Introduction - Machine Learning in Azure

April 14, 2018

LendingClub

Lending Club is a peer-to-peer lending company to connect borrowers (people who need money) with investors (people who have money).
This notebook will demostrate following tasks:

- Build a machine learning model to predict whether or not borrower paid back their loan in full
- Operationalize the model by deploying it as a web service in Microsoft Azure platform
- Data can be downloaded in csv format from here https://www.lendingclub.com/info/download-data.action (https://www.lendingclub.com/info/download-data.action)
- This is a Python 2 notebook. Although Azure supports Python 3, its web services deployment does not support Python 3 coding yet

★ Azure Work Space Settings

Log in to Microsoft Azure Machine Learning Studio and, under SETTINGS, retrieve **WORKSPACE ID** and **AUTHORIZATION TOKEN**

In [1]:

```
# install libraries to local machine if needed
# pip install azure
# pip install azure-ml-api-sdk
# pip install azureml
```

In [68]:

```
import azureml

workspace_id = "enter workspace id from azure"
authorization_token = "enter authorization token from azure"
ws = azureml.Workspace(workspace_id=workspace_id, authorization_token=authorization_
```

★ Import Libraries and Dataset

In [69]:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
%matplotlib inline

# dataset was uploaded to Azure. import dataset from Azure
ds = ws.datasets["loan_data.csv"]

# convert dataset to pandas dataframe
df = ds.to_dataframe()
```

★ Explore Dataset and Perform Data Preprocessing

§ Check dataframe information

- "purpose" is a categorical feature
- "not.fully.paid" is the label/target variable

In [70]:

```
# check dataframe information
df.info()
```

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

§ View first five rows of data

In [71]:

```
df.head()
```

Out[71]:

	credit.policy	purpose	int.rate	installment	log.annual.inc	dti	fico	days.with.cr.lin
0	1	debt_consolidation	0.1189	829.10	11.350407	19.48	737	5639.95833
1	1	credit_card	0.1071	228.22	11.082143	14.29	707	2760.00000
2	1	debt_consolidation	0.1357	366.86	10.373491	11.63	682	4710.00000
3	1	debt_consolidation	0.1008	162.34	11.350407	8.10	712	2699.95833
4	1	credit_card	0.1426	102.92	11.299732	14.97	667	4066.00000

§ Initially columns use dot notation. We want to rename them to prevent future issues when creating web services

In [72]:

```
df.rename( columns=(lambda value: value.replace(".", "_")), inplace=True )
df.head()
```

Out[72]:

	credit_policy	purpose	int_rate	installment	log_annual_inc	dti	fico	days_with_cr
0	1	debt_consolidation	0.1189	829.10	11.350407	19.48	737	5639.95
1	1	credit_card	0.1071	228.22	11.082143	14.29	707	2760.00
2	1	debt_consolidation	0.1357	366.86	10.373491	11.63	682	4710.00
3	1	debt_consolidation	0.1008	162.34	11.350407	8.10	712	2699.95
4	1	credit_card	0.1426	102.92	11.299732	14.97	667	4066.00

§ Check if any feature has NULL value

In [73]:

```
# check null value
df.isnull().sum()
```

Out[73]:

```
credit_policy        0
purpose              0
int_rate             0
installment          0
log_annual_inc       0
dti                  0
fico                 0
days_with_cr_line    0
revol_bal            0
revol_util           0
inq_last_6mths       0
delinq_2yrs          0
pub_rec              0
not_fully_paid       0
dtype: int64
```

§ View distribution of target variable

In [74]:

```
df["not_fully_paid"].value_counts()
```

Out[74]:

```
0    8045
1    1533
Name: not_fully_paid, dtype: int64
```

§ Since "purpose" column is categorical, we want to transform it using one-hot-encoding, And remove one column to avoid multicollinearity.

In [75]:

```
# show 7 purpose values
print( "There are 7 purpose values:\n{}".format(df["purpose"].value_counts()) )

df_final = pd.get_dummies(data=df, columns=["purpose"], drop_first=True)
df_final.filter(regex=("purpose.*")).head() # show one less purpose value (6)
```

```
There are 7 purpose values:
debt_consolidation    3957
all_other             2331
credit_card           1262
home_improvement       629
small_business         619
major_purchase         437
educational            343
Name: purpose, dtype: int64
```

Out[75]:

	purpose_credit_card	purpose_debt_consolidation	purpose_educational	purpose_home_improver
0	0	1	0	
1	1	0	0	
2	0	1	0	
3	0	1	0	
4	1	0	0	

In [76]:

```
# confirm final dataset information
df_final.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 9578 entries, 0 to 9577
Data columns (total 19 columns):
credit_policy                 9578 non-null int64
int_rate                      9578 non-null float64
installment                   9578 non-null float64
log_annual_inc                9578 non-null float64
dti                           9578 non-null float64
fico                          9578 non-null int64
days_with_cr_line             9578 non-null float64
revol_bal                     9578 non-null int64
revol_util                    9578 non-null float64
inq_last_6mths                9578 non-null int64
delinq_2yrs                   9578 non-null int64
pub_rec                       9578 non-null int64
not_fully_paid                9578 non-null int64
purpose_credit_card           9578 non-null uint8
purpose_debt_consolidation    9578 non-null uint8
purpose_educational           9578 non-null uint8
purpose_home_improvement      9578 non-null uint8
purpose_major_purchase        9578 non-null uint8
purpose_small_business        9578 non-null uint8
dtypes: float64(6), int64(7), uint8(6)
memory usage: 1.0 MB
```

★ Split data into training and test set

There are syntax differences between Python 3 and Python 2 for calling sklearn. Use following to check current running Python version:

```
    import sys
    print(sys.executable)
    print(sys.version)
    print(sys.version_info)
```

In [78]:

```
import platform; print(platform.platform())
import sys; print("Python", sys.version)
import numpy; print("NumPy", numpy.__version__)
import scipy; print("SciPy", scipy.__version__)
```

```
Darwin-16.7.0-x86_64-i386-64bit
('Python', '2.7.14 |Anaconda, Inc.| (default, Mar 27 2018, 12:28:59) \
n[GCC 4.2.1 Compatible Clang 4.0.1 (tags/RELEASE_401/final)]')
('NumPy', '1.14.2')
('SciPy', '1.0.1')
```

In [79]:

```
# python 3 version
# from sklearn.model_selection import train_test_split

# python 2 version
from sklearn.cross_validation import train_test_split

y = df_final["not_fully_paid"]
X = df_final.drop("not_fully_paid", axis=1)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.33,
                                                     random_state=42)
```

★ Build a Decision Tree model

In [80]:

```
from sklearn.tree import DecisionTreeClassifier

dt = DecisionTreeClassifier()
dt.fit(X_train, y_train)
```

Out[80]:

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

§ Make predictions based on DT

In [81]:

```
# prediction based on decision tree
dt_prediction = dt.predict(X_test)
```

★ Build a Random Forest model

In [82]:

```
from sklearn.ensemble import RandomForestClassifier

rf = RandomForestClassifier()
rf.fit(X_train, y_train)
```

Out[82]:

```
RandomForestClassifier(bootstrap=True, class_weight=None, criterion='g
ini',
            max_depth=None, max_features='auto', max_leaf_nodes=None,
            min_impurity_decrease=0.0, min_impurity_split=None,
            min_samples_leaf=1, min_samples_split=2,
            min_weight_fraction_leaf=0.0, n_estimators=10, n_jobs=1,
            oob_score=False, random_state=None, verbose=0,
            warm_start=False)
```

§ Make predictions based on RF

In [83]:

```
rf_prediction = rf.predict(X_test)
```

★ Evaluate and compare model performance

Clearly Random Forest yields better result than single Decision Tree model

In [84]:

```
from sklearn.metrics import confusion_matrix, classification_report

print("*** From Decision Tree:")
print( confusion_matrix(y_test, dt_prediction) )
print( classification_report(y_test, dt_prediction) )

print("*** From Random Forest:")
print( confusion_matrix(y_test, rf_prediction) )
print( classification_report(y_test, rf_prediction) )
```

```
*** From Decision Tree:
[[2231  419]
 [ 399  112]]
             precision    recall  f1-score   support

          0       0.85      0.84      0.85      2650
          1       0.21      0.22      0.21       511

avg / total       0.75      0.74      0.74      3161

*** From Random Forest:
[[2603   47]
 [ 494   17]]
             precision    recall  f1-score   support

          0       0.84      0.98      0.91      2650
          1       0.27      0.03      0.06       511

avg / total       0.75      0.83      0.77      3161
```

§ Check and visualize model features importance

We'll choose Random Forest model

In [85]:

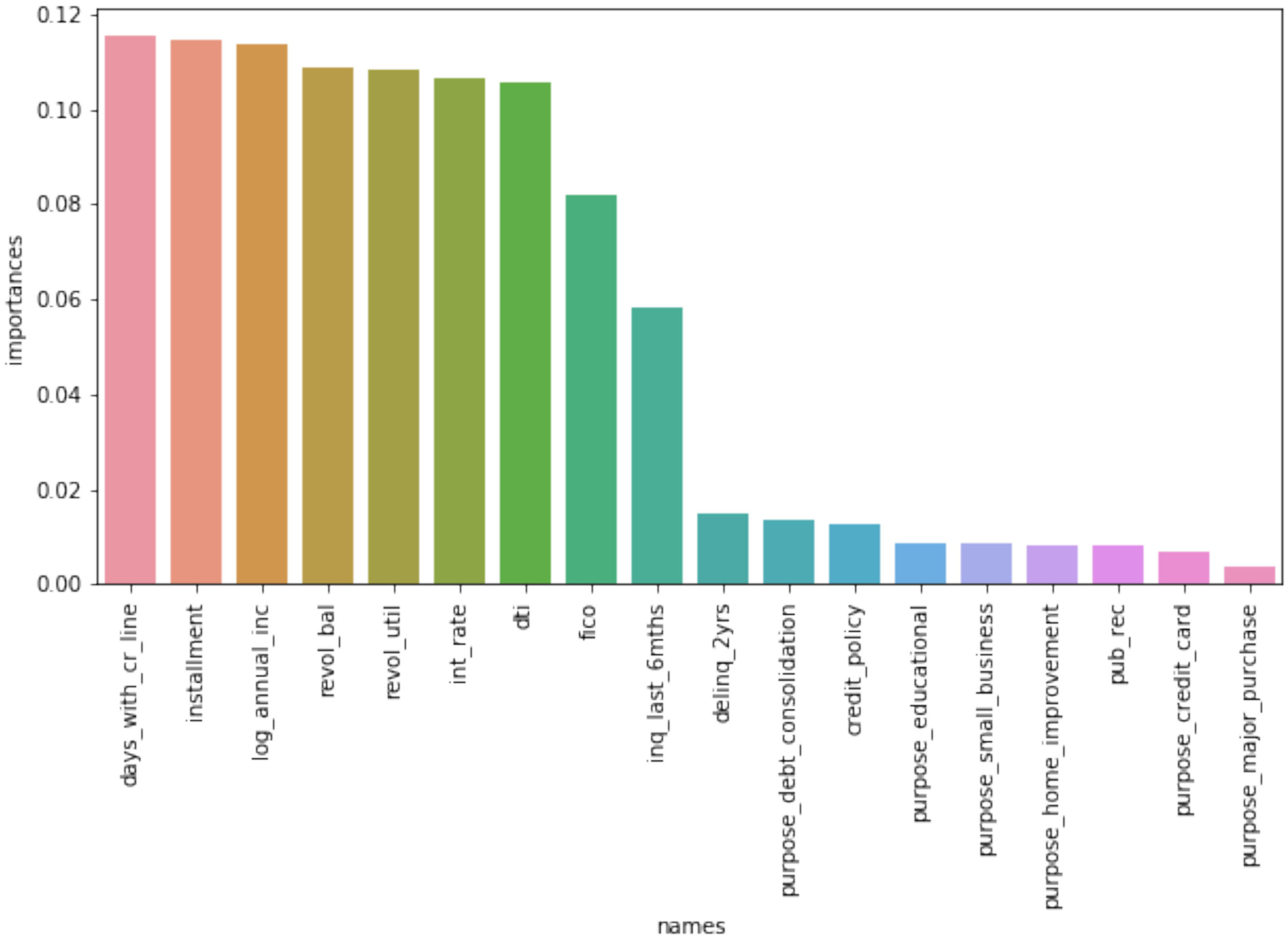
```
import seaborn as sns

# get feature importance
importances = rf.feature_importances_

# create a df containing column names and corresponding importances
list_feature = [pd.DataFrame(X.columns.values, columns=["names"]), pd.DataFrame(impo
df_feature = pd.concat(list_feature, axis=1)

# sort in descending order before plotting
df_feature.sort_values(by="importances", ascending=False, inplace=True)

# plotting
plt.figure(figsize=(10,5))
g = sns.barplot(data=df_feature, x="names", y="importances")
g.set_xticklabels(g.get_xticklabels(), rotation=90)
plt.show()
```



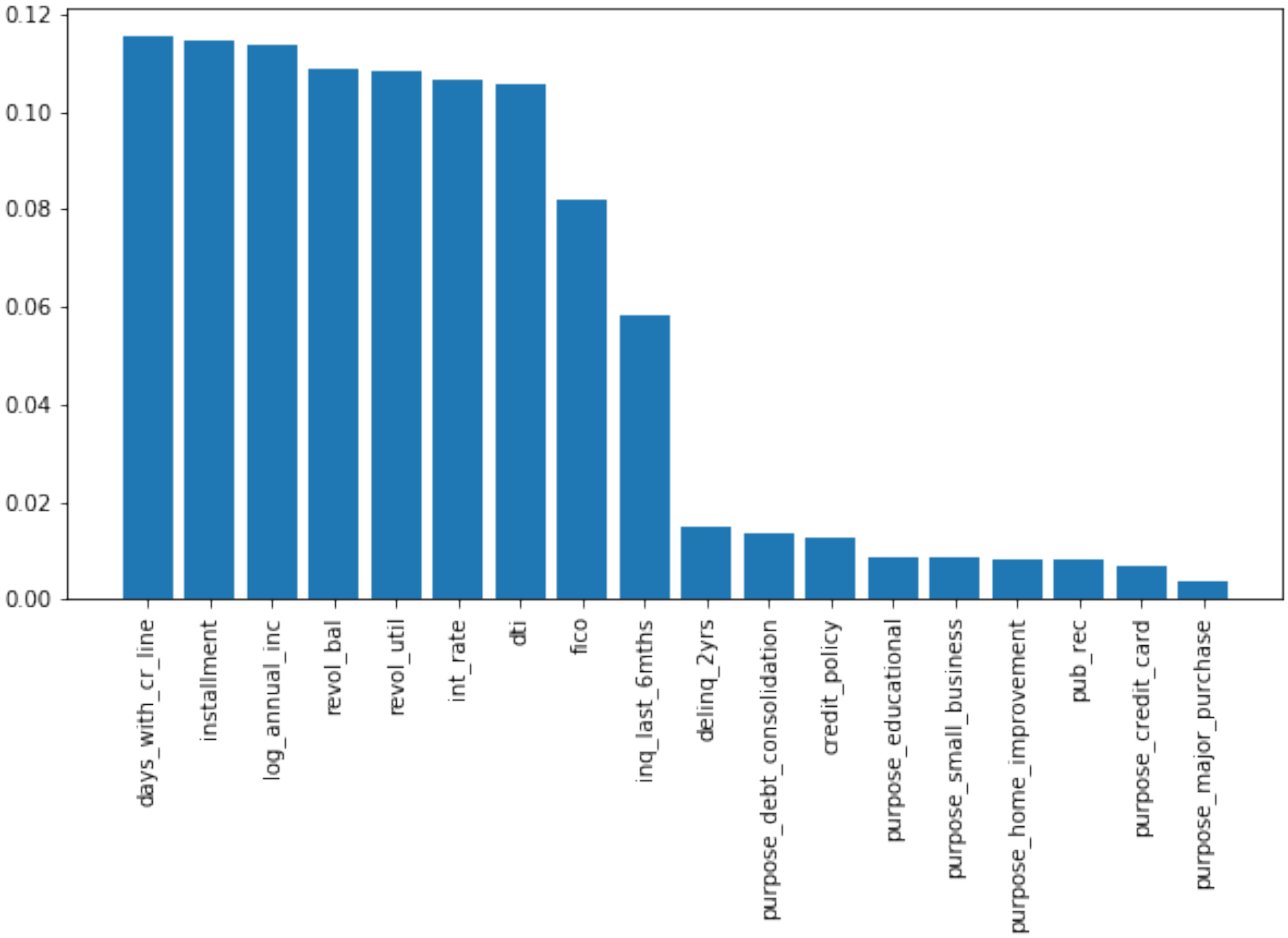
In [86]:

```
# get feature importance
importances = rf.feature_importances_

# sort feature importances in descending order
indices = np.argsort(importances)[::-1]

# use list comprehension to extract list of feature names which has same order of i
names = [X.columns[i] for i in indices]

# plot importnce in descending order
plt.figure(figsize=(10,5))
plt.bar(range(X.shape[1]), importances[indices])
plt.xticks(range(X.shape[1]), names, rotation=90)
plt.show()
```



★ Rebuild model using top 5 features (weighted by feature importances found earlier)

We'll use this simpler feature set to deploy web services

In [87]:

```
df_feature.iloc[:5,:]
```

Out[87]:

	names	importances
6	days_with_cr_line	0.115676
2	installment	0.114646
3	log_annual_inc	0.113702
7	revol_bal	0.108914
8	revol_util	0.108386

In [88]:

```
y_op = y

X_op = X[["days_with_cr_line", "log_annual_inc", "installment", "dti", "revol_util"
X_op_train, X_op_test, y_op_train, y_op_test = train_test_split(X_op, y_op, test_si

# build a Random Forest model
rf.fit(X_op_train, y_op_train)

# prediction
prediction_op = rf.predict(X_op_test)

# evaluate
print( confusion_matrix(y_op_test, prediction_op) )
print( classification_report(y_op_test, prediction_op) )
```

```
[[2603   47]
 [ 500   11]]
             precision    recall  f1-score   support

          0       0.84      0.98      0.90      2650
          1       0.19      0.02      0.04       511

avg / total       0.73      0.83      0.76      3161
```

§ Deploying the model as a web service

We create a wrapper function that takes input features as an argument. Then the function calls predict( ) method of our trained model and returns prediction result - classification as 0 (not fully paid) or 1 (fully paid)

§ Template for building web services

```
    from azureml import services
    @services.publish(..)
    @services.types(..)
    @services.returns(..)
    def user_defined_function(..)
```

In [89]:

```
X_op.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 9578 entries, 0 to 9577
Data columns (total 5 columns):
days_with_cr_line    9578 non-null float64
log_annual_inc       9578 non-null float64
installment          9578 non-null float64
dti                  9578 non-null float64
revol_util           9578 non-null float64
dtypes: float64(5)
memory usage: 374.2 KB
```

In [90]:

```
from azureml import services

# workspace_id and authorization_token are defined in the beginning of notebook
@services.publish(workspace_id, authorization_token)
# top 5 features with corresponding types
@services.types(days_with_cr_line=float, log_annual_inc=float, installment=float, dt
# predicted label
@services.returns(int)

# actural function called by web services
def credit_check(days_with_cr_line, log_annual_inc, installment, dti, revol_util):
    return rf.predict([days_with_cr_line, log_annual_inc, installment, dti, revol_ut

# for testing run within the notebook
service_url = credit_check.service.url
api_key = credit_check.service.api_key
help_url = credit_check.service.help_url
service_id = credit_check.service.service_id
```

★ Example for consuming web services from the notebook

In [91]:

```
# samples
# df[df["not_fully_paid"]==0][["days_with_cr_line","log_annual_inc","installment","c
# df[df["not_fully_paid"]==1][["days_with_cr_line","log_annual_inc","installment","c

# test output
result = credit_check.service(2760.000000, 11.082143, 228.22, 14.29, 76.7)
# result = credit_check.service(4209.95, 11.884489, 678.08, 10.15, 74.1)

# print customized result
transform_result = lambda x: "NO" if x=="0" else "YES"
print( "fully paid? {}".format(transform_result(result)) )

# alternative approach to print result
# print( lambda x: "NO" if x==0 else "YES" )(result)
```

```
fully paid? YES
```

★ Example for consuming web services from an application

In [94]:

```python
import urllib2
import json

example_values = [10.71, 4, 3180.041667, 76.8, 194.02]
# example_values = [4209.95, 11.884489, 678.08, 10.15, 74.1]

data = {"Inputs": {"input1": { "ColumnNames": ["days_with_cr_line","log_annual_inc"
                               "Values": [example_values] } }, # specified feature
        "GlobalParameters": {} }

body = json.dumps(data)
headers = {'Content-Type':'application/json', 'Authorization':('Bearer '+ api_key)}
req = urllib2.Request(service_url, body, headers)

try:
    response = urllib2.urlopen(req)
    result = json.loads(response.read())  # load json-formatted string response as
    text = result['Results']['output1']['value']['Values'][0][0] # convert numeric
    transform_result = lambda x: "NO" if x=="0" else "YES"
    print("fully paid? {}={}".format(text,transform_result(text))) # get the returne
#     print(result['Results']['output1']['value']['Values'][0][0]) # get the returne
except urllib2.HTTPError, error:
    print("The request failed with status code: " + str(error.code))
    print(error.info())
    print(json.loads(error.read()))
```

```
fully paid? 0=NO
```

In [ ]:

```python
```