Lending Club Assignment

Hung Hoang Nguyen

Minerva Schools at KGI

CS156 - Fall 2020

For this assignment, I chose variables that are included in both the rejected and accepted dataset. Because the rejected dataset has much fewer variables, I had to drop a lot of variables from the accepted dataset so that both datasets are consistent with each other prior to merging. These variables, among with their preprocessing steps, are:

1. **`loan_amnt`**: The loan amount people applied for.
2. **`risk_score`**: The credit scores of applicants.
3. **`dti`**: Debt-to-income ratio of each applicant.
4. **`addr_state`**: The state provided by the applicant. Because this is a non-ordinal categorical variable, I made dummy boolean columns for each unique value in the original dataset.
5. **`emp_length`**: Length of employment for each applicant. Because this is an ordinal categorical variable, I mapped each level to a number, from 0 (<1 year) to 10 (10+ years).
6. **`policy_code`**: Given by the dataset.

The dependent variable is **`funded_amnt`**. People in the accepted dataset already have this property, but for those in the rejected dataset, I assumed that they were all funded with an amount of $0.

The rejected dataset has a lot of NaN values, especially in the **`risk_score`** column, which I decided to deal with by removing rows with these entries, to make the analysis simpler. However, by doing so, I potentially removed a nuance: the fact that people do not have a credit score (N/A for that field) can help predict how much money they can apply for. My hypothesis is that not having a credit score is even worse than having a bad one. This analysis, however, is beyond what I have chosen to do for this assignment. Additionally, prior to merging, I made sure that the rejected and accepted portion of the dataset are equal to each other.

I used Ridge regression for this assignment so that the predictions are not sensitive to changes in the dataset. To prevent overfitting, I split the merged dataset by 70/30, 70% for training, and 30% for testing. To choose the best lambda for the Ridge class, I used cross-validation to try out various settings and picked the best one. I used R^2, RMSE, and MAE to quantify how well the predictions fit the test data, after training the model. I chose RMSE to see how well Ridge regression removed sensitivity to the independent variables, and MAE because it's contextually relevant - it would be useful to know how off we are in terms of actual dollar value. Even though the model was able to explain a large portion of variance for the dependent variable (0.74), the errors were still too large, compared to the practical scale of the dependent variable. RMSE and MAE were around 5000 and 4000, respectively, meaning our predictions are around at least $4000 off, on average.

A potential reason for this error is the fact that people in the rejected dataset get $0 in funding, but the model still predicts a dollar value, because that's what regression does. One fix is that we could do a hybrid model, first pass an application through classification to see if it is rejected and accepted, then proceed with regression to get a predicted funding amount. For those who are rejected, we automatically return 0. A problem with this approach, however, is that it would not be very useful for people with weak financial profiles. Instead of providing them with an amount that they would be approved for, no matter how low that is, they would get $0 by default (perhaps in reality it is $0, but that seems unlikely).

In [1]:
```
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
import random


from sklearn.model_selection import train_test_split
from sklearn.linear_model import RidgeCV
from sklearn.compose import make_column_transformer
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import mean_squared_error, mean_absolute_error
```

In [2]:
```
# I'm using Colab connected to my Google Drive, so change the path accordingly if you want to run this on loc
al

random.seed(42)

# Import
accepted_full = pd.read_csv("./drive/My Drive/CS156/assignment-2/accepted_2007_to_2018Q4.csv")
rejected_full = pd.read_csv("./drive/My Drive/CS156/assignment-2/rejected_2007_to_2018Q4.csv")
```

```
/usr/local/lib/python3.6/dist-packages/IPython/core/interactiveshell.py:2718: DtypeWarning: Columns (0,19,49,
59,118,129,130,131,134,135,136,139,145,146,147) have mixed types.Specify dtype option on import or set low_me
mory=False.
  interactivity=interactivity, compiler=compiler, result=result)
```

Mapping Rejected -> Accepted

- Policy Code -> policy_code
- Employment Length -> emp_length: turn this into numbers because it's in strings right now
- State -> addr_state
- Zip Code -> zip_code
- Debt-To-Income -> dti
- Risk_Score -> last_fico_range_high and last_fico_range_low
- Application Date -> Not available
- Amount Requested -> loan_amnt

In [3]:
```
# Data preprocessing

# Step 1: Work with Rejected dataset

# Choose 150000 at random (seeded)
rejected = rejected_full.sample(150000, random_state=42)

# Rename columns
rejected = rejected.drop(labels=["Application Date", "Loan Title", "Zip Code"], axis=1)
rejected_old_cols = list(rejected.columns)
new_col = ["loan_amnt", "risk_score", "dti", "addr_state", "emp_length", "policy_code"]
new_col_mapping = {rejected_old_cols[i]: new_col[i] for i in range(len(new_col))}
rejected = rejected.rename(columns=new_col_mapping)

# Remove rows with nan values
print("Before", rejected.shape)
rejected = rejected.dropna(axis=0)  # Lost a lot of data
print("After", rejected.shape)
rejected.head()

# Convert dti row to float instead of string
def transform_dti_val(val):
    return round(float(val[:-1])/100, 2)

rejected["dti"] = [transform_dti_val(val) for val in list(rejected["dti"])]

# Add funded_amnt column, all 0
rejected["funded_amnt"] = [0.0] * rejected.shape[0]

# Preview
rejected.head()
```

```
Before (150000, 6)
After (49169, 6)
```

Out[3]:

	loan_amnt	risk_score	dti	addr_state	emp_length	policy_code	funded_amnt
26499034	7000.0	571.0	0.07	OK	< 1 year	0.0	0.0
9963011	20000.0	626.0	0.32	FL	< 1 year	0.0	0.0
7457258	15000.0	673.0	0.34	NH	< 1 year	0.0	0.0
3751745	10000.0	665.0	0.25	PA	< 1 year	0.0	0.0
14407407	2000.0	601.0	0.01	AZ	< 1 year	0.0	0.0

In [4]:
```
# Step 2: Work with accepted dataset

# Choose 150000 at random (seeded)
accepted = accepted_full.sample(150000, random_state=42)

# Select relevant columns
accepted = accepted[["loan_amnt", "last_fico_range_high", "last_fico_range_low", "dti", "addr_state", "emp_le
ngth", "policy_code", "funded_amnt"]]

# Get risk_score for accepted
accepted["risk_score"] = (accepted["last_fico_range_high"] + accepted["last_fico_range_low"]) / 2
accepted = accepted[["loan_amnt", "funded_amnt", "risk_score", "dti", "addr_state", "emp_length", "policy_cod
e"]]

# Remove rows with nan
print("Before", accepted.shape)
accepted = accepted.dropna(axis=0)  # Lost ~ 10,000 data points, around 6% of the data
print("After", accepted.shape)
accepted.head()

# Turn dti into ratios instead of percentages
accepted["dti"] = [val/100 for val in list(accepted["dti"])]

# Preview
accepted.head()
```

```
Before (150000, 7)
After (140263, 7)
```

Out[4]:

	loan_amnt	funded_amnt	risk_score	dti	addr_state	emp_length	policy_code
392949	32000.0	32000.0	792.0	0.2405	CA	10+ years	1.0
324024	4000.0	4000.0	622.0	0.1953	FL	4 years	1.0
2066630	6025.0	6025.0	742.0	0.0916	MA	10+ years	1.0
477199	25000.0	25000.0	657.0	0.3626	CA	10+ years	1.0
1975547	20000.0	20000.0	807.0	0.1643	NV	10+ years	1.0

In [5]:
```
# Step 3: Make sure the two dataset is balanced before merging

# Because rejected is smaller than accepted, randomly sample the same number of rows from accepted
accepted = accepted.sample(n=rejected.shape[0], random_state=42)
```

In [6]:
```
# Step 4: Merge the two datasets and do some encoding for the categorical variables

df = accepted.append(rejected)

# Shuffle the data
df = df.sample(frac = 1, random_state=42)

# Get dummy data for states
df = pd.get_dummies(df, prefix="state_is", columns=["addr_state"])

# Get ordinal encoding for emp_length
mapping = {(str(i) + " years"):i for i in range(2, 10)}
mapping['1 year']  = 1
mapping['< 1 year'] = 0
mapping['10+ years'] = 10
df["emp_length"] = [mapping[val] for val in list(df["emp_length"])]


# Select features and the output variable
X = df.drop("funded_amnt", axis=1)
y = df["funded_amnt"]
```

In [7]:
```
# Preview
df.head()
```

Out[7]:

	loan_amnt	funded_amnt	risk_score	dti	emp_length	policy_code	state_is_AK	state_is_AL	state_is_AR	state_is_AZ	s
8364469	2000.0	0.0	574.0	0.1700	0	0.0	0	0	0	0	
26279286	20000.0	0.0	626.0	0.3300	0	0.0	0	0	0	0	
1358999	30000.0	30000.0	747.0	0.1233	8	1.0	0	0	0	0	
244138	25000.0	25000.0	737.0	0.0855	9	1.0	0	0	0	0	
999257	15325.0	15325.0	617.0	0.2234	3	1.0	0	0	0	0	

In [8]:
```
# Split the data into training set and testing set
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)
```

In [9]:
```
high = 1000
regr = RidgeCV(store_cv_values=True, alphas=[x for x in range(1, high)])
regr.fit(X_train, y_train)

# See what the best alpha and score is using CV and Ridge Regression
print("Best Alpha", regr.alpha_)
print("Training set R^2", regr.score(X_train, y_train))
```

```
Best Alpha 96
Training set R^2 0.7486664186629699
```

In [10]:
```
# Print results for the test set

y_pred = regr.predict(X_test)
print("R^2", regr.score(X_test, y_test))
print("RMSE", (mean_squared_error(y_pred, y_test)**(1/2)))
print("MAE", mean_absolute_error(y_pred, y_test)) # Off by an average amount of 4000 dollars
```

```
R^2 0.7492526004307951
RMSE 5084.437634688042
MAE 4046.7810330685975
```