# STA 208 Final Project

Wenjia Wang, Zijun Chen

University of California, Davis
June, 12

# Introduction

For promotion purpose, patient self-evaluation system served for new medicine product is necessary to let target customers know quickly if the medicine would be recommended based on their situations. A detailed chart review of over thousand potential patients about their relevant treatment histories and present circumstances was collected out of this motivation. With these treatments and diagnoses records, a prediction model is desired to build to give whether the new medicine would be recommended to a patient provided his/her disease history.

# Data Description

There are 1773 observations with 83 features to decide if the medicine should be recommended. Among these 83 features, there are 3 continuous numerical variables, 9 ordinal categorical variable and 71 binary variables. They are ages related records, appearance times of relative or irrelative risk factors and diagnoses record history respectively. Response value (here 1 is denoted as yes and 0 as no) has nearly same amount of 1 and 0, which is balance case in this data set.

# Missing Value Imputation

Pushed by lots of missing values in this data set, different imputation methods were applied on numerical variables and categorical variables separately. For categorical variables, missing value in each feature was set as a new level. In the following model selection, one-hot coding was used for all categorical variables. For numeric feature "age_at_rf_r1_dx", 1/3 missing values were imputed by linear regression since high correlation was observed between itself and another 2 numeric variables as Figure 1 shows.

# Method Implementation

In this section, we implemented and compared different methods, including SVM, random forest, Adaboost, as well as feed forward neural network.

In general, for binary classification problem, assessment of models can be based on confusion matrix, F1 score, Precision-Recall curve and ROC curve. The precision is the ratio $\frac{TP}{TP+FP}$ where 'TP' is the number of true positives and 'TF' the number of false positives. The precision is intuitively the ability of the classifier not to label as positive a sample that is negative. The recall is the ratio $\frac{TP}{TP+FN}$ 'FN' is the number of false negatives. The recall is intuitively the ability of the classifier to find all the positive samples. In our case, the purpose is to

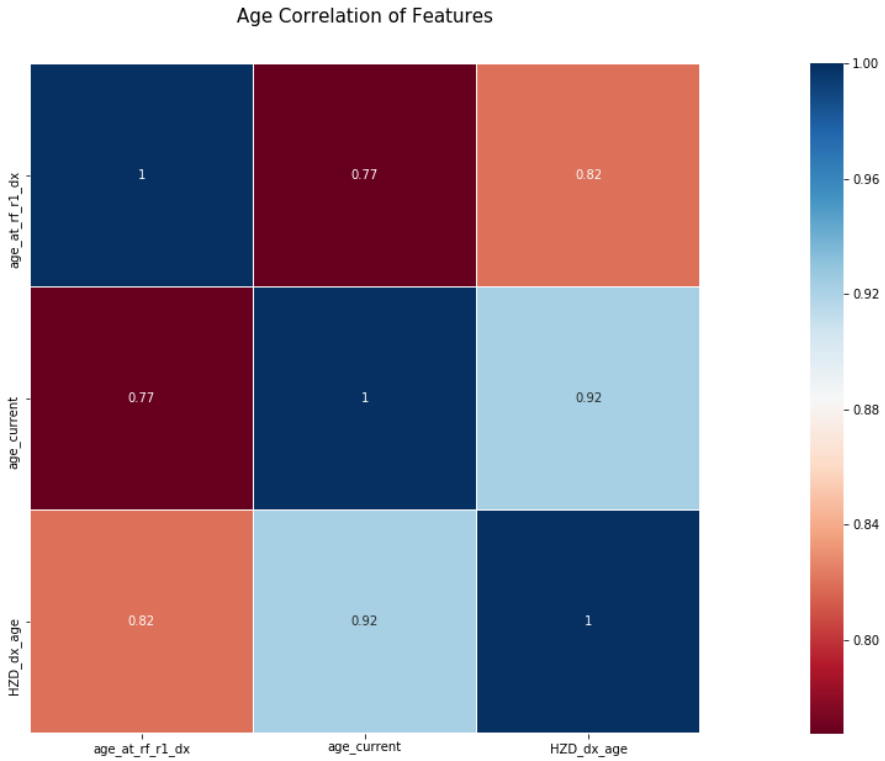



Figure 1: This figure displays the correlation between numerical variables for imputation purpose.

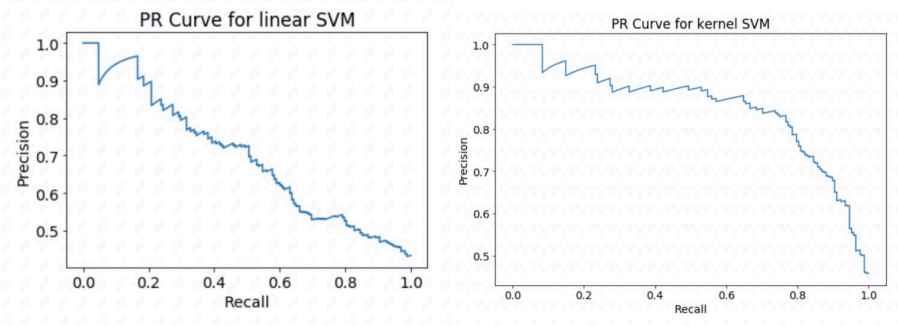



Figure 2: This figure shows PR curves of linear SVM and selected kernel SVM. PR of an efficient model should be large for all recall values, which means it should be on the top right. This figure implies that the selected kernel SVM model is superior to the linear SVM in terms of PR curves.

recommend the new drug to the potential customers. There the main concern should be the sensitivity and precision of the classifier.

