Decision Trees and Random Forest

January 8, 2021

0.1 Decision Tree and Random Forest

[1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline
```

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

[3]:
```
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
```

[4]:
```
loans.describe()
```

[4]:
```
       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
```

[5]:
```
loans.head()
```

[5]:
```
   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
```

0.1.1 Exploratory Data Analysis

Histogram of two FICO distributions, one for each credit.policy outcome.

[6]:
```
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')
```

[6]:
```
<matplotlib.text.Text at 0x119963f28>
```

Histogram of two FICO distributions, one for each not.fully.paid outcome.

[7]:
```
plt.figure(figsize=(10,6))
loans[loans['not.fully.paid']==1]['fico'].hist(alpha=0.5,color='blue',
                                              bins=30,label='not.fully.paid=1')
loans[loans['not.fully.paid']==0]['fico'].hist(alpha=0.5,color='red',
                                              bins=30,label='not.fully.paid=0')
plt.legend()
```

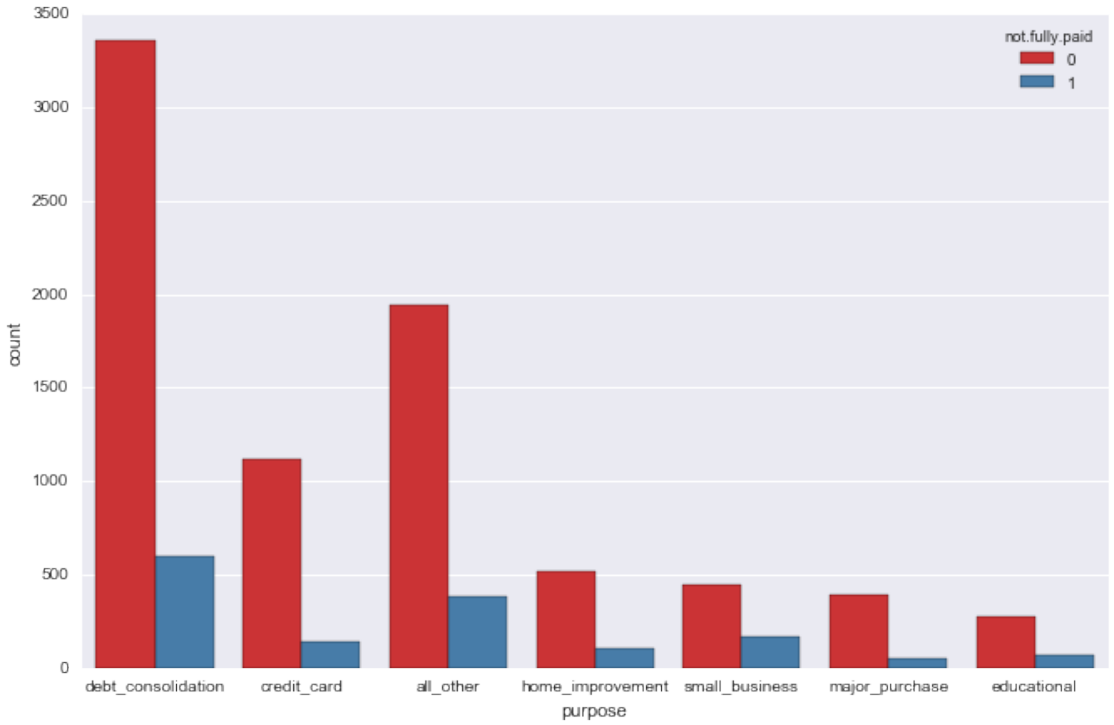
```
plt.xlabel('FICO')
```

[7]: <matplotlib.text.Text at 0x11c47a7f0>

Countplot using seaborn showing the counts of loans by purpose, with hue = not.fully.paid

[8]:
```
plt.figure(figsize=(11,7))
sns.countplot(x='purpose',hue='not.fully.paid',data=loans,palette='Set1')
```

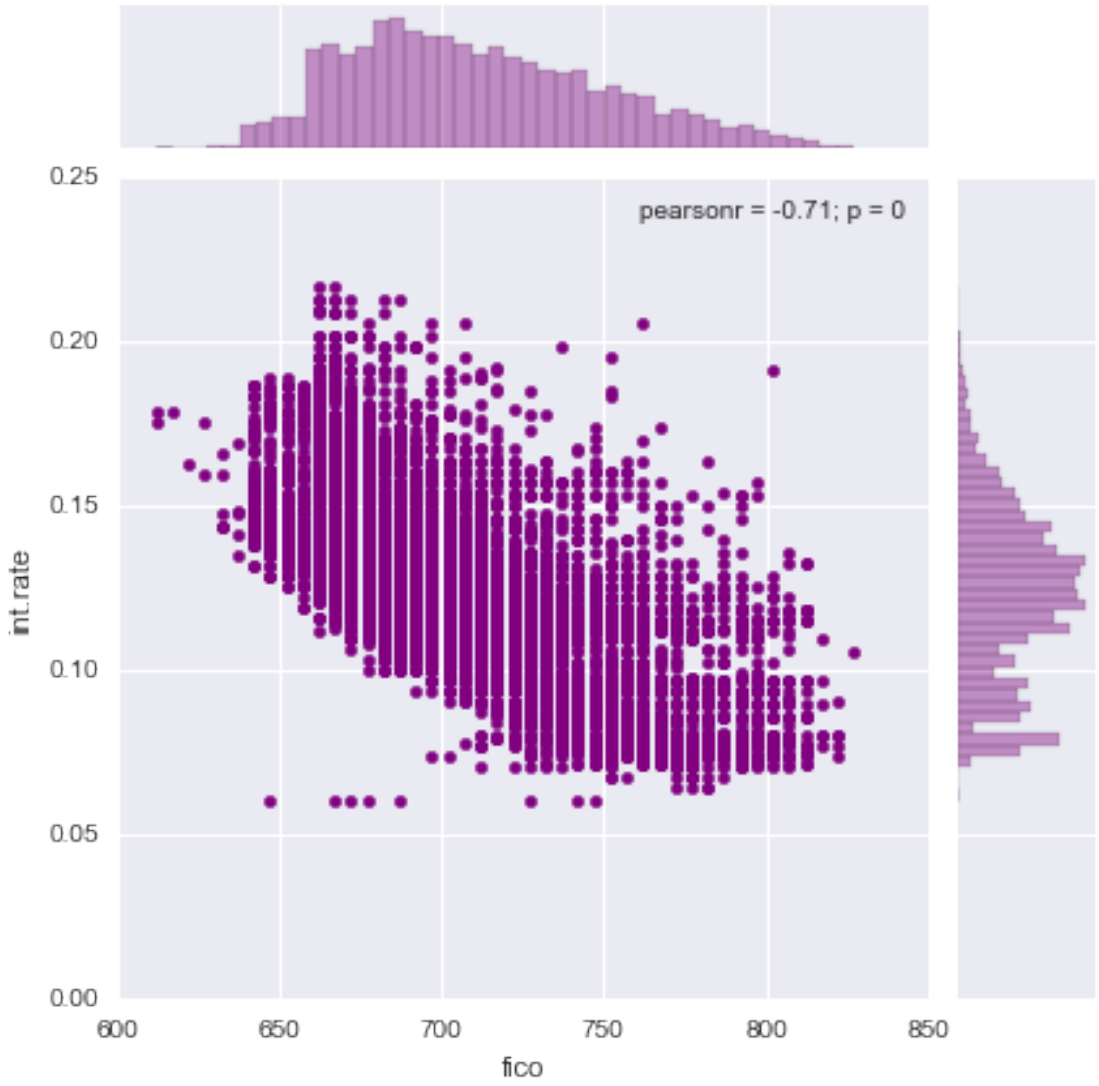
[8]: <matplotlib.axes._subplots.AxesSubplot at 0x119996828>



Visualize the trend between FICO score and interest rate

[9]:
```
sns.jointplot(x='fico',y='int.rate',data=loans,color='purple')
```

[9]:
```
<seaborn.axisgrid.JointGrid at 0x119963320>
```



lmplots to see if the trend differed between not.fully.paid and credit.policy

[10]:
```
plt.figure(figsize=(11,7))
sns.lmplot(y='int.rate',x='fico',data=loans,hue='credit.policy',
           col='not.fully.paid',palette='Set1')
```

[10]:
```
<seaborn.axisgrid.FacetGrid at 0x11d34b668>

<matplotlib.figure.Figure at 0x11d3094e0>
```

[12]:
```
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 The "purpose" column is categorical. Hence, transform them using dummy variables so sklearn will be able to understand them.

[36]:
```
cat_feats = ['purpose']
```

Using pd.get_dummies(loans,columns=cat_feats,drop_first=True), create a fixed larger dataframe that has new feature columns with dummy variables. Set this dataframe as final_data

[37]:
```
final_data = pd.get_dummies(loans,columns=cat_feats,drop_first=True)
```

[38]:
```
final_data.info()
```

```
<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 float64
purpose_debt_consolidation    9578 non-null float64
purpose_educational           9578 non-null float64
purpose_home_improvement      9578 non-null float64
purpose_major_purchase        9578 non-null float64
purpose_small_business        9578 non-null float64
dtypes: float64(12), int64(7)
memory usage: 1.4 MB
```

0.1.2 Train Test Split

[20]:
```
from sklearn.model_selection import train_test_split
```

[21]:
```
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=101)
```

Training the Decision Tree Model

[22]:
```
from sklearn.tree import DecisionTreeClassifier
```

Create an instance of DecisionTreeClassifier() called dtree and fit it to the training data

[23]:
```
dtree = DecisionTreeClassifier()
```

[24]:
```
dtree.fit(X_train,y_train)
```

[24]:
```
DecisionTreeClassifier(class_weight=None, criterion='gini', max_depth=None,
            max_features=None, max_leaf_nodes=None, min_samples_leaf=1,
            min_samples_split=2, min_weight_fraction_leaf=0.0,
            presort=False, random_state=None, splitter='best')
```

Predicting and Evaluating the Decision Tree Model

[25]:
```
predictions = dtree.predict(X_test)
```

[26]:
```
from sklearn.metrics import classification_report,confusion_matrix
```

[27]:
```
print(classification_report(y_test,predictions))
```

```
             precision    recall  f1-score   support

          0       0.85      0.82      0.84      2431
          1       0.19      0.23      0.20       443

avg / total       0.75      0.73      0.74      2874
```

[28]:
```
print(confusion_matrix(y_test,predictions))
```

```
[[1995  436]
 [ 343  100]]
```

Training the Random Forest model Create an instance of the RandomForestClassifier class and fit it to the training data

[29]:
```
from sklearn.ensemble import RandomForestClassifier
```

[30]:
```
rfc = RandomForestClassifier(n_estimators=600)
```

[31]:
```
rfc.fit(X_train,y_train)
```

[31]:
```
RandomForestClassifier(bootstrap=True, class_weight=None, criterion='gini',
            max_depth=None, max_features='auto', max_leaf_nodes=None,
            min_samples_leaf=1, min_samples_split=2,
            min_weight_fraction_leaf=0.0, n_estimators=600, n_jobs=1,
            oob_score=False, random_state=None, verbose=0,
            warm_start=False)
```

Predicting and Evaluating the Random Forest Model

[32]:
```
predictions = rfc.predict(X_test)
```

[33]:
```
from sklearn.metrics import classification_report,confusion_matrix
```

[34]:
```
print(classification_report(y_test,predictions))
```

```
             precision    recall  f1-score   support

          0       0.85      1.00      0.92      2431
          1       0.57      0.03      0.05       443

avg / total       0.81      0.85      0.78      2874
```

[35]:
```
print(confusion_matrix(y_test,predictions))
```

```
[[2422    9]
 [ 431   12]]
```