Python Portfolio

Herman D. Schaumburg

August 27, 2019

Abstract

Here are some examples of my Python codes. Included are Scikit-learn example: credit card fraud Pandas coding challenge, Python coding challenge, and Matplotlib script. I also include some problem statements and descriptions of how the codes and solutions work. I may add more to this and post them here. The python scripts with data files are also located within this directory. All files may be downloaded in the Python_Portfolio.zip file.

Scikit-learn example: credit card fraud

1.1 Origin and data set

For this example, I followed the tutorial found at the URL below and added some code to measure the accuracy of predictions made by three estimators (Naive Bayes, LinearSVC, and K-Neighbors Classifier).
https://www.dataquest.io/blog/sci-kit-learn-tutorial/

Based on the machine_learning_map found at the URL below, I should have tried SVC or Ensemble Classifiers instead of Naive Bayes.
https://scikit-learn.org/stable/tutorial/machine_learning_map/

I choose to use a credit card fraud data set from kaggle instead of the one in the tutorial. The columns of the transaction data set are Time (time elapsed from first transaction to current one), V1, V2, . . . , V28, Amount, and Class. The class indicates if the transaction was a fraud Class=1 or not Class=0. The V's are a result of applying Principal Component Analysis (PCA).
https://www.kaggle.com/mlg-ulb/creditcardfraud

The script below employs `train_test_split` to split the data into training and testing components. I used 70% of the data for training and 30% for testing. I first used all the data in its raw form for the task and then compared this to a few other strategies. Since the data set has 492 frauds out of 284,807 transactions, . The script I wrote follows:

1.2 code

```python
import pandas as pd
from sklearn.model_selection import train_test_split

import numpy as np

#Read data file
#https://www.kaggle.com/mlg-ulb/creditcardfraud/version/3
data = pd.read_csv('creditcard.csv')

#Seperate data from target
```

```python
#       Last column of data is currently the target (class = 1 for fraudulant transaction 0 for normal transaction)
col_labels=[]
for col_label in data.columns:
        col_labels.append(col_label)
#print(col_labels)
target_col_label=col_labels[len(col_labels)-1]
col_labels=col_labels[0:len(col_labels)-1]
target=data[target_col_label]
data=data[col_labels]
#print(data.head(n=40))
#print(target.head(n=40))

data_train, data_test, target_train, target_test = train_test_split(data,target, test_size = 0.30
, random_state = 10)


#Naive-Bayes Estimator
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import auc
from sklearn.metrics import accuracy_score
gnb = GaussianNB()
pred = gnb.fit(data_train, target_train).predict(data_test)

num_false_pos=0
num_false_neg=0
num_fraud_cases=0
caught_frauds=0
num_of_pred=len(pred)
for i in range(0,num_of_pred):
#        print(target_test[i])
        if pred[i]==0 and target_test.iloc[i]==1:
                num_false_neg+=1
        elif pred[i]==1 and target_test.iloc[i]==0:
                num_false_pos+=1
        if target_test.iloc[i]==1:
                num_fraud_cases+=1
                if pred[i]==1:
                        caught_frauds+=1
print("*** Naive-Bayes Estimater Results ***")
print("Length of target data  "+str(target_test.shape[0]))
print("Number of predictions made:  "+str(num_of_pred))
print("Number of frauds committed:  "+str(num_fraud_cases))
print("Number of frauds caught:  "+str(caught_frauds))
print("Number of false positives:  "+str(num_false_pos))
print("Number of false negatives:  "+str(num_false_neg))
print("Naive-Bayes accuracy : ",accuracy_score(target_test, pred, normalize = True))
#print("Naive-Bayes AUC : ",auc(target_test, pred))

#LinearSVC
from sklearn.svm import LinearSVC
#create an object of type LinearSVC
#Got convergence warning without having dual=False.  max_iter default is 1000
svc_model = LinearSVC(random_state=0, dual=False)
#train the algorithm on training data and predict using the testing data
pred = svc_model.fit(data_train, target_train).predict(data_test)

```

```python
num_false_pos=0
num_false_neg=0
num_fraud_cases=0
caught_frauds=0
num_of_pred=len(pred)
for i in range(0,num_of_pred):
#        print(target_test[i])
        if pred[i]==0 and target_test.iloc[i]==1:
                num_false_neg+=1
        elif pred[i]==1 and target_test.iloc[i]==0:
                num_false_pos+=1
        if target_test.iloc[i]==1:
                num_fraud_cases+=1
                if pred[i]==1:
                        caught_frauds+=1
print("*** LinearSVC Results ***")
print("Length of target data  "+str(target_test.shape[0]))
print("Number of predictions made:  "+str(num_of_pred))
print("Number of frauds committed:  "+str(num_fraud_cases))
print("Number of frauds caught:  "+str(caught_frauds))
print("Number of false positives:  "+str(num_false_pos))
print("Number of false negatives:  "+str(num_false_neg))
print("LinearSVC accuracy : ",accuracy_score(target_test, pred, normalize = True))


#K-Neighbors Classifier
from sklearn.neighbors import KNeighborsClassifier
#create object of the lassifier
neigh = KNeighborsClassifier(n_neighbors=3, p=1)
#Train the algorithm
neigh.fit(data_train, target_train)
# predict the response
pred = neigh.predict(data_test)
num_false_pos=0
num_false_neg=0
num_fraud_cases=0
caught_frauds=0
num_of_pred=len(pred)
for i in range(0,num_of_pred):
#        print(target_test[i])
        if pred[i]==0 and target_test.iloc[i]==1:
                num_false_neg+=1
        elif pred[i]==1 and target_test.iloc[i]==0:
                num_false_pos+=1
        if target_test.iloc[i]==1:
                num_fraud_cases+=1
                if pred[i]==1:
                        caught_frauds+=1
print("*** K-Neighbors Classifier Results ***")
print("Length of target data  "+str(target_test.shape[0]))
print("Number of predictions made:  "+str(num_of_pred))
print("Number of frauds committed:  "+str(num_fraud_cases))
print("Number of frauds caught:  "+str(caught_frauds))
print("Number of false positives:  "+str(num_false_pos))
print("Number of false negatives:  "+str(num_false_neg))
print("K-Neighbors Classifier accuracy : ",accuracy_score(target_test, pred, normalize = True))
```

1.3 Results

LinearSVC gave convergence warnings with max_iter=1000 and max_iter=1500. I settled on setting this option `dual=False` in LinearSVC, which I need to understand further.

The script above uses the columns Time, V1, V2, ..., V28, Amount for the training data. Running the script gave this result:

```
*** Naive-Bayes Estimater Results ***
Length of target data 85443
Number of predictions made: 85443
Number of frauds committed: 141
Number of frauds caught: 91
Number of false positives: 585
Number of false negatives: 50
Naive-Bayes accuracy : 0.9925681448451014
*** LinearSVC Results ***
Length of target data 85443
Number of predictions made: 85443
Number of frauds committed: 141
Number of frauds caught: 86
Number of false positives: 12
Number of false negatives: 55
LinearSVC accuracy : 0.9992158515033414
*** K-Neighbors Classifier Results ***
Length of target data 85443
Number of predictions made: 85443
Number of frauds committed: 141
Number of frauds caught: 22
Number of false positives: 1
Number of false negatives: 119
K-Neighbors Classifier accuracy : 0.9985955549313578
```

Removing the Time column made sense to me since there is no way to tell if the transactions were from the same account. The only change in the script was changing line 17 to `col_labels=col_labels[1:len(col_labels)-1]`. Of the three ways I considered the Time column, this way worked produced the best results. Dropping Time from the dataset gave these results:

```
*** Naive-Bayes Estimater Results ***
Length of target data 85443
Number of predictions made: 85443
Number of frauds committed: 141
Number of frauds caught: 120
Number of false positives: 1900
Number of false negatives: 21
Naive-Bayes accuracy : 0.9775171751928186
*** LinearSVC Results ***
Length of target data 85443
Number of predictions made: 85443
Number of frauds committed: 141
Number of frauds caught: 82
Number of false positives: 10
```

```
Number of false negatives:  59
LinearSVC accuracy :  0.9991924440855307
*** K-Neighbors Classifier Results ***
Length of target data  85443
Number of predictions made:  85443
Number of frauds committed:  141
Number of frauds caught:  94
Number of false positives:  6
Number of false negatives:  47
K-Neighbors Classifier accuracy :  0.9993797034280163
```

The Naive-Bayes caught more frauds, but had a much larger number of false positives than before. LinearSVC caught fewer frauds and K-Neighbors Classifier caught many more frauds and had only 6 false positives.

In this test, I replaced the Time column with the increment of time between the current transaction and the previous one. There was a substantial increase in runtime with slightly different results. The Naive-Bayes had a higher number of false positives.

```
*** Naive-Bayes Estimater Results ***
Length of target data  85443
Number of predictions made:  85443
Number of frauds committed:  141
Number of frauds caught:  120
Number of false positives:  2001
Number of false negatives:  21
Naive-Bayes accuracy :  0.976335100593378
*** LinearSVC Results ***
Length of target data  85443
Number of predictions made:  85443
Number of frauds committed:  141
Number of frauds caught:  82
Number of false positives:  10
Number of false negatives:  59
LinearSVC accuracy :  0.9991924440855307
*** K-Neighbors Classifier Results ***
Length of target data  85443
Number of predictions made:  85443
Number of frauds committed:  141
Number of frauds caught:  93
Number of false positives:  7
Number of false negatives:  48
K-Neighbors Classifier accuracy :  0.9993562960102056
```

Pandas coding challenge

Dataset The dataset ny-demographics.csv contains information on the residential demographics of each census tract in New York state. The dataset contains the following variables:

geoid11	11-digit geographic identifier for census tract
geoid11name	Name of census tract, county, and state
population	Number of residents in census tract
asian	Number of tract residents who are non-Hispanic Asians
black	Number of tract residents who are non-Hispanic blacks
hispanic	Number of tract residents who are Hispanic
white	Number of tract residents who are non-Hispanic whites

A row in the dataset describes one census tract. For example, the row that begins

geoid11	geoid11name
36001000100	Census Tract 1, Albany County, New York

indicates that there are 2139 residents in Census Tract 1 of Albany County, New York, of whom 55 are non-Hispanic Asians. There are 4919 census tracts in New York state and there are no missing values in the dataset. The "geoid11" variable has the property that the first 2 digits identify the state (36 = New York) and the first 5 digits identify the county within the state (36001 = Albany, New York). Tracts partition a county: if you add up all the residents in the 75 tracts in Albany County, this equals the population of Albany County (304,204 residents). There are nine tracts in which the population variable is the population count followed by "(rXXXXX)", where XXXXX is a 5-digit revision number, indicating that the Census Bureau revised the population count at some point after the initial data release.

Task Your assignment is to write a short script that generates a county-level dataset describing each New York county's demographics. In particular, please produce a CSV file containing the following variables:

geoid5	5-digit geographic identifier for county
geoid5name	Name of county and state
population	Number of residents in county
asian share	Fraction of county residents who are non-Hispanic Asians
black share	Fraction of county residents who are non-Hispanic blacks
hispanic share	Fraction of county residents who are Hispanic
white share	Fraction of county residents who are non-Hispanic whites
tracts	Number of census tracts in county
asian majority tracts	Number of tracts in county where > 50% of residents are non-Hispanic Asians
black majority tracts	Number of tracts in county where > 50% of residents are non-Hispanic blacks
hispanic majority tracts	Number of tracts in county where > 50% of residents are Hispanic
white majority tracts	Number of tracts in county where > 50% of residents are non-Hispanic whites
nomajority tracts	Number of tracts in county where no demographic category has > 50% share

The dataset should include one observation for each of New York's 62 counties and should be sorted by the 5-digit code that identifies the county. If you need to make judgment calls about how to process the data, please write us a short note describing the decisions you made.

Solution: Assumptions:

- The demographics are based on the unrevised populations. (I based this on line 375 where the revised population is more than 10X the sum of the ethnic populations)
- I assumed that I should count any tract(s) with no one living in them.