In order to make comparison, the data is split into training set and testing set. The process of tuning parameters is carried out by using 5-fold cross validation on training set.

## 1. Preliminary Exploration: SVM

To begin with, we appeal to the classical linear classifier SVM. Taking the gap of magnitude between numerical predictors(ages) and dummy predictors into consideration, we scaled the numerical predictors before fitting SVM.

After tuning the penalty parameter 'C' of the error term, the best linear SVM model was given when C=10. The F1 scores for training set and testing set are only 0.583 and 0.559 respectively. This unsatisfactory results indicates that linear classifier might be inappropriate for our case, which lead us to kernel SVM.

To obtain an efficient model, both 'kernel' and penalty parameter 'C' were tuned. It turned out that the choice of largest penalty and polynomial kernel with degree 2 rendered a better result. The F1 scores for training set and testing set increased to 0.740 and 0.747 respectively. Figure 2 compares the PR curves of linear SVM and selected kernel SVM.

Actually, it is reasonable that polynomial kernel is the eventual choice. Intuitively, the polynomial kernel looks not only at the given features of input samples to determine their similarity, but also combinations of these (similar to the interaction effect in regression analysis). Because most of the input features are binary-valued, there exists some implicit features correspond to logical conjunctions of those binary-valued features. In addition, in natural language process-

ing(NLP) which shares high similarity with our binary-valued sparse data, the polynomial kernel is quite popular[1]. This also make the result justifiable.

However, always choosing the largest penalty value is also unusual. As we know, support vector machine always try to achieve a larger margin and a lower misclassification rate simultaneously. But these two goals are contradictory. To be more specific, smaller margin tends to co-occur with lower misclassification rate but result in over fitting. Larger penalty value here corresponds to a narrow margin, which suggests that the data may be over fitted by this complex decision boundary.

## 2. Random Forest Model

Motivated by the defect of linear classifier, decision tree was firstly considered since this data set is structured data filed table, such as records of questionnaire like here. However, even decision tree might give high accuracy rate, it usually cause high prediction variance. To avoid this potential issue, random forest model as ensemble method was fitted. 100 trees were selected for each model considering cost of learning. We used Gini as criterion for less computationally intensive. Different depths, max features and criterions were tuned during fitting. The model with highest accuracy rate is as below:

Table 1: parameters of best random forest model

| criterion | max_depth | max_features | n_estimators |
|---|---|---|---|
| Gini | 30 | 50 | 100 |

Based on the model given in table 2, F1 score is 0.73 on test set while accuracy rate was 0.82 on training set and 0.80 on testing set. Figure 3 also displays the ROC curve and PR curve for tesing data. The improvement of F1 score compared with SVM might results from the avoidance of giving "improper" distance between numeric features and categorical features even after standardlization. Additionally, we can infer that the fraction of irrelative features to response value is small based on the consist performance of model on training set and testing set. The reason is that if the fraction of relative features were small, the chance for them being selected can be small at each split due to the random selection of random forest fitting. Lower F1 score on test set given by random forest model comparing with SVM motivates us to improve model in further step. Random forest gives robust result by reducing variance. To decrease the bias of each decision tree as base estimator in the random forest, Adaboost was applied with still decision tree as base estimator.

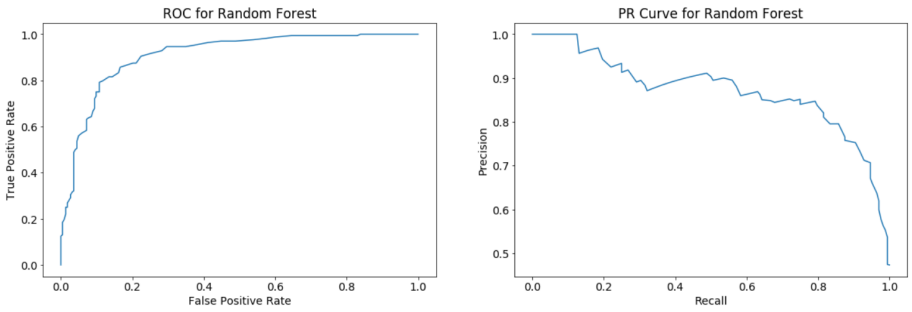



Figure 3: This figure shows ROC and PR curves for testing data by using random forest model. The ROC curve of a good model should be in top left. Compared to the PR curve of kernel SVM, there is no obvious improvement.

## 3. Adaboost

As boosting algorithm, Adaboost gives more attention on misclassified points and turns weak learner into strong learner by adjusting the weight of each base estimator based on their prediction capability. Minimizing the bias of the ensemble and not only the variance is advantage of Adaboost comparing with random forest[2].

Learning rate here was set for lightening the weight for each estimator to avoid potential overfitting. Combinations of iterations and learning rate were tuned to increase the chance of convergence. Besides, depth of base decision tree was tuned in each combination of learning rate and iteration. The model with highest score is given in table 2.

