applied-project-codes

August 16, 2023

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

import time
import warnings
warnings.simplefilter(action='ignore', category=FutureWarning)
```

```
In C:\Users\HP\Anaconda3\lib\site-packages\matplotlib\mpl-
data\stylelib\_classic_test.mplstyle:
The savefig.frameon rcparam was deprecated in Matplotlib 3.1 and will be removed
in 3.3.
In C:\Users\HP\Anaconda3\lib\site-packages\matplotlib\mpl-
data\stylelib\_classic_test.mplstyle:
The verbose.level rcparam was deprecated in Matplotlib 3.1 and will be removed
in 3.3.
In C:\Users\HP\Anaconda3\lib\site-packages\matplotlib\mpl-
data\stylelib\_classic_test.mplstyle:
The verbose.fileo rcparam was deprecated in Matplotlib 3.1 and will be removed
in 3.3.
```

1 1. Data Overview and Data Processing

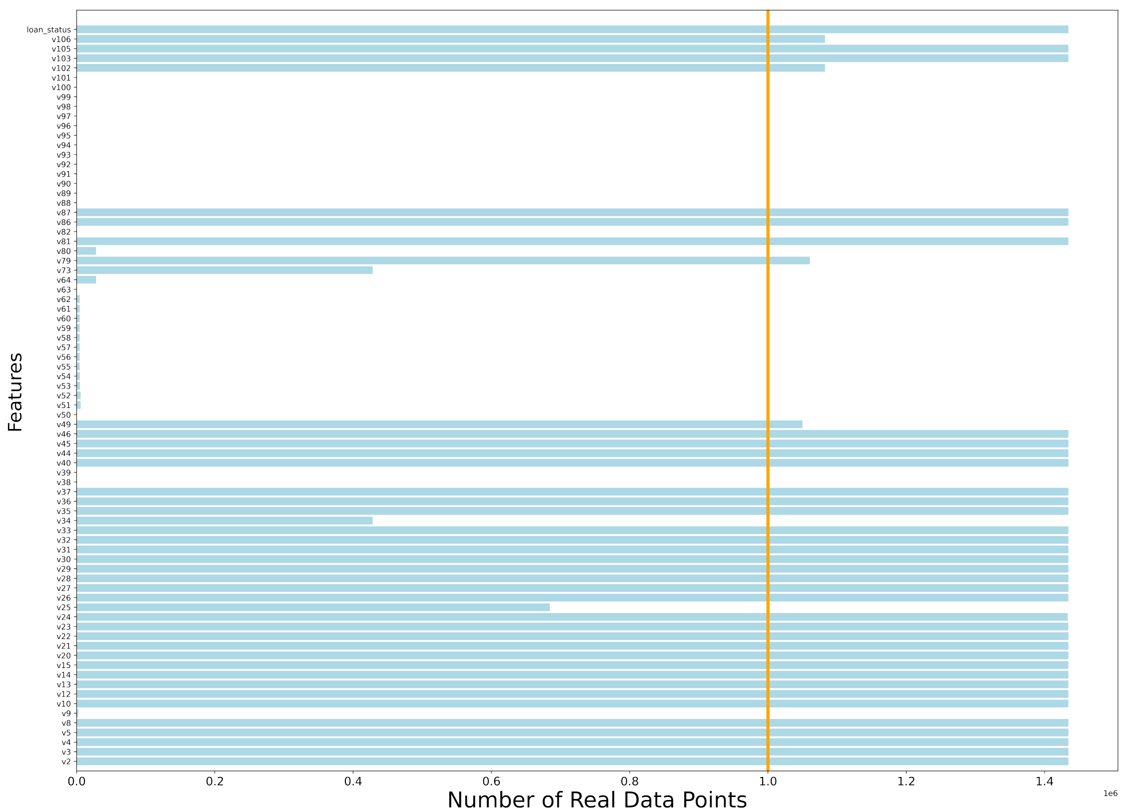
[2]:
```
df = pd.read_csv('data.csv',sep = '\t', error_bad_lines=False)

print(df.shape)
print(df['loan_status'].value_counts())
```

```
C:\Users\HP\Anaconda3\lib\site-packages\IPython\core\interactiveshell.py:3057:
DtypeWarning: Columns (72,75) have mixed types.Specify dtype option on import or
set low_memory=False.
  interactivity=interactivity, compiler=compiler, result=result)
```

```
(1434388, 77)
fully repaid    1406537
defaulted         27851
Name: loan_status, dtype: int64
```

[3]:
```
plt.figure(figsize=(24,18), dpi=500)
plt.ylim(0,79)

plt.barh( range(1,78,1), height=0.8 ,width=(-df.isna()).sum(),␣
 ↪color="lightblue")
plt.xticks(fontsize=15)
plt.yticks(range(1,78,1), df.columns)
plt.vlines(x=1000000,ymin=0,ymax=79,color='orange',linewidth=4)

plt.xlabel("Number of Real Data Points",fontsize=28)
plt.ylabel("Features",fontsize=24)

plt.legend(frameon=False)
plt.savefig('Non NAN Number.jpg',dpi=500)
plt.show()
```

```
No handles with labels found to put in legend.
```



1.1 1.1 Value Dropping

1.1.1 We will drop columns that have less than 1,000,000 real data points (Non NAN values). And then rename the columns according to the SFLP_dictionary

[4]:
```
df = df[df.columns[((-df.isna()).sum() > 1000000)]]
vnums = np.array(df.columns)

df = df.rename(columns={'v2': 'LOAN_ID', 'v3': 'MNTH_REP', 'v4': 'ORIG_CHN',␣
↪'v5': 'Seller.Name', 'v8': 'ORIG_RT',
                        'v10': 'ORIG_AMT', 'v12': 'CUR_AMT', 'v13': 'ORIG_TRM',␣
↪'v14': 'ORIG_DTE', 'v15': 'FRST_DTE',
                        'v20': 'OLTV', 'v21': 'OCLTV', 'v22': 'NUM_BO', 'v23':␣
↪'DTI', 'v24': 'CSCORE_B', 'v25': 'CSCORE_C' ,
                        'v26': 'FTHB_FLG', 'v27': 'PURPOSE', 'v28': 'PROP_TYP',␣
↪'v29': 'NUM_UNIT', 'v30': 'OCC_STAT', 'v31': 'STATE',
                        'v32': 'MSA', 'v33': 'ZIP_3', 'v35': 'AM_TYPE', 'v36':␣
↪'PreP_FLG',
                        'v37': 'INTOnlly_FLG', 'v40': 'Delq.Status', 'v44':␣
↪'Zero.Bal.Code', 'v45': 'Zero.Bal.Date',
                        'v46': 'REM_AMT', 'v49': 'DIFF_UPB', 'v79':␣
↪'SPEC_PRG','v81': 'RELOCATION_FLG',
                        'v86': 'VAL_METH', 'v87': 'HBL_FLG','v102': 'ASS_PLAN',␣
↪'v103': 'HLTV_FLG',
                        'v105': 'REPUR_FLG','v106': 'ALT_DELINQ'})
```

1.1.2 Some columns need to be dropped because they are unmatched to the data description. Eg: CUR_AMT, AM_TYPE contain all the same type of data, which do not have categories.

[5]:
```
df["ASS_PLAN"] = df["ASS_PLAN"].replace({7:"F"})
df["ALT_DELINQ"] = df["ALT_DELINQ"].replace({7:"P"})

