In [1]:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
import warnings
warnings.filterwarnings('ignore')

X = pd.read_csv('loan_stat542.csv')
X['Y'] = np.where(X.loan_status=='Fully Paid',0,1)
X.drop('loan_status',axis=1,inplace= True)
# X.drop('id',axis=1,inplace=True)
X['term'] = X['term'].apply(lambda s: np.int8(s.split()[0]))
X.drop('grade', axis=1, inplace=True)
X.drop(labels='emp_title', axis=1, inplace=True)
X['emp_length'].replace(to_replace='10+ years', value='10 years', inplace=True)
X['emp_length'].replace('< 1 year', '0 years', inplace=True)
def emp_length_to_int(s):
    if pd.isnull(s):
        return s
    else:
        return np.int8(s.split()[0])
X['emp_length'] = X['emp_length'].apply(emp_length_to_int)
X['home_ownership'].replace(['NONE', 'ANY'], 'OTHER', inplace=True)
X['log_annual_inc'] = X['annual_inc'].apply(lambda x: np.log10(x+1))
X.drop('annual_inc', axis=1, inplace=True)
X.drop('title', axis=1, inplace=True)
X.drop(labels='zip_code', axis=1, inplace=True)
X['earliest_cr_line'] = X['earliest_cr_line'].apply(lambda s: int(s[-4:]))
X['fico_score'] = 0.5*X['fico_range_low'] + 0.5*X['fico_range_high']
X.drop(['fico_range_high', 'fico_range_low'], axis=1, inplace=True)
X['log_revol_bal'] = X['revol_bal'].apply(lambda x: np.log10(x+1))
X.drop('revol_bal', axis=1, inplace=True)
X = pd.get_dummies(X,
                   columns=['sub_grade', 'home_ownership', 'verification_status', 'purpose', 'addr_stat
e', 'initial_list_status', 'application_type'],
                   drop_first=True)
```

Data Processing

First, I append the train and test data and create response variable 'Y' by assigning 'default' to 1 and others to 0. I changed the 'term' to the number of months, dropped the 'grade' since it is a subset of 'sub-grade'. 'emp_title' is dropped too as the employment doesn't provide meaningful information. Changed the 'emp_length' to the number of years. Grouped the 'NONE','ANY' into 'OTHER' for 'home_ownership' variable. Took log of 'annual_inc' because the large range of this variable. Also dropped the 'title' as 'purpose' already provide the information. Kept the years of 'earliest_cr_line'. Created a new variable by taking the mean of 'fico_range_high' and 'fico_range_low'. Took log for 'revol_bal' as the data is skewed. Finally, called get_dummies to convert categorial variable to indicator variables.

In [2]:

```
# first split
test_id = pd.read_csv('Project3_test_id.csv')
train = X[~X.id.isin(test_id.test1)]
test = X[X.id.isin(test_id.test1)]
y_train = train['Y']
y_test = test['Y']
X_train = train.drop(columns = ['Y','id'],axis=1)
X_test = test.drop(columns = ['Y','id'],axis=1)
train1 = X[~X.id.isin(test_id.test2)]
test1 = X[X.id.isin(test_id.test2)]
y_train1 = train1['Y']
y_test1 = test1['Y']
X_train1 = train1.drop(columns = ['Y','id'],axis=1)
X_test1 = test1.drop(columns = ['Y','id'],axis=1)
train2 = X[~X.id.isin(test_id.test3)]
test2 = X[X.id.isin(test_id.test3)]
y_train2 = train2['Y']
y_test2 = test2['Y']
X_train2 = train2.drop(columns = ['Y','id'],axis=1)
X_test2 = test2.drop(columns = ['Y','id'],axis=1)
```

In [3]:

```
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler
from sklearn.preprocessing import Imputer
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
from sklearn.model_selection import GridSearchCV
from sklearn.linear_model import SGDClassifier
from sklearn.metrics import log_loss
def model_logreg(train_x,train_y,test_x,test_y):
    pipeline_sgdlogreg = Pipeline([
        ('imputer', Imputer(copy=False)), # Mean imputation by default
        ('scaler', StandardScaler(copy=False)),
        ('model', SGDClassifier(loss='log', max_iter=1000, tol=1e-3, random_state=1, warm_start=True))
    ])
    param_grid_sgdlogreg = {
        'model__alpha': [10**1],
        'model__penalty': ['l1']
    }
    grid_sgd= GridSearchCV(estimator=pipeline_sgdlogreg, param_grid=param_grid_sgdlogreg, scoring='roc_
auc', n_jobs=1, pre_dispatch=1, cv=5, verbose=1, return_train_score=False)
    grid_sgd.fit(train_x, train_y)
    y_prob = grid_sgd.predict_proba(test_x)
    return log_loss(test_y, y_prob)
```

I use SGD logistice regression for the first model. The model penalty 'l1' and 5 folds are used for each train and test split. The total running time for 3 tests are 90 seconds.

In [4]:

```
avg = np.zeros((3,3))
avg[0,0] = model_logreg(X_train,y_train,X_test,y_test)
avg[0,1] = model_logreg(X_train1,y_train1,X_test1,y_test1)
avg[0,2] = model_logreg(X_train2,y_train2,X_test2,y_test2)
```

