Finance 6500

Assignment 1

Willem van der Schans

Outline

1. Target Variable
2. Data Cleaning
3. Web scraping
4. Feature Engineering
5. Machine Learning
6. Applet

Codebook

	Variable name	Data type	Description of variable
0	LoanNr_ChkDgt	Text	Identifier – Primary key
1	Name	Text	Borrower name
2	City	Text	Borrower city
3	State	Text	Borrower state
4	Zip	Text	Borrower zip code
5	Bank	Text	Bank name
6	BankState	Text	Bank state
7	NAICS	Text	North American industry classification system ...
8	ApprovalDate	Date/Time	Date SBA commitment issued
9	ApprovalFY	Text	Fiscal year of commitment
10	Term	Number	Loan term in months
11	NoEmp	Number	Number of business employees
12	NewExist	Text	1 = Existing business, 2 = New business
13	CreateJob	Number	Number of jobs created
14	RetainedJob	Number	Number of jobs retained
15	FranchiseCode	Text	Franchise code, (00000 or 00001) = No franchise
16	UrbanRural	Text	1 = Urban, 2 = rural, 0 = undefined
17	RevLineCr	Text	Revolving line of credit: Y = Yes, N = No
18	LowDoc	Text	LowDoc Loan Program: Y = Yes, N = No
19	ChgOffDate	Date/Time	The date when a loan is declared to be in default
20	DisbursementDate	Date/Time	Disbursement date
21	DisbursementGross	Currency	Amount disbursed
22	BalanceGross	Currency	Gross amount outstanding
23	MIS_Status	Text	Loan status charged off = CHGOFF, Paid in full...
24	ChgOffPrinGr	Currency	Charged-off amount
25	GrAppv	Currency	Gross amount of loan approved by bank
26	SBA_Appv	Currency	SBA's guaranteed amount of approved loan

1.

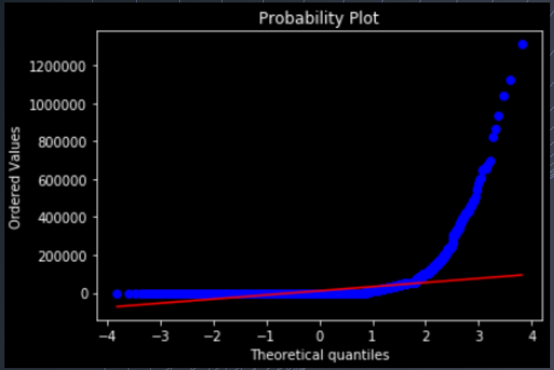
Target Variable

1.

Target variable

ChgOffPrinGr

Initially, I looked at ChgOffPrinGr as my target variable; however, due to the zero-inflated probability distribution (see plot) of the variable it was hard to work with. I therefore decided to focus on the categorical variables since the implication for the loan officer would be the same; give the loan or not.



MIS_Status

I chose not to try to predict MIS-Status since according to the codebook there were several faulty entries. The codebook only describes two possible inputs: CHGOFF and P I F. The data itself included a number of missing values (see excerpt). While I mode imputed all the missing entries, these missing values in made MIS_Status a less optimal predictor then ChgOffDate due to losing entries.

	Sum of Missing
ChgOffDate	8053
MIS_Status	21
DisbursementDate	16
Bank	6
BankState	6
Name	4
RevLineCr	1

1.

Target variable

ChgOffDate

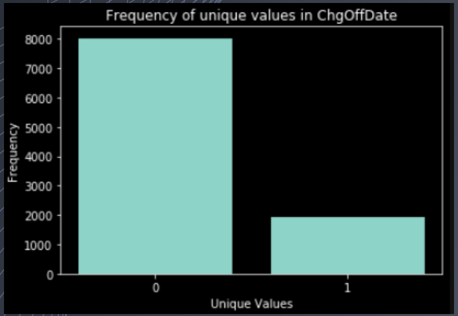
On the previous slide I said that ChgOffDate supposedly is a better predictor then MIS_Status due to the missing values in MIS_Status, while it seems that ChgOffDate has more missing values (see excerpt).

In this case missing values are not a problem however since the implication of them is clear: " A missing value means no default". Knowing this I changed the variable to a binominal categorical variable with 0 = no default and 1 = default. (see code).

The final frequencies of 0 and 1 can be seen on the plot to the right. 0 has a frequency of 8028 and 1 has a frequency of 1937, this gives us the majority class predictor accuracy = 80.56%. This 80.56% accuracy is the benchmark score to beat by the machine learning algorithms deployed.

	Sum of Missing
ChgOffDate	8053
MIS_Status	21
DisbursementDate	16
Bank	6
BankState	6
Name	4
RevLineCr	1

```
#Number 5
train['ChgOffDate'] = train['ChgOffDate'].fillna(0)

for i in train.index:
    if train.iloc[i,train.columns.get_loc('ChgOffDate')] != 0:
        train.at[i, 'ChgOffDate'] = 1
    else:
        train.at[i, 'ChgOffDate'] = 0
```



2.
Data Cleaning

2.
Data Cleaning

Considerations

I made all data cleaning decisions based on the codebook entries. If the data did not reflect what the codebook described, then it was considered an error and should therefore be fixed.

2.1 Removal of Variables

The first decision I made was removing all variables that could be used as identifiers by the algorithms. While this is more feature engineering then data cleaning I put it under this header since I made this decision here in my workflow. The removed Identifiers were: LoanNr_ChkDgt, Name, Zip, City and NAICs

I also removed BalanceGross as this variable only consists of 0's and therefore holds no information.

Lastly, I removed the precise data variables since I did not have enough time to test how the algorithms would interpret these. If I had more time I would test and reassess this decision. The variables affected are ApprovalDate and DisbursementDate

```
# Number 1
train.drop(columns=['LoanNr_ChkDgt', 'Name'], inplace=True)

# Number 2
train.dropna(subset=['NAICS'], inplace=True)

# Number 6
train.dropna(subset=['Bank'], inplace=True)
train.dropna(subset=['BankState'], inplace=True)
train.dropna(subset=['DisbursementDate'], inplace=True)

# Number 7
train.drop(columns=['BalanceGross'], inplace=True)

# Number 8
train.dropna(subset=['MIS_Status'], inplace=True)

#Number 9
train.drop(columns=['City'], inplace=True)
train.drop(columns=['Zip'], inplace=True)

#Remove Specific Dates?
train.drop(columns=['ApprovalDate'], inplace=True)
train.drop(columns=['DisbursementDate'], inplace=True)
train.drop(columns=['NAICS'], inplace=True)
```

2.

Data Cleaning

Data imputation

First off, there is RevLineCr which should only contain Y’s and N’s but also includes 0, T, ‘ and `. I changed all the wrong entries to no. If I would continue on this project I would most likely take this variable out, since the assumption made here is extremely bold.

Then there is LowDoc ; this variable should only include N and Y however also includes a C. This C got mode imputed to a N.

Lastly, there is Franchise Code which shows if a business is a franchise and if so, what Franchise code they hold. While I thought this information was useful I wanted to remove the identifiable codes from the data set. I therefore changed all 1’s and 0’s to 0 (no franchise) and the other entries to 1 (franchise)

```
N    4344
Y    3546
0    1688
T     419
,       1
`       1
Name: RevLineCr, dtype: int64
N    6454
Y    3546
Name: RevLineCr, dtype: int64
```

```
N     9046
Y      953
C        1
Name: LowDoc, dtype: int64
N     9047
Y      953
Name: LowDoc, dtype: int64
```

```
1        9434
0         125
78760      23
50564      16
68020      14
         ...
8047        1
8015        1
9800        1
30210       1
75985       1
Name: FranchiseCode, Length: 250, dtype: int64
0    9559
1     441
Name: FranchiseCode, dtype: int64
```

3.

Web Scraping

3.

Web scraping

Intention

For the web scraping part of the assignment, I wanted to find data that added information to the data set that would help the algorithms predict loan default.

The data I collected was past recession data and came from the St. Louis federal reserve. I created a dictionary with the keys being years and the values a binominal variable that took 0 for no-recession and 1 for recession.

I then created a new data column (recession) based on ApprovedFY and renamed the years in this new column to the values in the dictionary. A code excerpt can be seen on the right.

```
recession.rename(columns = {"JHDUSRGDPBR" : "Recession", "DATE" : "Date"}, inplace=True)

recession['Date'] = pd.to_datetime(recession['Date'])
recession['Recession'] = recession['Recession'].astype('int64').astype(dtype="category")

recession['Year'] = recession['Date'].dt.year
```

```
recession = recession.reset_index()
recession = recession.pivot(index='index', columns='Year', values='Recession')
recession = recession.mode(axis = 0, dropna=True)
recession = recession.drop(index=recession.index.difference([0]))

keys = tuple(recession.columns)
values = tuple(recession.to_numpy()[0])

rename_dict = dict(zip(keys, values))
```

```
alldata['Recession'] = alldata['ApprovalFY']
alldata['Recession'] = alldata['Recession'].replace(rename_dict)
alldata['Recession'].fillna(0, inplace=True)

print(alldata['Recession'].value_counts())
```

```
0    10830
1      133
Name: Recession, dtype: int64
```

4.

Feature Engineering

4.

Feature Engineering

Label Encoding

I label encoded Bank since one-hot encoding this variable created a data frame of over 5000 columns. The performance impact such a data frame has on the code execution made me decide to opt for label encoding for this variable.

```
from sklearn.preprocessing import LabelEncoder

lbl = LabelEncoder()
lbl.fit(list(alldata['Bank'].values))
alldata['Bank'] = lbl.transform(list(alldata['Bank'].values))

# shape
print(format(alldata.shape))
```

```
(10963, 17)
```

Skewness and Box-Cox Transformations

I box-cox transformed all variables with a skew higher than 0.75 (see the table to the right). I used lambda 0.15 as this seemed to be the most used value and produced the results I wanted.

	Skew
NoEmp	70.872035
RetainedJob	16.636037
CreateJob	14.703799
Recession	8.912960
SBA_Appv	5.031105
DisbursementGross	4.699813
GrAppv	4.611168
FranchiseCode	4.496963
Term	1.786472
NewExist	0.723012
Bank	0.417791
UrbanRural	0.261263
ApprovalFY	-1.995248

One-Hot Encoding

Any leftover categorical variables at this point were one-hot encoded. The variables affected were 'State', 'BankState', 'RevLineCr', 'LowDoc'. The resulting all-data data frame had dimensions: (10963, 120)

5.

Machine Learning

5. Machine Learning

Target Measure

Since I am running classification algorithms my target measure was the accuracy of the algorithms. This measure also allows me to easily compare the score to the benchmark of the majority class predictor at 80.56%

Evaluations

I decided to do both cross-validation and hold-out evaluation. This to make sure that the results correspond. Using both evaluations allows for more insight and also shows errors in, for example, the cross-validation function, which was self-written.

I did my cross-validation with 5 folds and my hold-out evaluation with a 0.85 – 0.15 split, using random seed(123)

```
from sklearn.model_selection import train_test_split

HO_X_train1, HO_X_test1, HO_y_train_CL, HO_y_test_CL = train_test_split(train, y_ChgOffDate, test_size=0.15)

HO_X_train2, HO_X_test2, HO_y_train_CO, HO_y_test_CO = train_test_split(train, y_ChgOffPrinGr, test_size=0.15)
```

```
#Validation function
n_folds = 5

def acc_cv(model):
    kf = KFold(n_folds, shuffle=True, random_state=42).get_n_splits(train.values)
    acc= cross_val_score(model, train.values, y_train, scoring='accuracy', cv = kf)
    return(acc)
```

5.