remove_cols = ["v12","v35","v36","v37"]
vnums = np.setdiff1d(vnums,remove_cols)

df = df.drop( columns=["CUR_AMT","AM_TYPE","PreP_FLG","INTOnlly_FLG"] )
df.columns
```

[5]:
```
Index(['LOAN_ID', 'MNTH_REP', 'ORIG_CHN', 'Seller.Name', 'ORIG_RT', 'ORIG_AMT',
       'ORIG_TRM', 'ORIG_DTE', 'FRST_DTE', 'OLTV', 'OCLTV', 'NUM_BO', 'DTI',
       'CSCORE_B', 'FTHB_FLG', 'PURPOSE', 'PROP_TYP', 'NUM_UNIT', 'OCC_STAT',
       'STATE', 'MSA', 'ZIP_3', 'Delq.Status', 'Zero.Bal.Code',
       'Zero.Bal.Date', 'REM_AMT', 'DIFF_UPB', 'SPEC_PRG', 'RELOCATION_FLG',
       'VAL_METH', 'HBL_FLG', 'ASS_PLAN', 'HLTV_FLG', 'REPUR_FLG',
       'ALT_DELINQ', 'loan_status'],
      dtype='object')
```

1.1.3 Some variables are only available after the loans have been issued. Should drop them because they are the future data.

[6]:
```
future_data = ["ORIG_DTE","FRST_DTE","Delq.Status","Zero.Bal.Code","Zero.Bal.
↪Date","REM_AMT"]
df = df.drop( columns=future_data )

vnums= np.setdiff1d(vnums,["v14","v15","v40","v44","v45","v46"])
vnums
```

[6]:
```
array(['loan_status', 'v10', 'v102', 'v103', 'v105', 'v106', 'v13', 'v2',
       'v20', 'v21', 'v22', 'v23', 'v24', 'v26', 'v27', 'v28', 'v29',
       'v3', 'v30', 'v31', 'v32', 'v33', 'v4', 'v49', 'v5', 'v79', 'v8',
       'v81', 'v86', 'v87'], dtype=object)
```

1.1.4 Some columns contain no meaning (eg. LOAN_ID and Seller Name) in predicting default. Drop them as well.

[7]:
```
df = df.drop( columns=["LOAN_ID",  "Seller.Name"] )
vnums = np.setdiff1d(vnums,["v2","v5"])
vnums
```

[7]:
```
array(['loan_status', 'v10', 'v102', 'v103', 'v105', 'v106', 'v13', 'v20',
       'v21', 'v22', 'v23', 'v24', 'v26', 'v27', 'v28', 'v29', 'v3',
       'v30', 'v31', 'v32', 'v33', 'v4', 'v49', 'v79', 'v8', 'v81', 'v86',
       'v87'], dtype=object)
```

1.2 1.2 Encoding and Fillna

1.2.1 Encode the "string" type columns into numeric type and then fill NAN value with mean() function.

[8]:
```
# Encode categorical variables to numeric type
df['loan_status'] = df['loan_status'].replace({'defaulted' : 1, 'fully repaid' :
↪ 0})

#########
# encoding dummy variables
#########
from sklearn.preprocessing import LabelEncoder
lbecd = LabelEncoder()


df["SPEC_PRG"] = lbecd.fit_transform( df["SPEC_PRG"] )  # values: N,Y
df["FTHB_FLG"] = lbecd.fit_transform( df["FTHB_FLG"] )  # values: N,Y
df["RELOCATION_FLG"] = lbecd.fit_transform( df["RELOCATION_FLG"] ) # Whether␣
↪the loan is Relocation Mortgage loan. Values: N,Y
```

```
df["HBL_FLG"] = lbecd.fit_transform( df["HBL_FLG"] ) # if original loan␣
↪priciple is greater than general conforming loan limit. Values: N,Y
df["HLTV_FLG"] = lbecd.fit_transform( df["HLTV_FLG"] ) # if original reference␣
↪loan is refinanced under Fannie Mae's HLTV refinance option. Values: N,Y
df["REPUR_FLG"] = lbecd.fit_transform( df["REPUR_FLG"] ) # if Fannie Mae␣
↪received warranty arrangements for the repurchase of the mortgage loan.␣
↪Values: N,Y
```

[9]:

```
#########
# encoding categorical variables
#########

def rank_lbecd(column):
    le = LabelEncoder()

    # get the default case proportion in each category of input variable
    # and encode the input vairable by the rank of default rate in each␣
↪category
    rank_order = df.copy().groupby([column],dropna=False)["loan_status"].mean().
↪sort_values().index
    le.classes_ = np.array(rank_order)

    return le.transform( column.values )


df["PURPOSE"] = rank_lbecd( df["PURPOSE"] )  # loaning purpose: Cash-Out␣
↪Refinance = C; Refinance = R; Purchase = P
df["OCC_STAT"] = rank_lbecd( df["OCC_STAT"] ) # propetety occupation status:␣
↪Principal = P; Second = S; Investor = I
df['VAL_METH'] = df['VAL_METH'].replace({'.' : "O"})
df["VAL_METH"] = rank_lbecd( df["VAL_METH"] ) # the mothod by which the value␣
↪of property is obtained: A = Appraisal; P = Onsite Property Data Collection;
↪R = GSE Targeted Refinance; W = Appraisal Waiver; O = Other
df["ORIG_CHN"] = rank_lbecd( df["ORIG_CHN"] ) #"ORIG_CHN": Retail = R;␣
↪Correspondent = C; Broker = B
df["PROP_TYP"] = rank_lbecd( df["PROP_TYP"] ) # "PROP_TYP": CO = condominium;␣
↪CP = co-operative; PU = Planned Urban Development; MH = manufactured home;␣
↪SF = single-family home
df["STATE"] = rank_lbecd( df["STATE"] ) # 56 USA states
df["ASS_PLAN"] = rank_lbecd( df["ASS_PLAN"] ) # "ASS_PLAN": F = Forbearance␣
↪Plan; 7= Not Applicable; N = No Workout Plan
df["ALT_DELINQ"] = rank_lbecd( df["ALT_DELINQ"] ) # "ALT_DELINQ": P = payment␣
↪deferral option; C = payment deferral option specific to COVID-19; 7 = Not␣
↪Applicable
```

