Important Dataset and splite Train, Testing and Target

In [40]:

```
import numpy as np
import pandas as pd
from sklearn.preprocessing import LabelEncoder
#from xgboost.sklearn import XGBClassifier
np.random.seed(0)

#Loading data
train = pd.read_csv('train_users_2.csv', encoding = "ISO-8859-1")
print(train.shape)
test = pd.read_csv('test_users.csv', encoding = "ISO-8859-1")
labels = train['country_destination'].values
train = train.drop(['country_destination'], axis=1)
train = train.drop(['id'],axis=1)
test = test.drop(['id'],axis=1)

print("Training Data: \n", train)
print("Testing Data: \n", test)
print("Target Variable: \n",labels)

print("Training Data shape:\n",train.shape )
print("Testing Data shape:\n",test.shape )
print("Target Data shape:\n",labels.shape )
```

```
(213451, 16)
('Training Data: \n',        date_account_created  timestamp_first_act
ive date_first_booking  \
0                 2010-06-28           20090319043255                NaN
1                 2011-05-25           20090523174809                NaN
2                 2010-09-28           20090609231247         2010-08-02
3                 2011-12-05           20091031060129         2012-09-08
4                 2010-09-14           20091208061105         2010-02-18
5                 2010-01-01           20100101215619         2010-01-02
6                 2010-01-02           20100102012558         2010-01-05
7                 2010-01-03           20100103191905         2010-01-13
8                 2010-01-04           20100104004211         2010-07-29
9                 2010-01-04           20100104023758         2010-01-04
10                2010-01-04           20100104194251         2010-01-06
11                2010-01-05           20100105051812                NaN
12                2010-01-05           20100105060859         2010-01-18
13                2010-01-05           20100105083259                NaN
14                2010-01-07           20100107055820                NaN
15                2010-01-07           20100107204555         2010-01-08
16                2010-01-07           20100107215135                NaN
```

Preprocessing on Training and Testing data by using same

Preprocessing on Training and Testing data by using same standard

In [41]:

```
# Training + Testing
full = pd.concat((train, test), axis=0, ignore_index=True)
print(full.shape)
print(full)
headers= (list(full))
```

```
(275547, 14)
       date_account_created  timestamp_first_active date_first_booking
\
0                2010-06-28          20090319043255                NaN
1                2011-05-25          20090523174809                NaN
2                2010-09-28          20090609231247         2010-08-02
3                2011-12-05          20091031060129         2012-09-08
4                2010-09-14          20091208061105         2010-02-18
5                2010-01-01          20100101215619         2010-01-02
6                2010-01-02          20100102012558         2010-01-05
7                2010-01-03          20100103191905         2010-01-13
8                2010-01-04          20100104004211         2010-07-29
9                2010-01-04          20100104023758         2010-01-04
10               2010-01-04          20100104194251         2010-01-06
11               2010-01-05          20100105051812                NaN
12               2010-01-05          20100105060859         2010-01-18
13               2010-01-05          20100105083259                NaN
14               2010-01-07          20100107055820                NaN
15               2010-01-07          20100107204555         2010-01-08
```

In [42]:

```
full.isnull().sum()
```

Out[42]:

```
date_account_created           0
timestamp_first_active         0
date_first_booking         186639
gender                         0
age                       116866
signup_method                  0
signup_flow                    0
language                       0
affiliate_channel              0
affiliate_provider             0
first_affiliate_tracked     6085
signup_app                     0
first_device_type              0
first_browser                  0
dtype: int64
```

In [43]:

```
## Fill Missing values
full['age'].fillna(-1, inplace=True)
full['first_affiliate_tracked'].fillna('NaN',inplace=True)
full['date_first_booking'].fillna('0-0-0',inplace=True)
full.isnull().sum()
print(full)
```

```
       date_account_created  timestamp_first_active date_first_booking
\
0                2010-06-28          20090319043255              0-0-0
1                2011-05-25          20090523174809              0-0-0
2                2010-09-28          20090609231247         2010-08-02
3                2011-12-05          20091031060129         2012-09-08
4                2010-09-14          20091208061105         2010-02-18
5                2010-01-01          20100101215619         2010-01-02
6                2010-01-02          20100102012558         2010-01-05
7                2010-01-03          20100103191905         2010-01-13
8                2010-01-04          20100104004211         2010-07-29
9                2010-01-04          20100104023758         2010-01-04
10               2010-01-04          20100104194251         2010-01-06
11               2010-01-05          20100105051812              0-0-0
12               2010-01-05          20100105060859         2010-01-18
13               2010-01-05          20100105083259              0-0-0
14               2010-01-07          20100107055820              0-0-0
15               2010-01-07          20100107204555         2010-01-08
16               2010-01-07          20100107215125              0-0-0
17               2010-01-07          20100107224635         2010-01-09
```

