Loan_Payback

September 6, 2020

0.1 Kaushal Rao - Predicting Loan Payback with Decision Tree & Random Forest

0.1.1 In this notebook, we will try to predict which people have a high probability of paying an investor back with the Decision Tree and Random Forest statistical models. We will be exploring publicly available lending data from 2007-2010 (can be found on LendingClub.com).

Here are what the columns of the dataset represent: * credit.policy: 1 if the customer meets the credit underwriting criteria of LendingClub.com, and 0 otherwise. * purpose: The purpose of the loan (takes values "credit_card", "debt_consolidation", "educational", "major_purchase", "small_business", and "all_other"). * int.rate: The interest rate of the loan, as a proportion (a rate of 11% would be stored as 0.11). Borrowers judged by LendingClub.com to be more risky are assigned higher interest rates. * installment: The monthly installments owed by the borrower if the loan is funded. * log.annual.inc: The natural log of the self-reported annual income of the borrower. * dti: The debt-to-income ratio of the borrower (amount of debt divided by annual income). * fico: The FICO credit score of the borrower. * days.with.cr.line: The number of days the borrower has had a credit line. * revol.bal: The borrower's revolving balance (amount unpaid at the end of the credit card billing cycle). * revol.util: The borrower's revolving line utilization rate (the amount of the credit line used relative to total credit available). * inq.last.6mths: The borrower's number of inquiries by creditors in the last 6 months. * delinq.2yrs: The number of times the borrower had been 30+ days past due on a payment in the past 2 years. * pub.rec: The borrower's number of derogatory public records (bankruptcy filings, tax liens, or judgments).

0.1.2 Importing

```
In [1]: import pandas as pd
        import numpy as np
        import matplotlib.pyplot as plt
        import seaborn as sns
        %matplotlib inline
        from sklearn.model_selection import train_test_split
        from sklearn.tree import DecisionTreeClassifier
        from sklearn.ensemble import RandomForestClassifier
        from sklearn.metrics import classification_report,confusion_matrix
```

```
In [2]: loans = pd.read_csv('loan_data.csv')
```

```
In [3]: loans.describe()
```

Out[3]:

	credit.policy	int.rate	installment	log.annual.inc	dti
count	9578.000000	9578.000000	9578.000000	9578.000000	9578.000000
mean	0.804970	0.122640	319.089413	10.932117	12.606679
std	0.396245	0.026847	207.071301	0.614813	6.883970
min	0.000000	0.060000	15.670000	7.547502	0.000000
25%	1.000000	0.103900	163.770000	10.558414	7.212500
50%	1.000000	0.122100	268.950000	10.928884	12.665000
75%	1.000000	0.140700	432.762500	11.291293	17.950000
max	1.000000	0.216400	940.140000	14.528354	29.960000

	fico	days.with.cr.line	revol.bal	revol.util
count	9578.000000	9578.000000	9.578000e+03	9578.000000
mean	710.846314	4560.767197	1.691396e+04	46.799236
std	37.970537	2496.930377	3.375619e+04	29.014417
min	612.000000	178.958333	0.000000e+00	0.000000
25%	682.000000	2820.000000	3.187000e+03	22.600000
50%	707.000000	4139.958333	8.596000e+03	46.300000
75%	737.000000	5730.000000	1.824950e+04	70.900000
max	827.000000	17639.958330	1.207359e+06	119.000000

	inq.last.6mths	delinq.2yrs	pub.rec	not.fully.paid
count	9578.000000	9578.000000	9578.000000	9578.000000
mean	1.577469	0.163708	0.062122	0.160054
std	2.200245	0.546215	0.262126	0.366676
min	0.000000	0.000000	0.000000	0.000000
25%	0.000000	0.000000	0.000000	0.000000
50%	1.000000	0.000000	0.000000	0.000000
75%	2.000000	0.000000	0.000000	0.000000
max	33.000000	13.000000	5.000000	1.000000

In [4]:
```
loans.head()
```

Out[4]:

	credit.policy	purpose	int.rate	installment	log.annual.inc
0	1	debt_consolidation	0.1189	829.10	11.350407
1	1	credit_card	0.1071	228.22	11.082143
2	1	debt_consolidation	0.1357	366.86	10.373491
3	1	debt_consolidation	0.1008	162.34	11.350407
4	1	credit_card	0.1426	102.92	11.299732

	dti	fico	days.with.cr.line	revol.bal	revol.util	inq.last.6mths
0	19.48	737	5639.958333	28854	52.1	0
1	14.29	707	2760.000000	33623	76.7	0
2	11.63	682	4710.000000	3511	25.6	1
3	8.10	712	2699.958333	33667	73.2	1
4	14.97	667	4066.000000	4740	39.5	0