```
df["ZIP_3"] = rank_lbecd( df["ZIP_3"] ) # 892 categories of first three digits␣
↪of the code designated by the U.S. Postal Service where the subject property␣
↪is located.
```

[10]:
```
#########
# fillna and other data processing processes
#########

# type modification
df = df.astype({"ORIG_AMT": 'float64', "OLTV": 'float64', "OCLTV": 'float64' })
# "MSA": object

# NA features filtering
#### whether using mean filtering or other methods is not sure
df["DIFF_UPB"] = df["DIFF_UPB"].fillna(df["DIFF_UPB"].mean())
df["ORIG_RT"] = df["ORIG_RT"].fillna(df["ORIG_RT"].mean())
df["DTI"] = df["DTI"].fillna(df["DTI"].mean())
df["CSCORE_B"] = df["CSCORE_B"].fillna(df["CSCORE_B"].mean())

print(df.isna().sum().sum()) # no nan remains
print(df.shape)
```

```
0
(1434388, 28)
```

1.3 1.3 Generate a Subsample dataset to replace the entire one

[11]:
```
# to save the run time and memory cost, a small sample of 5% original␣
↪observations is selected for the coming processes.
# the target variables distribution is the same as the original one
sample = df.groupby("loan_status").apply(lambda x: x.sample(frac=0.06,␣
↪random_state=2023)).reset_index(drop=True)
print(sample.shape)
sample.to_csv("data_sample.csv",index=False)
```

```
(86063, 28)
```

1.3.1 When the data sample is ready, can actually start running whole notebook from the next cell.

[12]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

sample = pd.read_csv("data_sample.csv")
sample = sample/sample.abs().max()
print(sample.shape)
```

```
sample.head()
```

```
(86063, 28)
```

[12]:

```
   MNTH_REP  ORIG_CHN   ORIG_RT  ORIG_AMT  ORIG_TRM      OLTV     OCLTV  \
0  0.426320       1.0  0.660377  0.143164       1.0  0.814433  0.718182
1  0.836086       0.5  0.603774  0.087330       1.0  0.979381  0.863636
2  0.836086       0.0  0.566038  0.234789       1.0  0.824742  0.727273
3  0.754132       1.0  0.547170  0.100215       0.5  0.432990  0.381818
4  0.180477       0.5  0.471698  0.088762       1.0  0.731959  0.645455

   NUM_BO       DTI  CSCORE_B  …  DIFF_UPB  SPEC_PRG  RELOCATION_FLG  \
0    0.25  0.921569  0.903571  …  0.196069       0.0             0.0
1    0.25  0.823529  0.857143  …  0.117912       0.5             0.0
2    0.50  0.450980  0.938095  …  0.315402       0.0             0.0
3    0.25  0.372549  0.946429  …  0.051795       0.0             0.0
4    0.25  0.450980  0.913095  …  0.120490       0.0             0.0

   VAL_METH  HBL_FLG  ASS_PLAN  HLTV_FLG  REPUR_FLG  ALT_DELINQ  loan_status
0      1.00      0.0  0.666667       0.0        0.0    0.000000          0.0
1      1.00      0.0  0.666667       0.0        0.0    0.333333          0.0
2      0.25      0.0  0.666667       0.0        0.0    0.333333          0.0
3      1.00      0.0  0.666667       0.0        0.0    0.333333          0.0
4      1.00      0.0  0.666667       0.0        0.0    0.333333          0.0

[5 rows x 28 columns]
```

2 2. Federated Learning Simulation

2.1 2.1 Train-test-split and dealing with imbalanced data problem with SMOTE

[13]:

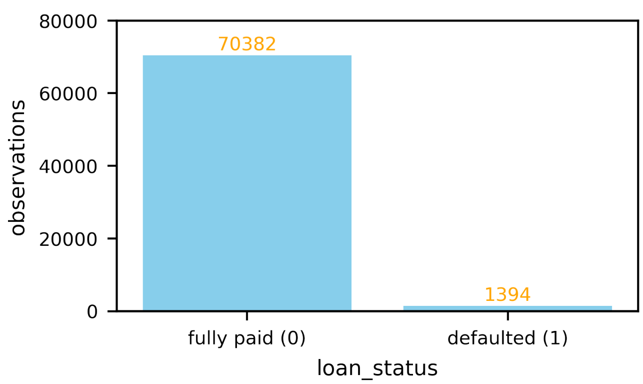
```
from sklearn.model_selection import train_test_split

Xtrain, Xtest, Ytrain, Ytest = train_test_split( sample.iloc[:, sample.columns!
 ↪= "loan_status"], sample.loan_status,
                                                test_size=0.166,␣
 ↪stratify=sample.loan_status, random_state=2023)

print(Xtrain.shape, Xtest.shape, Ytrain.shape, Ytest.shape)
```

```
(71776, 27) (14287, 27) (71776,) (14287,)
```

[14]:

```
plt.figure(figsize=(3.5,2),dpi=500)

plt.xlim(0.5,2.5)
plt.ylim(0,80000)
plt.xticks(range(1,3,1),  ["fully paid (0)","defaulted (1)"],fontsize=7)
```

```
plt.yticks(fontsize=7)
plt.xlabel('loan_status',fontsize=8)
plt.ylabel("observations",fontsize=8)

orginal_editor = Ytrain.value_counts()

plt.hlines(y=0,xmin=0.5,xmax=2.5,color='black',linewidth=1)
plt.bar( range(1,3,1),orginal_editor,width=0.8,color='skyblue')


for i in range(1,3,1):
    plt.
 ↪text(i,orginal_editor[i-1]+1500,str(orginal_editor[i-1]),ha="center",fontsize=7,color="oran

plt.show()
```

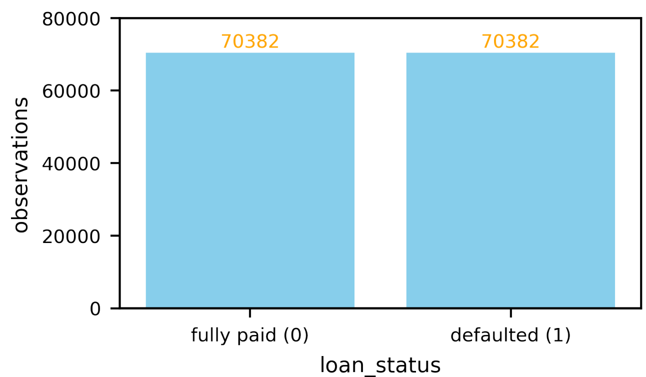


[15]:
```
# Applying SMOTE technique
# SMOTE increases recall at the cost of lower precision
from imblearn.over_sampling import SMOTE

Xtrain_smote, Ytrain_smote = SMOTE().fit_resample(Xtrain, Ytrain)
print(Xtrain_smote.shape)
print(Ytrain_smote.value_counts()) # Now the #1:#0 = 1:1
```

```
(140764, 27)
0.0    70382
1.0    70382
Name: loan_status, dtype: int64
```

[16]:
```
plt.figure(figsize=(3.5,2),dpi=500)

plt.xlim(0.5,2.5)
plt.ylim(0,80000)
```

```
plt.xticks(range(1,3,1),  ["fully paid (0)","defaulted (1)"],fontsize=7)
plt.yticks(fontsize=7)
plt.xlabel('loan_status',fontsize=8)
plt.ylabel("observations",fontsize=8)

orginal_editor = Ytrain_smote.value_counts()

plt.hlines(y=0,xmin=0.5,xmax=2.5,color='black',linewidth=1)
plt.bar( range(1,3,1),orginal_editor,width=0.8,color='skyblue')


for i in range(1,3,1):
    plt.
 ↪text(i,orginal_editor[i-1]+1500,str(orginal_editor[i-1]),ha="center",fontsize=7,color="oran

plt.show()
```



[17]:
```
Xtrain_smote = Xtrain_smote.reset_index(drop=True)
Ytrain_smote = Ytrain_smote.reset_index(drop=True)
Xtest = Xtest.reset_index(drop=True)
Ytest = Ytest.reset_index(drop=True)
```

2.2 2.2 Federated Learning Structure

2.2.1 2.2.1 Define Configurations (using linear regression as defaulted settings)

[18]:
```
from sklearn.linear_model import LinearRegression

conf =  {
    "model_name" : LinearRegression(),
    "no_models" : 10,
    "global_epochs" : 20,
    "k" : 5,
```

```
    "lambda" : 0.15
    }
```

2.2.2 2.2.2 Define a Server

[19]:
```
from sklearn.metrics import average_precision_score, log_loss

class Server():

    def __init__(self, conf, Xtest, Ytest):
        self.conf = conf
        self.global_model = self.conf["model_name"]
        self.Xtest = Xtest
        self.Ytest = Ytest

        # to execute the fit() first to get the .coef_ and .intercept_␣
↪attribute available to invoke
        self.global_model.fit( np.zeros(self.Xtest.shape), np.zeros( len(self.
↪Ytest)) )

        #when initalized, self.global_model.coef_ = [0*27]; self.global_model.
↪intercept_ = 0



    def model_aggregate(self, grads, global_epoch):

        if global_epoch==0:

            self.global_model.coef_ = np.array(grads["Beta_base"]).mean(axis=0)
            self.global_model.intercept_ = np.array(grads["Intercept_base"]).
↪mean()


        self.global_model.coef_ -=  np.array(grads["gBetas"]).mean(axis=0) *␣
↪self.conf["lambda"]
        self.global_model.intercept_ -=  np.array(grads["gIntercepts"]).mean()␣
↪* self.conf["lambda"]



    def model_eval(self):
        # calculate the precision-recall AUC score
        precision_recall_auc = average_precision_score(Ytest, self.global_model.
↪predict(self.Xtest))
```

```
        # calculate the cross-entropy loss
        global_loss = log_loss(Ytest, self.global_model.predict(self.Xtest))

        return precision_recall_auc, global_loss
```

2.2.3 2.2.3 Define The Client Class

[20]:
```
class Client():

    def __init__(self, conf, Xtrain_full, Ytrain_full, cid = -1):

        self.conf = conf
        self.local_model = self.conf["model_name"]
        self.client_id = cid
        self.Xtrain_full = Xtrain_full
        self.Ytrain_full = Ytrain_full

        # get the local dataset of a client
        data_len = int(len(self.Xtrain_full) / self.conf['no_models'])

        if (cid+1) == self.conf['no_models']:
            self.local_Xtrain = self.Xtrain_full.iloc[cid * data_len: ]
            self.local_Ytrain = self.Ytrain_full.iloc[cid * data_len: ]
        else:
            self.local_Xtrain = self.Xtrain_full.iloc[cid * data_len: (cid+1) *␣
↪data_len]
            self.local_Ytrain = self.Ytrain_full.iloc[cid * data_len: (cid+1) *␣
↪data_len]


    def local_train(self, global_model, global_epoch):

        if global_epoch==0:
            # first iteration, train fit on a local subsample to generate the␣
↪parameters base
            # it will let the global model converge faster than starting from␣
↪all zeros
            local_subsample_id = random.sample(  range(len(self.
↪local_Xtrain)),int(len(self.local_Xtrain)/5)   )
            self.local_model.fit(  self.local_Xtrain.iloc[local_subsample_id],␣
↪self.local_Ytrain.iloc[local_subsample_id]  )

            self.local_model.coef_ = np.zeros(27)
            self.local_model.intercept_ = 0

        else:
```

```
            # overwrite the local_model's coefficients by the global_models'
            self.local_model.coef_ = global_model.coef_.copy()
            self.local_model.intercept_ = global_model.intercept_.copy()


        # get gradients of parameters in the linear regression model (using␣
↪cross-entropy as the loss function)
        ypred = self.local_model.predict(self.local_Xtrain)

        grad_coef = (1/len(ypred)) * self.local_Xtrain.T.dot(ypred - self.
↪local_Ytrain)
        grad_intercept = np.mean(ypred - self.local_Ytrain)

        return grad_coef, grad_intercept
```

2.2.4 2.2.4 Main Structure

[21]:
```
import random
import time
start = time.time()
prauc_scores = {"federated":[]}
loss_scores = {"federated":[]}


if __name__ == '__main__':

    # generate a server and the client instances
    server = Server(conf, Xtest, Ytest)
    clients = []

    for cid in range(conf["no_models"]):
        clients.append( Client(conf, Xtrain_smote, Ytrain_smote, cid)  )

    print("Generated one Server and",conf["no_models"],"Clients...\n\n")



    # global iterations epochs
    for e in range(conf["global_epochs"]):

        gradients = {   "Beta_base":[],
                        "Intercept_base":[],
                        "gBetas":[],
                        "gIntercepts":[]
                    }

        if e==0:
```

```python
            candidates = clients.copy()
        else:
            # in every epoch, just select k clients for federated training
            candidates = random.sample(clients, conf["k"])


        for c in candidates:
            coef_grads, itcp_grads = c.local_train(server.global_model, e)

            if e==0:
                gradients["Beta_base"].append(c.local_model.coef_)
                gradients["Intercept_base"].append(c.local_model.intercept_)


            gradients["gBetas"].append(coef_grads)
            gradients["gIntercepts"].append(itcp_grads)

        # pass the gradients data to the server for aggregation.
        server.model_aggregate(gradients, e)

        pr_auc, loss = server.model_eval()

        print("Epoch %d, precision_recall score: %f, loss: %f" % (e, pr_auc,␣
 ↪loss))

        prauc_scores["federated"].append(pr_auc)
        loss_scores["federated"].append(loss)

end = time.time()
print("run time cost is", end-start, "seconds")
```

```
Generated one Server and 10 Clients…


Epoch 0, precision_recall score: 0.066318, loss: 1.158814
Epoch 1, precision_recall score: 0.283805, loss: 0.360570
Epoch 2, precision_recall score: 0.286746, loss: 0.535055
Epoch 3, precision_recall score: 0.388485, loss: 0.490388
Epoch 4, precision_recall score: 0.475235, loss: 0.458294
Epoch 5, precision_recall score: 0.269407, loss: 3.281494
Epoch 6, precision_recall score: 0.628286, loss: 0.236439
Epoch 7, precision_recall score: 0.422979, loss: 1.138762
Epoch 8, precision_recall score: 0.556164, loss: 0.637114
Epoch 9, precision_recall score: 0.622775, loss: 0.393575
Epoch 10, precision_recall score: 0.548270, loss: 0.861265
Epoch 11, precision_recall score: 0.630261, loss: 0.321409
Epoch 12, precision_recall score: 0.568984, loss: 0.926967
```

```
Epoch 13, precision_recall score: 0.631212, loss: 0.294680
Epoch 14, precision_recall score: 0.629162, loss: 0.443202
Epoch 15, precision_recall score: 0.631437, loss: 0.147109
Epoch 16, precision_recall score: 0.628995, loss: 0.505616
Epoch 17, precision_recall score: 0.624215, loss: 0.680658
Epoch 18, precision_recall score: 0.602123, loss: 1.062550
Epoch 19, precision_recall score: 0.630839, loss: 0.479796
run time cost is 0.8995251655578613 seconds
```

[ ]:

```

```

2.3 2.3 Result Comparison

2.3.1 2.3.1 Centralized Model on Full Dataset

[22]:

```python
# centralized training
import warnings
warnings.simplefilter(action='ignore')

Cen_lgr =  conf["model_name"]
Cen_lgr.coef_ = np.zeros(27)
Cen_lgr.intercept_ = 0

prauc_scores["centralized"] = []
loss_scores["centralized"] = []

for e in range(conf["global_epochs"]):

    ypred = Cen_lgr.predict(Xtrain_smote.values)

    Cen_lgr.coef_ -= (1/len(ypred)) * Xtrain_smote.T.dot(ypred - Ytrain_smote)␣
 ↪* conf["lambda"]
    Cen_lgr.intercept_ -= np.mean(ypred - Ytrain_smote) * conf["lambda"]

    # evaluation
    pr_auc = average_precision_score(Ytest, Cen_lgr.predict(Xtest.values))
    loss = log_loss(Ytest, Cen_lgr.predict(Xtest.values))
    print("Epoch %d, precision_recall score: %f, loss: %f" % (e, pr_auc, loss))

    prauc_scores["centralized"].append(pr_auc)
    loss_scores["centralized"].append(loss)
```

```
Epoch 0, precision_recall score: 0.066318, loss: 1.158878
Epoch 1, precision_recall score: 0.198240, loss: 0.526320
Epoch 2, precision_recall score: 0.236489, loss: 0.706895
Epoch 3, precision_recall score: 0.337982, loss: 0.614675
Epoch 4, precision_recall score: 0.397467, loss: 0.633619
Epoch 5, precision_recall score: 0.462737, loss: 0.610672
```

```
Epoch 6, precision_recall score: 0.508346, loss: 0.604806
Epoch 7, precision_recall score: 0.543364, loss: 0.592907
Epoch 8, precision_recall score: 0.568303, loss: 0.583945
Epoch 9, precision_recall score: 0.588038, loss: 0.574387
Epoch 10, precision_recall score: 0.602341, loss: 0.565584
Epoch 11, precision_recall score: 0.612446, loss: 0.556982
Epoch 12, precision_recall score: 0.619337, loss: 0.548773
Epoch 13, precision_recall score: 0.622673, loss: 0.540860
Epoch 14, precision_recall score: 0.625588, loss: 0.533258
Epoch 15, precision_recall score: 0.627274, loss: 0.525941
Epoch 16, precision_recall score: 0.628310, loss: 0.518902
Epoch 17, precision_recall score: 0.629297, loss: 0.512124
Epoch 18, precision_recall score: 0.630009, loss: 0.505598
Epoch 19, precision_recall score: 0.630644, loss: 0.499311
```

[ ]:

```

```

2.3.2 2.3.2 Single Local Model Average Performance

[23]:

```
conf["no_models"] = 10


clients = []
for cid in range(conf["no_models"]):
    clients.append( Client(conf, Xtrain_smote, Ytrain_smote, cid)  )


for c in clients:
    c.local_model.coef_ = np.zeros(27)
    c.local_model.intercept_ = 0



prauc_scores["Single"] = []
loss_scores["Single"] = []

prauc_e = []
loss_e = []

for e in range(conf["global_epochs"]):
    for c in clients:
        ypred = c.local_model.predict(c.local_Xtrain.values)
        c.local_model.coef_ -= (1/len(ypred)) * c.local_Xtrain.T.dot(ypred - c.
 ↪local_Ytrain) * conf["lambda"]
        c.local_model.intercept_ -= np.mean(ypred - c.local_Ytrain) *␣
 ↪conf["lambda"]
```

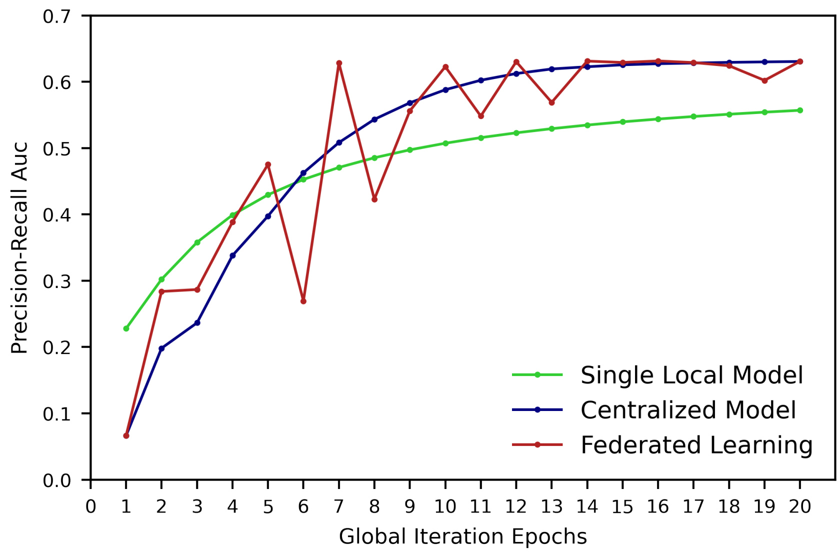
```
        # evaluation
        pr_auc = average_precision_score(Ytest, c.local_model.predict(Xtest.
↪values))
        loss = log_loss(Ytest, c.local_model.predict(Xtest.values))

        prauc_e.append(pr_auc)
        loss_e.append(loss)

    avg_prauc = np.mean(prauc_e)
    avg_loss = np.mean(loss_e)
    print("Epoch %d, precision_recall score: %f, loss: %f" % (e, avg_prauc,␣
↪avg_loss) )

    prauc_scores["Single"].append(avg_prauc)
    loss_scores["Single"].append(avg_loss)
```

```
Epoch 0, precision_recall score: 0.227903, loss: 6.802936
Epoch 1, precision_recall score: 0.302180, loss: 6.234017
Epoch 2, precision_recall score: 0.357944, loss: 5.747979
Epoch 3, precision_recall score: 0.399177, loss: 5.351075
Epoch 4, precision_recall score: 0.429589, loss: 5.022111
Epoch 5, precision_recall score: 0.452682, loss: 4.747003
Epoch 6, precision_recall score: 0.470826, loss: 4.513260
Epoch 7, precision_recall score: 0.485407, loss: 4.311598
Epoch 8, precision_recall score: 0.497366, loss: 4.135463
Epoch 9, precision_recall score: 0.507344, loss: 3.981135
Epoch 10, precision_recall score: 0.515790, loss: 3.844527
Epoch 11, precision_recall score: 0.523026, loss: 3.722925
Epoch 12, precision_recall score: 0.529289, loss: 3.614137
Epoch 13, precision_recall score: 0.534759, loss: 3.516308
Epoch 14, precision_recall score: 0.539578, loss: 3.428258
Epoch 15, precision_recall score: 0.543854, loss: 3.348441
Epoch 16, precision_recall score: 0.547669, loss: 3.275573
Epoch 17, precision_recall score: 0.551094, loss: 3.209097
Epoch 18, precision_recall score: 0.554190, loss: 3.148399
Epoch 19, precision_recall score: 0.556996, loss: 3.092499
```

2.3.3 2.3.3 Result Visualization

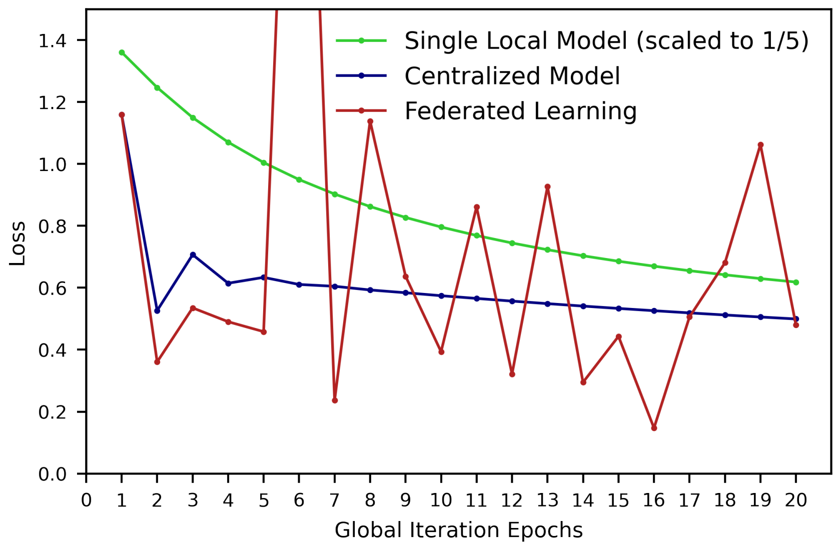
[24]:
```
import matplotlib.pyplot as plt

plt.figure(figsize=(5,3.2),dpi=500)

plt.xlim(0,21)
plt.ylim(0,0.7)
plt.xticks(range(0,21,1),fontsize=7)
plt.yticks(fontsize=7)
```

```
plt.xlabel("Global Iteration Epochs", fontsize=8)
plt.ylabel("Precision-Recall Auc", fontsize=8)

plt.plot(range(1,21,1),prauc_scores["Single"],linewidth=1, marker='.',␣
↪markersize='2.5', color="limegreen",label="Single Local Model")
plt.plot(range(1,21,1),prauc_scores["centralized"],linewidth=1, marker='.',␣
↪markersize='2.5', color="navy", label="Centralized Model")
plt.plot(range(1,21,1),prauc_scores["federated"],linewidth=1, marker='.',␣
↪markersize='2.5', color="firebrick",label="Federated Learning")
plt.legend(loc="lower right", fontsize=9, frameon=False)


plt.show()
```



[25]:
```
plt.figure(figsize=(5,3.2),dpi=500)

plt.xlim(0,21)
plt.ylim(0,1.5)
plt.xticks(range(0,21,1),fontsize=7)
plt.yticks(fontsize=7)
plt.xlabel("Global Iteration Epochs", fontsize=8)
plt.ylabel("Loss", fontsize=8)

plt.plot(range(1,21,1),np.array(loss_scores["Single"])/5,linewidth=1, marker='.
↪', markersize='2.5', color="limegreen",label="Single Local Model (scaled to␣
↪1/5)")
plt.plot(range(1,21,1),loss_scores["centralized"],linewidth=1, marker='.',␣
↪markersize='2.5', color="navy", label="Centralized Model")
```

```
plt.plot(range(1,21,1),loss_scores["federated"],linewidth=1, marker='.',␣
↪markersize='2.5', color="firebrick",label="Federated Learning")
plt.legend(loc="upper right", fontsize=9, frameon=False)


plt.show()
```



[ ]:

```

```

2.3.4 2.4 Applying Homomorphic Encryption

[26]:

```
pip install phe
```

```
Requirement already satisfied: phe in c:\users\hp\anaconda3\lib\site-packages
(1.5.0)
Note: you may need to restart the kernel to use updated packages.
```

```
WARNING: Ignoring invalid distribution -cipy (c:\users\hp\anaconda3\lib\site-
packages)
WARNING: Ignoring invalid distribution -cipy (c:\users\hp\anaconda3\lib\site-
packages)

[notice] A new release of pip is available: 23.1.2 -> 23.2.1
[notice] To update, run: python.exe -m pip install --upgrade pip
```

[27]:

```
from phe import paillier
public_key, private_key = paillier.generate_paillier_keypair()
```

[28]:
```python
class Server_homo():

    def __init__(self, conf, Xtest, Ytest):
        self.conf = conf
        self.global_model = self.conf["model_name"]
        self.Xtest = Xtest
        self.Ytest = Ytest

        # to execute the fit() first to get the .coef_ and .intercept_ attribute available to invoke
        self.global_model.fit( np.zeros(self.Xtest.shape), np.zeros( len(self.Ytest)) )

    def model_aggregate(self, grads, global_epoch):

        if global_epoch==0:

            self.global_model.coef_ = np.array(grads["Beta_base"]).mean(axis=0)
            self.global_model.intercept_ = np.array(grads["Intercept_base"]).mean()


        self.global_model.coef_ -=  np.array(grads["gBetas"]).mean(axis=0) * self.conf["lambda"]
        self.global_model.intercept_ -=  np.array(grads["gIntercepts"]).mean() * self.conf["lambda"]

    def model_eval(self):
        # calculate the precision-recall AUC score
        self.global_model.coef_ = np.array([private_key.decrypt(x) for x in self.global_model.coef_])
        self.global_model.intercept_ = private_key.decrypt(self.global_model.intercept_)

        precision_recall_auc = average_precision_score(Ytest, self.global_model.predict(self.Xtest))

        # calculate the cross-entropy loss
        global_loss = log_loss(Ytest, self.global_model.predict(self.Xtest))

        return precision_recall_auc, global_loss
```

[29]:
```python
class Client_homo():

    def __init__(self, conf, Xtrain_full, Ytrain_full, cid = -1):
```

```
        self.conf = conf
        self.local_model = self.conf["model_name"]
        self.client_id = cid
        self.Xtrain_full = Xtrain_full
        self.Ytrain_full = Ytrain_full

        # get the local dataset of a client
        data_len = int(len(self.Xtrain_full) / self.conf['no_models'])

        if (cid+1) == self.conf['no_models']:
            self.local_Xtrain = self.Xtrain_full.iloc[cid * data_len: ]
            self.local_Ytrain = self.Ytrain_full.iloc[cid * data_len: ]
        else:
            self.local_Xtrain = self.Xtrain_full.iloc[cid * data_len: (cid+1) *␣
↪data_len]
            self.local_Ytrain = self.Ytrain_full.iloc[cid * data_len: (cid+1) *␣
↪data_len]

    def local_train(self, global_model, global_epoch):

        if global_epoch==0:
            # first iteration, train fit on a local subsample to generate the␣
↪parameters base
            # it will let the global model converge faster than starting from␣
↪all zeros
            local_subsample_id = random.sample(  range(len(self.
↪local_Xtrain)),int(len(self.local_Xtrain)/5)    )
            self.local_model.fit(  self.local_Xtrain.iloc[local_subsample_id],␣
↪self.local_Ytrain.iloc[local_subsample_id]   )

            self.local_model.coef_ = np.zeros(27)
            self.local_model.intercept_ = 0

        else:
            # overwrite the local_model's coefficients by the global_models'
            self.local_model.coef_ = global_model.coef_

            self.local_model.intercept_ =  global_model.intercept_


        # get gradients of parameters in the linear regression model (using␣
↪cross-entropy as the loss function)
        ypred = self.local_model.predict(self.local_Xtrain)

        grad_coef = (1/len(ypred)) * self.local_Xtrain.T.dot(ypred - self.
↪local_Ytrain)
```

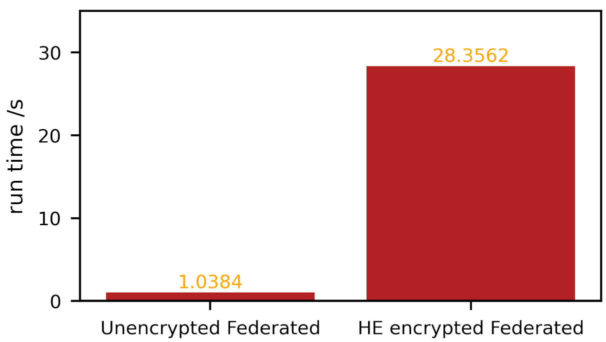
```
        grad_intercept = np.mean(ypred - self.local_Ytrain)

        enc_grad_coef = [public_key.encrypt(x) for x in grad_coef]
        enc_grad_intercept = public_key.encrypt(grad_intercept)

        return enc_grad_coef, enc_grad_intercept
```

[ ]:
```
prauc_scores = {"federated":[]}
loss_scores = {"federated":[]}
start = time.time()

if __name__ == '__main__':

    # generate a server and the client instances
    server = Server_homo(conf, Xtest, Ytest)
    clients = []

    for cid in range(conf["no_models"]):
        clients.append( Client_homo(conf, Xtrain_smote, Ytrain_smote, cid)  )

    print("Generated one Server and",conf["no_models"],"Clients...\n\n")

    # global iterations epochs
    for e in range(conf["global_epochs"]):

        gradients = {   "Beta_base":[],
                        "Intercept_base":[],
                        "gBetas":[],
                        "gIntercepts":[]
                    }
        if e==0:
            candidates = clients.copy()
        else:
            # in every epoch, just select k clients for federated training
            candidates = random.sample(clients, conf["k"])


        for c in candidates:
            coef_grads, itcp_grads = c.local_train(server.global_model, e)

            if e==0:
                gradients["Beta_base"].append(c.local_model.coef_)
                gradients["Intercept_base"].append(c.local_model.intercept_)


            gradients["gBetas"].append(coef_grads)
            gradients["gIntercepts"].append(itcp_grads)
```

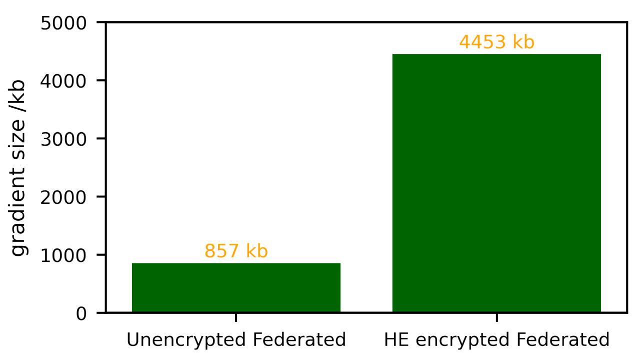
```
        # pass the gradients data to the server for aggregation.
        server.model_aggregate(gradients, e)

        pr_auc, loss = server.model_eval()

        print("Epoch %d, precision_recall score: %f, loss: %f" % (e, pr_auc,␣
 ↪loss))

        prauc_scores["federated"].append(pr_auc)
        loss_scores["federated"].append(loss)

end = time.time()
print("run time cost is", end-start, "seconds")
```

[30]:

```
# visualize running time
plt.figure(figsize=(3.5,2),dpi=500)

plt.xlim(0.5,2.5)
plt.ylim(0,35)
plt.xticks(range(1,3,1),  ["Unencrypted Federated","HE encrypted␣
 ↪Federated"],fontsize=7)
plt.yticks(fontsize=7)

plt.ylabel("run time /s",fontsize=8)

run_time = [1.0384, 28.3562]
plt.hlines(y=0,xmin=0.5,xmax=2.5,color='black',linewidth=1)
plt.bar( range(1,3,1),run_time,width=0.8,color='firebrick')


for i in range(1,3,1):
    plt.text(i,run_time[i-1]+0.
 ↪5,str(run_time[i-1]),ha="center",fontsize=7,color="orange")

plt.show()
```



[31]:
```
# visualize gradient size
plt.figure(figsize=(3.5,2),dpi=500)

plt.xlim(0.5,2.5)
plt.ylim(0,5000)
plt.xticks(range(1,3,1),  ["Unencrypted Federated","HE encrypted␣
↪Federated"],fontsize=7)
plt.yticks(fontsize=7)

plt.ylabel("gradient size /kb",fontsize=8)

gradient_size = [857, 4453]
plt.hlines(y=0,xmin=0.5,xmax=2.5,color='black',linewidth=1)
plt.bar( range(1,3,1),gradient_size,width=0.8,color='darkgreen')


for i in range(1,3,1):
    plt.text(i,gradient_size[i-1]+100, str(gradient_size[i-1])+"␣
↪kb",ha="center",fontsize=7,color="orange")

plt.show()
```



[ ]:

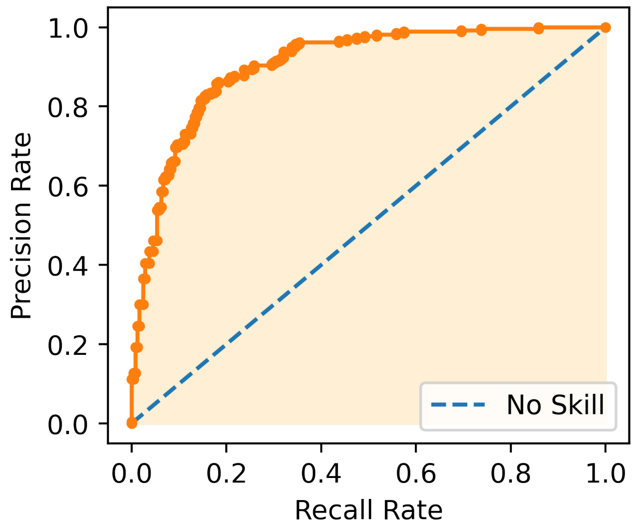
[ ]:

[ ]:

[32]:
```
# roc curve and auc
from sklearn.datasets import make_classification
from sklearn.linear_model import LogisticRegression
```

```python
from sklearn.model_selection import train_test_split
from sklearn.metrics import roc_curve
from sklearn.metrics import roc_auc_score
from matplotlib import pyplot
# generate 2 class dataset
X, y = make_classification(n_samples=1000, n_classes=2, random_state=1)
# split into train/test sets
trainX, testX, trainy, testy = train_test_split(X, y, test_size=0.5,␣
↪random_state=2)
# generate a no skill prediction (majority class)
ns_probs = [0 for _ in range(len(testy))]
# fit a model
model = LogisticRegression(solver='lbfgs')
model.fit(trainX, trainy)
# predict probabilities
lr_probs = model.predict_proba(testX)
# keep probabilities for the positive outcome only
lr_probs = lr_probs[:, 1]
# calculate scores
ns_auc = roc_auc_score(testy, ns_probs)
lr_auc = roc_auc_score(testy, lr_probs)
# summarize scores
print('No Skill: ROC AUC=%.3f' % (ns_auc))
print('Logistic: ROC AUC=%.3f' % (lr_auc))

plt.figure(figsize=(3.5,3),dpi=500)
# calculate roc curves
ns_fpr, ns_tpr, _ = roc_curve(testy, ns_probs)
lr_fpr, lr_tpr, _ = roc_curve(testy, lr_probs)

# plot the roc curve for the model
pyplot.plot(ns_fpr, ns_tpr, linestyle='--', label='No Skill')
pyplot.plot(lr_fpr, lr_tpr, marker='.')
# axis labels
pyplot.xlabel('Recall Rate')
pyplot.ylabel('Precision Rate')

plt.fill_between(lr_fpr,lr_tpr,color="papayawhip")

# show the legend
pyplot.legend(loc="lower right")
# show the plot
pyplot.show()
```

```
No Skill: ROC AUC=0.500
Logistic: ROC AUC=0.903
```


1.0
0.8
0.6
0.4
0.2
0.0
Precision Rate
0.0
0.2
0.4
0.6
0.8
1.0
Recall Rate
No Skill

[ ]:

[ ]: