Diabetes Risk Prediction of Patient Readmission - Using Python

Part A

In [2]:
```
import math
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
import time
```

In [3]:
```
#load the CSV file
df = pd.read_csv('diabetic_data.csv')
print('Number of samples: ',len(df))
```

```
Number of samples:  101766
```

In [4]:
```
df.head(10)
```

Out[4]:

counter_id	patient_nbr	race	gender	age	weight	admission_type_id	discharge_dispositi
2278392	8222157	Caucasian	Female	[0-10)	?	6	
149190	55629189	Caucasian	Female	[10-20)	?	1	
64410	86047875	AfricanAmerican	Female	[20-30)	?	1	
500364	82442376	Caucasian	Male	[30-40)	?	1	
16680	42519267	Caucasian	Male	[40-50)	?	1	
35754	82637451	Caucasian	Male	[50-60)	?	2	
55842	84259809	Caucasian	Male	[60-70)	?	3	
63768	114882984	Caucasian	Male	[70-80)	?	1	
12522	48330783	Caucasian	Female	[80-90)	?	2	
15738	63555939	Caucasian	Female	[90-100)	?	3	

/s × 50 columns

In [5]:
```
#count of the number of rows for each type
df.groupby('readmitted').size()
```

Out[5]:
```
readmitted
<30    11357
>30    35545
NO     54864
dtype: int64
```

In [6]:
```
#11,13,14,19,20,21 are related to death or hospice.
df=df.loc[~df.discharge_disposition_id.isin([11,13,14,19,20,21])]
```

In [7]:
```
#we will try to predict if a patient is likely to be re-admitted within 30 days o
df['OUTPUT_LABEL'] = (df.readmitted == '<30').astype('int')
```

In [8]:
```
def calc_prevalence(y_actual):
    return (sum(y_actual)/len(y_actual))
```

In [9]:
```
print('Prevalence:%.3f'%calc_prevalence(df['OUTPUT_LABEL'].values))
```

```
Prevalence:0.114
```

In [10]:
```
#replace ? with nan
df=df.replace('?',np.nan)
df.head(10)
```

Out[10]:

	encounter_id	patient_nbr	race	gender	age	weight	admission_type_id	discharge_c
0	2278392	8222157	Caucasian	Female	[0-10)	NaN	6	
1	149190	55629189	Caucasian	Female	[10-20)	NaN	1	
2	64410	86047875	AfricanAmerican	Female	[20-30)	NaN	1	
3	500364	82442376	Caucasian	Male	[30-40)	NaN	1	
4	16680	42519267	Caucasian	Male	[40-50)	NaN	1	
5	35754	82637451	Caucasian	Male	[50-60)	NaN	2	
6	55842	84259809	Caucasian	Male	[60-70)	NaN	3	
7	63768	114882984	Caucasian	Male	[70-80)	NaN	1	
8	12522	48330783	Caucasian	Female	[80-90)	NaN	2	
9	15738	63555939	Caucasian	Female	[90-100)	NaN	3	

10 rows × 51 columns

In [19]:
```
#The columns that are numerical
cols_num = ['time_in_hospital','num_lab_procedures', 'num_procedures','num_medic
```

In [20]:
```
df[cols_num].isnull().sum()
```

Out[20]:
```
time_in_hospital      0
num_lab_procedures    0
num_procedures        0
num_medications       0
number_outpatient     0
number_emergency      0
number_inpatient      0
number_diagnoses      0
dtype: int64
```

In [40]:
```
#Categorical variables are non-numeric data such as race and gender
cat_cols = ['race', 'gender', 'max_glu_serum', 'A1Cresult', 'metformin', 'repagl
```

In [23]:
```
#addition for another category 'UNK'
df['race']=df['race'].fillna('UNK')
df['payer_code']=df['payer_code'].fillna('UNK')
df['medical_specialty']=df['medical_specialty'].fillna('UNK')
```

In [25]:
```
print('Number medical specialty: ',df.medical_specialty.nunique())
df.groupby('medical_specialty').size().sort_values(ascending=False)
```

```
Number medical specialty:  73
```

Out[25]:
```
medical_specialty
UNK                                 48616
InternalMedicine                    14237
Emergency/Trauma                     7419
Family/GeneralPractice               7252
Cardiology                           5279
                                    ...
Psychiatry-Addictive                    1
Dermatology                             1
Speech                                  1
SportsMedicine                          1
Surgery-PlasticwithinHeadandNeck        1
Length: 73, dtype: int64
```

In [30]:
```
#Consider top 10 count from above 73
top_10=['UNK','InternalMedicine', 'Emergency/Trauma', 'Family/GeneralPractice',
#make a new column with duplicated data
df['med_spec']=df['medical_specialty'].copy()
#replace all specialties not in top 10 with 'other' category
df.loc[~df.med_spec.isin(top_10),'med_spec']='other'
```

In [32]:
```
#created a category other is added so we have 11 in total
df.groupby('med_spec').size().sort_values(ascending=False)
```

Out[32]:
```
med_spec
UNK                             48616
InternalMedicine                14237
other                            8199
Emergency/Trauma                 7419
Family/GeneralPractice           7252
Cardiology                       5279
Surgery-General                  3059
Nephrology                       1539
Orthopedics                      1392
Orthopedics-Reconstructive       1230
Radiologist                      1121
dtype: int64
```

In [33]:
```
#The get_dummies function does not work on numerical data
cols_cat_num=['admission_type_id','discharge_disposition_id','admission_source_i
df[cols_cat_num]=df[cols_cat_num].astype('str')
```

In [41]:
```
#performed one-hot encoding(dummy variables)
df_cat=pd.get_dummies(df[cat_cols+cols_cat_num+['med_spec']],drop_first=True)
df_cat.head(10)
```

Out[41]:

	race_Asian	race_Caucasian	race_Hispanic	race_Other	race_UNK	gender_Male	gender_Unknov
0	0	1	0	0	0	0	
1	0	1	0	0	0	0	
2	0	0	0	0	0	0	
3	0	1	0	0	0	1	
4	0	1	0	0	0	1	
5	0	1	0	0	0	1	
6	0	1	0	0	0	1	
7	0	1	0	0	0	1	
8	0	1	0	0	0	0	
9	0	1	0	0	0	0	

10 rows × 133 columns

In [42]:
```
df=pd.concat([df,df_cat],axis=1)
df.head(10)
```

In [43]:
```
cols_all_cat=list(df_cat.columns)
cols_all_cat
```

Out[43]:
```
['race_Asian',
 'race_Caucasian',
 'race_Hispanic',
 'race_Other',
 'race_UNK',
 'gender_Male',
 'gender_Unknown/Invalid',
 'max_glu_serum_>300',
 'max_glu_serum_None',
 'max_glu_serum_Norm',
 'A1Cresult_>8',
 'A1Cresult_None',
 'A1Cresult_Norm',
 'metformin_No',
 'metformin_Steady',
 'metformin_Up',
 'repaglinide_No',
 'repaglinide_Steady',
 'repaglinide_Up',
 'nateglinide_No',
 'nateglinide_Steady',
 'nateglinide_Up',
 'chlorpropamide_No',
 'chlorpropamide_Steady',
 'chlorpropamide_Up',
 'glimepiride_No',
 'glimepiride_Steady',
 'glimepiride_Up',
 'acetohexamide_Steady',
 'glipizide_No',
 'glipizide_Steady',
 'glipizide_Up',
 'glyburide_No',
 'glyburide_Steady',
 'glyburide_Up',
 'tolbutamide_Steady',
 'pioglitazone_No',
 'pioglitazone_Steady',
 'pioglitazone_Up',
 'rosiglitazone_No',
 'rosiglitazone_Steady',
 'rosiglitazone_Up',
 'acarbose_No',
 'acarbose_Steady',
 'acarbose_Up',
 'miglitol_No',
 'miglitol_Steady',
 'miglitol_Up',
 'troglitazone_Steady',
 'tolazamide_Steady',
 'tolazamide_Up',
 'insulin_No',
 'insulin_Steady',
```

```
'insulin_Up',
'glyburide-metformin_No',
'glyburide-metformin_Steady',
'glyburide-metformin_Up',
'glipizide-metformin_Steady',
'glimepiride-pioglitazone_Steady',
'metformin-rosiglitazone_Steady',
'metformin-pioglitazone_Steady',
'change_No',
'diabetesMed_Yes',
'payer_code_CH',
'payer_code_CM',
'payer_code_CP',
'payer_code_DM',
'payer_code_FR',
'payer_code_HM',
'payer_code_MC',
'payer_code_MD',
'payer_code_MP',
'payer_code_OG',
'payer_code_OT',
'payer_code_PO',
'payer_code_SI',
'payer_code_SP',
'payer_code_UN',
'payer_code_UNK',
'payer_code_WC',
'admission_type_id_2',
'admission_type_id_3',
'admission_type_id_4',
'admission_type_id_5',
'admission_type_id_6',
'admission_type_id_7',
'admission_type_id_8',
'discharge_disposition_id_10',
'discharge_disposition_id_12',
'discharge_disposition_id_15',
'discharge_disposition_id_16',
'discharge_disposition_id_17',
'discharge_disposition_id_18',
'discharge_disposition_id_2',
'discharge_disposition_id_22',
'discharge_disposition_id_23',
'discharge_disposition_id_24',
'discharge_disposition_id_25',
'discharge_disposition_id_27',
'discharge_disposition_id_28',
'discharge_disposition_id_3',
'discharge_disposition_id_4',
'discharge_disposition_id_5',
'discharge_disposition_id_6',
'discharge_disposition_id_7',
'discharge_disposition_id_8',
'discharge_disposition_id_9',
'admission_source_id_10',
'admission_source_id_11',
'admission_source_id_13',
```

```
 'admission_source_id_14',
 'admission_source_id_17',
 'admission_source_id_2',
 'admission_source_id_20',
 'admission_source_id_22',
 'admission_source_id_25',
 'admission_source_id_3',
 'admission_source_id_4',
 'admission_source_id_5',
 'admission_source_id_6',
 'admission_source_id_7',
 'admission_source_id_8',
 'admission_source_id_9',
 'med_spec_Emergency/Trauma',
 'med_spec_Family/GeneralPractice',
 'med_spec_InternalMedicine',
 'med_spec_Nephrology',
 'med_spec_Orthopedics',
 'med_spec_Orthopedics-Reconstructive',
 'med_spec_Radiologist',
 'med_spec_Surgery-General',
 'med_spec_UNK',
 'med_spec_other']
```

In [44]:
```
df[['age','weight']].head(10)
```

Out[44]:

	age	weight
0	[0-10)	NaN
1	[10-20)	NaN
2	[20-30)	NaN
3	[30-40)	NaN
4	[40-50)	NaN
5	[50-60)	NaN
6	[60-70)	NaN
7	[70-80)	NaN
8	[80-90)	NaN
9	[90-100)	NaN

In [46]:
```
df.groupby('age').size().sort_values(ascending=False)
```

Out[46]:
```
age
[70-80)     25331
[60-70)     22059
[50-60)     17060
[80-90)     16434
[40-50)      9607
[30-40)      3764
[90-100)     2589
[20-30)      1649
[10-20)       690
[0-10)        160
dtype: int64
```

In [47]:
```
#mapping the category range to category
age_id={'[0-10)':0,
       '[10-20)':10,
       '[20-30)':20,
       '[30-40)':30,
       '[40-50)':40,
       '[50-60)':50,
       '[60-70)':60,
       '[70-80)':70,
       '[80-90)':80,
       '[90-100)':90}
```

In [48]:
```
df['age_group']=df.age.replace(age_id)
df.head(10)
```

Out[48]:

	encounter_id	patient_nbr	race	gender	age	weight	admission_type_id	discharge_
0	2278392	8222157	Caucasian	Female	[0-10)	NaN	6	
1	149190	55629189	Caucasian	Female	[10-20)	NaN	1	
2	64410	86047875	AfricanAmerican	Female	[20-30)	NaN	1	
3	500364	82442376	Caucasian	Male	[30-40)	NaN	1	
4	16680	42519267	Caucasian	Male	[40-50)	NaN	1	
5	35754	82637451	Caucasian	Male	[50-60)	NaN	2	
6	55842	84259809	Caucasian	Male	[60-70)	NaN	3	
7	63768	114882984	Caucasian	Male	[70-80)	NaN	1	
8	12522	48330783	Caucasian	Female	[80-90)	NaN	2	
9	15738	63555939	Caucasian	Female	[90-100)	NaN	3	

10 rows × 186 columns

In [50]:
```
df.weight.notnull().sum()
```

Out[50]: 3125

In [51]:
```
df['has_weight']=df.weight.notnull().astype('int')
```

In [52]:
```
cols_extra=['age_group','has_weight']
```

In [54]:
```
#2 columns have been added to the dataframe
col2use=cols_num+cols_all_cat+cols_extra
df_data=df[col2use+['OUTPUT_LABEL']]
```

In [55]:
```
#shuffle the samples
df_data=df_data.sample(n=len(df_data),random_state=42)
df_data=df_data.reset_index(drop=True)
```

In [56]:
```
#save 30% of the data as validation and testdata
df_valid_test=df_data.sample(frac=.3,random_state=42)
print('Split Size: %.3f'%(len(df_valid_test)/len(df_data)))
```

```
Split Size: 0.300
```

In [57]:
```
#split into test and validation using 50% fraction
df_test=df_valid_test.sample(frac=.5,random_state=42)
df_valid=df_valid_test.drop(df_test.index)
```

In [58]:
```
#use the rest of the data as training data
df_train_all = df_data.drop(df_valid_test.index)
```

In [204]:
```
print('Test prevalence(n = %d):%.3f'%(len(df_test),calc_prevalence(df_test.OUTPU
print('Valid prevalence(n = %d):%.3f'%(len(df_valid),calc_prevalence(df_valid.OU
print('Train all prevalence(n = %d):%.3f'%(len(df_train_all), calc_prevalence(df_
```

```
Test prevalence(n = 14902):0.117
Valid prevalence(n = 14901):0.113
Train all prevalence(n = 69540):0.113
```

In [59]:
```
#we will create a balanced training data set that has 50% positive and 50% negat
#split the training data into positive and negative
rows_pos = df_train_all.OUTPUT_LABEL==1
df_train_pos=df_train_all.loc[rows_pos]
df_train_neg=df_train_all.loc[~rows_pos]
#merge the balanced data
df_train=pd.concat([df_train_pos,df_train_neg.sample(n=len(df_train_pos),random_
#shuffle the order of training sample
df_train=df_train.sample(n=len(df_train),random_state=42).reset_index(drop=True)
print('Train Balance prevalence(n=%d):%.3f'%(len(df_train),calc_prevalence(df_tra
```

```
Train Balance prevalence(n=15766):0.500
```

In [62]:
```
#input matrix X
X_train=df_train[col2use].values
X_train_all=df_train_all[col2use].values
X_valid=df_valid[col2use].values
#output vector y
y_train=df_train['OUTPUT_LABEL'].values
y_valid=df_valid['OUTPUT_LABEL'].values
print('Training All shapes: ',X_train_all.shape)
print('Training shapes: ',X_train.shape,y_train.shape)
print('Validation shapes: ',X_valid.shape,y_valid.shape)
```

```
Training All shapes:  (69540, 143)
Training shapes:  (15766, 143) (15766,)
Validation shapes:  (14901, 143) (14901,)
```

In [63]:
```
from sklearn.preprocessing import StandardScaler
scaler=StandardScaler()
scaler.fit(X_train_all)
```

Out[63]:
```
StandardScaler(copy=True, with_mean=True, with_std=True)
```

In [66]:
```
import pickle
scalerfile='scaler.sav'
pickle.dump(scaler,open(scalerfile,'wb'))
```

In [67]:
```
#load it back
scaler=pickle.load(open(scalerfile,'rb'))
```

In [68]:
```
X_train_tf=scaler.transform(X_train)
X_valid_tf=scaler.transform(X_valid)
```

In [144]:
```
#Model Selection
from sklearn.metrics import roc_auc_score, accuracy_score, precision_score, reca
def calc_specificity (y_actual, y_pred, thresh):
    # calculate specificity
    return sum((y_pred <thresh) &(y_actual ==0)) /sum (y_actual ==0)

def print_report (y_actual, y_pred, thresh):
    auc = roc_auc_score(y_actual, y_pred)
    accuracy =accuracy_score(y_actual, (y_pred >thresh))
    recall =recall_score(y_actual, (y_pred > thresh) )
    precision = precision_score(y_actual, (y_pred > thresh))
    specificity =calc_specificity(y_actual, y_pred, thresh)
    print('AUC:%.3f'% auc)
    print ('accuracy:%.3f'%accuracy)
    print ('recall:%.3f'%recall)
    print ('precision:%.3f'% precision)
    print('specificity:%.3f'%specificity)
    print('prevalence:%.3f'%calc_prevalence(y_actual))
    print (' ')
    return auc, accuracy, recall, precision, specificity
```

In [145]:
```
#threshold at 0.5 to label a predicted sample as positive
thresh=0.5
```

In [146]:
```
#k-nearest neighbors
from sklearn.neighbors import KNeighborsClassifier
knn=KNeighborsClassifier(n_neighbors=100)
knn.fit(X_train_tf,y_train)
```

Out[146]:
```
KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',
                     metric_params=None, n_jobs=None, n_neighbors=100, p=2,
                     weights='uniform')
```

In [147]:
```
y_train_preds=knn.predict_proba(X_train_tf)[:,1]
y_valid_preds=knn.predict_proba(X_valid_tf)[:,1]
print('knn')
print('training')
knn_train_auc,knn_train_accuracy,knn_train_recall,knn_train_precision,knn_train_
print('validation:')
knn_valid_auc,knn_valid_accuracy,knn_valid_recall,knn_valid_precision,knn_valid_
```

```
knn
training
AUC:0.650
accuracy:0.603
recall:0.491
precision:0.633
specificity:0.673
prevalence:0.500

validation:
AUC:0.621
accuracy:0.670
recall:0.469
precision:0.165
specificity:0.655
prevalence:0.113
```

In [148]:

```
# logistic regression
from sklearn.linear_model import LogisticRegression
lr=LogisticRegression(random_state = 42)
lr.fit(X_train_tf, y_train)
```

```
C:\Users\nikhi\Anaconda3\lib\site-packages\sklearn\linear_model\logistic.py:43
2: FutureWarning: Default solver will be changed to 'lbfgs' in 0.22. Specify a
solver to silence this warning.
  FutureWarning)
```

Out[148]:

```
LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True,
                   intercept_scaling=1, l1_ratio=None, max_iter=100,
                   multi_class='warn', n_jobs=None, penalty='l2',
                   random_state=42, solver='warn', tol=0.0001, verbose=0,
                   warm_start=False)
```

In [149]:

```
y_train_preds = lr.predict_proba(X_train_tf)[:,1]
y_valid_preds = lr.predict_proba(X_valid_tf)[:,1]
print('Logistic Regression')
print('Training:')
lr_train_auc, lr_train_accuracy, lr_train_recall, \
    lr_train_precision, lr_train_specificity = print_report(y_train,y_train_pred:
print('Validation:')
lr_valid_auc, lr_valid_accuracy, lr_valid_recall, \
    lr_valid_precision, lr_valid_specificity = print_report(y_valid,y_valid_pred:
```

```
Logistic Regression
Training:
AUC:0.678
accuracy:0.628
recall:0.558
precision:0.649
specificity:0.698
prevalence:0.500

Validation:
AUC:0.661
accuracy:0.662
recall:0.558
precision:0.180
specificity:0.675
prevalence:0.113
```

In [150]:
```
#Stochastic Gradient Descent
from sklearn.linear_model import SGDClassifier
sgdc=SGDClassifier(loss = 'log',alpha = 0.1,random_state = 42)
sgdc.fit(X_train_tf, y_train)
```

Out[150]:
```
SGDClassifier(alpha=0.1, average=False, class_weight=None, early_stopping=Fals
e,
              epsilon=0.1, eta0=0.0, fit_intercept=True, l1_ratio=0.15,
              learning_rate='optimal', loss='log', max_iter=1000,
              n_iter_no_change=5, n_jobs=None, penalty='l2', power_t=0.5,
              random_state=42, shuffle=True, tol=0.001, validation_fraction=0.
1,
              verbose=0, warm_start=False)
```

In [151]:
```
y_train_preds = sgdc.predict_proba(X_train_tf)[:,1]
y_valid_preds = sgdc.predict_proba(X_valid_tf)[:,1]
print('Stochastic Gradient Descend')
print('Training:')
sgdc_train_auc, sgdc_train_accuracy, sgdc_train_recall, sgdc_train_precision, sg
print('Validation:')
sgdc_valid_auc, sgdc_valid_accuracy, sgdc_valid_recall, sgdc_valid_precision, sg
```

```
Stochastic Gradient Descend
Training:
AUC:0.676
accuracy:0.624
recall:0.550
precision:0.645
specificity:0.698
prevalence:0.500

Validation:
AUC:0.661
accuracy:0.664
recall:0.553
precision:0.180
specificity:0.678
prevalence:0.113
```

In [152]:
```
# Naive Bayes
from sklearn.naive_bayes import GaussianNB
nb = GaussianNB()
nb.fit(X_train_tf, y_train)
```

Out[152]:
```
GaussianNB(priors=None, var_smoothing=1e-09)
```

In [153]:
```
y_train_preds = nb.predict_proba(X_train_tf)[:,1]
y_valid_preds = nb.predict_proba(X_valid_tf)[:,1]
print('Naive Bayes')
print('Training:')
nb_train_auc, nb_train_accuracy, nb_train_recall, nb_train_precision, nb_train_s
print('Validation:')
nb_valid_auc, nb_valid_accuracy, nb_valid_recall, nb_valid_precision, nb_valid_s
```

```
Naive Bayes
Training:
AUC:0.508
accuracy:0.506
recall:0.989
precision:0.503
specificity:0.022
prevalence:0.500

Validation:
AUC:0.505
accuracy:0.129
recall:0.988
precision:0.114
specificity:0.020
prevalence:0.113
```

In [154]:
```
#Decision Tree Classifier
from sklearn.tree import DecisionTreeClassifier
tree = DecisionTreeClassifier(max_depth = 10, random_state = 42)
tree.fit(X_train_tf, y_train)
```

Out[154]:
```
DecisionTreeClassifier(class_weight=None, criterion='gini', max_depth=10,
                       max_features=None, max_leaf_nodes=None,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=1, min_samples_split=2,
                       min_weight_fraction_leaf=0.0, presort=False,
                       random_state=42, splitter='best')
```

In [155]:
```
y_train_preds = tree.predict_proba(X_train_tf)[:,1]
y_valid_preds = tree.predict_proba(X_valid_tf)[:,1]
print('Decision Tree')
print('Training:')
tree_train_auc, tree_train_accuracy, tree_train_recall, tree_train_precision, tr
print('Validation:')
tree_valid_auc, tree_valid_accuracy, tree_valid_recall, tree_valid_precision, tr
```

```
Decision Tree
Training:
AUC:0.736
accuracy:0.672
recall:0.629
precision:0.688
specificity:0.713
prevalence:0.500

Validation:
AUC:0.625
accuracy:0.636
recall:0.572
precision:0.170
specificity:0.643
prevalence:0.113
```

In [156]:
```
#Random Forest Classifier
from sklearn.ensemble import RandomForestClassifier
rf=RandomForestClassifier(max_depth = 6, random_state = 42)
rf.fit(X_train_tf, y_train)
```

```
C:\Users\nikhi\Anaconda3\lib\site-packages\sklearn\ensemble\forest.py:245: Futu
reWarning: The default value of n_estimators will change from 10 in version 0.2
0 to 100 in 0.22.
  "10 in version 0.20 to 100 in 0.22.", FutureWarning)
```

Out[156]:
```
RandomForestClassifier(bootstrap=True, class_weight=None, criterion='gini',
                       max_depth=6, max_features='auto', max_leaf_nodes=None,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=1, min_samples_split=2,
                       min_weight_fraction_leaf=0.0, n_estimators=10,
                       n_jobs=None, oob_score=False, random_state=42, verbose=
0,
                       warm_start=False)
```

In [157]:
```
y_train_preds = rf.predict_proba(X_train_tf)[:,1]
y_valid_preds = rf.predict_proba(X_valid_tf)[:,1]
print('Random Forest')
print('Training:')
rf_train_auc, rf_train_accuracy, rf_train_recall, rf_train_precision, rf_train_s
print('Validation:')
rf_valid_auc, rf_valid_accuracy, rf_valid_recall, rf_valid_precision, rf_valid_s
```

```
Random Forest
Training:
AUC:0.681
accuracy:0.630
recall:0.576
precision:0.646
specificity:0.685
prevalence:0.500

Validation:
AUC:0.646
accuracy:0.642
recall:0.559
precision:0.170
specificity:0.653
prevalence:0.113
```

In [158]:
```
#Gradient Boosting Classifier
from sklearn.ensemble import GradientBoostingClassifier
gbc =GradientBoostingClassifier(n_estimators=100, learning_rate=1.0,
     max_depth=3, random_state=42)
gbc.fit(X_train_tf, y_train)
```

Out[158]:
```
GradientBoostingClassifier(criterion='friedman_mse', init=None,
                           learning_rate=1.0, loss='deviance', max_depth=3,
                           max_features=None, max_leaf_nodes=None,
                           min_impurity_decrease=0.0, min_impurity_split=None,
                           min_samples_leaf=1, min_samples_split=2,
                           min_weight_fraction_leaf=0.0, n_estimators=100,
                           n_iter_no_change=None, presort='auto',
                           random_state=42, subsample=1.0, tol=0.0001,
                           validation_fraction=0.1, verbose=0,
                           warm_start=False)
```

In [159]:
```
y_train_preds = gbc.predict_proba(X_train_tf)[:,1]
y_valid_preds = gbc.predict_proba(X_valid_tf)[:,1]
print('Gradient Boosting Classifier')
print('Training:')
gbc_train_auc, gbc_train_accuracy, gbc_train_recall, gbc_train_precision, gbc_tra
print('Validation:')
gbc_valid_auc, gbc_valid_accuracy, gbc_valid_recall, gbc_valid_precision, gbc_val
```

```
Gradient Boosting Classifier
Training:
AUC:0.772
accuracy:0.695
recall:0.668
precision:0.706
specificity:0.722
prevalence:0.500

Validation:
AUC:0.640
accuracy:0.621
recall:0.574
precision:0.164
specificity:0.627
prevalence:0.113
```

In [160]:
```
#Analyze results baseline models
df_results1 = pd.DataFrame({'classifier':['KNN','KNN','LR','LR','SGD','SGD','NB',
                            'data_set':['train','valid']*7,
                           'auc':[knn_train_auc, knn_valid_auc,lr_train_auc,lr_va
                           'accuracy':[knn_train_accuracy, knn_valid_accuracy,lr_
                           'recall':[knn_train_recall, knn_valid_recall,lr_train_
                           'precision':[knn_train_precision, knn_valid_precision,
                           'specificity':[knn_train_specificity, knn_valid_specif
```

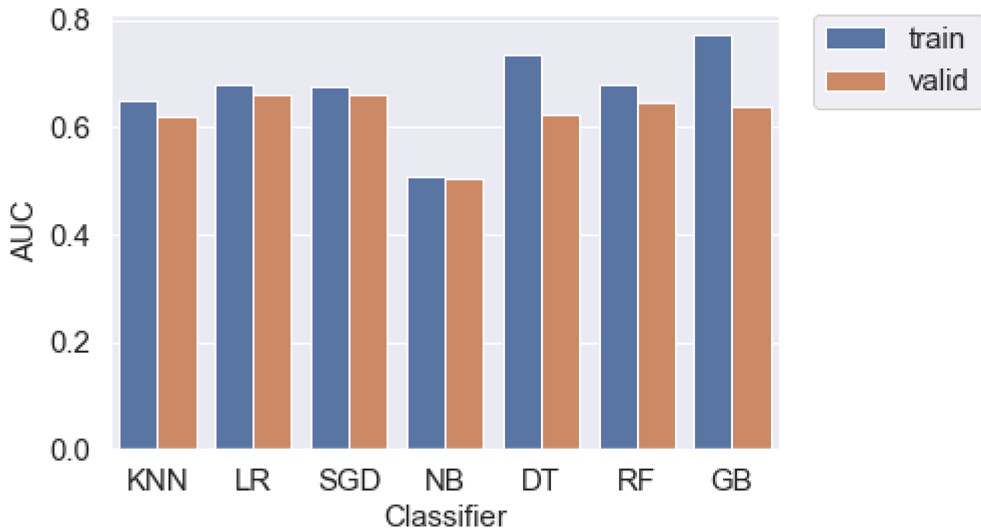
In [161]:
```
df_results1
```

Out[161]:

	classifier	data_set	auc	accuracy	recall	precision	specificity
0	KNN	train	0.649591	0.603070	0.490676	0.632957	0.673094
1	KNN	valid	0.621118	0.670425	0.469436	0.164517	0.654812
2	LR	train	0.678000	0.627807	0.557529	0.648708	0.698084
3	LR	valid	0.660551	0.661969	0.557864	0.179664	0.675242
4	SGD	train	0.675524	0.624001	0.550425	0.645396	0.697577
5	SGD	valid	0.661396	0.663915	0.553116	0.179680	0.678042
6	NB	train	0.508168	0.505835	0.989217	0.502967	0.022453
7	NB	valid	0.505320	0.128985	0.987537	0.113801	0.019522
8	DT	train	0.735899	0.671889	0.628948	0.688038	0.712673
9	DT	valid	0.625254	0.636266	0.572107	0.170228	0.642554
10	RF	train	0.680951	0.630217	0.575796	0.646121	0.684638
11	RF	valid	0.645864	0.642105	0.559050	0.170282	0.652694
12	GB	train	0.771748	0.694913	0.668147	0.771748	0.721680
13	GB	valid	0.639688	0.620831	0.574481	0.164040	0.626740