In [44]:

```
#  timestamp_first_active
tfa = np.vstack(full.timestamp_first_active.astype(str).apply(lambda x: list(map(int
full['tfa_year'] = tfa[:,0]
full['tfa_month'] = tfa[:,1]
full['tfa_day'] = tfa[:,2]
full = full.drop(['timestamp_first_active'], axis=1)

#Age
agee = full.age.values
full['age'] = np.where(np.logical_or(agee<14, agee>90), -1, agee)

print(full)
```

```
       date_account_created date_first_booking     gender   age signup
_method  \
0                2010-06-28              0-0-0  -unknown-  -1.0      f
acebook
1                2011-05-25              0-0-0       MALE  38.0      f
acebook
2                2010-09-28         2010-08-02     FEMALE  56.0
basic
3                2011-12-05         2012-09-08     FEMALE  42.0      f
acebook
4                2010-09-14         2010-02-18  -unknown-  41.0
basic
5                2010-01-01         2010-01-02  -unknown-  -1.0
basic
6                2010-01-02         2010-01-05     FEMALE  46.0
basic
7                2010-01-03         2010-01-13     FEMALE  47.0
basic
8                2010-01-04         2010-07-29     FEMALE  50.0
basic
```

In [45]:

```
# LabelBinarizer & One-hot Encoding
from sklearn import preprocessing
lb = preprocessing.LabelBinarizer()
enc = preprocessing.OneHotEncoder()



## gender (Unknown, Female, male, Other)
temp_gender = full.gender
print(type(temp_gender))
cate1= lb.fit_transform(temp_gender)
print(cate1.shape)
names1 = ['Unknown', 'Female', 'Male', 'Other']
```

```
cate1 = pd.DataFrame(cate1, index=range(275547), columns=names1)
# print(cate1)       # [unknown female male Other]


##signup_method (basic,facebook)
temp_sm = full.signup_method
print(type(temp_sm))
cate2= lb.fit_transform(temp_sm)
print(cate2.shape)
cate2= cate2[:,0:2]
names2 = ['basic','facebook']
cate2 = pd.DataFrame(cate2, index=range(275547), columns=names2)
# print(cate2)       # ['basic','facebook']


## language (ca,cs,da,de,el,en,es,fi,fr,hr,hu,id,it,ja,ko,nl,no,pl,pt,ru,\
#           sv,th,tr,zh)
temp_lg = full.language
print(type(temp_lg))
cate3= lb.fit_transform(temp_lg)
print(cate3.shape)
cate3= cate3[:,0:24]
names3 = ['ca','cs','da','de','el','en','es','fi','fr','hr','hu',\
          'id','it','ja','ko','nl','no','pl','pt','ru','sv','th','tr','zh']
names3 = sorted(names3)
cate3 = pd.DataFrame(cate3, index=range(275547), columns=names3)
# print(cate3)
```

```
## affiliate_channel (api,content,direct,other,remarketing,sem-brand,sem-non-brand,s
temp_ac = full.affiliate_channel
print(type(temp_ac))
cate4= lb.fit_transform(temp_ac)
print(cate4.shape)
cate4= cate4[:,0:8]
names4 = ['api','content','direct','other','remarketing','sem-brand','sem-non-brand
names4 = sorted(names4)
cate4 = pd.DataFrame(cate4, index=range(275547), columns=names4)
# print(cate4)

## affiliate_provider
temp_ap = full.affiliate_provider
print(type(temp_ap))
cate5= lb.fit_transform(temp_ap)
print(cate5.shape)
cate5= cate5[:,0:16]
names5 = ['baidu','bing','craigslist','direct','email-marketing',\
          'facebook','facebook-open-graph','google','gsp','meetup',\
         'naver','other','padmapper','vast','wayn','yahoo']
names5 = sorted(names5)
cate5 = pd.DataFrame(cate5, index=range(275547), columns=names5)
# print(cate5)


## first_affiliate_tracked
```

```
temp_fat = full.first_affiliate_tracked
print(type(temp_fat))
cate6= lb.fit_transform(temp_fat)
print(cate6.shape)
cate6= cate6[:,0:7]
names6 = ['linked','local ops','marketing','omg','product',\
          'tracked-other','untracked']
names6 = sorted(names6)
cate6 = pd.DataFrame(cate6, index=range(275547), columns=names6)
# print(cate6)

## signup_app
temp_sa = full.signup_app
print(type(temp_sa))
cate7= lb.fit_transform(temp_sa)
print(cate7.shape)
cate7= cate7[:,0:4]
names7 = ['Android','iOS','Moweb','Web']
names7 = sorted(names7)
cate7 = pd.DataFrame(cate7, index=range(275547), columns=names7)
# print(cate7)


## first_device_type
temp_fdt = full.first_device_type
print(type(temp_fdt))
cate8= lb.fit_transform(temp_fdt)
print(cate8.shape)
cate8= cate8[:,0:9]
names8 = ['Android Phone','Android Tablet','Desktop(Other)','iPad',\
          'iPhone','Mac Desktop','Other/Unknown','SmartPhone(Other)','Windows Desktop
names8 = sorted(names8)
cate8 = pd.DataFrame(cate8, index=range(275547), columns=names8)
# print(cate8)


## first_browser
temp_fb = full.first_browser
print(type(temp_fb))
cate9= lb.fit_transform(temp_fb)
print(cate9.shape)
cate9= cate9[:,0:25]
names9 = ['Unknown','Android Browser','AOL Explo','Apple Mail','Arora',\
          'BalckBerry Browser','Camino','Chrome','Chrome Mobile','Chromium',\
           'Firfox','IceWeasel','IE','IE Mobile','Iron','Mobile Firfox','Mibile Safar
           'Opera','PP Web Browser','RockMelt','Safari','SeaMonkey','Silk',\
           'Sogou Explorer','TenFourFox']
names9 = sorted(names9)
cate9 = pd.DataFrame(cate9, index=range(275547), columns=names9)
# print(cate9)

# date_account_created
temp_fb = full.date_account_created
cate10 = np.vstack(temp_fb.astype(str).apply(lambda x: list(map(int, x.split('-')))
```

```
names10 = ['year_ac','month_ac','day_ac']
names10 = sorted(names10)
cate10 = pd.DataFrame(cate10, index=range(275547), columns=names10)
# print(cate10)


## date_first_booking
temp_dfb = full.date_first_booking
cate11 = np.vstack(temp_dfb.astype(str).apply(lambda x: list(map(int, x.split('-'))))
names11 = ['year_dfb','month_dfb','day_dfb']
names11 = sorted(names11)
cate11 = pd.DataFrame(cate11, index=range(275547), columns=names11)
# print(cate11)

full_1 = full.drop(['gender','date_account_created','date_first_booking','signup_met
                    'affiliate_provider','first_affiliate_tracked','signup_app',\
                    'first_device_type','first_browser',],axis = 1)

full_data = np.concatenate((full_1,cate1,cate2,cate3,cate4,cate5,cate6,cate7,cate8,c
print(full_data.shape)
print(full_data)
```

```
<class 'pandas.core.series.Series'>
(275547, 4)
<class 'pandas.core.series.Series'>
(275547, 4)
<class 'pandas.core.series.Series'>
(275547, 26)
<class 'pandas.core.series.Series'>
(275547, 8)
<class 'pandas.core.series.Series'>
(275547, 18)
<class 'pandas.core.series.Series'>
(275547, 8)
<class 'pandas.core.series.Series'>
(275547, 4)
<class 'pandas.core.series.Series'>
(275547, 9)

<class 'pandas.core.series.Series'>
(275547, 55)
(275547, 110)
```

