Supporting Software Code

Isotopically encoded nanotags for multiplexed ion beam imaging

Authors:

Dr. Stefan Harmsen[1,2,*], Dr. Ahmet F. Coskun[1,2,3,4,*, #], Shambavi Ganesh[4,5]
Dr. Garry P. Nolan[3, #], and Dr. Sanjiv S. Gambhir[1,2,6,7#]

Affiliations:

[1]Department of Radiology, Stanford University School of Medicine, Stanford, CA, USA
[2]Molecular Imaging Program at Stanford University (MIPS), Bio-X Program, Stanford University School of Medicine, Stanford, USA
[3]Department of Microbiology and Immunology, Stanford University School of Medicine, Stanford, CA, USA
[4]Wallace H. Coulter Department of Biomedical Engineering, Georgia Institute of Technology and Emory University, Atlanta, GA, USA
[5]School of Electrical and Computer Engineering, Georgia institute of Technology and Emory University, Atlanta, GA, USA
[6]Department of Bioengineering, Stanford University Schools of Medicine and of Engineering, Stanford, USA
[7]Department of Materials Science & Engineering, At Stanford University, School of Medicine, Stanford, CA, United States.

#Corresponding authors
Ahmet F. Coskun, Ph.D. (ahmet.coskun@bme.gatech.edu)
Sanjiv S. Gambhir, M.D., Ph.D. (sgambhir@stanford.edu)
Garry P. Nolan, Ph.D. (gnolan@stanford.edu)

*These authors contributed equally

Summary:

This document contains 2 supporting codes, each of which classifies the nanotags into a chosen number of classes (6 or 7), and plots the results obtained. Supporting Code 1, after preprocessing the data, classifies the nanotag data into 6 classes using SVM and KNN (2 to 4 neighbors) and plots a bar graph comparing the ground truth data with an algorithmic result. Supporting Code 2, after preprocessing the data, classifies the nanotag data into 7 classes using SVM and KNN (2 to 4 neighbors), and plots a bar graph comparing the ground truth data with an algorithmic result, along with a 3-Dimensional Scatter plot of the classified data points obtained. The difference between Supporting code 1 and Supporting code 2 is the desired number of classes the nanotags need to be classified into. It has been divided into 2 versions for ease of implementation.

Supporting Code 1:

```
## Implementing KNN/SVM for 6 classes
```

Part 1: Organization and Parsing of Data

```
##The following code imports the modules necessary. It is suggested to cross-verify that all
##required modules are installed. In the event it is not, it may give rise to an error.

import numpy as np
import matplotlib.pyplot as plt
import matplotlib.ticker as ticker
from matplotlib.ticker import (MultipleLocator, FormatStrFormatter,
import pandas as pd
from sklearn.model_selection import train_test_split, StratifiedShuffleSplit, GridSearchCV
from sklearn.svm import SVC
from sklearn.metrics import classification_report, confusion_matrix, accuracy_score
from sklearn import svm ,neighbors
```

```
from pylab import *

##Preprocessing of Data, from nanotaglistsortednorm.csv (An excel file which contains the
#relevant information about the nanotags. This information is also displayed at the end of the file,
#in the form of a table.)

#read csv file
df1 = pd.read_csv("nanotaglistsortednorm.csv")

#assign group numbers to the pre-existing labels
dictLabel = {'b001':1,'b010':2,'b100':3,'b101':4,'b110':5,'b111':6}
df1 = df1.query('code == "b001" or code =="b010" or code == "b100" or code == "b101" or code
== "b110" or code == "b111" ')
labels = df1['code']

## Checking the number of datapoints in each class
new_labels = [dictLabel[l] for l in labels]
class_count = { 1: 0, 2: 0, 3:0 , 4:0 , 5:0, 6:0}
for l in new_labels:
    class_count[l] += 1

## Splitting it into test and train data which are modified depending on the size of the dataset
X=df1[['F','Br','I']]
y=new_labels
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.15)
```

Part 2: Training of Data

```
## Searching over the grid space for the best parameters for the SVM Model.
C_range = np.logspace(-2, 10, 13)
gamma_range = np.logspace(-9, 3, 13)
```

```
param_grid = dict(gamma=gamma_range, C=C_range)
cv = StratifiedShuffleSplit(n_splits=5, test_size=0.15, random_state=42)
grid = GridSearchCV(SVC(), param_grid=param_grid, cv=cv)
grid.fit(X, y)

## SVM Model Fit, using the parameters previously obtained that are best fit for the model. The
#best parameters can be selected from the train-test split, in order to validate the robustness of the
#model and get an unbiased estimate of the algorithm's performance on a dataset. Once the
#parameters are determined, the algorithm can either be implemented on the original dataset, or
#the test data.

svclassifier = SVC(kernel='rbf',C = 10.0, gamma = 1.0)
svclassifier.fit(X, y)

#Using the SVM model created, values are predicted for the input chosen.
y_pred = svclassifier.predict(X)

## K-NN Model fit, for different instances of the model, with number of neighbors
varying from ##2 to 4.

#Please change n_neighbors using [2,3,4] values to get the desired results and graphs.
#The KNN classifier is created, using the appropriate "n_neighbors" value and can be used on the
#whole dataset, or the test dataset as per experimental protocol.

neigh = neighbors.KNeighborsClassifier(n_neighbors=3)
neigh.fit(X, y)

# Using the KNN model created, values are predicted for the input chosen.
y_pred = neigh.predict(X)
acc_test = 100*np.sum(y_pred == y)/len(y)
```

#Output obtained is printed.

print(confusion_matrix(y, y_pred))

print(classification_report(y, y_pred))

Example output:

Confusion Matrix:

```
[[21  0  0  0  0  0]
 [ 0 14  0  0  0  0]
 [ 0  0 13  1  0  0]
 [ 0  0  0  8  0  2]
 [ 0  0  1  0  5  1]
 [ 0  0  0  1  0 15]]
```

Classification Report:

	Precision	recall	f1-score	support
1	1.00	1.00	1.00	21
2	1.00	1.00	1.00	14
3	0.93	0.93	0.93	14
4	0.80	0.80	0.80	10
5	1.00	0.71	0.83	7
6	0.83	0.94	0.88	16
avg / total	0.93	0.93	0.93	82

Part 3: Plotting of Data

Defining plotting functions

```
def make_meshgrid(x, y, h=.02):
    x_min, x_max = x.min() - 1, x.max() + 1
    y_min, y_max = y.min() - 1, y.max() + 1
    xx, yy = np.meshgrid(np.arange(x_min, x_max, h), np.arange(y_min, y_max, h))
```

