Kiva Microloans Project Report

Zhiwei Cindy Cai

May 7, 2014

Part I

Main research question: Will a loan be fully funded?

1 Introduction

1.1 KIVA Microloans

KIVA Microloans is a non-profit organization that runs an "online career support and empowerment program". It allows its users to connect through lending money to other people (mainly entrepreneurs and students) in over 70 countries. It is becoming a helpful Internet application that provides safe and accessible capital support to the borrowers and a quick and easy way for the loaners to invest their kindness. On its website, one can search for loans by categories, sectors, and other attributes of the borrowers and read the story of the loan, the repayment schedule and possibly profiles of other lenders who contributed. It is a very handy product for borrowers to fundraise overseas in all kinds of job fields and social roles. For this capstone project, I will be analyzing the Microloans data to give some insights on the lender's behavior and answering the questions: can we predict if a loan will be fully funded? What kinds of loans are likely to be funded?

1.2 Data from KIVA API

The data I am using for this project is an archived public JSON snapshot from the Kiva API. It contains more than 1.2 million lenders information and more than 650 thousands of loans. The raw data lives in thousands of JSON format files generated from hundreds or thousands of requests to the API. The API data has three objects, namely the lender object, the loans object and the loans_lenders object that connects the two main objects. Each object has attributes saved in a key-value pair dictionary data type. For this project I am only using the loans object data of the raw dataset. Since the original dataset has more than 650,000 entries, and not all of the data fields are relevant to answering my research questions, it needs a lot of pre-processing.

2 Data sources

Download the data from the KIVA API snapshot: http://s3.kiva.org/snapshots/kiva_ds_json.zip

Also available on my github: https://github.com/zhiweic/master-s-capstone-project After downloading the data, please unzip it. Also make sure to change your working directory before loading the data.

```
In [7]: os.chdir('/Users/admin/.../kiva_ds_json/loans')
```

Part II

Data cleansing and extraction

3 Load JSON files

The data fields retrieved from the raw JSON files include numerical fields: funded amount, journal entries, lender count, paid amount, loan amount, categorical fields: gender, sector, country, town, activity, id, status, first name, pictured and free user input field: description. The preprocessing mainly has 5 steps: pulling data values out of the JSON dictionary and unlisting the items, encoding the text fields from Unicode, exporting relevant variables into a easy-to-read .csv file, loading data into a panda data frame, and filtering and filling in missing information.

```
In [ ]: import sys
        import os
        import json
        from collections import defaultdict
        import csv
        from string import printable
```

```
In [ ]: # Initialize the dictionaries
        loans_info=defaultdict(list)
```

Variables of interest:

```
In [ ]: loans_colname=['funded_amount','sector','journal_totals','id','paid_amount
```

Function to encode text fields from Unicode to utf-8

```
In [ ]: def get_string(rawstr):
        if isinstance(rawstr,unicode):
        return rawstr.encode('utf-8').replace('\n', ' ').replace('\r', '').replace
        elif rawstr==None:
        return 'NA'
        else:
        return rawstr
```

```
In [ ]: for jfile in os.listdir(os.getcwd()):
        if 'json' in jfile:
        open_file=open(jfile,'r')
        pyresponse=json.load(open_file)
        results=pyresponse["loans"] #list
        for i in range(len(results)):
        for key in loans_colname:
        if key=='location':
        town=get_string(results[i][key]['town'])
        country=get_string(results[i][key]['country'])
        loans_info['town'].append(town)
        loans_info['country'].append(country)
```

```
elif key=='borrowers':
first_name=get_string(results[i][key][0]['first_name'])
gender=get_string(results[i][key][0]['gender'])
loans_info['first_name'].append(first_name)
loans_info['gender'].append(gender)
loans_info['pictured'].append(results[i][key][0]['pictured'])
elif key=='journal_totals':
loans_info['journal_entries'].append(results[i][key]['entries'])
elif key=='description':
if 'en' in results[i][key]['languages']:
description=get_string(results[i][key]['texts']['en'])
line="".join([ ch for ch in description if ch in printable ])
loans_info[key].append(line)
else:
loans_info[key].append('Non-English Text')
elif isinstance(results[i][key],unicode):
loans_info[key].append(results[i][key].encode('utf-8'))
else:
loans_info[key].append(results[i][key])
```

3.1 Nose test 1: making sure that the data extraction is correct for different data types

In [1]:

```
%%file get_string_nose.py
def get_string(rawstr):
if isinstance(rawstr,unicode):
return rawstr.encode('utf-8').replace('\n', ' ').replace('\r', '').replace
elif rawstr==None:
return 'NA'
else:
return rawstr

def test_unicode():
print 'test string is type(unicode)'
str_a = u'Hi!\n I am a piece of <i>unicode</i>.'
result = get_string(str_a)
assert result == 'Hi!  I am a piece of unicode.'

def test_number():
print 'test string is type(int)'
str_b = 12345
result = get_string(str_b)
assert result == 12345

def test_boolean():
print 'test string is type(bool)'
str_c = type(12345)==int #True
result = get_string(str_c)
assert result == True

def test_na():
print 'test string is type(Nonetype)'
str_d = None
result = get_string(str_d)
assert result == 'NA'
```

```
Overwriting get_string_nose.py
```

In [2]:

```
!nosetests get_string_nose.py
```

```
...
----------------------------------------------------------------------
Ran 4 tests in 0.001s

OK
```

4 Output the processed data into a csv file

A preprocessed clean version of loans.csv is also available on my github: https://github.com/zhiweic/master-s-capstone-project

In [ ]:
```
# Change the directory to where you want to save the csv
os.chdir('/Users/admin/desktop/...')
```

In [ ]:
```
with open('loans.csv','wb') as f:
w=csv.writer(f)
w.writerow(['funded_amount', 'sector', 'first_name', 'gender', 'lender_cou
for i in range(len(loans_info['sector'])):
templist=[]
for k in loans_info.keys():
templist.append(loans_info[k][i])
w.writerow(templist)
```

In [3]:
```
import numpy as np
```

In [4]:
```
import pandas as pd
```

Load the extraced data from the csv file into a panda data frame

In [5]:
```
loans=pd.read_csv('loans.csv')
```

In [6]:
```
loans.keys()
```

Out [6]:
```
Index([u'funded_amount', u'sector', u'first_name', u'gender',
u'lender_count', u'paid_amount', u'country', u'activity', u'town',
u'loan_amount', u'status', u'journal_entries', u'id', u'pictured',
u'description'], dtype='object')
```

Filter out the system error cases where status is null

In [7]:
```
loans=loans.ix[loans['status'].isnull()==False]
```

Need to fill in the NAs for first_name, description and town

In [8]:
```
loans['first_name']=loans['first_name'].fillna('NA')
loans['description']=loans['description'].fillna('NA')
loans['town']=loans['town'].fillna('NA')
```

In [9]:
```
from pylab import *
```

```
In [11]: %pylab inline
```

```
Populating the interactive namespace from numpy and matplotlib
```

Part III

Exploratory Data Analysis

After cleaning everything up into a usable data frame, I did some exploratory data analysis by looking at some summary statistics and identifying trends in how some of the variables are correlated.An intuition to check is if loans with higher funded amount have larger lenders count. I plotted the scatter plot with funded amount on the y-axis and lender count on the x-axis, and then fitted a regression line to the data points.

```
In [12]: from statsmodels.formula.api import ols
```

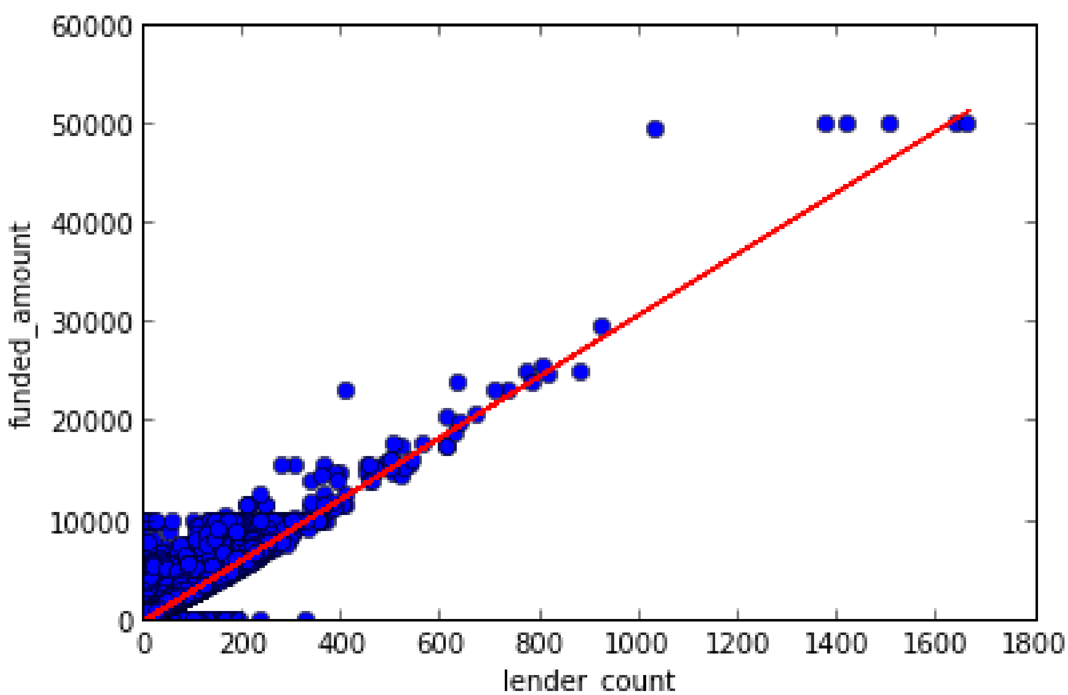
```
In [13]: lm0 = ols('funded_amount ~ lender_count', loans).fit()
         lm0.summary()
```

```
Out [13]: <class 'statsmodels.iolib.summary.Summary'>
"""
                            OLS Regression Results
======================================================================
========
Dep. Variable:          funded_amount   R-squared:
0.858
Model:                            OLS   Adj. R-squared:
0.858
Method:                 Least Squares   F-statistic:
3.991e+06
Date:                Tue, 06 May 2014   Prob (F-statistic):
0.00
Time:                        16:48:30   Log-Likelihood:
-4.7500e+06
No. Observations:              659956   AIC:
9.500e+06
Df Residuals:                  659954   BIC:
9.500e+06
Df Model:                           1
======================================================================
==========
                   coef    std err          t      P>|t|      [95.0%
Conf. Int.]
----------------------------------------------------------------------
----------
Intercept       40.1699      0.545     73.650      0.000      39.101
41.239
lender_count    30.8941      0.015   1997.869      0.000      30.864
30.924
======================================================================
========
```

```
Omnibus:                     852163.735   Durbin-Watson:
1.777
Prob(Omnibus):                    0.000   Jarque-Bera (JB):
351836802.564
Skew:                             6.929   Prob(JB):
0.00
Kurtosis:                       115.262   Cond. No.
48.4
========================================================================
========
"""
```

In [14]:
```
interc,slope=lm0.params
plot(loans['lender_count'],loans['funded_amount'],'ob')
xlabel('lender_count')
ylabel('funded_amount')
fitted=interc+slope*loans['lender_count']
plot(loans['lender_count'],fitted,'r-')
```

Out [14]:
```
[<matplotlib.lines.Line2D at 0x6c6d150>]
```



The slope of the regression line is significantly positive, and it tells us that each additional lender of a loan contributes to around 30 dollars. However, from the scatterplot we observe that this is only true when the funded amount is big. It's pretty clear that once the funded amount hit $10,000, the more money funded the more lenders needed to achieve that. With loans lower than $10,000, we cannot identify a pattern.Also it is interesting to see which sectors have the most number of loans, and later on I would want to further discuss if the sector of a loan affects its probability of being fully funded.

In [15]:
```
sector=pd.Categorical.from_array(loans['sector'])
```

In [16]:
```
print interc,slope
```

```
40.1698595904 30.8941215651
```

In [17]:
```
def jitter(series, factor):
    z = float(series.max()) - float(series.min())
    a = float(factor) * z / 50.
    return map(lambda x: x + np.random.uniform(-a, a), series)
```

In [12]:
```
import matplotlib.pyplot as plt
```

In [18]:
```
loans.groupby('sector')['country'].describe()
```

Out [18]:

sector		
Agriculture	count	142347
	unique	73
	top	Philippines
	freq	25691
Arts	count	13630
	unique	69
	top	Peru
	freq	2650
Clothing	count	44552
	unique	70
	top	Kenya
	freq	4540
Construction	count	11980
	unique	67
	top	Nicaragua
	freq	1640
Education	count	10055
	unique	57
	top	Jordan
	freq	1016
Entertainment	count	1161
	unique	55
	top	Philippines
	freq	179
Food	count	168675
	unique	72
	top	Philippines
	freq	26168
Health	count	5513
	unique	62
	top	Kenya
	freq	716
Housing	count	21806
	unique	52
	top	Nicaragua
	freq	5505
Manufacturing	count	9009
	unique	67
	top	Philippines

```
                 freq              1250
Personal Use     count             7944
                 unique              52
                 top           Cambodia
                 freq              2776
Retail           count           148469
                 unique              78
                 top        Philippines
                 freq             34919
Services         count            51515
                 unique              72
                 top              Kenya
                 freq              5621
Transportation   count            21869
                 unique              60
                 top        Philippines
                 freq              5122
Wholesale        count             1431
                 unique              60
                 top               Peru
                 freq               151
Length: 60, dtype: object
```

From the crosstab of sector and loan count, we see that the top three sectors are Food, Retail and Agriculture. Philippines, the country that owns the most number of loans, also owns the most loans in these three sectors. Entertainment and wholesale have a lot fewer loans comparing the the other sectors.

In [20]:
```
lm1 = ols('loan_amount ~ sector', loans).fit()
lm1.summary()
```

Out [20]:
```
<class 'statsmodels.iolib.summary.Summary'>
"""
                            OLS Regression Results
==============================================================================
========
Dep. Variable:            loan_amount   R-squared:
0.006
Model:                            OLS   Adj. R-squared:
0.006
Method:                 Least Squares   F-statistic:
295.6
Date:                Tue, 06 May 2014   Prob (F-statistic):
0.00
Time:                        16:53:49   Log-Likelihood:
-5.4168e+06
No. Observations:              659956   AIC:
1.083e+07
Df Residuals:                  659941   BIC:
1.083e+07
Df Model:                          14
==============================================================================
======================
                              coef    std err          t      P>|t|
[95.0% Conf. Int.]
------------------------------------------------------------------------------
```

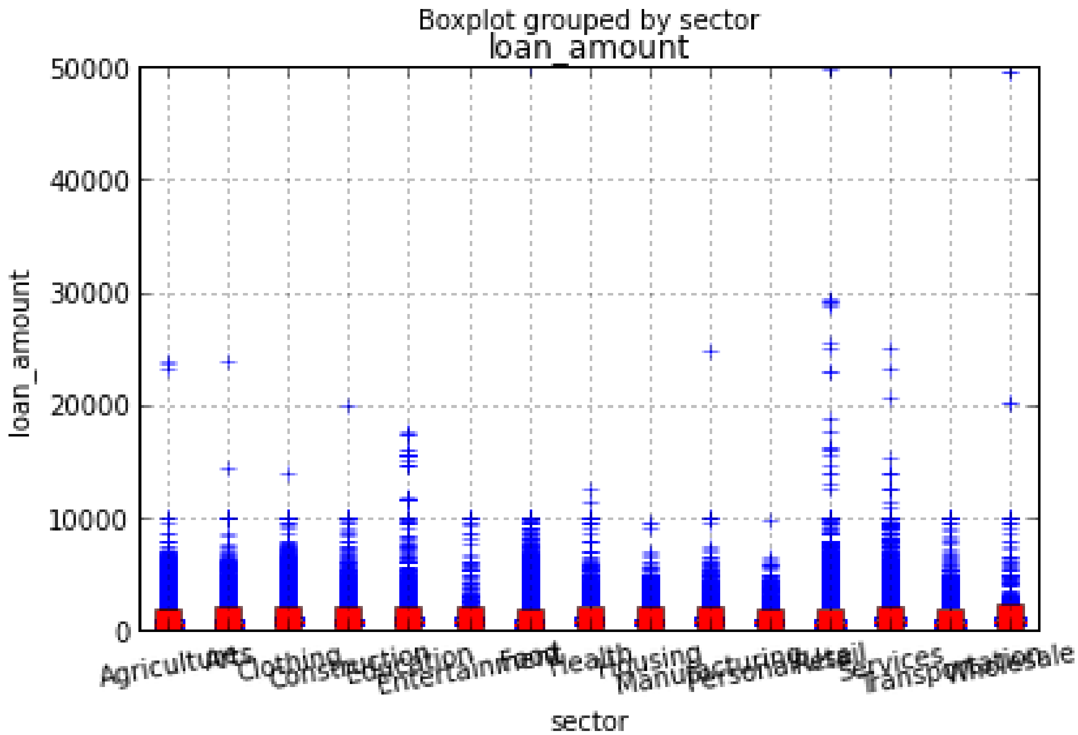
```
----------------------
Intercept                    804.6445      2.353    341.924      0.000
800.032   809.257
sector[T.Arts]                70.0107      7.961      8.794      0.000
54.408    85.614
sector[T.Clothing]           132.6625      4.820     27.523      0.000
123.216   142.109
sector[T.Construction]        82.7804      8.446      9.801      0.000
66.226    99.335
sector[T.Education]          168.3640      9.162     18.377      0.000
150.407   186.321
sector[T.Entertainment]      217.0825     26.164      8.297      0.000
165.803   268.362
sector[T.Food]               -41.5978      3.196    -13.017      0.000
-47.861   -35.335
sector[T.Health]             176.7223     12.187     14.501      0.000
152.836   200.609
sector[T.Housing]             53.1997      6.457      8.239      0.000
40.545    65.855
sector[T.Manufacturing]       58.8587      9.646      6.102      0.000
39.953    77.764
sector[T.Personal Use]       -17.9438     10.236     -1.753      0.080
-38.006     2.118
sector[T.Retail]               8.3441      3.294      2.533      0.011
1.889    14.799
sector[T.Services]           173.2843      4.565     37.958      0.000
164.337   182.232
sector[T.Transportation]     -19.6314      6.449     -3.044      0.002
-32.271    -6.992
sector[T.Wholesale]          404.1780     23.589     17.135      0.000
357.945   450.411
======================================================================
========
Omnibus:                     798340.652   Durbin-Watson:
1.707
Prob(Omnibus):                    0.000   Jarque-Bera (JB):
607371260.671
Skew:                             5.854   Prob(JB):
0.00
Kurtosis:                       151.157   Cond. No.
25.6
======================================================================
========
"""
```

In [21]:

```python
boxdata=loans[['sector','loan_amount']]
boxdata.boxplot(by='sector')
xticks(np.unique(sector.labels)+1, sector.levels,rotation=10,fontsize=10)
xlabel('sector')
ylabel('loan_amount')

plot(range(16)[1:], lm1.params[0] + np.append(0, lm1.params[1:]), 'sr', ma
```

Out [21]: [<matplotlib.lines.Line2D at 0x2426e290>]



To see if loan amount differs by sectors, I looked at a side-by-side boxplot of all the sectors. Since we have relatively few loans more than 10,000, those over are all treated as outliers, and we cannot really see what is going on with the smaller loans. Therefore, I decided to focus on the mini-loans of amount less than or equal to $2000. I also observed that the loans in the top three sectors are three out of five of sectors with the least average loan amount. Wholesale, entertainment and health are the top three sectors with the largest average loan amount.

Part IV

Data analysis and modeling

5 Take a closer look at a subset of the data: Focusing on smaller loans

The next step is to clean up the data that I wanted to focus on doing more advance analysis. I filtered out all the loans higher than $2000, and those with status “refunded” and “issue”. These two statuses mean that there were some exception and unexpected situation with the loans, so I would want to clear these noises from my dataset.

In [13]:
```
small=loans[loans['loan_amount']<2000]
small=small[small['loan_amount']>0]
small=small[small['status']!='refunded']
small=small[small['status']!='issue']
```

In [14]:
```
small=small[['sector','first_name','gender','country','town','activity','l
```

My first interesting discovery is that 20% of the countries (about 15 of them) own around 432,000 loans (more than 70% of total loans), which agrees with the 80-20 rule saying that roughly 80% of the effects come from 20% of the causes. This is a classical rule reflecting the distribution of capital and wealth. Even in this mini-scale Internet trading environment, we can see that the cash flows follow the rule. The top five countries with the most number of loans are Philippines, which has twice as many loans as the second place, Peru, followed by Kenya, Cambodia and Nicaragua.

In [15]:
```
sum(small['country'].value_counts()[0:15])
```

Out [15]:
```
432490
```

In [25]:
```
country=pd.Categorical.from_array(small['country'])
color=country.labels
small['country'].value_counts()
```

Out [25]:
```
Philippines     101323
Peru             58804
Kenya            56143
Cambodia         38581
Nicaragua        29208
Uganda           20310
El Salvador      20149
Tajikistan       18176
Ecuador          17051
Ghana            16539
Pakistan         13660
Bolivia          12027
Mexico           11300
Togo             10003
Sierra Leone      9216
...
South Africa                              164
Burundi                                   149
The Democratic Republic of the Congo      134
Haiti                                     133
Zambia                                    116
United States                              95
Belize                                     88
Turkey                                     77
Thailand                                   35
Israel                                     20
Bangladesh                                 14
Brazil                                     14
Gaza                                        7
Chad                                        5
Suriname                                    1
Length: 76, dtype: int64
```

In [83]:
```
li=small.groupby('country')['loan_amount'].sum()
li.sort(ascending=False)
li[0:15]
```

Out [83]:
```
country
Philippines    34450100
Peru           33583850
Cambodia       26383775
Kenya          25759300
Nicaragua      17443575
Ecuador        14797925
Tajikistan     14287000
Uganda         13210925
El Salvador    11955850
Bolivia        10173200
Pakistan        9197850
Ghana           8373550
Lebanon         8086675
Togo            7822075
Rwanda          7429075
Name: loan_amount, dtype: int64
```

With the clean data, I again looked at the relationship between sector and loan amount. From table, we can see that now wholesale, health and education are the top three sectors with the most expensive average loans. Comparing to my previous results, entertainment is kicked out of the top 5, which indicates that there are some loans in the entertainment sector with a huge loan amount that are now filtered out as "outliers". The bottom three sectors are now food, retail and arts.

In [28]:
```
sector=pd.Categorical.from_array(small['sector'])
small.groupby('sector')['loan_amount'].mean()
```

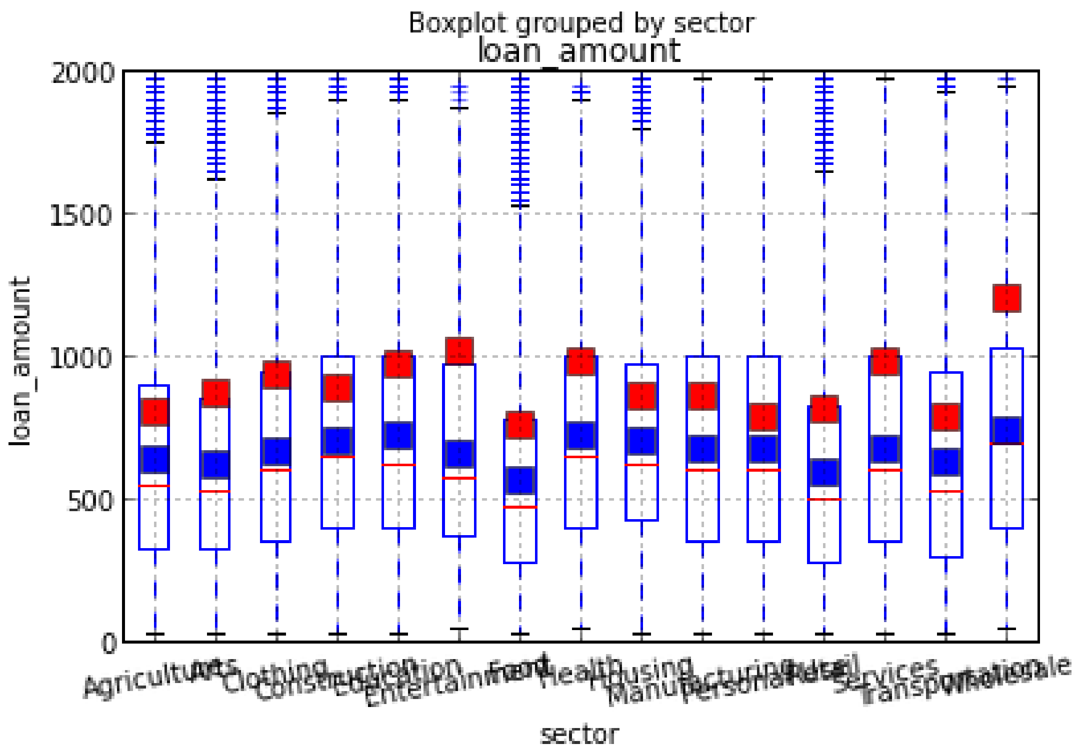
Out [28]:
```
sector
Agriculture       643.049944
Arts              622.157476
Clothing          672.014087
Construction      706.904394
Education         721.252618
Entertainment     661.951220
Food              566.907460
Health            726.370902
Housing           706.699558
Manufacturing     673.872938
Personal Use      676.903417
Retail            594.016683
Services          674.200931
Transportation    630.248449
Wholesale         744.330105
Name: loan_amount, dtype: float64
```

In [27]:
```
boxdata=small[['sector','loan_amount']]
boxdata.boxplot(by='sector')
xticks(np.unique(sector.labels)+1, sector.levels,rotation=10,fontsize=10)
xlabel('sector')
ylabel('loan_amount')

lm2 = ols('loan_amount ~ sector', small).fit()

plot(range(16)[1:], lm1.params[0] + np.append(0, lm1.params[1:]), 'sr', ma
plot(range(16)[1:], lm2.params[0] + np.append(0, lm2.params[1:]), 'sb', ma
```

Out [27]: [<matplotlib.lines.Line2D at 0x33611a10>]



Side to side boxplots of sectors comparing the loan amount. The blue squares are the loan amount means of each sector, while the red squares are the loan amount means of each sector before filtering the “outliers”. We can see that Entertainment and Wholesale have the two means very far apart. Actually for these two sectors, the red square is outside of the 25-75 interquantile. These are the sectors that have most number of expensive loans.

6 Text Classification: loan status vs. description

Whenever a lender browses through the website, the first thing that he/she will look at is most probably going to be the description of the loan. Then naturally we would want to know if the description of the borrowers tells us anything about the likeliness of a loan being funded. To do this, I focus on only the description and loan status fields and trying to classify loans based on the text information. Loans with status “fundraising” are not included for training my model, since it is unclear if they will be funded or not. In other words, the usage of my classifier is to help predict whether a fundraising loan will be fully funded.Loans with status “fundraising/reviewed” are loans that we don’t know if they will get funded or not. REAL test set: fundraising

In [16]:
```
test_cond=(small['status']=='fundraising') | (small['status']=='reviewed')
fundraising=small[test_cond]
fundraising['status'].value_counts()
```

Out [16]:
```
fundraising    2231
reviewed        605
dtype: int64
```

Training set includes all the other loans that we already know if they werer funded or not.

```
In [17]: small=small[~test_cond]
         small['status'].value_counts()
```

```
Out [17]: paid                469260
          in_repayment         99974
          defaulted            11245
          inactive_expired      8462
          expired               7471
          deleted               2275
          inactive               494
          funded                 483
          dtype: int64
```

I recoded the loan status as either fully funded (1) or not fully funded (0) as following:

```
In [18]: reencode={'paid':1, 'in_repayment':1, 'funded':1, 'defaulted':1, 'inactive
         small['status']=small['status'].map(reencode)
```

6.1 Nose Test 2: making sure that the recoding is working

```
In [3]: %%file encode_nose.py
        def encode(status):
        test_cond = (status =='fundraising' or status =='reviewed')
        if test_cond==True:
        return 'NA'
        else:
        reencode={'paid':1, 'in_repayment':1, 'funded':1, 'defaulted':1, 'inactive
        return reencode[status]

        def test_1():
            status = ['paid','paid','defaulted','inactive_expired']
            result = map(encode,status)
            assert result == [1, 1, 1, 0]

        def test_2():
          status2 = ['defaulted', 'inactive_expired', 'fundraising', 'deleted', 'r
          result = map(encode,status2)
          assert result == [1, 0, 'NA', 0, 'NA']
```

```
Overwriting encode_nose.py
```

```
In [4]: !nosetests encode_nose.py
```

```
..
----------------------------------------------------------------------
Ran 2 tests in 0.000s

OK
```

6.2 Naive Bayes Classifier Training and Evaluation

After recoding the statuses, I have more than 580 thousands of fully funded loans, and 18.7 thousands of not fully funded ones, which is only about 3.2% of the funded ones. This imbalance of 0/1 cases might result in some over-fitting of under-fitting problems that will be discussed later.

In [19]:
```
small['status'].value_counts()
```

Out [19]:
```
1    580962
0     18702
dtype: int64
```

In [20]:
```
from scipy.stats import sem
```

In [21]:
```
from sklearn.cross_validation import cross_val_score, KFold
```

In [22]:
```
small.keys()
```

Out [22]:
```
Index([u'sector', u'first_name', u'gender', u'country', u'town',
u'activity', u'loan_amount', u'journal_entries', u'pictured',
u'description', u'status'], dtype='object')
```

In order to recognize the patterns in descriptions, I decided to use statistical classifiers to capture the features in the text. A Naïve Bayes Classifier is the most desired method. Some advantages for the Naïve Bayes Classifiers are: 1) it is a very simple probabilistic classifier based on the Bayes rule; 2) it can be trained very efficiently in a supervised learning setting. Usually the strong independence assumptions it requires are not true. However, the multinomial Naïve Bayes Classifier is suitable for classification with discrete features (e.g., word counts for text classification). In other words, for the description I have, the presence of a particular word is usually independent of any other words. Some exceptions are when the author uses some common phrases, which is taken into consideration and will be solved later in the process.

In [23]:
```
SPLIT = 0.8
split = int(len(small['status'])*SPLIT)
Y = small['status']
X = small['description']
```

Sample randomly for the training set to split it into the real training set and evaluation set.

In [24]:
```
from sklearn.cross_validation import train_test_split
X_train, X_test, Y_train, Y_test = train_test_split(X,Y,test_size=0.2, ran
```

Create a k-fold croos validation iterator of k=5 folds

In [25]:
```
def evaluate_cross_validation(clf, X, y, K):
    cv = KFold(len(y), K, shuffle=True, random_state=0)
    # by default the score used is the one returned by score method of the
    scores = cross_val_score(clf, X, y, cv=cv)
    print scores
    print ("Mean score: {0:.3f} (+/-{1:.4f})").format(
        np.mean(scores), sem(scores))
```

In [26]:
```
from sklearn.naive_bayes import MultinomialNB
from sklearn.pipeline import Pipeline
from sklearn.feature_extraction.text import CountVectorizer,TfidfVectorize
```

CV on two different weighting methods: Count Vectorizer and Tfidf Vectorizer

The descriptions can be viewed as a sample from the “bag-of-words model”, which is represented by a bag of words, disregarding grammar or word order to keep the simplicity. This model is commonly used for natural language processing and document classification. Since word order and meaning cannot be quantified easily, the frequency of each word is used as a key feature for training a classifier. In my first step of determining which classifier to use, I evaluated performance of the simple word count weighting and the tf-idf (term frequency–inverse document frequency) weighting method with 5-fold cross validation. Here I used the sklearn and scipy stats module. The main difference of the tf-idf value compared to the usual count weighting is that it controls for the fact that some words generally occur more often than others by increasing the measure proportionally to the number of times a word appears in the paragraph but also downscaling by its frequency in other paragraphs.

In [27]:
```
clf = Pipeline([
    ('vect', CountVectorizer()),
    ('clf', MultinomialNB()),
])
clf2 = Pipeline([
    ('vect', TfidfVectorizer()),
    ('clf', MultinomialNB()),
])
```

In [22]:
```
clfs = [clf, clf2]
for clf in clfs:
   evaluate_cross_validation(clf, X, Y, 5)
```

```
[ 0.94623665  0.9458531   0.94688701  0.94571969  0.94737018]
Mean score: 0.946 (+/-0.0003)
[ 0.96902437  0.96834899  0.96905772  0.96818224  0.96964947]
Mean score: 0.969 (+/-0.0003)
```

Add stop words

From the cross validation results, we were able to tell that the tf-idf weighting did a superior job. Tf-idf can be used for stop-words filtering with its special weighting scheme. To ensure that the common words and phrases were thrown out, I decided to impose a list of stop-word to the classifier so that these words and phrases will be filtered out prior to the text processing of descriptions.

In [28]:
```
def get_stop_words():
    result = set()
    for line in open('english.stop.txt', 'r').readlines():
        result.add(line.strip())
    return result
```

In [30]:
```
stop_words = get_stop_words()
```

In [31]:
```
clf3 = Pipeline([
    ('vect', TfidfVectorizer(
                stop_words=stop_words,
    )),
    ('clf', MultinomialNB()),
])
```

In [26]:
```
evaluate_cross_validation(clf3, X, Y, 5)
```

```
[ 0.96899102  0.96831564  0.96899102  0.96815722  0.96971617]
Mean score: 0.969 (+/-0.0003)
```

Again, with 5-fold cross validation, results showed that stop-words effectively improved the performance of the classifier.

Adjust the alpha parameter for Lidstone smoothing

Another possible change to the classifier has to do with the fact that a multinomial distribution deals with categories. Applying an additive smoothing technique to the classifier permits the assignment of non-zero probabilities to words not in the sample. Additive smoothing, a.k.a Lidstone smoothing is commonly a component of naïve Bayes classifiers. Although this did not improve the cross validation result, it could be help for training the data, since we have limited data on unfunded loans. With additive smoothing, we will be able to have a more complete word frequency list. For all of these text classifiers, I used a pipeline to combine the process of weighting the words and cross validating.

In [32]:
```
clf4 = Pipeline([
    ('vect', TfidfVectorizer(
                stop_words=stop_words,
    )),
    ('clf', MultinomialNB(alpha=0.01)),
])
```

In [37]:
```
evaluate_cross_validation(clf4, X, Y, 5)
```

```
[ 0.96688151  0.96598934  0.96708996  0.96634788  0.96791515]
Mean score: 0.967 (+/-0.0003)
```

Training and evaluation

Now that I have two candidates for my classifier, I could train and evaluate them with my built-up pipeline. The sklearn module gives a very nicely formatted metric report on classification, which came in to be handy for classifier evaluation.

In [33]:
```
from sklearn import metrics

def train_and_evaluate(clf, X_train, X_test, y_train, y_test):

    clf.fit(X_train, y_train)

    print "Accuracy on training set:"
    print clf.score(X_train, y_train)
    print "Accuracy on testing set:"
    print clf.score(X_test, y_test)

    y_pred = clf.predict(X_test)

    print "Classification Report:"
    print metrics.classification_report(y_test, y_pred)
```

In the classification metrics report, precision (1) = true positive/(true positive + false positive), recall (1) = true positive/(true positive+ false negative), f1-score (1), harmonic mean of precision and recall, = 2tp/(2tp+fp+fn).

In [30]:
```
train_and_evaluate(clf3, X_train, X_test, Y_train, Y_test)
```

```
Accuracy on training set:
0.968769998186
Accuracy on testing set:
0.968991019986
Classification Report:
             precision    recall  f1-score   support

          0       0.59      0.01      0.02      3732
          1       0.97      1.00      0.98    116201

avg / total       0.96      0.97      0.95    119933
```

In [41]:
```
train_and_evaluate(clf4, X_train, X_test, Y_train, Y_test)
```

```
Accuracy on training set:
0.971871736452
Accuracy on testing set:
0.966881508842
Classification Report:
             precision    recall  f1-score   support

          0       0.38      0.10      0.16      3732
          1       0.97      0.99      0.98    116201

avg / total       0.95      0.97      0.96    119933
```

In both of the reports, the precision rate, recall rate and f1-score for funded loans are very high, around 0.97-0.99. This tells us that we are doing very well in predicting the status of the funded ones. However, when we look at the precision, recall and f1-score for the not fully funded loans, the metrics are a little disappointing. For precision (0), we see that only 40-50% of the unfunded loans are correctly classified. While for recall (0), even with Lidstone smoothing, 90% of the loans predicted to be unfunded are actually fully funded. Since I want to give positive feedbacks for as much cases as possible, i.e. I don't want the funded loans to be misclassified, I chose the classifier with Lidstone smoothing and sacrificing the precision for a higher recall for unfunded loans. Although the testing and training sets have very high classification score, it does not mean that our classifier is doing excellent. As discussed earlier, we have much more funded loans than unfunded ones, therefore the classifier has more information to predict the funded ones and thus does better in precision (1) and recall (1).

ROC Curve

To be clearer on how the classifier is doing in classifying the loans, I plotted the ROC (Receiver Operating Characteristic) curve.

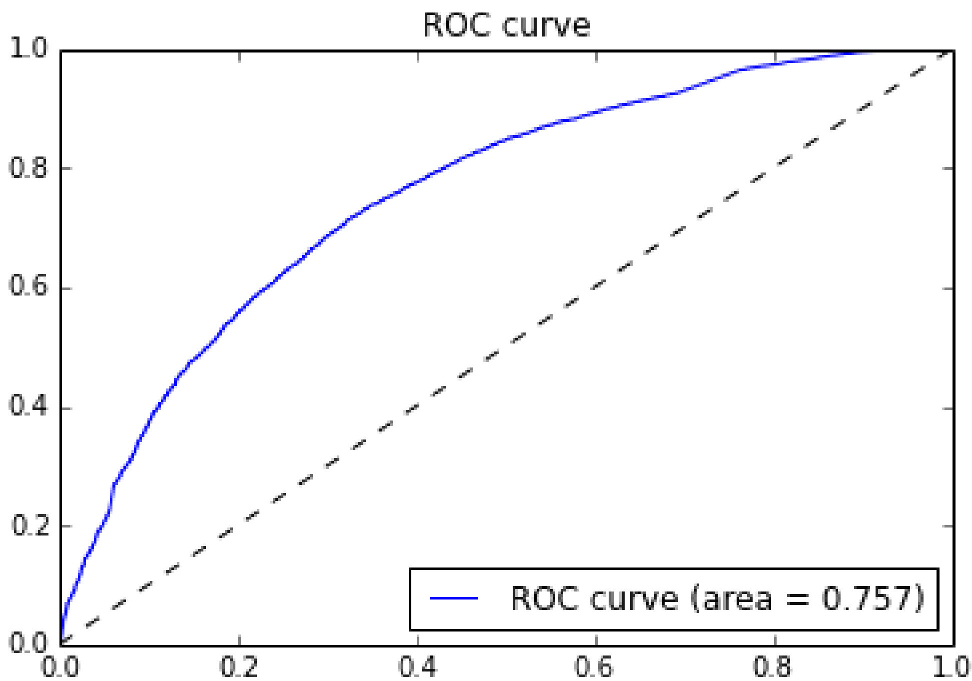
In [34]:
```
from sklearn.metrics import roc_curve, auc
```

In [35]:
```
probas=clf4.fit(X_train,Y_train).predict_proba(X_test)
fpr, tpr, thresholds = roc_curve(Y_test, probas[:,1])
roc_auc=auc(fpr, tpr)
print "Area under the ROC curve: %f" % roc_auc
```

```
Area under the ROC curve: 0.756752
```

In [36]:
```
plot(fpr,tpr,label='ROC curve (area = %0.3f)' % roc_auc)
plot([0,1],[0,1],'k--')
xlim([0.0,1.0])
ylim([0.0,1.0])
title('ROC curve')
legend(loc="lower right")
```

Out [36]: <matplotlib.legend.Legend at 0x1117fd610>



The area under the curve is the overall accuracy of my classifier, about 75.7%. There are two limitations in my data and model. 1) I have a huge imbalance in information on funded and unfunded loans; 2) Prediction based on only the description field could be presumably not enough data.

Part V

Logistic Regression

To make a better prediction, I then looked at the demographic information in the dataset, and chose the following variables to enter my logistic regression model: loan amount, journal entries, sector, gender and pictured. The pictured field is a Boolean variable with True meaning the user has a profile picture and False otherwise.

In [33]:
```
small['sector']=pd.Categorical.from_array(small['sector'])
small['country']=pd.Categorical.from_array(small['country'])
small['gender']=pd.Categorical.from_array(small['gender'])
small['pictured']=pd.Categorical.from_array(small['pictured'])
```

In [15]:
```
import statsmodels.api as sm
```

In [14]:
```
formula = ('status ~ loan_amount + journal_entries + sector + gender + pic
```

In [68]:
```
X=sm.add_constant(X)
```

In [84]:
```
model=sm.GLM.from_formula(formula=formula, data=small,family=sm.families.B
```

In [85]:
```
print model.summary()
```

```
                 Generalized Linear Model Regression Results
==============================================================================
========
Dep. Variable:                 status   No. Observations:
599664
Model:                            GLM   Df Residuals:
599645
Model Family:                Binomial   Df Model:
18
Link Function:                  logit   Scale:
1.0
Method:                          IRLS   Log-Likelihood:
nan
Date:                Sat, 15 Mar 2014   Deviance:
1.3603e+05
Time:                        19:50:30   Pearson chi2:
6.65e+05
No. Iterations:                    14
==============================================================================
======================
                               coef    std err          t      P>|t|
[95.0% Conf. Int.]
------------------------------------------------------------------------------
----------------------
Intercept                    3.8692      0.439      8.820      0.000
3.009     4.729
sector[T.Arts]               0.3410      0.077      4.446      0.000
0.191     0.491
sector[T.Clothing]          -0.4397      0.032    -13.631      0.000
-0.503    -0.376
sector[T.Construction]       0.3585      0.066      5.414      0.000
0.229     0.488
sector[T.Education]          1.0084      0.086     11.746      0.000
0.840     1.177
sector[T.Entertainment]      0.8454      0.273      3.096      0.002
0.310     1.381
sector[T.Food]              -0.0959      0.024     -3.993      0.000
-0.143    -0.049
sector[T.Health]             0.2371      0.090      2.643      0.008
0.061     0.413
sector[T.Housing]           -0.8972      0.031    -28.695      0.000
```

```
-0.958      -0.836
sector[T.Manufacturing]       0.6429      0.089      7.264      0.000
0.469       0.816
sector[T.Personal Use]       -0.4958      0.054     -9.180      0.000
-0.602      -0.390
sector[T.Retail]             -0.3404      0.023    -14.630      0.000
-0.386      -0.295
sector[T.Services]            0.0297      0.034      0.876      0.381
-0.037       0.096
sector[T.Transportation]     -0.1552      0.043     -3.624      0.000
-0.239      -0.071
sector[T.Wholesale]           0.1919      0.184      1.042      0.298
-0.169       0.553
gender[T.M]                  -0.5948      0.016    -36.144      0.000
-0.627      -0.563
pictured[T.True]              0.0400      0.438      0.091      0.927
-0.818       0.898
loan_amount                  -0.0011   1.71e-05    -64.433      0.000
-0.001      -0.001
journal_entries               4.2089      0.071     59.658      0.000
4.071       4.347
================================================================================
======================
```

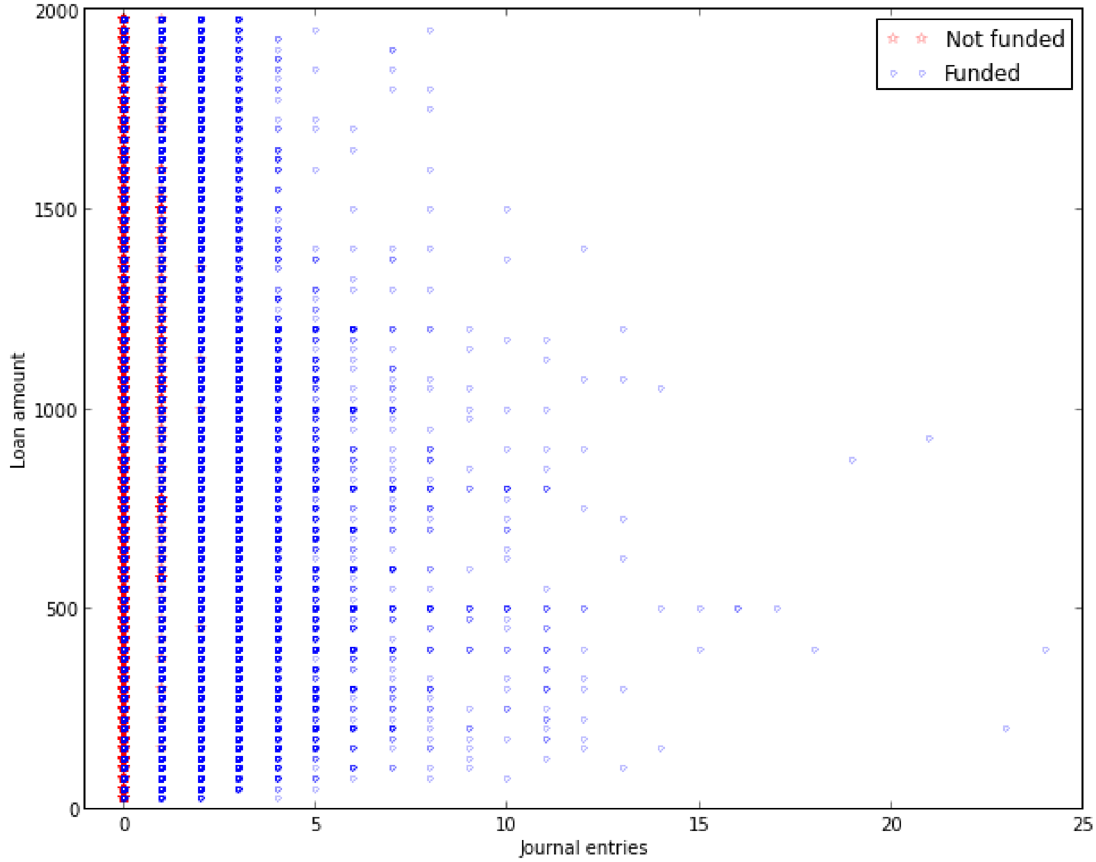
The logistics regression results tells me that an average male is less likely to have a fully funded loan than a similar female with log odd around 0.59. Having more journal entries increases the probability of a loan getting funded by a lot, while having a profile picture does not really help the borrower at all. Surprisingly, having a larger loan amount only lowers the probability of getting funded by a small margin, less than the effect of journal entries.

In [34]:
```
loan_1 = small['loan_amount'][small['status'] == 1]
loan_0 = small['loan_amount'][small['status'] == 0]
journal_1 = small['journal_entries'][small['status'] == 1]
journal_0 = small['journal_entries'][small['status'] == 0]
```

In [35]:
```
fig = plt.figure(figsize = (10, 8))
# Not funded
plt.plot(journal_0, loan_0, '*', label = 'Not funded',
         mfc = 'None', mec='red', alpha = .5)
# Funded
plt.plot(journal_1,loan_1, '.', label = 'Funded',
         mfc = 'None', mec='blue', alpha = .5)

plt.xlim([-1,25])
plt.xlabel('Journal entries')
plt.ylabel('Loan amount')
plt.legend(loc='upper right')
```

Out [35]:
```
<matplotlib.legend.Legend at 0x2f75dbd0>
```



This scatterplot of journal entries and loan amount color coded with red as not funded and blue as funded shows that it is almost unlikely to have an unfunded loan if a user has more than 2 journal entries.

Part VI

Conclusion

Education, entertainment and manufacturing are the three sectors that are funded most often, while housing, personal use and clothing are the least funded. Education, even being one of the larger loan sectors, still gets funded very often. Entertainment and manufacturing also have relatively more expensive loans but at the same time are funded more often as well. The least funded sectors do not have the most expensive or the cheapest loan amounts, and the uses are more personal than job oriented. Description of the loan is the first thing that the lenders will read, however, it only partly determines whether the loan will be funded. Moreover, telling one's story and goal in journals are more important than having a profile picture. This confirms the intuition that people lending using KIVA cares more about the motivation and inspiration of the borrowers and less about the actual amount of money.

Part VII

Reference

Natural Language Processing with Python, by Steven Bird, Ewan Klein and Edward Loper 2009

scikit-learn-book Chap 2 Supervised Learning Text Classification with Naive Bayes.ipynb http://nbviewer.ipython.org/github/gmonce/scikit-learn-book/tree/master/