西安交通大學
XI'AN JIAOTONG UNIVERSITY

机器学习 II 大作业

Give Me Some Credit

姓名：林可可

学号：2183521635

课程名称：机器学习 II

指导老师：林绍波

2021 年 7 月 13 日

目录

课程名称：　机器学习 II　　指导老师：　林绍波
实验名称：　机器学习 II-大作业　　实验类型：　大作业

一、　实验目的和要求

通过自选数据集完成数据清洗与数据描述性分析，并寻找最少三个算法处理该数据并说明优劣，而后撰写数据分析报告。

二、　实验内容和步骤

1.　实验内容

首先对数据的背景以及研究意义进行阐述，而后进行数据的描述性统计，再对不同的算法进行对比，而后分析数据结果。

2.　实验步骤

(1) Introduction：说明数据的背景及研究意义

(2) 描述性分析：说明对数据的直观感觉

(3) 算法描述：对不同的算法，优缺点进行描述并对比

(4) 数据分析结果

(5) 结论

三、　研究背景及意义

1.　背景介绍

本次大作业使用的数据集来自 kaggle 的公开数据集，Give Me Some Credit. 该数据集提供了贷款人的信用卡以及个人信贷的相关情况，包括每月的收入，开放信贷的数量和信用额度等相关指标。在了解这些指标之后，我们可以通过构建信用评分算法对贷款人违约概率进行预测，从而辅助银行进行是否应授予贷款的决策。

2.　研究意义

银行在市场经济中发挥着至关重要的作用。他们决定谁可以获得资金以及以什么条件获得资金，并且可以做出进行或终止投资的决策。为了让市场和社会的运作，个人和公司需要获得信贷。

考虑到数据集中提供了“个人经历了逾期 90 天的拖欠或者更糟糕的情况”（SeriousDlqin2yrs）这一指标，我们可以采取最先进的信用评分算法，通过预测某人在未来两年内遇到财务困境的可能性，来支持借款人做出最佳财务决策。

四、 描述性统计

1. 整体性描述

首先对数据进行整体的查看。Kaggle 提供的文件 Data Dictionary.xls 包含了每一列的基本信息以及相关的示意，通过查看发现一共有十一列，其中第一列是样本的标签也即上文提到的“SeriousDlqin2yrs”，其余十列为样本的其他属性。

0	Variable Name	Description	Type
1	SeriousDlqin2yrs	Person experienced 90 days past due delinquenc...	Y/N
2	RevolvingUtilizationOfUnsecuredLines	Total balance on credit cards and personal lin...	percentage
3	age	Age of borrower in years	integer
4	NumberOfTime30-59DaysPastDueNotWorse	Number of times borrower has been 30-59 days p...	integer
5	DebtRatio	Monthly debt payments, alimony,living costs di...	percentage
6	MonthlyIncome	Monthly income	real
7	NumberOfOpenCreditLinesAndLoans	Number of Open loans (installment like car loa...	integer
8	NumberOfTimes90DaysLate	Number of times borrower has been 90 days or m...	integer
9	NumberRealEstateLoansOrLines	Number of mortgage and real estate loans inclu...	integer
10	NumberOfTime60-89DaysPastDueNotWorse	Number of times borrower has been 60-89 days p...	integer
11	NumberOfDependents	Number of dependents in family excluding thems...	integer

图 1: 数据概述

而后分别对训练集以及测试集的样本的大小进行查看，结果如下所示，其中训练集共有 150000 条样本，每个样本有 11 列属性（第一列为样本的顺序），同样的，测试集共有 101503 条样本，每个样本也是 11 列属性。

(a) 训练集大小　　(b) 测试集大小

图 2: 数据集大小

2. 分属性描述

(1) SeriousDlqin2yrs 表示个人经历了逾期 90 天的拖欠或者更糟糕的情况，通过简单的对比条形图我们可以发现该属性有明显的类别不均衡现象，通过进一步统计具体的数值我们知道其中标签为 0 的样本个数有 139974 个，而标签为 1 的样本仅有 10026 个。

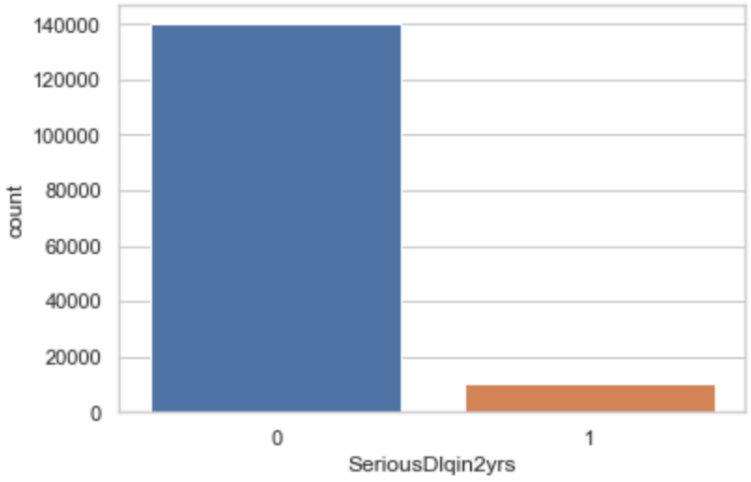


图 3: SeriousDlqin2yrs 分组条形图

(2) RevolvingUtilizationOfUnsecuredLines 表示信用卡和个人信贷的总余额，通过箱线图我们发现，该属性各个值之间的差异较大。

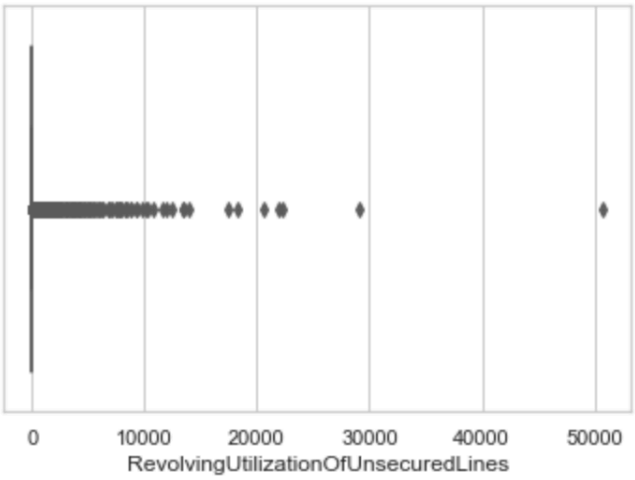
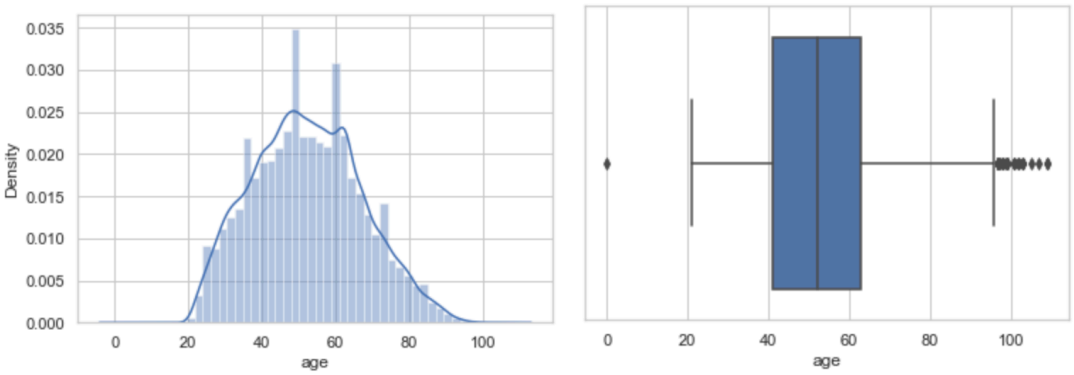
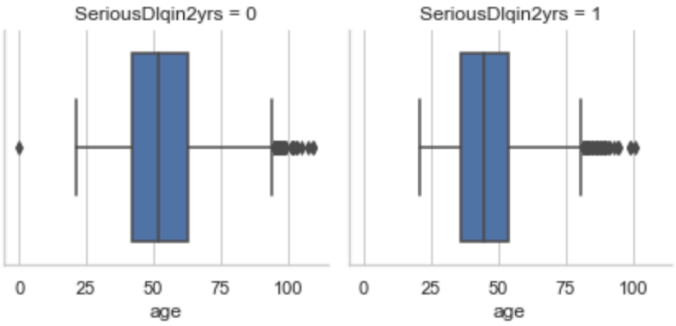


图 4: RevolvingUtilizationOfUnsecuredLines 箱线图

(3) age 表示贷款者的年龄，我们通过绘制直方图发现贷款者的年龄分布较为集中，其中 40-60 岁为贷款者年龄分布较多的区间，而后我们再次绘制了箱线图，发现确实是这样，落在箱线图区间外的 outliers 较少，年龄分布较为集中。而后我们结合 SeriousDlqin2yrs 绘制了分组箱线图，发现在两个类别下的贷款者年龄分布都比较集中，而没有逾期贷款的贷款者年龄稍微大一些略大于 50 岁，而有逾期贷款的贷款者的平均年龄略小于 50 岁，并且没有逾期贷款的贷款者年龄大于 100 岁的人数较多。



(a) 直方图　　(b) 箱线图



(c) 对比箱线图

图 5: Age

(4) NumberOfTime30-59DaysPastDueNotWorse 表示贷款人逾期 30-59 天的次数，我们根据数值进行了分布直方图的绘制，发现次数存在大于 96 次的情况，我们认为这是数据搜集时的偏误，因此我们设置 15 次为最大的逾期次数，即 15 次是最大的逾期次数。

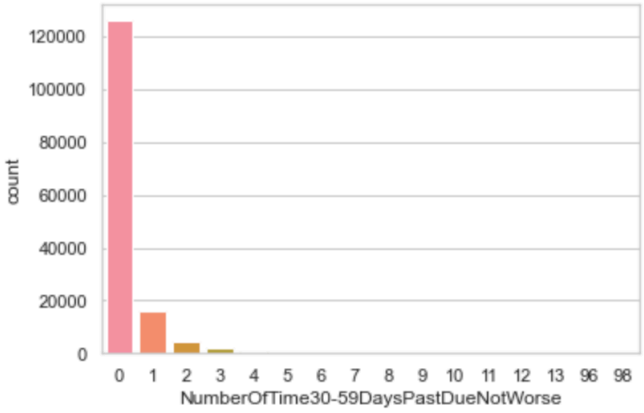


图 6: NumberOfTime30-59DaysPastDueNotWorse 直方图

(5) DebtRatio 指的是每月的债务支付占每月收入的比率，发现该属性样本之间的差距还是比较大的，并且没有明显的发现该属性于样本标签值之间存在明显的关系，后续可以考虑对该属性进行删除。

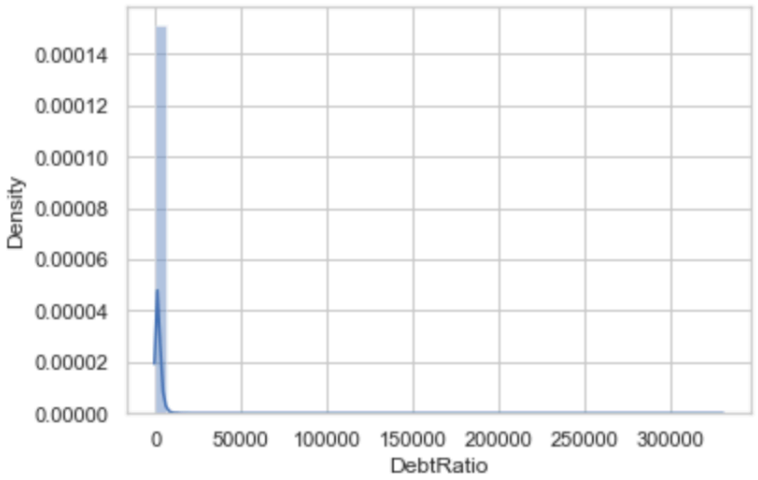


图 7: DebtRatio 直方图

(6) MonthlyIncome 指的是月收入，发现该属性存在较多的缺失值，我们通过筛选发现在训练集中存在 1699 条语句存在该属性的缺失，因此我们采用平均值对缺失值进行填补。

```
train.loc[train.MonthlyIncome.isnull(), 'SeriousDlqin2yrs'].sum()

#Fill none value with mean value
train['MonthlyIncome'] = train['MonthlyIncome'].fillna(train['MonthlyIncome'].mean())
```

(7) NumberOfOpenCreditLinesAndLoans 指的是未偿还的贷款和信用额度，我们通过绘制对比直方图查看不同标签下未偿还贷款的差异，发现有逾期贷款的贷款者在未偿还贷款的分布上也比较集中。

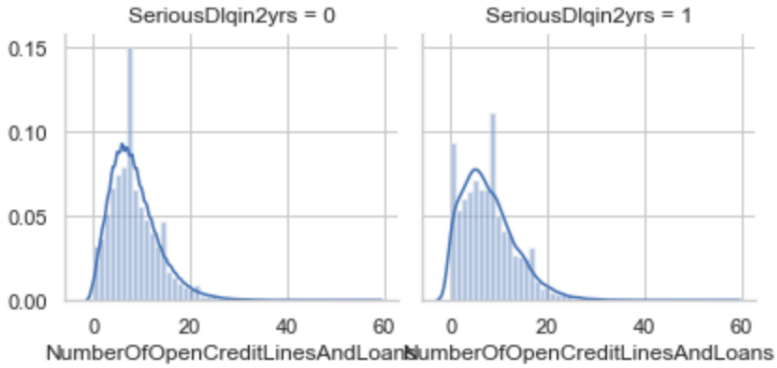


图 8: MonthlyIncome 直方图

(8) NumberOfTimes90DaysLate 表示的是贷款人逾期 90 天或者更长的时间次数，我们通过绘制对比数值统计图发现，该属性大的贷款者更容易成为目标样本。同时我们也可以看到一些数据上的问题，我们知道超过 90 天的次数应该小于等于逾期 60-89 天的人数，而数据中存在一些不满足的情况，因此同样的，我们将 15 设置为该属性的最大值，大于的 15 的值统一替换为 15。

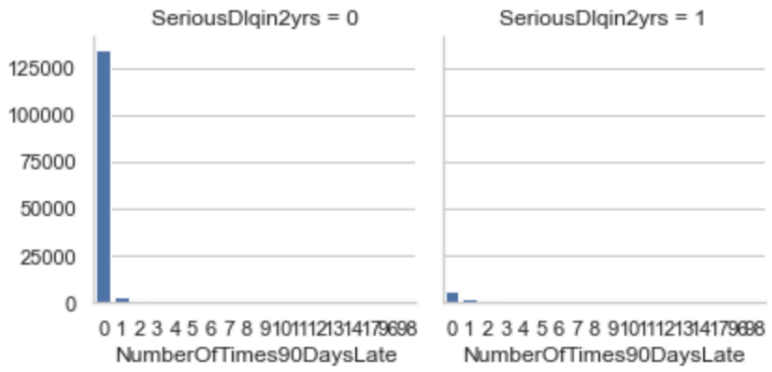


图 9: NumberOfTimes90DaysLate 直方图

(9) NumberRealEstateLoansOrLines 指的是抵押贷款和房地产贷款的数量，可以看出该属性的类别分布还是存在一定差异的。

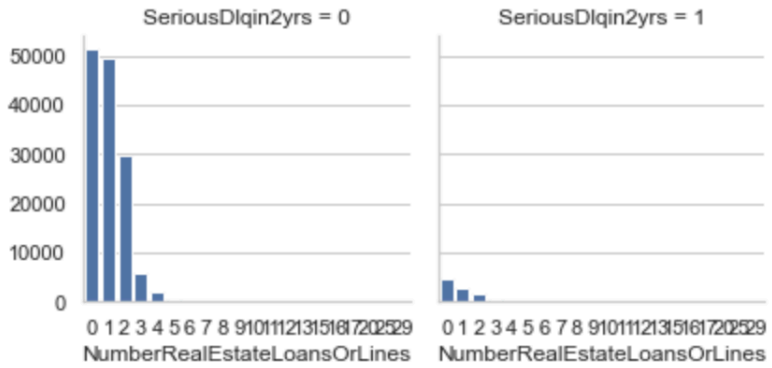


图 10: NumberRealEstateLoanOrLines 直方图

(10) NumberOfTime60-89DaysPastDueNotWorse 指的是在过去两年中，借款人逾期 60-89 天但没有恶化的次数，通过对比数值统计图，我们发现，改属性值大的贷款者更容易成为目标样本，并且类似的，我们可以发现一些数据上的问题，也即逾期 60-89 天的次数应该小于等于逾期 30-59 天的人数，而数据中有一些情况是不满足的，因此，我们将 15 设置为该属性的最大值，大于的 15 的值统一替换为 15。

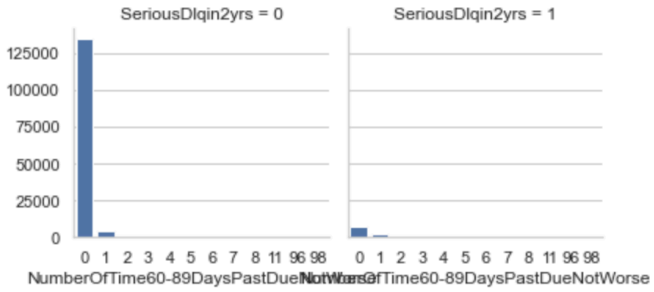


图 11: NumberOfTime60-89DaysPastDueNotWorse 直方图

(11) NumberOfDependents 指的是家庭成员中受抚养人的数量，通过绘制类别对比的箱线图我们发现类别之间的差异较大，也即更高的赡养负担可能会导致有更多的负债，因此也会有更高的风险，同时我们也发现该数据存在一定确实的情况，同样的我们用平均值填补了缺失。

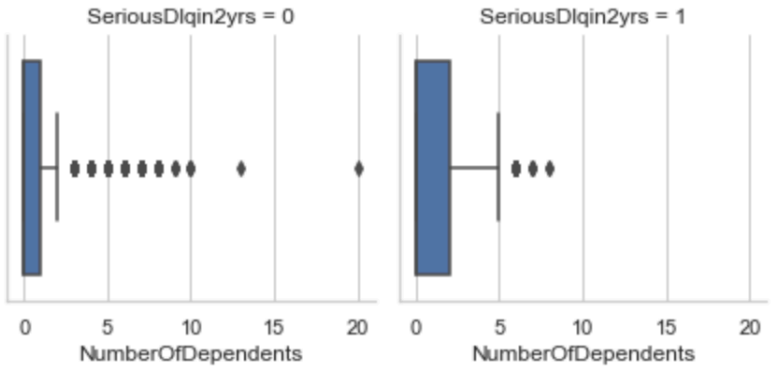


图 12: NumberOfDependdents 箱线图

3. 相关性分析

为了了解各个属性之间的相关度，我们选用热图来辨识每个属性之间的皮尔逊积相关系数，通过热图我们可以发现 SeriousDlqin2yrs 和几个逾期天数之间有很大的关系，但是 SeriousDlqin2yrs 和年龄的关系较小。

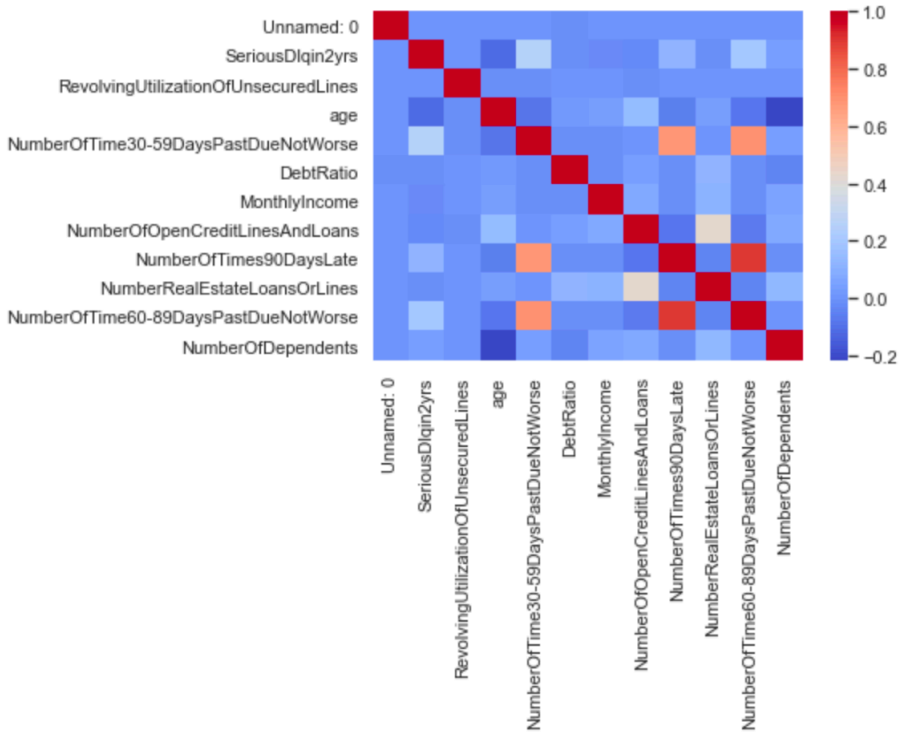


图 13: 相关系数图

五、 算法描述与结果分析

1. NaiveBayes

采用的第一个算法是朴素贝叶斯分类器。对于分类问题：已知集合: $C = \{y_1, y_2, \ldots, y_n\}$ 和 $I = \{x_1, x_2, \ldots, x_m, \ldots\}$ 确定映射规则 $y = f(x)$ 使得任意 $x_i \in I$ 有且仅有一个 $y_j \in C$ 使得 $y_j = f(x_i)$ 成立。

自然想到的就是贝叶斯分类器。而朴素贝叶斯是一种构建分类器的简单方法。该分类器模型会给问题实例分配用特征值表示的类标签，类标签取自有限集合。它不是训练这种分类器的单一算法，而是一系列基于相同原理的算法：所有朴素贝叶斯分类器都假定样本每个特征与其他特征都不相关。

朴素贝叶斯分类器的一个优势在于只需要根据少量的训练数据估计出必要的参数（变量的均值和方差）。由于变量独立假设，只需要估计各个变量的方法，而不需要确定整个协方差矩阵。

考虑到我们要处理的数据中许多属性是连续数据，故我们采用一般假设，认为这些连续数值为高斯分布。因此，对于训练集中某一个连续属性，x 。我们首先对数据根据类别分类，然后计算每个类别中 x 的均值和方差。令 μ_c 表示为 x 在 C 类上的均值，令 σ_c^2 为 x 在 C 类上的方差。在给定类中某个值的概率, $P(x = v \mid c)$, 可以通过将 v 表示为均值为 μ_c 方差为 σ_c^2 正态分布计算出来。也即, $P(x = v \mid c) = \frac{1}{\sqrt{2\pi\sigma_c^2}} e^{-\frac{(v-\mu_c)^2}{2\sigma_c^2}}$ 。但是在大量样本的情形下，另一种方式：离散化连续数值的方法表现更优，因为大量的样本可以学习到数据的分布。由于朴素贝叶斯是一种典型的用到大量样本的方法（越大计算量的模型可以产生越高的分类精确度），所以朴素贝叶斯方法都用到离散化方法，而不是概率分布估计的方法。

考虑到 sklearn 已经提供了相关的代码库，供我们直接调用，因此我们可以用如下的语句进行简单的训练：

```
from sklearn.naive_bayes import GaussianNB

gaussian = GaussianNB()
gaussian.fit(val_X, val_y)
y_pred = gaussian.predict_proba(val_X)[:,1]
score = roc_auc_score(val_y, y_pred)
print(score)

# 绘制ROC曲线
FPR_gaussian, TPR_gaussian, THRESH_gaussian = roc_curve(val_y, y_pred)
draw_roc(FPR_gaussian, TPR_gaussian)
```

得到的 ROC 曲线如下所示，ROC-AUC score 为 0.794。

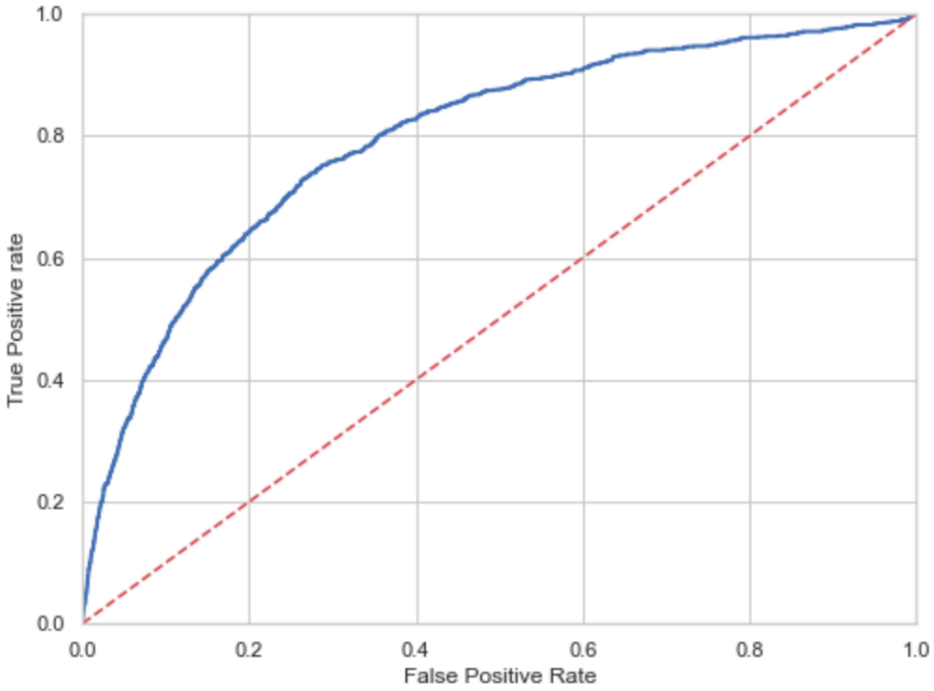


图 14: Naive Bayes

2. RandomForest

(1) Traditional-Random Forest

随机森林属于集成学习中 bagging 算法的延展,Random Forest(RF) 在以决策树作为基学习器构建 Bagging 集成的基础上，进一步在决策树的训练过程中加入了随机属性的选择。具体来说，传统决策树在选择划分属性时是在当前结点的所有候选属性（假定有 d 个）中选择一个最优属性；而在 RF 中，对基决策树的每个结点，先从该结点的候选属性集合中随机选择一个包含 k 个属性的子集，然后再从这个子集中选择一个最优属性用于划分。抽取的属性数 k 的选择比较重要，我们一般采用的是 $K = log_2 d$ 。由此，随机森林的基学习器的“多样性”不仅来自样本的扰动，还来自属性的扰动，使得最终集成的泛化能力进一步增强。

随机森林在训练时可以高度并行化，可以有效运行在大数据集上。但是取值划分比较多的特征容易对随机森林的决策产生更大的影响，从而影响拟合的模型效果。同时，随机森林由于对决策树候选划分属性的采样，这样在样本特征维度较高的时候，仍然可以高效的训练模型。但是在某些噪声比较大的样本集上，随机森林容易陷入过拟合。

同样的我们采用 sklearn 提供的库进行了测试：

```
from sklearn.ensemble import RandomForestClassifier

grid = GridSearchCV(
    estimator=RandomForestClassifier(),
    param_grid={
        'n_estimators':[30,50,80,100,200]
```

```
    },
    scoring='roc_auc',
    verbose=3
)

grid.fit(train_X, train_y)
# for result in grid.cv_results_:
    # print(result, grid.cv_results_[result])
grid.best_params_['n_estimators']
```

得到训练的结果如下所示，ROC-AUC 值为：0.850，训练时间为 416.1s。

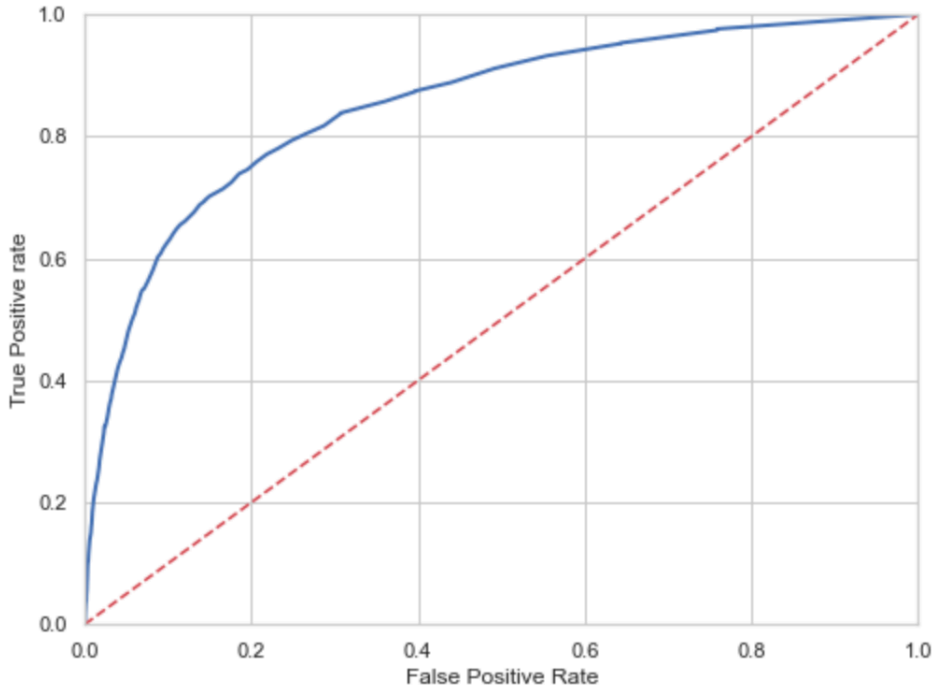


图 15: Random Forest

(2) UnderSampling - Random Forest

考虑到我们的数据存在类别不均衡的情况 (这在前文数据探索性分析时已经发现)，我们对传统的随机森林进行了降采样的处理，当训练数据集很大时，它可以通过减少训练数据样本的数量来帮助改善运行时间和存储问题。但是它可能会丢弃有用的信息，并且丢弃的信息对于构建规则分类器可能很重要。另外通过随机欠采样选择的样本可能是有偏差的样本。可能导致实际测试数据集的结果不准确。通过引入降采样的模块来跟踪值出现的次数，而后返回降采样后的数值，对重采样以后的数据划分训练集与测试集，然后利用随机森林对重采样之后的数据进行分类，训练过程的代码如下所示：

```
# 引入降采样模块
from imblearn.under_sampling import RandomUnderSampler
# Counter类的目的是用来跟踪值出现的次数
```

```
from collections import Counterx
print('Original dataset shape :', Counter(train_y))

# 调用模块
rus = RandomUnderSampler(random_state=111)

# 直接降采样后返回采样后的数值
X_resampled, y_resampled = rus.fit_resample(train_X, train_y)
print('Resampled dataset shape:', Counter(y_resampled))

# 划分训练集和测试集
from sklearn.model_selection import train_test_split
X_train_rus, X_test_rus, y_train_rus, y_test_rus = train_test_split (X_resampled,
    y_resampled, random_state=111)
X_train_rus.shape, y_train_rus.shape

# 对重采样以后的数据进行分类
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import roc_curve, roc_auc_score
import matplotlib.pyplot as plt
logit_resampled = LogisticRegression(random_state=111, solver='saga', penalty='l1',
    class_weight='balanced', C=1.0, max_iter=500)

logit_resampled.fit(X_resampled, y_resampled)
logit_resampled_proba_res = logit_resampled.predict_proba(X_resampled)
logit_resampled_scores = logit_resampled_proba_res[:, 1]
fpr_logit_resampled, tpr_logit_resampled, thresh_logit_resampled =
    roc_curve(y_resampled, logit_resampled_scores)
draw_roc(fpr_logit_resampled, tpr_logit_resampled)
print('AUC score: ', roc_auc_score(y_resampled, logit_resampled_scores))

# 采用随机森林法分类和梯度上升法
from sklearn.ensemble import RandomForestClassifier, GradientBoostingClassifier
forest = RandomForestClassifier(n_estimators=300, random_state=111, max_depth=5,
    class_weight='balanced')
forest.fit(X_train_rus, y_train_rus)
y_scores_prob = forest.predict_proba(X_train_rus)
y_scores = y_scores_prob[:, 1]
fpr, tpr, thresh = roc_curve(y_train_rus, y_scores)
draw_roc(fpr, tpr)
print('AUC score:', roc_auc_score(y_train_rus, y_scores))
```

得到训练的结果如下所示，ROC-AUC 值为：0.867, 训练时间为：16.7s。

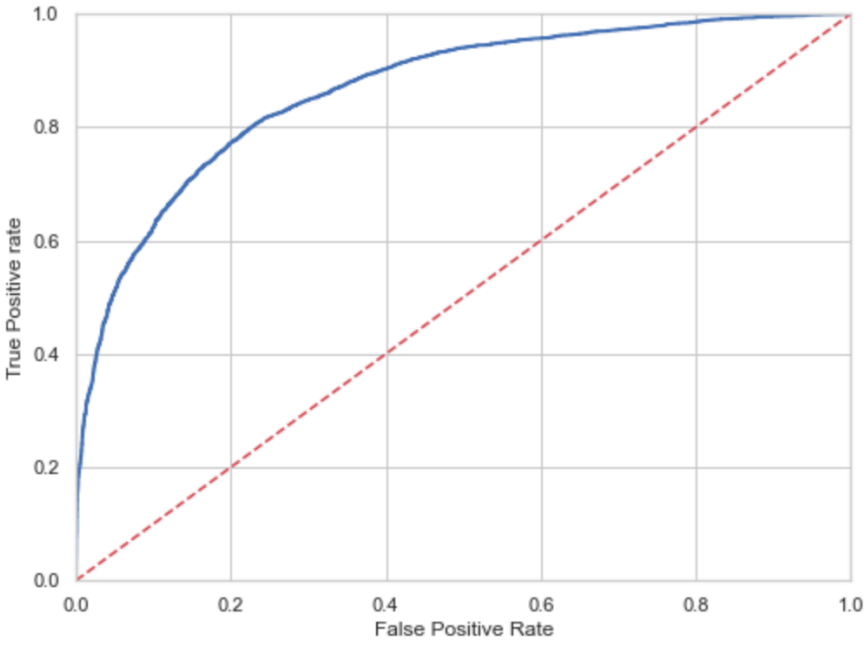


图 16: UnderSampling - Random Forest

可以发现训练时间相对于原来的 Random Forest 得到了明显的下降，并且 ROC-AUC 值得到了略微的提高，再综合训练时间和模型的表现效果来看，采用 undersampling 的 random forest 在该数据集上的效果是更优秀的。

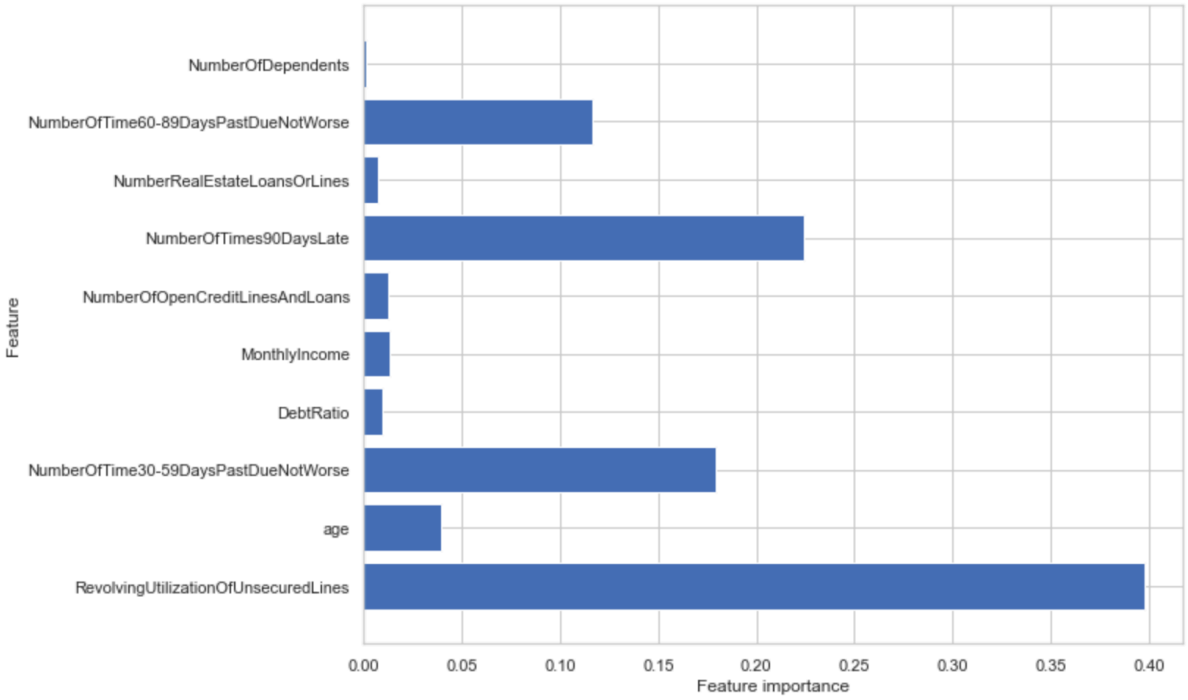


图 17: UnderSampling - Random Forest 对各个特征的重视度

而后我们可以查看 Under sampling 对于各个属性的重视程度，发现模型对于 RevolvingUtilizationOfUnsecuredLines 这一属性是最为看重的，这和我们在数据探索性分析步骤时作出的假设是相一致的，也符合我们的常规认知，也即信用卡和个人信贷余额的总余额越高的贷款者出现逾期信贷的可能性越低。

3. SVM

(1) SVM-RBF

我们尝试的第三个算法的带 RBF kernel 的 SVM 算法。RBF 核函数可以表示为 $k(x,z) = exp(-\rho d(x,z)); \rho > 0, d(x,z)$ 是任意距离的度量。采用高斯核函数按一定规律统一改变样本的特征数据得到新的样本，新的样本按新的特征数据能更好的分类，由于新的样本的特征数据与原始样本的特征数据呈一定规律的对应关系，因此根据新的样本的分布及分类情况，得出原始样本的分类情况。主要的方法是得到类似 $x \to \left(e^{-\gamma||x-l_1|^2|}, e^{-\gamma||x-l_2|^2}\right)$ 的映射，从而衡量样本和样本之间的 “相似度”，在一个刻画 “相似度” 的空间中，让同类样本更好的聚在一起，使得线性不可分的数据线性可分。

可以采用如下的程序实现：

```
from sklearn import svm
from sklearn import datasets
from sklearn.model_selection import train_test_split as ts



#split the data to 7:3
X_train,X_test,y_train,y_test = ts(train_X,train_y, test_size=0.1)

# select different type of kernel function and compare the score

# kernel = 'rbf'
begin = time.time()
clf_rbf = svm.SVC(kernel='rbf')
clf_rbf.fit(X_train,y_train)
score_rbf = clf_rbf.score(X_test,y_test)
print("The score of rbf is : %f"%score_rbf)
end = time.time()
print("training time:",end-begin,"s")
```

在运行之后发现采用 RBF kernel 需要运行的时间为 115.6s 得到的准确率为：0.931。我们发现运行时间还是较长的，并且我们已经运用 sklearn 内置的算法包但还是得到了较长的运行时间，因此我们希望能够寻求更高效的算法来提高模型的计算效率。

(2) FPC

在查阅相关文献后，我们发现了 FPC(Fast Polynomial Kernel Classification) 算法，在观察 FPC 算法的的设计流程时中，我们发现 FPC 它提供了一个不同的思路，以避免分类中的 maximal margin

thoery 。并且 FCP 使用多项式核的特殊特征，通过 ADMM 算法而不是 QP 来寻找分类器，而这样做在理论意义上减少了计算负担。我们通过观察 FPC 的计算流程就可以发现这一点：

Algorithm 1 Fast Polynomial kernel Classification(FPC)

Input: $Input: trainingsamples D := \{(x_i, y_i)\}_{i=1}^{m}$

Update:

1: **for** k=0,1, ..., **do**

2: $u^{k+1} = \left(\beta A^T A + \alpha \mathbf{I}_n\right)^{-1} \left(\alpha u^k + \beta A^T v^k - A^T w^k\right)$

3: $v^{k+1} = Hinge_{m\beta}\left(y, Au^{k+1} + \beta^{-1} w^k\right)$

4: $w^{k+1} = w^k + \beta\left(Au^{k+1} - v^{k+1}\right)$

End: the first k satisfying $\left\|p^{k+1} - p^k\right\|_H^2 <$ tol with H

Output: $f_{D,s}(\cdot) = \sum_{j=1}^{n} u_j^{output} K_s\left(\eta_j, \cdot\right)$

可以发现 FPC 的优点是减少可调参数，配合以有十分简洁直观的用户友好的设计流程，以及上文提到的 ADMM 算法降低计算负担。同时阅读论文发现其给出的最优泛化误差保证。

于是我们通过运行如下程序，

```
%%%FPC for UCI data: musk2
close all;clear all; clc;
%warning off all;
% UCI dataset experiments
function [beta, trainerr , testerr , traintime , testtime]=FPC(xtr,ytr,xte,yte,s,cx)
%%% Input %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%% xtr        -- inputs of training samples
%%% ytr        -- labels of training samples
%%% xte        -- inputs of testing samples
%%% yte        -- labels of testing samples
%%% s          -- algorithmic parameter: the order of the used polynomial (generally,
    an integer less than 10)
%%% cx         -- algorithmic parameter: center points in X-space (generally, taking the
    center points as the first nc columns of the polynomial kernel matrix)
%%% gamma      -- augmented lagrangian parameter (default:1)
%%% alpha      -- proximal parameter (defualt: 1), mainly to overcome the
    ill-condedness of the induced kernel matrix and improve the numerical stability
%%% MaxIter    -- the default number of maximal iterations: 5
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%%% Output %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%% beta       -- the parameters of model
%%% testerr    -- test error ( misclassification rate)
%%% trainerr   -- training error ( misclassification rate)
%%% testtime   -- test time (seconds)
%%% traintime  -- training time (seconds)
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%



start_cpu_time = cputime;
% the default parameters for FPC
alpha = 1; % proximal parameter
gamma = 1; % the augmented parameter
MaxIter = 50; % the maximal iterations

ATrain = PolynomialKerMat(xtr,cx,s); % the associated matrix via polynomial kernel using
    training samples
ATest = PolynomialKerMat(xte,cx,s); % the associated matrix via polynomial kernel using
    test samples
[m,n]=size(ATrain);

% calculate the inverse and restore
tempA = (gamma*(ATrain'*ATrain)+alpha*eye(n))\eye(n);

% initialization
u0=zeros(n,1);
v0=ytr;
w0=zeros(m,1);

 iter = 1;
while iter <=MaxIter
    ut = tempA*(alpha*u0+ATrain'*(gamma*v0-w0)); % update ut
    vt = hinge_prox(ytr,ATrain*ut+gamma^(-1)*w0,m*gamma); % update vt
    wt = w0+gamma*(ATrain*ut-vt); % update multiplier wt

    u0 = ut;
    v0 = vt;
    w0 = wt;
     iter = iter +1;
end
end_cpu_time = cputime;
 traintime = end_cpu_time - start_cpu_time; % calculating the training time

beta = u0;
 trainerr = sum((ytr~=mysign(ATrain*u0)))/size(ytr,1); % calculating test error

start_test_time = cputime;
 testerr = sum((yte~=mysign(ATest*u0)))/size(yte,1); % calculating test error
```

```
end_test_time = cputime;
testtime = end_test_time - start_test_time;
end


%%% Constructing a matrix given the center points of kernel x, coefficients c and the
    order of polynomial kernel s.
function A = PolynomialKerMat(x,c,s)
% c: Polynomial kernel coefficient
% s: the order of Polynomial kernel
A = (ones(size(x,1),size(c,1))+x*c').^s;
end

function tempsign = mysign(u)
tempsign = (u>=0)-(u<0);
end

function z = hinge_prox(a,b,gamma)
% hinge_prox(a,b,gamma) = argmin_u max{0,1-a*u} + gamma/2 (u-b)^2
% a: m*1 vector
% b: m*1 vector
% gamma>0: parameter
% z: output of the proximal of hinge
% m = size(a,1);
tol = 1e-10;
z = b.*(a==0)+(b+gamma^(-1)*a).*(a~=0&a.*b<=1-gamma^(-1)*a.*a)+...
    (a~=0&a.*b>1-gamma^(-1)*a.*a&a.*b<1).*((a+tol).^(-1))+b.*(a~=0&a.*b>=1);
end

%% Load dataset and data processing
%load('testpro2.mat');
% normalization: input [0,1], output {-1,1}
dataset = csvread('trainpro2.csv'); % file name of the dataset
% total number of samples 6598*167
% 2-class classification
% the last column is the target
% 50% for training, 25% for validation, 25% for test
% TrainDataSize = 3299;
% ValidDataSize = 1649
% TestDataSize = 1650;

% no. of negative labels : 5581
% no. of poisitve labels : 1017
```

```
107
108 %% data normalization:
109 %%%% input [0,1]
110 for i=1:size(dataset,2)-1
111     if max(dataset(:,i))~=min(dataset(:,i))
112
113         dataset(:,i)=((dataset(:,i)-min(dataset(:,i)))/(max(dataset(:,i))-min(dataset(:,i))));
114     else
115         dataset(:,i)=0;
116     end
117 end
118
119 % output {-1,1}
120 T=dataset(:,size(dataset,2));
121 if max(T)~=min(T)
122     T=2*(T-min(T))/(max(T)-min(T))-1;
123 else
124     T=ones(size(T))*0.5;
125 end
126 dataset(:,size(dataset,2))=T;
127
128 %% Generate the training and test samples
129 [DataSize,d] = size(dataset);
130 m0 = 95000; % number of training samples
131 m1 = 37500; % number of validation samples
132 m2 = DataSize-m0-m1; % Number of Test samples
133
134 trail = 2; % 20 trails
135 trainerr = zeros(trail,1); % recording training error
136 testerr = zeros(trail,1); % recording test error
137 traintime = zeros(trail,1); % recording training time
138 testtime = zeros(trail,1); % recording test time
139 best_s = zeros(trail,1); % recording best polynomial degree s
140 best_nc = zeros(trail,1); % recording best number of centers, i.e. nc = (s+d,s)
141
142 for i=1:trail
143     %% Generate the training and test samples
144     templ = randperm(DataSize);
145     TrainI = templ(1:m0); % index set of training samples
146     ValidI = templ(m0+1:m0+m1); % index set of validation samples
147     TestI = templ(m0+m1+1:DataSize); % index set of testing samples
148     % Training samples
149     xtr = dataset(TrainI,1:d-1);
```

```
150     ytr = dataset(TrainI,d);
151     % Validation samples
152     xvalid = dataset(ValidI,1:d-1);
153     yvalid = dataset(ValidI,d);
154     % Test samples
155     xte = dataset(TestI,1:d-1);
156     yte = dataset(TestI,d);
157
158     max_s = min(ceil((m0/log(m0))^(1/(d-1))),10); % the maximal bound for s in theory,
            max_s = 2
159
160     %% training the best parameters via validation
161     s = (1:max_s)'; % the candidates of s
162     ValidErr = zeros(max_s,1); % validation error for each s
163     temp_TrainTime = zeros(max_s,1); % training time for each s
164     for j=1:max_s
165         nc = min(nchoosek(s(j)+d-1,d-1),m0); % number of centers
166         cx = xtr(1:nc,:); % strategy 1
167
168         [~,~, ValidErr(j),temp_TrainTime(j),~] = FPC(xtr,ytr,xvalid,yvalid,s(j),cx);
169         fprintf('trail:%d, s:%d,validerr:%f\n',i, s(j), ValidErr(j));
170     end
171     traintime(i) = sum(temp_TrainTime); % total training time
172
173     bestj = find(ValidErr==min(ValidErr));
174     best_s(i) = s(bestj(1)); % the best degree of polynomial parameters
175
176
177     %% training and testing using the best parameter obtained via validation
178     best_nc(i) = min(nchoosek(best_s(i)+d-1,d-1),m0); % the best number of centers
179     best_cx = xtr(1:best_nc(i),:); %the best centers
180     [beta, trainerr(i), testerr(i),~, testtime(i)]=FPC(xtr,ytr,xte,yte,best_s(i),best_cx);
181
182     fprintf('trail:%d, trainerr:%f, testerr:%f, traintime:%f, testtime:%f\n', ...
183         i, trainerr(i), testerr(i), traintime(i), testtime(i));
184 end
185 fprintf('testerr:%f,trainerr:%f\n',(1-mean(testerr)),(1-mean(trainerr)));
186
187 fprintf('stdoftesterr:%f\n',std(testerr));
188
189 fprintf('traintime:%f, testtime:%f\n',mean(traintime),mean(testtime));
190
191 fprintf('best_s:%f, best_nc:%f\n',mean(best_s), mean(best_nc));
```

