Udacity Machine Learning Capstone Project

Prepared for: Udacity

Prepared by: Damilola Omifare June 2018.

Project Overview

Home Credit Group

Home Credit B.V. is an international non-bank financial institution founded in 1997 in the Czech Republic.The company operates in 14 countries and focuses on lending primarily to people with little or no credit history. As of 2016 the company has over 15 million active customers, with two-thirds of them in Asia and 7.3 million in China. Major shareholder of company is PPF, a privately held international financial and investment group, which controls an 88.62% stake.

In 1999, Home Credit a.s. was founded in the Czech Republic and in 1999 company expanded to Slovakia. In 2000s company started to expand to Commonwealth of Independent States countries - Russia, Kazakhstan, Ukraine and Belarus.As of 2007 the company was the second largest consumer lender in Russia.In 2010s company expanded to Asia - China, India, Indonesia, Philippines and Vietnam.In 2010 the company was first foreign company to set up as a consumer finance lender in China. In 2015 company launched its operations in the United States of America through a partnership with Sprint Corporation.

Home Credit Group Loans:

Many people struggle to get loans due to insufficient or non-existent credit histories. And, unfortunately, this population is often taken advantage of by untrustworthy lenders.

Home Credit strives to broaden financial inclusion for the unbanked population by providing a positive and safe borrowing experience. In order to make sure this underserved population has a positive loan experience, Home Credit makes use of a variety of alternative data--including telco and transactional information--to predict their clients' repayment abilities.

While Home Credit is currently using various statistical and machine learning methods to make these predictions, they're challenging Kagglers to help them unlock the full potential of their data. Doing so will ensure that clients capable of repayment are not rejected and that loans are given with a principal, maturity, and repayment calendar that will empower their clients to be successful.
Home Credit Group (http://www.homecredit.net/about-us.aspx)

Home Credit Default Risk

This project was inspired by that fact that many people who deserves loan do not get it and ends up in the hands of untrustworthy lenders. This project is a competition from Kaggle. Below is the link: Kaggle | Home Credit Default Risk Competition (https://www.kaggle.com/c/home-credit-default-risk)

Many people struggle to get loans due to insufficient or non-existent credit histories. And, unfortunately, this population is often taken advantage of by untrustworthy lenders.

Source : Kaggle (https://storage.googleapis.com/kaggle-media/competitions/home-credit/about-us-home-credit.jpg)

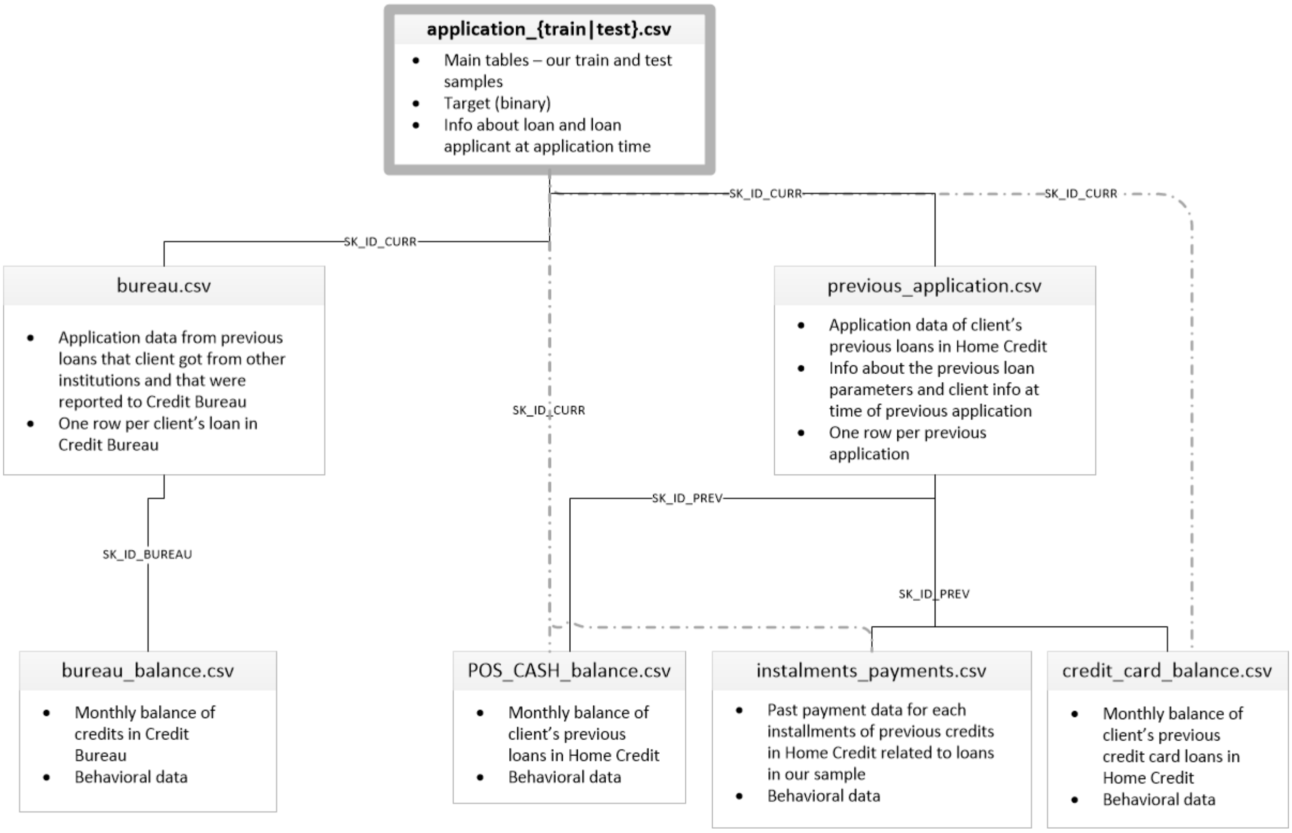
Home Credit strives to broaden financial inclusion for the unbanked population by providing a positive and safe borrowing experience. In order to make sure this underserved population has a positive loan experience, Home Credit makes use of a

variety of alternative data--including telco and transactional information--to predict their clients' repayment abilities. While Home Credit is currently using various statistical and machine learning methods to make these predictions, they're challenging Kagglers to help them unlock the full potential of their data. Doing so will ensure that clients capable of repayment are not rejected and that loans are given with a principal, maturity, and repayment calendar that will empower their clients to be successful.

Datasets and Inputs.

The dataset for this project has been provided by Kaggle.
Data description is below : There are 7 different sources of data:

- **application_train/application_test**: the main training and testing data with information about each loan application at Home Credit. Every loan has its own row and is identified by the feature SK_ID_CURR. The training application data comes with the TARGET indicating **0** if the loan was repaid `Repayers` and `1` for default `Defaulters`
- **bureau**: data concerning client's previous credits from other financial institutions. Each previous credit has its own row in bureau, but one loan in the application data can have multiple previous credits.
- **bureau_balance**: monthly data about the previous credits in bureau. Each row is one month of a previous credit, and a single previous credit can have multiple rows, one for each month of the credit length.
- **previous_application**: previous applications for loans at Home Credit of clients who have loans in the application data. Each current loan in the application data can have multiple previous loans. Each previous application has one row and is identified by the feature SK_ID_PREV.
- **POS_CASH_BALANCE:** monthly data about previous point of sale or cash loans clients have had with Home Credit. Each row is one month of a previous point of sale or cash loan, and a single previous loan can have many rows.
- **credit_card_balance**: monthly data about previous credit cards clients have had with Home Credit. Each row is one month of a credit card balance, and a single credit card can have many rows.
- **installments_payment**: payment history for previous loans at Home Credit. There is one row for every made payment and one row for every missed payment. For more information on what each data represents, please read the PROPOSAL ('/Users/bhetey/version_control/machine-learning/projects/capstone/proposal.pdf'), or Kaggle (https://www.kaggle.com/c/home-credit-default-risk)
- Below is a diagram of how the data are connected.



Problem Statement.

Can you predict how capable each applicant is of repaying a loan ?

- My analysis will be predicting how capable each applicant is at repaying a loan.
  That is predicting the probability that they will default.
- However amongst other things i will also be looking at the :
  1. What income class and family type default the most ?
  2. What family status default the most ?
     This is considered to be a classification problem as the outcome i am looking is just between `1 or 0`, `True or False`. This is called a `binary classification`

METRICS

There are different kind of evaluation metrics we can used since this is a **Classification problem.** Below are some of the metrics :

- Classification accuracy
- Logarithmic Loss
- Area Under ROC Curve
- Confusion Matrix
- Classification report

Not all these evaluation metrics were used in this project but i think it is important for readers to know. **Read more about them** (https://machinelearningmastery.com/metrics-evaluate-machine-learning-algorithms-python/)

Data Exploration

Loading Data

In [5]:
```
from __future__ import division
import pandas as pd # this is to import the pandas module
import numpy as np # importing the numpy module
import os # file system management
import zipfile # module to read ZIP archive files.
from glob import glob
import seaborn as sns
import matplotlib.pyplot as plt

# Figures inline and set visualization style
%matplotlib inline
sns.set()

filenames = glob('/Users/bhetey/.kaggle/competitions/home-credit-de
fault-risk/*.csv')
filenames
```

Out[5]:
```
['/Users/bhetey/.kaggle/competitions/home-credit-default-risk/appl
ication_test.csv',
 '/Users/bhetey/.kaggle/competitions/home-credit-default-risk/Home
Credit_columns_description.csv',
 '/Users/bhetey/.kaggle/competitions/home-credit-default-risk/POS_
CASH_balance.csv',
 '/Users/bhetey/.kaggle/competitions/home-credit-default-risk/cred
it_card_balance.csv',
 '/Users/bhetey/.kaggle/competitions/home-credit-default-risk/inst
allments_payments.csv',
 '/Users/bhetey/.kaggle/competitions/home-credit-default-risk/appl
ication_train.csv',
 '/Users/bhetey/.kaggle/competitions/home-credit-default-risk/bure
au.csv',
 '/Users/bhetey/.kaggle/competitions/home-credit-default-risk/prev
ious_application.csv',
 '/Users/bhetey/.kaggle/competitions/home-credit-default-risk/bure
au_balance.csv',
 '/Users/bhetey/.kaggle/competitions/home-credit-default-risk/samp
le_submission.csv']
```

In [6]:
```
# reading the data with pandas
def reading_csv_file(filename):
    """- This function takes the data path as an input
       - then returns it as a pandas dataframe
    """
    return pd.read_csv(filename)
```

In [7]:
```
app_train = reading_csv_file(filenames[5])
print 'Training data shape :{}'.format(app_train.shape)
app_train.head()
```

```
Training data shape :(307511, 122)
```

Out[7]:

	SK_ID_CURR	TARGET	NAME_CONTRACT_TYPE	CODE_GENDER	FLAG_OWN_
0	100002	1	Cash loans	M	N
1	100003	0	Cash loans	F	N
2	100004	0	Revolving loans	M	Y
3	100006	0	Cash loans	F	N
4	100007	0	Cash loans	M	N

5 rows × 122 columns

Training data has 307511 rows (*each one represents separate loan*) and 112 featurees (columns) including the TARGET(What is to be predicted)

In [8]:
```
y = app_train.TARGET # y is going to be our target variable
y.head()
```

Out[8]:
```
0    1
1    0
2    0
3    0
4    0
Name: TARGET, dtype: int64
```

In [9]:
```
print 'We have {0} Repayers and {1} Defaulters in the TARGET'.forma
t(y.value_counts()[0], y.value_counts()[1])
```

```
We have 282686 Repayers and 24825 Defaulters in the TARGET
```

In [10]:
```
app_test = reading_csv_file(filenames[0])
print 'Testing test contains :{}'.format(app_test.shape)
app_test.head(5)
```

```
Testing test contains :(48744, 121)
```

Out[10]:

	SK_ID_CURR	NAME_CONTRACT_TYPE	CODE_GENDER	FLAG_OWN_CAR	FLA
0	100001	Cash loans	F	N	Y
1	100005	Cash loans	M	N	Y
2	100013	Cash loans	M	Y	Y
3	100028	Cash loans	F	N	Y
4	100038	Cash loans	M	Y	N

5 rows × 121 columns

The Testing set does not have target variable

.Describe enerates descriptive statistics that summarize the central tendency, dispersion and shape of a dataset's distribution, excluding NaN values.

DAYS_BIRTH was originally in days and now it will be converted to years. The columns has negative as they were recorded relative to the current loan application

In [11]:
```
app_train.describe()
```

Out[11]:

	SK_ID_CURR	TARGET	CNT_CHILDREN	AMT_INCOME_TOTAL	AM
count	307511.000000	307511.000000	307511.000000	3.075110e+05	3.07
mean	278180.518577	0.080729	0.417052	1.687979e+05	5.9
std	102790.175348	0.272419	0.722121	2.371231e+05	4.0
min	100002.000000	0.000000	0.000000	2.565000e+04	4.5
25%	189145.500000	0.000000	0.000000	1.125000e+05	2.7
50%	278202.000000	0.000000	0.000000	1.471500e+05	5.1
75%	367142.500000	0.000000	1.000000	2.025000e+05	8.0
max	456255.000000	1.000000	19.000000	1.170000e+08	4.0

8 rows × 106 columns

In [12]:
```
# traing and testing set do not have the same shape.
app_train.shape == app_test.shape
```

Out[12]: False

Data conversion in pandas DataFrame

In [13]:
```
# converted all csv in pandas dataframe.
homeCredit_col_description = reading_csv_file(filenames[1])
pos_cash_balancee = reading_csv_file(filenames[2])
credit_card_bal = reading_csv_file(filenames[3])
installment_payment = reading_csv_file(filenames[4])
app_train = reading_csv_file(filenames[5])
bureau = reading_csv_file(filenames[6])
previous_app = reading_csv_file(filenames[7])
bureau_bal = reading_csv_file(filenames[8])
```

HomeCredit Columns Description gives us the details about each features in the dataset

In [14]:
```
# this is done for indexing of the joint data later
data = (os.listdir("/Users/bhetey/.kaggle/competitions/home-credit-
default-risk/"))
data.remove('.DS_Store')
data.remove('HomeCredit_columns_description.csv')
data.remove('sample_submission.csv')
data
```

Out[14]:
```
['application_test.csv',
 'POS_CASH_balance.csv',
 'credit_card_balance.csv',
 'installments_payments.csv',
 'application_train.csv',
 'bureau.csv',
 'previous_application.csv',
 'bureau_balance.csv']
```

In [15]:
```
# get the whole description of each columns
homeCredit_col_description['Description'].unique()
```

Out[15]:
```
array(['ID of loan in our sample',
       'Target variable (1 - client with payment difficulties: he/
she had late payment more than X days on at least one of the first
Y installments of the loan in our sample, 0 - all other cases)',
       'Identification if loan is cash or revolving',
       'Gender of the client', 'Flag if the client owns a car',
       'Flag if client owns a house or flat',
       'Number of children the client has', 'Income of the client'
,
       'Credit amount of the loan', 'Loan annuity',
```

```
       'For consumer loans it is the price of the goods for which
the loan is given',
       'Who was accompanying client when he was applying for the l
oan',
       'Clients income type (businessman, working, maternity leave
,\x85)',
       'Level of highest education the client achieved',
       'Family status of the client',
       'What is the housing situation of the client (renting, livi
ng with parents, ...)',
       'Normalized population of region where client lives (higher
number means the client lives in more populated region)',
       "Client's age in days at the time of application",
       'How many days before the application the person started cu
rrent employment',
       'How many days before the application did client change his
registration',
       'How many days before the application did client change the
identity document with which he applied for the loan',
       "Age of client's car",
       'Did client provide mobile phone (1=YES, 0=NO)',
       'Did client provide work phone (1=YES, 0=NO)',
       'Did client provide home phone (1=YES, 0=NO)',
       'Was mobile phone reachable (1=YES, 0=NO)',
       'Did client provide email (1=YES, 0=NO)',
       'What kind of occupation does the client have',
       'How many family members does client have',
       'Our rating of the region where client lives (1,2,3)',
       'Our rating of the region where client lives with taking ci
ty into account (1,2,3)',
       'On which day of the week did the client apply for the loan
',
       'Approximately at what hour did the client apply for the lo
an',
       "Flag if client's permanent address does not match contact
address (1=different, 0=same, at region level)",
       "Flag if client's permanent address does not match work add
ress (1=different, 0=same, at region level)",
       "Flag if client's contact address does not match work addre
ss (1=different, 0=same, at region level)",
       "Flag if client's permanent address does not match contact
address (1=different, 0=same, at city level)",
       "Flag if client's permanent address does not match work add
ress (1=different, 0=same, at city level)",
       "Flag if client's contact address does not match work addre
ss (1=different, 0=same, at city level)",
       'Type of organization where client works',
       'Normalized score from external data source',
       'Normalized information about building where the client liv
es, What is average (_AVG suffix), modus (_MODE suffix), median (_
MEDI suffix) apartment size, common area, living area, age of buil
ding, number of elevators, number of entrances, state of the build
ing, number of floor',
```

```
        "How many observation of client's social surroundings with
observable 30 DPD (days past due) default",
        "How many observation of client's social surroundings defau
lted on 30 DPD (days past due) ",
        "How many observation of client's social surroundings with
observable 60 DPD (days past due) default",
        "How many observation of client's social surroundings defau
lted on 60 (days past due) DPD",
        'How many days before application did client change phone',
        'Did client provide document 2', 'Did client provide docume
nt 3',
        'Did client provide document 4', 'Did client provide docume
nt 5',
        'Did client provide document 6', 'Did client provide docume
nt 7',
        'Did client provide document 8', 'Did client provide docume
nt 9',
        'Did client provide document 10', 'Did client provide docum
ent 11',
        'Did client provide document 12', 'Did client provide docum
ent 13',
        'Did client provide document 14', 'Did client provide docum
ent 15',
        'Did client provide document 16', 'Did client provide docum
ent 17',
        'Did client provide document 18', 'Did client provide docum
ent 19',
        'Did client provide document 20', 'Did client provide docum
ent 21',
        'Number of enquiries to Credit Bureau about the client one
hour before application',
        'Number of enquiries to Credit Bureau about the client one
day before application (excluding one hour before application)',
        'Number of enquiries to Credit Bureau about the client one
week before application (excluding one day before application)',
        'Number of enquiries to Credit Bureau about the client one
month before application (excluding one week before application)',
        'Number of enquiries to Credit Bureau about the client 3 mo
nth before application (excluding one month before application)',
        'Number of enquiries to Credit Bureau about the client one
day year (excluding last 3 months before application)',
        'ID of loan in our sample - one loan in our sample can have
0,1,2 or more related previous credits in credit bureau ',
        'Recoded ID of previous Credit Bureau credit related to our
loan (unique coding for each loan application)',
        'Status of the Credit Bureau (CB) reported credits',
        'Recoded currency of the Credit Bureau credit',
        'How many days before current application did client apply
for Credit Bureau credit',
        'Number of days past due on CB credit at the time of applic
ation for related loan in our sample',
        'Remaining duration of CB credit (in days) at the time of a
pplication in Home Credit',
```

```
       'Days since CB credit ended at the time of application in H
ome Credit (only for closed credit)',
       'Maximal amount overdue on the Credit Bureau credit so far
(at application date of loan in our sample)',
       'How many times was the Credit Bureau credit prolonged',
       'Current credit amount for the Credit Bureau credit',
       'Current debt on Credit Bureau credit',
       'Current credit limit of credit card reported in Credit Bur
eau',
       'Current amount overdue on Credit Bureau credit',
       'Type of Credit Bureau credit (Car, cash,...)',
       'How many days before loan application did last information
about the Credit Bureau credit come',
       'Annuity of the Credit Bureau credit',
       'Recoded ID of Credit Bureau credit (unique coding for each
application) - use this to join to CREDIT_BUREAU table ',
       'Month of balance relative to application date (-1 means th
e freshest balance date)',
       'Status of Credit Bureau loan during the month (active, clo
sed, DPD0-30,\x85 [C means closed, X means status unknown, 0 means
no DPD, 1 means maximal did during month between 1-30, 2 means DPD
31-60,\x85 5 means DPD 120+ or sold or written off ] )',
       'ID of previous credit in Home Credit related to loan in ou
r sample. (One loan in our sample can have 0,1,2 or more previous
loans in Home Credit)',
       'Month of balance relative to application date (-1 means th
e information to the freshest monthly snapshot, 0 means the inform
ation at application - often it will be the same as -1 as many ban
ks are not updating the information to Credit Bureau regularly )',
       'Term of previous credit (can change over time)',
       'Installments left to pay on the previous credit',
       'Contract status during the month',
       'DPD (days past due) during the month of previous credit',
       'DPD during the month with tolerance (debts with low loan a
mounts are ignored) of the previous credit',
       'ID of previous credit in Home credit related to loan in ou
r sample. (One loan in our sample can have 0,1,2 or more previous
loans in Home Credit)',
       'Balance during the month of previous credit',
       'Credit card limit during the month of the previous credit'
,
       'Amount drawing at ATM during the month of the previous cre
dit',
       'Amount drawing during the month of the previous credit',
       'Amount of other drawings during the month of the previous
credit',
       'Amount drawing or buying goods during the month of the pre
vious credit',
       'Minimal installment for this month of the previous credit'
,
       'How much did the client pay during the month on the previo
us credit',
       'How much did the client pay during the month in total on t
```

```
he previous credit',
       'Amount receivable for principal on the previous credit',
       'Amount receivable on the previous credit',
       'Total amount receivable on the previous credit',
       'Number of drawings at ATM during this month on the previou
s credit',
       'Number of drawings during this month on the previous credi
t',
       'Number of other drawings during this month on the previous
credit',
       'Number of drawings for goods during this month on the prev
ious credit',
       'Number of paid installments on the previous credit',
       'Contract status (active signed,...) on the previous credit
',
       'DPD (Days past due) during the month on the previous credi
t',
       'DPD (Days past due) during the month with tolerance (debts
with low loan amounts are ignored) of the previous credit',
       'ID of previous credit in Home credit related to loan in ou
r sample. (One loan in our sample can have 0,1,2 or more previous
loan applications in Home Credit, previous application could, but
not necessarily have to lead to credit) ',
       'Contract product type (Cash loan, consumer loan [POS] ,...
) of the previous application',
       'Annuity of previous application',
       'For how much credit did client ask on the previous applica
tion',
       'Final credit amount on the previous application. This diff
ers from AMT_APPLICATION in a way that the AMT_APPLICATION is the
amount for which the client initially applied for, but during our
approval process he could have received different amount - AMT_CRE
DIT',
       'Down payment on the previous application',
       'Goods price of good that client asked for (if applicable)
on the previous application',
       'On which day of the week did the client apply for previous
application',
       'Approximately at what day hour did the client apply for th
e previous application',
       'Flag if it was last application for the previous contract.
Sometimes by mistake of client or our clerk there could be more ap
plications for one single contract',
       'Flag if the application was the last application per day o
f the client. Sometimes clients apply for more applications a day.
Rarely it could also be error in our system that one application i
s in the database twice',
       'Flag Micro finance loan',
       'Down payment rate normalized on previous credit',
       'Interest rate normalized on previous credit',
       'Purpose of the cash loan',
       'Contract status (approved, cancelled, ...) of previous app
lication',
```

```
       'Relative to current application when was the decision abou
t previous application made',
       'Payment method that client chose to pay for the previous a
pplication',
       'Why was the previous application rejected',
       'Who accompanied client when applying for the previous appl
ication',
       'Was the client old or new client when applying for the pre
vious application',
       'What kind of goods did the client apply for in the previou
s application',
       'Was the previous application for CASH, POS, CAR, \x85',
       'Was the previous application x-sell o walk-in',
       'Through which channel we acquired the client on the previo
us application',
       'Selling area of seller place of the previous application',
       'The industry of the seller',
       'Term of previous credit at application of the previous app
lication',
       'Grouped interest rate into small medium and high of the pr
evious application',
       'Detailed product combination of the previous application',
       'Relative to application date of current application when w
as the first disbursement of the previous application',
       'Relative to application date of current application when w
as the first due supposed to be of the previous application',
       'Relative to application date of current application when w
as the first due of the previous application',
       'Relative to application date of current application when w
as the last due date of the previous application',
       'Relative to application date of current application when w
as the expected termination of the previous application',
       'Did the client requested insurance during the previous app
lication',
       'Version of installment calendar (0 is for credit card) of
previous credit. Change of installment version from month to month
signifies that some parameter of payment calendar has changed',
       'On which installment we observe payment',
       'When the installment of previous credit was supposed to be
paid (relative to application date of current loan)',
       'When was the installments of previous credit paid actually
(relative to application date of current loan)',
       'What was the prescribed installment amount of previous cre
dit on this installment',
       'What the client actually paid on previous credit on this i
nstallment'], dtype=object)
```

Checking for missing values.

In [16]:
```
# below is a function to check for missing values.
def check_missing_values(input):
    """
    - This function is to return the selected dataframe.
    - how many columns has missing values
    - how many missing values
    - returns also the total percentage of the missing values
    """
    # checking total missing values
    total_miss_values = input.isnull().sum()

    # percentage of missing values.
    miss_val_percent = total_miss_values/len(input)*100

    # table of total_miss_values and it's percentage
    miss_val_percent_tab = pd.concat([total_miss_values, miss_val_p
ercent], axis=1)

    # columns renamed
    new_col_names = ('Missing values', 'Total missing values in %')
    miss_val_percent_tab.columns = new_col_names
    renamed_miss_val_percent_tab = miss_val_percent_tab

    # descending table sort
    renamed_miss_val_percent_tab = renamed_miss_val_percent_tab[
        renamed_miss_val_percent_tab.iloc[:,1] != 0
    ].sort_values('Total missing values in %', ascending = False).r
ound(1)

    # display information
    print 'The selected dataframe has {} columns.\n'.format(input.s
hape[1])
    print 'There are {} columns missing in the dataset'.format(rena
med_miss_val_percent_tab.shape[0])

    return renamed_miss_val_percent_tab
```

In [17]:
```
missing_value = check_missing_values(app_train)
missing_value.head(10)
```

```
The selected dataframe has 122 columns.

There are 67 columns missing in the dataset
```

Out[17]:

	Missing values	Total missing values in %
COMMONAREA_MEDI	214865	69.9
COMMONAREA_AVG	214865	69.9
COMMONAREA_MODE	214865	69.9
NONLIVINGAPARTMENTS_MEDI	213514	69.4
NONLIVINGAPARTMENTS_MODE	213514	69.4
NONLIVINGAPARTMENTS_AVG	213514	69.4
FONDKAPREMONT_MODE	210295	68.4
LIVINGAPARTMENTS_MODE	210199	68.4
LIVINGAPARTMENTS_MEDI	210199	68.4
LIVINGAPARTMENTS_AVG	210199	68.4

Removing missing values

In [18]:
```
# function is to drop the missing values
def dropping_missing_columns(input_set):
    """this function removes the columns with missing values.
    However input_set is the set you will put inside in the functio
n
    either the training set or the test set
    """
    to_drop_missing_missing_values = [
        col for col in input_set.columns if X[col].isnull().any()
    ]
    return input_set.drop(to_drop_missing_missing_values, axis = 1)
```

Visual Exploratory Data Analysis (EDA)

In [19]:
```
print app_train['TARGET'].value_counts()
app_train.head(5)
```

```
0    282686
1     24825
Name: TARGET, dtype: int64
```

Out[19]:

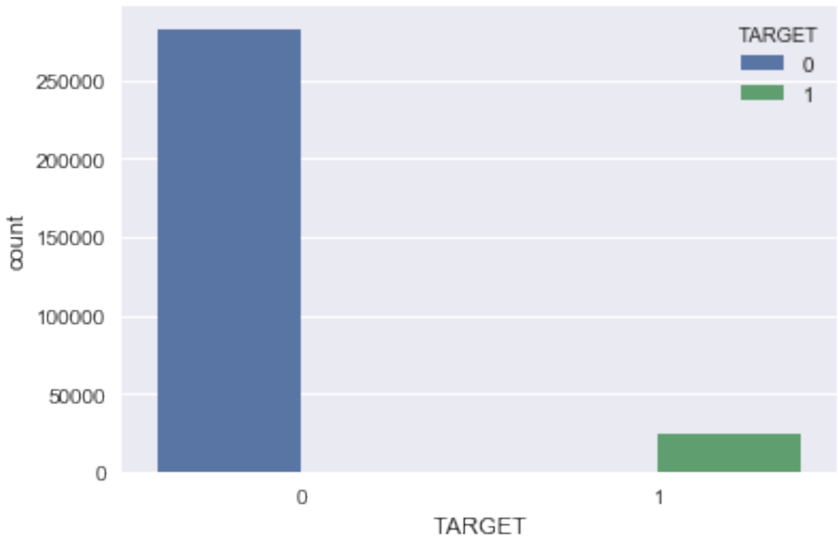
	SK_ID_CURR	TARGET	NAME_CONTRACT_TYPE	CODE_GENDER	FLAG_OWN_
0	100002	1	Cash loans	M	N
1	100003	0	Cash loans	F	N
2	100004	0	Revolving loans	M	Y
3	100006	0	Cash loans	F	N
4	100007	0	Cash loans	M	N

5 rows × 122 columns

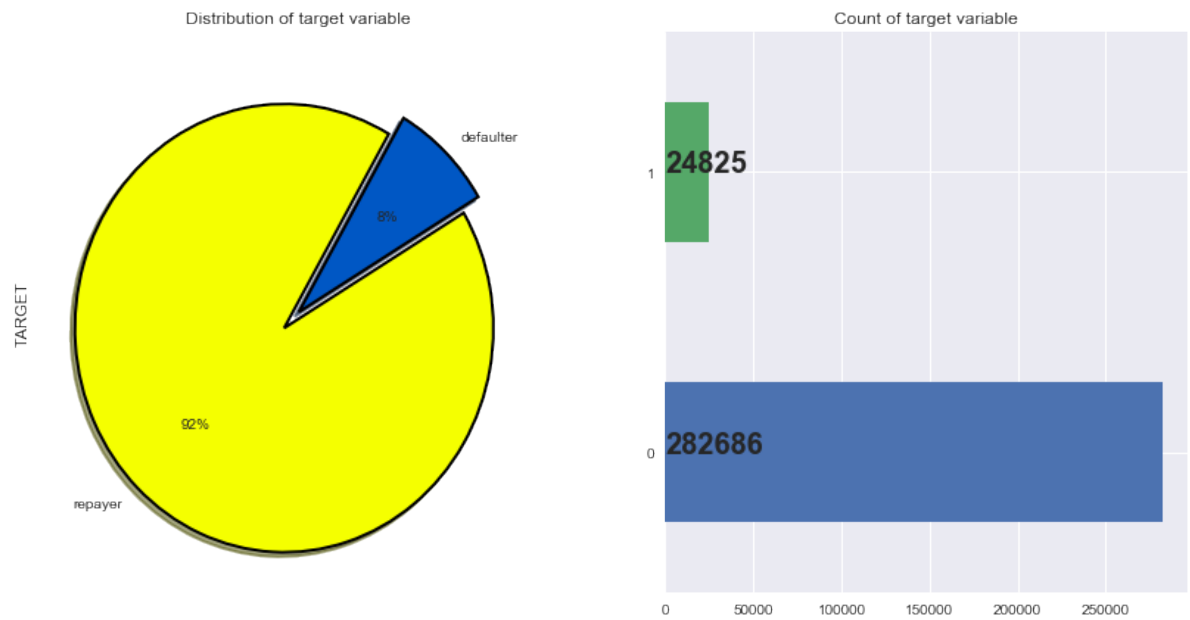
- ### How many people repay loans : **Take away here is that**: Looking at the picture below, **1** for **Defaulter** and **0** for **Repayers**. The image below shows that most applicant pay back the loan. This is what we called [Imbalanced Class Problem (http://www.chioka.in/class-imbalance-problem/)](http://www.chioka.in/class-imbalance-problem/). The differences between Repayer and Defaulter is too big

In [20]:
```
sns.countplot(x='TARGET',hue='TARGET', data=app_train)
```

Out[20]: <matplotlib.axes._subplots.AxesSubplot at 0x1a1525cc90>

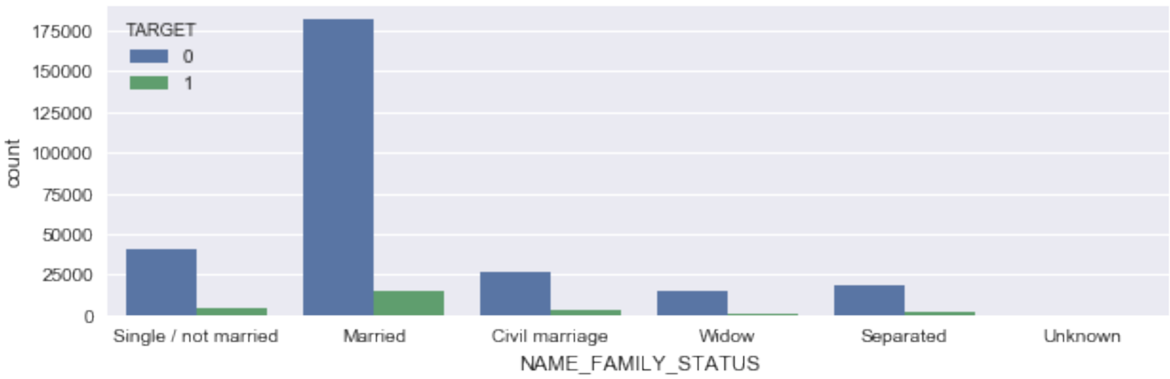


In [21]:
```
plt.figure(figsize=(14,7))
plt.subplot(121)
app_train["TARGET"].value_counts().plot.pie(
    autopct = "%1.0f%%",
    colors = sns.color_palette("prism",7),
    startangle = 60,labels=["repayer","defaulter"],

wedgeprops={"linewidth":2,"edgecolor":"k"},explode=[.1,0],shadow =True)
plt.title("Distribution of target variable")

plt.subplot(122)
ax = app_train["TARGET"].value_counts().plot(kind="barh")

for i,j in enumerate(app_train["TARGET"].value_counts().values):
    ax.text(.7,i,j,weight = "bold",fontsize=20)

plt.title("Count of target variable")
plt.savefig('target variable distribution')
plt.show()
```



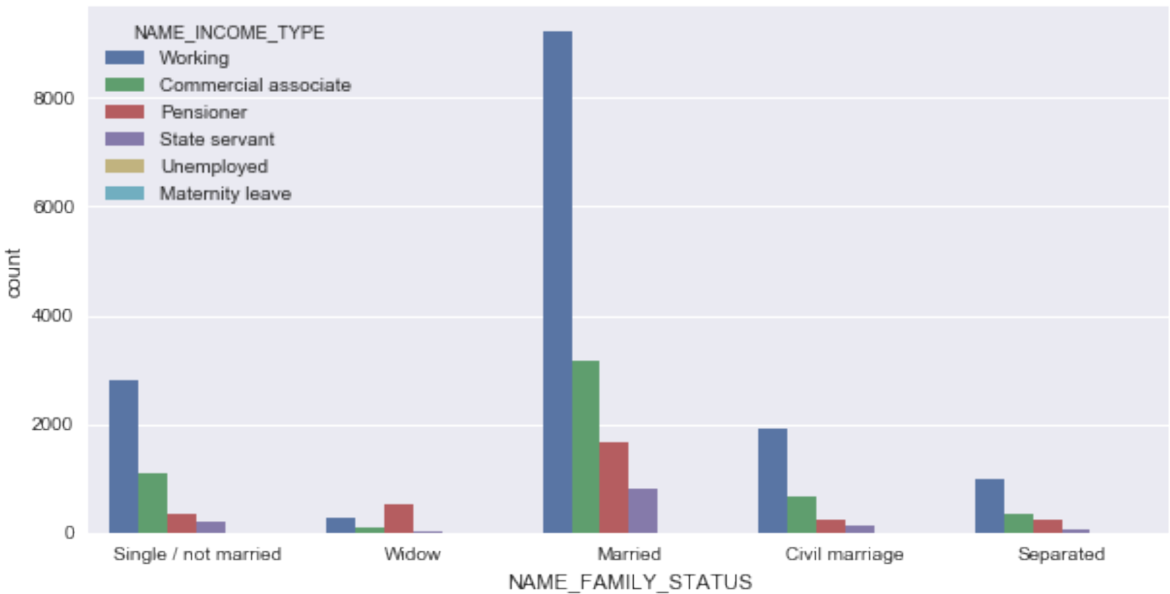
- ### What is the family status of the applicant: In the image, it is shown that more married candidates pay back thier loans

In [22]:
```
plt.figure(figsize=(10,3))
sns.countplot(x = "NAME_FAMILY_STATUS", hue = "TARGET", data = app_
train)
fam_stat_target = pd.crosstab(app_train['NAME_FAMILY_STATUS'], app_
train['TARGET'])
kind_of_applicant = ("Repayers", "Defaulters")
fam_stat_target.columns = kind_of_applicant
fam_stat_target
plt.savefig('NAME_FAMILY_STATUS')
```



- ### What is the Income Class and Family type that default the most :
  The Takeaway: Most married and working class mostly default on loan payment

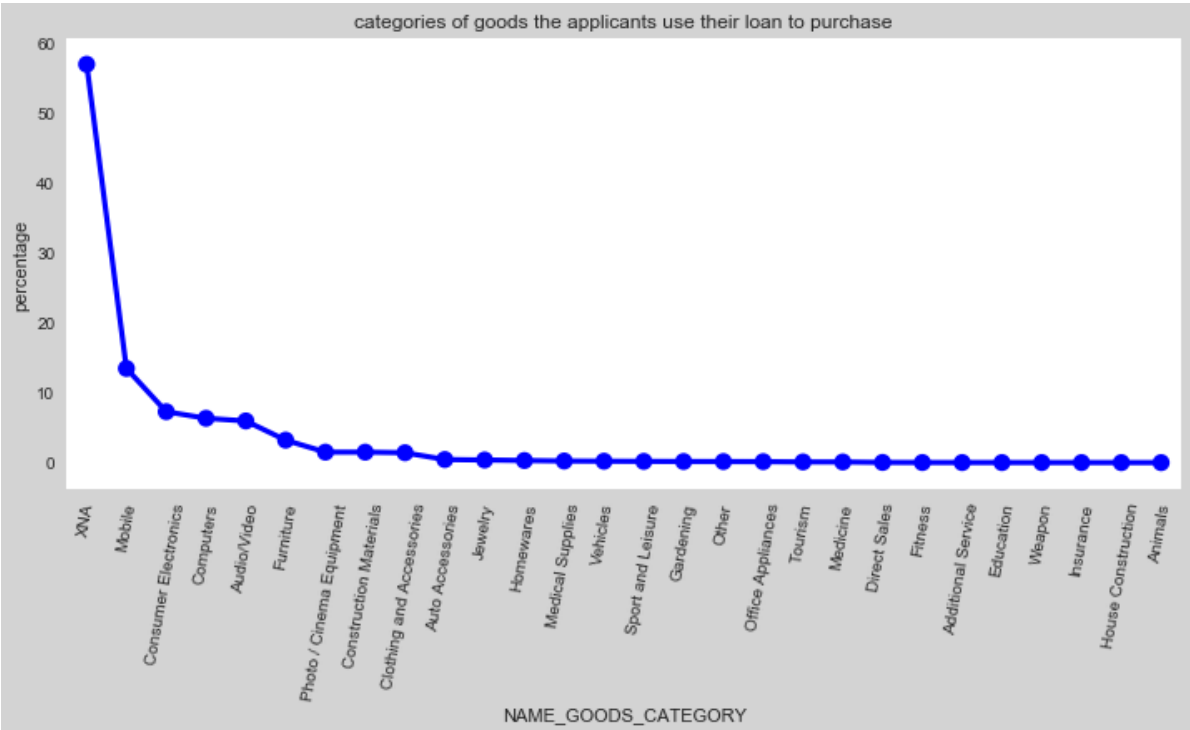
In [23]:
```
plt.figure(figsize= (10,5))# plot the figure
plt.show(sns.countplot(x = "NAME_FAMILY_STATUS",
                       hue = "NAME_INCOME_TYPE" ,
                       # filter the train set by using TARGET colum
n == 1
                       data= app_train.loc[app_train['TARGET'] == 1
]))
```



Take away : For some machine learning models, we have to deal with the missing values buy imputing or dropping either the roles or the columns with the highest percentage of missing values. However we might be loosing some data from them. We also do not know if the data removed will harm the analysis or help it ahead of time until we experiment on them. Algorithmns like **XGBoost** can handle missing data without imputation. It automatically learn how to deal with missing data point. (https://machinelearningmastery.com/data-preparation-gradient-boosting-xgboost-python/)

- Additonal reading (https://arxiv.org/abs/1603.02754)

In [24]:
```
goods_category = previous_app["NAME_GOODS_CATEGORY"].value_counts()
.reset_index()
goods_category["percentage"] = goods_category["NAME_GOODS_CATEGORY"
]*100/goods_category["NAME_GOODS_CATEGORY"].sum()
fig = plt.figure(figsize=(12,5))
ax = sns.pointplot("index","percentage",data=goods_category,color="
blue")
plt.xticks(rotation = 80)
plt.xlabel("NAME_GOODS_CATEGORY")
plt.ylabel("percentage")
plt.title("categories of goods the applicants use their loan to pur
chase")
ax.set_facecolor("w")
fig.set_facecolor('lightgrey')
#plt.savefig('goods categories')
```



Dealing with features

Obviously we have 3 data types : Numeric and Non-numeric (e.g Text) called *object*.

Numeric can be of discrete time or continuous time horizon. Non_numeric are variables containing label values rather numeric values. (https://machinelearningmastery.com/why-one-hot-encode-data-in-machine-learning/) They are sometimes called nominal (https://en.wikipedia.org/wiki/Nominal_category)

In [25]:
```
app_train.get_dtype_counts() # Shows the numbers of types of values
```

Out[25]:
```
float64    65
int64      41
object     16
dtype: int64
```

Looking at the dataset with object type, below is the total number of object. However since we want to work with them we will need to hot encode them.

However this depends on personal view. it depend on how big the categorical variables are.

One of the major problems with categorical data is that only few machine learning alogorithms works with them without any special form of implementation while others needs some implementation where the data needs to be encoded into numeric variables.

How to convert categorical data into numerical data:

- **Integer Encoding** *where integer values have a natural ordered relationship between each other and machine learning algorithms may be able to understand and harness this relationship.*
- **One-Hot Encoding** *where the integer encoded variable is removed and a new binary variable is added for each unique integer value.*

Read more (https://machinelearningmastery.com/why-one-hot-encode-data-in-machine-learning/)

In [26]:
```
print 'There are {} columns with object type'.format(len(app_train.
select_dtypes('object').nunique()))
```

```
There are 16 columns with object type
```

Checking the number of unique class in each object column

In [27]:
```
app_train.select_dtypes('object').apply(pd.Series.nunique, axis = 0
)
```

Out[27]:
```
NAME_CONTRACT_TYPE             2
CODE_GENDER                    3
FLAG_OWN_CAR                   2
FLAG_OWN_REALTY                2
NAME_TYPE_SUITE                7
NAME_INCOME_TYPE               8
NAME_EDUCATION_TYPE            5
NAME_FAMILY_STATUS             6
NAME_HOUSING_TYPE              6
OCCUPATION_TYPE               18
WEEKDAY_APPR_PROCESS_START     7
ORGANIZATION_TYPE             58
FONDKAPREMONT_MODE             4
HOUSETYPE_MODE                 3
WALLSMATERIAL_MODE             7
EMERGENCYSTATE_MODE            2
dtype: int64
```

In [28]:
```
(app_train['DAYS_BIRTH']/-365).describe()
```

Out[28]:
```
count    307511.000000
mean         43.936973
std          11.956133
min          20.517808
25%          34.008219
50%          43.150685
75%          53.923288
max          69.120548
Name: DAYS_BIRTH, dtype: float64
```

Looking at the result above everything seems okay. Cannot seems to find any outlier in this analysis

DAYS_EMPOYED: How many days before the application the person started current employment'

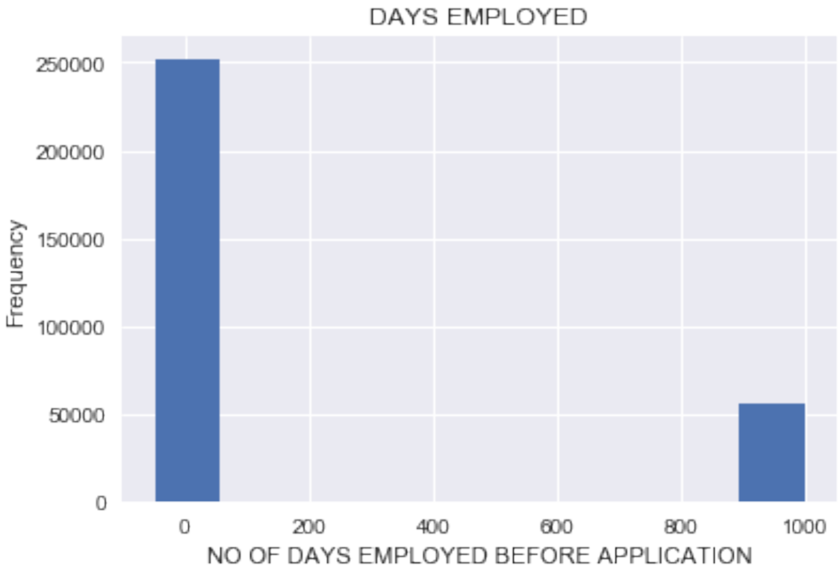
This is also relative to the current loan application

In [29]:
```
years_employed = (app_train['DAYS_EMPLOYED']/365)
years_employed.describe()
```

Out[29]:
```
count    307511.000000
mean        174.835742
std         387.056895
min         -49.073973
25%          -7.561644
50%          -3.323288
75%          -0.791781
max        1000.665753
Name: DAYS_EMPLOYED, dtype: float64
```

In [30]:
```
(years_employed.plot.hist(title = 'DAYS EMPLOYED'))
plt.xlabel('NO OF DAYS EMPLOYED BEFORE APPLICATION')
```

Out[30]:
```
Text(0.5,0,'NO OF DAYS EMPLOYED BEFORE APPLICATION')
```



Looking at the image above, 1000 years does not seem right. We will use imputation to solve this. There seems to be some kind of anomalies here.

Checking correlation of the data.

It helps to show possible relationship within our data. This article helps in interpreting correlation (http://www.statstutor.ac.uk/resources/uploaded/pearsons.pdf), How to interpret a Correlation Coefficient (https://www.dummies.com/education/math/statistics/how-to-interpret-a-correlation-coefficient-r/)

In [31]:
```
data_corr = app_train.corr()['TARGET'].sort_values()
print 'These are samples of negative correlations : \n',data_corr.h
ead(20)
```

```
These are samples of negative correlations :
EXT_SOURCE_3                   -0.178919
EXT_SOURCE_2                   -0.160472
EXT_SOURCE_1                   -0.155317
DAYS_EMPLOYED                  -0.044932
FLOORSMAX_AVG                  -0.044003
FLOORSMAX_MEDI                 -0.043768
FLOORSMAX_MODE                 -0.043226
AMT_GOODS_PRICE                -0.039645
REGION_POPULATION_RELATIVE     -0.037227
ELEVATORS_AVG                  -0.034199
ELEVATORS_MEDI                 -0.033863
FLOORSMIN_AVG                  -0.033614
FLOORSMIN_MEDI                 -0.033394
LIVINGAREA_AVG                 -0.032997
LIVINGAREA_MEDI                -0.032739
FLOORSMIN_MODE                 -0.032698
TOTALAREA_MODE                 -0.032596
ELEVATORS_MODE                 -0.032131
LIVINGAREA_MODE                -0.030685
AMT_CREDIT                     -0.030369
Name: TARGET, dtype: float64
```

In [32]:
```
print 'These are samples of positive correlations : \n', data_corr.
tail(20)
```

```
These are samples of positive correlations :
OBS_30_CNT_SOCIAL_CIRCLE        0.009131
CNT_FAM_MEMBERS                 0.009308
CNT_CHILDREN                    0.019187
AMT_REQ_CREDIT_BUREAU_YEAR      0.019930
FLAG_WORK_PHONE                 0.028524
DEF_60_CNT_SOCIAL_CIRCLE        0.031276
DEF_30_CNT_SOCIAL_CIRCLE        0.032248
LIVE_CITY_NOT_WORK_CITY         0.032518
OWN_CAR_AGE                     0.037612
DAYS_REGISTRATION               0.041975
FLAG_DOCUMENT_3                 0.044346
REG_CITY_NOT_LIVE_CITY          0.044395
FLAG_EMP_PHONE                  0.045982
REG_CITY_NOT_WORK_CITY          0.050994
DAYS_ID_PUBLISH                 0.051457
DAYS_LAST_PHONE_CHANGE          0.055218
REGION_RATING_CLIENT            0.058899
REGION_RATING_CLIENT_W_CITY     0.060893
DAYS_BIRTH                      0.078239
TARGET                          1.000000
Name: TARGET, dtype: float64
```

ONE-HOT ENCODING

Let's **One-hot Encode** the categorical variable
We need to import the module from scikit-learn library

In [33]:
```
app_train.select_dtypes('object').columns
```

Out[33]:
```
Index([u'NAME_CONTRACT_TYPE', u'CODE_GENDER', u'FLAG_OWN_CAR',
       u'FLAG_OWN_REALTY', u'NAME_TYPE_SUITE', u'NAME_INCOME_TYPE'
,
       u'NAME_EDUCATION_TYPE', u'NAME_FAMILY_STATUS', u'NAME_HOUSI
NG_TYPE',
       u'OCCUPATION_TYPE', u'WEEKDAY_APPR_PROCESS_START', u'ORGANI
ZATION_TYPE',
       u'FONDKAPREMONT_MODE', u'HOUSETYPE_MODE', u'WALLSMATERIAL_M
ODE',
       u'EMERGENCYSTATE_MODE'],
      dtype='object')
```

In [34]:
```
len (app_train.columns) == len(app_train.select_dtypes('object').co
lumns)
```

Out[34]:
```
False
```

In [35]:
```
from sklearn.preprocessing import OneHotEncoder

one_hot_encoded_app_train = pd.get_dummies(app_train)
one_hot_encoded_app_test = pd.get_dummies(app_test)

print 'Shape of the training set after one hot encoding {}'.format(
one_hot_encoded_app_train.shape)
print 'Shape of the test set after one hot encoding {}'.format(one_
hot_encoded_app_test.shape)
```

```
Shape of the training set after one hot encoding (307511, 246)
Shape of the test set after one hot encoding (48744, 242)
```

In [36]:
```
app_train.shape == one_hot_encoded_app_train.shape
```

Out[36]:
```
False
```

Looking at the analysis above is obvious that **One-Hot Encoding** has added extra features to the original ones we have hereby leaving our data unaligned.

We need to have same features in both the training and testing data for our machine learning model to work if not we will get error when running the algorithm.

STEPS TAKING:

- I decided to remove any column that is present on the training set but not on our testing set.
- Intuitively the **y** which is our **TARGET** is expected to be removed as well but will add it back

In [37]:
```
#https://pandas.pydata.org/pandas-docs/version/0.21/generated/panda
s.DataFrame.align.html
one_hot_encoded_app_train, one_hot_encoded_app_test = one_hot_encod
ed_app_train.align(one_hot_encoded_app_test,

join='inner', axis=1)

print 'Shape of the training set after alignment {}'.format(one_hot
_encoded_app_train.shape)
print 'Shape of the test set after alignment {}'.format(one_hot_enc
oded_app_test.shape)
```

```
Shape of the training set after alignment (307511, 242)
Shape of the test set after alignment (48744, 242)
```

In [38]:
```
one_hot_encoded_app_train['TARGET'] = y # adding it back to the dat
a
```

In [39]:
```
# dropping the target to get our X
X = one_hot_encoded_app_train.drop(['TARGET'], axis=1)
```

In [40]:
```
#assigned a variable to data after dropping the missing values
after_removing_missing_values = dropping_missing_columns(X)
test_removing_missing_values = dropping_missing_columns(one_hot_enc
oded_app_test)
```

In [41]:
```
print 'The shape of training set after removing missing values : {}
'.format(after_removing_missing_values.shape)
print 'The shape of testing set after removing missing values :{}'.
format(test_removing_missing_values.shape)
```

```
The shape of training set after removing missing values : (307511,
181)
The shape of testing set after removing missing values :(48744, 18
1)
```

Looking at the dataset now after removing the **NaN** in the data, we have the columns reduced to **181 columns**

CLASSIFICATION MODELS

Classification depends on whether the variables we are trying to predict are **Binary or Non-Binary**.

Binary variables are those variables where the outcome we are looking are either 1 or 0, True or False.

Non-Binary variables are those variables where the outcome we are looking are categorical. for example looking at the dataset and predicting where the color of the dress of a person will be `Yellow, Brown or Blue`

Binary Classification Model :

- Logistic regression
- Decision Trees (Bagging, Boosted)
- Random Forest
- Support Vector Machine (SVM) : *good for anomaly detection especially in large feature sets*

Non- Binary / Multiclassification Classification Model:

- Adaboost
- Random Forest
- Decision Tree
- Neural Networks

Considering choosing an algorithm, :

- Take note of the accuracy
- Training time
- Linearity
- Number of parameters
- Number of features

The Machine Learning Algorithm Cheat Sheet (https://docs.microsoft.com/en-us/azure/machine-learning/studio/algorithm-choice)

BENCHMARK

According to the word itself `Benchmark`, it is like a previous solution either as a standard, or from previous analysis or from a research paper that you are trying to beat.

Normal a `benchmark` should be given, that testing your prediction against their actual values (*this you will not be provided though.*)

When it is not given, we need a **Baseline model**, that is something we can work with, with the intention of beating. One i can think of is **Random guess** which has a probability of 0.5 for each outcome. That is either an applicant will default or not, both has equal chances or occuring.

Naive Predictor

Suppose we have a model that is always predicting **1** (`Defaulters`), what would that model' accuracy and F-score be on this dataset ? This is a Naive Predictor that is showing what a baseline model without any intelligence.

In [43]:

```
# import the module needed
from sklearn.metrics import accuracy_score,recall_score,fbeta_score
,precision_score

predicted_target = [1 for a in y]
# Calculate accuracy, precision and recall
accuracy = accuracy_score(y_true=y, y_pred=predicted_target)
recall = recall_score(y_true=y,y_pred=predicted_target)
precision = precision_score(y_true=y,y_pred=predicted_target)

# TODO: Calculate F-score using the formula above for beta = 0.5 an
d correct values for precision and recall.
beta = 0.5
fscore = (1+beta**2)*(accuracy*recall)/(beta**2*accuracy+recall)

# Print the results
print("Naive Predictor: [Accuracy score: {:.4f}, F-score: {:.4f}]".
format(accuracy, fscore))
```

```
Naive Predictor: [Accuracy score: 0.0807, F-score: 0.0989]
```

Data Preprocessing

Data preprocessing is a data mining technique that involves transforming raw data into an understandable format due to real world data inconsistency, incompleteness. Data preprocessing is a proven method of resolving such issues.
[What is data preprocessing ? (https://www.techopedia.com/definition/14650/data-preprocessing)](https://www.techopedia.com/definition/14650/data-preprocessing) Data preprocessing includes cleaning, Instance selection, normalization, transformation, feature extraction and selection, etc

Why am i doing data preprocessing ?

- This is because we have a lot of missing values in the dataset.

Fortunately for us, we have a package from the [Scikit learn documentation page (http://scikit-learn.org/stable/modules/preprocessing.html)](http://scikit-learn.org/stable/modules/preprocessing.html). There are various modules of preprocessing that can be used but i used `MinMaxScaler` which scales features to range. Here is the range 0 and 1.

Going back to the data exploration part, where you had anomalies in `DAYS_EMPLYED` with max age of **`1000`**, and **`-49`**.

In [44]:
```
# day employed here is not divided by 365
anomalies = app_train[app_train['DAYS_EMPLOYED'] == 365243] # this
is also the max from using describe
without_anomalies = app_train[app_train['DAYS_EMPLOYED'] != 365243]

# checking anomalies with the target variable and calculating the p
ercentage
print 'Anomalies percentages with respect to mean {}'.format(anomal
ies['TARGET'].mean() * 100)
print 'Non-anomalies percentage with respect to mean {} '.format(wi
thout_anomalies['TARGET'].mean() * 100)
# length of anomalies
print 'Length of anomalies are {} values'.format(len(anomalies))

# here is one helpful kernel
# https://www.kaggle.com/willkoehrsen/start-here-a-gentle-introduct
ion
```

```
Anomalies percentages with respect to mean 5.39964604327
Non-anomalies percentage with respect to mean 8.65997453765
Length of anomalies are 55374 values
```

In [45]:
```
# to fix anomalies i replaced anomalies values with np.Nan
# note what you do for the training do it also for the testing set.
app_train['DAYS_EMPLOYED'].replace({365243: np.nan}, inplace = True
)
app_test['DAYS_EMPLOYED'].replace({365243: np.nan}, inplace = True)
```

Imputing Nan values

In [85]:
```
from sklearn.preprocessing import MinMaxScaler, Imputer

# making a copy for the data before imputing
train_tobe_imputed = one_hot_encoded_app_train.copy()
test_tobe_imputed = one_hot_encoded_app_test.copy()
new_y = y.copy() # a copy of our target

# dropping the target columns before imputing
new_X = train_tobe_imputed.drop(['TARGET'], axis=1)

# calling imputer and transforming the data
imputer = Imputer(strategy='median')
transformed_X = imputer.fit_transform(new_X)
transformed_test_X = imputer.fit_transform(test_tobe_imputed)

print 'Transformed training set :{}'.format(transformed_X.shape)
print 'Transformed testing set :{}'.format(transformed_test_X.shape
)
print 'The data is back to the same shape we had during the Hot cod
ing'
```

```
Transformed training set :(307511, 242)
Transformed testing set :(48744, 242)
The data is back to the same shape we had during the Hot coding
```

Now that the anomalies case has been solved. One thing to take note of is that when :

- when we remove missing values from a dataset before imputing it , we need to do the same after imputing it on training and testing set so as to avoid inconsistency in data shape

Implementation

Logistic Regression Model

This is my first model.
C is used to control overfitting and a small tends to reduce overfitting

Model Evaluation using a validation set

- Logistic regression using the data set with dropped missing values.

Normalization

- fit the data set
- transform it

In [46]:
```
# Normalize time series data
from pandas import Series
from sklearn.preprocessing import MinMaxScaler

# Scale each feature to 0-1
scaler = MinMaxScaler(feature_range=(0, 1))
# fit on training
scaler = scaler.fit(after_removing_missing_values)
#transform train and test
scaled_train_withoutMissingValues = scaler.transform(after_removing
_missing_values)
scaled_test_withoutMissingValues = scaler.transform(test_removing_m
issing_values)
```

In [48]:
```
# Import statements
from sklearn.metrics import classification_report
from sklearn import model_selection
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score
from sklearn.metrics import confusion_matrix
from sklearn.metrics import r2_score
from sklearn.cross_validation import train_test_split
from sklearn import metrics
from sklearn.metrics import roc_auc_score

X_train, X_test, y_train, y_test = train_test_split(
    scaled_train_withoutMissingValues, y, test_size=0.25, random_st
ate=42)

# instantiate a logistic regression model, and fit with X and y
model = LogisticRegression()

# evaluate the model by splitting into train and test sets
model.fit(X_train, y_train)
print 'This accuracy seems good {} but need to check further for th
e prediction and on testing set'.format(
    model.score(X_train, y_train))

prob_for_checking = model.predict_proba(X_test)
y_pred = model.predict(X_test)
matrix = confusion_matrix(y_test, y_pred)
print y_pred
print matrix
print model.score(X_test, y_test)
print roc_auc_score(y_test, prob_for_checking[:,1])
```

```
This accuracy seems good 0.919204970668 but need to check further
for the prediction and on testing set
[0 0 0 ..., 0 0 0]
[[70686     1]
 [ 6191     0]]
0.919456801686
0.666741360297
```

Logistic regression on imputed datapoint

In [114]:
```
scalerImput = scaler.fit(transformed_X)
# scaling and transformed the features
imputed_train = scaler.transform(transformed_X)
imputed_test = scaler.transform(transformed_test_X)

#splitting into training , testing set
X_train_lr_imputed, X_test_lr_imputed, y_train_lr_imputed, y_test_l
r_imputed = train_test_split(
    transformed_X, y, test_size=0.25, random_state=42)


# instantiate a logistic regression model, and fit with X and y
model_lr = LogisticRegression()

# evaluate the model by splitting into train and test sets
model_lr.fit(X_train_lr_imputed, y_train_lr_imputed)
print 'This accuracy seems good {} but need to check further for th
e prediction and on testing set'.format(
    model_lr.score(X_train_lr_imputed, y_train_lr_imputed))

prob_for_checking_imputed = model_lr.predict_proba(X_test_lr_impute
d)
y_pred_lr_imputed = model_lr.predict(X_test_lr_imputed)
matrix_lr_imputed = confusion_matrix(y_test_lr_imputed, y_pred_lr_i
mputed)
print y_pred_lr_imputed
print matrix_lr_imputed
print model_lr.score(X_test_lr_imputed, y_test_lr_imputed)
print roc_auc_score(y_test_lr_imputed, prob_for_checking_imputed[:,
1])
```

```
This accuracy seems good 0.919196298882 but need to check further
for the prediction and on testing set
[0 0 0 ..., 0 0 0]
[[70686     1]
 [ 6191     0]]
0.919456801686
0.623066552248
```

Refinement

Understanding how decision regions change when using different regularization values. Remember that we use paramter C as our regularization parameter.

- Parameter`C = 1/λ`

`**Lambda (λ)**` controls the trade-off between allowing the model to increase it's complexity as much as it wants with trying to keep it simple. For example, if λ is very low or 0, the model will have enough power to increase it's complexity (overfit) by assigning big values to the weights for each parameter. If, in the other hand, we increase the value of λ, the model will tend to underfit, as the model will become too simple

`**Lambda (λ)**` controls the trade-off between allowing the model to increase it's complexity as much as it wants with trying to keep it simple. For example, if λ is very low or 0, the model will have enough power to increase it's complexity (overfit) by assigning big values to the weights for each parameter. If, in the other hand, we increase the value of λ, the model will tend to underfit, as the model will become too simple.

Parameter `C` will work the other way around. For small values of C, we increase the regularization strength which will create simple models which underfit the data. For big values of C, we low the power of regularization which imples the model is allowed to increase it's complexity, and therefore, overfit the data.

I found this article on [Understanding how decision regions change when using different regularization values (https://www.kaggle.com/joparga3/2-tuning-parameters-for-logistic-regression)](https://www.kaggle.com/joparga3/2-tuning-parameters-for-logistic-regression)

In [98]:
```
X_train_lr, X_test_lr, y_train_lr, y_test_lr = train_test_split(
    scaled_train_withoutMissingValues, y, test_size=0.25, random_st
ate=42)

# instantiate a logistic regression model, and fit with X and y
parameter_for_c = [0.001,0.01,0.1,1,10,100]
for b in parameter_for_c:

    model = LogisticRegression(penalty = 'l2', C = b)

    # evaluate the model by splitting into train and test sets
    model.fit(X_train_lr, y_train_lr)
    print 'This accuracy seems good {} but need to check further fo
r the prediction and on testing set'.format(
        model.score(X_train, y_train))

    prob_for_checking = model.predict_proba(X_test_lr)
    y_pred_lr = model.predict(X_test_lr)
    matrix = confusion_matrix(y_test_lr, y_pred_lr)
    print 'This is the parameter for :', b
    print matrix
    print model.score(X_test_lr, y_test_lr)
    print roc_auc_score(y_test, prob_for_checking[:,1])


#predicting_with_test = model.predict(scaled_test_withoutMissingVal
ues)
#predicting_probability = model.predict_proba(X_test)
#matrix = confusion_matrix(y_test, y_pred)
#scoring = accuracy_score(y_test, y_pred)
#r_score = r2_score(y_test, y_pred)
#report = classification_report(y_test, y_pred)
#print report
#print matrix
#print 'Accuracy of the model :{}'.format(scoring)
#print 'R2 Score for the prediction :'.format(r_score)
#print metrics.roc_auc_score(y_test, predicting_probability[:, 1])
```

```
This accuracy seems good 0.919204970668 but need to check further
for the prediction and on testing set
This is the parameter for : 0.001
[[70687     0]
 [ 6191     0]]
0.919469809308
0.653769653633
This accuracy seems good 0.919204970668 but need to check further
for the prediction and on testing set
This is the parameter for : 0.01
[[70687     0]
 [ 6191     0]]
0.919469809308
0.664319774881
This accuracy seems good 0.919204970668 but need to check further
for the prediction and on testing set
This is the parameter for : 0.1
[[70687     0]
 [ 6191     0]]
0.919469809308
0.665370902842
This accuracy seems good 0.919204970668 but need to check further
for the prediction and on testing set
This is the parameter for : 1
[[70686     1]
 [ 6191     0]]
0.919456801686
0.666741360297
This accuracy seems good 0.919191962989 but need to check further
for the prediction and on testing set
This is the parameter for : 10
[[70686     1]
 [ 6190     1]]
0.919469809308
0.669431642609
This accuracy seems good 0.919187627096 but need to check further
for the prediction and on testing set
This is the parameter for : 100
[[70685     2]
 [ 6190     1]]
0.919456801686
0.670258561716
```

In [93]:
```
scalerImput = scaler.fit(transformed_X)
# scaling and transformed the features
imputed_train = scaler.transform(transformed_X)
imputed_test = scaler.transform(transformed_test_X)

# splitting into traning and testing set.
X_train_lr_imputed, X_test_lr_imputed, y_train_lr_imputed, y_test_l
r_imputed = train_test_split(
    transformed_X, y, test_size=0.25, random_state=42)

# instantiate a logistic regression model, and fit with X and y
parameter_for_c = [0.001,0.01,0.1,1,10,100]

for b in parameter_for_c:

    model = LogisticRegression(penalty = 'l2', C = b)

    model.fit(imputed_train, y)

    # instantiate a logistic regression model, and fit with X and y
    model = LogisticRegression()

    # evaluate the model by splitting into train and test sets
    model.fit(X_train_lr_imputed, y_train_lr_imputed)
    print 'This accuracy seems good {} but need to check further fo
r the prediction and on testing set'.format(
        model.score(X_train_lr_imputed, y_train_lr_imputed))

    prob_for_checking_imputed = model.predict_proba(X_test_lr_imput
ed)
    y_pred_lr_imputed = model.predict(X_test_lr_imputed)
    matrix_lr_imputed = confusion_matrix(y_test_lr_imputed, y_pred_
lr_imputed)
    print 'This is the parameter for :', b
    print y_pred_lr_imputed
    print matrix_lr_imputed
    print model.score(X_test_lr_imputed, y_test_lr_imputed)
    print roc_auc_score(y_test_lr_imputed, prob_for_checking_impute
d[:,1])
```

```
This accuracy seems good 0.919196298882 but need to check further
for the prediction and on testing set
This is the parameter for : 0.001
[0 0 0 ..., 0 0 0]
[[70686     1]
 [ 6191     0]]
0.919456801686
0.623066552248
This accuracy seems good 0.919196298882 but need to check further
for the prediction and on testing set
This is the parameter for : 0.01
[0 0 0 ..., 0 0 0]
[[70686     1]
 [ 6191     0]]
0.919456801686
0.623066552248
This accuracy seems good 0.919196298882 but need to check further
for the prediction and on testing set
This is the parameter for : 0.1
[0 0 0 ..., 0 0 0]
[[70686     1]
 [ 6191     0]]
0.919456801686
0.623066552248
This accuracy seems good 0.919196298882 but need to check further
for the prediction and on testing set
This is the parameter for : 1
[0 0 0 ..., 0 0 0]
[[70686     1]
 [ 6191     0]]
0.919456801686
0.623066552248
This accuracy seems good 0.919196298882 but need to check further
for the prediction and on testing set
This is the parameter for : 10
[0 0 0 ..., 0 0 0]
[[70686     1]
 [ 6191     0]]
0.919456801686
0.623066552248
This accuracy seems good 0.919196298882 but need to check further
for the prediction and on testing set
This is the parameter for : 100
[0 0 0 ..., 0 0 0]
[[70686     1]
 [ 6191     0]]
0.919456801686
0.623066552248
```

Model evaluation and validation

Model Evaluation Using Cross-Validation

Now let's try 10-fold cross-validation, to see if the accuracy holds up more rigorously.

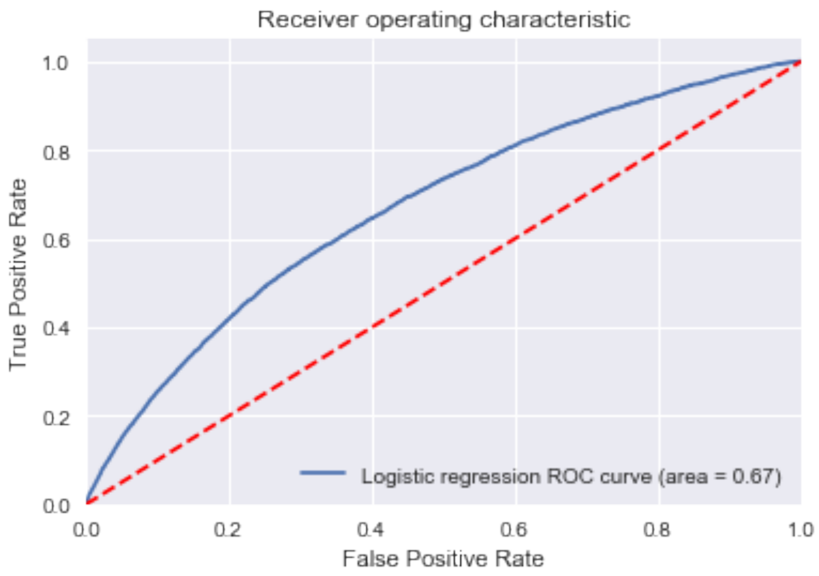
In [104]:
```
# evaluate the model using 10-fold cross-validation
from sklearn.cross_validation import cross_val_score
predicting_with_test = model.predict(scaled_test_withoutMissingValu
es)

scores = cross_val_score(LogisticRegression(penalty = 'l2',C=0.001)
,
                         scaled_train_withoutMissingValues, y, scor
ing='accuracy', cv=10)
print scores
print scores.mean()
print predicting_with_test
```

```
[ 0.91925728  0.91925728  0.91925728  0.91925728  0.91925728  0.91
928718
  0.91928455  0.91928455  0.91928455  0.91928455]
0.919271180943
[0 0 0 ..., 0 0 0]
```

In [116]:
```
# evaluate the model using 10-fold cross-validation
from sklearn.cross_validation import cross_val_score
predicting_with_test_imputed = model_lr.predict(transformed_test_X)

scores = cross_val_score(LogisticRegression(penalty = 'l2', C=0.001
),
                         transformed_X, y, scoring='accuracy', cv=1
0)
print scores
print scores.mean()
print predicting_with_test_imputed
```

```
[ 0.91925728  0.91919225  0.91925728  0.91922477  0.91925728  0.91
928718
  0.91928455  0.91928455  0.91925203  0.91928455]
0.919258173447
[0 0 0 ..., 0 0 0]
```

In [117]:
```
from sklearn.metrics import roc_auc_score
from sklearn.metrics import roc_curve
roc_auc = roc_auc_score(y_test_lr, prob_for_checking[:, 1])
fpr, tpr, thresholds = roc_curve(y_test_lr, prob_for_checking[:, 1]
)
plt.figure()
plt.plot(fpr, tpr, label='Logistic regression ROC curve (area = %0.
2f)' % roc_auc)
plt.plot([0, 1], [0, 1],'r--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver operating characteristic')
plt.legend(loc="lower right")
plt.savefig('Logistic_roc')
plt.show()
```



We want to predict the probabilities of not paying a loan, so we use the model predict.proba method. This returns an m x 2 array where m is the number of observations. The first column is the probability of the target being 0 and the second column is the probability of the target being 1 (so for a single row, the two columns must sum to 1). We want the probability the loan is not repaid, so we will select the second column.

The following code makes the predictions and selects the correct column.

- Below model is trained with training and using the provided test set for prediction

In [123]:
```
logit_model = LogisticRegression(penalty='l2', C=0.0001)
# Train on the training data
logit_model.fit(scaled_train_withoutMissingValues, y)
```

Out[123]:
```
LogisticRegression(C=0.0001, class_weight=None, dual=False,
          fit_intercept=True, intercept_scaling=1, max_iter=100,
          multi_class='ovr', n_jobs=1, penalty='l2', random_state=
None,
          solver='liblinear', tol=0.0001, verbose=0, warm_start=Fa
lse)
```

In [124]:
```
# Make sure to select the second column only
logit_model = logit_model.predict_proba(scaled_test_withoutMissingV
alues)[:, 1]
```

In [125]:
```
print logit_model
```

```
[ 0.06618764  0.12526075  0.10853851 ...,  0.05510693  0.07030372
  0.0820307 ]
```

In [126]:
```
#my_submission = pd.DataFrame({'SK_ID_CURR': one_hot_encoded_app_te
st.SK_ID_CURR, 'TARGET': logit_model})
#my_submission.to_csv('homecredit.csv', index=False)
```

- Logistic regression with missing data imputation.

In [128]:
```
imputed_log_model = LogisticRegression(penalty='l2', C=0.0001)
imputed_log_model.fit(imputed_train, y)
```

Out[128]:
```
LogisticRegression(C=0.0001, class_weight=None, dual=False,
          fit_intercept=True, intercept_scaling=1, max_iter=100,
          multi_class='ovr', n_jobs=1, penalty='l2', random_state=
None,
          solver='liblinear', tol=0.0001, verbose=0, warm_start=Fa
lse)
```

In [129]:
```
imputed_log_model = imputed_log_model.predict_proba(imputed_test)[:
,1]
```

In [130]:
```
print imputed_log_model
```

```
[ 0.06696085  0.12826838  0.08436034 ...,  0.05764537  0.07173938
  0.08816771]
```

With this prediction on kaggle i had a score of 0.675

Decision Tree Model

Here is my second model.

- Using the data where i removed the NaN values

In [135]:
```
# import the module needed for decision tree from scikitlearn
from sklearn import tree

X_train_tree, X_test_tree, y_train_tree, y_test_tree = train_test_s
plit(
    scaled_train_withoutMissingValues, y, test_size=0.25, random_st
ate=42)

# instantiate a logistic regression model, and fit with X and y
decision_tree = tree.DecisionTreeClassifier(min_samples_split=50,
                                            criterion='entropy',
                                            max_leaf_nodes=3, rando
m_state=0)

# evaluate the model by splitting into train and test sets
decision_tree.fit(X_train_tree, y_train_tree)
print 'This accuracy seems good {} but need to check further for th
e prediction and on testing set'.format(
    decision_tree.score(X_train_tree, y_train_tree))
y_pred_tree = decision_tree.predict(X_test_tree)
print y_pred_tree
print decision_tree.score(X_test_tree, y_test_tree)
```

```
This accuracy seems good 0.919204970668 but need to check further
for the prediction and on testing set
[0 0 0 ..., 0 0 0]
0.919469809308
```

In [136]:
```
decision_tree = tree.DecisionTreeClassifier(min_samples_split=50,
                                            criterion='entropy',
                                            max_leaf_nodes=3, rando
m_state=0)
decision_tree = decision_tree.fit(scaled_train_withoutMissingValues
, y)
```

In [137]:
```
decisionTreePrediction = decision_tree.predict_proba(scaled_test_wi
thoutMissingValues)[:,1]
```

In [138]:
```
decisionTreePrediction
```

Out[138]:
```
array([ 0.06521654,  0.06521654,  0.06521654, ...,  0.06521654,
        0.05989243,  0.1205374 ])
```

- using data where the data has been imputed.

In [140]:
```
# import the module needed for decision tree from scikitlearn
from sklearn import tree

X_train_tree_imputed, X_test_tree_imputed, y_train_tree_imputed, y_
test_tree_imputed = train_test_split(
    scaled_train_withoutMissingValues, y, test_size=0.25, random_st
ate=42)

# instantiate a logistic regression model, and fit with X and y
decision_tree_imputed = tree.DecisionTreeClassifier(min_samples_spl
it=50,
                                                     criterion='entropy',
                                                     max_leaf_nodes=3, rando
m_state=0)

# evaluate the model by splitting into train and test sets
decision_tree_imputed.fit(X_train_tree, y_train_tree)
print 'This accuracy seems good {} but need to check further for th
e prediction and on testing set'.format(
    decision_tree_imputed.score(X_train_tree_imputed, y_train_tree_
imputed))
y_pred_tree_imputed = decision_tree_imputed.predict(X_test_tree_imp
uted)
print y_pred_tree_imputed
print decision_tree_imputed.score(X_test_tree_imputed, y_test_tree_
imputed)
```

```
This accuracy seems good 0.919204970668 but need to check further
for the prediction and on testing set
[0 0 0 ..., 0 0 0]
0.919469809308
```

In [144]:
```
# fit the trainning set and target to the model
imputed_decisionTreeModel = decision_tree_imputed.fit(transformed_X
, y)
# make prediction on testing set using probability
imputed_prediction = decision_tree_imputed.predict_proba(transforme
d_test_X)[:,1]
imputed_prediction
```

Out[144]:
```
array([ 0.12974312,  0.13929687,  0.04763728, ...,  0.12974312,
        0.04763728,  0.12974312])
```

Random Forest

Here is my third model.

In [147]:
```
from sklearn.ensemble import RandomForestClassifier

# split the data into train test split
X_train, X_test, y_train, y_test = train_test_split(scaled_train_wi
thoutMissingValues,
                                                            y,
                                                            test_size=0.25,
                                                            random_state=0)
# Make the random forest classifier
clf_random_forest = RandomForestClassifier(n_estimators = 100,
                                           random_state = 50,
                                           verbose = 1, n_jobs = -1)
# train model
clf_random_forest.fit(X_train, y_train)
print 'This is the score : ',clf_random_forest.score(X_train, y_tra
in)
y_predict = clf_random_forest.predict(X_test)
print y_predict
print clf_random_forest.score(X_test, y_test)
random_forest_into_training_set = clf_random_forest.fit(scaled_trai
n_withoutMissingValues, y)
random_forest_predict = clf_random_forest.predict_proba(scaled_test
_withoutMissingValues)[:,1]
print random_forest_predict
```

```
[Parallel(n_jobs=-1)]: Done  42 tasks      | elapsed:   13.5s
[Parallel(n_jobs=-1)]: Done 100 out of 100 | elapsed:   31.4s fini
shed
```

```
This is the score :
```

```
[Parallel(n_jobs=4)]: Done  42 tasks      | elapsed:    1.1s
[Parallel(n_jobs=4)]: Done 100 out of 100 | elapsed:    2.5s finis
hed
```

```
 0.999947969285
```

```
[Parallel(n_jobs=4)]: Done  42 tasks      | elapsed:    0.4s
[Parallel(n_jobs=4)]: Done 100 out of 100 | elapsed:    0.8s finis
hed
```

```
[0 0 0 ..., 0 0 0]
```

```
[Parallel(n_jobs=4)]: Done  42 tasks      | elapsed:    0.4s
[Parallel(n_jobs=4)]: Done 100 out of 100 | elapsed:    1.1s finis
hed
```

```
0.920770571555
```

```
[Parallel(n_jobs=-1)]: Done  42 tasks      | elapsed:   28.0s
[Parallel(n_jobs=-1)]: Done 100 out of 100 | elapsed:   57.5s fini
shed
[Parallel(n_jobs=4)]: Done  42 tasks      | elapsed:    0.4s
```

```
[ 0.16  0.1   0.11 ...,  0.07  0.11  0.2 ]
```

```
[Parallel(n_jobs=4)]: Done 100 out of 100 | elapsed:    0.9s finis
hed
```

Doing the same on the imputed data

In [149]:
```
from sklearn.ensemble import RandomForestClassifier

# split the data into train test split
X_train, X_test, y_train, y_test = train_test_split(transformed_X,
                                                    y,
                                                    test_size=0.25,
                                                    random_state=0)
# Make the random forest classifier
clf_random_forest = RandomForestClassifier(n_estimators = 100,
                                           random_state = 50,
                                           verbose = 1, n_jobs = -1)
# train model
clf_random_forest.fit(X_train, y_train)
print 'This is the score : ',clf_random_forest.score(X_train, y_tra
in)
y_predict = clf_random_forest.predict(X_test)
print y_predict
print clf_random_forest.score(X_test, y_test)
random_forest_into_training_set = clf_random_forest.fit(transformed
_X, y)
random_forest_predict = clf_random_forest.predict_proba(transformed
_test_X)[:,1]
print random_forest_predict
```

```
[Parallel(n_jobs=-1)]: Done  42 tasks      | elapsed:   49.1s
[Parallel(n_jobs=-1)]: Done 100 out of 100 | elapsed:  1.7min fini
shed
```

```
This is the score :
```

```
[Parallel(n_jobs=4)]: Done  42 tasks      | elapsed:    2.7s
[Parallel(n_jobs=4)]: Done 100 out of 100 | elapsed:    6.2s finis
hed
```

```
 0.999960976963
```

```
[Parallel(n_jobs=4)]: Done  42 tasks      | elapsed:    1.1s
[Parallel(n_jobs=4)]: Done 100 out of 100 | elapsed:    2.4s finis
hed
```

```
[0 0 0 ..., 0 0 0]
```

```
[Parallel(n_jobs=4)]: Done  42 tasks      | elapsed:    1.0s
[Parallel(n_jobs=4)]: Done 100 out of 100 | elapsed:    2.4s finis
hed
```

```
0.920822602045
```

```
[Parallel(n_jobs=-1)]: Done  42 tasks      | elapsed:   57.5s
[Parallel(n_jobs=-1)]: Done 100 out of 100 | elapsed:  2.0min fini
shed
[Parallel(n_jobs=4)]: Done  42 tasks      | elapsed:    0.4s
```

```
[ 0.1   0.15  0.04 ...,  0.07  0.08  0.16]
```

```
[Parallel(n_jobs=4)]: Done 100 out of 100 | elapsed:    0.9s finis
hed
```

In [150]:

```
# evaluate the model using 10-fold cross-validation
from sklearn.cross_validation import cross_val_score
cv_with_random_forest = model.predict(scaled_test_withoutMissingVal
ues)

scores = cross_val_score(RandomForestClassifier(n_estimators = 100,
                                       random_state = 50,
                                       verbose = 1, n_jobs = -1),
                         scaled_train_withoutMissingValues, y, scor
ing='accuracy', cv=10)
print scores
print scores.mean()
print cv_with_random_forest
```

```
[Parallel(n_jobs=-1)]: Done  42 tasks      | elapsed:   18.5s
[Parallel(n_jobs=-1)]: Done 100 out of 100 | elapsed:   44.7s fini
shed
[Parallel(n_jobs=4)]: Done  42 tasks      | elapsed:    0.2s
[Parallel(n_jobs=4)]: Done 100 out of 100 | elapsed:    0.5s finis
hed
[Parallel(n_jobs=-1)]: Done  42 tasks      | elapsed:   18.6s
[Parallel(n_jobs=-1)]: Done 100 out of 100 | elapsed:   42.5s fini
```

```
shed
[Parallel(n_jobs=4)]: Done  42 tasks      | elapsed:    0.2s
[Parallel(n_jobs=4)]: Done 100 out of 100 | elapsed:    0.5s finis
hed
[Parallel(n_jobs=-1)]: Done  42 tasks      | elapsed:   21.3s
[Parallel(n_jobs=-1)]: Done 100 out of 100 | elapsed:   49.4s fini
shed
[Parallel(n_jobs=4)]: Done  42 tasks      | elapsed:    0.2s
[Parallel(n_jobs=4)]: Done 100 out of 100 | elapsed:    0.4s finis
hed
[Parallel(n_jobs=-1)]: Done  42 tasks      | elapsed:   21.4s
[Parallel(n_jobs=-1)]: Done 100 out of 100 | elapsed:   44.6s fini
shed
[Parallel(n_jobs=4)]: Done  42 tasks      | elapsed:    0.2s
[Parallel(n_jobs=4)]: Done 100 out of 100 | elapsed:    0.5s finis
hed
[Parallel(n_jobs=-1)]: Done  42 tasks      | elapsed:   18.0s
[Parallel(n_jobs=-1)]: Done 100 out of 100 | elapsed:   41.8s fini
shed
[Parallel(n_jobs=4)]: Done  42 tasks      | elapsed:    0.2s
[Parallel(n_jobs=4)]: Done 100 out of 100 | elapsed:    0.5s finis
hed
[Parallel(n_jobs=-1)]: Done  42 tasks      | elapsed:   20.4s
[Parallel(n_jobs=-1)]: Done 100 out of 100 | elapsed:   47.5s fini
shed
[Parallel(n_jobs=4)]: Done  42 tasks      | elapsed:    0.4s
[Parallel(n_jobs=4)]: Done 100 out of 100 | elapsed:    0.8s finis
hed
[Parallel(n_jobs=-1)]: Done  42 tasks      | elapsed:   20.5s
[Parallel(n_jobs=-1)]: Done 100 out of 100 | elapsed:   44.3s fini
shed
[Parallel(n_jobs=4)]: Done  42 tasks      | elapsed:    0.2s
[Parallel(n_jobs=4)]: Done 100 out of 100 | elapsed:    0.5s finis
hed
[Parallel(n_jobs=-1)]: Done  42 tasks      | elapsed:   17.8s
[Parallel(n_jobs=-1)]: Done 100 out of 100 | elapsed:   40.9s fini
shed
[Parallel(n_jobs=4)]: Done  42 tasks      | elapsed:    0.2s
[Parallel(n_jobs=4)]: Done 100 out of 100 | elapsed:    0.4s finis
hed
[Parallel(n_jobs=-1)]: Done  42 tasks      | elapsed:   18.9s
[Parallel(n_jobs=-1)]: Done 100 out of 100 | elapsed:   41.9s fini
shed
[Parallel(n_jobs=4)]: Done  42 tasks      | elapsed:    0.2s
[Parallel(n_jobs=4)]: Done 100 out of 100 | elapsed:    0.4s finis
hed
[Parallel(n_jobs=-1)]: Done  42 tasks      | elapsed:   20.4s
[Parallel(n_jobs=-1)]: Done 100 out of 100 | elapsed:   44.1s fini
shed
[Parallel(n_jobs=4)]: Done  42 tasks      | elapsed:    0.2s
```

```
[ 0.91925728  0.91925728  0.91925728  0.9192898   0.91925728  0.91
928718
  0.91928455  0.91928455  0.91928455  0.91925203]
0.919271180732
[0 0 0 ..., 0 0 0]

[Parallel(n_jobs=4)]: Done 100 out of 100 | elapsed:    0.5s finis
hed
```

Conclusion

- ## Reflection Honestly the whole project was quite tedious. There is a lot to know and there is a lot i do not know. The most tedious part of this project was the data mining part of the project and implementation of the algorithms since there are a lot of algorithms and little about them that is important to know.
  Also the visualization part. I think next time it is better to write a python file that i can just call when i want a certain visualization.