The python script outputs two csv files with revised and unrevised numbers in the populations collumn.

```
#[2019-08-22] Herman Schaumburg herman.schaumburg@gmail.com
#
#Revision 2
#
#Use the command below to run this script with the example data file
# ny-demographics.csv:
# python3 task1.py ny-demographics ny-demographics-out
#
#It produces two output files ny-demographics-out_revised.csv ny-demographics-out_unrevised.csv
#
#
```

```
#To run, you need python and the library pandas.
#
#Here are instructions on installing pandas.
#
#https://pandas.pydata.org/pandas-docs/stable/install.html
#
#
#I used the following method to install pandas:
#Installing from PyPI
#pandas can be installed via pip from PyPI.
#
#pip install pandas
#
#
#

#importing libraries
import pandas as pd
import numpy as np
import sys
from datetime import datetime

#The following handles arguements and throws an error if the wrong number are given.
num_arg=len(sys.argv)-1
if num_arg!=2:
        print( "Two arguements must be supplied -- input file and output file without csv extension.")
input_file=sys.argv[1]
input_file+=".csv"
print( "Input File....."+input_file)
output_file=sys.argv[2]
print("Output Files...."+output_file+"_unrevised.csv"+" "+output_file+"_revised.csv")

#The following lines are for timing
print ("Starting...")
startTime = datetime.now()

#Create data frame for input
df = pd.read_csv(input_file, dtype={'geoid11': object})

#Create data frame to hold output
columns=['geoid5',
'geoid5name',
'population',
'asian share',
'black share',
'hispanic share',
'white share',
'tracts',
'asian majority tracts',
'black majority tracts',
'hispanic majority tracts',
'white majority tracts',
'nomajority tracts']
df_out=pd.DataFrame(columns=columns)

#Converting geoid11 to 5
```

```
def get_digits(item):
        return str(item)[0:5]

df['geoid11'] =df['geoid11'].map(get_digits)
df['geoid11'] = pd.to_numeric(df['geoid11'])
#Finished converting geoid11 to 5

#Extracting unique geoid5 id's for output
df_out['geoid5']=pd.Series(df['geoid11'], name='geoid5').unique()



#Split up geoid11name name
df['geoid11name'].replace(regex=True,inplace=True,to_replace=r'Census Tract ',value=r'')

# new data frame with split value columns
new = df['geoid11name'].str.split(",", n = 1, expand = True)

# making seperate last name column from new data frame
df['County, State']= new[1]

# Dropping old Name columns
df.drop(columns =['geoid11name'], inplace = True)

#Extract unique values in geoid5name
df_out['geoid5name']=pd.Series(df['County, State'], name='geoid5').unique()
num_rows_of_output=df_out.shape[0]

#Get only revised populations in input data frame
df['Rpopulation'] = df['population'].str.extract(r"\(r(.*?)\)", expand=False)
#Get only unrevised populations that are revised later
df['Population'] = df['population'].str.extract(r"(.*?)\(r", expand=False)
#Fill in these collumns with ones that were not revised
df.Rpopulation.fillna(df.population, inplace=True)
df['Rpopulation'] = pd.to_numeric(df['Rpopulation'])
df.Population.fillna(df.population, inplace=True)
df['Population']=pd.to_numeric(df['Population'])
#Replace population in input data frame by deleting and renaming collumns
df=df.drop('population', axis=1)
df.rename(columns = {'Population':'population'}, inplace = True)

#Initializing lists for counting the majority tracts
num_rows_of_input=df.shape[0]
asian_majority_tracts=[]
black_majority_tracts=[]
hispanic_majority_tracts=[]
white_majority_tracts=[]
nomajority_tracts=[]
tracts=[]

#This for loop is used to find the nonempty tracts and identify those tracts with a majority  or
#nonmajority ethnicity.
#Rev 2 improved syntax
for j in range(0, num_rows_of_input):
        if df['population'][j]>0:
                half_population=float(0.5*df['population'][j])
                val_w=float(df['white'][j])
```

```
                val_h=float(df['hispanic'][j])
                val_b=float(df['black'][j])
                val_a=float(df['asian'][j])
                w=float(0)
                h=float(0)
                b=float(0)
                a=float(0)
                n=float(0)
                if (val_w>half_population):
                        w=1
                elif (val_h>half_population):
                        h=1
                elif (val_b>half_population):
                        b=1
                elif (val_a>half_population):
                        a=1
                if w+h+b+a==0:
                        n=1
                white_majority_tracts.append(w)
                hispanic_majority_tracts.append(h)
                black_majority_tracts.append(b)
                asian_majority_tracts.append(a)
                nomajority_tracts.append(n)
                tracts.append(1)
        else:
                white_majority_tracts.append(0)
                hispanic_majority_tracts.append(0)
                black_majority_tracts.append(0)
                asian_majority_tracts.append(0)
                nomajority_tracts.append(0)
                tracts.append(1)

#Storing lists into data frame
df['asian_major_tract']=asian_majority_tracts
df['white_major_tract']=white_majority_tracts
df['hispanic_major_tract']=hispanic_majority_tracts
df['black_major_tract']=black_majority_tracts
df['nonmajority_tract']=nomajority_tracts
df['tract']=tracts

#Initializing variable to store the numbers of county populations
num_rows_of_output=df_out.shape[0]
population_sum=[]
asian_share=[]
black_share=[]
hispanic_share=[]
white_share=[]
asian_majority_tracts=[]
black_majority_tracts=[]
hispanic_majority_tracts=[]
white_majority_tracts=[]
nomajority_tracts=[]
num_tracts=[]

#Most work is done in this loop, it computes the population sums, those for each demographic,
#how many tracts are in each county, and different share fractions.
```

```
for j in range(0, num_rows_of_output):
        #population_sum.append(df.loc[df['geoid11'] == df_out['geoid5'][j], 'population'].sum())
        #Rev2 fixed syntax to store the indexes used in the above command to find them only once rather
        #than repeatedly.

        indices=df.loc[df['geoid11'] == df_out['geoid5'][j], 'population'].index.values

        population_sum.append(df.loc[indices, 'population'].sum())
        asian_sum= df.loc[indices,'asian'].sum()
        black_sum=df.loc[indices,'black'].sum()
        hispanic_sum=df.loc[indices,'hispanic'].sum()
        white_sum=df.loc[indices,'white'].sum()

        asian_majority_tracts_sum=df.loc[indices, 'asian_major_tract'].sum()
        black_majority_tracts_sum=df.loc[indices, 'black_major_tract'].sum()
        hispanic_majority_tracts_sum=df.loc[indices, 'hispanic_major_tract'].sum()
        white_majority_tracts_sum=df.loc[indices, 'white_major_tract'].sum()
        nomajority_tracts_sum=df.loc[indices, 'nonmajority_tract'].sum()
        tracts_sum=df.loc[indices,  'tract'].sum()

        num_tracts.append(tracts_sum)
        if population_sum[j]>0:
                asian_share.append(asian_sum/float(population_sum[j]))
                black_share.append(black_sum/float(population_sum[j]))
                hispanic_share.append(hispanic_sum/float(population_sum[j]))
                white_share.append(white_sum/float(population_sum[j]))
                asian_majority_tracts.append(asian_majority_tracts_sum)
                black_majority_tracts.append(black_majority_tracts_sum)
                hispanic_majority_tracts.append(hispanic_majority_tracts_sum)
                white_majority_tracts.append(white_majority_tracts_sum)
                nomajority_tracts.append(nomajority_tracts_sum)
        else:
                asian_share.append('NA')
                black_share.append('NA')
                hispanic_share.append('NA')
                white_share.append('NA')
                asian_majority_tracts.append('NA')
                black_majority_tracts.append('NA')
                hispanic_majority_tracts.append('NA')
                white_majority_tracts.append('NA')
                nomajority_tracts.append('NA')
                num_tracts.append(tracts_sum)

df_out['population']=population_sum
df_out['asian share']=asian_share
df_out['black share']=black_share
df_out['hispanic share']=hispanic_share
df_out['white share']=white_share
df_out['tracts']=num_tracts
df_out['asian majority tracts']=asian_majority_tracts
df_out['black majority tracts']=black_majority_tracts
df_out['hispanic majority tracts']=hispanic_majority_tracts
df_out['white majority tracts']=white_majority_tracts
df_out['nomajority tracts']=nomajority_tracts

#Export dataframe to csv
```

```python
df_out.to_csv('output_unrevised.csv', index=False)

#Find total county populations based on revised population numbers.
population_sum=[]
for j in range(0, num_rows_of_output):
        population_sum.append(df.loc[df['geoid11'] == df_out['geoid5'][j], 'Rpopulation'].sum())

df_out['population']=population_sum

#Export dataframe to csv
df_out.to_csv('output_revised.csv', index=False)
total_time=datetime.now() - startTime
print( "CPU time used ", total_time)
print( "...Done")
```