Train and Testing

In [46]:

```
print(full_data)
```

```
[[ -1.00000000e+00   0.00000000e+00   2.00900000e+03 ...,   0.00000000
e+00
    0.00000000e+00   0.00000000e+00]
 [  3.80000000e+01   0.00000000e+00   2.00900000e+03 ...,   0.00000000
e+00
    0.00000000e+00   0.00000000e+00]
 [  5.60000000e+01   3.00000000e+00   2.00900000e+03 ...,   2.01000000
e+03
    8.00000000e+00   2.00000000e+00]
 ...,
 [ -1.00000000e+00   0.00000000e+00   2.01400000e+03 ...,   0.00000000
e+00
    0.00000000e+00   0.00000000e+00]
 [ -1.00000000e+00   0.00000000e+00   2.01400000e+03 ...,   0.00000000
e+00
    0.00000000e+00   0.00000000e+00]
 [  4.90000000e+01   0.00000000e+00   2.01400000e+03 ...,   0.00000000
e+00
    0.00000000e+00   0.00000000e+00]]
```

In [47]:

```
# Train and Testing
training = full_data[0:213451]
testing = full_data[213451:]
print(training.shape)
print(testing.shape)
cv =  labels.reshape((len(labels), 1))

# np.savetxt('training.csv',training, delimiter=',',fmt='%d')
# np.savetxt('testing.csv',testing, delimiter=',',fmt='%d')
```

```
(213451, 110)
(62096, 110)
```