Machine Learning

Algorithms

I decided to run 4 classification algorithms, after which I could evaluate each one and select one or two for the final predictions. The algorithms are Random Forest Classification, Support Vector Classification, K Neighbors Classifier, and XGBoost.

Hyperparameters

I wanted my algorithms to be conservative to combat overfitting on the data set. Hyperparameter settings are shown in the excerpt on the right. Conservative settings can be seen in KNN with 5 Neighbors and XGB with a learning rate of 0.1 (default=0.3). Added parameters are settings for parallel processing where possible, which dramatically lowered the run time of the Notebook with a final time of 101 seconds.

```
from sklearn.model_selection import KFold, cross_val_score, train_test_split
from sklearn.metrics import mean_squared_error

# XGBoost
import xgboost as xgb

# KNN
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import KFold

#Random Forest
from sklearn.ensemble import RandomForestClassifier

#Support Vector Classification.
from sklearn.svm import SVC

# Calculate RMSE
from sklearn.metrics import mean_squared_error
from math import sqrt

#Create Confusion Matrix
from sklearn.metrics import confusion_matrix, accuracy_score
```

```
model_SVC = SVC(cache_size = 8000)

model_KNN = KNeighborsClassifier(n_neighbors=5, weights='distance',algorithm = 'auto', n_jobs = -1)

model_RFC = RandomForestClassifier(n_jobs = -1)


model_xgb = xgb.XGBClassifier(colsample_bytree=0.5, gamma=0.5,
                             learning_rate=0.1, max_depth=6,
                             min_child_weight=1.5, n_estimators=2200,
                             reg_alpha=0.25, reg_lambda=0.5,
                             subsample=0.5, silent=1,
                             random_state =7, nthread = -1)
```

5.

Machine Learning

Evaluation Code

The code I used to evaluate all models is shown below. This code outputs a cross-validation accuracy score, an in-sample old-out score, and an out-of-sample hold-out score. An example output can be seen on the right as well.

```
Random Forest Cross-Val Accuracy score: 0.9390 (0.0249)


########In-Sample Hold-Out########
Predicted     0     1   All
Actual
0          6822     0  6822
1             0  1648  1648
All        6822  1648  8470
In-Sample Hold-Out Accuracy =  1.0


########Out-Of-Sample Hold-Out########
Predicted     0    1   All
Actual
0          1190   16  1206
1            72  217   289
All        1262  233  1495
Out-Of-Sample Hold-Out Accuracy =  0.9411371237458194
```

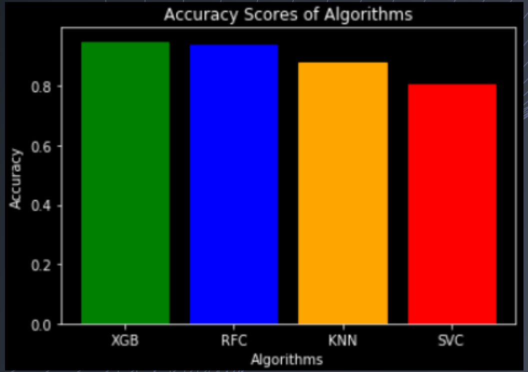
```
score_RFC = acc_cv(model_RFC)
print("Random Forest Cross-Val Accuracy score: {:.4f} ({:.4f})\n".format(score_RFC.mean(), score_RFC.std()))

model_RFC.fit(HO_X_train1, HO_y_train_CL) #Hold-Out

RFC_train_pred = model_RFC.predict(HO_X_train1) #InSample Holdout RMSE

RFC_pred_HO = model_RFC.predict(HO_X_test1) #Out of Sample Holdout RMSE
print("\n")
print("########In-Sample Hold-Out########")
print(pd.crosstab(HO_y_train_CL, RFC_train_pred, rownames=['Actual'], colnames=['Predicted'], margins=True))
print("In-Sample Hold-Out Accuracy = ",accuracy_score(HO_y_train_CL, RFC_train_pred, normalize=True, sample_weight=None))
print("\n")
print("########Out-Of-Sample Hold-Out########")
print(pd.crosstab(HO_y_test_CL, RFC_pred_HO, rownames=['Actual'], colnames=['Predicted'], margins=True))
print("Out-Of-Sample Hold-Out Accuracy = ",accuracy_score(HO_y_test_CL, RFC_pred_HO, normalize=True, sample_weight=None))
```

5.

Machine Learning

Outcome

To the right, the plotted cross-validation scores per algorithm can be seen. As is evident, Random Forest Classification and XGBoost have the best cross-validation accuracy scores of 93.8% and 94.9% respectively. Below the outputs can be seen of the hold-out evaluation. I chose to predict with both of these algorithms in the Applet since they are both sufficiently accurate.




```
Xgboost Accuracy score: 0.9497 (0.0164)


########In-Sample Hold-Out########
Predicted     0     1   All
Actual
0          6819     3  6822
1             8  1640  1648
All        6827  1643  8470
Out-Of-Sample Hold-Out Accuracy =  0.9987012987012988


########Out-Of-Sample Hold-Out########
Predicted     0    1   All
Actual
0          1184   22  1206
1            50  239   289
All        1234  261  1495
Out-Of-Sample Hold-Out Accuracy =  0.9518394648829431
```

```
Random Forest Cross-Val Accuracy score: 0.9390 (0.0249)


########In-Sample Hold-Out########
Predicted     0     1   All
Actual
0          6822     0  6822
1             0  1648  1648
All        6822  1648  8470
In-Sample Hold-Out Accuracy =  1.0


########Out-Of-Sample Hold-Out########
Predicted     0    1   All
Actual
0          1190   16  1206
1            72  217   289
All        1262  233  1495
Out-Of-Sample Hold-Out Accuracy =  0.9411371237458194
```

6.

Applet

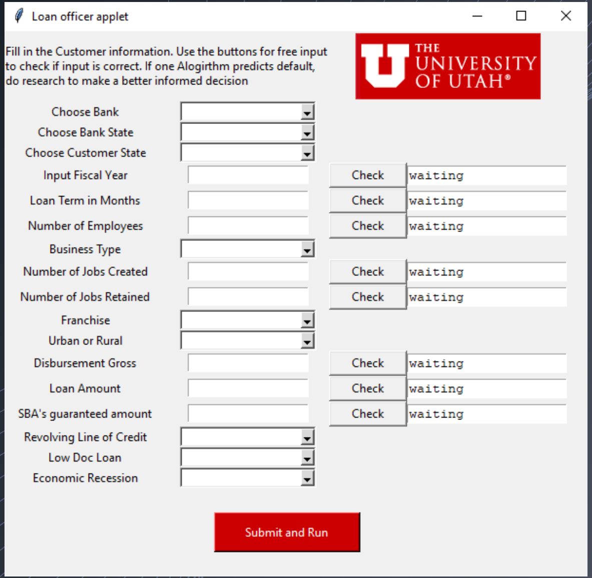
6. Applet

Objective

The objective of the applet is to allow loan officers to qualify a business for a potential loan. I set a personal objective for myself, namely that the applet has to be standalone from the machine learning code. This made the applet code longer (657 lines) however, now it can be used on any computer without having to run trough the Machine learning code first to train models. I also uploaded all data to GitHub to be able to pull data directly from the internet further allowing the app to be standalone from any environment. The applet also incorporates the web scraping column Recession.

Use

The applet is a tool and therefore has an advisory role in the process it, therefore, outputs only objective results and no implication, the means possible output is as follows: e.g., “No Default predicted” and “Only Random Forest predicts Default.”



6. Applet

Functions

What the applet does run all of the relevant machine learning and web scraping code in a large function when the submit button has been clicked. The applet downloads, cleans, feature engineers, and submits both train and test data sets to both XGBoost and RFC algorithms and which then individually predict the outcome of the test data set created from user input. An example output is shown on the right.

Additional Features;

1. A data dump to csv of the last input made by the user so the data can be stored.
2. Free -submission check buttons. Small functions that check if the data entered is an integer and has the right format.