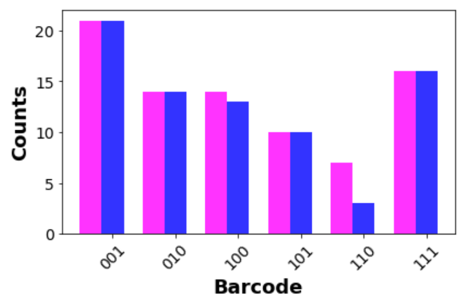
```
    return xx, yy

def plot_contours(ax, clf, xx, yy, **params):
    Z = clf.predict(np.c_[xx.ravel(), yy.ravel()])
    Z = Z.reshape(xx.shape)
    out = ax.contourf(xx, yy, Z, **params)
    return out

 ## Plotting of data by setting appropriate parameters for the graphs

# data to plot
X = np.array(X)
X0 = X[:, 0]
n_groups = 6

#manual refers to ground truth number of tags in each class
manual = (21,14,14,10,7,16)

# One of the following variables can be chosen depending on which algorithm has to be used.
#This variables are updated after the above code has been run for SVM and KNN(2-4 neighbors)
SVM = (21,14,13,10,3,16)
KNN_2 = (21,12,12,10,5,15)
KNN_3 = (21,14,13,8,5,15)
KNN_4 = (14,20,12,13,8,3,14)

# create plot outline
fig, ax = plt.subplots()
index = np.arange(n_groups)
bar_width = 0.35
opacity = 0.8
```

```
#Create Bar plots
rects1 = plt.bar(index, manual, bar_width,
alpha=opacity,
color='magenta',
label='Manual')

rects2 = plt.bar(index + bar_width, kmeans_3, bar_width,
alpha=opacity,
color='blue',
label='Kmeans_3')

#Changing properties of labels and axes
plt.xlabel('Barcode',fontweight = 'bold',fontsize = 18)
plt.ylabel('Counts',fontweight = 'bold',fontsize = 18)
#plt.title('Bar plot')
plt.xticks(index + bar_width, ('001', '010', '100','101','110','111'))
plt.yticks(np.arange(5), ('0', '5','10','15','20' ))
fontsize = 14

ax = gca()
ax.set_xticklabels(['001', '010', '100','101','110','111'], minor=False, rotation=45)
for tick in ax.xaxis.get_major_ticks():
    tick.label1.set_fontsize(fontsize)
    tick.label1.set_fontweight('bold')
for tick in ax.yaxis.get_major_ticks():
    tick.label1.set_fontsize(fontsize)
  #  tick.label1.set_fontweight('bold')
for axis in [ax.yaxis]:
    axis.set_major_locator(plt.MaxNLocator(5))

plt.tight_layout()
```

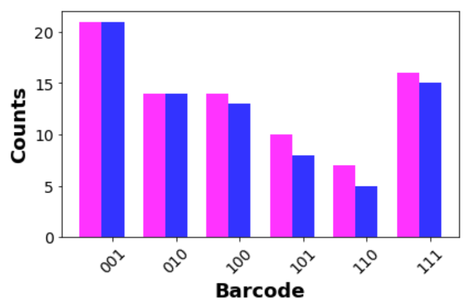
plt.show()

#Example plots for SVM and KNN(2-4 neighbours) as compared to ground truth data

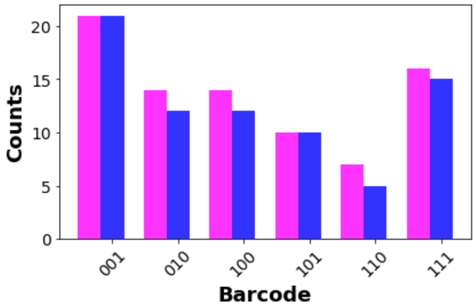
#Manual counts vs SVM classification:



#Manual counts vs KNN-2 classification:



#Manual counts vs KNN-3 classification:



Supporting Code 2:

Implementing KNN/SVM for 7 classes

Part 1: Organization and Parsing of Data

Importing Modules necessary. It is suggested to cross-verify that all required modules are ##installed. In the event it is not, it may give rise to an error

```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
from sklearn.svm import SVC
from sklearn.metrics import classification_report, confusion_matrix ,accuracy_score
from sklearn import svm,  neighbors
from sklearn.model_selection import StratifiedShuffleSplit ,GridSearchCV , train_test_split
from pylab import *
import matplotlib.ticker as ticker
import umap
import seaborn as sns
from sklearn.decomposition import PCA
```

```
from mpl_toolkits.mplot3d import axes3d, Axes3D

##Preprocessing of Data, from nanotaglistsortednorm.csv (An excel file which contains the
#relevant information about the nanotags. This information is also displayed at the end of the file,
#in the form of a table.)

#read csv files
df1 = pd.read_csv("nanotaglistsortednorm.csv")

#assign group numbers to the pre-existing labels
dictLabel = {'b000':0,'b001':1,'b010':2,'b100':3,'b101':4,'b110':5,'b111':6}
#df1 = df1.query('code == "b000" or code == "b001" or code =="b100" or code == "b110" or code
== "b111" or code == "b010" or code == "b101"')
labels = df1['code']


# ## Checking the number of datapoints in each class
new_labels = [dictLabel[l] for l in labels]
class_count = {0: 0, 1: 0, 2: 0, 3:0 , 4:0 , 5:0, 6:0}
for l in new_labels:
    class_count[l] += 1
print(class_count)


# ## Splitting it into test and train Data (can vary depending on the size of the dataset)
X=df1[['F','Br','I']]
y=new_labels
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.15)
```

Part 2: Training of Data

```
##Grid search over the parameter space for SVM model

C_range = np.logspace(-2, 10, 13)
gamma_range = np.logspace(-9, 3, 13)
param_grid = dict(gamma=gamma_range, C=C_range)
cv = StratifiedShuffleSplit(n_splits=5, test_size=0.15, random_state=42)
grid = GridSearchCV(SVC(), param_grid=param_grid, cv=cv)
grid.fit(X_train, y_train)

## SVM Model Fit, using the parameters previously obtained that are best fit for the model. The
#best parameters can be selected from the train-test split, in order to validate the robustness of the
#model, and get an unbiased estimate of the algorithm's performance on a dataset. Once the
#parameters are determined, the algorithm can either be implemented on the original dataset, or
#the test data.

svclassifier = SVC(kernel='rbf',C = 10.0, gamma = 1.0)
svclassifier.fit(X, y)

#Using the SVM model created, values are predicted for the input.
y_pred = svclassifier.predict(X)

## K-NN Model fit, for different instances of the model, with number of neighbors varying from
##2 to 4.

#Please change n_neighbors using [2,3,4] values to get the desired results and graphs.
#The KNN classifier is created, using the appropriate "n_neighbors" value and can be used on the
#whole dataset, or the test dataset as per experimental protocol.

neigh = neighbors.KNeighborsClassifier(n_neighbors=4)
neigh.fit(X, y)
```

#Using the KNN model created, values are predicted for the input.

y_pred = neigh.predict(X)

acc_test = 100*np.sum(y_pred == y)/len(y)

#The output is printed

print(confusion_matrix(y, y_pred))

print(classification_report(y, y_pred))

#Example Output:

#Confusion Matrix:

```
[[14  1  1  0  1  0  0]
 [ 1 20  0  0  0  0  0]
 [ 2  0 12  0  0  0  0]
 [ 0  0  0 13  1  0  0]
 [ 0  0  0  0  8  0  2]
 [ 0  0  0  3  0  3  1]
 [ 1  0  0  0  1  0 14]]
```

#Classification matrix:

	precision	recall	f1-score	support
0	0.78	0.82	0.80	17
1	0.95	0.95	0.95	21
2	0.92	0.86	0.89	14
3	0.81	0.93	0.87	14
4	0.73	0.80	0.76	10
5	1.00	0.43	0.60	7
6	0.82	0.88	0.85	16
avg / total	0.86	0.85	0.84	99

Part 3: Plotting of Data

Defining plotting functions

def make_meshgrid(x, y, h=.02):