Classification

First Try with ALL features

Naive Bayes

In [48]:

```
# Naive Bayes Full Model
import pandas as pd
from sklearn import metrics
from sklearn.naive_bayes import GaussianNB
import numpy as np
from sklearn.cross_validation import KFold, cross_val_score
from sklearn.naive_bayes import MultinomialNB
NB = GaussianNB()
NB.fit(training, cv)


#Model
# print("Probability of the classes: ", NB.class_prior_)
# print("Mean of each feature per class:\n", NB.theta_)
# print("Variance of each feature per class:\n", NB.sigma_)

#predict the class for each data point
predicted = NB.predict(training)
print("Predictions:\n",np.array([predicted]).T)
print("Accuracy of the model: ",NB.score(training,cv))

predictedtest = NB.predict(testing)
print("test Predictions:\n",np.array([predictedtest]).T)

# np.savetxt('testnb.csv',output.reshape(-1,output.shape[-1]), delimiter=',',fmt='%

model = GaussianNB()
kf = KFold(len(cv), n_folds=5)
scores = cross_val_score(model, training, cv, cv=kf)
print("MSE of every fold in 5 fold cross validation: ", abs(scores))
print("Mean of the 5 fold cross-validation: %0.2f" % abs(scores.mean()))
```

```
('Predictions:\n', array([[u'NDF'],
       [u'NDF'],
       [u'GB'],
       ...,
       [u'NDF'],
       [u'NDF'],
       [u'NDF']],
      dtype='<U5'))
('Accuracy of the model: ', 0.63450159521388982)
('test Predictions:\n', array([[u'NDF'],
       [u'NDF'],
       [u'NDF'],
       ...,
       [u'NDF'],
       [u'NDF'],
       [u'NDF']],
      dtype='<U5'))
('MSE of every fold in 5 fold cross validation: ', array([ 0.55793961,
0.61393769,  0.62368236,  0.6662216 ,  0.62574373]))
Mean of the 5 fold cross-validation: 0.62
```

Decision Tree

In [49]:

```
# Decision Tree Full Model
from sklearn.tree import DecisionTreeClassifier
from sklearn import tree
from sklearn import preprocessing
from sklearn.cross_validation import KFold, cross_val_score
import numpy

# Create linear regression object
DT = DecisionTreeClassifier(criterion="entropy", min_samples_leaf = 4)
# Train the model using the training sets
DT.fit(training, cv)

#predict the class for each data point
predicted = DT.predict(training)
print("Predictions: \n", np.array([predicted]).T)
print("Accuracy training score: \n", DT.score(training,cv))
print("Feature importance: ", DT.feature_importances_)

testpredict = DT.predict(testing)
print("testpredict: \n", np.array([testpredict]).T)
output = np.array([testpredict]).T
# np.savetxt('testdt.csv',output.reshape(-1,output.shape[-1]), delimiter=',',fmt='%s
```

```
# 5-folder Cross-Validation DT
model = DecisionTreeClassifier()
kf = KFold(len(cv), n_folds=5)
CV = cv
scores = cross_val_score(model, training, cv, cv=kf)
print("Full model MSE of every fold in 5 fold cross validation: ", abs(scores))
print("Full model Mean of the 5 fold cross-validation: %0.2f" % abs(scores.mean()))
```

```
('Predictions: \n', array([[u'NDF'],
       [u'NDF'],
       [u'US'],
       ...,
       [u'NDF'],
       [u'NDF'],
       [u'NDF']], dtype=object))
('Accuracy training score: \n', 0.8984357065556029)
('Feature importance: ', array([  4.54477752e-02,   4.76398559e-03,
2.32709437e-03,
         1.04670824e-02,   2.84295790e-02,   3.41174992e-03,
         5.29911275e-03,   3.99832361e-03,   0.00000000e+00,
         3.91799944e-03,   4.17886072e-03,   0.00000000e+00,
         0.00000000e+00,   0.00000000e+00,   0.00000000e+00,
         6.56619048e-05,   0.00000000e+00,   2.50207106e-04,
         6.01795510e-05,   0.00000000e+00,   2.02314156e-04,
         0.00000000e+00,   0.00000000e+00,   0.00000000e+00,
         0.00000000e+00,   0.00000000e+00,   0.00000000e+00,
```

```
       0.00000000e+00,   0.00000000e+00,   0.00000000e+00,
       0.00000000e+00,   0.00000000e+00,   0.00000000e+00,
       0.00000000e+00,   0.00000000e+00,   1.39527152e-04,
       2.16983466e-04,   1.81205525e-03,   7.07746659e-04,
       1.20523201e-04,   1.99527599e-03,   1.31878211e-03,
       4.52093958e-04,   0.00000000e+00,   1.11872904e-04,
       2.47845538e-04,   0.00000000e+00,   1.75785925e-03,
       0.00000000e+00,   1.01948908e-04,   3.01203234e-05,
       2.30180307e-03,   0.00000000e+00,   0.00000000e+00,
       0.00000000e+00,   3.79595535e-04,   0.00000000e+00,
       5.57497630e-05,   0.00000000e+00,   2.90471765e-04,
       7.00222754e-03,   0.00000000e+00,   0.00000000e+00,
       3.40556851e-03,   9.28888691e-05,   1.93212934e-04,
       2.05065403e-04,   3.38354670e-04,   4.72581061e-04,
       6.50195978e-04,   8.19069607e-05,   2.56449015e-05,
       1.53339969e-05,   6.18463744e-03,   3.61629952e-04,
       0.00000000e+00,   5.35929925e-03,   1.43112521e-03,
       8.30760085e-04,   7.88743512e-04,   0.00000000e+00,
       0.00000000e+00,   0.00000000e+00,   0.00000000e+00,
       0.00000000e+00,   0.00000000e+00,   0.00000000e+00,
       9.27754267e-03,   7.33054443e-06,   0.00000000e+00,
       0.00000000e+00,   0.00000000e+00,   0.00000000e+00,
       0.00000000e+00,   0.00000000e+00,   0.00000000e+00,
       6.46416736e-03,   0.00000000e+00,   0.00000000e+00,
       0.00000000e+00,   0.00000000e+00,   3.00668997e-03,
       0.00000000e+00,   0.00000000e+00,   1.87969285e-03,
       1.05493002e-02,   2.89090196e-02,   3.55583452e-03,
         1.66867160e-02,   7.67364349e-01]))
('testpredict: \n', array([[u'NDF'],
       [u'NDF'],
       [u'NDF'],
       ...,
       [u'NDF'],
       [u'NDF'],
       [u'NDF']], dtype=object))
('Full model MSE of every fold in 5 fold cross validation: ', array([
0.7225879 ,  0.77034434,  0.80016397,  0.7864371 ,  0.78955259]))
Full model Mean of the 5 fold cross-validation: 0.77
```