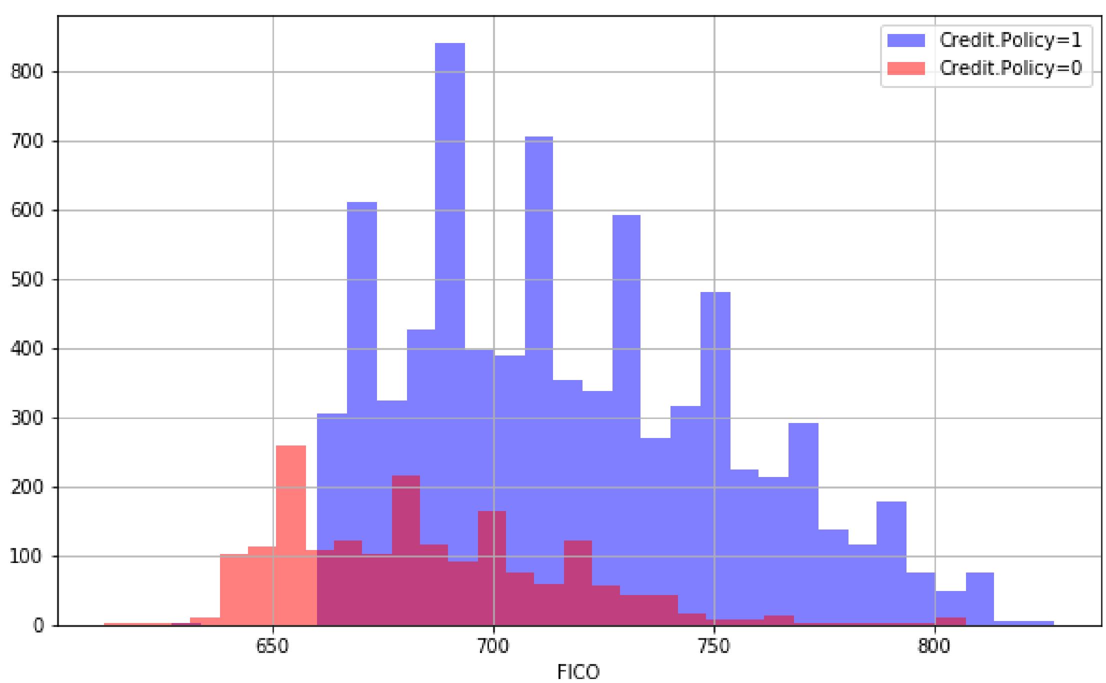
	delinq.2yrs	pub.rec	not.fully.paid
0	0	0	0

1	0	0	0
2	0	0	0
3	0	0	0
4	1	0	0

0.1.3 Exploratory Data Analysis (EDA)

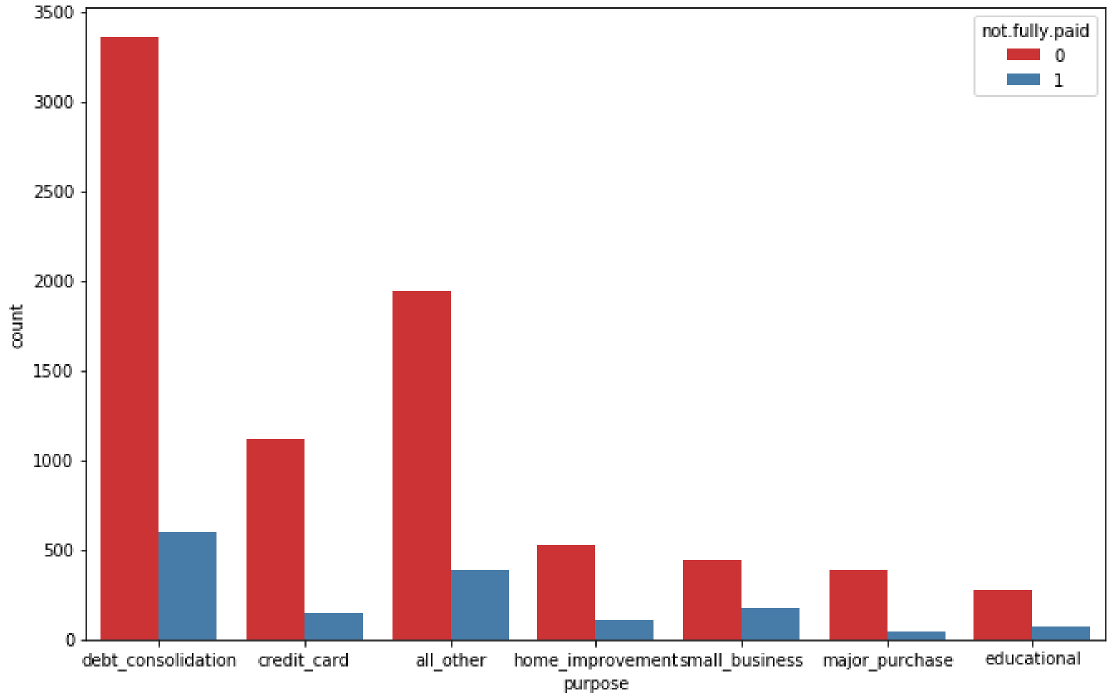
In [5]:
```
plt.figure(figsize=(10,6))
loans[loans['credit.policy']==1]['fico'].hist(alpha=0.5,color='blue',
                                              bins=30,label='Credit.Policy=1')
loans[loans['credit.policy']==0]['fico'].hist(alpha=0.5,color='red',
                                              bins=30,label='Credit.Policy=0')
plt.legend()
plt.xlabel('FICO')
# creating histogram of the two FICO distributions, one for each credit policy outcome
# we see that those that have met the credit requirements generally have a higher FICO
```

Out[5]:
```
Text(0.5, 0, 'FICO')
```

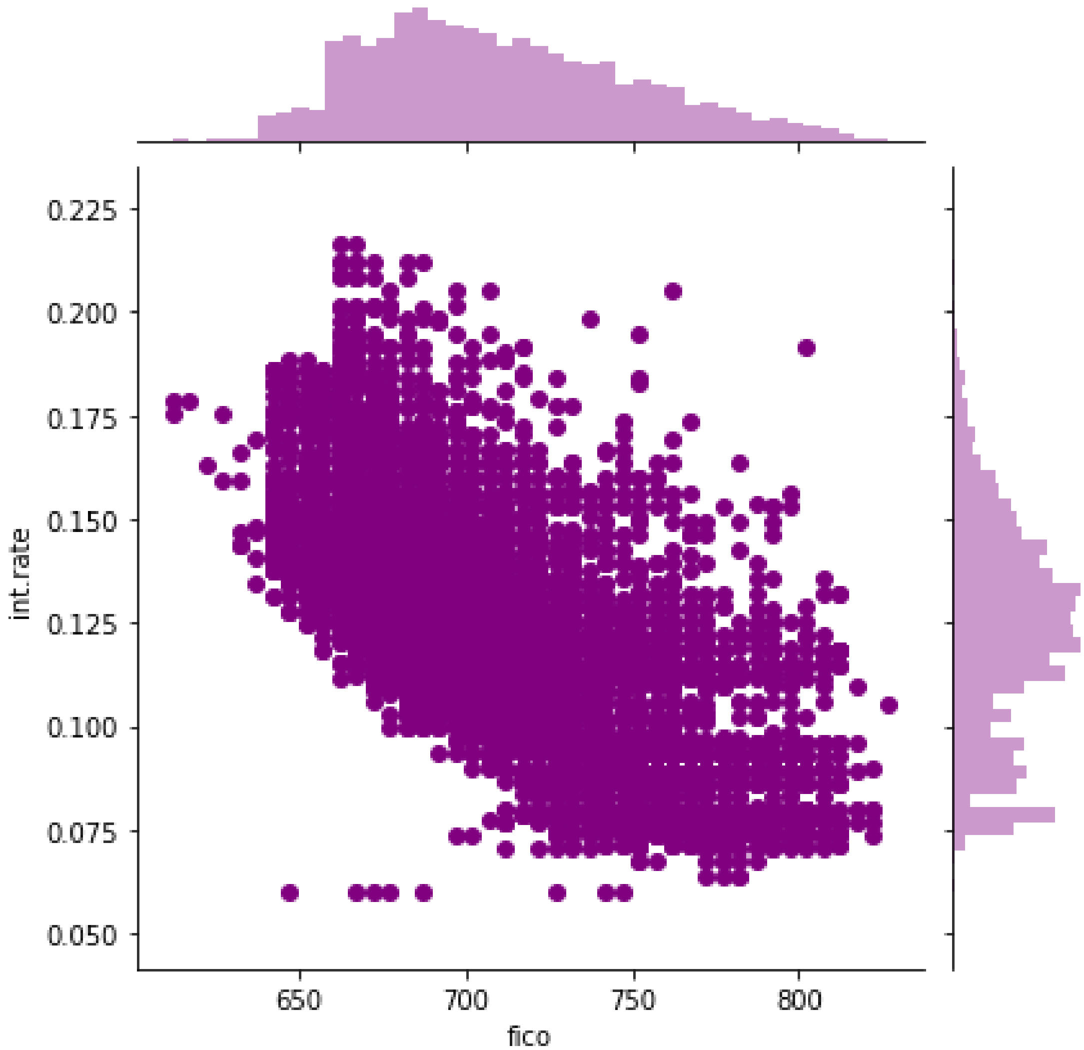


In [6]:
```
plt.figure(figsize=(11,7))
sns.countplot(x='purpose',hue='not.fully.paid',data=loans,palette='Set1')
# countplot showing the counts of loans by purpose
```

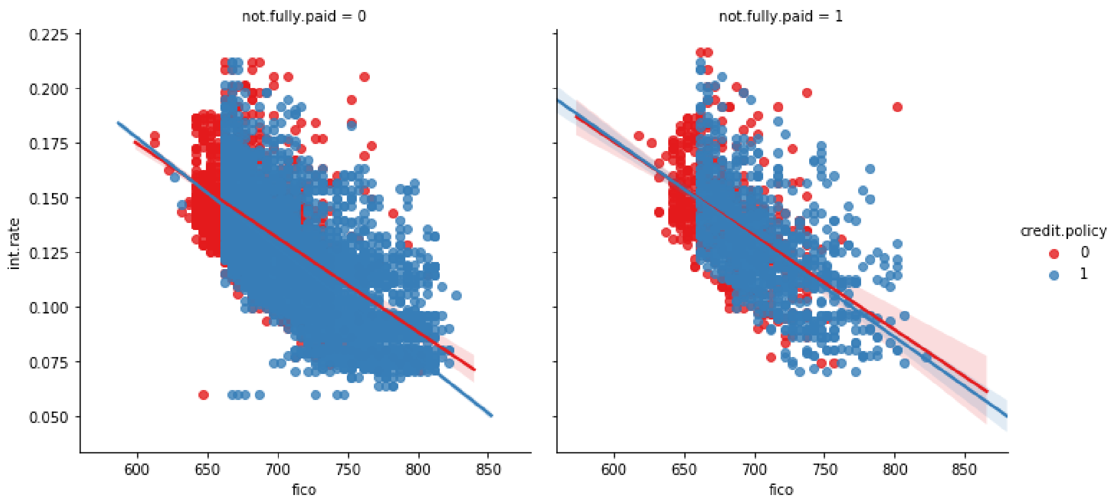
Out[6]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x103f012e8>
```




```
In [7]: sns.jointplot(x='fico',y='int.rate',data=loans,color='purple')
        # jointplot between FICO scores and interest rate
        # generally inverse correlation

Out[7]: <seaborn.axisgrid.JointGrid at 0x1a182c45c0>
```




```
In [8]: plt.figure(figsize=(11,7))
        sns.lmplot(y='int.rate',x='fico',data=loans,hue='credit.policy',
                   col='not.fully.paid',palette='Set1')
        # linear model plots to see trends in FICO scores based on "fully paid" and credit pol

Out[8]: <seaborn.axisgrid.FacetGrid at 0x1a1858aba8>

<Figure size 792x504 with 0 Axes>
```



0.1.4 Setting up the Data

```
In [9]: loans.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 9578 entries, 0 to 9577
Data columns (total 14 columns):
credit.policy        9578 non-null int64
purpose              9578 non-null object
int.rate             9578 non-null float64
installment          9578 non-null float64
log.annual.inc       9578 non-null float64
dti                  9578 non-null float64
fico                 9578 non-null int64
days.with.cr.line    9578 non-null float64
revol.bal            9578 non-null int64
revol.util           9578 non-null float64
inq.last.6mths       9578 non-null int64
delinq.2yrs          9578 non-null int64
pub.rec              9578 non-null int64
not.fully.paid       9578 non-null int64
dtypes: float64(6), int64(7), object(1)
memory usage: 1.0+ MB
```

Categorical Features

```
In [10]: # since the "purpose" column is categorical, we need to transform it
         # we can use dummy variables so sklearn is able to understand it as a feature
         cat_feats = ['purpose']
```

```
final_data = pd.get_dummies(loans,columns=cat_feats,drop_first=True)
final_data.info()
# we can see that the purpose binary columns have been added
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 9578 entries, 0 to 9577
Data columns (total 19 columns):
credit.policy                 9578 non-null int64
int.rate                      9578 non-null float64
installment                   9578 non-null float64
log.annual.inc                9578 non-null float64
dti                           9578 non-null float64
fico                          9578 non-null int64
days.with.cr.line             9578 non-null float64
revol.bal                     9578 non-null int64
revol.util                    9578 non-null float64
inq.last.6mths                9578 non-null int64
delinq.2yrs                   9578 non-null int64
pub.rec                       9578 non-null int64
not.fully.paid                9578 non-null int64
purpose_credit_card           9578 non-null uint8
purpose_debt_consolidation    9578 non-null uint8
purpose_educational           9578 non-null uint8
purpose_home_improvement      9578 non-null uint8
purpose_major_purchase        9578 non-null uint8
purpose_small_business        9578 non-null uint8
dtypes: float64(6), int64(7), uint8(6)
memory usage: 1.0 MB
```

Train-Test Split

```
In [11]: X = final_data.drop('not.fully.paid',axis=1)
         y = final_data['not.fully.paid']
         X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.30, random_state
```

0.1.5 Decision Tree

Training

```
In [12]: dtree = DecisionTreeClassifier()
         dtree.fit(X_train,y_train)
```

```
Out[12]: DecisionTreeClassifier(class_weight=None, criterion='gini', max_depth=None,
                     max_features=None, max_leaf_nodes=None,
                     min_impurity_decrease=0.0, min_impurity_split=None,
                     min_samples_leaf=1, min_samples_split=2,
                     min_weight_fraction_leaf=0.0, presort=False, random_state=None,
                     splitter='best')
```

Predictions & Evaluation

```
In [13]: predictions = dtree.predict(X_test)
```

```
In [14]: print(classification_report(y_test,predictions))
         # not great for predicting class 1!
```

```
              precision    recall  f1-score   support

           0       0.86      0.82      0.84      2431
           1       0.20      0.24      0.22       443

   micro avg       0.73      0.73      0.73      2874
   macro avg       0.53      0.53      0.53      2874
weighted avg       0.75      0.73      0.74      2874
```

```
In [15]: print(confusion_matrix(y_test,predictions))
```

```
[[2001  430]
 [ 336  107]]
```

0.1.6 Random Forest

Training

```
In [16]: rfc = RandomForestClassifier(n_estimators=600)
         rfc.fit(X_train,y_train)
```

```
Out[16]: RandomForestClassifier(bootstrap=True, class_weight=None, criterion='gini',
                     max_depth=None, max_features='auto', max_leaf_nodes=None,
                     min_impurity_decrease=0.0, min_impurity_split=None,
                     min_samples_leaf=1, min_samples_split=2,
                     min_weight_fraction_leaf=0.0, n_estimators=600, n_jobs=None,
                     oob_score=False, random_state=None, verbose=0,
                     warm_start=False)
```

Predictions & Evaluation

```
In [17]: predictions = rfc.predict(X_test)
```

```
In [18]: print(classification_report(y_test,predictions))
         # better than decision tree but still not great for class 1
```

```
              precision    recall  f1-score   support

           0       0.85      1.00      0.92      2431
           1       0.53      0.02      0.04       443
```

```
   micro avg       0.85      0.85      0.85      2874
   macro avg       0.69      0.51      0.48      2874
weighted avg       0.80      0.85      0.78      2874
```

In [19]:
```python
print(confusion_matrix(y_test,predictions))
# recall was not great for predicting class 1, more feature engineering is needed
```

```
[[2422    9]
 [ 433   10]]
```