Mastering the second hand market for peer to peer loans

Giulianna Perrotti

December 2016

The Situation

- Lending Club is a provider of peer to peer loans
- FolioInvesting is a trading company that trades second hand notes for Lending Club
- Trading notes on Folio is like trading in the stock market

The Situation

- Lending Club is a provider of peer to peer loans
- FolioInvesting is a trading company that trades second hand notes for Lending Club
- Trading notes on Folio is like trading in the stock market …ish!

The main difference between Folio and the Stock Market

Folio is a highly non optimized and inefficient platform. That means that, unlike the stock market, it is not fair to say that 'the note is worth its face value'.

The Question

Is it possible to pin point what are the factors that impact the **return on a note** and, therefore, not only understand how much a note should be worth but also **automate the buy-sell mechanism**?

The Question

Is it possible to pin point what are the factors that impact the **return on a note** and, therefore, not only understand how much a note should be worth but also **automate the buy-sell mechanism**?

Let's try!

The Process – Basic strategy

1. Estimate what are the factors that impact the value of a note in its inception
2. Create a model that predicts how much a note is worth
3. Automate the buying process
4. Estimate accuracy of returns
5. Put feedback loop in place

The Process – 2^{nd} hand market

6. Feed the model with additional data, to estimate 2^{nd} hand market
7. Automate selling process for notes that are overpriced
8. Automate buying process for notes that are underpriced
9. Profit

The Dataset

The dataset is a publicly available Lending Club dataset, with notes since inception.

- 1.2 MM observations
- 100 + columns
- Loan Information
- 'Social' Information
- Tradelines Information
- Public Records Information

```
In [6]: df.shape
Out[6]: (1218332, 111)
```

The Coding: Data Cleaning

In [7]:

```
#MAPPINGS
df['loan_status'] = df.loan_status.map({'Current':0, 'Fully Paid':1, 'Charged Off':2, 'Late (31-120 days)':0,
                    'In Grace Period':0, 'Late (16-30 days)':0, 'Does not meet the credit policy. Status:Fully Paid':0,
                    'Does not meet the credit policy. Status:Charged Off':0, 'Default':2})
df['home_ownership'] = df.home_ownership.map({'MORTGAGE':0, 'RENT':1, 'OWN':2, 'OTHER':3, 'NONE':3,
                                              'ANY':3}).astype('float')
df['verification_status'] = df.verification_status.map({'Source Verified':0, 'Verified':1,
                                                        'Not Verified':2}).astype('float')
df['verification_status_joint'] = df.verification_status_joint.map({'Source Verified':0, 'Verified':1,
                                                        'Not Verified':2}).astype('float')
df['application_type'] = df.application_type.map({'INDIVIDUAL':0, 'JOINT':1, 'DIRECT_PAY':2})
df['term'] = df.term.map({' 36 months':36, ' 60 months':60})
df['grade'] = df.grade.map({'A':1, 'B':2, 'C':3, 'D':4, 'E':5, 'F':6, 'G':7})
df['sub_grade'] = df.sub_grade.map({'A1':11, 'A2':12, 'A3':13, 'A4':14, 'A5':15, 'B1':21, 'B2':22, 'B3':23,
                                    'B4':24, 'B5':25, 'C1':31, 'C2':32, 'C3':33, 'C4':34, 'C5':35, 'D1':41,
                                    'D2':42, 'D3':43, 'D4':44, 'D5':45, 'E1':51, 'E2':52, 'E3':53, 'E4':54,
                                    'E5':55, 'F1':61, 'F2':62, 'F3':63, 'F4':64, 'F5':65, 'G1':71, 'G2':72,
                                    'G3':73, 'G4':74, 'G5':75})
df['pymnt_plan'] = df.pymnt_plan.map({'n':0, 'y':1})
df['addr_state'] = df.addr_state.map({'CA':1, 'NY':2, 'TX':3, 'FL':4, 'IL':5, 'NJ':6, 'PA':7, 'OH':8, 'GA':9,
                                      'VA':10, 'NC':11, 'MI':12, 'MD':13, 'AZ':14, 'MA':15, 'WA':16, 'CO':17,
                                      'MN':18, 'MO':19, 'IN':20, 'CT':21, 'TN':22, 'NV':23, 'WI':24, 'AL':25,
                                      'SC':26, 'OR':27, 'LA':28, 'KY':29, 'OK':30, 'KS':31, 'AR':32, 'UT':33,
                                      'NM':34, 'HI':35, 'NH':36, 'MS':37, 'WV':38, 'RI':39, 'MT':40, 'DE':41,
                                      'DC':42, 'AK':43, 'WY':44, 'VT':45, 'SD':46, 'NE':47, 'ME':48, 'ND':49,
                                      'ID':50, 'IA':51})
df['initial_list_status'] = df.initial_list_status.map({'w':0, 'f':1})
df['emp_length'] = df.emp_length.map({'<1 year':0, '1 year':1, '2 years':2, '3 years':3, '4 years':4, '5 years':5,
                                      '6 years':6, '7 years':7, '8 years':8, '9 years':9, '10 years':10, 'n/a': 11})
df['purpose'] = df.purpose.map({'debt_consolidation':0, 'credit_card':1, 'home_improvement':2, 'major_purchase':3,
                                'small_business':4, 'car':5, 'medical':6,'other':7, 'moving':8,
                                'vacation':9, 'house':10, 'wedding':11, 'renewable_energy':12, 'educational':13})
```

The Coding: Data Cleaning

```
#REPLACES
df['int_rate'] = df['int_rate'].replace('%','',regex=True).astype('float')
df['zip_code'] = df['zip_code'].replace('xx','',regex=True).astype('float')
df['revol_util'] = df['revol_util'].replace('%','',regex=True).astype('float')

#IRRELEVANTS
del df['url']

#NEW FIELDS
df['total_received'] = df['total_rec_prncp'] + df['total_rec_int'] + df['total_rec_late_fee']
df['percent_received'] = df['total_received']/df['funded_amnt']

#MISSING VALUES
df = df.replace([np.inf, -np.inf], np.nan)
df = df[df.id.isnull() == False]
df = df[df.member_id.isnull() == False]
df = df[df.percent_received.isnull() == False]
```

```
In [8]: #TEMPORARY PATCHES
del df['id']
del df['emp_title']
del df['desc']
del df['title']
```

The Coding: Dates

In [40]:

```
#DATES
import datetime

@lru_cache(maxsize=1000)
def dater(x):
    try:
        return datetime.datetime.strptime(x, '%b-%y')
    except:
        return None

@lru_cache(maxsize=1000)
def yearer(x):
    try:
        return float(dater(x).year)
    except:
        return None

@lru_cache(maxsize=1000)
def monther(x):
    try:
        return float(dater(x).month)
    except:
        return None

@lru_cache(maxsize=1000)
def numberer(x):
    try:
        return float(yearer(x) * 12 + monther(x))
    except:
        return None
```

The Coding: Dates (cont)

```
df['year_issue_d'] = df.issue_d.apply(yearer)
df['month_issue_d'] = df.issue_d.apply(monther)
df['new_issue_d'] = df.issue_d.apply(numberer)
#del df['issue_d']

df['year_earliest_cr_line'] = df.earliest_cr_line.apply(yearer)
df['month_earliest_cr_line'] = df.earliest_cr_line.apply(monther)
df['new_earliest_cr_line'] = df.earliest_cr_line.apply(numberer)
#del df['earliest_cr_line']

df['year_last_pymnt_d'] = df.last_pymnt_d.apply(yearer)
df['month_last_pymnt_d'] = df.last_pymnt_d.apply(monther)
df['new_last_pymnt_d'] = df.last_pymnt_d.apply(numberer)
#del df['last_pymnt_d']

df['year_next_pymnt_d'] = df.next_pymnt_d.apply(yearer)
df['month_next_pymnt_d'] = df.next_pymnt_d.apply(monther)
df['new_next_pymnt_d'] = df.next_pymnt_d.apply(numberer)
#del df['next_pymnt_d']

df['year_last_credit_pull_d'] = df.last_credit_pull_d.apply(yearer)
df['month_last_credit_pull_d'] = df.last_credit_pull_d.apply(monther)
df['new_last_credit_pull_d'] = df.last_credit_pull_d.apply(numberer)
#del df['last_credit_pull_d']

#NEW FIELDS
df['year_time_paying'] = df['year_last_pymnt_d'] - df['year_issue_d']
df['months_time_paying'] = df['month_last_pymnt_d'] - df['month_issue_d']
df['total_time_paying'] = df['new_last_pymnt_d'] - df['new_issue_d']


#ADDRESSING MISSING VALUES AGAIN...
df = df[df.year_time_paying.isnull() == False]
df = df[df.months_time_paying.isnull() == False]
df = df[df.total_time_paying.isnull() == False]
```

The Coding: missing values

In [56]:
```
df = df[df.year_time_paying.isnull() == False]
df = df[df.months_time_paying.isnull() == False]
df = df[df.total_time_paying.isnull() == False]
```

In [57]:
```
#Addressing the missing values

df['annual_inc'].fillna(value = 0,  inplace=True)
df['open_acc'].fillna(value = 0,  inplace=True)
df['pub_rec'].fillna(value = 0,  inplace=True)
df['delinq_2yrs'].fillna(value = 0,  inplace=True)
df['delinq_amnt'].fillna(value = 0,  inplace=True)
df['acc_now_delinq'].fillna(value = 0,  inplace=True)
df['total_acc'].fillna(value = 0,  inplace=True)
df['tax_liens'].fillna(value = 0,  inplace=True)
df['chargeoff_within_12_mths'].fillna(value = 0,  inplace=True)
df['collections_12_mths_ex_med'].fillna(value = 0,  inplace=True)
df['revol_util'].fillna(value = 0,  inplace=True)
df['pub_rec_bankruptcies'].fillna(value = 0,  inplace=True)
df['total_bal_ex_mort'].fillna(value = 0,  inplace=True)
df['total_bc_limit'].fillna(value = 0,  inplace=True)
df['acc_open_past_24mths'].fillna(value = 0,  inplace=True)
df['num_sats'].fillna(value = 0,  inplace=True)
df['num_bc_sats'].fillna(value = 0,  inplace=True)
df['mths_since_recent_bc'].fillna(value = 0,  inplace=True)
df['bc_open_to_buy'].fillna(value = 0,  inplace=True)
df['percent_bc_gt_75'].fillna(value = 0,  inplace=True)
df['mort_acc'].fillna(value = 0,  inplace=True)
df['num_tl_op_past_12m'].fillna(value = 0,  inplace=True)
df['num_tl_90g_dpd_24m'].fillna(value = 0,  inplace=True)
df['num_tl_30dpd'].fillna(value = 0,  inplace=True)
df['num_rev_tl_bal_gt_0'].fillna(value = 0,  inplace=True)
df['num_op_rev_tl'].fillna(value = 0,  inplace=True)
df['num_il_tl'].fillna(value = 0,  inplace=True)
```

Goes on and on and on…

The Coding: Training Dataset

```
In [60]: #MAKING THE X, y

In [148]: dffortraining = df[df.loan_status != 0]

          dffortraining.shape

Out[148]: (407427, 126)

In [149]: ## GETTING THE NULL ACCURACY ##

In [150]: dffortraining.loan_status.value_counts() /407427*100 #total number got by shape

Out[150]: 1.0    81.366723
          2.0    18.633277
          Name: loan_status, dtype: float64
```

For Phase 1, disconsidering loans still ‘going’

```
In [7]: #MAPPINGS
        df['loan_status'] = df.loan_status.map({'Current':0, 'Fully Paid':1, 'Charged Off':2, 'Late (31-120 days)':0,
                            'In Grace Period':0, 'Late (16-30 days)':0, 'Does not meet the credit policy. Status:Fully Paid':0,
                           'Does not meet the credit policy. Status:Charged Off':0, 'Default':2})
```

The Coding: Tree Regressor

```
In [160]: #REGRESSION TREE

In [161]: from sklearn.tree import DecisionTreeRegressor
          treereg = DecisionTreeRegressor(random_state=1)
          treereg

Out[161]: DecisionTreeRegressor(criterion='mse', 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=1,
                     splitter='best')

In [162]: from sklearn.cross_validation import cross_val_score
          scores = cross_val_score(treereg, X, y, cv=5, scoring='mean_squared_error')
          np.mean(np.sqrt(-scores))

Out[162]: 0.065848543259739795

In [163]: treereg = DecisionTreeRegressor(max_depth=5, random_state=1)
          treereg.fit(X, y)

Out[163]: DecisionTreeRegressor(criterion='mse', max_depth=5, 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=1,
                     splitter='best')
```

The Coding: Chosen features

In [164]:
```
pd.DataFrame({'feature':feature_cols, 'importance':treereg.feature_importances_}).sort_values(by='importance',
                                        ascending = False).head(10)
```

Out[164]:

	feature	importance
92	total_received	0.461023
1	funded_amnt	0.273879
3	term	0.077394
29	last_pymnt_amnt	0.067814
95	new_issue_d	0.049933
99	year_last_pymnt_d	0.027096
2	funded_amnt_inv	0.022186
101	new_last_pymnt_d	0.009713
4	int_rate	0.007159
0	loan_amnt	0.002764

The Coding: Bagging

```
In [ ]: #BAGGING REGRESSOR
```

```
In [165]: from sklearn.ensemble import BaggingRegressor
          bagreg = BaggingRegressor(DecisionTreeRegressor(), n_estimators=15, bootstrap=True, oob_score=True, random_state=1)
```

```
In [168]: # fit and predict
          bagreg.fit(X, y)
          y = bagreg.predict(X)
          y
```

```
/Users/giuli/anaconda/lib/python2.7/site-packages/sklearn/ensemble/bagging.py:920: UserWarning: Some inputs do not ha
ve OOB scores. This probably means too few estimators were used to compute any reliable oob estimates.
  warn("Some inputs do not have OOB scores. "
```

```
Out[168]: array([ 1.17477965,  1.128568  ,  1.10145574, ...,  1.0171503 ,
                  1.02128035,  1.0008369 ])
```

```
In [173]: bagreg.oob_score_
```

```
Out[173]: 0.97904411889612197
```

The Coding

In [175]:
```
pd.DataFrame({'feature':feature_cols, 'importance':treereg.feature_importances_}).sort_values(by='importance',
                ascending = False).head(10)
```

Out[175]:

	feature	importance
92	total_received	0.461023
1	funded_amnt	0.273879
3	term	0.077394
29	last_pymnt_amnt	0.067814
95	new_issue_d	0.049933
99	year_last_pymnt_d	0.027096
2	funded_amnt_inv	0.022186
101	new_last_pymnt_d	0.009713
4	int_rate	0.007159
0	loan_amnt	0.002764

The Coding: Max Depth

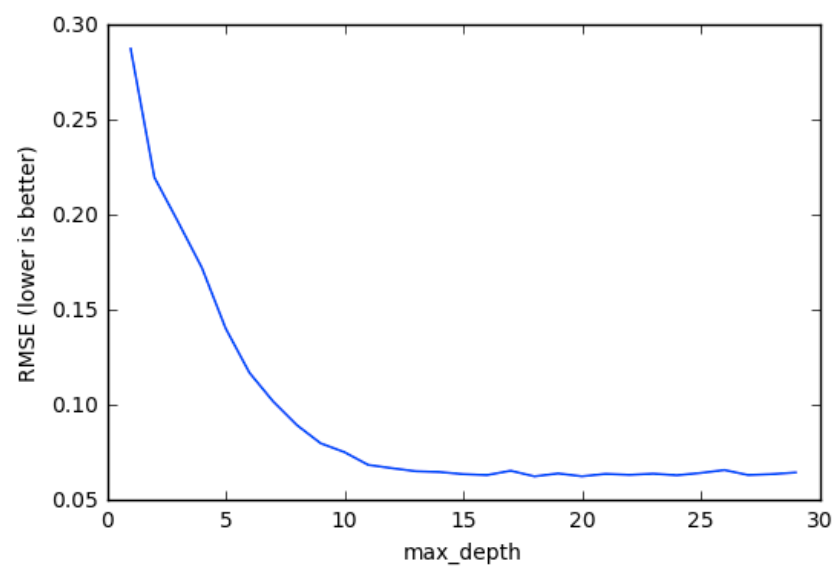
```
In [184]: # list of values to try for max_depth
          max_depth_range = range(1, 30)

          # list to store the average RMSE for each value of max_depth
          RMSE_scores = []

          # use 10-fold cross-validation with each value of max_depth
          from sklearn.cross_validation import cross_val_score
          for depth in max_depth_range:
              treereg = DecisionTreeRegressor(max_depth=depth, random_state=1)
              MSE_scores = cross_val_score(treereg, X, y, cv=5, scoring='mean_squared_error')
              RMSE_scores.append(np.mean(np.sqrt(-MSE_scores)))
```

```
In [185]: # plot max_depth (x-axis) versus RMSE (y-axis)
          plt.plot(max_depth_range, RMSE_scores)
          plt.xlabel('max_depth')
          plt.ylabel('RMSE (lower is better)')

Out[185]: <matplotlib.text.Text at 0x13e1adb10>
```




```
In [186]: # show the best RMSE and the corres
          sorted(zip(RMSE_scores, max_depth_r

Out[186]: (0.062596575263381887, 18)
```

Next Steps!

2. Finish developing the model for over-the-counter notes

- Enrich dataset with Esri Tapestry for zip code information
- Use NLP to assess whether description / profession / purpose play a role in the return rates

Next Steps!

3/7/8. Create buy/sell robot

- Infrastructure with Docker containers on ECS, for high availability
- Lending Club offers buy/sell REST API
- Folio offers buy/sell REST API

Next Steps!

6. Make the model for second hand market

- Enrich dataset with payment information *(can be scrapped from Lending Club with note id)*
- Assess which are the most relevant features, rebuild model

Next Steps!

9. Profit!!

QUESTIONS?