In [162]:
```
import seaborn as sns
import matplotlib.pyplot as plt
sns.set(style="darkgrid")
```

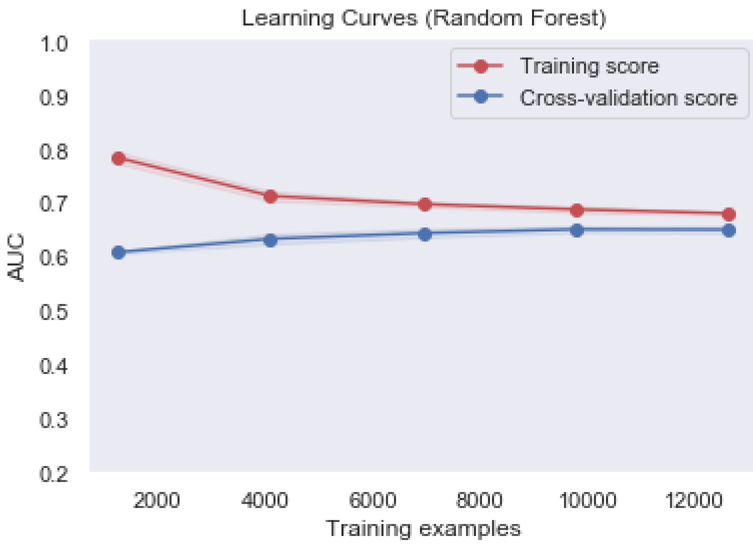
In [163]:
```
ax = sns.barplot(x="classifier", y="auc", hue="data_set", data=df_results1)
ax.tick_params(labelsize=15)
plt.legend(bbox_to_anchor=(1.05, 1), loc=2, borderaxespad=0., fontsize = 15)
plt.xlabel('Classifier')
plt.ybale('AUC')
plt.figure(figsize=(25,15))
plt.show()
```



<Figure size 1800x1080 with 0 Axes>

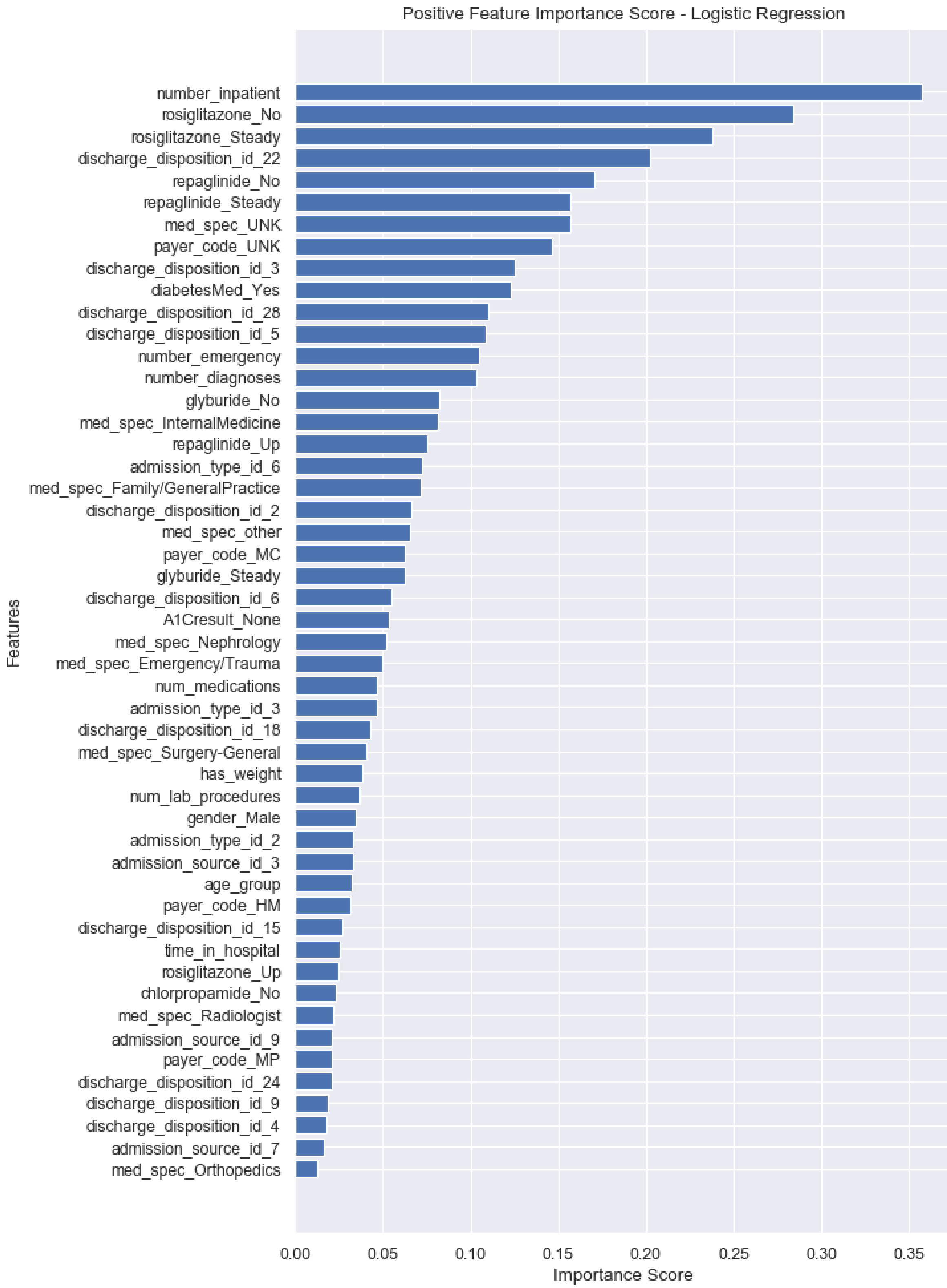
In [164]:
```
import numpy as np
from sklearn.model_selection import learning_curve
from sklearn.model_selection import ShuffleSplit

def plot_learning_curve(estimator, title, X, y, ylim=None, cv=None,
                        n_jobs=1, train_sizes=np.linspace(.1, 1.0, 5)):
    plt.figure()
    plt.title(title)
    if ylim is not None:
        plt.ylim(*ylim)
    train_sizes, train_scores, test_scores = learning_curve(
        estimator, X, y, cv=cv, n_jobs=n_jobs, train_sizes=train_sizes, scoring 
    train_scores_mean = np.mean(train_scores, axis=1)
    train_scores_std = np.std(train_scores, axis=1)
    test_scores_mean = np.mean(test_scores, axis=1)
    test_scores_std = np.std(test_scores, axis=1)
    plt.grid()
    plt.fill_between(train_sizes, train_scores_mean - train_scores_std,
                     train_scores_mean + train_scores_std, alpha=0.1,
                     color="r")
    plt.fill_between(train_sizes, test_scores_mean - test_scores_std,
                     test_scores_mean + test_scores_std, alpha=0.1, color="b")
    plt.plot(train_sizes, train_scores_mean, 'o-', color="r",
             label="Training score")
    plt.plot(train_sizes, test_scores_mean, 'o-', color="b",
             label="Cross-validation score")
    plt.legend(loc="best")
    return plt
```

In [165]:
```
cv = ShuffleSplit(n_splits=5, test_size=0.2, random_state=42)
estimator = RandomForestClassifier(max_depth = 6, random_state = 42)
plot_learning_curve(estimator, title, X_train_tf, y_train, ylim=(0.2, 1.01), cv=
plt.xlabel('Training examples')
plt.ylabel('AUC')
plt.title('Learning Curves (Random Forest)')
plt.show()
```

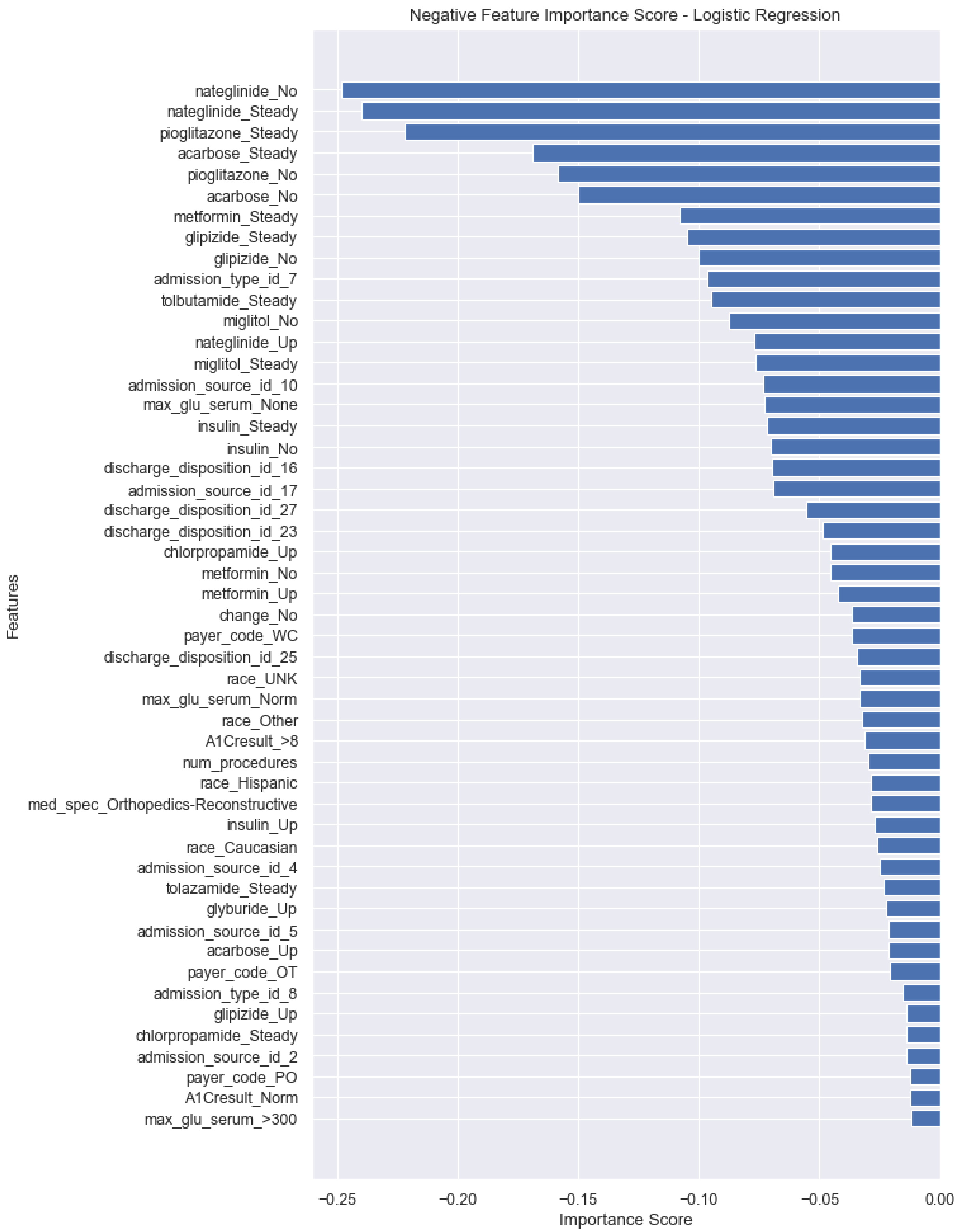


In [109]:
```
feature_importances = pd.DataFrame(lr.coef_[0],
                                   index = col2use,
                                    columns=['importance']).sort_values('importa
                                                                       ascendin
```

In [110]:
```
num = 50
ylocs = np.arange(num)
values_to_plot = feature_importances.iloc[:num].values.ravel()[::-1]
feature_labels = list(feature_importances.iloc[:num].index)[::-1]
plt.figure(num=None, figsize=(8, 15), dpi=80, facecolor='w', edgecolor='k');
plt.barh(ylocs, values_to_plot, align = 'center')
plt.ylabel('Features')
plt.xlabel('Importance Score')
plt.title('Positive Feature Importance Score - Logistic Regression')
plt.yticks(ylocs, feature_labels)
plt.show()
```


Positive Feature Importance Score - Logistic Regression
number_inpatient
rosiglitazone_No
rosiglitazone_Steady
discharge_disposition_id_22
repaglinide_No
repaglinide_Steady
med_spec_UNK
payer_code_UNK
discharge_disposition_id_3
diabetesMed_Yes
discharge_disposition_id_28
discharge_disposition_id_5
number_emergency
number_diagnoses
glyburide_No
med_spec_InternalMedicine
repaglinide_Up
admission_type_id_6
med_spec_Family/GeneralPractice
discharge_disposition_id_2
med_spec_other
payer_code_MC
glyburide_Steady
discharge_disposition_id_6
A1Cresult_None
med_spec_Nephrology
med_spec_Emergency/Trauma
num_medications
admission_type_id_3
discharge_disposition_id_18
med_spec_Surgery-General
has_weight
num_lab_procedures
gender_Male
admission_type_id_2
admission_source_id_3
age_group
payer_code_HM
discharge_disposition_id_15
time_in_hospital
rosiglitazone_Up
chlorpropamide_No
med_spec_Radiologist
admission_source_id_9
payer_code_MP
discharge_disposition_id_24
discharge_disposition_id_9
discharge_disposition_id_4
admission_source_id_7
med_spec_Orthopedics
Features
0.00
0.05
0.10
0.15
0.20
0.25
0.30
0.35
Importance Score

In [111]:
```
values_to_plot = feature_importances.iloc[-num:].values.ravel()
feature_labels = list(feature_importances.iloc[-num:].index)
plt.figure(num=None, figsize=(8, 15), dpi=80, facecolor='w', edgecolor='k');
plt.barh(ylocs, values_to_plot, align = 'center')
plt.ylabel('Features')
plt.xlabel('Importance Score')
plt.title('Negative Feature Importance Score - Logistic Regression')
plt.yticks(ylocs, feature_labels)
plt.show()
```



In [112]:
```
from sklearn.model_selection import RandomizedSearchCV
# number of trees
n_estimators = range(200,1000,200)
# maximum number of features to use at each split
max_features = ['auto','sqrt']
# maximum depth of the tree
max_depth = range(1,10,1)
# minimum number of samples to split a node
min_samples_split = range(2,10,2)
# criterion for evaluating a split
criterion = ['gini','entropy']
# random grid
random_grid = {'n_estimators':n_estimators,
              'max_features':max_features,
              'max_depth':max_depth,
              'min_samples_split':min_samples_split,
              'criterion':criterion}
print(random_grid)
```

```
{'n_estimators': range(200, 1000, 200), 'max_features': ['auto', 'sqrt'], 'max_
depth': range(1, 10), 'min_samples_split': range(2, 10, 2), 'criterion': ['gin
i', 'entropy']}
```

In [113]:
```
from sklearn.metrics import make_scorer, roc_auc_score
auc_scoring = make_scorer(roc_auc_score)
```

In [ ]:
```
#Randomized Search random forest
```

In [114]:
```
# create the randomized search cross-validation
rf_random = RandomizedSearchCV(estimator = rf, param_distributions = random_grid
                               n_iter = 20, cv = 2, scoring=auc_scoring,
                               verbose = 1, random_state = 42)
```

In [115]:
```
# fit the random search model (this will take a few minutes)
t1 = time.time()
rf_random.fit(X_train_tf, y_train)
t2 = time.time()
print(t2-t1)
```

```
Fitting 2 folds for each of 20 candidates, totalling 40 fits

[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent worker
s.
[Parallel(n_jobs=1)]: Done  40 out of  40 | elapsed:  2.5min finished

156.14898324012756
```

In [116]:
```
rf_random.best_params_
```

Out[116]:
```
{'n_estimators': 600,
 'min_samples_split': 6,
 'max_features': 'auto',
 'max_depth': 8,
 'criterion': 'gini'}
```

In [187]:
```
y_train_preds = rf.predict_proba(X_train_tf)[:,1]
y_valid_preds = rf.predict_proba(X_valid_tf)[:,1]
print('Baseline Random Forest')
rf_train_auc_base = roc_auc_score(y_train, y_train_preds)
rf_valid_auc_base = roc_auc_score(y_valid, y_valid_preds)
print('Training AUC:%.3f'%(rf_train_auc_base))
print('Validation AUC:%.3f'%(rf_valid_auc_base))
print('Optimized Random Forest')
y_train_preds_random = rf_random.best_estimator_.predict_proba(X_train_tf)[:,1]
y_valid_preds_random = rf_random.best_estimator_.predict_proba(X_valid_tf)[:,1]
rf_train_auc = roc_auc_score(y_train, y_train_preds_random)
rf_valid_auc = roc_auc_score(y_valid, y_valid_preds_random)
print('Training AUC:%.3f'%(rf_train_auc))
print('Validation AUC:%.3f'%(rf_valid_auc))
```

```
Baseline Random Forest
Training AUC:0.681
Validation AUC:0.646
Optimized Random Forest
Training AUC:0.723
Validation AUC:0.661
```

In [188]:
```
print(rf_valid_auc_base)
print(rf_valid_auc)
```

```
0.6458642208706648
0.6614703156321624
```

In [118]:
```
#optimize the performnce of stochastic gradient descent
penalty = ['none','l2','l1']
max_iter = range(100,500,100)
alpha = [0.001,0.003,0.01,0.03,0.1,0.3]
random_grid_sgdc = {'penalty':penalty,
              'max_iter':max_iter,
              'alpha':alpha}
# create the randomized search cross-validation
sgdc_random = RandomizedSearchCV(estimator = sgdc, param_distributions = random_g
                                n_iter = 20, cv = 2, scoring=auc_scoring,verbos
                                random_state = 42)
t1 = time.time()
sgdc_random.fit(X_train_tf, y_train)
t2 = time.time()
print(t2-t1)
```

```
C:\Users\nikhi\Anaconda3\lib\site-packages\sklearn\linear_model\stochastic_grad
ient.py:561: ConvergenceWarning: Maximum number of iteration reached before con
vergence. Consider increasing max_iter to improve the fit.
  ConvergenceWarning)
C:\Users\nikhi\Anaconda3\lib\site-packages\sklearn\linear_model\stochastic_grad
ient.py:561: ConvergenceWarning: Maximum number of iteration reached before con
vergence. Consider increasing max_iter to improve the fit.
  ConvergenceWarning)
C:\Users\nikhi\Anaconda3\lib\site-packages\sklearn\linear_model\stochastic_grad
ient.py:561: ConvergenceWarning: Maximum number of iteration reached before con
vergence. Consider increasing max_iter to improve the fit.
  ConvergenceWarning)
C:\Users\nikhi\Anaconda3\lib\site-packages\sklearn\linear_model\stochastic_grad
ient.py:561: ConvergenceWarning: Maximum number of iteration reached before con
vergence. Consider increasing max_iter to improve the fit.
  ConvergenceWarning)

18.213062524795532
```

In [119]:
```
sgdc_random.best_params_
```

Out[119]:
```
{'penalty': 'none', 'max_iter': 400, 'alpha': 0.001}
```

In [191]:
```
y_train_preds = sgdc.predict_proba(X_train_tf)[:,1]
y_valid_preds = sgdc.predict_proba(X_valid_tf)[:,1]

print('Baseline sgdc')
sgdc_train_auc_base = roc_auc_score(y_train, y_train_preds)
sgdc_valid_auc_base = roc_auc_score(y_valid, y_valid_preds)

print('Training AUC:%.3f'%(sgdc_train_auc_base))
print('Validation AUC:%.3f'%(sgdc_valid_auc_base))
print('Optimized sgdc')
y_train_preds_random = sgdc_random.best_estimator_.predict_proba(X_train_tf)[:,1
y_valid_preds_random = sgdc_random.best_estimator_.predict_proba(X_valid_tf)[:,1
sgdc_train_auc = roc_auc_score(y_train, y_train_preds_random)
sgdc_valid_auc = roc_auc_score(y_valid, y_valid_preds_random)

print('Training AUC:%.3f'%(sgdc_train_auc))
print('Validation AUC:%.3f'%(sgdc_valid_auc))
```

```
Baseline sgdc
Training AUC:0.676
Validation AUC:0.661
Optimized sgdc
Training AUC:0.676
Validation AUC:0.657
```

In [192]:
```
print(sgdc_valid_auc_base)
print(sgdc_valid_auc)
```

```
0.6613956376948003
0.6566861227466392
```

In [166]:
```
#optimize the performnce of Gradient boosting classifier
# number of trees
n_estimators = range(100,500,100)
# maximum depth of the tree
max_depth = range(1,5,1)
# learning rate
learning_rate = [0.001,0.01,0.1]
# random grid
random_grid_gbc = {'n_estimators':n_estimators,
              'max_depth':max_depth,
              'learning_rate':learning_rate}
# create the randomized search cross-validation
gbc_random = RandomizedSearchCV(estimator = gbc, param_distributions = random_gr
                               n_iter = 20, cv = 2, scoring=auc_scoring,
                               verbose = 0, random_state = 42)
t1 = time.time()
gbc_random.fit(X_train_tf, y_train)
t2 = time.time()
print(t2-t1)
```

```
458.43239283561707
```

In [167]:
```
gbc_random.best_params_
```

Out[167]:
```
{'n_estimators': 200, 'max_depth': 2, 'learning_rate': 0.1}
```

In [168]:
```
y_train_preds = gbc.predict_proba(X_train_tf)[:,1]
y_valid_preds = gbc.predict_proba(X_valid_tf)[:,1]
print('Baseline gbc')
gbc_train_auc_base = roc_auc_score(y_train, y_train_preds)
gbc_valid_auc_base = roc_auc_score(y_valid, y_valid_preds)
print('Training AUC:%.3f'%(gbc_train_auc_base))
print('Validation AUC:%.3f'%(gbc_valid_auc_base))
print('Optimized gbc')
y_train_preds_random = gbc_random.best_estimator_.predict_proba(X_train_tf)[:,1]
y_valid_preds_random = gbc_random.best_estimator_.predict_proba(X_valid_tf)[:,1]
gbc_train_auc = roc_auc_score(y_train, y_train_preds_random)
gbc_valid_auc = roc_auc_score(y_valid, y_valid_preds_random)
print('Training AUC:%.3f'%(gbc_train_auc))
print('Validation AUC:%.3f'%(gbc_valid_auc))
```

```
Baseline gbc
Training AUC:0.772
Validation AUC:0.640
Optimized gbc
Training AUC:0.690
Validation AUC:0.671
```

In [195]:
```
print(gbc_valid_auc_base)
print(gbc_valid_auc)
```

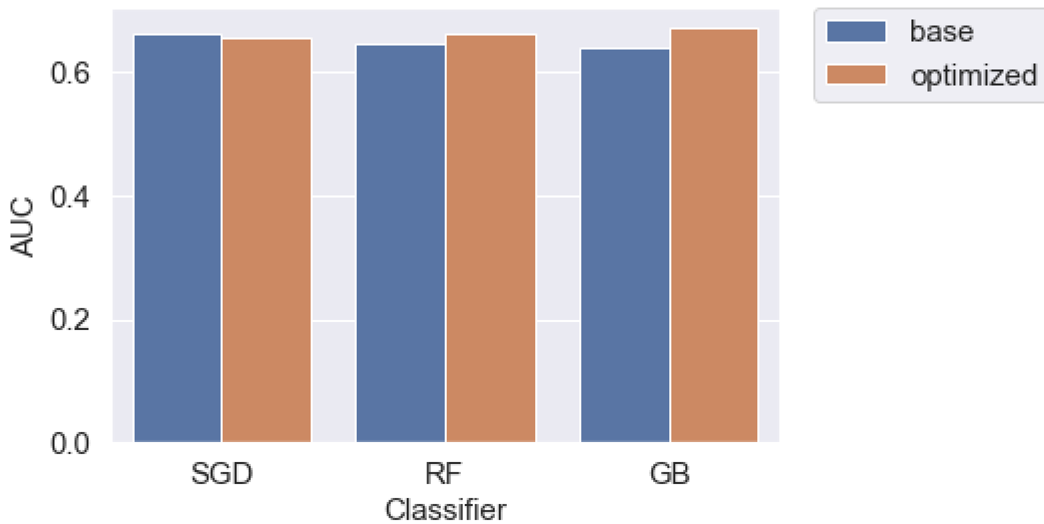
```
0.6396884722052578
0.6711373140011927
```

In [203]:
```
#Hyperparameter tuning result
df_results = pd.DataFrame({'classifier':['SGD','SGD','RF','RF','GB','GB'],
                           'data_set':['base','optimized']*3,
                          'auc':[sgdc_valid_auc_base,sgdc_valid_auc,
                                 rf_valid_auc_base,rf_valid_auc,
                                 gbc_valid_auc_base,gbc_valid_auc,],
                          })
df_results
```

Out[203]:

	classifier	data_set	auc
0	SGD	base	0.661396
1	SGD	optimized	0.656686
2	RF	base	0.645864
3	RF	optimized	0.661470
4	GB	base	0.639688
5	GB	optimized	0.671137

In [194]:
```
#Aggregate the results to compare baseline model on validation set
ax = sns.barplot(x="classifier", y="auc", hue="data_set", data=df_results)
ax.set_xlabel('Classifier',fontsize = 15)
ax.set_ylabel('AUC', fontsize = 15)
ax.tick_params(labelsize=15)
plt.legend(bbox_to_anchor=(1.05, 1), loc=2, borderaxespad=0., fontsize = 15)
plt.figure(figsize=(20,15))
plt.show()
```




```
<Figure size 1440x1080 with 0 Axes>
```

In [126]:
```
#model selection: best classifier
pickle.dump(gbc_random.best_estimator_, open('best_classifier.pkl', 'wb'),protoco
```

In [127]:
```
#model evaluation
X_test = df_test[col2use].values
y_test = df_test['OUTPUT_LABEL'].values
scaler = pickle.load(open('scaler.sav', 'rb'))
X_test_tf = scaler.transform(X_test)
```

In [128]:
```
best_model = pickle.load(open('best_classifier.pkl','rb'))
y_train_preds = best_model.predict_proba(X_train_tf)[:,1]
y_valid_preds = best_model.predict_proba(X_valid_tf)[:,1]
y_test_preds = best_model.predict_proba(X_test_tf)[:,1]
```

In [196]:
```
thresh = 0.5
print('Training:')
train_auc, train_accuracy, train_recall, train_precision, train_specificity = pr
print('Validation:')
valid_auc, valid_accuracy, valid_recall, valid_precision, valid_specificity = pr
print('Test:')
test_auc, test_accuracy, test_recall, test_precision, test_specificity = print_r
```

```
Training:
AUC:0.676
accuracy:0.624
recall:0.550
precision:0.645
specificity:0.698
prevalence:0.500

Validation:
AUC:0.661
accuracy:0.664
recall:0.553
precision:0.180
specificity:0.678
prevalence:0.113

Test:
AUC:0.667
accuracy:0.650
recall:0.578
precision:0.184
specificity:0.660
prevalence:0.117
```

In [ ]:
```
#final evaluation
```

In [202]:
```
from IPython.display import Image
Image('untitled.png')
```

Out[202]:

	Training	**Validation**	**Test**
AUC	0.676	0.661	0.667
Accuracy	0.624	0.664	0.65
Recall	0.55	0.553	0.578
Precision	0.645	0.18	0.184
Specificity	0.698	0.678	0.66
Prevalence	0.5	0.113	0.117

In [130]:
```
from sklearn.metrics import roc_curve
fpr_train, tpr_train, thresholds_train = roc_curve(y_train, y_train_preds)
auc_train = roc_auc_score(y_train, y_train_preds)
fpr_valid, tpr_valid, thresholds_valid = roc_curve(y_valid, y_valid_preds)
auc_valid = roc_auc_score(y_valid, y_valid_preds)
fpr_test, tpr_test, thresholds_test = roc_curve(y_test, y_test_preds)
auc_test = roc_auc_score(y_test, y_test_preds)
plt.plot(fpr_train, tpr_train, 'r-',label ='Train AUC:%.3f'%auc_train)
plt.plot(fpr_valid, tpr_valid, 'b-',label ='Valid AUC:%.3f'%auc_valid)
plt.plot(fpr_test, tpr_test, 'g-',label ='Test AUC:%.3f'%auc_test)
plt.plot([0,1],[0,1],'k--')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.legend()
plt.show()
```