Risk Assessment for Life Insurance via Predictive Models

Ni **Guan**, Ruian **Ge**, Xiaoxu **Ruan**, Zhilun **Zhang**, Zhiwei **Ma**

1. Motivation

The conventional application procedure of life insurance is lengthy. Clients used to take 30 days or more to provide extensive information to prove their eligibility and risk levels, which is one main reason why the holding rate, only around 40% households in the United States, always stayed low.

To expand this business, machine learning is promising to be leveraged. With its virtues of more validity and less labor-dependence, we built a predictive model to surpass the old system on both the new and existing clients. Our model scrutinizes the input parameters of the client attributes, and gives out ordinal outcomes as their risk levels.

Firstly, we developed four candidate models to predict risk levels; then, cross-validation was deployed to improve these models; and finally, key metrics were selected to indicate the best model. Instead of classifying client data into multiple categories, our method mainly focuses on premium clients who theoretically have much lower risk of critical illness than ordinary people. Our model distinguish these clients and recommend their applications to be facilitated.

2. Analytics

2.1 Data collection

Prudential is one of the largest life insurance issuers in the United States, so we researched on its published marketing data. This dataset is comprised of 12 independent variables of client attributes, with the outcome Response as their respective risk level.

Variable	**Description**
ID	A unique identifier associated with an application
Product_Info_1-7	A set of normalized variables relating to the product applied for
Ins_Age	Normalized age of applicant
Ht	Normalized height of applicant
Wt	Normalized weight of applicant
BMI	Normalized BMI of applicant
Employment_Info_1-6	A set of normalized variables relating to the employment history of the applicant
InsuredInfo_1-6	A set of normalized variables providing information about the applicant
Insurance_History_1-9	A set of normalized variables relating to the insurance history of the applicant
Family_Hist_1-5	A set of normalized variables relating to the family history of the applicant
Medical_History_1-41	A set of normalized variables relating to the medical history of the applicant
Medical_Keyword_1-48	A set of dummy variables relating to the presence of/absence of a medical keyword being associated with the application
Response	The target variable, an ordinal variable relating to the final decision associated with an application, from highest “1” to lowest “8”

2.2 Pre-processing

We found the dependent variable “Response” too dispersed and concentrated in "6" and "8", where “8” represents those premium clients who are our target, so we referred to the isB method that re-label "8" as "1" and all the others as "0" to rebalance the data.

Response	Count
8	19489
6	11233
7	8027
2	6552
1	6207
5	5432
4	1428
3	1013

We dropped variables such as “BMI” to prevent the overfitting issue, as it’s directly obtained from “Ht” and “Wt”. Impertinent variables such as “ID” are dropped as well. As for the missing data, we dropped the variables whose missing rate > 40%, and filled the vacancies of the other variables with their respective mean.

We then randomly split the data into 70% (41566) as the train set and 30% (17815) as the test set.

2.3 Modeling

We used LDA, CART, Random Forest, and XGBoost as tentative models.

Firstly, we set a Baseline Model, which has an accuracy at 67.0783%.

Secondly, we determined four metrics, namely the Accuracy, Recall (TPR), Precision, and FPR, to evaluate performance of our models (see Comparative Analysis).

Thirdly, we adjusted the key parameters to improve the models. For instance, we optimized the CART model by tuning the cpp_alpha using HalvingGridSearchCV, as it’s proven faster than GridSearchCV; we adjusted the hyperparameter min_sample_leaf of the Random Forest model; XGBoost model was also calibrated on learning rate.

Finally, we finitely iterated these models using the aforementioned parameters on the training data, and culminated in their respective best performance.