In [50]:

```
## Random Forest
import pandas as pd
from sklearn import metrics
from sklearn.ensemble import RandomForestClassifier
from sklearn.cross_validation import KFold, cross_val_score
import numpy as np

clf = RandomForestClassifier(n_estimators=4)
clf.fit(training, cv)

predicted = clf.predict(training)
print("Predictions: \n", np.array([predicted]).T)
print("Accuracy training score: \n", clf.score(training,cv))

testpredict = clf.predict(testing)
print("testpredict: \n", np.array([testpredict]).T)
output = np.array([testpredict]).T
# np.savetxt('testrf.csv',output.reshape(-1,output.shape[-1]), delimiter=',',fmt='%

# Calculating 5 fold cross validation results RF
model = RandomForestClassifier()
kf = KFold(len(cv), n_folds=5)
scores = cross_val_score(model, training, cv, cv=kf)
print("MSE of every fold in 5 fold cross validation: ", abs(scores))
print("Mean of the 5 fold cross-validation: %0.2f" % abs(scores.mean()))
```

```
/anaconda/lib/python2.7/site-packages/ipykernel/__main__.py:9: DataCon
versionWarning: A column-vector y was passed when a 1d array was expec
ted. Please change the shape of y to (n_samples,), for example using r
avel().

('Predictions: \n', array([[u'NDF'],
       [u'NDF'],
       [u'US'],
       ...,
       [u'NDF'],
       [u'NDF'],
       [u'NDF']], dtype=object))
('Accuracy training score: \n', 0.97079891872139268)
('testpredict: \n', array([[u'NDF'],
       [u'NDF'],
       [u'NDF'],
       ...,
       [u'NDF'],
       [u'NDF'],
       [u'NDF']], dtype=object))
('MSE of every fold in 5 fold cross validation: ', array([ 0.83401654,
0.85586788,  0.88163504,  0.88051066,  0.87217147]))
Mean of the 5 fold cross-validation: 0.86
```

KNN

KNN

In [51]:

```
import pandas as pd
import numpy as np
from sklearn import metrics
from sklearn.neighbors import KNeighborsClassifier
from sklearn.cross_validation import KFold, cross_val_score

# KNN full model
# Create a KNN object
# KNN = KNeighborsClassifier(n_neighbors=3)
# Train the model using the training sets
# KNN.fit(training, cv)
# predict the class for each data point
# predicted = KNN.predict(training)
# print("Predictions: \n", np.array([predicted]).T)
# print("Accuracy score for the model: \n", KNN.score(training,cv))
# predictedtest = KNN.predict(testing)
# print("testPredictions: \n", np.array([predictedtest]).T)
```

Second Round: Entropy Based Feature Reduction

In [52]:

```
Reduced1_data = np.concatenate((full_1,cate1,cate2,cate4,cate5,cate6,cate7,cate8,cat
print(Reduced1_data.shape)
```

```
(275547, 61)
```

In [53]:

```
# Train and Testing
training1 = Reduced1_data[0:213451]
testing1 = Reduced1_data[213451:]
print(training1.shape)
print(testing1.shape)
cv =  labels.reshape((len(labels), 1))
```

```
(213451, 61)
(62096, 61)
```

