In [1]:
```
# import warnings
# warnings.filterwarnings('ignore')
```

In [2]:
```
# import libraries
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
from scipy import sparse
%matplotlib inline

from sklearn.model_selection import RandomizedSearchCV
from scipy.stats import uniform
from sklearn.neighbors import KNeighborsClassifier
from sklearn.svm import LinearSVC
from sklearn.calibration import CalibratedClassifierCV
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from xgboost import XGBClassifier
from catboost import CatBoostClassifier
import pickle
```

Amazon Employee Access Challenge

In [3]:
```
train = pd.read_csv('data/train.csv')
test = pd.read_csv('data/test.csv')
```

In [4]:
```
train.shape
```

Out[4]:
```
(32769, 10)
```

In [5]:
```
test.shape
```

Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD

Out[5]: (58921, 10)

In [6]:
```
y_train = train['ACTION']
```

In [7]:
```
y_train.shape
```

Out[7]: (32769,)

In [8]:
```
train_data = train.drop('ACTION', axis=1)
train_data.shape
```

Out[8]: (32769, 9)

In [9]:
```
test_data = test.drop('id', axis=1)
test_data.shape
```

Out[9]: (58921, 9)

Common Variables

In [10]:
```
# define variables
random_state = 42
cv = 5
scoring = 'roc_auc'
verbose=2
```

Common functions

In [11]:
```
def save_submission(predictions, filename):
    '''
    Save predictions into csv file
    '''
    global test
    submission = pd.DataFrame()
```

Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD

```
    submission["Id"] = test["id"]
    submission["ACTION"] = predictions
    filepath = "result/sampleSubmission_"+filename
    submission.to_csv(filepath, index = False)
```

In [12]:
```
def print_graph(results, param1, param2, xlabel, ylabel, title='Plot sh
owing the ROC_AUC score for various hyper parameter values'):
    '''
    Plot the graph
    '''
    plt.plot(results[param1],results[param2]);
    plt.grid();
    plt.xlabel(xlabel);
    plt.ylabel(ylabel);
    plt.title(title);
```

In [13]:
```
def get_rf_params():
    '''
    Return dictionary of parameters for random forest
    '''
    params = {
         'n_estimators':[10,20,50,100,200,500,700,1000],
         'max_depth':[1,2,5,10,12,15,20,25],
         'max_features':[1,2,3,4,5],
         'min_samples_split':[2,5,7,10,20]
    }

    return params
```

In [14]:
```
def get_xgb_params():
    '''
    Return dictionary of parameters for xgboost
    '''
    params = {
        'n_estimators': [10,20,50,100,200,500,750,1000],
        'learning_rate': uniform(0.01, 0.6),
        'subsample': uniform(),
        'max_depth': [3, 4, 5, 6, 7, 8, 9],
```

Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD

```
        'colsample_bytree': uniform(),
        'min_child_weight': [1, 2, 3, 4]
    }

    return params
```

We will try following models

1. KNN
2. SVM
3. Logistic Regression
4. Random Forest
5. Xgboost

Build Models on the raw data

1.1 KNN with raw features

In [15]:
```
parameters={'n_neighbors':np.arange(1,100, 5)}
clf = RandomizedSearchCV(KNeighborsClassifier(n_jobs=-1),parameters,ran
dom_state=random_state,cv=cv,verbose=verbose,scoring=scoring,n_jobs=-1)
best_model = clf.fit(train_data,y_train)
```

```
Fitting 5 folds for each of 10 candidates, totalling 50 fits
```

```
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 8 concurrent work
ers.
[Parallel(n_jobs=-1)]: Done  25 tasks      | elapsed:    5.2s
[Parallel(n_jobs=-1)]: Done  50 out of  50 | elapsed:    8.8s finished
```

In [16]:
```
results = pd.DataFrame.from_dict(best_model.cv_results_)
results=results.sort_values('param_n_neighbors')
results
```

Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD

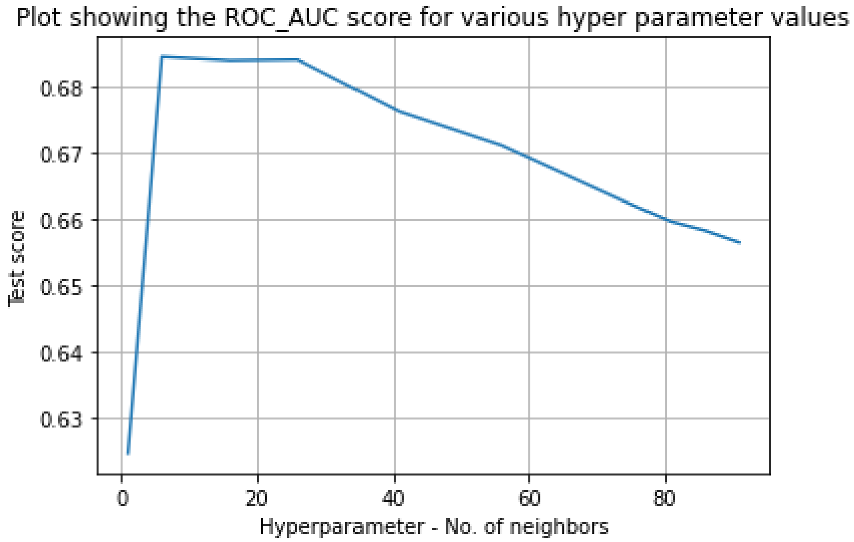
Out[16]:

	mean_fit_time	std_fit_time	mean_score_time	std_score_time	param_n_neighbors	param
0	0.083934	0.007898	0.265385	0.036920	1	{'n_neighbors
3	0.240013	0.066697	0.500296	0.106391	6	{'n_neighbors (
7	0.226630	0.185183	0.727930	0.213502	16	{'n_neighbors 1(
5	0.243259	0.052800	0.871988	0.100550	26	{'n_neighbors 2(
4	0.183503	0.043958	0.783068	0.125342	41	{'n_neighbors 4
6	0.228750	0.048025	1.059379	0.235010	56	{'n_neighbors 5(
2	0.311753	0.040799	1.216632	0.265773	76	{'n_neighbors 7(
9	0.270957	0.199804	0.948423	0.458374	81	{'n_neighbors 8
1	0.168152	0.078784	1.293272	0.219475	86	{'n_neighbors 8(
8	0.108329	0.024517	1.590826	0.059955	91	{'n_neighbors 9

In [17]:

```
print_graph(results, 'param_n_neighbors', 'mean_test_score', 'Hyperpara
meter - No. of neighbors', 'Test score')
```

Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD



In [18]:
```
best_c=best_model.best_params_['n_neighbors']
best_c
```

Out[18]: 6

In [19]:
```
model = KNeighborsClassifier(n_neighbors=best_c,n_jobs=-1)
model.fit(train_data,y_train)
```

Out[19]: KNeighborsClassifier(n_jobs=-1, n_neighbors=6)

In [20]:
```
predictions = model.predict_proba(test_data)[:,1]
save_submission(predictions, "knn_raw.csv")
```

knn-raw

1.2 SVM with raw feature

In [21]:
```
C_val = uniform(loc=0, scale=4)
```

Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD

```
model= LinearSVC(verbose=verbose,random_state=random_state,class_weight
='balanced',max_iter=2000)
parameters={'C':C_val}
clf = RandomizedSearchCV(model,parameters,random_state=random_state,cv=
cv,verbose=verbose,scoring=scoring,n_jobs=-1)
best_model = clf.fit(train_data,y_train)
```

```
Fitting 5 folds for each of 10 candidates, totalling 50 fits
```

```
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 8 concurrent work
ers.
[Parallel(n_jobs=-1)]: Done  25 tasks      | elapsed:  1.3min
[Parallel(n_jobs=-1)]: Done  50 out of  50 | elapsed:  2.0min finished
```

```
[LibLinear]
```

```
/home/auw-mayank/.local/lib/python3.6/site-packages/sklearn/svm/_base.p
y:977: ConvergenceWarning: Liblinear failed to converge, increase the n
umber of iterations.
  "the number of iterations.", ConvergenceWarning)
```

In [22]:

```
best_c=best_model.best_params_['C']
best_c
```

Out[22]:

```
1.49816047538945
```

In [23]:

```
results = pd.DataFrame.from_dict(best_model.cv_results_)
results=results.sort_values('param_C')
results
```

Out[23]:

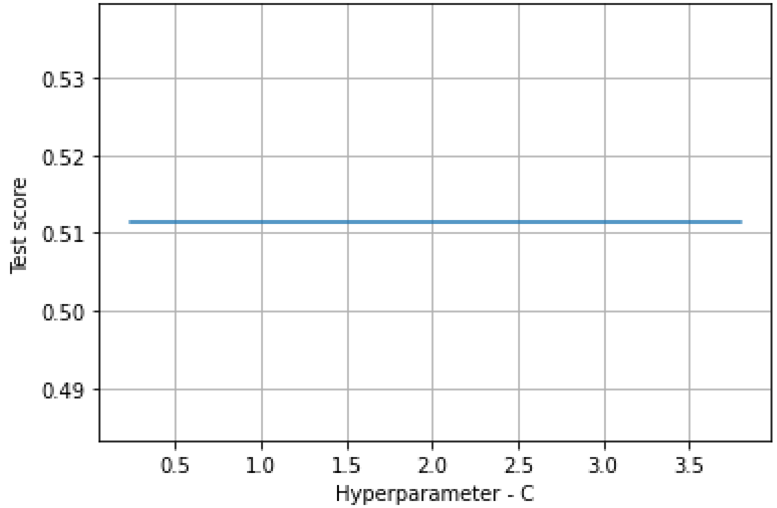
	mean_fit_time	std_fit_time	mean_score_time	std_score_time	param_C	params
6	18.684481	0.499781	0.009284	0.000217	0.232334	{'C': 0.23233444867279784}
5	19.870518	0.305372	0.009752	0.000904	0.623978	{'C': 0.6239780813448106}
4	19.918902	0.358008	0.011202	0.001661	0.624075	{'C': 0.6240745617697461}

Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD

	mean_fit_time	std_fit_time	mean_score_time	std_score_time	param_C	params
0	19.442288	0.216182	0.009220	0.000190	1.49816	{'C': 1.4981604753894 5}
3	19.357509	0.560751	0.009917	0.000562	2.39463	{'C': 2.3946339367881464}
8	18.831271	0.356759	0.008356	0.001407	2.40446	{'C': 2.404460046972835}
9	14.130057	5.223100	0.006211	0.001171	2.83229	{'C': 2.832290311184182}
2	18.946967	0.453250	0.009577	0.000543	2.92798	{'C': 2.9279757672456204}
7	18.603018	0.394303	0.009629	0.000678	3.4647	{'C': 3.4647045830997407}
1	19.380741	0.234885	0.009244	0.000622	3.80286	{'C': 3.8028572256396647}

In [24]:
```
print_graph(results, 'param_C', 'mean_test_score', 'Hyperparameter - C'
, 'Test score')
```

Plot showing the ROC_AUC score for various hyper parameter values



Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD

In [25]:

```python
#https://stackoverflow.com/questions/26478000/converting-linearsvcs-decision-function-to-probabilities-scikit-learn-python
model = LinearSVC(C=best_c,verbose=verbose,random_state=random_state,class_weight='balanced',max_iter=2000)
model = CalibratedClassifierCV(model)
model.fit(train_data,y_train)
```

```
[LibLinear]
```

```
/home/auw-mayank/.local/lib/python3.6/site-packages/sklearn/svm/_base.py:977: ConvergenceWarning: Liblinear failed to converge, increase the number of iterations.
  "the number of iterations.", ConvergenceWarning)
```

```
[LibLinear]
```

```
/home/auw-mayank/.local/lib/python3.6/site-packages/sklearn/svm/_base.py:977: ConvergenceWarning: Liblinear failed to converge, increase the number of iterations.
  "the number of iterations.", ConvergenceWarning)
```

```
[LibLinear]
```

```
/home/auw-mayank/.local/lib/python3.6/site-packages/sklearn/svm/_base.py:977: ConvergenceWarning: Liblinear failed to converge, increase the number of iterations.
  "the number of iterations.", ConvergenceWarning)
```

```
[LibLinear]
```

```
/home/auw-mayank/.local/lib/python3.6/site-packages/sklearn/svm/_base.py:977: ConvergenceWarning: Liblinear failed to converge, increase the number of iterations.
  "the number of iterations.", ConvergenceWarning)
```

```
[LibLinear]
```

```
/home/auw-mayank/.local/lib/python3.6/site-packages/sklearn/svm/_base.py:977: ConvergenceWarning: Liblinear failed to converge, increase the number of iterations.
  "the number of iterations.", ConvergenceWarning)
```

Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD

Out[25]:
```
CalibratedClassifierCV(base_estimator=LinearSVC(C=1.49816047538945,
                                                class_weight='balance
d',
                                                max_iter=2000, random_s
tate=42,
                                                verbose=2))
```

In [26]:
```
predictions = model.predict_proba(test_data)[:,1]
save_submission(predictions, 'svm_raw.csv')
```

svm-raw

1.3 Logistic Regression with Raw Feature

In [27]:
```
C_val = uniform(loc=0, scale=4)
lr= LogisticRegression(verbose=verbose,random_state=random_state,class_
weight='balanced',solver='lbfgs',max_iter=500,n_jobs=-1)
parameters={'C':C_val}
clf = RandomizedSearchCV(lr,parameters,random_state=random_state,cv=cv,
verbose=verbose,n_iter=100,scoring=scoring,n_jobs=-1)
best_model = clf.fit(train_data,y_train)
```

```
Fitting 5 folds for each of 100 candidates, totalling 500 fits
```

```
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 8 concurrent work
ers.
[Parallel(n_jobs=-1)]: Done  25 tasks      | elapsed:    1.2s
[Parallel(n_jobs=-1)]: Done 146 tasks      | elapsed:    6.1s
[Parallel(n_jobs=-1)]: Done 349 tasks      | elapsed:   14.3s
[Parallel(n_jobs=-1)]: Done 500 out of 500 | elapsed:   20.5s finished
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 8 concurrent work
ers.
[Parallel(n_jobs=-1)]: Done   1 out of   1 | elapsed:    0.3s finished
```

In [28]:
```
best_c=best_model.best_params_['C']
```

Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD

```
best_c
```

Out[28]: 1.49816047538945

In [29]:
```
results = pd.DataFrame.from_dict(best_model.cv_results_)
results=results.sort_values('param_C')
results
```

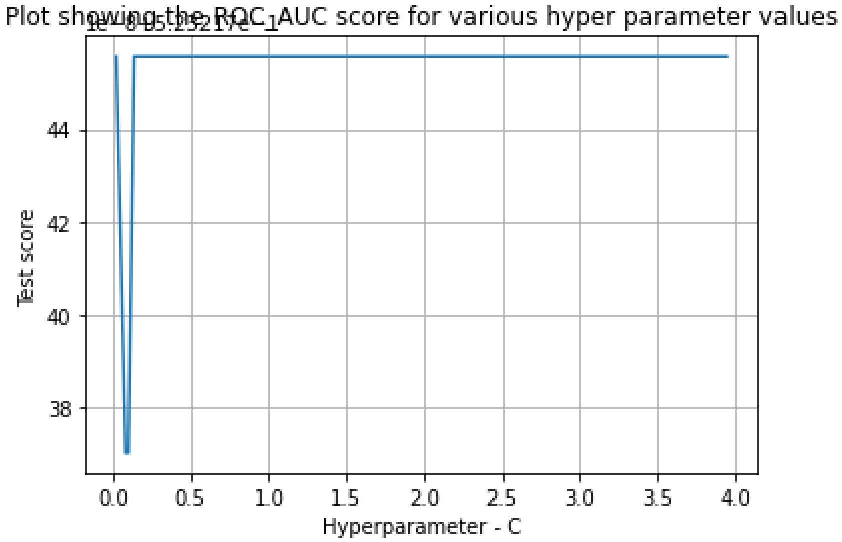
Out[29]:

	mean_fit_time	std_fit_time	mean_score_time	std_score_time	param_C	parai
72	0.305453	0.042840	0.007123	0.001411	0.0220885	{' 0.0220884684944095
10	0.299544	0.043052	0.009865	0.001636	0.082338	{' 0.082337977183209
98	0.324737	0.029356	0.009632	0.001840	0.101677	{' 0.101676506976380
42	0.322684	0.047769	0.008097	0.001223	0.137554	{' 0.137554084460873
58	0.312479	0.040692	0.010287	0.003491	0.180909	{' 0.180909155642152
...	...	...	...	...	...	
1	0.330116	0.064433	0.008579	0.001581	3.80286	{' 3.80285722563966
34	0.307352	0.038496	0.009020	0.000806	3.86253	{' 3.86252813229823
50	0.286139	0.050608	0.008554	0.001624	3.87834	{' 3.87833851105823
11	0.311766	0.046068	0.009899	0.002444	3.87964	{' 3.87963940864797
69	0.288068	0.053532	0.009172	0.001290	3.94755	{' 3.9475477464020

100 rows × 14 columns

Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD

In [30]:
```
print_graph(results, 'param_C', 'mean_test_score', 'Hyperparameter - C'
, 'Test score')
```



In [31]:
```
model = LogisticRegression(C=best_c,verbose=verbose,n_jobs=-1,random_st
ate=random_state,class_weight='balanced',solver='lbfgs')
model.fit(train_data,y_train)
```

```
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 8 concurrent work
ers.
[Parallel(n_jobs=-1)]: Done   1 out of   1 | elapsed:    0.3s finished
```

Out[31]:
```
LogisticRegression(C=1.49816047538945, class_weight='balanced', n_jobs=
-1,
                   random_state=42, verbose=2)
```

In [32]:
```
predictions = model.predict_proba(test_data)[:,1]
save_submission(predictions, 'lr_raw.csv')
```

lr-raw

Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD

1.4 Random Forest with Raw Feature

In [33]:
```
rfc = RandomForestClassifier(random_state=random_state,class_weight='ba
lanced',n_jobs=-1)
clf = RandomizedSearchCV(rfc,get_rf_params(),random_state=random_state,
cv=cv,verbose=verbose,n_iter=100,scoring=scoring,n_jobs=-1)
best_model = clf.fit(train_data,y_train)
```

```
Fitting 5 folds for each of 100 candidates, totalling 500 fits
```

```
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 8 concurrent work
ers.
[Parallel(n_jobs=-1)]: Done  25 tasks      | elapsed:   21.1s
/home/auw-mayank/.local/lib/python3.6/site-packages/joblib/externals/lo
ky/process_executor.py:691: UserWarning: A worker stopped while some jo
bs were given to the executor. This can be caused by a too short worker
timeout or by a memory leak.
  "timeout or by a memory leak.", UserWarning
[Parallel(n_jobs=-1)]: Done 146 tasks      | elapsed:  4.6min
[Parallel(n_jobs=-1)]: Done 349 tasks      | elapsed: 10.3min
[Parallel(n_jobs=-1)]: Done 500 out of 500 | elapsed: 15.1min finished
```

In [34]:
```
results = pd.DataFrame(best_model.cv_results_)
results.sort_values('mean_test_score',ascending=False,inplace=True)
param_keys=['param_'+str(each) for each in get_rf_params().keys()]
param_keys.append('mean_test_score')
results[param_keys].head(10)
```

Out[34]:

	param_n_estimators	param_max_depth	param_max_features	param_min_samples_split	mean
78	700	25	2	7	
62	500	25	3	5	
79	500	25	1	10	
55	200	25	2	5	
22	200	25	4	10	

Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD

	param_n_estimators	param_max_depth	param_max_features	param_min_samples_split	mean
20	1000	25	3	2	
85	1000	20	3	7	
33	700	25	4	2	
84	1000	25	5	2	
27	50	25	2	10	

In [35]:
```
n_estimators=clf.best_params_['n_estimators']
max_features=clf.best_params_['max_features']
max_depth=clf.best_params_['max_depth']
min_samples_split=clf.best_params_['min_samples_split']
n_estimators,max_features,max_depth,min_samples_split
```

Out[35]:
```
(700, 2, 25, 7)
```

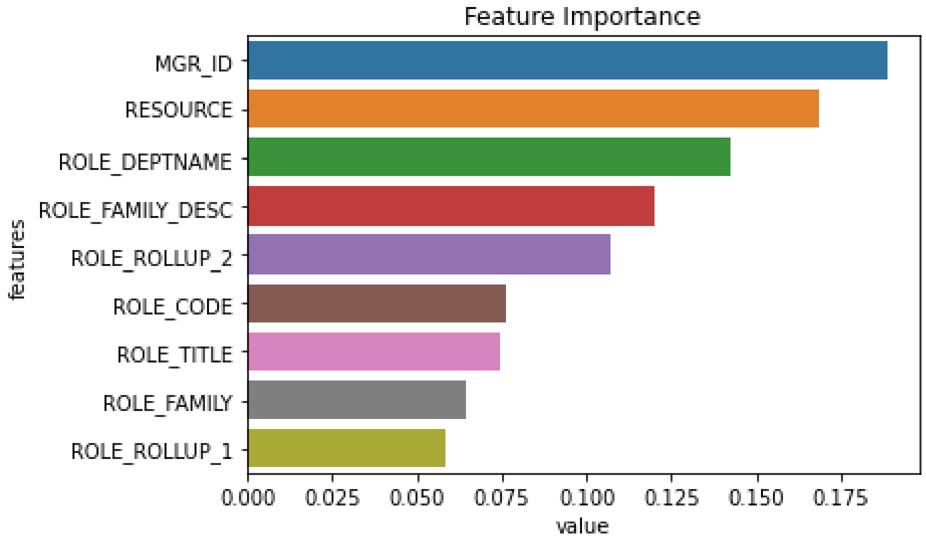
In [36]:
```
model=RandomForestClassifier(n_estimators=n_estimators,max_depth=max_de
pth,max_features=max_features,
                             min_samples_split=min_samples_split,
                             random_state=random_state,class_weight='ba
lanced',n_jobs=-1)

model.fit(train_data,y_train)
```

Out[36]:
```
RandomForestClassifier(class_weight='balanced', max_depth=25, max_featu
res=2,
                       min_samples_split=7, n_estimators=700, n_jobs=-
1,
                       random_state=42)
```

In [37]:
```
features=train_data.columns
importance=model.feature_importances_
features=pd.DataFrame({'features':features,'value':importance})
features=features.sort_values('value',ascending=False)
```

Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD

```
sns.barplot('value','features',data=features);
plt.title('Feature Importance');
```



Features Observations:

1. MGR_ID is the most important feature followed by RESOURCE and ROLE_DEPTNAME

In [38]:
```
predictions = model.predict_proba(test_data)[:,1]
save_submission(predictions, 'rf_raw.csv')
```

rf-raw

1.5 Xgboost with Raw Feature

In [39]:
```
xgb = XGBClassifier()
clf = RandomizedSearchCV(xgb,get_xgb_params(),random_state=random_state
```

Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD

```
,cv=cv,verbose=verbose,n_iter=100,scoring=scoring,n_jobs=-1)
best_model=clf.fit(train_data,y_train)
```

```
Fitting 5 folds for each of 100 candidates, totalling 500 fits
```

```
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 8 concurrent work
ers.
[Parallel(n_jobs=-1)]: Done  25 tasks      | elapsed:    7.2s
[Parallel(n_jobs=-1)]: Done 146 tasks      | elapsed:  1.5min
[Parallel(n_jobs=-1)]: Done 349 tasks      | elapsed:  4.1min
[Parallel(n_jobs=-1)]: Done 500 out of 500 | elapsed:  6.9min finished
```

In [40]:

```
results = pd.DataFrame(best_model.cv_results_)
results.sort_values('mean_test_score',ascending=False,inplace=True)
param_keys=['param_'+str(each) for each in get_xgb_params().keys()]
param_keys.append('mean_test_score')
results[param_keys].head(10)
```

Out[40]:

	param_n_estimators	param_learning_rate	param_subsample	param_max_depth	param_colsan
18	1000	0.048135	0.665922	9	
44	1000	0.060484	0.606429	6	
97	750	0.232385	0.907694	6	
96	500	0.0979629	0.98664	7	
62	500	0.0663892	0.328153	9	
49	500	0.160277	0.393098	8	
84	200	0.571989	0.967581	6	
53	200	0.540096	0.928319	6	
86	1000	0.475848	0.858413	9	
8	750	0.0686033	0.683264	6	

In [41]:

```
colsample_bytree = clf.best_params_['colsample_bytree']
learning_rate=clf.best_params_['learning_rate']
```

Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD

```
max_depth=clf.best_params_['max_depth']
min_child_weight=clf.best_params_['min_child_weight']
n_estimators=clf.best_params_['n_estimators']
subsample=clf.best_params_['subsample']
colsample_bytree,learning_rate,max_depth,min_child_weight,n_estimators,
subsample
```

Out[41]:

```
(0.3308980248526492, 0.04813501017161418, 9, 2, 1000, 0.665922356617496
7)
```

In [42]:

```
model = XGBClassifier(colsample_bytree=colsample_bytree,learning_rate=l
earning_rate,max_depth=max_depth,
                     min_child_weight=min_child_weight,n_estimators=n_e
stimators,subsample=subsample,n_jobs=-1)

model.fit(train_data,y_train)
```

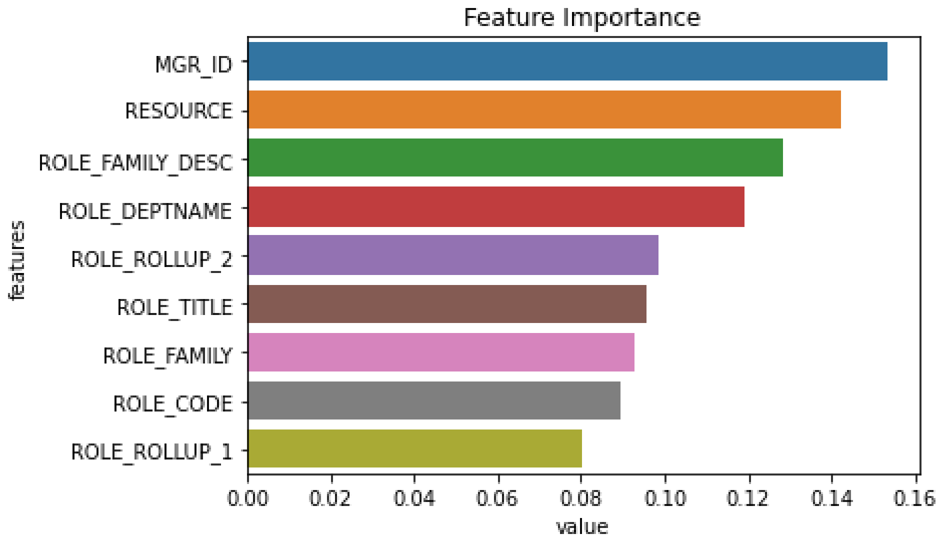
Out[42]:

```
XGBClassifier(base_score=0.5, booster='gbtree', colsample_bylevel=1,
              colsample_bynode=1, colsample_bytree=0.3308980248526492,
gamma=0,
              gpu_id=-1, importance_type='gain', interaction_constraint
s='',
              learning_rate=0.04813501017161418, max_delta_step=0, max_
depth=9,
              min_child_weight=2, missing=nan, monotone_constraints
='()',
              n_estimators=1000, n_jobs=-1, num_parallel_tree=1, random
_state=0,
              reg_alpha=0, reg_lambda=1, scale_pos_weight=1,
              subsample=0.6659223566174967, tree_method='exact',
              validate_parameters=1, verbosity=None)
```

In [43]:

```
features=train_data.columns
importance=model.feature_importances_
features=pd.DataFrame({'features':features,'value':importance})
features=features.sort_values('value',ascending=False)
sns.barplot('value','features',data=features);
plt.title('Feature Importance');
```

Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD



In [44]:
```
predictions = model.predict_proba(test_data)[:,1]
save_submission(predictions, 'xgb_raw.csv')
```

xgb-raw

kaggle-submission-raw

In [45]:
```
from prettytable import PrettyTable

x = PrettyTable(['Model', 'Feature', 'Private Score', 'Public Score'])
x.add_row(['KNN','Raw', 0.67224, 0.68148])
x.add_row(['SVM', 'Raw', 0.50286, 0.51390])
x.add_row(['Logistic Regression', 'Raw', 0.53857, 0.53034])
x.add_row(['Random Forest', 'Raw', 0.87269, 0.87567])
x.add_row(['Xgboost', 'Raw', 0.86988, 0.87909])

print(x)
```

```
+---------------------+---------+---------------+--------------+
|        Model        | Feature | Private Score | Public Score |
```

Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD

```
+---------------------+---------+--------------+-------------+
|         KNN         |   Raw   |   0.67224    |   0.68148   |
|         SVM         |   Raw   |   0.50286    |    0.5139   |
| Logistic Regression |   Raw   |   0.53857    |   0.53034   |
|    Random Forest    |   Raw   |   0.87269    |   0.87567   |
|       Xgboost       |   Raw   |   0.86988    |   0.87909   |
+---------------------+---------+--------------+-------------+
```

Observations:

1. Xgboost perform best on the raw features
2. Random forest also perform good on raw features
3. Tree based models performs better than linear models for raw features

Build model on one hot encoded features

2.1 KNN with one hot encoded features

In [46]:
```
train_ohe = sparse.load_npz('data/train_ohe.npz')
test_ohe = sparse.load_npz('data/test_ohe.npz')

train_ohe.shape, test_ohe.shape, y_train.shape
```

Out[46]:
```
((32769, 4500), (58921, 4500), (32769,))
```

In [47]:
```
parameters={'n_neighbors':np.arange(1,100, 5)}
clf = RandomizedSearchCV(KNeighborsClassifier(n_jobs=-1),parameters,ran
dom_state=random_state,cv=cv,verbose=verbose,scoring=scoring,n_jobs=4)
best_model = clf.fit(train_ohe,y_train)
```

```
Fitting 5 folds for each of 10 candidates, totalling 50 fits
```

```
[Parallel(n_jobs=4)]: Using backend LokyBackend with 4 concurrent worke
```

Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD

```
rs.
[Parallel(n_jobs=4)]: Done  33 tasks      | elapsed:  2.4min
[Parallel(n_jobs=4)]: Done  50 out of  50 | elapsed:  3.2min finished
```

In [48]:

```python
results = pd.DataFrame.from_dict(best_model.cv_results_)
results=results.sort_values('param_n_neighbors')
results
```

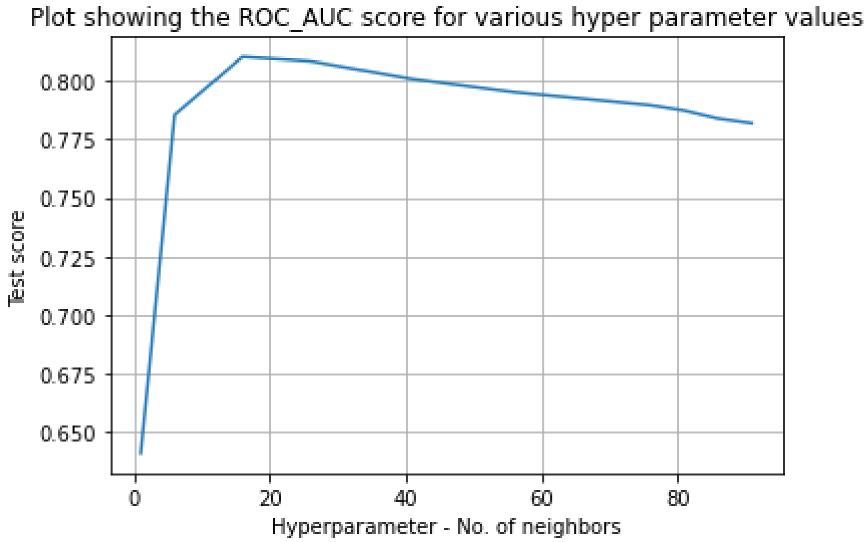
Out[48]:

	mean_fit_time	std_fit_time	mean_score_time	std_score_time	param_n_neighbors	param
0	0.008579	0.000699	12.889389	1.630046	1	{'n_neighbors
3	0.218653	0.141987	13.355838	1.399583	6	{'n_neighbors
7	0.014216	0.010986	11.480899	0.587346	16	{'n_neighbors 1(
5	0.007408	0.000506	11.399296	0.072001	26	{'n_neighbors 2(
4	0.029331	0.037009	11.730860	0.352833	41	{'n_neighbors 4
6	0.017152	0.013375	12.010613	0.940570	56	{'n_neighbors 5(
2	0.325559	0.159848	31.339497	13.256170	76	{'n_neighbors 7(
9	0.013539	0.006507	9.540637	2.054831	81	{'n_neighbors 8
1	0.103347	0.155776	24.211115	9.506957	86	{'n_neighbors 8(
8	0.016657	0.008175	11.831052	0.749635	91	{'n_neighbors 9

In [49]:

```python
print_graph(results, 'param_n_neighbors', 'mean_test_score', 'Hyperpara
meter - No. of neighbors', 'Test score')
```

Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD



In [50]:
```
best_c=best_model.best_params_['n_neighbors']
best_c
```

Out[50]: 16

In [51]:
```
model = KNeighborsClassifier(n_neighbors=best_c,n_jobs=-1)
model.fit(train_ohe,y_train)
```

Out[51]: KNeighborsClassifier(n_jobs=-1, n_neighbors=16)

In [52]:
```
predictions = model.predict_proba(test_ohe)[:,1]
save_submission(predictions, "knn_ohe.csv")
```

knn-ohe

2.2 SVM with one hot encoded features

In [53]:
```
C_val = uniform(loc=0, scale=4)
```

Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD

```
model= LinearSVC(verbose=verbose,random_state=random_state,class_weight
='balanced',max_iter=2000)
parameters={'C':C_val}
clf = RandomizedSearchCV(model,parameters,random_state=random_state,cv=
cv,verbose=verbose,scoring=scoring,n_jobs=-1)
best_model = clf.fit(train_ohe,y_train)
```

```
Fitting 5 folds for each of 10 candidates, totalling 50 fits
```

```
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 8 concurrent work
ers.
[Parallel(n_jobs=-1)]: Done  25 tasks      | elapsed:   31.0s
[Parallel(n_jobs=-1)]: Done  50 out of  50 | elapsed:   41.7s finished
```

```
[LibLinear]
```

In [54]:

```
best_c=best_model.best_params_['C']
best_c
```

Out[54]:

```
0.23233444867279784
```

In [55]:

```
results = pd.DataFrame.from_dict(best_model.cv_results_)
results=results.sort_values('param_C')
results
```

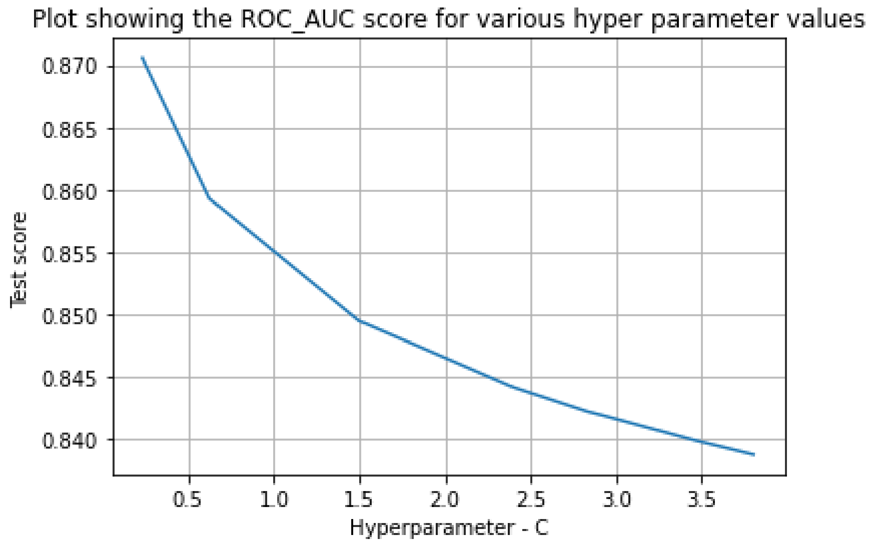
Out[55]:

	mean_fit_time	std_fit_time	mean_score_time	std_score_time	param_C	params
6	1.730894	0.069332	0.005605	0.000029	0.232334	{'C': 0.23233444867279784}
5	3.315943	0.201073	0.005879	0.000622	0.623978	{'C': 0.6239780813448106}
4	3.583880	0.365580	0.005711	0.000297	0.624075	{'C': 0.6240745617697461}
0	5.492834	0.287928	0.006418	0.000139	1.49816	{'C': 1.49816047538945}
3	4.961631	0.298074	0.005662	0.000146	2.39463	{'C': 2.3946339367881464}

Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD

	mean_fit_time	std_fit_time	mean_score_time	std_score_time	param_C	params
8	4.833444	0.644881	0.004830	0.000888	2.40446	{'C': 2.404460046972835}
9	3.986315	0.360801	0.003652	0.000362	2.83229	{'C': 2.832290311184182}
2	4.873211	0.174700	0.006166	0.000985	2.92798	{'C': 2.9279757672456204}
7	4.817450	0.265487	0.005635	0.000099	3.4647	{'C': 3.4647045830997407}
1	4.685154	0.254005	0.006274	0.000241	3.80286	{'C': 3.8028572256396647}

In [56]:
```
print_graph(results, 'param_C', 'mean_test_score', 'Hyperparameter - C'
, 'Test score')
```



In [57]:
```
#https://stackoverflow.com/questions/26478000/converting-linearsvcs-dec
ision-function-to-probabilities-scikit-learn-python
model = LinearSVC(C=best_c,verbose=verbose,random_state=random_state,cl
```

Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD

```
ass_weight='balanced',max_iter=2000)
model = CalibratedClassifierCV(model)
model.fit(train_ohe,y_train)
```

```
[LibLinear][LibLinear][LibLinear][LibLinear][LibLinear]
```

Out[57]:
```
CalibratedClassifierCV(base_estimator=LinearSVC(C=0.23233444867279784,
                                                class_weight='balance
d',
                                                max_iter=2000, random_s
tate=42,
                                                verbose=2))
```

In [58]:
```
predictions = model.predict_proba(test_ohe)[:,1]
save_submission(predictions, 'svm_ohe.csv')
```

svm-ohe

2.3 Logistic Regression with one hot encoded features

In [59]:
```
C_val = uniform(loc=0, scale=4)
lr= LogisticRegression(verbose=verbose,random_state=random_state,class_
weight='balanced',solver='lbfgs',max_iter=500,n_jobs=-1)
parameters={'C':C_val}
clf = RandomizedSearchCV(lr,parameters,random_state=random_state,cv=cv,
verbose=verbose,n_iter=100,scoring=scoring,n_jobs=-1)
best_model = clf.fit(train_ohe,y_train)
```

```
Fitting 5 folds for each of 100 candidates, totalling 500 fits
```

```
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 8 concurrent work
ers.
[Parallel(n_jobs=-1)]: Done  25 tasks      | elapsed:    7.2s
[Parallel(n_jobs=-1)]: Done 146 tasks      | elapsed:   32.8s
[Parallel(n_jobs=-1)]: Done 349 tasks      | elapsed:  1.4min
[Parallel(n_jobs=-1)]: Done 500 out of 500 | elapsed:  2.0min finished
```

Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD

```
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 8 concurrent work
ers.
[Parallel(n_jobs=-1)]: Done   1 out of   1 | elapsed:    0.8s finished
```

In [60]:
```
best_c=best_model.best_params_['C']
best_c
```

Out[60]: 0.6820964947491661

In [61]:
```
results = pd.DataFrame.from_dict(best_model.cv_results_)
results=results.sort_values('param_C')
results
```

Out[61]:

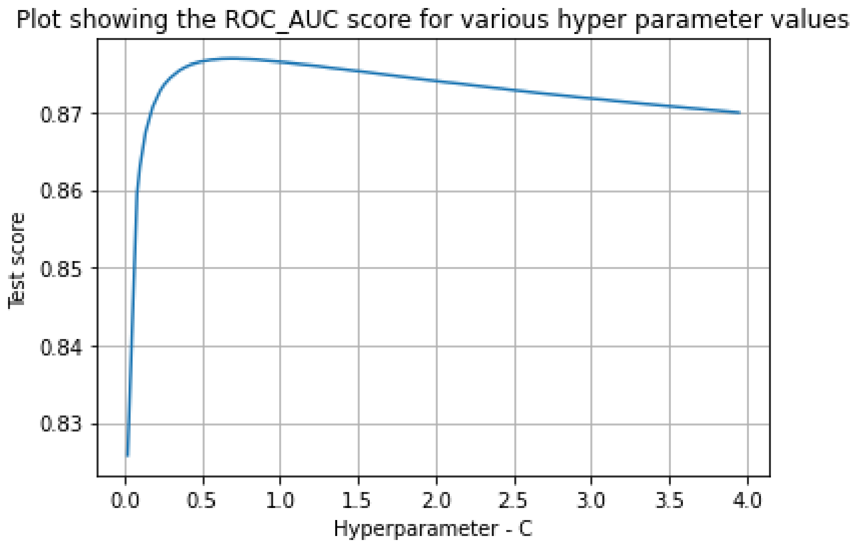
	mean_fit_time	std_fit_time	mean_score_time	std_score_time	param_C	parai
72	0.439680	0.041474	0.006678	0.000959	0.0220885	{' 0.0220884684944095
10	0.650064	0.051074	0.006208	0.001374	0.082338	{' 0.082337977183209
98	0.776743	0.046280	0.006456	0.000961	0.101677	{' 0.101676506976380
42	0.886947	0.055574	0.008943	0.003785	0.137554	{' 0.137554084460873
58	0.928974	0.072238	0.006947	0.000977	0.180909	{' 0.180909155642152
...	...	...	...	...	...	
1	2.534207	0.214963	0.006663	0.000782	3.80286	{' 3.80285722563966
34	2.376340	0.051137	0.006265	0.000269	3.86253	{' 3.86252813229823
50	2.498738	0.052591	0.007061	0.000964	3.87834	{' 3.87833851105823
11	2.401271	0.107580	0.006628	0.001671	3.87964	{' 3.87963940864797

Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD

	mean_fit_time	std_fit_time	mean_score_time	std_score_time	param_C	parai
69	2.674097	0.153710	0.007042	0.000979	3.94755	{' 3.94754774640206

100 rows × 14 columns

In [62]:
```
print_graph(results, 'param_C', 'mean_test_score', 'Hyperparameter - C'
, 'Test score')
```



In [63]:
```
model = LogisticRegression(C=best_c,verbose=verbose,n_jobs=-1,random_st
ate=random_state,class_weight='balanced',solver='lbfgs')
model.fit(train_ohe,y_train)
```

```
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 8 concurrent work
ers.
[Parallel(n_jobs=-1)]: Done   1 out of   1 | elapsed:    0.5s finished
```

Out[63]:
```
LogisticRegression(C=0.6820964947491661, class_weight='balanced', n_job
s=-1,
                   random_state=42, verbose=2)
```

Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD

In [64]:
```
predictions = model.predict_proba(test_ohe)[:,1]
save_submission(predictions, 'lr_ohe.csv')
```

lr-ohe

2.4 Random Forest with one hot encoded features

In [65]:
```
rfc = RandomForestClassifier(random_state=random_state,class_weight='ba
lanced',n_jobs=-1)
clf = RandomizedSearchCV(rfc,get_rf_params(),random_state=random_state,
cv=cv,verbose=verbose,n_iter=100,scoring=scoring,n_jobs=-1)
best_model = clf.fit(train_ohe,y_train)
```

```
Fitting 5 folds for each of 100 candidates, totalling 500 fits
```

```
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 8 concurrent work
ers.
[Parallel(n_jobs=-1)]: Done  25 tasks      | elapsed:    9.8s
[Parallel(n_jobs=-1)]: Done 146 tasks      | elapsed:  1.6min
[Parallel(n_jobs=-1)]: Done 349 tasks      | elapsed:  3.6min
[Parallel(n_jobs=-1)]: Done 500 out of 500 | elapsed:  5.3min finished
```

In [66]:
```
results = pd.DataFrame(best_model.cv_results_)
results.sort_values('mean_test_score',ascending=False,inplace=True)
param_keys=['param_'+str(each) for each in get_rf_params().keys()]
param_keys.append('mean_test_score')
results[param_keys].head(10)
```

Out[66]:

	param_n_estimators	param_max_depth	param_max_features	param_min_samples_split	mean
78	700	25	2	7	
85	1000	20	3	7	
62	500	25	3	5	
6	500	20	2	5	

Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD

	param_n_estimators	param_max_depth	param_max_features	param_min_samples_split	mean
11	1000	15	3	7	
25	700	15	4	7	
19	700	15	4	5	
22	200	25	4	10	
79	500	25	1	10	
82	700	20	5	20	

In [67]:

```
n_estimators=clf.best_params_['n_estimators']
max_features=clf.best_params_['max_features']
max_depth=clf.best_params_['max_depth']
min_samples_split=clf.best_params_['min_samples_split']
n_estimators,max_features,max_depth,min_samples_split
```

Out[67]:

```
(700, 2, 25, 7)
```

In [68]:

```
model=RandomForestClassifier(n_estimators=n_estimators,max_depth=max_de
pth,max_features=max_features,
                             min_samples_split=min_samples_split,
                             random_state=random_state,class_weight='ba
lanced',n_jobs=-1)

model.fit(train_ohe,y_train)
```

Out[68]:

```
RandomForestClassifier(class_weight='balanced', max_depth=25, max_featu
res=2,
                       min_samples_split=7, n_estimators=700, n_jobs=-
1,
                       random_state=42)
```

In [69]:

```
# features=train_ohe.columns
# importance=model.feature_importances_
# features=pd.DataFrame({'features':features,'value':importance})
# features=features.sort_values('value',ascending=False)
```

Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD

```
# sns.barplot('value','features',data=features);
# plt.title('Feature Importance');
```

In [70]:
```
predictions = model.predict_proba(test_ohe)[:,1]
save_submission(predictions, 'rf_ohe.csv')
```

rf-ohe

2.5 Xgboost with one hot encoded features

In [71]:
```
xgb = XGBClassifier()
clf = RandomizedSearchCV(xgb,get_xgb_params(),random_state=random_state
,cv=cv,verbose=verbose,n_iter=100,scoring=scoring,n_jobs=-1)
best_model=clf.fit(train_ohe,y_train)
```

```
Fitting 5 folds for each of 100 candidates, totalling 500 fits
```

```
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 8 concurrent work
ers.
[Parallel(n_jobs=-1)]: Done  25 tasks      | elapsed:    8.0s
[Parallel(n_jobs=-1)]: Done 146 tasks      | elapsed:  1.6min
[Parallel(n_jobs=-1)]: Done 349 tasks      | elapsed:  4.3min
[Parallel(n_jobs=-1)]: Done 500 out of 500 | elapsed:  7.1min finished
```

In [72]:
```
results = pd.DataFrame(best_model.cv_results_)
results.sort_values('mean_test_score',ascending=False,inplace=True)
param_keys=['param_'+str(each) for each in get_xgb_params().keys()]
param_keys.append('mean_test_score')
results[param_keys].head(10)
```

Out[72]:

	param_n_estimators	param_learning_rate	param_subsample	param_max_depth	param_colsan
97	750	0.232385	0.907694	6	
80	1000	0.385564	0.905351	3	
86	1000	0.475848	0.858413	9	

Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD

	param_n_estimators	param_learning_rate	param_subsample	param_max_depth	param_colsan
84	200	0.571989	0.967581	6	
14	200	0.374221	0.802197	7	
50	500	0.388683	0.645103	4	
53	200	0.540096	0.928319	6	
92	200	0.478778	0.49442	9	
96	500	0.0979629	0.98664	7	
22	1000	0.391846	0.695516	6	

In [73]:
```
colsample_bytree = clf.best_params_['colsample_bytree']
learning_rate=clf.best_params_['learning_rate']
max_depth=clf.best_params_['max_depth']
min_child_weight=clf.best_params_['min_child_weight']
n_estimators=clf.best_params_['n_estimators']
subsample=clf.best_params_['subsample']
colsample_bytree,learning_rate,max_depth,min_child_weight,n_estimators,
subsample
```

Out[73]:
```
(0.3742707957561203, 0.23238528824013455, 6, 1, 750, 0.907693706348546
3)
```

In [74]:
```
model = XGBClassifier(colsample_bytree=colsample_bytree,learning_rate=l
earning_rate,max_depth=max_depth,
                     min_child_weight=min_child_weight,n_estimators=n_e
stimators,subsample=subsample,n_jobs=-1)

model.fit(train_ohe,y_train)
```