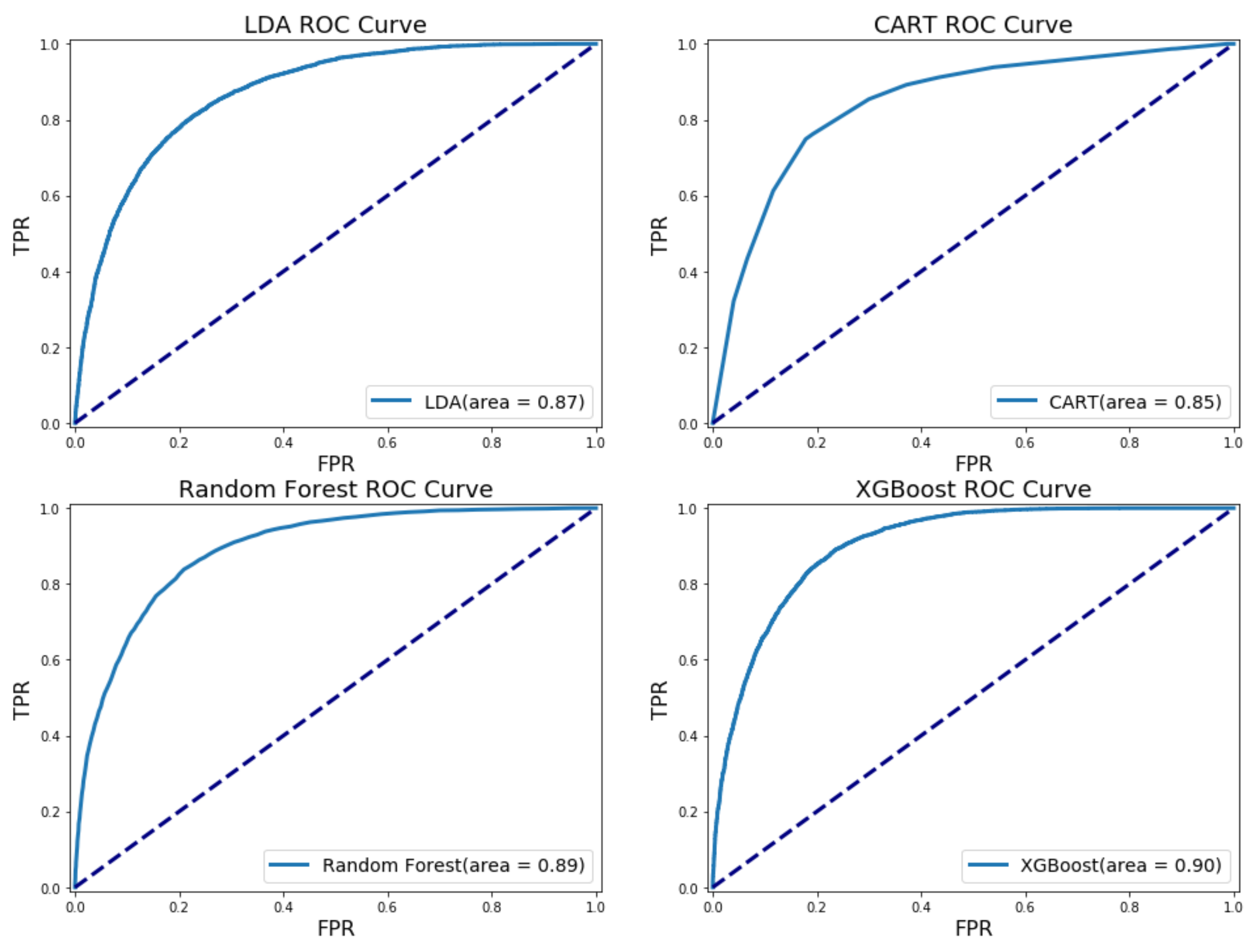
2.4 Comparative Analysis

From the result table, XGBoost model has the best Accuracy and Recall; CART model has the best FPR and a good Recall; and Random Forest model has the best Precision and a good Accuracy.

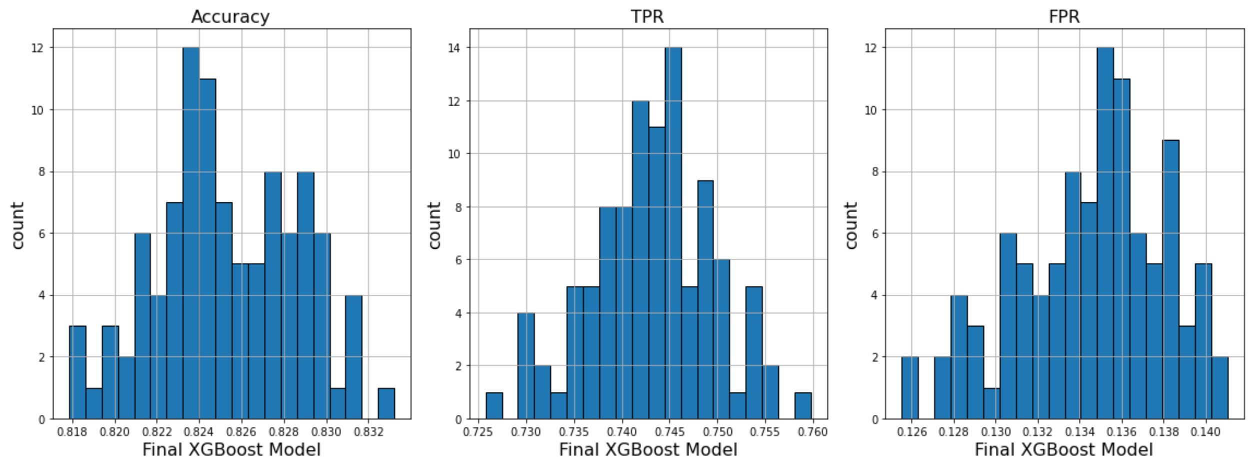
	LDA	CART	Random Forest	XGBoost
Accuracy	0.805782	0.802582	0.815605	0.825091
Recall	0.659335	0.740494	0.657971	0.742882
FPR	0.122343	0.166946	0.107029	0.134561
Precision	0.725652	0.685232	0.75107	0.730427
AUC	0.871432	0.855899	0.890339	0.904031

We plotted the ROC and calculated the AUC for each model. From the ROC curves, all four models performed much better than the baseline model, since they have higher TPR and lower FPR. In terms to AUC values, the XGBoost model is the largest (0.90), followed by Random Forest (0.89), LDA (0.87), and CART (0.85). Therefore, the XGBoost model has the best discriminative ability to distinguish prime clients.

Considering the profit-oriented nature of Prudential as a business entity, our top concern should be identifying those least risky clients, because they have the lowest possibility of incurring insurance compensation, but the highest contribution to the revenue from the insurance fees. Besides, our focus on premium clients and rebalancing process defined our critical goal as accuracy in the scope of positive samples, so Recall is the most appropriate metric to evaluate our models, along with AUC. Therefore, we appointed our XGBoost model as our final model.



We then bootstrapped the selected model for final trial. From the visualized histograms, the model achieved Accuracy from 0.818 to 0.833, TPR from 0.725 to 0.760, and FPR from 0.126 to 0.141.

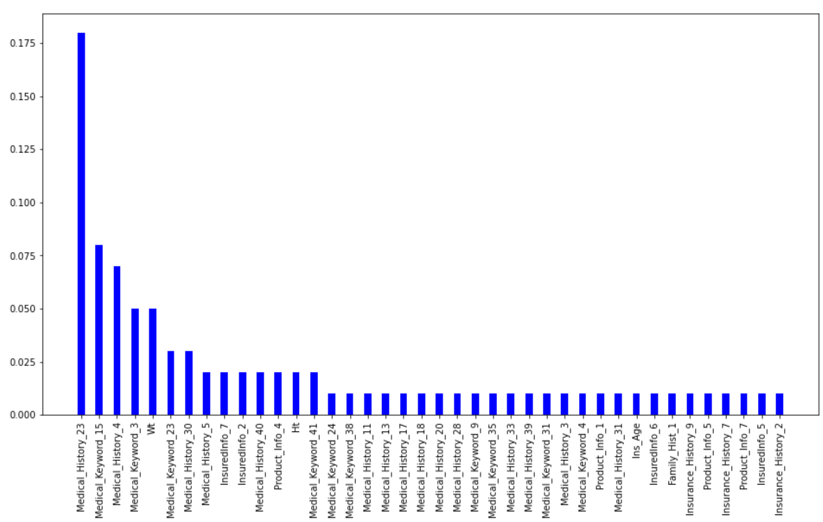


We also calculated the confidence intervals of three metrics. The 95% confidence interval of Accuracy is [-0.00633315, 0.00628964], that of TPR is [-0.01323232, 0.01209476], and that of FPR is [-0.00710744, 0.00552891]. Thus we concluded that our final model has significantly smaller FPR, larger accuracy and TPR, which indicates its excellent performance.

3. Impact

3.1 Client Risk Classification

We identified the top 15 features of the greatest importance to the final model, by defining the check_importance function and plotting the importance level of each independent variable. This helped to designate which indexes Prudential should pay higher attention to during risk assessment. For instance, "Medical_History_23" has significantly higher importance than other features, hence Prudential should investigate more strictly on the medical history of an applicant. There are 10 important features related to medical issues, which means those diseases clients carry have great influence on their risk levels.



3.2 Enterprise Operational Efficiency

With this model, Prudential will identify low risk clients precisely and efficiently, and accordingly accelerate their applications. These will benefit Prudential in product upgrading and client experience, and result in an increased market share and penetration. Moreover, our work can streamline the traditional insurance industry, both on product designing and price differentiating. In greater prospect, our model may to some extent ameliorate the societal burden on our healthcare system.

Acknowledgement

This research is conducted under the earnest guidance and instruction of Professor Paul Grigas and GSI Hyungki Im and Hong-Seok Choe. We would like to extend our sincere gratitude.

242_Final 2

December 14, 2021

[1]:
```python
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import statsmodels.formula.api as smf
from sklearn.model_selection import GridSearchCV
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import confusion_matrix
```

[2]:
```python
data = pd.read_csv('train.csv')
```

[3]:
```python
data.head()
```

[3]:
```
   Id  Product_Info_1 Product_Info_2  Product_Info_3  Product_Info_4  \
0   2               1             D3              10        0.076923
1   5               1             A1              26        0.076923
2   6               1             E1              26        0.076923
3   7               1             D4              10        0.487179
4   8               1             D2              26        0.230769

   Product_Info_5  Product_Info_6  Product_Info_7   Ins_Age        Ht  …  \
0               2               1               1  0.641791  0.581818  …
1               2               3               1  0.059701  0.600000  …
2               2               3               1  0.029851  0.745455  …
3               2               3               1  0.164179  0.672727  …
4               2               3               1  0.417910  0.654545  …

   Medical_Keyword_40  Medical_Keyword_41  Medical_Keyword_42  \
0                   0                   0                   0
1                   0                   0                   0
2                   0                   0                   0
3                   0                   0                   0
4                   0                   0                   0

   Medical_Keyword_43  Medical_Keyword_44  Medical_Keyword_45  \
0                   0                   0                   0
1                   0                   0                   0
2                   0                   0                   0
3                   0                   0                   0
```

```
4                    0                    0                    0

   Medical_Keyword_46  Medical_Keyword_47  Medical_Keyword_48  Response
0                    0                    0                    0         8
1                    0                    0                    0         4
2                    0                    0                    0         8
3                    0                    0                    0         8
4                    0                    0                    0         8

[5 rows x 128 columns]
```

[4]:
```
data.shape
```

[4]:
```
(59381, 128)
```

[5]:
```
#Reset response data to get balance
data['Response'].value_counts()
```

[5]:
```
8    19489
6    11233
7     8027
2     6552
1     6207
5     5432
4     1428
3     1013
Name: Response, dtype: int64
```

[6]:
```
data['risk'] = pd.Series([1 if x == 8 else 0 for x in data['Response']], index
↪= data['Response'].index)
```

[7]:
```
data.drop('Response',axis = 1, inplace = True)
```

[8]:
```
missing = data.isnull().sum()/len(data)
print(missing[missing>0].sort_values(ascending = False))
```

```
Medical_History_10     0.990620
Medical_History_32     0.981358
Medical_History_24     0.935990
Medical_History_15     0.751015
Family_Hist_5          0.704114
Family_Hist_3          0.576632
Family_Hist_2          0.482579
Insurance_History_5    0.427679
Family_Hist_4          0.323066
Employment_Info_6      0.182786
Medical_History_1      0.149694
Employment_Info_4      0.114161
```

```
Employment_Info_1      0.000320
dtype: float64
```

[9]:
```
#drop missing data
data = data.dropna(thresh=data.shape[0]*0.6,how='all',axis = 1)
```

[10]:
```
data.shape
```

[10]:
```
(59381, 120)
```

[11]:
```
data = data.fillna(data.mean())
```

[12]:
```
#drop unimportant data
data.drop('Product_Info_2',axis = 1, inplace = True)
```

[13]:
```
#BMI is related to weight and height
data.drop('BMI',axis = 1, inplace = True)
```

[14]:
```
data.drop('Id',axis = 1,inplace = True)
```

[15]:
```
from sklearn.model_selection import train_test_split
train, test = train_test_split(data, test_size=0.3, random_state=99)
train = train.reset_index(drop = True)
test = test.reset_index(drop = True)
train.shape, test.shape
```

[15]:
```
((41566, 117), (17815, 117))
```

[16]:
```
#baseline model accuracy
ACC = 1 - np.sum(test['risk'])/len(test['risk'])
ACC
```

[16]:
```
0.6707830479932642
```

[17]:
```
y_train = train['risk']
X_train = train.drop(['risk'],axis = 1)
y_test = test['risk']
X_test = test.drop(['risk'],axis = 1)
```

[18]:
```
# We use LDA, CART, Random Forest, XGBoost Model in this case
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from xgboost import XGBClassifier
from sklearn.experimental import enable_halving_search_cv
from sklearn.model_selection import HalvingRandomSearchCV, HalvingGridSearchCV
from sklearn.metrics import accuracy_score, recall_score, confusion_matrix,␣
 ↪precision_score
```

```python
from sklearn.metrics import roc_curve, auc
import warnings
# Defining the metrics function to return the result of the model
def metrics(y_true,y_pred,y_prob):
    cm = confusion_matrix(y_true,y_pred)
    accuracy = accuracy_score(y_true,y_pred)
    recall = recall_score(y_true,y_pred)
    precision = precision_score(y_true,y_pred)
    FPR = cm[0,1]/sum(cm[0])
    fpr, tpr, _ = roc_curve(y_true, y_prob)
    roc_auc = auc(fpr, tpr)
    return [accuracy,recall,FPR,precision,roc_auc,fpr,tpr]
# Defining the check_importance function to return the feature importance
def check_importance(model, X_train):
  #Checking importance of features
    importances = model.feature_importances_
  #List of columns and their importances
    importance_dict = {'Feature' : list(X_train.columns),
                'Feature Importance' : importances}
    importance_df = pd.DataFrame(importance_dict)
  #Rounding it off to 2 digits as we might get exponential numbers
    importance_df['Feature Importance'] = round(importance_df['Feature␣
↪Importance'],2)
    return importance_df.sort_values(by=['Feature Importance'],ascending=False)
# Defining the model
def model(x_train, y_train, x_test, y_test, model_name = 'LDA',scoring =␣
↪'accuracy'):
    if model_name == 'LDA':
        clf = LinearDiscriminantAnalysis()
        clf.fit(x_train,y_train)
        pred = clf.predict(x_test)
        prob = clf.predict_proba(x_test)[:,1]
        return metrics(y_test,pred,prob)
    if model_name == 'CART':
        clf = DecisionTreeClassifier()
        # Tuning the hyper parameter ccp_alpha
        grid_values = {'ccp_alpha': np.linspace(0.0, 0.002, 21),
#                 'min_samples_leaf': [3,4,5,6,7],
#                 'min_samples_split': [10,15,20,25,30],
              'random_state': [1]}
        # Using HalvingGridSearchCV method to tune the parameter. Faster than␣
↪GridSearch
        cv = HalvingGridSearchCV(clf,param_grid =␣
↪grid_values,scoring=scoring,cv=5)
        cv.fit(x_train,y_train)
        # printing the best parameters of the estimator
        print(cv.best_params_,flush=True)
```

```
        pred = cv.best_estimator_.predict(x_test)
        prob = cv.best_estimator_.predict_proba(x_test)[:,1]
        # return the metrics and feature importance of the model
        return metrics(y_test,pred,prob), check_importance(cv.
 ↪best_estimator_,x_train)
    if model_name == 'Random Forest':
        clf = RandomForestClassifier()
        # Tuning the hyperparameter min_sample_leaf. n_estimators and␣
 ↪max_features can also be tuned
        grid_values = {#'n_estimators': np.linspace(100,1000,10,dtype=int),
#                        'max_features': np.linspace(5,50,10),
                       'min_samples_leaf': np.linspace(1,5,5,dtype=int),
                       'random_state': [1],
                       'verbose': [0]}
        cv = HalvingGridSearchCV(clf, param_grid= grid_values, scoring =␣
 ↪scoring,cv=5)
        cv.fit(x_train,y_train)
        # printing the best parameters of the estimator
        print(cv.best_params_,flush=True)
        pred = cv.best_estimator_.predict(x_test)
        prob = cv.best_estimator_.predict_proba(x_test)[:,1]
        return metrics(y_test,pred,prob),check_importance(cv.
 ↪best_estimator_,x_train)
    if model_name == 'XGB':
        clf = XGBClassifier(use_label_encoder=False)
        # Tuning the learning rate in this case
        grid_values = {#'n_estimators': np.linspace(100,1000,10,dtype=int),
#                        'max_features': np.linspace(5,100,20),
#                        'min_samples_leaf': np.linspace(2,20,10),
                       'learning_rate': np.linspace(0.02,0.2,10),
                       'random_state': [1],
                       'verbosity':[0]}
        cv = HalvingGridSearchCV(clf, param_grid=grid_values,scoring=scoring,␣
 ↪cv=5)
        cv.fit(x_train,y_train)
        # printing the best parameters of the estimator
        print(cv.best_params_,flush=True)
        pred = cv.best_estimator_.predict(x_test)
        prob = cv.best_estimator_.predict_proba(x_test)[:,1]
        return metrics(y_test,pred,prob),check_importance(cv.
 ↪best_estimator_,x_train)
def plot_feature_importance(model, X_train):
  # PLotting features vs their importance factors
    fig = plt.figure(figsize = (15, 8))
  # Extracting importance values
```

```
    values =check_importance(model, X_train)[check_importance(model,␣
↪X_train)['Feature Importance']>0]['Feature Importance'].values
  # Extracting importance features
    features = check_importance(model, X_train)[check_importance(model,␣
↪X_train)['Feature Importance']>0]['Feature'].values
    plt.bar(features, values, color ='blue',
          width = 0.4)
    plt.xticks( rotation='vertical')
    plt.show()
```

[19]:
```
# Run the models and fit the data then make prediction and lastly evaluate
ld_me = model(X_train,y_train,X_test,y_test,'LDA')
cart_me, cart_fi = model(X_train,y_train,X_test,y_test,'CART')
rf_me, rf_fi = model(X_train,y_train,X_test,y_test,'Random Forest')
xgb_me, xgb_fi = model(X_train,y_train,X_test,y_test,'XGB')
```

```
{'ccp_alpha': 0.0011, 'random_state': 1}
{'min_samples_leaf': 2, 'random_state': 1, 'verbose': 0}
{'learning_rate': 0.12000000000000002, 'random_state': 1, 'verbosity': 0}
```

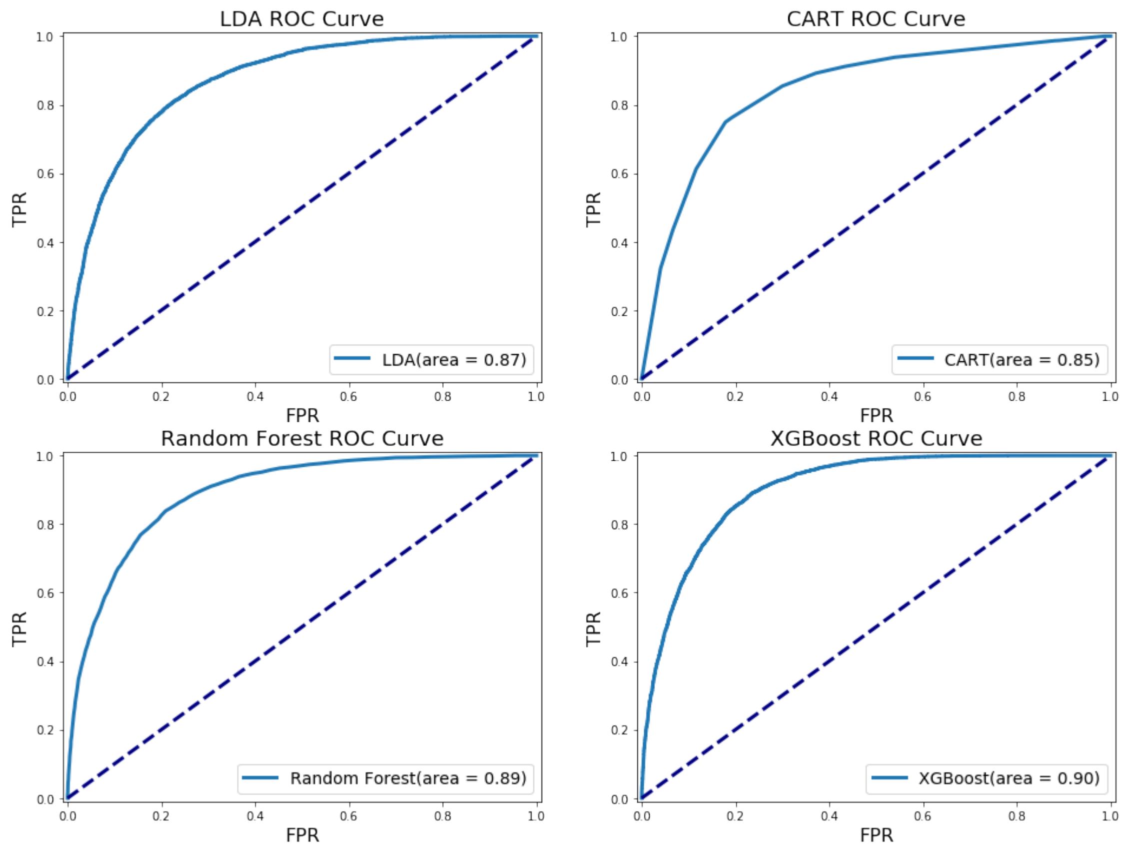
[20]:
```
# Printing out the metrics for each model
metric_all = pd.DataFrame()
col_name = ['LDA','CART','Random Forest','XGBoost']
for me,name in zip([ld_me,cart_me,rf_me,xgb_me],col_name):
    metric_all[name] = me[0:5]
metric_all.index = ['Accuracy','Recall','FPR','Precision','AUC']
metric_all
```

[20]:

	LDA	CART	Random Forest	XGBoost
Accuracy	0.805782	0.802582	0.815605	0.825091
Recall	0.659335	0.740494	0.657971	0.742882
FPR	0.122343	0.166946	0.107029	0.134561
Precision	0.725652	0.685232	0.751070	0.730427
AUC	0.871432	0.855899	0.890339	0.904031

[21]:
```
list=[221,222,223,224]
plt.figure(figsize=(16, 12))
for me,name,i in zip([ld_me,cart_me,rf_me,xgb_me],col_name,range(4)):
    tpr = me[-1]
    fpr = me[-2]
    roc_auc = me[-3]
    a = list[i]
    plt.subplot(a)
    plt.title(name+' '+'ROC Curve', fontsize=18)
    plt.xlabel('FPR', fontsize=16)
    plt.ylabel('TPR', fontsize=16)
    plt.xlim([-0.01, 1.01])
    plt.ylim([-0.01, 1.01])
```

```
    plt.plot(fpr, tpr, lw=3, label=name+'(area = {:0.2f})'.format(roc_auc))
    plt.plot([0, 1], [0, 1], color='navy', lw=3, linestyle='--')
    plt.legend(loc='lower right', fontsize=14)
    plt.show
```



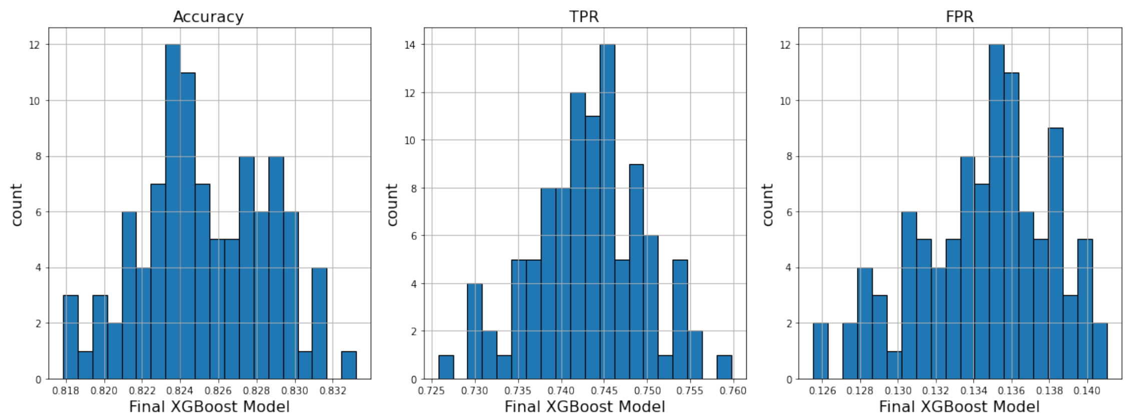
[21]:
```
#Bootstrap for final model
import time
def bootstrap_validation(test_data, test_label, model, metrics, sample=100,␣
↪random_state=99):
    tic = time.time()
    n_sample = sample
    n_metrics = 3
    output_array=np.zeros([n_sample, n_metrics])
    output_array[:]=np.nan
    print(output_array.shape)
    for bs_iter in range(n_sample):
        bs_index = np.random.choice(test_data.index, len(test_data.index),␣
↪replace=True)
        bs_data = test_data.loc[bs_index]
        bs_label = test_label.loc[bs_index]
```

```
        bs_predicted = model.predict(bs_data)
        bs_prob = model.predict_proba(bs_data)[:,1]
        output_array[bs_iter,:] = metrics(bs_label,bs_predicted,bs_prob)[0:3]
        if bs_iter % 20 == 0:
            print(bs_iter, time.time()-tic)
    output_df = pd.DataFrame(output_array)
    return output_df
```

[22]:
```
final_model = XGBClassifier(use_label_encoder=False,learning_rate=0.12,␣
 ↪random_state=1, verbosity=0)
final_model.fit(X_train,y_train)
bs_df = bootstrap_validation(X_test,y_test,final_model,metrics)
```

```
(100, 3)
0 0.06268477439880371
20 1.1571509838104248
40 2.2557969093322754
60 3.368259906768799
80 4.476579904556274
```

[23]:
```
#visualization of the outcome
fig, ax = plt.subplots(1,3,figsize = (16,6))
for name, i in zip(['Accuracy','TPR','FPR'], range(3)):
    ax[i].set_xlabel('Final XGBoost Model',fontsize=16)
    ax[i].set_ylabel('count',fontsize=16)
    ax[i].hist(bs_df[i],bins=20,edgecolor = 'black')
    ax[i].set_title(name,fontsize=16)
    ax[i].grid()
fig.tight_layout()
```



[24]:
```
#the confidence interval of three factors
CI1= np.quantile(bs_df.iloc[:,0]-xgb_me[0],np.array([0.025,0.975]))
print("The 95-percent confidence interval of acc is %s" % CI1)
```

```python
CI2= np.quantile(bs_df.iloc[:,1]-xgb_me[1],np.array([0.025,0.975]))
print("The 95-percent confidence interval of acc is %s" % CI2)
CI3= np.quantile(bs_df.iloc[:,2]-xgb_me[2],np.array([0.025,0.975]))
print("The 95-percent confidence interval of acc is %s" % CI3)
```

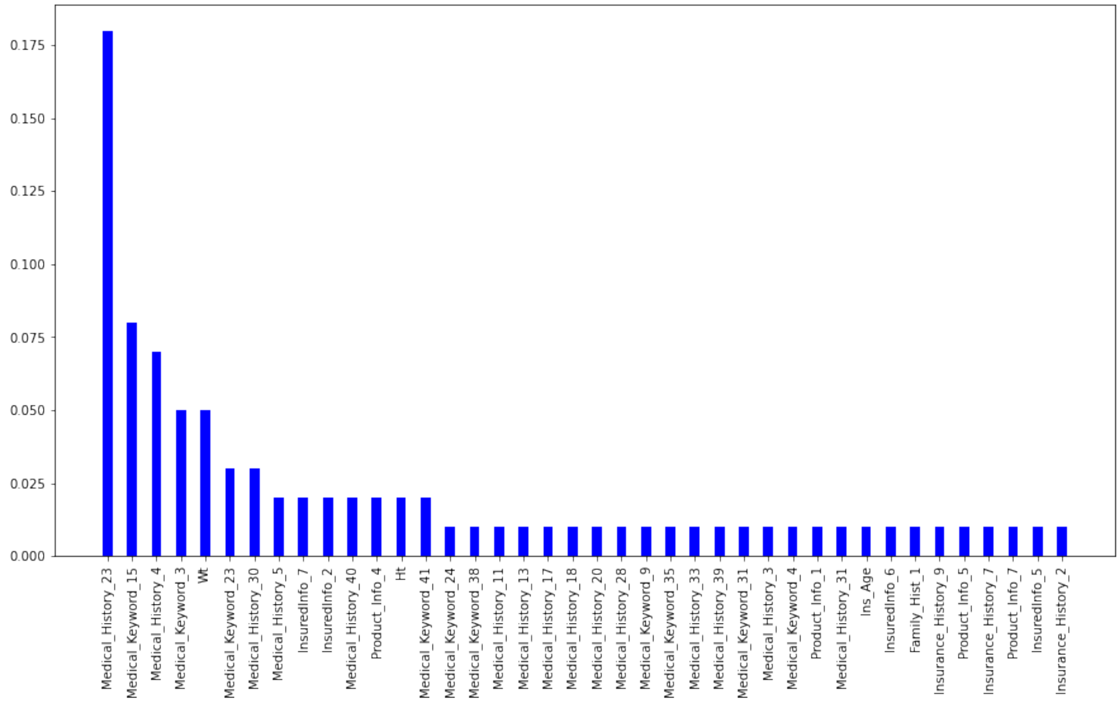
```
The 95-percent confidence interval of acc is [-0.00633315  0.00628964]
The 95-percent confidence interval of acc is [-0.01323232  0.01209476]
The 95-percent confidence interval of acc is [-0.00710744  0.00552891]
```

[27]:
```python
# Show the Top 15 features with greatest importance of the final model
check_importance(final_model,X_train).head(15)
```

[27]:

	Feature	Feature Importance
51	Medical_History_23	0.18
82	Medical_Keyword_15	0.08
34	Medical_History_4	0.07
70	Medical_Keyword_3	0.05
8	Wt	0.05
90	Medical_Keyword_23	0.03
57	Medical_History_30	0.03
35	Medical_History_5	0.02
21	InsuredInfo_7	0.02
16	InsuredInfo_2	0.02
66	Medical_History_40	0.02
2	Product_Info_4	0.02
7	Ht	0.02
108	Medical_Keyword_41	0.02
91	Medical_Keyword_24	0.01

[25]:
```python
# plot the feature importance
plot_feature_importance(final_model,X_train)
```



0.175
0.150
0.125
0.100
0.075
0.050
0.025
0.000
Medical_History_23
Medical_Keyword_15
Medical_History_4
Medical_Keyword_3
Wt
Medical_Keyword_23
Medical_History_30
Medical_History_5
InsuredInfo_7
InsuredInfo_2
Medical_History_40
Product_Info_4
Ht
Medical_Keyword_41
Medical_Keyword_24
Medical_Keyword_38
Medical_History_11
Medical_History_13
Medical_History_17
Medical_History_18
Medical_History_20
Medical_History_28
Medical_Keyword_9
Medical_Keyword_35
Medical_History_33
Medical_History_39
Medical_Keyword_31
Medical_History_3
Medical_Keyword_4
Product_Info_1
Medical_History_31
Ins_Age
InsuredInfo_6
Family_Hist_1
Insurance_History_9
Product_Info_5
Insurance_History_7
Product_Info_7
InsuredInfo_5
Insurance_History_2

[ ]: