Dan Schumacher, 5/7/24

Enhancing Donor Prediction Accuracy for the National Veterans' Organization through Advanced Modeling Techniques

---

Introduction

Non-profit organizations significantly depend on donations to fund their operations and deliver services. The National Veterans' Organization (NVO), with a vast database of over 13 million potential donors, primarily utilizes direct mail for fundraising. However, with an average donation return of $13 against a mailing cost of $13.60 for every 20 letters sent, the organization faces a net loss. This scenario underlines the critical need for a more targeted marketing strategy that reduces costs and enhances the efficiency of fundraising efforts.

Objectives

The objective of this project is twofold:

1. Utilize various predictive modeling techniques to optimize the direct mail campaign, aiming to increase the profitability of the National Veterans' Organization.
2. Apply and showcase the modeling techniques learned in Dr. Campbell's class and beyond this semester, emphasizing practical applications in a real-world context.

Data Sources

The data was made available on Canvas by Dr. Campbell.

Details of the Data

The training data contains 3,000 records and is evenly balanced between Donors and Non Donors. There are 22 variables in the dataset. These include:

- Zip code category, homeowner status, number of children, income, sex, wealth category, homevalue, median family income, average family income, number of promotions sent to the individual, lifetime number of promotions received by the individual, sum of all donations given, dollar amount of last gift, months since last donation, number of months between previous 2 gifts, average gift from donor.

Types of analysis Performed

I performed both traditional modeling and utilized an artificial neural network to generate my predictions.

Dan Schumacher, 5/7/24

Traditional Modeling

In the traditional modeling, I started by creating dummy variables of all of the predictors. Next I removed predictor variables that were highly correlated with one another. Starting with all predictors, I would calculate the Variance Inflation Factor (Vif) of each variable, then if any variable had a Vif greater than the cut off (I used 10), I would remove the variable and recalculate the Vif without that variable. This process repeated until all Vifs were under the cut off. At the end of the process I had removed five variables

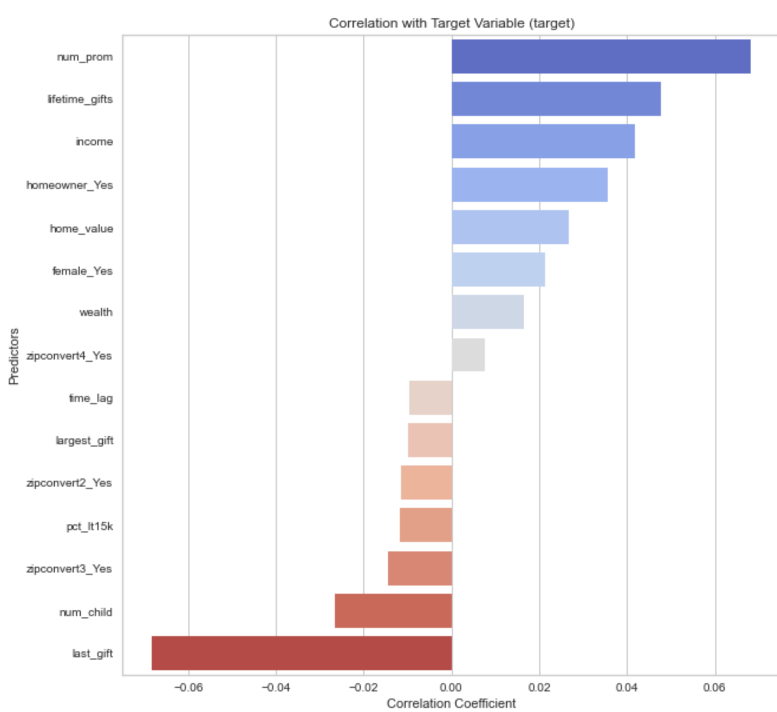
REMOVED: ['zipconvert5_Yes', 'avg_fam_inc', 'months_since_donate', 'med_fam_inc', 'avg_gift']

I fit the data to a Logistic Regression, LDA, QDA, KNN (k=1), and a Naive Bayes classifier.

The results are shown below. Our top performer on both the dev and test set was LDA. It is also noteworthy that LDA had the highest F1 score, achieving a good balance between precision and recall.

name	tp	tn	fp	fn	acc	prec	recall	f1	test_acc
Logistic Regression	167	151	145	137	0.530000	0.535256	0.549342	0.542208	0.533333
LDA	163	166	149	122	0.548333	0.522436	0.571930	0.546064	0.558333
QDA	26	268	286	20	0.490000	0.083333	0.565217	0.145251	0.525000
KNN	148	153	164	135	0.501667	0.474359	0.522968	0.497479	0.475000
Naïve Bayes	38	263	274	25	0.501667	0.121795	0.603175	0.202667	0.516667

To get a rough idea of what variables our model is using to generate predictions, we can make a correlation chart.



Here we can see that the larger the bar, the more correlated the variable is with the target. Red bars show negative correlations while blue bars show positive correlations. One thought I had, to achieve superior test accuracy, was to only use the most correlated variables on each side. I tested only using variables with an absolute correlation coefficient above .05. Unfortunately, This did not improve results.

Speaking of which, it is worth mentioning that I attempted many more models, and much more complex ones. I had a total of 79 entries on the competition's website. For the more complex models I tried transforming the continuous variables by log, squared, cubed, square root, inverse, boxcox, sigmoid, sine and cosine transformations. After Vif, in the complex modeling, I had a total of 84 variables. However, they did not help. Both the dev scores and test accuracy were several percentages shorter across each category.

ANN

Another approach that I took was making an artificial neural network for the classification task. This network had the following parameters:

1. An input layer with both categorical and continuous features processed via embedding and batch normalization respectively
2. Multiple hidden layers with configurations of [200, 100] neurons each incorporating ReLU activation and dropout regularization set at 0.4
3. Final output layer using a softmax function for binary classification
4. Robustly scaled input variables to boost performance
5. 100 epochs of training with an early stopping patience of 5
6. Cross validation with 20 splits

This approach achieved much better dev accuracy with an OOB accuracy of 59%. However, at test time it performed practically the same as LDA, and it had the downside of being much more complex, less interpretable, and doesn't always produce the same results.

Results

The models displayed varying levels of success:

- Accuracy Metrics: Each traditional model was evaluated based on accuracy, precision, recall, and F1-score. The LDA emerged as the most accurate model, as well as the one with the highest F1, showcasing a balance between precision and recall.
- Feature Importance: Looking at the correlation with target variable plot, we can see that the target variable is most strongly associated with number of promotions, and negatively associated with last gift. This loosely suggests that individuals who have donated recently are more likely to donate again, and people who receive a lot of promotions tend to donate more.

Discussion

The results indicated that, with all their bells and whistles, advanced modeling techniques such as artificial neural networks do not always have better predictive power then their traditional less complicated counterparts. However, none of the variables exhibited extremely high correlations with the target, suggesting potential limitations in the dataset's explanatory power.

Conclusion and Future Work

The study successfully demonstrated the potential predictive models to enhance donor prediction accuracy, thereby aiding the NVO in optimizing its fundraising strategies. For future work, further experimentation with ANN hyperparameters (such as learning rates, epochs, batch sizes, dropout rates) is recommended. Additionally, obtaining more comprehensive data, including demographic information, military service details, and any other variables that could potentially have a higher correlation with the target variable, would be useful. Having more variables more closely correlated to the target would significantly improve predictive power.

Appendices

The appendices contain two different notebooks. The first is the simple modeling I performed to achieve the best results. The second includes transformed variables (test accuracy was not included, as all results were in the mid 40s). Finally there is a python script that is my artificial neural network.

Simple Model By Hand

In [126]:
```
import pandas as pd
pd.set_option("max_colwidth", None)

import pycaret
import numpy as np
import matplotlib.pyplot as plt
from pycaret.classification import *
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score

from functions.homebrew import *
import numpy as np
import pandas as pd

from sklearn.linear_model import LogisticRegression
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis as LDA, QuadraticDiscriminantAnalysis as QDA
from sklearn.naive_bayes import GaussianNB
from sklearn.neighbors import KNeighborsClassifier
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split, cross_val_score
from sklearn.metrics import accuracy_score
from tqdm import tqdm
from itertools import combinations
import pickle
import os

# If you're using statsmodels or ISLP for specific tasks, keep these imports
import statsmodels.api as sm
# Assuming ISLP and homebrew are custom modules specific to your project
from ISLP import load_data, confusion_table
from ISLP.models import ModelSpec as MS, summarize, contrast
import statsmodels.api as sm
from scipy import stats
```

Helper Functions

In [127]:
```
def convert_confusion_matrix(df, name):
    """
    Converts a confusion matrix dataframe into a format with columns for model name, TP, TN, FP, FN.

    Args:
    df (pd.DataFrame): Confusion matrix dataframe with multi-index (Truth, Predicted) and columns [0, 1].

    Returns:
    pd.DataFrame: Reformatted dataframe with model evaluation metrics.
    """
    # Extracting the values from the confusion matrix
    tn, fp, fn, tp = df.iloc[0, 0], df.iloc[0, 1], df.iloc[1, 0], df.iloc[1, 1]
    acc = (tp + tn) / (tp + tn + fp + fn)
    prec = tp / (tp +fp)
    recall = tp / (tp + fn)
    f1 = 2 * ((prec * recall)/(prec + recall))
    # Creating a new dataframe with the desired format
    metrics_df = pd.DataFrame({
        "name": name,
        "tp": [tp],
        "tn": [tn],
        "fp": [fp],
        "fn": [fn],
        'acc': acc,
        'prec': prec,
        'recall': recall,
        'f1': f1
    })

    return metrics_df
```

In [128]:
```
def format_results(df):
    df = np.where(df == 1, 'Donor','No Donor')
    return df
```

LOAD DATA

In [140]:
```
df = pd.read_csv('./data/df.csv').drop('Unnamed: 0', axis=1)
```

In [141]:
```python
train = df[df['type'] == 'train'].drop('type',axis =1)
dev = df[df['type'] == 'dev'].drop('type',axis =1)
test = df[df['type'] == 'test'].drop('type',axis =1)
```

VIF

In [142]:
```python
dummies = pd.get_dummies(df, drop_first=True)

kept, removed = remove_high_vif_features(X=dummies.drop('target_No Donor', axis=1), y=dummies['target_No Donor'], vif_threshol
print('REMOVED:', removed)
```

```
REMOVED: ['avg_fam_inc', 'months_since_donate', 'zipconvert5_Yes', 'med_fam_inc', 'avg_gift']
```

In [144]:
```python
kept['target'] = (df['target'] == 'Donor').astype(int)
```

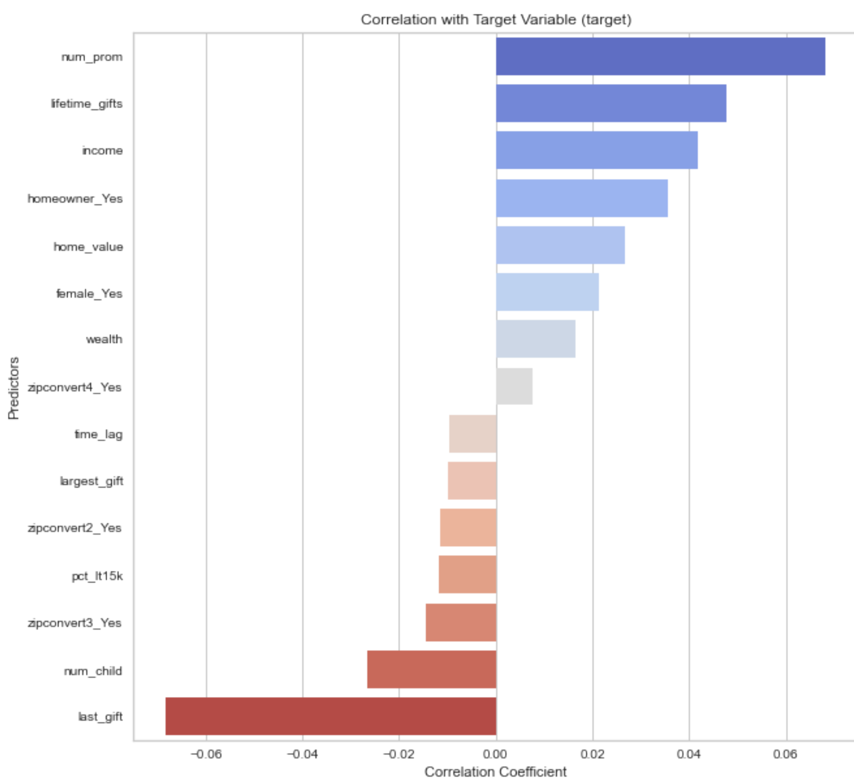
In [145]:
```python
train = kept[kept['type_train'] ==1]
dev = kept[(kept['type_test'] == 0) & (kept['type_train'] == 0)]
test = kept[kept['type_test'] ==1]
```

In [154]:
```python
# for data in [train, dev, test]:
#     data.drop('type_train', inplace = True, axis = 1)
#     data.drop('type_test', inplace = True, axis = 1)
test = test.drop('target',axis = 1)
```

In [174]:
```python
def cor_bars(df):
    corr = df.corr()
    # Isolating the column that represents the correlation with the target variable
    target_corr = corr['target'].sort_values(ascending=False)

    # Removing the target variable from itself to avoid a perfect correlation display
    target_corr = target_corr.drop(labels=['target'])

    # Plotting the correlations for visual representation
    plt.figure(figsize=(10, 10))
    sns.barplot(x=target_corr.values, y=target_corr.index, palette='coolwarm')
    plt.title('Correlation with Target Variable (target)')
    plt.xlabel('Correlation Coefficient')
    plt.ylabel('Predictors')
    plt.show()
cor_bars(train)
```



Logistic Regression

In [155]:
```python
for col in kept.columns:
    print(col)
```

```
num_child
income
wealth
home_value
pct_lt15k
num_prom
lifetime_gifts
largest_gift
last_gift
time_lag
zipconvert2_Yes
zipconvert3_Yes
zipconvert4_Yes
homeowner_Yes
female_Yes
type_test
type_train
target
```

In [156]:
```python
results_df = pd.DataFrame()
```

In [157]:
```python
# Selecting features and target variable for training data
X_train = train.drop(['target'], axis =1 )
y_train = train['target']
X_test = dev.drop(['target'], axis = 1)
y_test = dev['target']

# Fitting logistic regression model
glm = sm.GLM(y_train, X_train, family=sm.families.Binomial())
glm = glm.fit()

# Summarizing results
# print(results.summary())
```

In [158]:
```python
log_preds = (glm.predict(X_test) >= 0.5).astype(int)
log_acc = accuracy_score(log_preds, y_test)
print(log_acc)

d = confusion_table(log_preds,y_test)
results_df = pd.concat([results_df,convert_confusion_matrix(d, 'Logistic Regression')])

log_test_preds = (glm.predict(test) >= 0.5).astype(int)
log_test_preds = format_results(log_test_preds)

save_df = pd.DataFrame(log_test_preds, columns=['values'])
save_df.to_csv('./preds/log.csv', index=False)
```

```
0.53
```

LDA

In [159]:
```python
lda = LDA(store_covariance=True)
lda.fit(X_train, y_train)

lda_preds = lda.predict(X_test)

lda_acc = accuracy_score(lda_preds,y_test)
print(lda_acc)

d = confusion_table(lda_preds,y_test)
results_df = pd.concat([results_df,convert_confusion_matrix(d, 'LDA')])


lda_test_preds = (lda.predict(test) >= 0.5).astype(int)
lda_test_preds = format_results(lda_test_preds)

save_df = pd.DataFrame(lda_test_preds, columns=['values'])
save_df.to_csv('./preds/lda.csv', index=False)
```

```
0.5483333333333333
```

QDA.

In [160]:
```python
qda = QDA(store_covariance=True)
qda.fit(X_train, y_train)

qda_preds = qda.predict(X_test)

qda_acc = accuracy_score(qda_preds,y_test)

print(qda_acc)

d = confusion_table(qda_preds,y_test)
results_df = pd.concat([results_df,convert_confusion_matrix(d, 'QDA')])

qda_test_preds = (qda.predict(test) >= 0.5).astype(int)
qda_test_preds = format_results(qda_test_preds)

save_df = pd.DataFrame(qda_test_preds, columns=['values'])
save_df.to_csv('./preds/qda.csv', index=False)
```

```
0.49
```

KNN

In [161]:
```python
df['type']
```

Out[161]:
```
0       train
1       train
2         dev
3       train
4       train
        ...
3115     test
3116     test
3117     test
3118     test
3119     test
Name: type, Length: 3120, dtype: object
```

In [162]:
```python
knn1 = KNeighborsClassifier(n_neighbors=1)
knn1.fit(X_train, y_train)
knn1_pred = knn1.predict(X_test)
knn1_acc = accuracy_score(knn1_pred,y_test)

print(knn1_acc)

d = confusion_table(knn1_pred, y_test)
results_df = pd.concat([results_df,convert_confusion_matrix(d, 'KNN')])

knn1_test_preds = (knn1.predict(test) >= 0.5).astype(int)
knn1_test_preds = format_results(knn1_test_preds)

save_df = pd.DataFrame(knn1_test_preds, columns=['values'])
save_df.to_csv('./preds/knn1.csv', index=False)
```

```
0.5016666666666667
```

NB

In [163]:
```python
nb = GaussianNB()
nb.fit(X_train, y_train)
nb_preds = nb.predict(X_test)
nb_acc = accuracy_score(nb_preds,y_test)

print(nb_acc)
save_df = pd.DataFrame(nb_preds, columns=['values'])
save_df.to_csv('./preds/nb.csv', index=False)

d = confusion_table(nb_preds, y_test)
results_df = pd.concat([results_df,convert_confusion_matrix(d, 'Naïve Bayes')])

nb_test_preds = (nb.predict(test) >= 0.5).astype(int)
nb_test_preds = format_results(nb_test_preds)

save_df = pd.DataFrame(nb_test_preds, columns=['values'])
save_df.to_csv('./preds/nb.csv', index=False)
```

```
0.5016666666666667
```

In [1]:
```python
# results_df
```

In [168]:
```python
test_acc = { # these are from running on the website
    'log': 0.5333333,
    'lda': 0.5583333,
    'qda': 0.525,
    'knn':  0.475,
    'nb': 0.5166667,
}
```

In [169]:
```python
results_df['test_acc'] = test_acc.values()
```

In [170]:
```python
results_df
```

Out[170]:

	name	tp	tn	fp	fn	acc	prec	recall	f1	test_acc
0	Logistic Regression	167	151	145	137	0.530000	0.535256	0.549342	0.542208	0.533333
0	LDA	163	166	149	122	0.548333	0.522436	0.571930	0.546064	0.558333
0	QDA	26	268	286	20	0.490000	0.083333	0.565217	0.145251	0.525000
0	KNN	148	153	164	135	0.501667	0.474359	0.522968	0.497479	0.475000
0	Naïve Bayes	38	263	274	25	0.501667	0.121795	0.603175	0.202667	0.516667

By hand Complex

In [49]:

```
import pandas as pd
pd.set_option("max_colwidth", None)

import pycaret
import numpy as np
import matplotlib.pyplot as plt
from pycaret.classification import *
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score

from functions.homebrew import *
import numpy as np
import pandas as pd

from sklearn.linear_model import LogisticRegression
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis as LDA, QuadraticDiscriminantAnalysis as QDA
from sklearn.naive_bayes import GaussianNB
from sklearn.neighbors import KNeighborsClassifier
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split, cross_val_score
from sklearn.metrics import accuracy_score
from tqdm import tqdm
from itertools import combinations
import pickle
import os

# If you're using statsmodels or ISLP for specific tasks, keep these imports
import statsmodels.api as sm
# Assuming ISLP and homebrew are custom modules specific to your project
from ISLP import load_data, confusion_table
from ISLP.models import ModelSpec as MS, summarize, contrast
import statsmodels.api as sm
from scipy import stats
```

Helper Functions

In [50]:

```
def add_transformations(data, cont_cols):
    for var in cont_cols:
        data[f'log_{var}'] = np.log(data[var] + 1)
        data[f'sq_{var}'] = data[var]**2
        data[f'sqrt_{var}'] = np.sqrt(data[var])
        data[f'inv_{var}'] = 1 / (data[var] + 1)
        data[f'boxcox_{var}'], _ = stats.boxcox(data[var] + 1)
        data[f'sigmoid_{var}'] = 1 / (1 + np.exp(-data[var]))
        data[f'sin_{var}'] = np.sin(data[var])
        data[f'cos_{var}'] = np.cos(data[var])
```

In [51]:
```python
def convert_confusion_matrix(df, name):
    """
    Converts a confusion matrix dataframe into a format with columns for model name, TP, TN, FP, FN.

    Args:
    df (pd.DataFrame): Confusion matrix dataframe with multi-index (Truth, Predicted) and columns [0, 1].

    Returns:
    pd.DataFrame: Reformatted dataframe with model evaluation metrics.
    """
    # Extracting the values from the confusion matrix
    tn, fp, fn, tp = df.iloc[0, 0], df.iloc[0, 1], df.iloc[1, 0], df.iloc[1, 1]
    acc = (tp + tn) / (tp + tn + fp + fn)
    prec = tp / (tp +fp)
    recall = tp / (tp + fn)
    f1 = 2 * ((prec * recall)/(prec + recall))
    # Creating a new dataframe with the desired format
    metrics_df = pd.DataFrame({
        "name": name,
        "tp": [tp],
        "tn": [tn],
        "fp": [fp],
        "fn": [fn],
        'acc': acc,
        'prec': prec,
        'recall': recall,
        'f1': f1
    })

    return metrics_df
```

In [52]:
```python
def format_results(df):
    df = np.where(df == 1, 'Donor','No Donor')
    return df
```

LOAD DATA

In [53]:
```python
df = pd.read_csv('./data/df.csv').drop('Unnamed: 0', axis=1)
```

In [54]:
```python
train = df[df['type'] == 'train'].drop('type',axis =1)
dev = df[df['type'] == 'dev'].drop('type',axis =1)
test = df[df['type'] == 'test'].drop('type',axis =1)
```

VIF

In [7]:
```python
dummies = pd.get_dummies(df, drop_first=True)

cat_cols = [
    'zipconvert2_Yes', 'zipconvert3_Yes', 'zipconvert4_Yes', 'boxcox_zipconvert5_Yes',
    'homeowner_Yes', 'female_Yes', 'type_train', 'type_dev', 'type_test'
]

cont_cols = [col for col in dummies.columns if col not in cat_cols + ['target']]
add_transformations(dummies, cont_cols)

kept, removed = remove_high_vif_features(X=dummies.drop('target_No Donor', axis=1), y=dummies['target_No Donor'], vif_threshol
print('REMOVED:', removed)
```

```
REMOVED: ['num_child', 'income', 'cos_target_No Donor', 'sin_target_No Donor', 'sigmoid_target_No Donor', 'boxcox_target_No Dono
r', 'inv_target_No Donor', 'sqrt_target_No Donor', 'sq_target_No Donor', 'cos_zipconvert5_Yes', 'sin_zipconvert5_Yes', 'sigmoid_z
ipconvert5_Yes', 'boxcox_zipconvert5_Yes', 'inv_zipconvert5_Yes', 'sqrt_zipconvert5_Yes', 'sq_zipconvert5_Yes', 'log_zipconvert5_
Yes', 'log_income', 'sq_income', 'log_num_child', 'sq_num_child', 'sigmoid_avg_fam_inc', 'inv_num_child', 'sqrt_months_since_dona
te', 'sqrt_med_fam_inc', 'sq_wealth', 'boxcox_avg_fam_inc', 'boxcox_num_prom', 'log_last_gift', 'months_since_donate', 'log_avg_g
ift', 'inv_home_value', 'inv_avg_fam_inc', 'boxcox_time_lag', 'log_wealth', 'inv_med_fam_inc', 'log_largest_gift', 'boxcox_home_v
alue', 'sqrt_num_prom', 'log_lifetime_gifts', 'sqrt_income', 'boxcox_pct_lt15k', 'sqrt_wealth', 'sq_months_since_donate', 'sqrt_a
vg_fam_inc', 'sqrt_avg_gift', 'boxcox_med_fam_inc', 'sqrt_time_lag', 'boxcox_largest_gift', 'sqrt_pct_lt15k', 'inv_income', 'sqrt
_home_value', 'sigmoid_num_child', 'log_num_prom', 'sqrt_last_gift', 'log_months_since_donate', 'boxcox_avg_gift', 'sqrt_lifetime
_gifts', 'avg_fam_inc', 'wealth', 'largest_gift', 'med_fam_inc', 'boxcox_last_gift', 'inv_pct_lt15k', 'zipconvert5_Yes', 'num_pro
m', 'pct_lt15k', 'home_value', 'log_avg_fam_inc', 'inv_wealth', 'log_time_lag', 'boxcox_lifetime_gifts', 'sqrt_num_child', 'avg_g
ift', 'inv_time_lag', 'log_med_fam_inc', 'boxcox_income', 'inv_largest_gift', 'inv_last_gift', 'sq_avg_fam_inc', 'inv_num_prom',
'last_gift']
```

In [55]:
```python
final_vars = list(kept.corr().drop('target')[np.abs(kept.corr()['target'].drop('target')) > .05].index)
```

In [56]:
```python
regress = kept[final_vars]
```

In [57]:
```
regress['target'] = (df['target'] == 'Donor').astype(int)
```

In [72]:
```
kept = kept.drop('log_target_No Donor', axis =1)
```

In [59]:
```
train = kept[regress['type_train'] ==1]
dev = kept[(regress['type_test'] == 0) & (regress['type_train'] == 0)]
test = kept[regress['type_test'] ==1]
```

In [60]:
```
for data in [train, dev, test]:
    data.drop('type_train', inplace = True, axis = 1)
    data.drop('type_test', inplace = True, axis = 1)
test = test.drop('target', axis =1)
```

Logistic Regression

In [61]:
```
results_df = pd.DataFrame()
```

In [62]:
```
# Selecting features and target variable for training data
X_train = train.drop(['target'], axis =1 )
y_train = train['target']
X_test = dev.drop(['target'], axis = 1)
y_test = dev['target']

# Fitting logistic regression model
glm = sm.GLM(y_train, X_train, family=sm.families.Binomial())
glm = glm.fit()

# Summarizing results
# print(results.summary())
```

In [63]:
```
log_preds = (glm.predict(X_test) >= 0.5).astype(int)
log_acc = accuracy_score(log_preds, y_test)
print(log_acc)

d = confusion_table(log_preds,y_test)
results_df = pd.concat([results_df,convert_confusion_matrix(d, 'Logistic Regression')])

log_test_preds = (glm.predict(test) >= 0.5).astype(int)
log_test_preds = format_results(log_test_preds)

save_df = pd.DataFrame(log_test_preds, columns=['values'])
save_df.to_csv('./preds/log.csv', index=False)
```

```
0.5116666666666667
```

LDA

In [64]:
```
lda = LDA(store_covariance=True)
lda.fit(X_train, y_train)

lda_preds = lda.predict(X_test)

lda_acc = accuracy_score(lda_preds,y_test)
print(lda_acc)

d = confusion_table(lda_preds,y_test)
results_df = pd.concat([results_df,convert_confusion_matrix(d, 'LDA')])


lda_test_preds = (lda.predict(test) >= 0.5).astype(int)
lda_test_preds = format_results(lda_test_preds)

save_df = pd.DataFrame(lda_test_preds, columns=['values'])
save_df.to_csv('./preds/lda.csv', index=False)
```

```
0.5066666666666667
```

QDA.

In [65]:

```
qda = QDA(store_covariance=True)
qda.fit(X_train, y_train)

qda_preds = qda.predict(X_test)

qda_acc = accuracy_score(qda_preds,y_test)

print(qda_acc)

d = confusion_table(qda_preds,y_test)
results_df = pd.concat([results_df,convert_confusion_matrix(d, 'QDA')])

qda_test_preds = (qda.predict(test) >= 0.5).astype(int)
qda_test_preds = format_results(qda_test_preds)

save_df = pd.DataFrame(qda_test_preds, columns=['values'])
save_df.to_csv('./preds/qda.csv', index=False)
```

```
0.49166666666666664
```

KNN

In [66]:

```
knn1 = KNeighborsClassifier(n_neighbors=1)
knn1.fit(X_train, y_train)
knn1_pred = knn1.predict(X_test)
knn1_acc = accuracy_score(knn1_pred,y_test)

print(knn1_acc)

d = confusion_table(knn1_pred, y_test)
results_df = pd.concat([results_df,convert_confusion_matrix(d, 'KNN')])

knn1_test_preds = (knn1.predict(test) >= 0.5).astype(int)
knn1_test_preds = format_results(knn1_test_preds)

save_df = pd.DataFrame(knn1_test_preds, columns=['values'])
save_df.to_csv('./preds/knn1.csv', index=False)
```

```
0.4816666666666667
```

NB

In [67]:

```
nb = GaussianNB()
nb.fit(X_train, y_train)
nb_preds = nb.predict(X_test)
nb_acc = accuracy_score(nb_preds,y_test)

print(nb_acc)
save_df = pd.DataFrame(nb_preds, columns=['values'])
save_df.to_csv('./preds/nb.csv', index=False)

d = confusion_table(nb_preds, y_test)
results_df = pd.concat([results_df,convert_confusion_matrix(d, 'Naïve Bayes')])

nb_test_preds = (nb.predict(test) >= 0.5).astype(int)
nb_test_preds = format_results(nb_test_preds)

save_df = pd.DataFrame(nb_test_preds, columns=['values'])
save_df.to_csv('./preds/nb.csv', index=False)
```

```
0.495
```

In [68]:

```
results_df
```

Out[68]:

	name	tp	tn	fp	fn	acc	prec	recall	f1
0	Logistic Regression	159	148	153	140	0.511667	0.509615	0.531773	0.520458
0	LDA	156	148	156	140	0.506667	0.500000	0.527027	0.513158
0	QDA	26	269	286	19	0.491667	0.083333	0.577778	0.145658
0	KNN	140	149	172	139	0.481667	0.448718	0.501792	0.473773
0	Naïve Bayes	18	279	294	9	0.495000	0.057692	0.666667	0.106195

In [69]:

```
# test_acc = {
#     'log': 0.5333333,
#     'lda': 0.5583333,
#     'qda': 0.525,
#     'knn':  0.475,
#     'nb': 0.5166667,
# }
```

In [70]:

```
# results_df['test_acc'] = test_acc.values()
```

In [71]:

```
results_df
```

Out[71]:

	name	tp	tn	fp	fn	acc	prec	recall	f1	test_acc
0	Logistic Regression	159	148	153	140	0.511667	0.509615	0.531773	0.520458	0.533333
0	LDA	156	148	156	140	0.506667	0.500000	0.527027	0.513158	0.558333
0	QDA	26	269	286	19	0.491667	0.083333	0.577778	0.145658	0.525000
0	KNN	140	149	172	139	0.481667	0.448718	0.501792	0.473773	0.475000
0	Naïve Bayes	18	279	294	9	0.495000	0.057692	0.666667	0.106195	0.516667

FUNDRAISING/best_fundraising_cross_val.py

```python
# ============================================================================
# SET-UP AND IMPORTS
# ============================================================================
import os
import torch
import torch.nn as nn
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from torch.utils.data import DataLoader, TensorDataset, ConcatDataset
from sklearn.model_selection import KFold
from sklearn.preprocessing import StandardScaler

# Set directory and device setup
os.chdir('/home/dan/FUNDRAISING')
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
print(f"Using device: {device}")

# ============================================================================
# PREPARE DATA
# ============================================================================
df = pd.read_csv('./data/df.csv', index_col='Unnamed: 0')
df['zipconvert'] = df[['zipconvert2', 'zipconvert3', 'zipconvert4', 'zipconvert5']].apply(
    lambda row: 'zc' + str(row.idxmax()[-1]) if pd.notna(row.idxmax()) else 'unknown', axis=1
)
test = df[df['type']=='test']

df = df[df['type']!='test']

df.drop(['zipconvert2', 'zipconvert3', 'zipconvert4', 'zipconvert5', 'type'], axis=1, inplace=True)
test.drop(['zipconvert2', 'zipconvert3', 'zipconvert4', 'zipconvert5', 'type'], axis=1, inplace=True)

cat_cols = ['homeowner', 'female', 'zipconvert', 'wealth', 'income', 'num_child']
cont_cols = [col for col in df.columns if col not in cat_cols + ['target']]
emb_sizes = [(df[col].astype('category').cat.codes.max() + 1, min(50, (df[col].nunique() + 1) // 2)) for col in cat_cols]

# Function to process datasets
def process_data(data):
    cats = np.stack([data[col].astype('category').cat.codes.values for col in cat_cols], axis=1)
    conts = np.stack([data[col].values for col in cont_cols], axis=1)
    scaler = StandardScaler()

    conts = scaler.fit_transform(conts)
    y = data['target'].map({'Donor': 1, 'No Donor': 0}).values
    return torch.tensor(cats, dtype=torch.int64), torch.tensor(conts, dtype=torch.float), torch.tensor(y, dtype=torch.long)


cats, conts, targets = process_data(df)
test_cats, test_conts, test_targets = process_data(test)

dataset = TensorDataset(cats, conts, targets)

test_dataset = TensorDataset(test_cats, test_conts, test_targets)
test_loader = DataLoader(test_dataset, batch_size=2048, shuffle=False)

# ============================================================================
# DEFINE A TABULAR MODEL
# ============================================================================
class TabularModel(nn.Module):
    def __init__(self, emb_sizes, n_cont, out_sz, layers, p=0.5):
        super().__init__()
        self.embeds = nn.ModuleList([nn.Embedding(ni, nf) for ni, nf in emb_sizes])
        self.emb_drop = nn.Dropout(p)
        self.bn_cont = nn.BatchNorm1d(n_cont)

        # Calculate the total embedding output size
        total_emb_size = sum(nf for _, nf in emb_sizes)
        n_in = total_emb_size + n_cont  # Total input size for the first linear layer

        layerlist = nn.ModuleList()
        for output_size in layers:
            layerlist.append(nn.Linear(n_in, output_size))
            layerlist.append(nn.ReLU())
            layerlist.append(nn.BatchNorm1d(output_size))
            layerlist.append(nn.Dropout(p))
            n_in = output_size  # Update n_in to the output size of the current layer

        layerlist.append(nn.Linear(layers[-1], out_sz))  # Final output layer
        self.layers = nn.Sequential(*layerlist)

    def forward(self, x_cat, x_cont):
        embeddings = [e(x_cat[:, i]) for i, e in enumerate(self.embeds)]
        x = torch.cat(embeddings, 1)
        x = self.emb_drop(x)
```

```python
        x_cont = self.bn_cont(x_cont)
        x = torch.cat([x, x_cont], 1)
        return self.layers(x)

# Initialize model
model = TabularModel(emb_sizes, len(cont_cols), 2, [200, 100], p=0.4)
model = model.to(device)


#endregion
#region # EARLY STOPPING
# ============================================================================
# EARLY STOPPING
# ============================================================================
class EarlyStopping:
    def __init__(self, patience=5, verbose=False, delta=0):
        self.patience = patience
        self.verbose = verbose
        self.delta = delta
        self.best_score = None
        self.early_stop = False
        self.counter = 0

    def __call__(self, val_loss, model):
        score = -val_loss

        if self.best_score is None:
            self.best_score = score
        elif score < self.best_score + self.delta:
            self.counter += 1
            if self.verbose:
                print(f'EarlyStopping counter: {self.counter} out of {self.patience}')
            if self.counter >= self.patience:
                self.early_stop = True
        else:
            self.best_score = score
            self.counter = 0


# ============================================================================
# CROSS-VALIDATION SETUP
# ============================================================================
def calculate_accuracy(y_pred, y_true):
    y_pred_classes = torch.argmax(y_pred, dim=1)
    correct = (y_pred_classes == y_true).float()  # convert into float for division
    acc = correct.sum() / len(correct)
    return acc

kf = KFold(n_splits=20, shuffle=True, random_state=42)
early_stopping = EarlyStopping(patience=5, verbose=True)
results = []

for fold, (train_idx, val_idx) in enumerate(kf.split(dataset)):
    train_subsampler = torch.utils.data.SubsetRandomSampler(train_idx)
    val_subsampler = torch.utils.data.SubsetRandomSampler(val_idx)
    train_loader = DataLoader(dataset, batch_size=32, sampler=train_subsampler)
    val_loader = DataLoader(dataset, batch_size=32, sampler=val_subsampler)

    model = TabularModel(emb_sizes, len(cont_cols), 2, [100,200], p=0.4).to(device)
    criterion = nn.CrossEntropyLoss()
    optimizer = torch.optim.Adam(model.parameters(), lr=1e-5)

    for epoch in range(50):  # Adjust as needed
        model.train()
        total_loss, total_acc = 0, 0
        for cats, conts, y in train_loader:
            cats, conts, y = cats.to(device), conts.to(device), y.to(device)
            optimizer.zero_grad()
            outputs = model(cats, conts)
            loss = criterion(outputs, y)
            acc = calculate_accuracy(outputs, y)
            total_loss += loss.item()
            total_acc += acc.item()
            loss.backward()
            optimizer.step()
        avg_train_loss = total_loss / len(train_loader)
        avg_train_acc = total_acc / len(train_loader)

        model.eval()
        val_loss, val_acc = 0, 0
        with torch.no_grad():
            for cats, conts, y in val_loader:
                cats, conts, y = cats.to(device), conts.to(device), y.to(device)
                outputs = model(cats, conts)
                loss = criterion(outputs, y)
                acc = calculate_accuracy(outputs, y)
                val_loss += loss.item()
```

```
                val_acc += acc.item()
        avg_val_loss = val_loss / len(val_loader)
        avg_val_acc = val_acc / len(val_loader)

        print(f'Fold {fold+1}, Epoch {epoch+1}, Train Loss: {avg_train_loss:.4f}, Train Acc: {avg_train_acc:.4f}, Val Loss: {avg_val_loss:.4f}, Val Acc: {avg_val_acc:.4f}')

        # Call early stopping
        early_stopping(avg_val_loss, model)
        if early_stopping.early_stop:
            print("Early stopping")
            break

    results.append((avg_val_loss, avg_val_acc))

average_val_loss = sum(x[0] for x in results) / len(results)
average_val_acc = sum(x[1] for x in results) / len(results)
print(f'Average Validation Loss: {average_val_loss:.4f}, Average Validation Accuracy: {average_val_acc:.4f}')

preds = []
model.eval()

with torch.no_grad():
    for cats, conts, y in test_loader:
        cats, conts, y = cats.to(device), conts.to(device), y.to(device)
        output = model(cats, conts)
        predicted = output.argmax(dim=1)  # Ensure you use dim=1
        # print(predicted)
        preds.append(predicted.cpu())  # Move predictions to CPU
# print(type(preds[0]))
# Concatenate all batch predictions into a single tensor
# preds = torch.cat(preds)
preds = preds[0].tolist()
# print(preds)
final_preds = []
donor=1
no_donor=1
for i in preds:
    if i == 1:
        final_preds.append('Donor')
        donor +=1
    else:
        final_preds.append('No Donor')
        no_donor+=1
print(f'% Donor = {donor / (donor+no_donor)}')


save_df = pd.DataFrame(final_preds, columns=['values'])
save_df.to_csv('./preds/preds.csv', index=False)
```