Table 2: parameters of best Adaboost model

| Learning Rate | max_depth | n_estimators |
|---|---|---|
| 1 | 10 | 500 |

Accuracy rate given by the model in the table is 0.84 on train set and 0.87 on test set. F1 score on test set is 0.83. The result got improved dramatically. ROC and PR curves are plotted to compare with other models. Comparing with random forest, adaboost focus on high bias and low variance base estimator to minimize overall features. Chance of overfitting decreases by using low variance base estimator. As a result, adaboost here models the data distribution globally better.

## 4. Feed Forward Neural Network

In this part, we end up with another non-linear model: neural network. This is inspired by the similar characteristics of our sparse data to the natural language

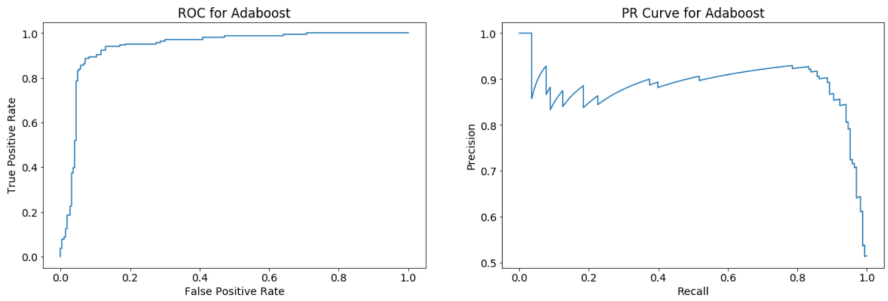



Figure 4: This figure shows ROC and PR curves for testing data by using Adaboost. In terms of the ROC, Adaboost is obviously superior to random forest. But compared to the PR curve of random forest, there only exists a slight improvement.

processing. However, since the number of feature is moderate, the embedding step is skipped here.

Specifically, fully connected neural network containing one hidden layer with ReLU activation is fitted. For binary classification, logistic loss may be appropriate and sigmoid activation is implemented at the last step to generate the probability. It turns out the F1 score on training and testing data set are 0.942 and 0.795 respectively. The PR curve as Figure 5 shows is not that satisfactory.

The much higher accuracy rate on training set compared to that on testing set suggests over-fitting. This is actually not surprising because there are much more parameters than observations. Hence, consider the relatively small data set, neural network may be not quite suitable.

Further more, in general a neural network can approximate any continuous or piece wise continuous association between the features and response by convolution. During the training phase the continuity of the data guarantees the convergence of the optimization, and during the prediction phase it ensures that slightly changing the values of the input keeps the output stable[3]. But the structured data here with lots of categorical features may not have continuity. Even after one-hot encoding, the informative relations among categories may by ignored the informative relations between them. To increase the continuity and capture the similarity among categories, embedding layers can be added below normal layers[4].

On the other hand, decision trees do not assume any continuity of the feature and can divide the states of a variable as fine as necessary. Therefore, tree based methods are usually applied to deal with structured data.

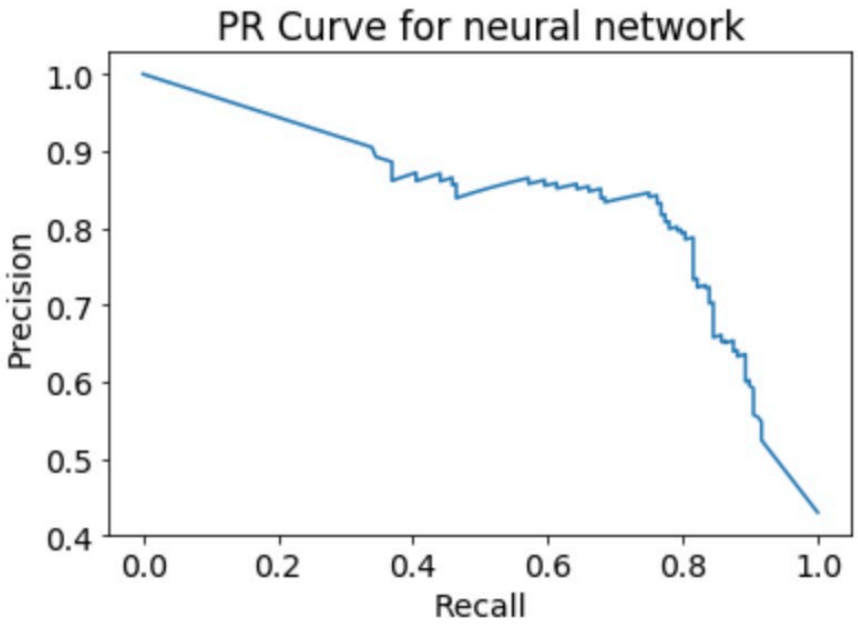



Figure 5: This figure shows PR curve of neural network. PR becomes much smaller for larger recall, which implies that the neural network model may not perform well in terms of PR curve.

# Conclusion

Table 3 compares the time consuming for tuning and training models, as well as the F1 scores on testing data. Based on the above discussion, in general non-linear models performs better than linear one. Additionally, tree based methods tend to be more stable than neural network, though the F1 score by implementing neural network is also acceptable.

Now move attention to polynomial SVM, random forest and Adaboost. The results of Adaboost outperforms the other two methods, whereas it is extremely prohibitive in terms of computation. Based on the analysis above, the SVM with polynomial kernel is more likely to over fit data, since it always looks for a larger penalty value. Therefore, overall the random forest may be more applicable in practice.

Table 3: Classification Results and Consuming Time on Testing Data

| | LinearSVM | PolySVM | RandomForest | Adaboost | NeuralNet |
|---|---|---|---|---|---|
| AccuracyRate | 0.69 | 0.81 | 0.80 | 0.87 | 0.83 |
| F1 score | 0.559 | 0.747 | 0.73 | 0.83 | 0.795 |
| Time | 18.304 | 26.778 | 40.855 | 240.191 | 33.170 |

# Limitation

Restricted by the small size of the data, further exploration of deep neural network is not applicable here. Additionally, embedding layers which should be helpful to improve the performance of the neural network is not applicable here either. For adaboost model, tuning space of iteration and learning rate combination was limited to be small due to its high computational complexity. Other imputation methods would be tried in further study, such as KNN imputation or mean imputation.

# References

[1] Morita, H., Takamura, H., Okumura, M. (2009). Structured Output Learning with Polynomial Kernel. RANLP.

[2] Schapire, R.E. (2013). Explaining AdaBoost. Empirical Inference.

[3] Cheng Guo, Felix Berkhahn:Entity Embeddings of Categorical Variables. CoRR abs/1604.06737 (2016).

[4] He, Xiangnan, Lizi Liao, Hanwang Zhang, Liqiang Nie, Xia Hu and Tat-Seng Chua. Neural Collaborative Filtering. (2017).

# Appendix

In [104]:
```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import GridSearchCV
from sklearn.ensemble import AdaBoostClassifier
from sklearn.tree import DecisionTreeClassifier
import sklearn.preprocessing as preprocess
from sklearn.svm import SVC
from sklearn.metrics import f1_score, confusion_matrix, roc_curve,
auc, precision_recall_curve
import time
import sklearn


import warnings
warnings.filterwarnings("ignore")

plt.rc("font", size=14)
```

In [157]:
```
data_raw = pd.read_excel('HZNP_DataScience_Exercise.xlsx',sheet_nam
e = 1)
data_raw.set_index('patientID',inplace = True)
data_raw.head()
```

Out[157]:

| patientID | age_at_rf_r1_dx | patient_gender | current_severity | severity_at_HZD_dx | age_curren |
|---|---|---|---|---|---|
| 0 | NaN | F | Moderate | Moderate | 6 |
| 1 | NaN | F | Moderate | Moderate | 4 |
| 2 | NaN | F | Moderate | Moderate | 6 |
| 3 | NaN | F | Moderate | Mild | 4 |
| 4 | 71.0 | M | Severe | Severe | 7 |

5 rows × 83 columns

In [158]:
```
#change data type
col_names = list(data_raw)
col_name_num = ["age_at_rf_r1_dx","age_current","HZD_dx_age"] #+dat
a_raw.columns[pd.Series(data_raw.columns).str.startswith('rf_r')].v
alues.tolist()+["px_count_for_HZD_tried","recommend_Product_X"]
col_name_cat = list(set(col_names)-set(col_name_num))

for col in col_name_cat:
    data_raw[col] = data_raw[col].astype('object')
```

In [159]:
```
feature_data = data_raw.drop('recommend_Product_X', 1)
response_data = data_raw.recommend_Product_X
```

In [160]:
```
# imputation categorical
categorical_feature = feature_data.select_dtypes(include=['object']
)
categorical_feature = categorical_feature.fillna('missing')
categorical_feature = pd.get_dummies(categorical_feature)
```

In [161]:
```
categorical_feature.shape
```

Out[161]:
```
(1773, 111)
```

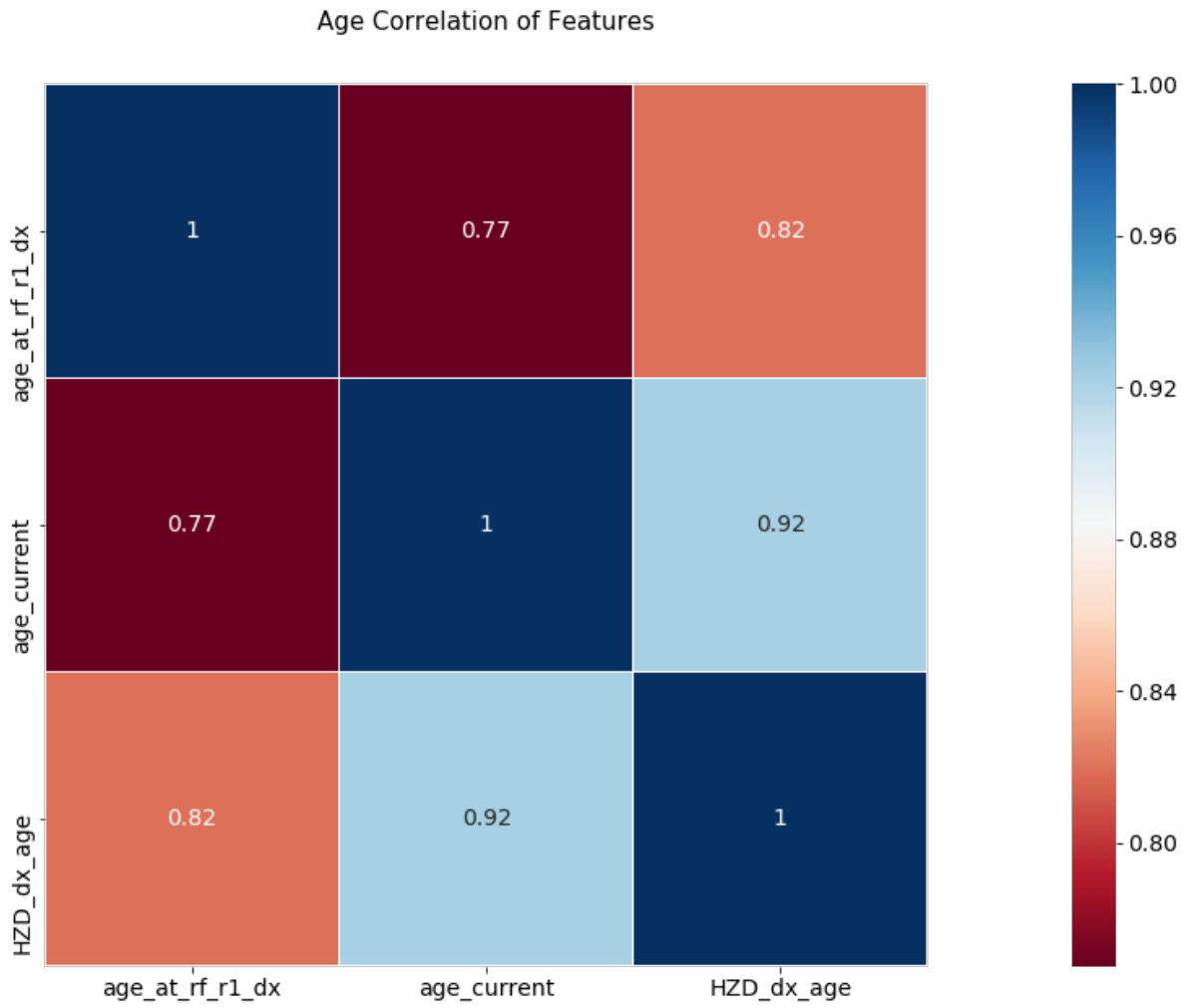
In [162]:
```
#imputation numerical
numerical_feature = feature_data.select_dtypes(include=[np.number])
#numerical_feature = numerical_feature.fillna(numerical_feature.med
ian())
#numerical_feature.iloc[:,:] = pd.DataFrame(preprocessing.scale(num
erical_feature.iloc[:,:])).values
```

In [163]:
```
#necessarity of imputation for age_at_rf_r1_dx
numerical_feature.isnull().sum()
```

Out[163]:
```
age_at_rf_r1_dx    531
age_current          0
HZD_dx_age           0
dtype: int64
```

In [164]:
```
colormap = plt.cm.RdBu
plt.figure(figsize=(32,10))
plt.title('Age Correlation of Features', y=1.05, size=15)
sns.heatmap(data_raw.corr(),linewidths=0.1,vmax=1.0,
            square=True, cmap=colormap, linecolor='white', annot=True)
# Note that the categorical features have been neglected in the
# correlation matrix.
```

Out[164]: <matplotlib.axes._subplots.AxesSubplot at 0x1c3b6d41d0>



In [165]:
```
#imputation for age_at_rf_r1_dx by linear regression
linreg = LinearRegression()
x_train = numerical_feature[numerical_feature['age_at_rf_r1_dx'].no
tnull()].drop(columns='age_at_rf_r1_dx')
y_train = numerical_feature[numerical_feature['age_at_rf_r1_dx'].no
tnull()]['age_at_rf_r1_dx']
x_test = numerical_feature[numerical_feature['age_at_rf_r1_dx'].isn
ull()].drop(columns='age_at_rf_r1_dx')
y_test = numerical_feature[numerical_feature['age_at_rf_r1_dx'].isn
ull()]['age_at_rf_r1_dx']
linreg.fit(x_train, y_train)
predicted = linreg.predict(x_test)
numerical_feature.age_at_rf_r1_dx[numerical_feature.age_at_rf_r1_dx
.isnull()] = predicted
```

In [166]:
```
feature_data_final = pd.concat([numerical_feature, categorical_feat
ure], axis=1)
```

In [167]:
```
feature_data_final.head()
```

Out[167]:

| patientID | age_at_rf_r1_dx | age_current | HZD_dx_age | has_had_surgery | rf_r1_present | rf_r2_ |
|---|---|---|---|---|---|---|
| **0** | 51.955585 | 65 | 55 | 0 | 0 | |
| **1** | 33.862071 | 45 | 35 | 0 | 0 | |
| **2** | 59.497798 | 68 | 64 | 0 | 1 | |
| **3** | 33.462159 | 41 | 35 | 0 | 0 | |
| **4** | 71.000000 | 74 | 72 | 0 | 1 | |

5 rows × 114 columns

In [210]:
```
#test and train split
X_train, X_test, y_train, y_test = train_test_split(feature_data_fi
nal, response_data, test_size=0.22, random_state=4)
X_train=X_train.values
X_test=X_test.values
y_train=y_train.astype("int").values
y_test=y_test.astype("int").values
```

In [172]:
```
# Scale the numerical features
numerical_feature_stand=pd.DataFrame(preprocess.scale(numerical_fea
ture))
feature_data_stand = pd.concat([numerical_feature_stand, categorica
l_feature], axis=1)
```

In [269]:
```
X_train_stand, X_test_stand, y_train_stand, y_test_stand = train_te
st_split(feature_data_stand, response_data, test_size=0.22, random_
state=4)
X_train_stand=X_train_stand.values
X_test_stand=X_test_stand.values
y_train_stand=y_train_stand.astype("int").values
y_test_stand=y_test_stand.astype("int").values
```

In [213]:
```
## Training and Tuning Process
def train_step(model,X_train, y_train, parameters):
    start_t = time.time()
    clf = GridSearchCV(model, parameters, cv=5, scoring='f1', n_job
s=-1)
    clf.fit(X_train, y_train)
    end_t = time.time()

    print('Time usage for training {}'.format(end_t-start_t))
    best_parameters = clf.cv_results_['params'][clf.best_index_]
    best_score = clf.best_score_
    return clf, best_parameters, best_score
```

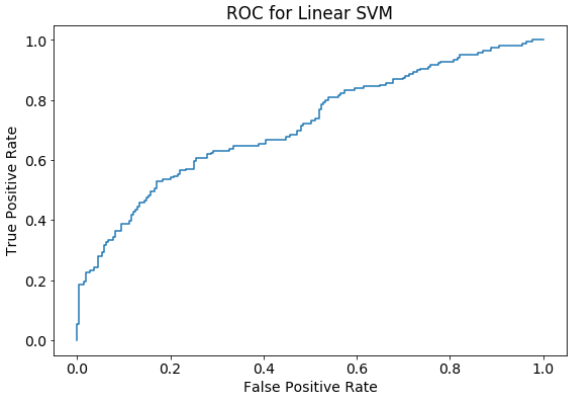
In [263]:
```
## Predicting process
def test_step(clf, X_test, y_test,model_name):

    y_pred = clf.predict(X_test)
    y_pred_prob = clf.decision_function(X_test)

    f1 = f1_score(y_test, y_pred)

    confusion = confusion_matrix(y_test, y_pred)

    roc_x, roc_y, _ = roc_curve(y_test, y_pred_prob)
    precision, recall, _ = precision_recall_curve(y_test, y_pred_pr
ob)
    AUC = auc(roc_x, roc_y)

    plt.figure(figsize=(20,6))
    plt.subplot(1,2,1)
    plt.plot(roc_x, roc_y)
    plt.xlabel('False Positive Rate')
    plt.ylabel('True Positive Rate')
    plt.title('ROC for {}'.format(model_name))

    plt.subplot(1,2,2)
    plt.plot(recall, precision)
    plt.xlabel('Recall')
    plt.ylabel('Precision')
    plt.title('PR Curve for recommendation')
    plt.title('PR Curve for {}'.format(model_name))

    plt.show()

    return f1, confusion, AUC, (roc_x, roc_y), (precision, recall)
```

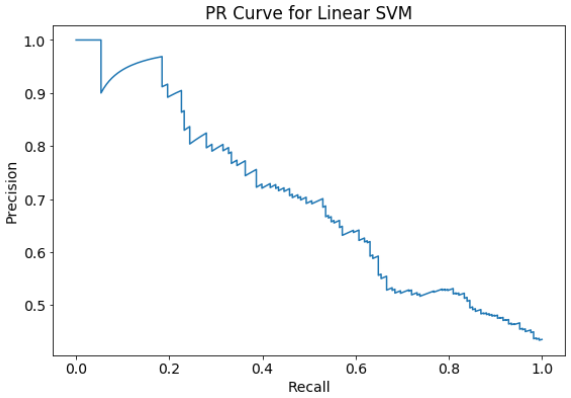
In [266]:
```
##linear SVM
params_linear = {'kernel':['linear'],'C':[5e-2, 1e-1,1e0,10,50]}
clf_svm_linear, best_params_svm_linear, best_score_svm_linear = tra
in_step(SVC(),X_train_stand.values, y_train_stand, params_linear)
best_params_svm_linear, best_score_svm_linear
```

```
Time usage for training 18.303667068481445
```

Out[266]:
```
({'C': 10, 'kernel': 'linear'}, 0.5825197359632809)
```

In [267]:
```
f1_svm_linear, confusion_svm_linear, AUC_svm_linear, _, _ = test_st
ep(clf_svm_linear, X_test_stand.values, y_test_stand,'Linear SVM')
confusion_svm_linear, f1_svm_linear, AUC_svm_linear
```




Out[267]:
```
(array([[190,  33],
        [ 90,  78]]), 0.5591397849462365, 0.7157537903053599)
```

In [290]:
```
#Accuracy rate for linear SVM
y_tr_pred_svm=clf_svm_linear.predict(X_train_stand)
sum(y_tr_pred_svm==y_train_stand)/1382
y_te_pred_svm=clf_svm_linear.predict(X_test_stand)
sum(y_te_pred_svm==y_test_stand)/391
```

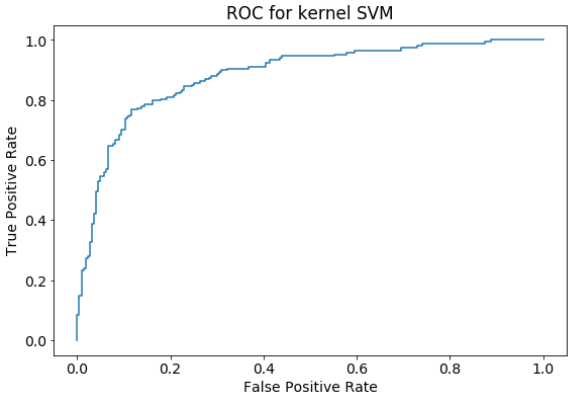
Out[290]:
```
0.6854219948849105
```

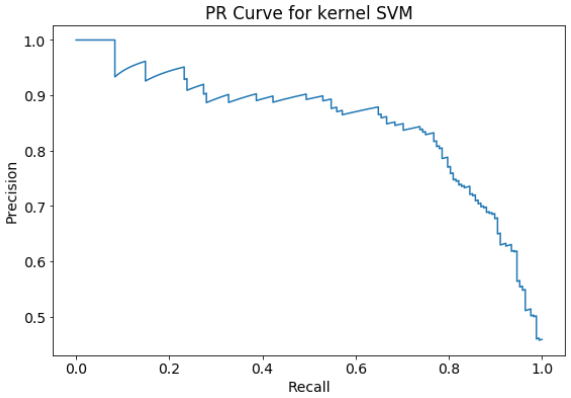
In [237]:
```
#Kernel SVM
params = {'kernel':['rbf','linear','poly','sigmoid'],'C':[5e-2, 1e-
1,1e0,10,50]}
clf_svm, best_params_svm, best_score_svm = train_step(SVC(),X_train
_stand.values, y_train_stand, params)
best_params_svm, best_score_svm
```

```
Time usage for training 26.77786612510681
```

Out[237]:
```
({'C': 50, 'kernel': 'poly'}, 0.7403163986129307)
```

In [264]:
```
f1_svm, confusion_svm, AUC_svm, _, _ = test_step(clf_svm, X_test_st
and.values, y_test_stand,'kernel SVM')
confusion_svm, f1_svm, AUC_svm
```




Out[264]:
```
(array([[203,  20],
        [ 56, 112]]), 0.7466666666666666, 0.8836216100790092)
```

In [291]:
```
#Accuracy Rate for Kernel SVM
y_te_pred_poly=clf_svm.predict(X_test_stand)
sum(y_te_pred_poly==y_test_stand)/391
```

Out[291]:
```
0.8056265984654731
```

In [13]:
```
# randome forest
parameters = {'max_depth':[10,30,50,100],'max_features':[30,50,80,1
14],'criterion':('gini','entropy')}
rdmfrst_model = RandomForestClassifier(random_state=0,n_estimators=
100)
clf0 = GridSearchCV(rdmfrst_model, parameters, cv=5,refit=True)
rdmfrst_model_final = clf0.fit(X_train, y_train).best_estimator_
```

In [15]:
```
# accurancy on training
clf0.best_score_
```

Out[15]:
```
0.8241678726483358
```

In [16]:
```
#accurancy on test
rdmfrst_model_final.score(X_test,y_test)
```

Out[16]:
```
0.8005115089514067
```

In [271]:
```
# randome forest time
parameters = {'max_depth':[10,30,50,100],'max_features':[30,50,80,1
14],'criterion':('gini','entropy')}
clf_ramdforest, best_params_ramdforest, best_score_ramdforest = tra
in_step(RandomForestClassifier(random_state=0,n_estimators=100),X_t
rain, y_train, parameters)
best_params_ramdforest, best_score_ramdforest
```

```
Time usage for training 40.85512661933899
```

Out[271]:
```
({'criterion': 'gini', 'max_depth': 30, 'max_features': 50}, 0.759
984934002361)
```

In [289]:
```
# AdaBoostClassifier
t1_ad=time.time()
learning_rate_tuning = [5,1,0.5,0.1,0.05]
n_estimators_tuning = [100,500,1000,5000,10000]
model_collection = []
for eta_rate,iteration_n in zip(learning_rate_tuning, n_estimators_
tuning):
    parameters = {'learning_rate':[eval('eta_rate')],'n_estimators'
:[eval('iteration_n')],'base_estimator__max_depth': [10, 30, 50, 10
0]}
    adaboost_model = AdaBoostClassifier(DecisionTreeClassifier(rand
om_state=0,max_depth=30,criterion='gini',max_features=50))
    clf1 = GridSearchCV(adaboost_model, parameters, cv=5,refit=True
)
    clf1_fit = clf1.fit(X_train, y_train).best_estimator_
    model_collection.append(clf1)
    print(clf1.best_score_,clf1.score(X_test,y_test))
t2_ad=time.time()

##Consuming time
print(t2_ad-t1_ad)
```

```
0.7626628075253257 0.7442455242966752
0.8400868306801736 0.8618925831202046
0.8400868306801736 0.8644501278772379
0.8350217076700435 0.8542199488491049
240.19182324409485
```

In [170]:
```
import tensorflow as tf
import numpy as np
import matplotlib.pyplot as plt
```