In [54]:

```
## Naive Bayes
# Naive Bayes Full Model
import pandas as pd
from sklearn import metrics
from sklearn.naive_bayes import GaussianNB
import numpy as np
from sklearn.cross_validation import KFold, cross_val_score
```

```
from sklearn.naive_bayes import MultinomialNB
NB = GaussianNB()
NB.fit(training1, cv)


#Model
# print("Probability of the classes: ", NB.class_prior_)
# print("Mean of each feature per class:\n", NB.theta_)
# print("Variance of each feature per class:\n", NB.sigma_)

#predict the class for each data point
predicted = NB.predict(training1)
print("Predictions:\n",np.array([predicted]).T)
print("Accuracy of the model: ",NB.score(training1,cv))

predictedtest = NB.predict(testing1)
print("test Predictions:\n",np.array([predictedtest]).T)

# np.savetxt('testnb1.csv',output.reshape(-1,output.shape[-1]), delimiter=',',fmt='

# Calculating 5 fold cross validation results NB
model = RandomForestClassifier()
kf = KFold(len(cv), n_folds=5)
scores = cross_val_score(model, training1, cv, cv=kf)
print("MSE of every fold in 5 fold cross validation: ", abs(scores))
print("Mean of the 5 fold cross-validation: %0.2f" % abs(scores.mean()))
```

```
('Predictions:\n', array([[u'NDF'],
       [u'NDF'],
       [u'FR'],
       ...,
       [u'NDF'],
       [u'NDF'],
       [u'NDF']],
      dtype='<U5'))
('Accuracy of the model: ', 0.63962689329166877)
('test Predictions:\n', array([[u'NDF'],
       [u'NDF'],
       [u'NDF'],
       ...,
       [u'NDF'],
       [u'NDF'],
       [u'NDF']],
      dtype='<U5'))
('MSE of every fold in 5 fold cross validation: ', array([ 0.82891007,
0.85546967,  0.8795971 ,  0.87882408,  0.87088311]))
Mean of the 5 fold cross-validation: 0.86
```

In [55]:

```
# Decision Tree Full Model
from sklearn.tree import DecisionTreeClassifier
from sklearn import tree
from sklearn import preprocessing
from sklearn.cross_validation import KFold, cross_val_score
```

```
from sklearn.cross_validation import KFold, cross_val_score
import numpy

# Create linear regression object
DT = DecisionTreeClassifier(criterion="entropy", min_samples_leaf = 4)
# Train the model using the training sets
DT.fit(training1, cv)

#predict the class for each data point
predicted = DT.predict(training1)
print("Predictions: \n", np.array([predicted]).T)
print("Accuracy training score: \n", DT.score(training1,cv))
print("Feature importance: ", DT.feature_importances_)

testpredict = DT.predict(testing1)
print("testpredict: \n", np.array([testpredict]).T)
output = np.array([testpredict]).T
# np.savetxt('testdt1.csv',output.reshape(-1,output.shape[-1]), delimiter=',',fmt='
model = DecisionTreeClassifier()
kf = KFold(len(cv), n_folds=5)
CV = cv
scores = cross_val_score(model, training1, cv, cv=kf)
print("Full model MSE of every fold in 5 fold cross validation: ", abs(scores))
print("Full model Mean of the 5 fold cross-validation: %0.2f" % abs(scores.mean()))
```

```
('Predictions: \n', array([[u'NDF'],
       [u'NDF'],
       [u'US'],
       ...,
       [u'NDF'],
       [u'NDF'],
       [u'NDF']], dtype=object))
('Accuracy training score: \n', 0.896931848527296 68)
('Feature importance: ', array([  4.66743762e-02,   5.38797939e-03,
2.20283502e-03,
        1.10791988e-02,   3.00897750e-02,   3.75837101e-03,
        4.91116304e-03,   4.20575500e-03,   0.00000000e+00,
        4.10437209e-03,   4.43138482e-03,   1.59825416e-04,
        2.87645271e-04,   1.48440206e-03,   6.93575046e-04,
        1.21007192e-04,   2.22206085e-03,   1.22414136e-03,
        8.46864235e-04,   0.00000000e+00,   1.12322157e-04,
        3.14688271e-04,   0.00000000e+00,   1.78472561e-03,
        0.00000000e+00,   1.28960161e-04,   3.02412790e-05,
        2.24108856e-03,   0.00000000e+00,   0.00000000e+00,
        0.00000000e+00,   3.49452036e-04,   0.00000000e+00,
        9.67268894e-05,   0.00000000e+00,   2.53359726e-04,
        7.10256959e-03,   0.00000000e+00,   1.55544925e-05,
        3.68624503e-03,   9.32618873e-05,   2.09231477e-04,
        2.23319713e-04,   2.94048534e-04,   5.66796539e-04,
        6.05107674e-04,   1.12101846e-04,   7.58132033e-05,
        3.05586141e-05,   7.48489783e-03,   3.93117348e-04,
        0.00000000e+00,   6.27912852e-03,   1.60802276e-03,
```

```
         8.79844698e-04,   2.08445955e-03,   1.09039583e-02,
         3.07443933e-02,   3.64849318e-03,   1.85911670e-02,
         7.75171613e-01]))
('testpredict: \n', array([[u'NDF'],
       [u'NDF'],
       [u'NDF'],
       ...,
       [u'NDF'],
       [u'NDF'],
       [u'NDF']], dtype=object))
('Full model MSE of every fold in 5 fold cross validation: ', array([
0.72483662,  0.7734364 ,  0.80545795,  0.78777231,  0.79997658]))
Full model Mean of the 5 fold cross-validation: 0.78
```

In [56]:

```
## Random Forest
import pandas as pd
from sklearn import metrics
from sklearn.ensemble import RandomForestClassifier
from sklearn.cross_validation import KFold, cross_val_score
import numpy as np

clf = RandomForestClassifier(n_estimators=4)
clf.fit(training1, cv)

predicted = clf.predict(training1)
print("Predictions: \n", np.array([predicted]).T)
print("Accuracy training score: \n", clf.score(training1,cv))

testpredict = clf.predict(testing1)
print("testpredict: \n", np.array([testpredict]).T)
output = np.array([testpredict]).T
# np.savetxt('testrf1.csv',output.reshape(-1,output.shape[-1]), delimiter=',',fmt='



# Calculating 5 fold cross validation results NB
model = RandomForestClassifier()
kf = KFold(len(cv), n_folds=5)
scores = cross_val_score(model, training1, cv, cv=kf)
print("MSE of every fold in 5 fold cross validation: ", abs(scores))
print("Mean of the 5 fold cross-validation: %0.2f" % abs(scores.mean()))
```

```
/anaconda/lib/python2.7/site-packages/ipykernel/__main__.py:9: DataCon
versionWarning: A column-vector y was passed when a 1d array was expec
ted. Please change the shape of y to (n_samples,), for example using r
avel().

('Predictions: \n', array([[u'NDF'],
       [u'NDF'],
       [u'FR'],
       ...,
       [u'NDF'],
       [u'NDF'],
       [u'NDF']], dtype=object))
('Accuracy training score: \n', 0.968081667455294 23)
('testpredict: \n', array([[u'NDF'],
       [u'NDF'],
       [u'NDF'],
       ...,
       [u'NDF'],
       [u'NDF'],
       [u'NDF']], dtype=object))
('MSE of every fold in 5 fold cross validation: ', array([ 0.83357148,
0.85610213,  0.87901148,  0.88037011,  0.86729913]))
Mean of the 5 fold cross-validation: 0.86
```

Third Round: NDCG

In [82]:

```
from xgboost.sklearn import XGBClassifier

xgb = XGBClassifier(max_depth=6, learning_rate=0.3, n_estimators=25,
                    objective='multi:softprob', subsample=0.5, colsample_bytree=0.5,
xgb.fit(training, cv)

predicted = xgb.predict(training)
print("Predictions: \n", np.array([predicted]).T)
print("Accuracy training score: \n", xgb.score(training,cv))

testpredict = xgb.predict(testing)
print("testpredict: \n", np.array([testpredict]).T)

output = np.array([testpredict]).T
np.savetxt('xgboost.csv',output.reshape(-1,output.shape[-1]), delimiter=',',fmt='%s
```

```
('Predictions: \n', array([[u'NDF'],
       [u'NDF'],
       [u'US'],
       ...,
       [u'NDF'],
       [u'NDF'],
       [u'NDF']], dtype=object))
('Accuracy training score: \n', 0.875779452895512 29)
('testpredict: \n', array([[u'NDF'],
       [u'NDF'],
       [u'NDF'],
       ...,
       [u'NDF'],
       [u'NDF'],
       [u'NDF']], dtype=object))
```

In [ ]:

In [85]:

```
from sklearn.cross_validation import KFold, cross_val_score
# Calculating 5 fold cross validation results XGB
model = XGBClassifier()
kf = KFold(len(cv), n_folds=5)
scores = cross_val_score(model, training, cv, cv=kf)
print("MSE of every fold in 5 fold cross validation: ", abs(scores))
print("Mean of the 5 fold cross-validation: %0.2f" % abs(scores.mean()))
```

```
('MSE of every fold in 5 fold cross validation: ', array([ 0.84875032,
0.86741626,  0.88821738,  0.89114547,  0.88297025]))
Mean of the 5 fold cross-validation: 0.88
```

In [83]:

```
from sklearn.preprocessing import LabelEncoder
y_pred = xgb.predict_proba(testing)
le = LabelEncoder()
y = le.fit_transform(labels)
print(y_pred)
ids = []  #list of ids
cts = []  #list of countries
for i in range(0,62096):
    cts += le.inverse_transform(np.argsort(y_pred[i])[::-1])[:5].tolist()
print(cts)

np.savetxt('xgboost.csv',cts.reshape(-1,output.shape[-1]), delimiter=',',fmt='%s')
```

```
[[  1.46176055e-04   1.03769176e-04   1.37732437e-04 ...,   1.09345157
e-04
    1.90820327e-04   1.52412598e-04]
 [  8.71254524e-05   7.51141415e-05   1.05884690e-04 ...,   8.59486172
e-05
    1.39419673e-04   1.20575416e-04]
 [  1.03901722e-04   9.11891693e-05   1.42366815e-04 ...,   9.33274496
e-05
    1.64437559e-04   1.36781411e-04]
 ...,
 [  1.19253076e-04   9.11933676e-05   1.11698144e-04 ...,   1.00267949
e-04
    1.64445140e-04   1.36940289e-04]
 [  1.05589905e-04   8.68007410e-05   1.14035196e-04 ...,   9.10792369
e-05
    1.51619693e-04   1.30344226e-04]
 [  1.54261812e-04   1.25565231e-04   1.44288395e-04 ...,   1.27154824
e-04
    2.21234703e-04   1.77722322e-04]]
```

In [ ]:

```
# np.savetxt('xgboost.csv',cts, delimiter=',',fmt='%s')
testnew = pd.read_csv('test_users.csv', encoding = "ISO-8859-1")
idtest = testnew['id']
ids = []  #list of ids
cts = []  #list of countries
for i in range(len(idtest)):
    idx = idtest[i]
    ids += [idx] * 5
    cts += le.inverse_transform(np.argsort(y_pred[i])[::-1])[:5].tolist()

sub = pd.DataFrame(np.column_stack((ids, cts)), columns=['id', 'country'])
sub.to_csv('xgsub.csv',index=False)
```

In [ ]:

```
# """Metrics to compute the model performance."""
# import numpy as np
# from sklearn.preprocessing import LabelBinarizer
# from sklearn.metrics import make_scorer

# def dcg_score(y_true, y_score, k=5):
#     """Discounted cumulative gain (DCG) at rank K.

#     Parameters
#     ----------
#     y_true : array, shape = [n_samples]
#         Ground truth (true relevance labels).
#     y_score : array, shape = [n_samples, n_classes]
#         Predicted scores.
#     k : int
#         Rank.

#     Returns
#     -------
#     score : float
#     """
#     order = np.argsort(y_score)[::-1]
#     y_true = np.take(y_true, order[:k])

#     gain = 2 ** y_true - 1

#     discounts = np.log2(np.arange(len(y_true)) + 2)
#     return np.sum(gain / discounts)


# def ndcg_score(ground_truth, predictions, k=5):
#     """Normalized discounted cumulative gain (NDCG) at rank K.

#     Normalized Discounted Cumulative Gain (NDCG) measures the performance of a
```

```
#     recommendation system based on the graded relevance of the recommended
#     entities. It varies from 0.0 to 1.0, with 1.0 representing the ideal
#     ranking of the entities.

#     Parameters
#     ----------
#     ground_truth : array, shape = [n_samples]
#         Ground truth (true labels represended as integers).
#     predictions : array, shape = [n_samples, n_classes]
#         Predicted probabilities.
#     k : int
#         Rank.

#     Returns
#     -------
#     score : float

#     Example
#     -------
#     >>> ground_truth = [1, 0, 2]
#     >>> predictions = [[0.15, 0.55, 0.2], [0.7, 0.2, 0.1], [0.06, 0.04, 0.9]]
#     >>> score = ndcg_score(ground_truth, predictions, k=2)
#     1.0
#     >>> predictions = [[0.9, 0.5, 0.8], [0.7, 0.2, 0.1], [0.06, 0.04, 0.9]]
#     >>> score = ndcg_score(ground_truth, predictions, k=2)
#     0.6666666666
#     """
#     lb = LabelBinarizer()
#     lb.fit(range(len(predictions) + 1))
#     T = lb.transform(ground_truth)

#     scores = []

#     # Iterate over each y_true and compute the DCG score
#     for y_true, y_score in zip(T, predictions):
#         actual = dcg_score(y_true, y_score, k)
#         best = dcg_score(y_true, y_true, k)
#         score = float(actual) / float(best)
#         scores.append(score)

#     return np.mean(scores)


# # NDCG Scorer function
# ndcg_scorer = make_scorer(ndcg_score, needs_proba=True, k=5)
```

In [ ]:

```
# #Classifier
# xgb = XGBClassifier(max_depth=6, learning_rate=0.3, n_estimators=25,
#                     objective='multi:softprob', subsample=0.5, colsample_bytree=0.
# xgb.fit(X, y)
# y_pred = xgb.predict_proba(X_test)

# #Taking the 5 classes with highest probabilities
# ids = []  #list of ids
# cts = []  #list of countries
# for i in range(len(id_test)):
#     idx = id_test[i]
#     ids += [idx] * 5
#     cts += le.inverse_transform(np.argsort(y_pred[i])[::-1])[:5].tolist()
```