得到的 training time 为 5.49s，相比于 SVM- RBF 确实显著降低了运行时间，但是得到的准确率仅有 0.810，这里认为主要是 poly-kernel 并不太适合处理不平衡且样本个数较多的数据集，但是我们可以从运行时间看到，运行时间是显著提高的，也说明 FPC 算法的对于 SVM 的加速是起到作用的。

六、 结果分析与对比

通过上述三种算法与其对应变形算法的尝试，我们将模型的训练结果进行了简单的对比，如下所示：

表 1: 训练时间与训练结果

算法	Naive Bayes	Traditional-RF	UnderSampling-RF	SVM-RBF	FPC
准确率	0.794(AUC)	0.850(AUC)	0.867(AUC)	0.931(Accuracy)	0.810(Accuracy)
运行时间	**0.7s**	416.1s	16.7s	115.6s	5.49s
Software	Python 3.8	Python 3.8	Python 3.8	Python 3.8	Matlab 2020b

可以发现在都只是使用简单的 CPU 的情况下,Naive Bayes Bayes 的运行时间是最短的，仅用了 0.7s, 但是在准确率上的表现并不尽如人意。同时也可以发现 RF 算法的改进大大缩短了 RF 算法的运行时间，同时也能够在一定程度上提高模型的准确率。在 SVM 算法的表现上来看，带有 RBF-kernel 的 SVM 算法更适用于解决我们的问题，可以达到 0.931 的准确率，Poly-kernel 在此数据集上的表现并不优秀，主要的原因可能是因为数据集的样本标签是不均衡的，而且数据集本身的样本数较大，特征数相对于样本数来说较小。但是我们可以发现 FPC 算法能够在很大程度上降低 SVM 算法的运行时间，并且还是在使用了较为重量的 Matlab 的情况下。

通过上述实验，我们认为不同的算法都某些角度都能为我们的问题提供一些帮助，但是结合准确率和运行时间来看，UnderSampling 的 Random Forest 以及 FPC 都具有较为不错，也即在权衡运行时间和准确率之后折衷的表现。总体而言，相较于原始的算法，两个改进的算法都具有明显提高的地方。如果不计运行时间的话，只是追求模型的准确率，RBF kernel 的 SVM 算法是实验中所有算法里表现最优的。

参考文献

[1] L. Breiman, Random forests, Machnie Learning, 45: 5-32, 2001.

[2] D. X. Zhou, and K. Jetter, Approximation with polynomial kernels and SVM classifiers, Adv. Comput. Math., 25: 323-344, 2006.

[3] Zeng J, Wu M, Lin S B, et al. Fast polynomial kernel classification for massive data[J]. arXiv preprint arXiv:1911.10558, 2019.