In [171]:
```
def tf_reset_graph():
    try:
        sess.close()
    except:
        pass
    new_graph = tf.Graph()
    return new_graph, tf.Session(graph=new_graph)

def tf_reset_default(): # this function is useful in case you forge
t run the command sess.close()
    try:
        sess.close()
    except:
        pass
    tf.reset_default_graph()
    return tf.Session()
```

In [174]:
```
##Neural Network
outputs=np.reshape(2*y_train_stand.values-1,(1382,1))
inputs=X_train_stand.values
inputs_test=X_test_stand.values
```

In [288]:
```
sess = tf_reset_default()

def create_model_relu():
    # create inputs
    input_ph = tf.placeholder(dtype=tf.float32, shape=[None, 114])
    output_ph = tf.placeholder(dtype=tf.float32, shape=[None, 1])

    # create variables
    W0 = tf.get_variable(name='W0', shape=[114, 114], initializer=t
f.contrib.layers.xavier_initializer())
    #W1 = tf.get_variable(name='W1', shape=[114, 114], initializer=
tf.contrib.layers.xavier_initializer())
    #W2 = tf.get_variable(name='W2', shape=[114, 114], initializer=
tf.contrib.layers.xavier_initializer())
    W3 = tf.get_variable(name='W3', shape=[114, 1], initializer=tf.
contrib.layers.xavier_initializer())

    b0 = tf.get_variable(name='b0', shape=[114], initializer=tf.con
stant_initializer(0.))
    #b1 = tf.get_variable(name='b1', shape=[40], initializer=tf.con
stant_initializer(0.))
    #b2 = tf.get_variable(name='b2', shape=[114], initializer=tf.co
nstant_initializer(0.))
    b3 = tf.get_variable(name='b3', shape=[1], initializer=tf.const
ant_initializer(0.))

    weights = [W0,W3]
    biases = [b0,b3]
    activations = [tf.nn.relu,tf.sigmoid]
```

```
    # create computation graph
    layer = input_ph
    for W, b, activation in zip(weights, biases, activations):
        layer = tf.matmul(layer, W) + b #linear activation between
layers
        if activation is not None:
            layer = activation(layer) #within each layers, apply se
fl-defined activation
    output_pred = layer
    return input_ph, output_ph, output_pred
t1=time.time()
input_ph, output_ph, output_pred= create_model_relu()
# create loss
#loss = tf.reduce_mean(tf.reduce_sum(-tf.multiply(output_ph,tf.log(
output_pred)))-tf.multiply((1-output_ph),tf.log(1-output_pred)))
# create optimizer
loss= tf.reduce_mean(tf.log(1+tf.exp(-tf.multiply(output_pred,outpu
t_ph))))
opt = tf.train.AdamOptimizer().minimize(loss)

# initialize variables
sess.run(tf.global_variables_initializer())
# create saver to save model variables
saver = tf.train.Saver()

# run training
batch_size = 32
for training_step in range(50000):
    # get a random subset of the training data
    indices = np.random.randint(low=0, high=len(outputs), size=batc
h_size)
    input_batch = inputs[indices,:]
    output_batch = outputs[indices]
    # run the optimizer and get the mse
    _, loss_run = sess.run([opt, loss], feed_dict={input_ph: input_
batch, output_ph: output_batch})

    # print the mse every so often
    if training_step % 10000 == 0:
        print('{0:04d} loss: {1:.3f}'.format(training_step, loss_ru
n))
        print(_)
        saver.save(sess, '/tmp/model.ckpt')
t2=time.time()
print(t2-t1)
```

```
0000 loss: 0.744
None
10000 loss: 0.575
None
20000 loss: 0.527
None
30000 loss: 0.563
None
40000 loss: 0.598
None
33.169519901275635
```

In [207]:
```
sess = tf_reset_default()

# create the model
input_ph, output_ph, output_pred = create_model_relu()

# restore the saved model
saver = tf.train.Saver()
saver.restore(sess, "/tmp/model.ckpt")

output_pred_run = sess.run(output_pred, feed_dict={input_ph: inputs
})
output_pred_test = sess.run(output_pred, feed_dict={input_ph: input
s_test})
```

```
INFO:tensorflow:Restoring parameters from /tmp/model.ckpt
```

In [208]:
```
#Accuracy rate
pred_class=output_pred_run>=0.5
sum(pred_class==np.reshape(y_train_stand.values,(1382,1)))/1382
```

Out[208]:
```
array([0.95441389])
```

In [279]:
```
pred_test_class=output_pred_test>=0.5
sum(pred_test_class==np.reshape(y_test_stand,(391,1)))/391
```

Out[279]:
```
array([0.83375959])
```

In [282]:
```
## F1 score
f1_nn_train = f1_score(y_train_stand,pred_class)
f1_nn_test=f1_score(y_test_stand,pred_test_class)
print(f1_nn_train)
print(f1_nn_test)
```

```
0.9426751592356688
0.7949526813880127
```

In [280]:
```
precision_nn, recall_nn, _ = precision_recall_curve(y_test_stand, o
utput_pred_test)
plt.plot(recall_nn, precision_nn)
plt.xlabel('Recall')
plt.ylabel('Precision')
plt.title('PR Curve for recommendation')
plt.title('PR Curve for neural network')
```

Out[280]: Text(0.5, 1.0, 'PR Curve for neural network')