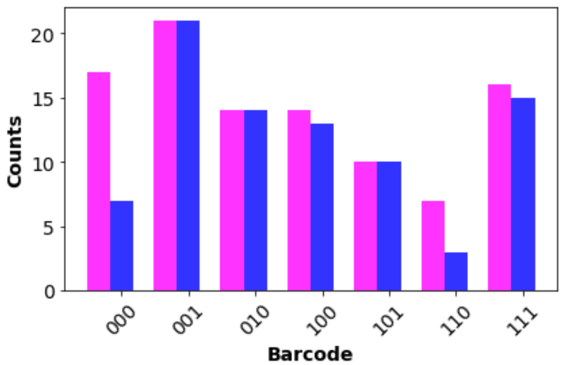
    x_min, x_max = x.min() - 1, x.max() + 1

```
    y_min, y_max = y.min() - 1, y.max() + 1
    xx, yy = np.meshgrid(np.arange(x_min, x_max, h), np.arange(y_min, y_max, h))
    return xx, yy

def plot_contours(ax, clf, xx, yy, **params):
    Z = clf.predict(np.c_[xx.ravel(), yy.ravel()])
    Z = Z.reshape(xx.shape)
    out = ax.contourf(xx, yy, Z, **params)
    return out

## Plotting of data

# data to plot
print(type(X))
X = np.array(X)
X0 = X[:, 0]
n_groups = 7
#manual refers to ground truth number of tags in each class
manual = (17,21,14,14,10,7,16)

# One of the following variables can be chosen depending on which algorithm has to be used.
#This variables are updated after the above code has been run for SVM and KNN(2-4 neighbours)
SVM = (7,21,14,13,10,3,15)
KNN_2 = (17,19,10,14,10,5,4)
KNN_3 = (14,20,12,13,8,5,14)
KNN_4 = (14,20,12,13,8,3,14)

# create plot
fig, ax = plt.subplots()
index = np.arange(n_groups)
bar_width = 0.35
```

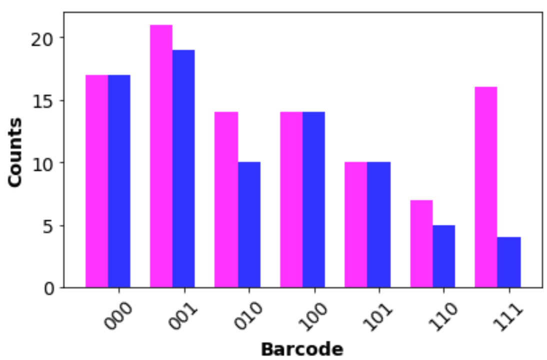
```
opacity = 0.8
rects1 = plt.bar(index, manual, bar_width,
alpha=opacity,
color='magenta',
label='Manual')

rects2 = plt.bar(index + bar_width, KNN_3, bar_width,
alpha=opacity,
color='blue',
label='kmeans_4')

plt.xlabel('Barcode',fontweight = 'bold',fontsize = 18)
plt.ylabel('Counts',fontweight = 'bold',fontsize = 18)

plt.xticks(index + bar_width, ('000', '001', '010', '100','101','110','111'))

fontsize = 14
ax = gca()
plt.yticks(np.arange(5), ('0', '5','10','15','20' ))
ax.set_xticklabels(['000','001', '010', '100','101','110','111'], minor=False, rotation=45)
for tick in ax.xaxis.get_major_ticks():
    tick.label1.set_fontsize(fontsize)
    #tick.label1.set_fontweight('bold')
for tick in ax.yaxis.get_major_ticks():
    tick.label1.set_fontsize(fontsize)
    #tick.label1.set_fontweight('bold')
for axis in [ax.yaxis]:
    axis.set_major_locator(ticker.MaxNLocator(5))
plt.tight_layout()
plt.show()
```

#Example Plots:

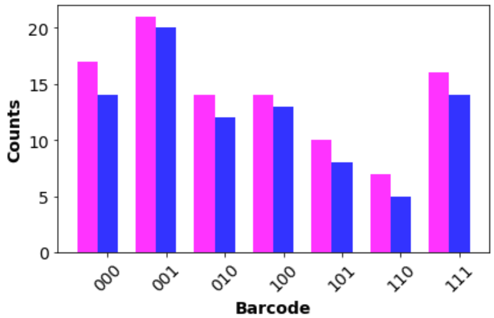
#Manual counts vs SVM classification:



#Manual counts vs KNN-2 classification:



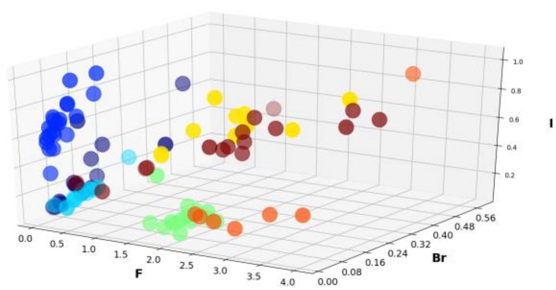
#Manual counts vs KNN-3 classification:

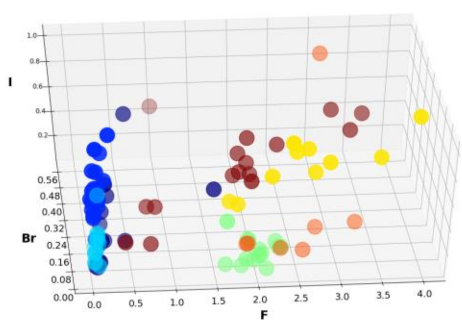



```
# ## Scatterplot 3D of data

X=df1[['F','Br','I']]
X_F= X[['F']]
X_Br= X[['Br']]
X_I =X[['I']]

N = 7

cmap = plt.cm.jet
cmaplist = [cmap(i) for i in range(cmap.N)]
cmap = cmap.from_list('Custom cmap', cmaplist, cmap.N)
bounds = np.linspace(0,N,N+1)
norm = mpl.colors.BoundaryNorm(bounds, cmap.N)
fig = plt.figure()

ax = Axes3D(fig)
scat = ax.scatter(X_F,X_Br,X_I,c=y,cmap=cmap,norm=norm)
ax.set_xlabel('X-axis')
ax.set_ylabel('Y-axis')
ax.set_zlabel('Z-axis')
```

```
ax.set_title("3D Scatterplot")
plt.show()
```

#Example scatterplots:





The original data of the nanotags is presented in the following table.

	Si	F	Br	I	code
3	2079	1.536797	0.102557	0.594991	000
11	1941	0.086038	0.03325	0.106314	000
18	591	0.169205	0.100464	0.199369	000
19	109	0.449541	0.084285	0.231398	000
22	884	0.127828	0.053069	0.112887	000
30	206	0.252427	0.090409	0.682067	000
37	961	0.123829	0.048984	0.126264	000

Formatted: Font: Bold
Formatted: Centered
Formatted Table
Formatted: Font: Bold
Formatted: Centered
Formatted: Font: Bold
Formatted: Centered
Formatted: Font: Bold
Formatted: Centered
Formatted: Font: Bold
Formatted: Centered
Formatted: Font: Bold
Formatted: Centered
Formatted: Font: Bold
Formatted: Centered
Formatted: Font: Bold
Formatted: Centered

46	499	0.202405	0.085395	0.234276	000
52	650	0.176923	0.156729	0.242843	000
73	772	0.17487	0.087157	0.231304	000
76	358	0.198324	0.136553	0.381357	000
78	2514	0.1428	0.114659	0.150025	000
81	376	0.284574	0.156786	0.47437	000
82	5011	0.070245	0.044158	0.095712	000
84	693	0.145743	0.08915	0.243408	000
86	1732	0.095843	0.052167	0.132567	000
92	194	0.603093	0.35538	0.787929	000
10	840	0.165476	0.072808	0.594025	001
15	980	0.084694	0.04501	0.510281	001
17	2281	0.088996	0.025107	0.56783	001
23	3512	0.086845	0.031786	0.346812	001
24	1548	0.082687	0.033058	0.517577	001
27	2997	0.143143	0.024318	0.622598	001
32	784	0.156888	0.055313	0.709943	001
45	622	0.090032	0.049162	0.606093	001
49	783	0.243934	0.089268	0.714592	001
57	2326	0.072657	0.029049	0.592738	001
59	1083	0.11819	0.043503	0.733092	001
60	1029	0.124393	0.050823	0.681062	001
66	1087	0.081877	0.047878	0.62791	001
67	1806	0.095792	0.048688	0.706362	001
74	778	0.169666	0.09058	0.954735	001
79	3158	0.064915	0.039588	0.658682	001
83	1506	0.120186	0.087811	0.795612	001
85	2553	0.135919	0.056331	0.683865	001
91	880	0.235227	0.147115	0.830637	001
97	2636	0.158953	0.080608	0.796936	001
98	1348	0.341246	0.137476	0.983541	001
6	1021	0.125367	0.11992	0.166844	010
12	2736	0.107822	0.033422	0.121284	010
31	1585	0.071924	0.078742	0.068821	010
38	964	0.078838	0.09916	0.106264	010
41	526	0.095057	0.096849	0.120862	010
43	440	0.193182	0.257072	0.305568	010
47	4664	0.063679	0.079883	0.059609	010
48	589	0.147708	0.165671	0.148352	010
50	840	0.080952	0.102659	0.102767	010
58	679	0.145803	0.12688	0.137454	010
69	956	0.118201	0.144645	0.151529	010
71	698	0.137536	0.163373	0.152847	010

72	2124	0.09275	0.11965	0.132112	010
75	987	0.139818	0.151239	0.142601	010
0	1572	2.052163	0.059322	0.16504	100
2	2523	1.968688	0.03608	0.12359	100
4	4815	2.338318	0.092799	0.121559	100
14	1355	1.884133	0.042984	0.123243	100
16	692	1.926301	0.087831	0.202561	100
28	1343	1.817573	0.033877	0.084635	100
36	410	1.660976	0.047544	0.430338	100
44	1089	1.623508	0.029457	0.1068	100
53	401	2.259352	0.081183	0.192514	100
54	801	2.042447	0.040796	0.149235	100
56	2312	2.115484	0.021886	0.074287	100
70	734	2.286104	0.072564	0.159689	100
88	1510	1.956954	0.110311	0.154286	100
90	2893	2.080539	0.062404	0.126287	100
7	144	3.979167	0.158327	1.0138	101
9	2132	1.781426	0.038681	0.580807	101
29	1470	1.692517	0.056774	0.574689	101
62	1311	2.194508	0.054065	0.766149	101
64	539	2.879406	0.074489	0.825732	101
68	1306	2.712864	0.075494	0.751369	101
80	1058	3.482987	0.087411	0.830915	101
94	752	2.444149	0.069528	0.989448	101
96	1231	2.523152	0.11869	0.840208	101
99	1788	2.657718	0.116781	0.863981	101
13	1849	1.943213	0.107398	0.130722	110
26	1739	2.326049	0.086003	0.123241	110
33	2498	1.933947	0.124291	0.098111	110
34	3077	2.603835	0.095006	0.085698	110
55	1576	2.802665	0.166823	0.143968	110
87	1726	3.289687	0.168146	0.173849	110
89	231	3.121212	0.580427	0.861454	110
1	1048	3.298664	0.30782	0.684914	111
5	24992	0.74912	0.058694	0.244284	111
8	27277	0.44367	0.040386	0.296691	111
20	1157	1.000864	0.577464	0.486388	111
21	5384	1.831538	0.223721	0.491724	111
25	1391	2.038821	0.283488	0.686907	111
35	2388	2.012982	0.255362	0.536272	111
39	3376	2.009182	0.14303	0.578873	111
40	3356	1.906138	0.224328	0.516557	111
42	1192	2.044463	0.241482	0.467393	111

51	1371	3.110868	0.373864	0.742043	111
61	2366	2.393914	0.275988	0.640108	111
65	3297	0.764331	0.194621	0.308549	111
77	1783	0.86203	0.179516	0.331535	111
93	1447	3.492053	0.37679	0.701297	111
95	5200	1.918077	0.270326	0.588173	111

Formatted: Font: Bold

Formatted: Centered

Formatted: Font: Bold

Formatted: Centered

Formatted: Font: Bold

Formatted: Centered

Formatted: Font: Bold

Formatted: Centered

Formatted: Font: Bold

Formatted: Centered

Formatted: Font: Bold

Formatted: Centered