MTGd (B,D -> 9,10) & MTGdS (B,D,R -> 9,10,11)
    df.OWN.replace(to_replace=dict(D=3,R=4), inplace=True)
    df.MTGd.replace(to_replace=dict(B=9,D=10), inplace=True)
    df.MTGdS.replace(to_replace=dict(B=9,D=10,R=11), inplace=True)
```

This function was very useful as it made it possible to use machine learning models afterwards.
Ultimately the code to address empty cells was discarded in my final implementation due to XGBoost automatically handling these blank empty cells

2. Data Mining Models

For every model except XGBoost I used Sci-Kit Learn to create models

Regression

- Logistic Regression - **Accuracy: 88.63%**

Decision Tree

- Decision Tree Classifier - **Accuracy: 88.59%**
- Random Forest Classifier - **Accuracy: 91.53%** *(After Hyperparameter tuning)*

Boosting

- Gradient Boosting Classifier - **Accuracy: 92.04%** *(After Hyperparameter tuning)*
- Ada Boost Classifier - **Accuracy: 91.51%**
- Histogram-based Gradient Boosting Classifier **Accuracy: 91.69%**
- XGBoost - **Accuracy: 92.55%** *(After Hyperparameter tuning)*

Neural Networks

- Custom Keras/TF Neural Net - **Accuracy: 88.99%**

As seen above I used a variety of modeling types in my approach. Logistic and Decision Tree's were obvious choices in a binary classification problem but were not delivering the accuracy desired. I moved to boosting classifiers which were giving me better results and had to find one I was comfortable with to tune. I used a Neural Network out of curiosity to see how it would fare against a binary classification problem. I was not able to push it past 88.99% accuracy no matter how many hidden layers I added.

3. Actual Model Used [XGBoost]

I ultimately went with XGBoost due to it's consistent high accuracy rate compared to the other models. It also had a much higher precision rate for True Positives as well which maximizes profit. The implementation I used was the following:

Import, store and append data into Pandas Data Frame

In [1]:
```
import pandas as pd


df1 = pd.read_excel('households_1.xlsx')
df2 = pd.read_excel('households_2_SOLN.xlsx')
df3 = pd.read_excel('households_3.xlsx')

# Combine together
df12 = df1.append(df2, ignore_index=True)
```

Create function to convert string labels to integers

```
In [2]: def cleanHH2(df):
            #pass list through and drop columns


            # Remove rows if empty
            #df.dropna(inplace=True)   #delete rows with empty cells

            # retain rows and replace None with -1
            #df.fillna(-1,inplace=True) #replace None w/ -1

            # Encode letters to ints in MTGd (B,D -> 9,10) & MTGdS (B,D,R -> 9,10,11)
            df.OWN.replace(to_replace=dict(D=3,R=4), inplace=True)
            df.MTGd.replace(to_replace=dict(B=9,D=10), inplace=True)
            df.MTGdS.replace(to_replace=dict(B=9,D=10,R=11), inplace=True)

        cleanHH2(df12)
        cleanHH2(df3)
```

Create train/test split (50/50), shuffle and stratify data

```
In [3]: from sklearn.model_selection import train_test_split

        X = df12.drop(columns='HiEdInCh')
        y = df12['HiEdInCh'].copy()


        X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.5, shuffle=True,st
        ratify=y)
```

Hyperparameter tuned XGBoost

```
In [4]: import xgboost as xgb
        from sklearn.metrics import classification_report,confusion_matrix


        model=xgb.XGBClassifier(random_state=1,
                                learning_rate=0.09,
                                n_estimators=200,
                                gamma=1)


        model.fit(X_train, y_train)
        model.score(X_test,y_test)

        print(classification_report(y_test,model.predict(X_test)))
        print("Accuracy: ",model.score(X_test, y_test))
```

```
              precision    recall  f1-score   support

           0       0.95      0.97      0.96      8889
           1       0.68      0.58      0.62      1111

    accuracy                           0.92     10000
   macro avg       0.82      0.77      0.79     10000
weighted avg       0.92      0.92      0.92     10000

Accuracy:  0.9233
```

Predict HouseHold 3

In [5]:
```
## convert your array into a dataframe
dfm = pd.DataFrame (model.predict(df3))
dfm = dfm.rename(columns={0 : "HiEdInCh"})

## save to xlsx file

filepath = 'Predictions_EXAMPLE_ONLY_NOT_FINAL_RESULTS.xlsx'

dfm.to_excel(filepath, index=False)
```

4. Insights

1. Use boosted models, they are the most efficient for this problem.
2. Use GPU accelerated libraries once datasets become larger.
   - XGBoost has support for GPU Acceleration
3. Collect more information on peoples background more features could mean better indicators for better predictions
   - Use LinkedIn to determine if someone is college educated, use their API or scrape the web.
   - Make a heatmap using addresses of current students and warm leads to determine if certain neighborhoods are hot spots.

5. Table of models

Model (Classifier):	Accuracy (%):
XGBoost	92.55
Gradient Boosting	92.04
Histogram Based Gradient Boosting	91.69
Random Forest	91.53
Adaptive Boosting	91.51
Neural Network	88.99
Logistic Regression	88.63
Decision Tree	88.59