# Predicting Kickstart Project State

March 11, 2018

DePaul University

CSC 478 Final Project

Tsung, Rebecca

Fox, Sidney

Introduction

The purpose of this document is to outline our approach in predicting the status of different projects presented to the public using the Kickstarter platform. The group leveraged many algorithms and functions available in the sklearn library to operationalize the defined machine learning workflow. This document will provide an overview of the dataset leveraged to complete the machine learning project, as well as an in-depth analysis of the methodologies implemented, the requisite results, and any conclusions, as well as constraints or blockers, the group was able to identify while completing the project.

# Overview

The dataset used to complete the group's analysis can be found on the Kaggle website as a competition dataset.[[1]](#footnote-5112) There are two datasets listed under the competition page. After a comparison of both datasets, it was determined that both datasets contain the same information in terms of relevant data points. The most recent dataset, entitled "ks-projects-201801.csv", contains a "cleansed" dataset, where all of the misnomers and formatting issues have been removed and / or rectified. After conducting this analysis, the group determined that the only dataset required to complete the project was the ks-projects-201801.csv. All of the analysis presented throughout this document solely references this dataset. After settling on the ks-projects-201801.csv dataset, the group created a GitHub repository and uploaded the .csv file in question to said repository. The group used GitHub to build their analysis leveraging the same base dataset, and complete the necessary objectives independently, within a central repository.

Concerning an overview of the features of the dataset specifically, the table below contains relevant high-level statistics in terms of row count and number of columns below:

|  |  |
| --- | --- |
| Statistic | **Value** |
| Number of Rows | 378,661 |
| Number of Columns | 15 |

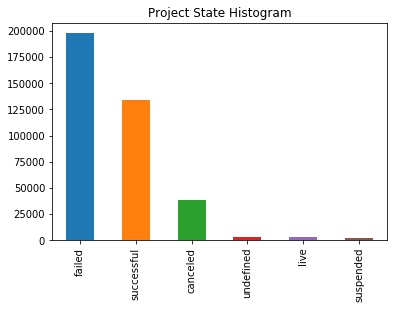
In addition to the number of rows and columns, the feature names and corresponding data types can be found below:

|  |  |  |
| --- | --- | --- |
| **Sequence Number** | **Column Name** | **Data Type** |
| 1 | ID | Numeric |
| 2 | name | String |
| 3 | category | String |
| 4 | main\_category | String |
| 5 | currency | String |
| 6 | deadline | DateTime |
| 7 | goal | Numeric |
| 8 | launched | DateTime |
| 9 | pledged | Numeric |
| 10 | state | String |
| 11 | backers | Numeric |
| 12 | country | String |
| 13 | usd pledged | Numeric |
| 14 | usd\_pledged\_real | Numeric |
| 15 | usd\_goal\_real | Numeric |

Once the dataset was identified, the group conducted a deeper dive of the data – analyzing features as well as observations.

# Analysis of Features and Observations

To gain better insight into the data, the group produced a series of histograms to show distributions by different segments of the data. Please find a histogram, as well as a summary table, of project status below:



|  |  |
| --- | --- |
| **Project State** | **Observation Count** |
| failed | 197,719 |
| successful | 133,956 |
| canceled | 38,779 |
| undefined | 3,562 |
| live | 2,799 |
| suspended | 1,846 |

From the information presented in this histogram and corresponding summary table, one can easily determine that most projects fail, with projects succeeding being the second most frequent project state. Aside from projects succeeding, applicable statuses include '"canceled", "undefined", "live", and "suspended". The group decided to keep all statuses aside from undefined, due to the ambiguity associated with the undefined project state. Upon removal, a distribution of the project states used for analysis can be found below:

Removing the undefined status reduced the total dataset record count to 375,099. Aside from removing observations where the project state was equal to undefined, the group also removed features "goal", "pledged", and "usd pledged". The dataset contains Kickstarter campaigns from a number of different countries, with the stated goal and pledged values in the native currency. The creator of the dataset converted the goal and pledged amounts to United States Dollars (usd). This conversion is represented in the dataset as "usd\_pledged\_real" and "usd\_goal\_real". Removal of the aforemtioned columns reduces the number of numeric variables to three – usd\_pledged\_real, usd\_goal\_real, and backers. Summary statistics for these variables can be found below:

|  |  |  |  |
| --- | --- | --- | --- |
| **Statistic** | backers | usd\_goal\_real | usd\_pledged\_real |
| **count** | 375099.000000 | 3.750990e+05 | 3.750990e+05 |
| **mean** | 106.620436 | 9.123935e+03 | 4.584708e+04 |
| **std** | 911.423593 | 9.140142e+04 | 1.158404e+06 |
| **min** | 0.000000 | 0.000000e+00 | 1.000000e-02 |
| **25%** | 2.000000 | 3.100000e+01 | 2.000000e+03 |
| **50%** | 12.000000 | 6.250000e+02 | 5.500000e+03 |
| **75%** | 57.000000 | 4.050180e+03 | 1.600000e+04 |

After producing high level summary statics for the numeric variables, the group produced a correlation matrix to view any weak or strong relationships within the data. A heatmap was also produced to present the relationships graphically. Please find the correlation matrix and corresponding heatmap below:

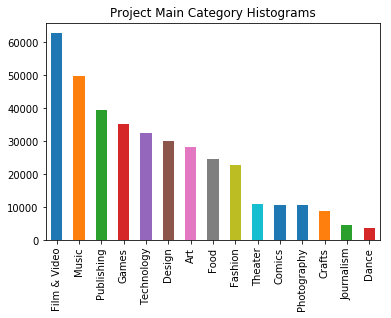
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | ID | backers | usd\_pledged\_real | usd\_goal\_real |
| **ID** | 1.000000 | 0.000676 | -0.000016 | 0.001868 |
| **backers** | 0.000676 | 1.000000 | 0.752528 | 0.004478 |
| **usd\_pledged\_real** | -0.000016 | 0.752528 | 1.000000 | 0.005571 |
| **usd\_goal\_real** | 0.001868 | 0.004478 | 0.005571 | 1.000000 |



The elicited results show strong relationships between all three of the numeric variables, with the relationship between usd\_pledged\_real and backers particularly strong with a value of 0.75. Based on the operational understanding of the data, this is to be expected. The number of backers positively drive the amount pledged for a given project.

Once correlations were identified, the group focused their analysis on the relationship between main\_category and category. The main\_category feature is a parent of the category feature, with 170 categories available and fifteen distinct main categories. A breakdown of observations by main\_category can be found below:

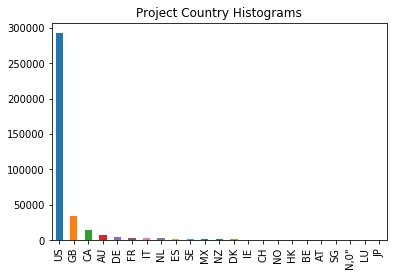
|  |  |
| --- | --- |
| Main Category | Observation Count |
| Art | 28,153 |
| Comics | 10,819 |
| Crafts | 8,809 |
| Dance | 3,767 |
| Design | 30,068 |
| Fashion | 22,813 |
| Film & Video | 62,731 |
| Food | 24,602 |
| Games | 35,230 |
| Journalism | 4,755 |
| Music | 49,684 |
| Photography | 10,778 |
| Publishing | 39,412 |
| Technology | 32,566 |
| Theater | 10,912 |



Based on this analysis, the top three categories in terms of observation count are "Film & Video", "Music", and "Publishing".

In addition to reviewing project status and category, the project group also reviewed the country of origin for each Kickstarter project. Please find a breakdown of projects by country, as well as a corresponding histogram, below:

|  |  |
| --- | --- |
| Country | Observation Count |
| AT | 597 |
| AU | 7,839 |
| BE | 617 |
| CA | 14,756 |
| CH | 761 |
| DE | 4,171 |
| DK | 1,113 |
| ES | 2,276 |
| FR | 2,939 |
| GB | 33,672 |
| HK | 618 |
| IE | 811 |
| IT | 2,878 |
| JP | 40 |
| LU | 62 |
| MX | 1,752 |
| N,0" | 235 |
| NL | 2,868 |
| NO | 708 |
| NZ | 1,447 |
| SE | 1,757 |
| SG | 555 |
| US | 292,627 |



The three most frequent countries in the Kickstarter dataset are the United States (US), Great Britain (GB), and Canada (CA). Completing this analysis brought to the group's attention a misnomer in the data – 235 observations with a country value of "N,0"", which is not a valid country code. These observations were removed from the dataset.

# Transformations

In addition to the analysis completed on the categorical and numeric features found within the Kickstarter dataset, two transformations were completed before algorithm application. The first transformation addressed Kickstarter campaigns without names, or titles. This issue impacted four records, and the group converted the name to the value of "Unknown". The second transformation dealt with invalid data types for the dataset. The Pandas library applied the wrong data type to a few of the features. The impacted features included "deadline", "launched", "usd\_goal\_real", and "usd\_pledged\_real". The "deadline" and "launched" features were converted to dates, removing time from the observations. The "usd\_goal\_real" and "usd\_pledged\_real" features were converted to integer data types. Transformations were applied in an attempt to align the working dataset with the data types presented on the project's Kaggle site.

Appendix

# coding: utf-8

# # Kickstarter Projects

# ### CSC 478 Final Project

# #### Synopsis:

# \* The purpose of this project is to predict whether a kickstarter campaign will fail, succeed, or cancel based on the available information available [here](https://raw.githubusercontent.com/stfox13/CSC478FinalProject/master/Data/ks-projects-201801.csv).

# \* We will use an array of machine learning algorithms, including KNN, Linear Regression, Logistic Regression, and / or SVM to find the most accurate model.

#

# #### Contributors:

# \* [Rebecca Tung (1448196)](https://github.com/rtungus)

# \* [Sidney Fox (1524992)](https://github.com/stfox13)

#

# ## Libraries used through the project:

get\_ipython().magic(u'matplotlib inline')

import matplotlib.pyplot as plt

import numpy as np

import pandas as pd

import seaborn as sns

import os

import math

import requests

import datetime as dt

import matplotlib as mpl

import io

from pandas import Series, DataFrame

from sklearn.pipeline import Pipeline

from sklearn.neighbors import KNeighborsClassifier

from sklearn.linear\_model import LogisticRegression, LinearRegression

from sklearn.ensemble import ExtraTreesClassifier

from sklearn import preprocessing

from sklearn.preprocessing import StandardScaler, Imputer, LabelEncoder

from sklearn import svm

from sklearn.metrics import f1\_score, confusion\_matrix, accuracy\_score, classification\_report

import itertools

from sklearn.feature\_selection import RFE

from collections import defaultdict

import pylab

#Environment settings

from IPython.core.interactiveshell import InteractiveShell

InteractiveShell.ast\_node\_interactivity = "all"

mpl.rcParams['figure.figsize'] = (10,10)

np.set\_printoptions(suppress=True)

# ## Load raw data as Pandas DataFrame:

url = '<https://raw.githubusercontent.com/stfox13/CSC478FinalProject/master/Data/ks-projects-201801.csv>'

kickproj\_org= pd.read\_csv(url)

print "Total rows in dataset:",len(kickproj\_org)

# ## Define Useful Functions

#Roundup function

def roundup(x, y):

return int(math.ceil(x / y) \* y)

#Define a fuction to print and plot confusin matrix

def plot\_confusion\_matrix(cm, classes,

normalize=False,

title='Confusion matrix',

cmap=plt.cm.Blues):

"""

This function prints and plots the confusion matrix.

Normalization can be applied by setting `normalize=True`.

"""

if normalize:

cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]

else:

pass

plt.imshow(cm, interpolation='nearest', cmap=cmap)

plt.title(title)

plt.colorbar()

tick\_marks = np.arange(len(classes))

plt.xticks(tick\_marks, classes, rotation=45)

plt.yticks(tick\_marks, classes)

fmt = '.2f' if normalize else 'd'

thresh = cm.max() / 2.

for i, j in itertools.product(range(cm.shape[0]), range(cm.shape[1])):

plt.text(j, i, format(cm[i, j], fmt),

horizontalalignment="center",

color="white" if cm[i, j] > thresh else "black")

plt.tight\_layout()

plt.ylabel('True label')

plt.xlabel('Predicted label')

#Define a fuction to calculate and print TP, TN, FP, and FN for each category

def show\_statistics(test\_y, y\_pred, matrix):

TP = np.diag(matrix)

FP = np.sum(matrix, axis=0) - TP

FN = np.sum(matrix, axis=1) - TP

TN = []

for i in range(len(matrix)):

temp = np.delete(matrix, i, 0) # delete ith row

temp = np.delete(temp, i, 1) # delete ith column

TN.append(sum(sum(temp)))

temp\_dic = {'TP': TP, 'FP' : FP,

'TN' : TN, 'FN' : FN}

scoreMatrix = DataFrame.from\_dict(temp\_dic)

return scoreMatrix

# Define a fuction to print F1 Score for each class and global (micro)

def formatResult(preResult, columnNames):

resultDF = DataFrame(preResult.values(), columns=columnNames, index=preResult.keys())

resultDF.loc['sum'] = np.sum(preResult.values(), axis=0)

resultDF['Sum of Class F1'] = np.append(np.sum(preResult.values(), axis=1), np.NaN)

return resultDF

#Define a function to run KNeighborsClassifier with different n\_neighbors and store f1 score

def runKNN(trainX, trainY, testX, testY, number, f1\_only = False, trainSetName = '', dic\_result\_knn = {}):

i = 3

cls = KNeighborsClassifier(n\_neighbors=i)

while i <= number:

cls = KNeighborsClassifier(n\_neighbors=i)

cls.fit(trainX, trainY)

predY = cls.predict(testX)

result = f1\_score(testY, predY, average=None).round(2)

result = np.append(result, f1\_score(testY, predY, average='micro').round(2))

dic\_result\_knn['N=' + str(i) +'-' + trainSetName] = result

i = i + 2

return dic\_result\_knn

# Define a function to run LogisticRegression with different class\_weight settings and store f1 score

def runLogistic(trainX, trainY, testX, testY, f1\_only = False, trainSetName = '', dic\_result\_log = {}):

cls = LogisticRegression()

cls.fit(trainX, trainY)

predY = cls.predict(testX)

result = f1\_score(testY, predY, average=None).round(2)

result = np.append(result, f1\_score(testY, predY, average='micro').round(2))

dic\_result\_log['CWeight = None - ' + trainSetName] = result

cls = LogisticRegression(class\_weight='balanced')

cls.fit(trainX, trainY)

predY = cls.predict(testX)

result = f1\_score(testY, predY, average=None).round(2)

result = np.append(result, f1\_score(testY, predY, average='micro').round(2))

dic\_result\_log['CWeight = balanced - ' + trainSetName] = result

return dic\_result\_log

# Define a function to run SVM with different kernel settings and store f1 score

def runSVM(trainX, trainY, testX, testY, f1\_only = False, trainSetName = '', dic\_result\_log = {}):

C = 1.0 # SVM regularization parameter

svc = svm.SVC(kernel='linear', C=C, decision\_function\_shape='ovr').fit(trainX, trainY)

predY = svc.predict(X\_plot)

result = f1\_score(testY, predY, average=None).round(2)

result = np.append(result, f1\_score(testY, predY, average='micro').round(2))

print results

dic\_result\_log['SVCKernel = linear - ' + trainSetName] = result

svc = svm.SVC(kernel='rbf', C=C, decision\_function\_shape='ovr').fit(trainX, trainY)

predY = svc.predict(X\_plot)

result = f1\_score(testY, predY, average=None).round(2)

result = np.append(result, f1\_score(testY, predY, average='micro').round(2))

print results

dic\_result\_log['SVCKernel = rbf - ' + trainSetName] = result

return dic\_result\_log

# ## Check the Y data:

#Plot histogram

kickproj\_org['state'].value\_counts()

kickproj\_org['state'].value\_counts().plot(kind='bar', title='Project State Histogram')

# ### Drop projects when the state is equal to "undefined":

# Remove state = 'undefined'

kickproj = kickproj\_org[kickproj\_org['state'] != 'undefined']

len(kickproj)

#Plot histogram

kickproj['state'].value\_counts().plot(kind='bar', title='Project State Histogram (without undefined)')

#Review first five records:

kickproj.head(5)

# #### Since we have the goal and pledge amounts converted to US dollars (usd), we will drop the original goal and pledged columns:

kickproj = kickproj.drop(['goal','pledged','usd pledged'], axis=1)

len(kickproj)

kickproj.head(5)

# ## Check the X data:

#High level descriptive stats and correlation:

kickproj.describe()

kickproj.corr()

print('Heat Map of Correlation Coefficients:')

sns.heatmap(kickproj.corr(), cmap="BuGn\_r", linewidths=0.5, annot=True)

#Review different categories in dataset:

categoryDF = kickproj.groupby(['category']).size().reset\_index(name='counts')

len(categoryDF)

categoryDF.sort(['counts']).head(5)

#Review main categories in dataset:

kickproj.groupby(['main\_category']).size().reset\_index(name='counts')

kickproj['main\_category'].value\_counts().plot(kind='bar', title='Project Main Category Histograms')

#Review relationship of categories to main\_categories (parent-child)

cateDF = kickproj.groupby(['main\_category', 'category']).size().reset\_index(name='counts')

len(cateDF)

cateDF.head(40)

#Distribution by country:

kickproj.groupby(['country']).size().reset\_index(name='counts')

kickproj['country'].value\_counts().plot(kind='bar', title='Project Country Histograms')

# ### Remove country with invalid value, N,0"

kickproj = kickproj[kickproj['country'] != 'N,0"']

kickproj.groupby(['country']).size().reset\_index(name='counts')

kickproj['country'].value\_counts().plot(kind='bar', title='Project Country Histograms')

# ### Check null value

null\_columns=kickproj.columns[kickproj.isnull().any()]

null\_columns

kickproj[null\_columns].isnull().sum()

kickproj[kickproj["name"].isnull()][null\_columns]

# ### Replace nan with Unknown for name

kickproj["name"].fillna('Unknown', inplace=True)

null\_columns=kickproj.columns[kickproj.isnull().any()]

null\_columns

# ### Apply correct data types to DataFrame:

print 'Data types do not align with the data types defined in the data dictionary:\n\n', kickproj.dtypes

# Columns that are of date data type:

datecols = ['deadline','launched']

# Columns that are of int data type:

intcols = ['usd\_pledged\_real','usd\_goal\_real']

# Convert date columns to date data type:

for col in datecols:

kickproj[col] = pd.to\_datetime(kickproj[col]).dt.date

kickproj[col] = pd.to\_datetime(kickproj[col])

# Convert int columns to int data type:

kickproj[intcols] = kickproj[intcols].fillna(0).astype(np.int64)

print 'Review converted data types:\n\n', kickproj.dtypes

#Review dataset after conversion:

kickproj.head(5)

# ### Model Classification Using All Features (Start)

#Encode non-numeric variables - needed to run most of the models, understand anything feature importance:

le = preprocessing.LabelEncoder

d = defaultdict(le)

le\_df = kickproj.apply(lambda x: d[x.name].fit\_transform(x))

#Split data into two subsets - features and target - looking to predict state of the project:

features = le\_df[le\_df.columns.drop('state')]

target = le\_df['state']

# #### Feature Selection Method One: Recursive Feature Elimination

#Look at recursive feature elimination (RFE) with logistic regression and select three features:

LogReg\_RFE = RFE(LogisticRegression(), 3).fit(features, target)

print('The three most important features according to Logistic Regression:\n'),(np.array(features.columns)[LogReg\_RFE.support\_])

#Look at recursive feature elimination (RFE) with linear regression and select three features:

LinReg\_RFE = RFE(LinearRegression(), 3).fit(features, target)

print('The three most important features according to Linear Regression:\n'),(np.array(features.columns)[LinReg\_RFE.support\_])

#We'll use extra trees classifier to calculate feature importance:

ETC = ExtraTreesClassifier().fit(features, target)

feat\_imp\_df = pd.DataFrame({'Columns':pd.Series(features.columns)})

feat\_imp\_df['Feature Importance'] = pd.Series(ETC.feature\_importances\_)

feat\_imp\_df.set\_index(['Columns'],inplace=True)

# #### Feature Selection Method Two: Feature Importance

print('Column Names and Associated Feature Importance:')

feat\_imp\_df

feat\_imp\_df.plot(kind="bar", title="Feature Importance Results", legend = False)

# Observation: according to the extra trees classifier, the most important features are:<br>usd\_pledged\_real, usd\_goal\_real, backers<br><br>Let's split the data into test and train and compare approaches.

# ### Model Execution

#split the dataset in two parts [:n] for training and [n:] for testing (n = 50%)

le\_len = int(math.floor(len(le\_df)/2))

le\_train, le\_test = le\_df.iloc[:le\_len], le\_df.iloc[le\_len:]

#Create x and y train and test data sets, where x contains features and y contains target

x\_train, y\_train = le\_train[le\_train.columns.drop('state')], le\_train['state']

x\_test, y\_test = le\_test[le\_test.columns.drop('state')],le\_test['state']

# #### Run battery of algorithms: KNN, Logistic Regression, SVM

#Initialize le\_result\_Dic:

le\_result\_Dic = {}

#Algorithm 1: KNN

le\_result\_Dic = runKNN(x\_train, y\_train.values.ravel(), x\_test, y\_test.values.ravel(), 18, trainSetName = 'LE KNN', dic\_result\_knn = le\_result\_Dic)

#Algorithm 2: Logistic Regression

le\_result\_Dic = runLogistic(x\_train, y\_train.values.ravel(), x\_test, y\_test.values.ravel(), trainSetName = 'LE LR', dic\_result\_log = le\_result\_Dic)

#Algorithm 3: (Attempt) SVM

most\_important\_cols = ['backers','usd\_pledged\_real','usd\_goal\_real']

svm\_x\_train, svm\_y\_train = le\_train[most\_important\_cols], le\_train['state']

svm\_x\_test, svm\_y\_test = le\_test[most\_important\_cols], le\_test['state']

svm.SVC(kernel='linear', C=1.0, decision\_function\_shape='ovr').fit(svm\_x\_train, svm\_y\_train)

#View results:

le\_result\_Dic

# ### Find out correlation among all variables:

kickproj.apply(lambda x : pd.factorize(x)[0]).corr(method='pearson', min\_periods=1)

print('Heat Map of Correlation Coefficients:')

sns.heatmap(kickproj.apply(lambda x : pd.factorize(x)[0]).corr(method='pearson', min\_periods=1), cmap="BuGn\_r", linewidths=0.5, annot=True)

#Define binrange

binrange = range(1, roundup(max(kickproj['usd\_pledged\_real']),100000), 5000000)

binrange

#Review minimum and maximum values for usd\_goal\_real

min(kickproj['usd\_goal\_real'])

max(kickproj['usd\_goal\_real'])

# All successful records have usd\_pledged\_real >= 0

min(kickproj['usd\_pledged\_real'])

max(kickproj['usd\_pledged\_real'])

kickproj[(kickproj['usd\_pledged\_real']==0) & (kickproj['state']=='successful')]

#Shuffle dataset

shffled\_kickproj = kickproj.sample(frac=1)

shffled\_kickproj['state\_cd'] = shffled\_kickproj['state'].apply(lambda a: True if a == 'successful' else False)

shffled\_kickproj.head(5)

le = preprocessing.LabelEncoder()

le.fit(shffled\_kickproj['country'])

shffled\_kickproj['country\_cd'] = le.transform(shffled\_kickproj['country'])

shffled\_kickproj.head(5)

num = shffled\_kickproj.shape[0]/2

train\_x, train\_y = shffled\_kickproj.iloc[0:num, [5,6,8,9,10,11,13]], shffled\_kickproj.iloc[0:num, 12]

test\_x, test\_y = shffled\_kickproj.iloc[num:, [5,6,8,9,10,11,13]], shffled\_kickproj.iloc[num:, 12]

train\_x.head(2)

train\_x.shape

train\_y.head(2)

train\_y.shape

test\_x.head(2)

test\_x.shape

test\_y.head(2)

test\_y.shape

# ### Train training set

temp\_features\_train = train\_x['country\_cd'].reshape(-1, 1) # Needs to be the correct shape

temp\_features\_test = test\_x['country\_cd'].reshape(-1, 1) # Needs to be the correct shape

ohe = preprocessing.OneHotEncoder(sparse=False) #Easier to read

#fit on training set only

ohe.fit(temp\_features\_train)

countryDF\_train = DataFrame(ohe.transform(temp\_features\_train), columns = ohe.active\_features\_, index = train\_x.index)

countryDF\_test = DataFrame(ohe.transform(temp\_features\_test), columns = ohe.active\_features\_, index = test\_x.index)

countryDF\_train.head(10)

countryDF\_test.head(10)

train\_x.shape

countryDF\_train.shape

test\_x.shape

countryDF\_test.shape

train\_X1 = pd.merge(train\_x.drop(['country','country\_cd'], axis=1), countryDF\_train, left\_index=True, right\_index=True)

train\_X1.head(10)

train\_X1.shape

test\_X1 = pd.merge(test\_x.drop(['country','country\_cd'], axis=1), countryDF\_test, left\_index=True, right\_index=True)

test\_X1.head(10)

test\_X1.shape

result\_Dic = {}

#Call function to run KNeighborsClassifier with different n\_neighbors settings (up to 20) and store the f1 score results

result\_Dic = runKNN(train\_X1, train\_y.values.ravel(), test\_X1, test\_y.values.ravel(), 18, trainSetName = 'Basic&Country', dic\_result\_knn = result\_Dic)

#Call function to run LogisticRegression with different class\_weight settings (None or Balance) and store the f1 score results

result\_Dic = runLogistic(train\_X1, train\_y.values.ravel(), test\_X1, test\_y.values.ravel(), trainSetName = 'Basic&Country', dic\_result\_log = result\_Dic)

#Store number of classes

n\_classes = np.unique(shffled\_kickproj['state\_cd'])

n\_classes

resultDF = formatResult(result\_Dic, np.append(n\_classes, 'micro'))

resultDF.head(10)

# Print out the top 10 Micro (overall) F1 score from all settings

resultDF.drop('sum').nlargest(5, 'micro')

resultDF.drop('sum').nlargest(5, 'Sum of Class F1')

knnClr = KNeighborsClassifier(n\_neighbors=5)

knnClr.fit(train\_X1, train\_y.values.ravel())

final\_y\_pred = knnClr.predict(test\_X1)

# Compute confusion matrix

cnf\_matrix = confusion\_matrix(test\_y, final\_y\_pred)

np.set\_printoptions(precision=2)

# Plot non-normalized confusion matrix

plt.figure()

plot\_confusion\_matrix(cnf\_matrix, classes=[False,True],

normalize=False,

title='Confusion matrix, without normalization (True = successful; False = Other)')

plt.figure()

plot\_confusion\_matrix(cnf\_matrix, classes=[False,True],

normalize=True,

title='Confusion matrix, with normalization (True = successful; False = Other)')

stats = show\_statistics(test\_y, final\_y\_pred, cnf\_matrix)

strName = map((lambda a: 'successful' if a == True else 'Failed/Cancelled/Live/Suspended'), [False, True])

print "Classificaiton Report:"

print classification\_report(test\_y, final\_y\_pred, target\_names=strName, digits=2)

1. Kemical. "Kickstarter projects." Kaggle.com.

   https://www.kaggle.com/kemical/kickstarter-projects (accessed February 13, 2018). [↑](#footnote-ref-5112)