Out[74]:
```
XGBClassifier(base_score=0.5, booster='gbtree', colsample_bylevel=1,
              colsample_bynode=1, colsample_bytree=0.3742707957561203,
gamma=0,
              gpu_id=-1, importance_type='gain', interaction_constraint
s='',
              learning_rate=0.23238528824013455, max_delta_step=0, max_
```

Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD

```
depth=6,
                     min_child_weight=1, missing=nan, monotone_constraints
='()',
                     n_estimators=750, n_jobs=-1, num_parallel_tree=1, random_
state=0,
                     reg_alpha=0, reg_lambda=1, scale_pos_weight=1,
                     subsample=0.9076937063485463, tree_method='exact',
                     validate_parameters=1, verbosity=None)
```

In [75]:

```
# features=train_ohe.columns
# importance=model.feature_importances_
# features=pd.DataFrame({'features':features,'value':importance})
# features=features.sort_values('value',ascending=False)
# sns.barplot('value','features',data=features);
# plt.title('Feature Importance');
```

In [76]:

```
predictions = model.predict_proba(test_ohe)[:,1]
save_submission(predictions, 'xgb_ohe.csv')
```

xgb-ohe

kaggle-submission-ohe

In [77]:

```
from prettytable import PrettyTable

x = PrettyTable(['Model', 'Feature', 'Private Score', 'Public Score'])
x.add_row(['KNN','ohe', 0.81657, 0.81723])
x.add_row(['SVM', 'ohe', 0.87249, 0.87955])
x.add_row(['Logistic Regression', 'ohe', 0.87436, 0.88167])
x.add_row(['Random Forest', 'ohe', 0.84541, 0.84997])
x.add_row(['Xgboost', 'ohe', 0.84717, 0.85102])

print(x)
```

```
+---------------------+---------+---------------+--------------+
|        Model        | Feature | Private Score | Public Score |
+---------------------+---------+---------------+--------------+
```

Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD

```
|         KNN         |  ohe   |    0.81657    |    0.81723    |
|         SVM         |  ohe   |    0.87249    |    0.87955    |
| Logistic Regression |  ohe   |    0.87436    |    0.88167    |
|    Random Forest    |  ohe   |    0.84541    |    0.84997    |
|       Xgboost       |  ohe   |    0.84717    |    0.85102    |
+---------------------+--------+---------------+---------------+
```

Observations:

1. One hot encoding features performs better than other encoding technique
2. Linear models (Logistic Regression and SVM) performs better on higher dimension

3 Build Model on frequency encoding feature

3.1 KNN with frequency encoding

In [78]:
```
train_df_fc = pd.read_csv('data/train_df_fc.csv')
test_df_fc = pd.read_csv('data/test_df_fc.csv')
```

In [79]:
```
train_df_fc.shape, test_df_fc.shape, y_train.shape
```

Out[79]:
```
((32769, 9), (58921, 9), (32769,))
```

In [80]:
```
parameters={'n_neighbors':np.arange(1,100, 5)}
clf = RandomizedSearchCV(KNeighborsClassifier(n_jobs=-1),parameters,ran
dom_state=random_state,cv=cv,verbose=verbose,scoring=scoring,n_jobs=-1)
best_model = clf.fit(train_df_fc,y_train)
```

```
Fitting 5 folds for each of 10 candidates, totalling 50 fits
```

```
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 8 concurrent work
ers.
```

Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD

```
[Parallel(n_jobs=-1)]: Done  25 tasks      | elapsed:    5.8s
[Parallel(n_jobs=-1)]: Done  50 out of  50 | elapsed:   10.4s finished
```

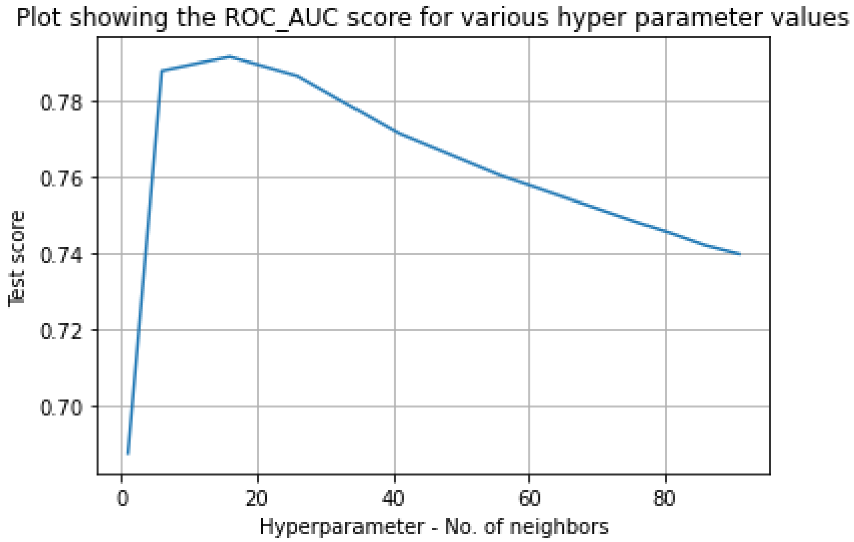
In [81]:
```python
results = pd.DataFrame.from_dict(best_model.cv_results_)
results=results.sort_values('param_n_neighbors')
results
```

Out[81]:

	mean_fit_time	std_fit_time	mean_score_time	std_score_time	param_n_neighbors	param
0	1.245504	0.191884	0.444588	0.057236	1	{'n_neighbors
3	0.966955	0.192374	0.425104	0.082150	6	{'n_neighbors (
7	0.915347	0.088786	0.510648	0.062272	16	{'n_neighbors 1(
5	0.910787	0.067158	0.587300	0.075423	26	{'n_neighbors 2(
4	0.867675	0.145174	0.647190	0.063128	41	{'n_neighbors 4
6	0.877225	0.208047	0.709570	0.113621	56	{'n_neighbors 5(
2	0.947447	0.128830	0.768725	0.107211	76	{'n_neighbors 7(
9	0.934347	0.100315	0.597521	0.257969	81	{'n_neighbors 8
1	1.011318	0.326486	0.772521	0.073501	86	{'n_neighbors 8(
8	0.832900	0.196410	0.861454	0.065058	91	{'n_neighbors 9

In [82]:
```python
print_graph(results, 'param_n_neighbors', 'mean_test_score', 'Hyperpara
meter - No. of neighbors', 'Test score')
```

Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD



In [83]:
```
best_c=best_model.best_params_['n_neighbors']
best_c
```

Out[83]: 16

In [84]:
```
model = KNeighborsClassifier(n_neighbors=best_c,n_jobs=-1)
model.fit(train_df_fc,y_train)
```

Out[84]: KNeighborsClassifier(n_jobs=-1, n_neighbors=16)

In [85]:
```
predictions = model.predict_proba(test_df_fc)[:,1]
save_submission(predictions, "knn_fc.csv")
```

knn-fc

3.2 SVM with frequency encoding

In [86]:
```
C_val = uniform(loc=0, scale=4)
```

Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD

```
model= LinearSVC(verbose=verbose,random_state=random_state,class_weight
='balanced',max_iter=2000)
parameters={'C':C_val}
clf = RandomizedSearchCV(model,parameters,random_state=random_state,cv=
cv,verbose=verbose,scoring=scoring,n_jobs=-1)
best_model = clf.fit(train_df_fc,y_train)
```

```
Fitting 5 folds for each of 10 candidates, totalling 50 fits
```

```
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 8 concurrent work
ers.
[Parallel(n_jobs=-1)]: Done  25 tasks      | elapsed:   17.8s
[Parallel(n_jobs=-1)]: Done  50 out of  50 | elapsed:   31.9s finished
```

```
[LibLinear]
```

In [87]:

```
best_c=best_model.best_params_['C']
best_c
```

Out[87]: 3.4647045830997407

In [88]:

```
results = pd.DataFrame.from_dict(best_model.cv_results_)
results=results.sort_values('param_C')
results
```

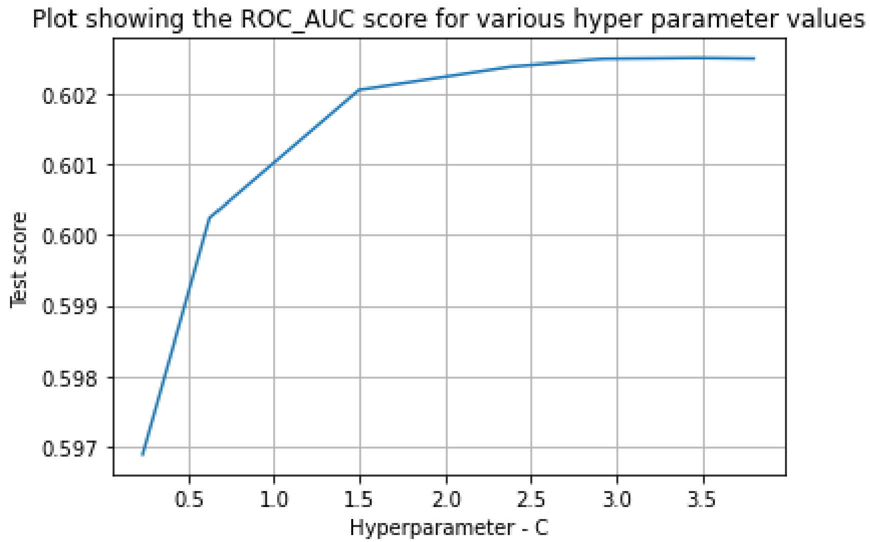
Out[88]:

	mean_fit_time	std_fit_time	mean_score_time	std_score_time	param_C	params
6	0.707697	0.038330	0.009112	0.000330	0.232334	{'C': 0.23233444867279784}
5	1.619894	0.069040	0.009326	0.000816	0.623978	{'C': 0.6239780813448106}
4	1.579539	0.025513	0.008887	0.000250	0.624075	{'C': 0.6240745617697461}
0	3.577643	0.097967	0.009332	0.000580	1.49816	{'C': 1.49816047538945}
3	5.732613	0.118254	0.009076	0.000258	2.39463	{'C': 2.3946339367881464}

Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD

	mean_fit_time	std_fit_time	mean_score_time	std_score_time	param_C	params
8	5.664490	0.136819	0.009409	0.001280	2.40446	{'C': 2.404460046972835}
9	5.364426	0.630964	0.005462	0.000641	2.83229	{'C': 2.832290311184182}
2	6.995208	0.140802	0.009158	0.000407	2.92798	{'C': 2.9279757672456204}
7	8.412424	0.706659	0.009016	0.000114	3.4647	{'C': 3.4647045830997407}
1	9.626733	0.661829	0.008899	0.000069	3.80286	{'C': 3.8028572256396647}

In [89]:
```
print_graph(results, 'param_C', 'mean_test_score', 'Hyperparameter - C'
, 'Test score')
```



In [90]:
```
#https://stackoverflow.com/questions/26478000/converting-linearsvcs-dec
ision-function-to-probabilities-scikit-learn-python
model = LinearSVC(C=best_c,verbose=verbose,random_state=random_state,cl
```

Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD

```
ass_weight='balanced',max_iter=2000)
model = CalibratedClassifierCV(model)
model.fit(train_df_fc,y_train)
```

```
[LibLinear][LibLinear][LibLinear][LibLinear][LibLinear]
```

Out[90]:
```
CalibratedClassifierCV(base_estimator=LinearSVC(C=3.4647045830997407,
                                                class_weight='balance
d',
                                                max_iter=2000, random_s
tate=42,
                                                verbose=2))
```

In [91]:
```
predictions = model.predict_proba(test_df_fc)[:,1]
save_submission(predictions, 'svm_fc.csv')
```

svm-fc

3.3 Logistic Regression with frequency encoding

In [92]:
```
C_val = uniform(loc=0, scale=4)
lr= LogisticRegression(verbose=verbose,random_state=random_state,class_
weight='balanced',solver='lbfgs',max_iter=500,n_jobs=-1)
parameters={'C':C_val}
clf = RandomizedSearchCV(lr,parameters,random_state=random_state,cv=cv,
verbose=verbose,n_iter=100,scoring=scoring,n_jobs=-1)
best_model = clf.fit(train_df_fc,y_train)
```

```
Fitting 5 folds for each of 100 candidates, totalling 500 fits
```

```
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 8 concurrent work
ers.
[Parallel(n_jobs=-1)]: Done  25 tasks      | elapsed:    1.7s
[Parallel(n_jobs=-1)]: Done 146 tasks      | elapsed:    8.3s
[Parallel(n_jobs=-1)]: Done 349 tasks      | elapsed:   20.2s
[Parallel(n_jobs=-1)]: Done 500 out of 500 | elapsed:   29.2s finished
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 8 concurrent work
```

Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD

```
ers.
[Parallel(n_jobs=-1)]: Done   1 out of   1 | elapsed:    0.4s finished
```

In [93]:
```
best_c=best_model.best_params_['C']
best_c
```

Out[93]:
```
3.947547746402069
```

In [94]:
```
results = pd.DataFrame.from_dict(best_model.cv_results_)
results=results.sort_values('param_C')
results
```

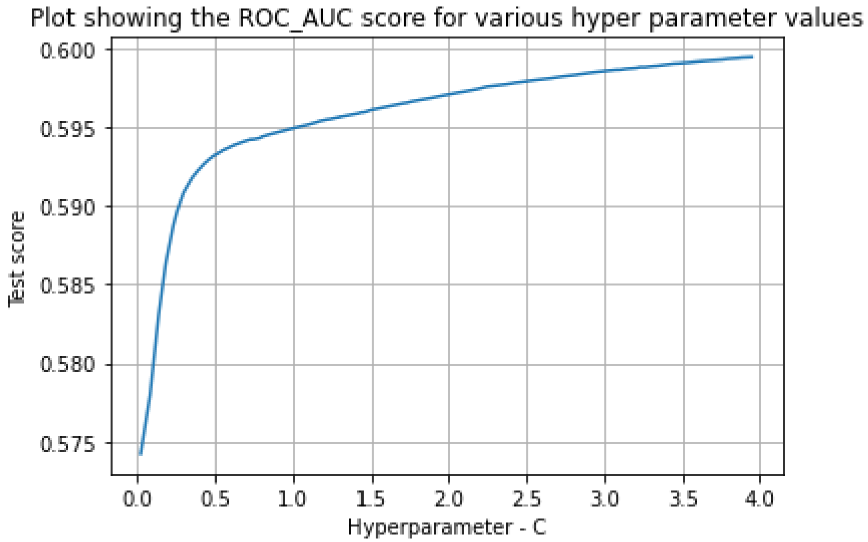
Out[94]:

	mean_fit_time	std_fit_time	mean_score_time	std_score_time	param_C	parai
72	0.182546	0.056575	0.010424	0.001924	0.0220885	{' 0.0220884684944095
10	0.230978	0.011034	0.010079	0.002740	0.082338	{' 0.082337977183209
98	0.246538	0.040240	0.008149	0.000789	0.101677	{' 0.101676506976380
42	0.264855	0.051649	0.007584	0.001100	0.137554	{' 0.137554084460873
58	0.294973	0.058159	0.009848	0.001887	0.180909	{' 0.180909155642152
...	...	...	...	...	...	
1	0.544588	0.077972	0.010987	0.002171	3.80286	{' 3.80285722563966
34	0.517311	0.047540	0.010529	0.004522	3.86253	{' 3.86252813229823
50	0.546014	0.078366	0.011045	0.001893	3.87834	{' 3.87833851105823
11	0.487671	0.048548	0.010200	0.002548	3.87964	{' 3.87963940864797
69	0.589011	0.053345	0.010957	0.001444	3.94755	{' 3.94754774640206

Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD

100 rows × 14 columns

In [95]:
```
print_graph(results, 'param_C', 'mean_test_score', 'Hyperparameter - C'
, 'Test score')
```



In [96]:
```
model = LogisticRegression(C=best_c,verbose=verbose,n_jobs=-1,random_st
ate=random_state,class_weight='balanced',solver='lbfgs')
model.fit(train_df_fc,y_train)
```

```
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 8 concurrent work
ers.
[Parallel(n_jobs=-1)]: Done   1 out of   1 | elapsed:    0.5s finished
```

Out[96]:
```
LogisticRegression(C=3.947547746402069, class_weight='balanced', n_jobs
=-1,
                   random_state=42, verbose=2)
```

In [97]:
```
predictions = model.predict_proba(test_df_fc)[:,1]
save_submission(predictions, 'lr_fc.csv')
```

Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD

lr-fc

3.4 Random Forest with frequency encoding

In [98]:

```
rfc = RandomForestClassifier(random_state=random_state,class_weight='balanced',n_jobs=-1)
clf = RandomizedSearchCV(rfc,get_rf_params(),random_state=random_state,
cv=cv,verbose=verbose,n_iter=100,scoring=scoring,n_jobs=-1)
best_model = clf.fit(train_df_fc,y_train)
```

```
Fitting 5 folds for each of 100 candidates, totalling 500 fits
```

```
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.
[Parallel(n_jobs=-1)]: Done  25 tasks      | elapsed:   18.9s
[Parallel(n_jobs=-1)]: Done 146 tasks      | elapsed:  3.5min
[Parallel(n_jobs=-1)]: Done 349 tasks      | elapsed:  8.3min
/home/auw-mayank/.local/lib/python3.6/site-packages/joblib/externals/loky/process_executor.py:691: UserWarning: A worker stopped while some jobs were given to the executor. This can be caused by a too short worker timeout or by a memory leak.
  "timeout or by a memory leak.", UserWarning
[Parallel(n_jobs=-1)]: Done 500 out of 500 | elapsed: 12.3min finished
```

In [99]:

```
results = pd.DataFrame(best_model.cv_results_)
results.sort_values('mean_test_score',ascending=False,inplace=True)
param_keys=['param_'+str(each) for each in get_rf_params().keys()]
param_keys.append('mean_test_score')
results[param_keys].head(10)
```

Out[99]:

	param_n_estimators	param_max_depth	param_max_features	param_min_samples_split	mean
78	700	25	2	7	
79	500	25	1	10	
85	1000	20	3	7	

Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD

	param_n_estimators	param_max_depth	param_max_features	param_min_samples_split	mean
55	200	25	2	5	
27	50	25	2	10	
62	500	25	3	5	
22	200	25	4	10	
6	500	20	2	5	
76	50	25	1	5	
84	1000	25	5	2	

In [100]:

```
n_estimators=clf.best_params_['n_estimators']
max_features=clf.best_params_['max_features']
max_depth=clf.best_params_['max_depth']
min_samples_split=clf.best_params_['min_samples_split']
n_estimators,max_features,max_depth,min_samples_split
```

Out[100]:

```
(700, 2, 25, 7)
```

In [101]:

```
model=RandomForestClassifier(n_estimators=n_estimators,max_depth=max_de
pth,max_features=max_features,
                             min_samples_split=min_samples_split,
                             random_state=random_state,class_weight='ba
lanced',n_jobs=-1)

model.fit(train_df_fc,y_train)
```

Out[101]:

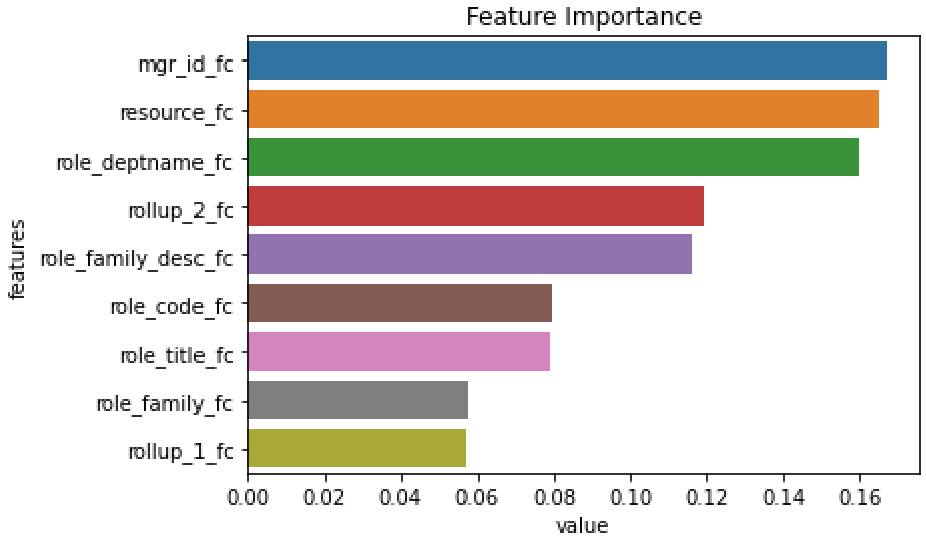
```
RandomForestClassifier(class_weight='balanced', max_depth=25, max_featu
res=2,
                       min_samples_split=7, n_estimators=700, n_jobs=-
1,
                       random_state=42)
```

In [103]:

```
features=train_df_fc.columns
importance=model.feature_importances_
```

Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD

```
features=pd.DataFrame({'features':features,'value':importance})
features=features.sort_values('value',ascending=False)
sns.barplot('value','features',data=features);
plt.title('Feature Importance');
```



In [106]:
```
predictions = model.predict_proba(test_df_fc)[:,1]
save_submission(predictions, 'rf_fc.csv')
```

rf-fc

3.5 Xgboost with frequency encoding

In [107]:
```
xgb = XGBClassifier()
clf = RandomizedSearchCV(xgb,get_xgb_params(),random_state=random_state
,cv=cv,verbose=verbose,n_iter=100,scoring=scoring,n_jobs=-1)
best_model=clf.fit(train_df_fc,y_train)
```

```
Fitting 5 folds for each of 100 candidates, totalling 500 fits
```

```
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 8 concurrent work
```

Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD

```
ers.
[Parallel(n_jobs=-1)]: Done  25 tasks      | elapsed:    8.5s
[Parallel(n_jobs=-1)]: Done 146 tasks      | elapsed:  1.6min
[Parallel(n_jobs=-1)]: Done 349 tasks      | elapsed:  4.4min
[Parallel(n_jobs=-1)]: Done 500 out of 500 | elapsed:  7.3min finished
```

In [108]:

```
results = pd.DataFrame(best_model.cv_results_)
results.sort_values('mean_test_score',ascending=False,inplace=True)
param_keys=['param_'+str(each) for each in get_xgb_params().keys()]
param_keys.append('mean_test_score')
results[param_keys].head(10)
```

Out[108]:

	param_n_estimators	param_learning_rate	param_subsample	param_max_depth	param_colsan
18	1000	0.048135	0.665922	9	
96	500	0.0979629	0.98664	7	
44	1000	0.060484	0.606429	6	
97	750	0.232385	0.907694	6	
86	1000	0.475848	0.858413	9	
53	200	0.540096	0.928319	6	
84	200	0.571989	0.967581	6	
49	500	0.160277	0.393098	8	
62	500	0.0663892	0.328153	9	
14	200	0.374221	0.802197	7	

In [109]:

```
colsample_bytree = clf.best_params_['colsample_bytree']
learning_rate=clf.best_params_['learning_rate']
max_depth=clf.best_params_['max_depth']
min_child_weight=clf.best_params_['min_child_weight']
n_estimators=clf.best_params_['n_estimators']
subsample=clf.best_params_['subsample']
```

Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD

```
colsample_bytree,learning_rate,max_depth,min_child_weight,n_estimators,
subsample
```

Out[109]:

```
(0.3308980248526492, 0.04813501017161418, 9, 2, 1000, 0.665922356617496
7)
```

In [110]:

```
model = XGBClassifier(colsample_bytree=colsample_bytree,learning_rate=l
earning_rate,max_depth=max_depth,
                      min_child_weight=min_child_weight,n_estimators=n_e
stimators,subsample=subsample,n_jobs=-1)

model.fit(train_df_fc,y_train)
```

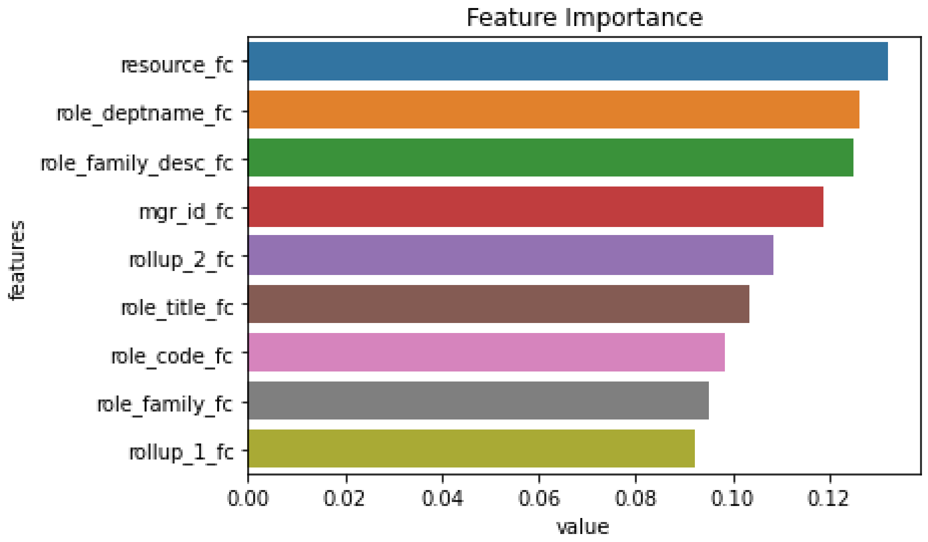
Out[110]:

```
XGBClassifier(base_score=0.5, booster='gbtree', colsample_bylevel=1,
              colsample_bynode=1, colsample_bytree=0.3308980248526492,
gamma=0,
              gpu_id=-1, importance_type='gain', interaction_constraint
s='',
              learning_rate=0.04813501017161418, max_delta_step=0, max_
depth=9,
              min_child_weight=2, missing=nan, monotone_constraints
='()',
              n_estimators=1000, n_jobs=-1, num_parallel_tree=1, random
_state=0,
              reg_alpha=0, reg_lambda=1, scale_pos_weight=1,
              subsample=0.6659223566174967, tree_method='exact',
              validate_parameters=1, verbosity=None)
```

In [111]:

```
features=train_df_fc.columns
importance=model.feature_importances_
features=pd.DataFrame({'features':features,'value':importance})
features=features.sort_values('value',ascending=False)
sns.barplot('value','features',data=features);
plt.title('Feature Importance');
```

Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD



In [112]:
```
predictions = model.predict_proba(test_df_fc)[:,1]
save_submission(predictions, 'xgb_fc.csv')
```

xgb-fc

kaggle-submission-fc

In [113]:
```
from prettytable import PrettyTable

x = PrettyTable(['Model', 'Feature', 'Private Score', 'Public Score'])
x.add_row(['KNN','fc', 0.79715, 0.79125])
x.add_row(['SVM', 'fc', 0.60085, 0.59550])
x.add_row(['Logistic Regression', 'fc', 0.59896, 0.59778])
x.add_row(['Random Forest', 'fc', 0.87299, 0.87616])
x.add_row(['Xgboost', 'fc', 0.86987, 0.86944])

print(x)
```

```
+---------------------+---------+---------------+--------------+
|        Model        | Feature | Private Score | Public Score |
```

Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD

KNN	fc	0.79715	0.79125
SVM	fc	0.60085	0.5955
Logistic Regression	fc	0.59896	0.59778
Random Forest	fc	0.87299	0.87616
Xgboost	fc	0.86987	0.86944

Observations:

1. Tree based models performs better for this feature than linear models
2. KNN is doing good for every feature

4 Build Model using response encoding feature

In [114]:
```
train_df_rc = pd.read_csv('data/train_df_rc.csv')
test_df_rc = pd.read_csv('data/test_df_rc.csv')
```

In [115]:
```
train_df_rc.shape, test_df_rc.shape, y_train.shape
```

Out[115]:
```
((32769, 9), (58921, 9), (32769,))
```

4.1 KNN with response encoding

In [116]:
```
parameters={'n_neighbors':np.arange(1,100, 5)}
clf = RandomizedSearchCV(KNeighborsClassifier(n_jobs=-1),parameters,ran
dom_state=random_state,cv=cv,verbose=verbose,scoring=scoring,n_jobs=-1)
best_model = clf.fit(train_df_rc,y_train)
```

```
Fitting 5 folds for each of 10 candidates, totalling 50 fits
```

Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD

```
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 8 concurrent work
ers.
[Parallel(n_jobs=-1)]: Done  25 tasks      | elapsed:    5.2s
[Parallel(n_jobs=-1)]: Done  50 out of  50 | elapsed:   10.7s finished
```

In [117]:

```
results = pd.DataFrame.from_dict(best_model.cv_results_)
results=results.sort_values('param_n_neighbors')
results
```

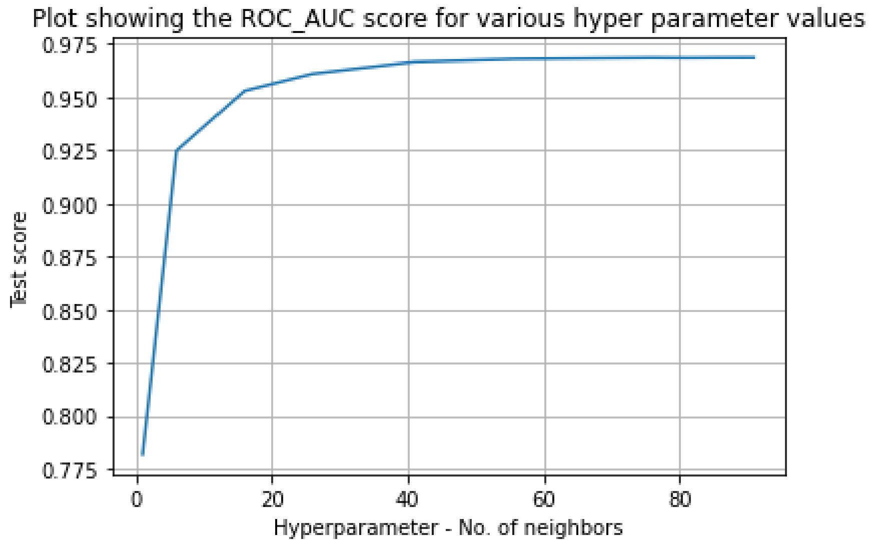
Out[117]:

	mean_fit_time	std_fit_time	mean_score_time	std_score_time	param_n_neighbors	param
0	0.087703	0.003754	0.270912	0.118324	1	{'n_neighbors
3	0.161678	0.063581	0.749612	0.191282	6	{'n_neighbors
7	0.365460	0.149200	1.097595	0.234131	16	{'n_neighbors 1(
5	0.274667	0.128723	1.106008	0.062541	26	{'n_neighbors 2(
4	0.196924	0.091312	1.247186	0.043935	41	{'n_neighbors 4
6	0.121391	0.037696	1.616742	0.078125	56	{'n_neighbors 5(
2	0.257782	0.047633	1.718469	0.251959	76	{'n_neighbors 7(
9	0.259518	0.169725	1.463110	0.571909	81	{'n_neighbors 8
1	0.166987	0.110239	2.083340	0.073632	86	{'n_neighbors 8(
8	0.215055	0.138284	2.578594	0.120702	91	{'n_neighbors 9

In [118]:

```
print_graph(results, 'param_n_neighbors', 'mean_test_score', 'Hyperpara
meter - No. of neighbors', 'Test score')
```

Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD



In [119]:
```
best_c=best_model.best_params_['n_neighbors']
best_c
```

Out[119]: 91

In [120]:
```
model = KNeighborsClassifier(n_neighbors=best_c,n_jobs=-1)
model.fit(train_df_rc,y_train)
```

Out[120]: KNeighborsClassifier(n_jobs=-1, n_neighbors=91)

In [121]:
```
predictions = model.predict_proba(test_df_rc)[:,1]
save_submission(predictions, "knn_rc.csv")
```

knn-rc

4.2 SVM with response encoding

In [122]:
```
C_val = uniform(loc=0, scale=4)
```

Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD

```
model= LinearSVC(verbose=verbose,random_state=random_state,class_weight
='balanced',max_iter=2000)
parameters={'C':C_val}
clf = RandomizedSearchCV(model,parameters,random_state=random_state,cv=
cv,verbose=verbose,scoring=scoring,n_jobs=-1)
best_model = clf.fit(train_df_rc,y_train)
```

```
Fitting 5 folds for each of 10 candidates, totalling 50 fits
```

```
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 8 concurrent work
ers.
[Parallel(n_jobs=-1)]: Done  25 tasks      | elapsed:   59.3s
[Parallel(n_jobs=-1)]: Done  50 out of  50 | elapsed:  1.6min finished
```

```
[LibLinear]
```