Python coding challenge

The following are my solutions to problems posed in a coding challenge.
Problem 1 On Pandora, the currency is called Unob, U. There are six coins in circulation:

U1, U5, U10, U20, U50, U100

It is possible to make U500 in the following way:

$$3 \times \text{U100} + 2 \times \text{U50} + 4 \times \text{U20} + 1 \times \text{U10} + 1 \times \text{U5} + 5 \times \text{U1}$$

How many different ways can U500 be made using any number of coins?

Solution: I used a generating function to solve this problem. A generating function is a power series

$$g(x) = \sum_{j=0}^{\infty} A_n x^n,$$

where A_n counts some set. An example relevant to the solution is the number of ways to write an integer n as a sum of integers 1 or 5 where order is irrelevant. The generating function is

$$\begin{aligned} h(x) &= (1 + x + x^{1+1} + x^{1+1+1} + \cdots)(1 + x^5 + x^{5+5} + x^{5+5+5} + \cdots) && (1) \\ &= \frac{1}{1-x}\frac{1}{1-x^5}. && (2) \end{aligned}$$

The fact that the two factors on the right hand side of (1) are geometric series gives (2). To understand this generating function consider the case where $n = 15$ the following make contributions to the coefficient of $x^1 1$:

$$x^{1+1+1+1+1+1+1+1+1+1+1+1+1+1+1} \quad x^{1+1+1+1+1+1+1+1+5} \quad x^{1+5+5}.$$

The exponents correspond to the three ways to make U15 from U1 and U5 coins. The following Python script gives the 500th coefficient of the generating function of

$$f(x) = \frac{1}{(1-x)(1-x^5)(1-x^{10})(1-x^{20})(1-x^{50})(1-x^{100})},$$

which is the answer for this problem.

```python
#Computes the 500th coefficient of the Taylor series for the
#generating function of partitions of n into parts of size 1,5,10,20,50, or 100.
#The generating function is 1/((1-x)*(1-x^5)*(1-x^10)*(1-x^20)*(1-x^50)*(1-x^100))
#The algopy library does most of the work here.  This script follows the method given here:
#https://pythonhosted.org/algopy/examples/series_expansion.html

import time
start_time=time.time()
import numpy
from algopy import UTPM

def f(x):
        return 1/((1-x)*(1-x**5)*(1-x**10)*(1-x**20)*(1-x**50)*(1-x**100))

D= 501; P=1
x = UTPM(numpy.zeros((D,P)))
x.data[0,0] = 0
x.data[1,0] = 1

y=f(x)
#Returns the 500 coefficient of the Taylor series of f(x) centered at 0:
print(int(y.data[500,0]))
end_time=time.time()
print("CPU time including library initialization: %f" %(end_time-start_time))
```

Problem 2

- Gregor has eight five-sided dice, each with faces numbered 1, 2, 3, 4, 5.
- Oberyn has four ten-sided dice, each with faces numbered 1, 2, 3, 4, 5, 6, 7, 8, 9, 10.

Gregor and Oberyn roll their dice and compare totals: the highest total wins. The result is a draw if the totals are equal. What is the probability that Gregor beats Oberyn (i.e. Gregor wins / N games)?

Solution: The idea for my solution is to find the sum

$$\sum_{i=9}^{40}(\text{Number of ways Oberyn rolls sum to } < i) \times (\text{Number of ways Gregor rolls sum to } i)$$

and divide by the number of games, which is

$$5^8 \times 10^4.$$

The part of the solution involving Oberyn is computed according using

$$\text{Number of ways Oberyn rolls sum to } < i = \sum_{j=4}^{i}(\text{Number of ways Oberyn rolls sum to } j).$$

The solution be completed by finding the number of ways an M sided dice sums to n. That is the problem we need to solve is

Find the number of solutions to

$$\textbf{(I)} \qquad x_1 + x_2 + \cdots + x_k = n$$

where $1 \le x_j \le M$.

This is related to problem

Find the number of solutions to

$$\textbf{(II)} \qquad y_1 + y_2 + \cdots + y_k = n$$

where $1 \leq x_j$.

whose solution is the binomial coefficient ${}_{n-1}C_{k-1}$, $n-1$ choose $k-1$. This fact may be understood by considering M ones with wedges $\wedge$ between them

$$1_\wedge 1_\wedge 1_\wedge 1_\wedge \cdots {}_\wedge 1.$$

Each wedge may is a place where one of $k-1$ plus signs may be placed. The number of ones between plus signs correspond to the numbers y_j. The number of ways to choose which of the $n-1$ wedges get one of the k plus signs is ${}_{n-1}C_{k-1}$.

Let A_p be the number of solutions to **(II)** where at least p of the y_j are greater than M. From the principle of inclusion and exclusion the solution to problem **(I)** is

$${}_{n-1}C_k + \sum_{p>1} (-1)^p A_p$$

If we subtract M from each of the p integers that are greater than M, we see that A_p is the solutions to problem **(II)** with $n - pM$ substituted for n times the number of ways to pick the p integers that are greater from the k integers. Thus,

$$A_p =_k C_p \times_{n-pM-1} C_k,$$

and

$$\text{number of ways an } M \text{ sided dice sums to } n =_{n-1} C_k + \sum_{p>1} (-1)^p_k C_p \times_{n-pM-1} C_{k-1}.$$

This is computed by the function `sum_soln_ct` in the python script below. The script returns the answer

$$\frac{2278263384}{3906250000} = 0.583235426304.$$

A potential area for improvement is to lower the number of flops to compute the the binomial coefficients by storing them in a table and using a recurrence formula to compute further rows.

```python
import time
start_time=time.time()

#Compute binomial coeficients
def nCr(n, r):
  if (r>n):
    return 0;
  else:
    numerator=1
    denominator=1
    for i in range(n-r+1,n+1):
      numerator*=i
    for i in range(1,r+1):
      denominator*=i
    return numerator/denominator

#Computes the number of solutions to x_1+x_2+...+x_k=n with 1<=x_i<=M
#Using principle of inclusion/exclusion
def sum_soln_ct(n,k,M):
```

```
  sum_soln=nCr(n-1,k-1)
  sign=1
  for i in range(1,k+1):
    if (n>i*M+1):
      sign*=-1
      sum_soln+=nCr(k,i)*sign*nCr(n-i*M-1,k-1)
  return sum_soln

oberyn_rolls_lt_i=0
gregor_rolls_i=0
gregor_wins_ct=0

for i in range(4, 8):
  oberyn_rolls_lt_i+=sum_soln_ct(i,4,10)

for i in range(9, 41):
  oberyn_rolls_lt_i+=sum_soln_ct(i-1,4,10)
  gregor_rolls_i=sum_soln_ct(i,8,5)
  gregor_wins_ct+=oberyn_rolls_lt_i*gregor_rolls_i
#  Used to test that there are 9999 oberyn rolls less than 40.
#  if (i==40):
#    print("==========")
#    print sum_soln_ct(i,8,5)
#    print oberyn_rolls_lt_i

print(gregor_wins_ct/float(5**8*10**4))
end_time=time.time()
print("CPU time including library initialization: %f" %(end_time-start_time))
```

Problem 3 Problem 3. Find the maximum value from the matrix where each number is the only one in its row and column. For example, for the matrix below the maximum value equals 3315 (= 863 + 383 + 343 + 959 + 767):

7	53	183	439	863
497	383	563	79	973
287	63	343	169	583
627	343	773	959	943
767	473	103	699	303

Find the maximum value of:

7	53	183	439	863	497	383	563	79	973	287	63	343	169	583
627	343	773	959	943	767	473	103	699	303	957	703	583	639	913
447	283	463	29	23	487	463	993	119	883	327	493	423	159	743
217	623	3	399	853	407	103	983	89	463	290	516	212	462	350
960	376	682	962	300	780	486	502	912	800	250	346	172	812	350
870	456	192	162	593	473	915	45	989	873	823	965	425	329	803
973	965	905	919	133	673	665	235	509	613	673	815	165	992	326
322	148	972	962	286	255	941	541	265	323	925	281	601	95	973
445	721	11	525	473	65	511	164	138	672	18	428	154	448	848
414	456	310	312	798	104	566	520	302	248	694	976	430	392	198
184	829	373	181	631	101	969	613	840	740	778	458	284	760	390
821	461	843	513	17	901	711	993	293	157	274	94	192	156	574
34	124	4	878	450	476	712	914	838	669	875	299	823	329	699
815	559	813	459	522	788	168	586	966	232	308	833	251	631	107
813	883	451	509	615	77	281	613	459	205	380	274	302	35	805

(Hint: the anser is > 13930, you can implement the Hungarian Algorithm but it is not required)

Solution: The hint lead me to the name of a closely related problem, namely the assignment problem. In the assignment problem, the minimum sum where where each summand is the only one in its row and column is returned. Thus the solution to the problem at hand is given by

$$\text{solution for matrix } M = -\text{ solution of assignment problem for matrix } (-M)$$

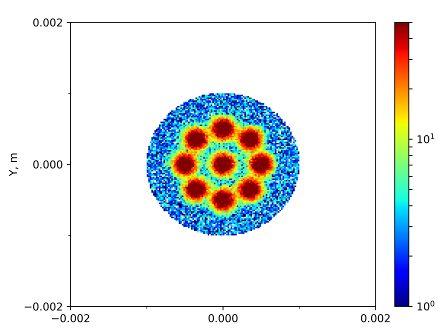
The script returns an answer of 13938. Uncommenting the line 18 gives the columns containing the summands which give this maximum sum, `[ 9 10 7 4 3 0 13 2 14 11 6 5 12 8 1]`.

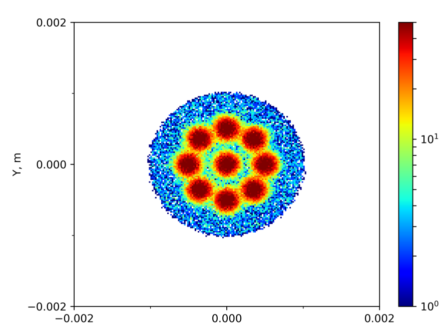
```
1  #This uses a built in solver in scipy.  It finds the minimum
2  # for -A, which is the opposite maximum of A.
3  import time
4  start_time=time.time()
5  import csv
6  import numpy as np
7  from scipy.optimize import linear_sum_assignment
8  
9  with open('matrix.ssv', 'r') as f:
10   matrix = list(csv.reader(f, delimiter=' '))
11 
12 matrix = np.array(matrix[0:], dtype=np.int)
13 matrix = -matrix
14 matrix_sz=len(matrix)
15 
16 [rows,cols]=linear_sum_assignment(matrix)
17 #print(rows)
18 #print(cols)
19 sum=0
20 for i in range(0,matrix_sz):
21   sum+=matrix[rows[i],cols[i]]
22 print(-sum)
23 end_time=time.time()
24 print("CPU time including library initialization: %f" %(end_time-start_time))
```

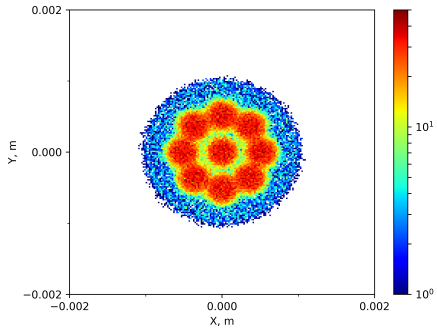
Matplotlib script

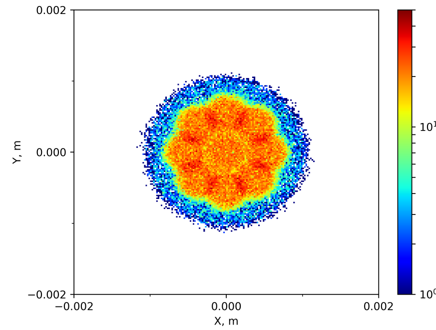
This plotting script takes files `x_PHAD_⟨index⟩.ssv` and `y_PHAD_⟨index⟩.ssv` then creates a series of density plots within a window specified by the ranges `xlims` and `ylims`.

Below is a sample of a few plots created with this script.









An example animation created by different script that employed LaTeXand imagemagik instead may be viewed here.

```
import numpy as np
from matplotlib.pyplot import *
from matplotlib.ticker import (MultipleLocator, FormatStrFormatter, AutoMinorLocator)
import os

def read_ascii(fn):
    """ reads ascii file, returns  list of lines """
    try:
        inputfile = open(fn, 'r')
    except Exception as error:
        print(error)
        return
    lines = inputfile.readlines()
    inputfile.close()
    return lines

def read_ascii_column(fn):
    lines = read_ascii(fn)
    data = []
    for line in lines:
        st = line.split()
        if st[0] != '#' and st[0] != '!' and st[0] != '*': # possible comments identifiers
            data.append(float(i) for i in list(st))
    data = list(zip(*data))
    return data

my_cmap=matplotlib.cm.get_cmap('jet')
my_cmap.set_bad('w')


# set up initial and last ID-numbers
start=0
end=78
increment=26

#Number of digits in output
digits=4

#Set plot ranges
xlimit=2E-3
xlims=[-xlimit,xlimit]
```

```
ylims=xlims

digits=digits-1



count=int(start/increment)
for i in range(start,end+1,increment):
        print("Processing file "+str(i)+" for image "+str(count))
        x_fn = 'x_PHAD_%s.ssv'%str(i)
        y_fn = 'y_PHAD_%s.ssv'%str(i)

        x = read_ascii_column(fn = x_fn)[0]
        y = read_ascii_column(fn = y_fn)[0]

        # Comment to turn off tick marks
        ax = matplotlib.pyplot.subplot(111)
        ax.set_xticks([xlims[0],0,xlims[1]])
        ax.set_yticks([ylims[0],0,ylims[1]])
        # Sets minor tick marks
        XminorLocator = MultipleLocator(xlimit/2)
        YminorLocator = MultipleLocator(xlimit/2)
        ax.xaxis.set_minor_locator(XminorLocator)
        ax.yaxis.set_minor_locator(YminorLocator)

        # Comment to turn on tick marks
        #gca().tick_params(axis='x',labelbottom='False')
        #gca().tick_params(axis='y',labelleft='False')

    #histogramm 2D
    hist2d(x, y, norm=matplotlib.colors.LogNorm(), range=[xlims,ylims], bins=200, cmap=my_cmap, vmax=50)

        colorbar()
#    clim(0,50) #colorscale limits

        xlim(xlims) # X-axis limits
        ylim(ylims)

        xlabel('X, m') #labels
        ylabel('Y, m')
#    title('Density')
        if (count!=0):
                num_zeros=int(digits-np.floor(np.math.log(float(count),10.0)))
        else:
                num_zeros=digits
        image_label=str(0)
        for j in range(1,num_zeros):
                image_label=image_label+str(0)
        savefig('%s.png'%(image_label+str(count)), dpi=600)
        matplotlib.pyplot.clf()
        count+=1

# Must be edited so that number of zeros and X in %0X are the same
#Sets command for ffmpeg:
#command='ffmpeg -framerate 10 -start_number 0000 -i %4d.png video.mov'
#command='ffmpeg -framerate 10 -start_number 0000 -i %4d.png video.mp4'
```

```
#command='ffmpeg -r 1/5 -start_number 0000 -i %4d.png -c:v libx264 -vf fps=10 -pix_fmt yuv420p video.mp4'

#Uncomment to run annimation command
#os.system(command)
```