```
Fitting 5 folds for each of 1 candidates, totalling 5 fits
[Parallel(n_jobs=1)]: Done   5 out of   5 | elapsed:   39.0s finished
Fitting 5 folds for each of 1 candidates, totalling 5 fits
[Parallel(n_jobs=1)]: Done   5 out of   5 | elapsed:   29.6s finished
Fitting 5 folds for each of 1 candidates, totalling 5 fits
[Parallel(n_jobs=1)]: Done   5 out of   5 | elapsed:   29.3s finished
```

In [7]:

```
import lightgbm as lgb
from sklearn.ensemble import RandomForestClassifier
def model_lgb(train_x,train_y,test_x,test_y):
    pipeline_xgb = Pipeline([
    ('imputer', Imputer(copy=False)), # Mean imputation by default
    ('scaler', StandardScaler(copy=False)),
    ('model', lgb.LGBMClassifier(silent=False,random_state=1))])

    param_xgb = {
        'model__max_depth': [50],
        'model__learning_rate': [0.1],
        'model__n_estimators': [200]
    }
    grid_xgb = GridSearchCV(estimator=pipeline_xgb, param_grid=param_xgb, scoring='roc_auc', n_jobs=1,
pre_dispatch=1, cv=3, verbose=1, return_train_score=False)
    grid_xgb.fit(train_x, train_y)
    y_prob = grid_xgb.predict_proba(test_x)
    return log_loss(test_y, y_prob)
```

```
C:\Users\xgao\Anaconda3\lib\site-packages\sklearn\ensemble\weight_boosting.py:29: DeprecationWarning: n
umpy.core.umath_tests is an internal NumPy module and should not be imported. It will be removed in a f
uture NumPy release.
  from numpy.core.umath_tests import inner1d
```

For the second model, I used lightgbm classifier as it's faster in training compared to xgboost. First I tried using max_depth =50, learning_rate =0.1, n_estimators=200 and this model is enough to beat the the score. The total training time for this model is 4.5 mins

training time for this model is 4.5 mins

In [8]:

```
avg[1,0] = model_lgb(X_train,y_train,X_test,y_test)
avg[1,1] = model_lgb(X_train1,y_train1,X_test1,y_test1)
avg[1,2] = model_lgb(X_train2,y_train2,X_test2,y_test2)
```

```
Fitting 3 folds for each of 1 candidates, totalling 3 fits
[Parallel(n_jobs=1)]: Done   3 out of   3 | elapsed:  1.5min finished
Fitting 3 folds for each of 1 candidates, totalling 3 fits
[Parallel(n_jobs=1)]: Done   3 out of   3 | elapsed:  1.5min finished
Fitting 3 folds for each of 1 candidates, totalling 3 fits
[Parallel(n_jobs=1)]: Done   3 out of   3 | elapsed:  1.5min finished
```

In [11]:

```
# second model, random forest
from sklearn.ensemble import RandomForestClassifier
def model_rfc(train_x,train_y,test_x,test_y):
    pipeline_rfc = Pipeline([
        ('imputer', Imputer(copy=False)),
        ('model', RandomForestClassifier(n_jobs=-1, random_state=1))
    ])
    param_grid_rfc = {
        'model__n_estimators': [50] # The number of randomized trees to build
    }
    grid_rfc = GridSearchCV(estimator=pipeline_rfc, param_grid=param_grid_rfc, scoring='roc_auc', n_job
s=1, pre_dispatch=1, cv=3, verbose=1, return_train_score=False)
    grid_rfc.fit(train_x,train_y)
    y_prob = grid_rfc.predict_proba(test_x)
    return log_loss(test_y, y_prob)
```

For the third model, I used randomforest with n_estimators = 50 as a starting point and it can get an average of 0.48 log loss. Since the training is sluggish so I didn't tried other parameters. The training time is 6.7mins for three splits

In [12]:

```
avg[2,0] = model_rfc(X_train,y_train,X_test,y_test)
avg[2,1] = model_rfc(X_train1,y_train1,X_test1,y_test1)
avg[2,2] = model_rfc(X_train2,y_train2,X_test2,y_test2)
```

```
Fitting 3 folds for each of 1 candidates, totalling 3 fits
[Parallel(n_jobs=1)]: Done   3 out of   3 | elapsed:  2.3min finished
Fitting 3 folds for each of 1 candidates, totalling 3 fits
[Parallel(n_jobs=1)]: Done   3 out of   3 | elapsed:  2.2min finished
Fitting 3 folds for each of 1 candidates, totalling 3 fits
[Parallel(n_jobs=1)]: Done   3 out of   3 | elapsed:  2.2min finished
```

In [13]:

```
final_result = pd.DataFrame(avg,columns=['test1','test2','test3'],index=['model1','model2','model3'])
final_result['average'] = final_result.mean(1)
final_result
```

Out[13]:

	test1	test2	test3	average
model1	0.558993	0.653265	0.563753	0.592004
model2	0.446692	0.448138	0.447188	0.447340
model3	0.479729	0.482980	0.479775	0.480828

Model 2 using lightgbm shows an average of 0.44734 while logistic regression only gives 0.59 and random forest is 0.48