In [123]:

```
best_c=best_model.best_params_['C']
best_c
```

Out[123]:

```
0.23233444867279784
```

In [124]:

```
results = pd.DataFrame.from_dict(best_model.cv_results_)
results=results.sort_values('param_C')
results
```

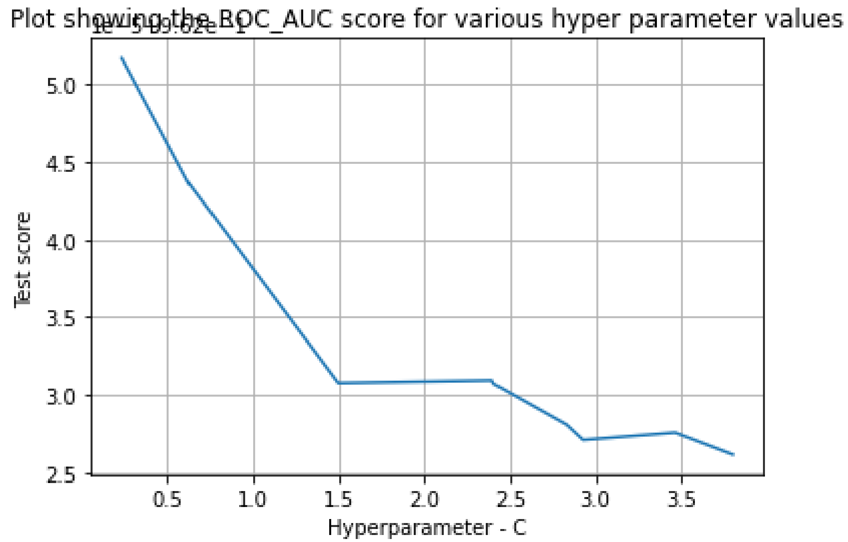
Out[124]:

	mean_fit_time	std_fit_time	mean_score_time	std_score_time	param_C	params
6	3.726326	0.130475	0.009511	0.000833	0.232334	{'C': 0.23233444867279784}
5	9.815611	0.452051	0.009016	0.000450	0.623978	{'C': 0.6239780813448106}
4	9.402443	0.604710	0.009335	0.000597	0.624075	{'C': 0.6240745617697461}
0	20.009992	0.735768	0.009698	0.000943	1.49816	{'C': 1.49816047538945}
3	19.451480	0.384670	0.013516	0.008621	2.39463	{'C': 2.3946339367881464}

Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD

	mean_fit_time	std_fit_time	mean_score_time	std_score_time	param_C	params
8	17.116975	1.498897	0.008659	0.001653	2.40446	{'C': 2.404460046972835}
9	14.348889	1.305182	0.005841	0.001067	2.83229	{'C': 2.832290311184182}
2	19.717969	0.395682	0.010328	0.002716	2.92798	{'C': 2.9279757672456204}
7	18.970401	0.625358	0.009083	0.000206	3.4647	{'C': 3.4647045830997407}
1	20.206970	0.710743	0.009152	0.000238	3.80286	{'C': 3.8028572256396647}

In [125]:
```
print_graph(results, 'param_C', 'mean_test_score', 'Hyperparameter - C'
, 'Test score')
```



In [126]:
```
#https://stackoverflow.com/questions/26478000/converting-linearsvcs-dec
ision-function-to-probabilities-scikit-learn-python
model = LinearSVC(C=best_c,verbose=verbose,random_state=random_state,cl
```

Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD

```
ass_weight='balanced',max_iter=2000)
model = CalibratedClassifierCV(model)
model.fit(train_df_rc,y_train)
```

```
[LibLinear][LibLinear][LibLinear][LibLinear][LibLinear]
```

Out[126]:

```
CalibratedClassifierCV(base_estimator=LinearSVC(C=0.23233444867279784,
                                                class_weight='balance
d',
                                                max_iter=2000, random_s
tate=42,
                                                verbose=2))
```

In [127]:

```
predictions = model.predict_proba(test_df_rc)[:,1]
save_submission(predictions, 'svm_rc.csv')
```

svm-rc

4.3 Logistic Regression with response encoding

In [128]:

```
C_val = uniform(loc=0, scale=4)
lr= LogisticRegression(verbose=verbose,random_state=random_state,class_
weight='balanced',solver='lbfgs',max_iter=500,n_jobs=-1)
parameters={'C':C_val}
clf = RandomizedSearchCV(lr,parameters,random_state=random_state,cv=cv,
verbose=verbose,n_iter=100,scoring=scoring,n_jobs=-1)
best_model = clf.fit(train_df_rc,y_train)
```

```
Fitting 5 folds for each of 100 candidates, totalling 500 fits
```

```
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 8 concurrent work
ers.
[Parallel(n_jobs=-1)]: Done  25 tasks      | elapsed:    1.8s
[Parallel(n_jobs=-1)]: Done 146 tasks      | elapsed:    9.9s
[Parallel(n_jobs=-1)]: Done 349 tasks      | elapsed:   24.2s
[Parallel(n_jobs=-1)]: Done 500 out of 500 | elapsed:   34.9s finished
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 8 concurrent work
```

Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD

```
ers.
[Parallel(n_jobs=-1)]: Done   1 out of   1 | elapsed:    0.5s finished
```

In [129]:
```
best_c=best_model.best_params_['C']
best_c
```

Out[129]: 3.8783385110582342

In [130]:
```
results = pd.DataFrame.from_dict(best_model.cv_results_)
results=results.sort_values('param_C')
results
```

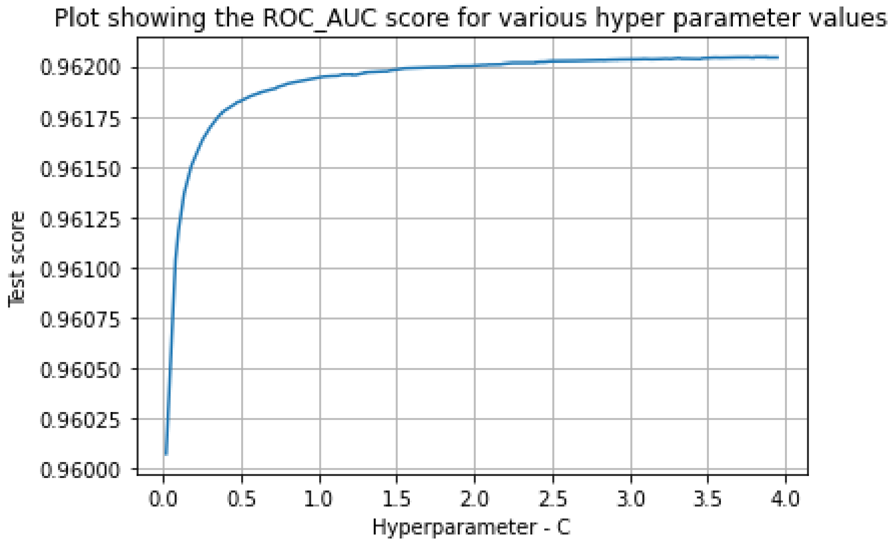
Out[130]:

	mean_fit_time	std_fit_time	mean_score_time	std_score_time	param_C	parai
72	0.419037	0.018913	0.014351	0.004590	0.0220885	{' 0.0220884684944095
10	0.377327	0.053768	0.012779	0.002567	0.082338	{' 0.0823379771832097
98	0.375209	0.049318	0.010560	0.000751	0.101677	{' 0.1016765069763807
42	0.392769	0.045860	0.014432	0.004220	0.137554	{' 0.1375540844608735
58	0.438569	0.077177	0.013336	0.003642	0.180909	{' 0.1809091556421522
...	...	...	...	...	...	
1	0.506320	0.115184	0.009774	0.001500	3.80286	{' 3.802857225639664
34	0.435505	0.063518	0.010841	0.001308	3.86253	{' 3.862528132298237
50	0.560126	0.144761	0.012199	0.003007	3.87834	{' 3.878338511058234
11	0.493065	0.118567	0.013844	0.004191	3.87964	{' 3.879639408647977
69	0.649569	0.209739	0.016016	0.005468	3.94755	{' 3.94754774640206

Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD

100 rows × 14 columns

In [131]:
```
print_graph(results, 'param_C', 'mean_test_score', 'Hyperparameter - C'
, 'Test score')
```



In [132]:
```
model = LogisticRegression(C=best_c,verbose=verbose,n_jobs=-1,random_st
ate=random_state,class_weight='balanced',solver='lbfgs')
model.fit(train_df_rc,y_train)
```

```
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 8 concurrent work
ers.
[Parallel(n_jobs=-1)]: Done   1 out of   1 | elapsed:    0.4s finished
```

Out[132]:
```
LogisticRegression(C=3.8783385110582342, class_weight='balanced', n_job
s=-1,
                   random_state=42, verbose=2)
```

In [133]:
```
predictions = model.predict_proba(test_df_rc)[:,1]
save_submission(predictions, 'lr_rc.csv')
```

Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD

lr-rc

4.4 Random Forest with response encoding

In [134]:
```
rfc = RandomForestClassifier(random_state=random_state,class_weight='ba
lanced',n_jobs=-1)
clf = RandomizedSearchCV(rfc,get_rf_params(),random_state=random_state,
cv=cv,verbose=verbose,n_iter=100,scoring=scoring,n_jobs=-1)
best_model = clf.fit(train_df_rc,y_train)
```

```
Fitting 5 folds for each of 100 candidates, totalling 500 fits
```

```
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 8 concurrent work
ers.
[Parallel(n_jobs=-1)]: Done  25 tasks      | elapsed:   19.0s
[Parallel(n_jobs=-1)]: Done 146 tasks      | elapsed:  3.0min
[Parallel(n_jobs=-1)]: Done 349 tasks      | elapsed:  7.0min
[Parallel(n_jobs=-1)]: Done 500 out of 500 | elapsed: 10.3min finished
```

In [135]:
```
results = pd.DataFrame(best_model.cv_results_)
results.sort_values('mean_test_score',ascending=False,inplace=True)
param_keys=['param_'+str(each) for each in get_rf_params().keys()]
param_keys.append('mean_test_score')
results[param_keys].head(10)
```

Out[135]:

	param_n_estimators	param_max_depth	param_max_features	param_min_samples_split	mean
68	1000	10	4	20	
26	700	12	4	20	
64	700	10	5	7	
82	700	20	5	20	
41	500	10	3	7	
96	100	10	3	10	

Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD

	param_n_estimators	param_max_depth	param_max_features	param_min_samples_split	mean
87	200	10	5	2	
11	1000	15	3	7	
85	1000	20	3	7	
25	700	15	4	7	

In [136]:
```
n_estimators=clf.best_params_['n_estimators']
max_features=clf.best_params_['max_features']
max_depth=clf.best_params_['max_depth']
min_samples_split=clf.best_params_['min_samples_split']
n_estimators,max_features,max_depth,min_samples_split
```

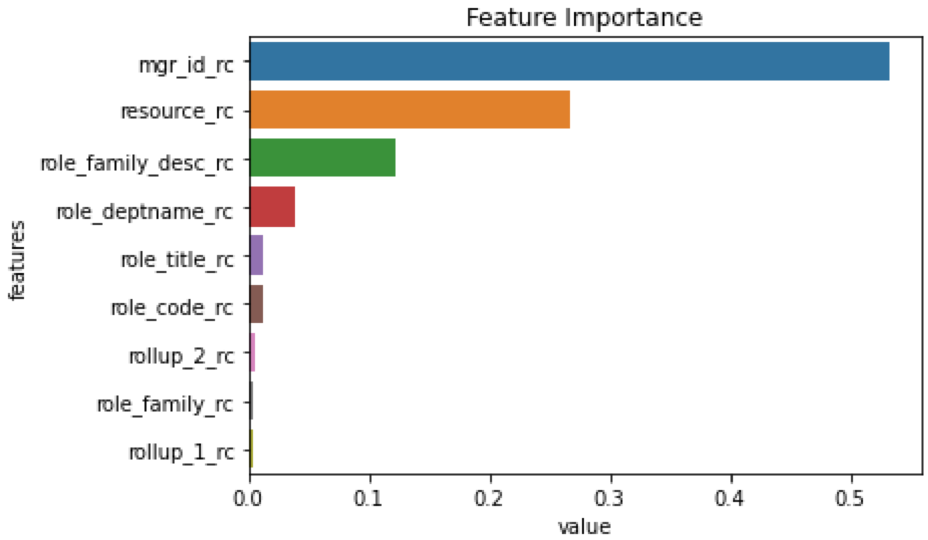
Out[136]:
```
(1000, 4, 10, 20)
```

In [137]:
```
model=RandomForestClassifier(n_estimators=n_estimators,max_depth=max_de
pth,max_features=max_features,
                             min_samples_split=min_samples_split,
                             random_state=random_state,class_weight='ba
lanced',n_jobs=-1)

model.fit(train_df_rc,y_train)
```

Out[137]:
```
RandomForestClassifier(class_weight='balanced', max_depth=10, max_featu
res=4,
                       min_samples_split=20, n_estimators=1000, n_jobs=
-1,
                       random_state=42)
```

In [138]:
```
features=train_df_rc.columns
importance=model.feature_importances_
features=pd.DataFrame({'features':features,'value':importance})
features=features.sort_values('value',ascending=False)
sns.barplot('value','features',data=features);
plt.title('Feature Importance');
```

Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD



In [139]:
```
predictions = model.predict_proba(test_df_rc)[:,1]
save_submission(predictions, 'rf_rc.csv')
```

rf-rc

4.5 Xgboost with response encoding

In [140]:
```
xgb = XGBClassifier()
clf = RandomizedSearchCV(xgb,get_xgb_params(),random_state=random_state
,cv=cv,verbose=verbose,n_iter=100,scoring=scoring,n_jobs=-1)
best_model=clf.fit(train_df_rc,y_train)
```

```
Fitting 5 folds for each of 100 candidates, totalling 500 fits
```

```
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 8 concurrent work
ers.
[Parallel(n_jobs=-1)]: Done  25 tasks      | elapsed:    7.3s
[Parallel(n_jobs=-1)]: Done 146 tasks      | elapsed:  1.5min
```

Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD

```
[Parallel(n_jobs=-1)]: Done 349 tasks      | elapsed:  4.0min
[Parallel(n_jobs=-1)]: Done 500 out of 500 | elapsed:  6.7min finished
```

In [141]:

```
results = pd.DataFrame(best_model.cv_results_)
results.sort_values('mean_test_score',ascending=False,inplace=True)
param_keys=['param_'+str(each) for each in get_xgb_params().keys()]
param_keys.append('mean_test_score')
results[param_keys].head(10)
```

Out[141]:

	param_n_estimators	param_learning_rate	param_subsample	param_max_depth	param_colsan
1	200	0.0699849	0.601115	5	
28	500	0.0141713	0.222108	5	
7	500	0.017959	0.808397	3	
98	50	0.220131	0.777147	6	
41	20	0.34312	0.996254	6	
58	50	0.454461	0.708911	3	
33	100	0.153737	0.447783	4	
74	20	0.432195	0.763364	6	
94	20	0.591191	0.618218	3	
88	200	0.307937	0.895523	3	

In [142]:

```
colsample_bytree = clf.best_params_['colsample_bytree']
learning_rate=clf.best_params_['learning_rate']
max_depth=clf.best_params_['max_depth']
min_child_weight=clf.best_params_['min_child_weight']
n_estimators=clf.best_params_['n_estimators']
subsample=clf.best_params_['subsample']
colsample_bytree,learning_rate,max_depth,min_child_weight,n_estimators,
subsample
```

Out[142]:

```
(0.44583275285359114, 0.06998494949080172, 5, 4, 200, 0.601115011743208
8)
```

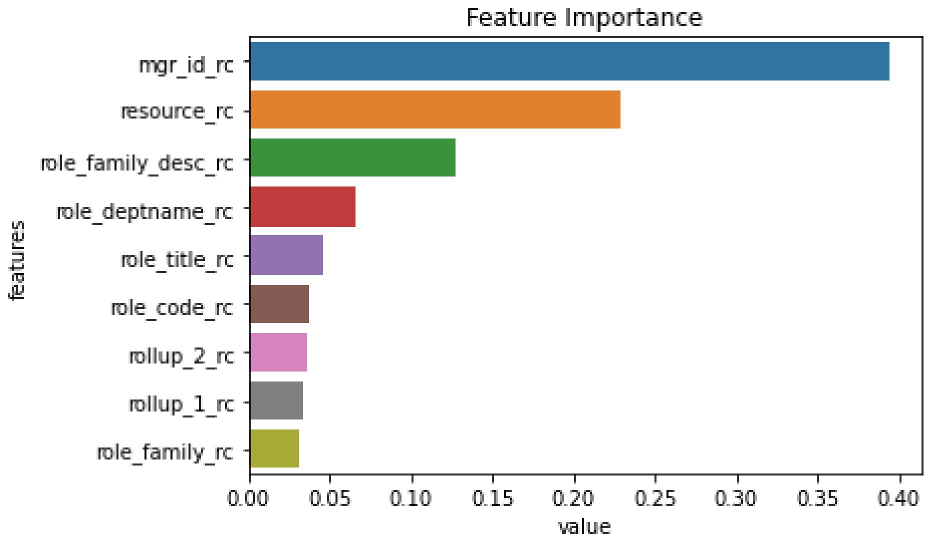
Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD

In [143]:
```
model = XGBClassifier(colsample_bytree=colsample_bytree,learning_rate=l
earning_rate,max_depth=max_depth,
                     min_child_weight=min_child_weight,n_estimators=n_e
stimators,subsample=subsample,n_jobs=-1)

model.fit(train_df_rc,y_train)
```

Out[143]:
```
XGBClassifier(base_score=0.5, booster='gbtree', colsample_bylevel=1,
              colsample_bynode=1, colsample_bytree=0.44583275285359114,
gamma=0,
              gpu_id=-1, importance_type='gain', interaction_constraint
s='',
              learning_rate=0.06998494949080172, max_delta_step=0, max_
depth=5,
              min_child_weight=4, missing=nan, monotone_constraints
='()',
              n_estimators=200, n_jobs=-1, num_parallel_tree=1, random_
state=0,
              reg_alpha=0, reg_lambda=1, scale_pos_weight=1,
              subsample=0.6011150117432088, tree_method='exact',
              validate_parameters=1, verbosity=None)
```

In [144]:
```
features=train_df_rc.columns
importance=model.feature_importances_
features=pd.DataFrame({'features':features,'value':importance})
features=features.sort_values('value',ascending=False)
sns.barplot('value','features',data=features);
plt.title('Feature Importance');
```

Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD



In [145]:
```
predictions = model.predict_proba(test_df_rc)[:,1]
save_submission(predictions, 'xgb_rc.csv')
```

xgb-rc

kaggle-submission-rc

In [146]:
```
from prettytable import PrettyTable

x = PrettyTable(['Model', 'Feature', 'Private Score', 'Public Score'])
x.add_row(['KNN','rc', 0.84352, 0.85351])
x.add_row(['SVM', 'rc', 0.85160, 0.86031])
x.add_row(['Logistic Regression', 'rc', 0.85322, 0.86180])
x.add_row(['Random Forest', 'rc', 0.83136, 0.83892])
x.add_row(['Xgboost', 'rc', 0.84135, 0.84190])

print(x)
```

```
+---------------------+---------+---------------+--------------+
|        Model        | Feature | Private Score | Public Score |
```

Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD

```
+--------------------+---------+---------------+-------------+
|        KNN         |    rc   |    0.84352    |   0.85351   |
|        SVM         |    rc   |     0.8516    |   0.86031   |
| Logistic Regression |    rc   |    0.85322    |    0.8618   |
|    Random Forest   |    rc   |    0.83136    |   0.83892   |
|       Xgboost      |    rc   |    0.84135    |    0.8419   |
+--------------------+---------+---------------+-------------+
```

Observations:

1. Every model performs good for this feature
2. Linear models performs better than Tree based models

5 Build model on SVD feature

In [147]:
```
train_svd = pd.read_csv('data/train_svd.csv')
test_svd = pd.read_csv('data/test_svd.csv')
```

In [148]:
```
train_svd.shape, test_svd.shape, y_train.shape
```

Out[148]:
```
((32769, 72), (58921, 72), (32769,))
```

5.1 KNN with SVD

In [149]:
```
parameters={'n_neighbors':np.arange(1,100, 5)}
clf = RandomizedSearchCV(KNeighborsClassifier(n_jobs=-1),parameters,ran
dom_state=random_state,cv=cv,verbose=verbose,scoring=scoring,n_jobs=-1)
best_model = clf.fit(train_svd,y_train)
```

```
Fitting 5 folds for each of 10 candidates, totalling 50 fits
```

```
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 8 concurrent wo
rkers.
```

Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD

```
[Parallel(n_jobs=-1)]: Done  25 tasks      | elapsed:   19.0s
[Parallel(n_jobs=-1)]: Done  50 out of  50 | elapsed:   38.0s finishe
d
```

In [150]:

```python
results = pd.DataFrame.from_dict(best_model.cv_results_)
results=results.sort_values('param_n_neighbors')
results
```

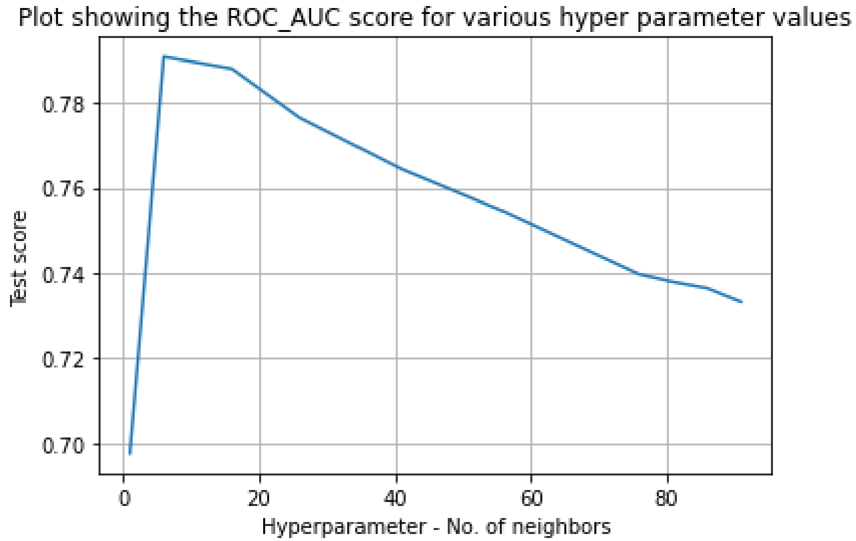
Out[150]:

	mean_fit_time	std_fit_time	mean_score_time	std_score_time	param_n_neighbors	param
0	0.526564	0.011169	1.004034	0.149404	1	{'n_neighbors
3	1.944336	0.551025	1.655248	0.153292	6	{'n_neighbors (
7	2.016851	0.954735	3.149633	0.351339	16	{'n_neighbors 1(
5	2.240037	0.818733	3.354217	0.898707	26	{'n_neighbors 2(
4	1.967969	0.228458	3.715187	1.218701	41	{'n_neighbors 4
6	1.281747	0.189357	5.637847	0.125865	56	{'n_neighbors 5(
2	2.475243	0.578326	5.100802	0.859254	76	{'n_neighbors 7(
9	2.189955	1.122140	3.499732	1.791585	81	{'n_neighbors 8
1	1.204084	0.808660	5.773037	0.319467	86	{'n_neighbors 8(
8	0.703830	0.153865	7.607220	0.172469	91	{'n_neighbors 9

In [151]:

```python
print_graph(results, 'param_n_neighbors', 'mean_test_score', 'Hyperpara
meter - No. of neighbors', 'Test score')
```

Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD



In [152]:
```
best_c=best_model.best_params_['n_neighbors']
best_c
```

Out[152]: 6

In [153]:
```
model = KNeighborsClassifier(n_neighbors=best_c,n_jobs=-1)
model.fit(train_svd,y_train)
```

Out[153]: KNeighborsClassifier(n_jobs=-1, n_neighbors=6)

In [154]:
```
predictions = model.predict_proba(test_svd)[:,1]
save_submission(predictions, "knn_svd.csv")
```

knn-svd

5.2 SVM with SVD

In [155]:
```
C_val = uniform(loc=0, scale=4)
```

Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD

```
model= LinearSVC(verbose=verbose,random_state=random_state,class_weight
='balanced',max_iter=2000)
parameters={'C':C_val}
clf = RandomizedSearchCV(model,parameters,random_state=random_state,cv=
cv,verbose=verbose,scoring=scoring,n_jobs=-1)
best_model = clf.fit(train_svd,y_train)
```

```
Fitting 5 folds for each of 10 candidates, totalling 50 fits
```

```
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 8 concurrent work
ers.
[Parallel(n_jobs=-1)]: Done  25 tasks      | elapsed:  1.5min
[Parallel(n_jobs=-1)]: Done  50 out of  50 | elapsed:  2.7min finished
```

```
[LibLinear]
```

In [156]:

```
best_c=best_model.best_params_['C']
best_c
```

Out[156]: 3.8028572256396647

In [157]:

```
results = pd.DataFrame.from_dict(best_model.cv_results_)
results=results.sort_values('param_C')
results
```

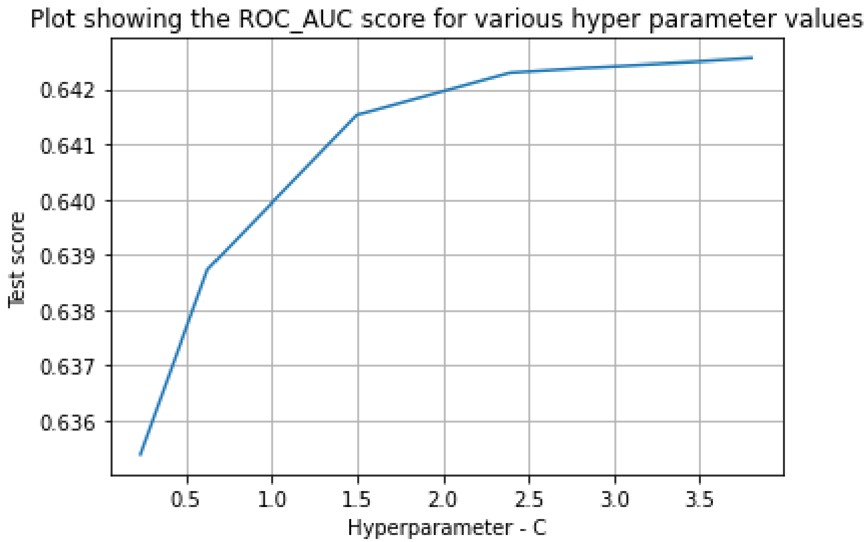
Out[157]:

	mean_fit_time	std_fit_time	mean_score_time	std_score_time	param_C	params
6	2.968880	0.150177	0.016558	0.002776	0.232334	{'C': 0.23233444867279784}
5	6.587434	0.301619	0.012963	0.000876	0.623978	{'C': 0.6239780813448106}
4	6.343227	0.065596	0.012513	0.000442	0.624075	{'C': 0.6240745617697461}
0	17.291072	1.080804	0.016185	0.007285	1.49816	{'C': 1.49816047538945}
3	29.455153	0.767611	0.015290	0.003827	2.39463	{'C': 2.3946339367881464}

Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD

	mean_fit_time	std_fit_time	mean_score_time	std_score_time	param_C	params
8	29.985117	1.952692	0.012303	0.002009	2.40446	{'C': 2.404460046972835}
9	27.816009	2.354661	0.007257	0.000748	2.83229	{'C': 2.832290311184182}
2	36.411890	2.210566	0.012634	0.000489	2.92798	{'C': 2.9279757672456204}
7	44.704208	1.334583	0.012921	0.000702	3.4647	{'C': 3.4647045830997407}
1	48.909459	2.384907	0.016626	0.005413	3.80286	{'C': 3.8028572256396647}

In [158]:
```
print_graph(results, 'param_C', 'mean_test_score', 'Hyperparameter - C'
, 'Test score')
```



In [159]:
```
#https://stackoverflow.com/questions/26478000/converting-linearsvcs-dec
ision-function-to-probabilities-scikit-learn-python
model = LinearSVC(C=best_c,verbose=verbose,random_state=random_state,cl
```

Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD

```
ass_weight='balanced',max_iter=2000)
model = CalibratedClassifierCV(model)
model.fit(train_svd,y_train)
```

```
[LibLinear][LibLinear][LibLinear][LibLinear][LibLinear]
```

Out[159]:

```
CalibratedClassifierCV(base_estimator=LinearSVC(C=3.8028572256396647,
                                                class_weight='balance
d',
                                                max_iter=2000, random_s
tate=42,
                                                verbose=2))
```

In [160]:

```
predictions = model.predict_proba(test_svd)[:,1]
save_submission(predictions, 'svm_svd.csv')
```

svm-svd

5.3 Logistic Regression with SVD

In [161]:

```
C_val = uniform(loc=0, scale=4)
lr= LogisticRegression(verbose=verbose,random_state=random_state,class_
weight='balanced',solver='lbfgs',max_iter=500,n_jobs=-1)
parameters={'C':C_val}
clf = RandomizedSearchCV(lr,parameters,random_state=random_state,cv=cv,
verbose=verbose,n_iter=100,scoring=scoring,n_jobs=-1)
best_model = clf.fit(train_svd,y_train)
```

```
Fitting 5 folds for each of 100 candidates, totalling 500 fits
```

```
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 8 concurrent work
ers.
[Parallel(n_jobs=-1)]: Done  25 tasks      | elapsed:   18.9s
[Parallel(n_jobs=-1)]: Done 146 tasks      | elapsed:  1.4min
[Parallel(n_jobs=-1)]: Done 349 tasks      | elapsed:  3.4min
[Parallel(n_jobs=-1)]: Done 500 out of 500 | elapsed:  5.1min finished
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 8 concurrent work
```

Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD

```
ers.
[Parallel(n_jobs=-1)]: Done   1 out of   1 | elapsed:   14.7s finished
```

In [162]:
```
best_c=best_model.best_params_['C']
best_c
```

Out[162]: 3.947547746402069

In [163]:
```
results = pd.DataFrame.from_dict(best_model.cv_results_)
results=results.sort_values('param_C')
results
```

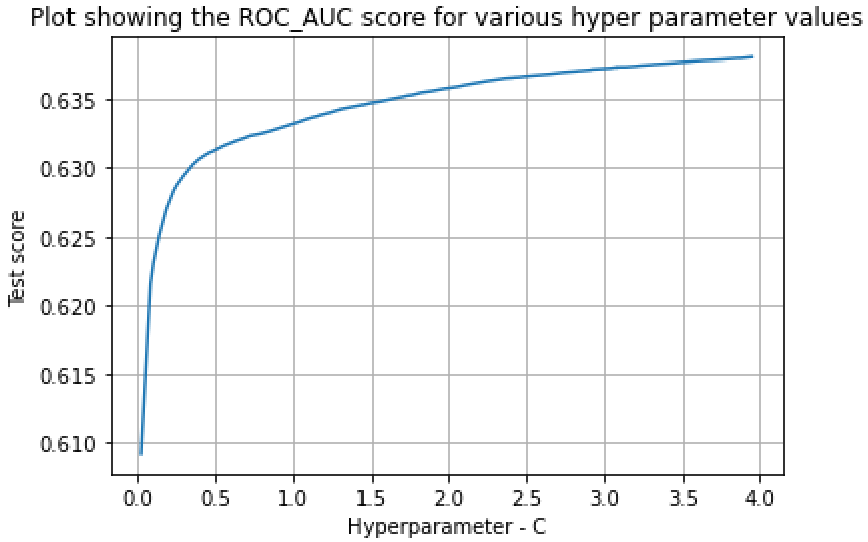
Out[163]:

	mean_fit_time	std_fit_time	mean_score_time	std_score_time	param_C	parai
72	1.042337	0.102578	0.035350	0.017093	0.0220885	{' 0.022088468494409
10	1.598916	0.145812	0.029739	0.011829	0.082338	{' 0.08233797718320
98	1.946853	0.134926	0.031715	0.005840	0.101677	{' 0.10167650697638
42	1.629183	0.083901	0.022801	0.003346	0.137554	{' 0.13755408446087
58	1.859177	0.136240	0.025097	0.007581	0.180909	{' 0.18090915564215
...	...	...	...	...	...	
1	7.015733	0.438911	0.026228	0.005973	3.80286	{' 3.80285722563966
34	6.146194	0.345854	0.028273	0.008512	3.86253	{' 3.86252813229823
50	6.477816	0.289469	0.026133	0.013208	3.87834	{' 3.87833851105823
11	6.560687	0.689193	0.033242	0.007402	3.87964	{' 3.87963940864797
69	7.289347	0.445322	0.030326	0.003518	3.94755	{' 3.94754774640206

Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD

100 rows × 14 columns

In [164]:
```
print_graph(results, 'param_C', 'mean_test_score', 'Hyperparameter - C'
, 'Test score')
```



In [165]:
```
model = LogisticRegression(C=best_c,verbose=verbose,n_jobs=-1,random_st
ate=random_state,class_weight='balanced',solver='lbfgs')
model.fit(train_svd,y_train)
```

```
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 8 concurrent work
ers.
[Parallel(n_jobs=-1)]: Done   1 out of   1 | elapsed:    3.5s finished
```

Out[165]:
```
LogisticRegression(C=3.947547746402069, class_weight='balanced', n_jobs
=-1,
                   random_state=42, verbose=2)
```

In [166]:
```
predictions = model.predict_proba(test_svd)[:,1]
save_submission(predictions, 'lr_svd.csv')
```

Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD

lr-svd

5.4 Random Forest with SVD

In [167]:
```
rfc = RandomForestClassifier(random_state=random_state,class_weight='ba
lanced',n_jobs=-1)
clf = RandomizedSearchCV(rfc,get_rf_params(),random_state=random_state,
cv=cv,verbose=verbose,n_iter=100,scoring=scoring,n_jobs=-1)
best_model = clf.fit(train_svd,y_train)
```

```
Fitting 5 folds for each of 100 candidates, totalling 500 fits
```

```
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 8 concurrent work
ers.
[Parallel(n_jobs=-1)]: Done  25 tasks      | elapsed:   36.2s
/home/auw-mayank/.local/lib/python3.6/site-packages/joblib/externals/lo
ky/process_executor.py:691: UserWarning: A worker stopped while some jo
bs were given to the executor. This can be caused by a too short worker
timeout or by a memory leak.
  "timeout or by a memory leak.", UserWarning
[Parallel(n_jobs=-1)]: Done 146 tasks      | elapsed:  6.8min
[Parallel(n_jobs=-1)]: Done 349 tasks      | elapsed: 16.5min
[Parallel(n_jobs=-1)]: Done 500 out of 500 | elapsed: 24.5min finished
```

In [168]:
```
results = pd.DataFrame(best_model.cv_results_)
results.sort_values('mean_test_score',ascending=False,inplace=True)
param_keys=['param_'+str(each) for each in get_rf_params().keys()]
param_keys.append('mean_test_score')
results[param_keys].head(10)
```

Out[168]:

	param_n_estimators	param_max_depth	param_max_features	param_min_samples_split	mean
84	1000	25	5	2	
20	1000	25	3	2	
33	700	25	4	2	

Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD

	param_n_estimators	param_max_depth	param_max_features	param_min_samples_split	mean
22	200	25	4	10	
78	700	25	2	7	
85	1000	20	3	7	
62	500	25	3	5	
82	700	20	5	20	
79	500	25	1	10	
92	500	20	3	2	

In [169]:

```
n_estimators=clf.best_params_['n_estimators']
max_features=clf.best_params_['max_features']
max_depth=clf.best_params_['max_depth']
min_samples_split=clf.best_params_['min_samples_split']
n_estimators,max_features,max_depth,min_samples_split
```

Out[169]:

```
(1000, 5, 25, 2)
```

In [170]:

```
model=RandomForestClassifier(n_estimators=n_estimators,max_depth=max_de
pth,max_features=max_features,
                             min_samples_split=min_samples_split,
                             random_state=random_state,class_weight='ba
lanced',n_jobs=-1)

model.fit(train_svd,y_train)
```

Out[170]:

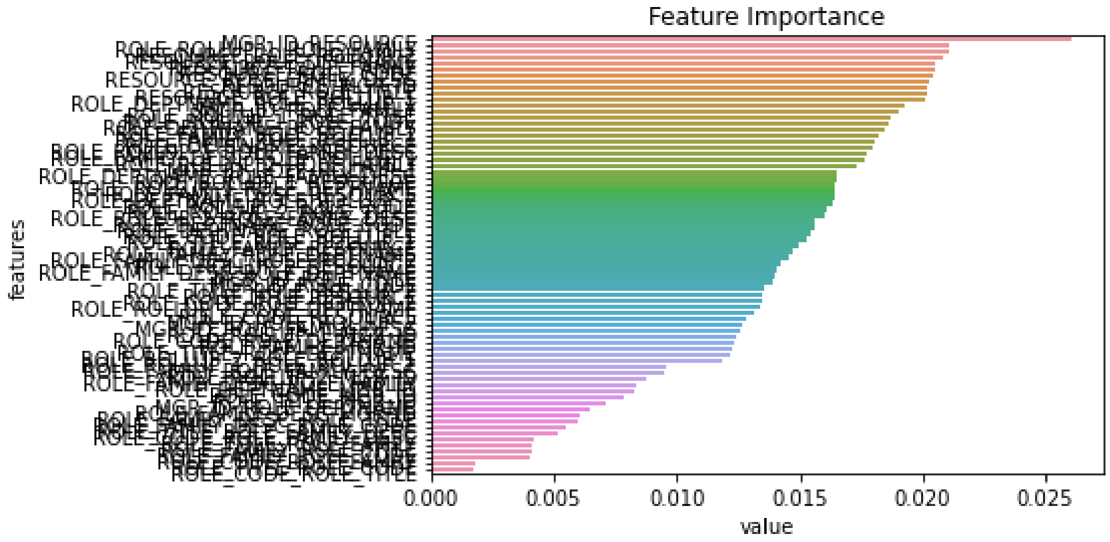
```
RandomForestClassifier(class_weight='balanced', max_depth=25, max_featu
res=5,
                       n_estimators=1000, n_jobs=-1, random_state=42)
```

In [171]:

```
features=train_svd.columns
importance=model.feature_importances_
features=pd.DataFrame({'features':features,'value':importance})
features=features.sort_values('value',ascending=False)
```

Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD

```
sns.barplot('value','features',data=features);
plt.title('Feature Importance');
```



In [172]:
```
predictions = model.predict_proba(test_svd)[:,1]
save_submission(predictions, 'rf_svd.csv')
```

rf-svd

5.5 Xgboost with SVD

In [173]:
```
xgb = XGBClassifier()
clf = RandomizedSearchCV(xgb,get_xgb_params(),random_state=random_state
,cv=cv,verbose=verbose,n_iter=100,scoring=scoring,n_jobs=-1)
best_model=clf.fit(train_svd,y_train)
```

```
Fitting 5 folds for each of 100 candidates, totalling 500 fits
```

```
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 8 concurrent work
ers.
/home/auw-mayank/.local/lib/python3.6/site-packages/joblib/externals/lo
```

Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD

```
ky/process_executor.py:691: UserWarning: A worker stopped while some jo
bs were given to the executor. This can be caused by a too short worker
timeout or by a memory leak.
  "timeout or by a memory leak.", UserWarning
[Parallel(n_jobs=-1)]: Done  25 tasks      | elapsed:   54.6s
[Parallel(n_jobs=-1)]: Done 146 tasks      | elapsed:  9.9min
[Parallel(n_jobs=-1)]: Done 349 tasks      | elapsed: 28.0min
[Parallel(n_jobs=-1)]: Done 500 out of 500 | elapsed: 50.8min finished
```

In [174]:

```
results = pd.DataFrame(best_model.cv_results_)
results.sort_values('mean_test_score',ascending=False,inplace=True)
param_keys=['param_'+str(each) for each in get_xgb_params().keys()]
param_keys.append('mean_test_score')
results[param_keys].head(10)
```

Out[174]:

	param_n_estimators	param_learning_rate	param_subsample	param_max_depth	param_colsan
62	500	0.0663892	0.328153	9	
18	1000	0.048135	0.665922	9	
96	500	0.0979629	0.98664	7	
44	1000	0.060484	0.606429	6	
8	750	0.0686033	0.683264	6	
97	750	0.232385	0.907694	6	
49	500	0.160277	0.393098	8	
80	1000	0.385564	0.905351	3	
53	200	0.540096	0.928319	6	
78	1000	0.576551	0.94023	6	

In [175]:

```
colsample_bytree = clf.best_params_['colsample_bytree']
learning_rate=clf.best_params_['learning_rate']
max_depth=clf.best_params_['max_depth']
min_child_weight=clf.best_params_['min_child_weight']
```

Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD

```
n_estimators=clf.best_params_['n_estimators']
subsample=clf.best_params_['subsample']
colsample_bytree,learning_rate,max_depth,min_child_weight,n_estimators,
subsample
```

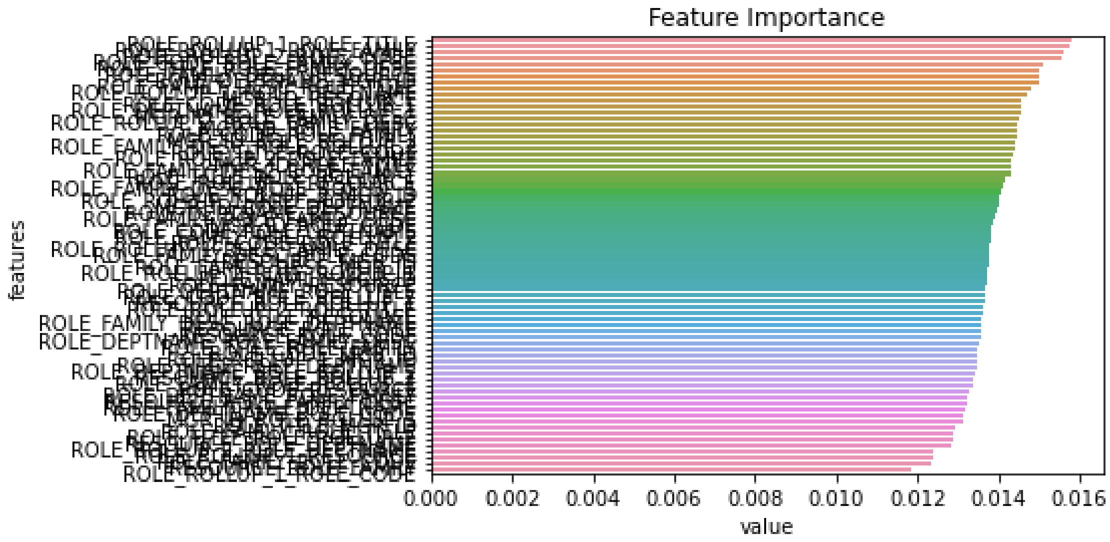
Out[175]:
```
(0.375582952639944, 0.06638916390452139, 9, 3, 500, 0.3281526674747319
3)
```

In [176]:
```
model = XGBClassifier(colsample_bytree=colsample_bytree,learning_rate=l
earning_rate,max_depth=max_depth,
                     min_child_weight=min_child_weight,n_estimators=n_e
stimators,subsample=subsample,n_jobs=-1)

model.fit(train_svd,y_train)
```

Out[176]:
```
XGBClassifier(base_score=0.5, booster='gbtree', colsample_bylevel=1,
              colsample_bynode=1, colsample_bytree=0.375582952639944, g
amma=0,
              gpu_id=-1, importance_type='gain', interaction_constraint
s='',
              learning_rate=0.06638916390452139, max_delta_step=0, max_
depth=9,
              min_child_weight=3, missing=nan, monotone_constraints
='()',
              n_estimators=500, n_jobs=-1, num_parallel_tree=1, random_
state=0,
              reg_alpha=0, reg_lambda=1, scale_pos_weight=1,
              subsample=0.32815266747473193, tree_method='exact',
              validate_parameters=1, verbosity=None)
```

In [177]:
```
features=train_svd.columns
importance=model.feature_importances_
features=pd.DataFrame({'features':features,'value':importance})
features=features.sort_values('value',ascending=False)
sns.barplot('value','features',data=features);
plt.title('Feature Importance');
```

Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD



In [178]:
```
predictions = model.predict_proba(test_svd)[:,1]
save_submission(predictions, 'xgb_svd.csv')
```

xgb-svd

kaggle-submission-svd

In [179]:
```
from prettytable import PrettyTable

x = PrettyTable(['Model', 'Feature', 'Private Score', 'Public Score'])
x.add_row(['KNN','svd', 0.79245, 0.78572])
x.add_row(['SVM', 'svd', 0.63648, 0.63806])
x.add_row(['Logistic Regression', 'svd', 0.63255, 0.63314])
x.add_row(['Random Forest', 'svd', 0.87119, 0.86924])
x.add_row(['Xgboost', 'svd', 0.86909, 0.86664])

print(x)
```

```
+---------------------+---------+---------------+--------------+
|        Model        | Feature | Private Score | Public Score |
```

Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD

```
+--------------------+---------+--------------+-------------+
|        KNN         |   svd   |   0.79245    |   0.78572   |
|        SVM         |   svd   |   0.63648    |   0.63806   |
| Logistic Regression |  svd   |   0.63255    |   0.63314   |
|   Random Forest    |   svd   |   0.87119    |   0.86924   |
|      Xgboost       |   svd   |   0.86909    |   0.86664   |
+--------------------+---------+--------------+-------------+
```

Observations:

1. Tree based models works better than linear model
2. KNN is performing overall good

We have to improve our model to reach into 5-10% on kaggle

In [180]:
```
# https://www.kaggle.com/mitribunskiy/tutorial-catboost-overview
```

In [181]:
```
# https://www.kaggle.com/prashant111/catboost-classifier-tutorial
```

https://catboost.ai/

CatBoost is a high-performance open source library for gradient boosting on decision trees

About

CatBoost is an algorithm for gradient boosting on decision trees. It is developed by Yandex researchers and engineers, and is used for search, recommendation systems, personal assistant, self-driving cars, weather prediction and many other tasks at Yandex and in other

Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD

companies, including CERN, Cloudflare, Careem taxi. It is in open-source and can be used by anyone.

Features

1. Reduce time spent on parameter tuning, because CatBoost provides great results with default parameters
2. Improve your training results with CatBoost that allows you to use non-numeric factors, instead of having to pre-process your data or spend time and effort turning it to numbers.
3. Reduce overfitting when constructing your models with a novel gradient-boosting scheme.
4. Apply your trained model quickly and efficiently even to latency-critical tasks using CatBoost's model applier

In [182]:
```
params = {
            'loss_function':'Logloss',
            'eval_metric':'AUC',
            'cat_features':list(range(train_data.shape[1])),
            'verbose':100,
            'random_seed':random_state
        }
```

In [183]:
```
clf= CatBoostClassifier(**params)
clf.fit(train_data,y_train)
```

```
Learning rate set to 0.045713
0:      total: 99.2ms   remaining: 1m 39s
100:    total: 2.33s    remaining: 20.7s
200:    total: 5.6s     remaining: 22.3s
300:    total: 8.79s    remaining: 20.4s
400:    total: 11.9s    remaining: 17.8s
500:    total: 15.2s    remaining: 15.2s
600:    total: 18.4s    remaining: 12.2s
700:    total: 21.7s    remaining: 9.27s
800:    total: 24.8s    remaining: 6.16s
900:    total: 28s      remaining: 3.08s
999:    total: 31.1s    remaining: 0us
```

Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD

Out[183]:
```
<catboost.core.CatBoostClassifier at 0x7f361d4d8780>
```

In [184]:
```
predictions = clf.predict_proba(test_data)[:,1]
```

In [185]:
```
save_submission(predictions, 'catboost.csv')
```

catboost

Catboost perform better than all our previous models and it's AUC score is much better than previous models so I am selecting this for predicting future data

In [186]:
```
# Save model on disk
pickle.dump(clf, open('models/catboost_model.pkl', 'wb'))
```

In [ ]:
```

```

Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD