# ADVANCE DATA MINING

# MEDICAL APPOINTMENT NO SHOW

Yuehchao Wu

DePaul University

TABLE OF CONTENTS

## Introduction

In this Project, I want to explore and analyze the medical appointment dataset from Kaggle using the basic and advanced machine learning algorithms that are suitable for classifying binary parameter of interest. The investigated algorithms are Naïve Bayes, Ensemble - begging, boosting, and stacking, and Support Vector Machine.

The goals are to find out why patients miss their doctor appointments and find the best algorithms with tuned parameters to be used for predicting target variable: show and no show, specifically no show, based on the given information. The investigated algorithms then will be evaluated based on the prediction/classification performance and other characteristics such as Time to build the model, Interpretability… and so on.

The report includes non-technical summary and technical summary. Non-technical part will briefly talked about the analysis being implemented and summarize the finding for non-technical audience; whereas, technical part will provide detail of the analysis on the medical appointment dataset. The analyses for the technical part are divided into 6 subsections: data description, data cleaning, data analysis, experimental results, experimental analysis, and conclusion.

## Non-Technical Summary

The purpose of the project is find out why patients miss their doctor appointments and find the best algorithms with tuned parameters to be used for identifying no-show cases based on the given information.

The features (gender', 'age', 'wait period', 'SMS-received', 'diabetes', ' hypertension', 'alcoholism', 'Handicap', and 'scholarship') were selected for investigation after data has been thoroughly prepared and explored. For example, I removed the imbalance problem using an oversampling technique called smote.

Then, several machine-learning algorithms were investigated. The technique called grid searching was used to optimize the parameter setting during the model searching process. The appropriate performance metric to be used is recall. Recall measures how many no-show cases our model can find. Linear support vector machine turned out to have best performance in term of recall for no-show case. The recall value for it is 77%. The second best is stacking (71%). As for general accuracy, gradient boosting (65%) and random forest (64%) out perform others. Other evaluation techniques, such as roc curve, pair t test, are also conducted during the model evaluation. The conclusion that, support vector machine is chosen as our solution for the study is made due to the significantly better recall value for no-show case. The downside of support vector machine is that it takes very long to generate a model and make prediction. Last but not least, the top three most important features for the classifying no-show and show based on gradient boosting tree are: age, wait period (within two days), and wait period (with in a month).

## Technical summary

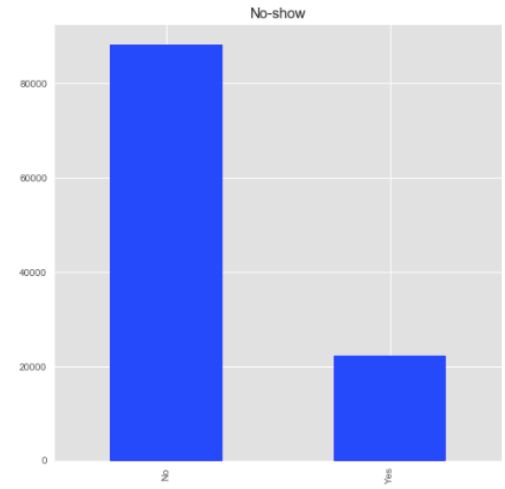
### Data Description

The Medical appointment dataset (KaggleV2-May-2016.csv) used in this project is attained from website named Kaggle. The dataset contain 110527 observations and has 14 variables. The parameter of interest, 'No-show', is categorical (binary) and labeled (No-show: yes and show: no). The rest of the variables will be our potential predictors, containing both categorical and continuous factors.

The following images tell us the basic structure and information of raw (has not been processed) dataset and that our data is imbalanced. ~ 80% of observations are show and only ~20% are no-show.

```
#look at data struture
med_Aptdata.info()
print  'data struture(row, colunm) : ', med_Aptdata.shape
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 110527 entries, 0 to 110526
Data columns (total 14 columns):
PatientId         110527 non-null float64
AppointmentID     110527 non-null int64
Gender            110527 non-null object
ScheduledDay      110527 non-null object
AppointmentDay    110527 non-null object
Age               110527 non-null int64
Neighbourhood     110527 non-null object
Scholarship       110527 non-null int64
Hipertension      110527 non-null int64
Diabetes          110527 non-null int64
Alcoholism        110527 non-null int64
Handcap           110527 non-null int64
SMS_received      110527 non-null int64
No-show           110527 non-null object
dtypes: float64(1), int64(8), object(5)
memory usage: 11.8+ MB
data struture(row, colunm) :  (110527, 14)
```




```
Out[34]: No-show
No     79.806561
Yes    20.193439
dtype: float64
```

When faced the imbalanced dataset, the limitations or challenges of machine learning techniques may raised. There is high probability that we will get unsatisfactory results toward minority class (no-show = yes).

## Data Cleaning

In this section, I will show how the dataset is prepared by providing python code and output from Jupiter notebook. The preprocess includes checking missing values, deleting irrelevant records and attributes, and creating new attributes.

Missing Value

```
#check missing values
print "attribute with missing values: \n"
print [col for col in med_Aptdata.columns if med_Aptdata[col].isnull().any()]
```

```
attribute with missing values:

[]
```

No missign value

After running the code, we can see that there aren't any missing values found in the dataset.

Irrelevant records

```
med_Aptdata.describe()
```

| | Age |
|---|---|
| count | 110527.000000 |
| mean | 37.088874 |
| std | 23.110205 |
| min | -1.000000 |
| 25% | 18.000000 |
| 50% | 37.000000 |
| 75% | 55.000000 |
| max | 115.000000 |

```
med_Aptdata[med_Aptdata.Age < 0].count()
```

```
Gender            1
ScheduledDay      1
AppointmentDay    1
Age               1
Neighbourhood     1
Scholarship       1
Hipertension      1
Diabetes          1
Alcoholism        1
Handcap           1
SMS_received      1
No-show           1
dtype: int64
```

age min = -1 make no sense.

Looking at the five number summaries of data, attribute 'age' has invalid record (age = -1).

```
#only ONE record has age < 0 so remove it wont hurt
med_Aptdata= med_Aptdata[med_Aptdata['Age'] >= 0]
```

The solution is to remove it since there is only one irrelevant record.

Irrelevant attribute and relabeling

```
#drop useless columns
med_Aptdata.drop(['PatientId','AppointmentID'], axis=1, inplace = True)
```

```
#relable categorical datatype to str
med_Aptdata.Scholarship=med_Aptdata.Scholarship.astype(str)
med_Aptdata.Hipertension=med_Aptdata.Hipertension.astype(str)
med_Aptdata.Diabetes=med_Aptdata.Diabetes.astype(str)
med_Aptdata.Alcoholism=med_Aptdata.Alcoholism.astype(str)
med_Aptdata.Handcap=med_Aptdata.Handcap.astype(str)
med_Aptdata.SMS_received=med_Aptdata.SMS_received.astype(str)
```

Next, I dropped the patient ID and appointment ID since it obviously won't be useful predictors for predicting show or no-show. Then, I re-labeled the attributes, 'scholarship', 'hypertension', 'diabetes', 'alcoholism', 'handicap', and 'SMS_received', as 'str' (categorical) since they were miss-labeled as continuous.

New feature

```
#new colum, waitday  (day between ScheduledDay  and AppointmentDay)
import numpy as np

# Converts the two variables to datetime variables
med_Aptdata['ScheduledDay'] = pd.to_datetime(med_Aptdata['ScheduledDay'])
med_Aptdata['AppointmentDay'] = pd.to_datetime(med_Aptdata['AppointmentDay'])

# Create a variable called "waitday" by subtracting the date.
med_Aptdata['waitday'] = med_Aptdata["AppointmentDay"].sub(med_Aptdata["ScheduledDay"], axis=0)

# Convert the result "waitday" to number of days between appointment day and scheduled day.
med_Aptdata["waitday"] = (med_Aptdata["waitday"] / np.timedelta64(1, 'D')).abs().apply(np.floor)
```

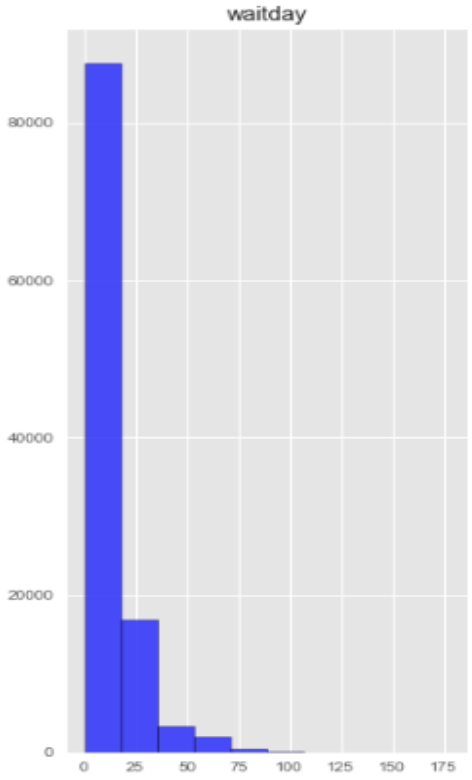
```
med_Aptdata.drop(['ScheduledDay','AppointmentDay'],axis=1,inplace=True)
```

```
med_Aptdata.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 110526 entries, 0 to 110526
Data columns (total 11 columns):
Gender          110526 non-null object
Age             110526 non-null int64
Neighbourhood   110526 non-null object
Scholarship     110526 non-null object
Hipertension    110526 non-null object
Diabetes        110526 non-null object
Alcoholism      110526 non-null object
Handcap         110526 non-null object
SMS_received    110526 non-null object
No-show         110526 non-null object
waitday         110526 non-null float64
dtypes: float64(1), int64(1), object(9)
memory usage: 10.1+ MB
```

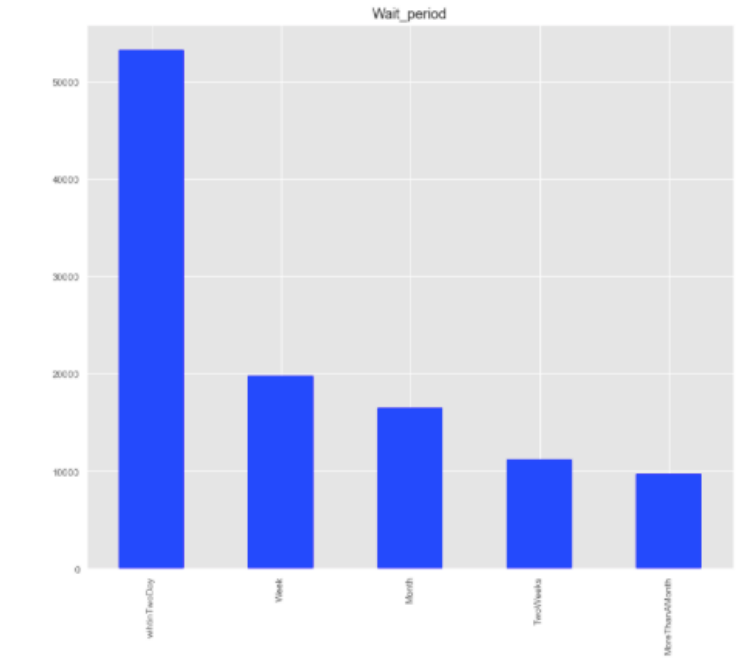
Innitial data cleaning done. Now, we have two numerical attributes(Age and waitday), and 8 categorical attributes.

Based on the common sense, I thought the number of days between schedule and appointment date would be better and less complicated predictor than schedule date and appointment date. Therefore, I created new feature called ‘waitday’ and drop both schedule and appointment date.




```
#additional preparation after exploring the data(binning the waitday)
# group: (sameday, 0; withinweek,7. over_a_month,30 )

bins = [-1,2,7,14,30,360]
labels = ["wihtinTwoDay", "Week", "TwoWeeks", "Month","MoreThanAMonth"]
wait_period = pd.cut(med_Aptdata['waitday'], bins, labels=labels)
med_Aptdata['Wait_period'] = wait_period
```

```
med_Aptdata.drop(['waitday'],axis=1,inplace=True)
```




Then, I created a histogram for “waitday” to observe the distribution. From the histogram above (the one on the left), we can see that the data “waitday” is right skewed. This means that we have some extreme high values (long tail on the right). To reduce the skewedness, I decided to bin the “waitday” to 5 groups: within two-days, a week, two weeks, a month, and more than a month. (Refer to the code and histogram on the right above). This helps to make the data less sparse and easier to understand.

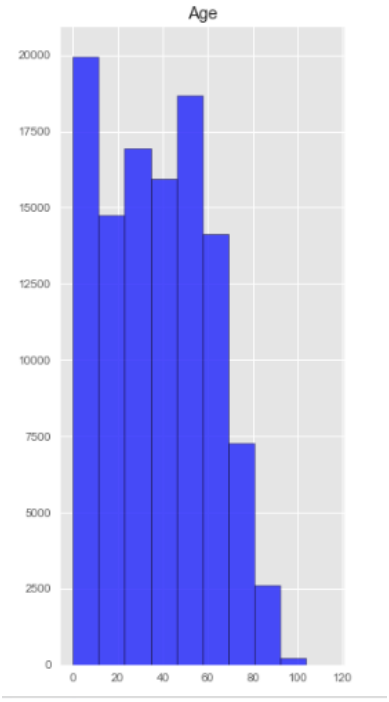
## Data Analysis

After the initial clean up on raw data, we ended up with 10 predictors: 9 categorical and 1 numerical ('age'). For data analysis (uni-variate and muti-variate), we will look at the each attribute deeper and conduct feature selection based on the results.

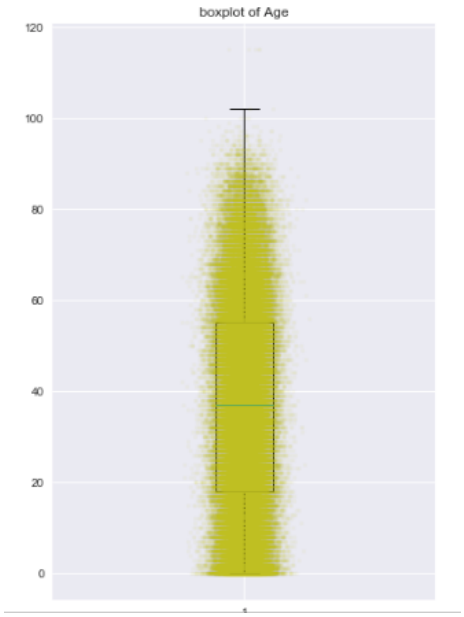
Also, I will divide the dataset with selected featured to training and testing set based on 80-20 ratios. Then, use an oversample technique (smote) to remove imbalance before searching the model.

Univariate

*Numerical attribute (Age):*



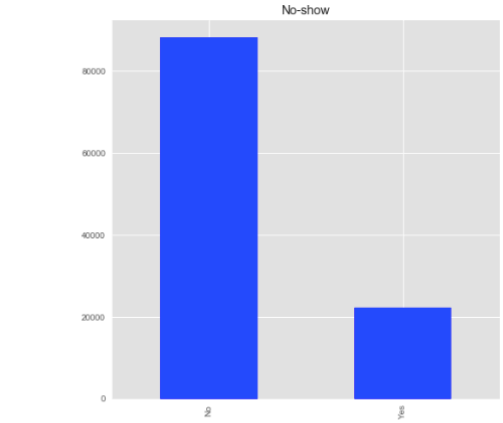

| | Age |
|---|---|
| count | 110526.000000 |
| mean | 37.089219 |
| std | 23.110026 |
| min | 0.000000 |
| 25% | 18.000000 |
| 50% | 37.000000 |
| 75% | 55.000000 |
| max | 115.000000 |

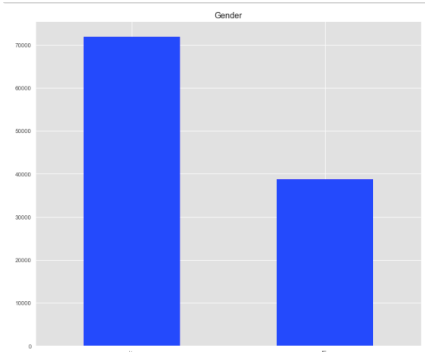


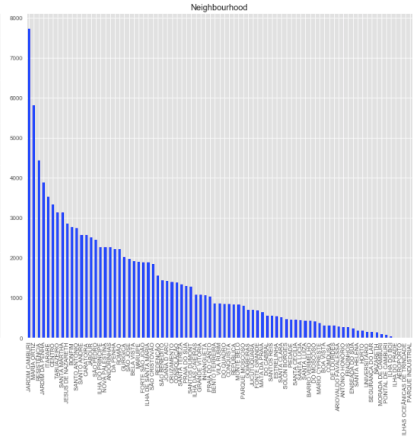

From the boxplots, histograms, and 5 number summaries above for age, we can see that median and mean in age is ~37. We have few observations above 100. Max (oldest), 115

years old and Min, Youngest is 0. Overall, we have roughly equally observation between 0-60.

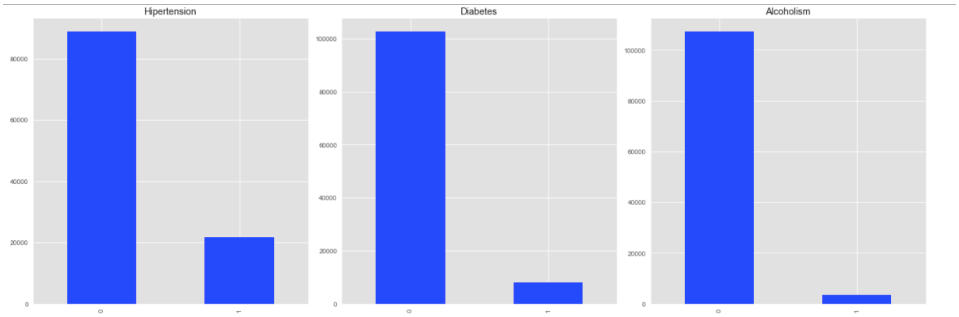
*Categorical ():*

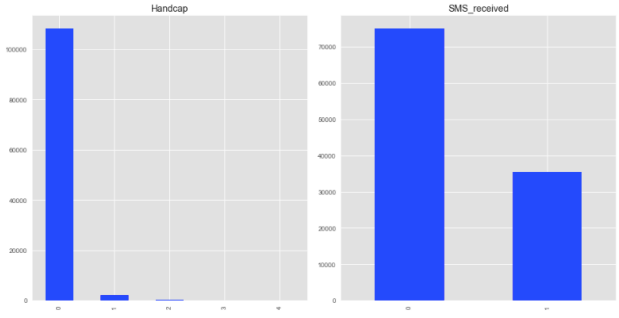


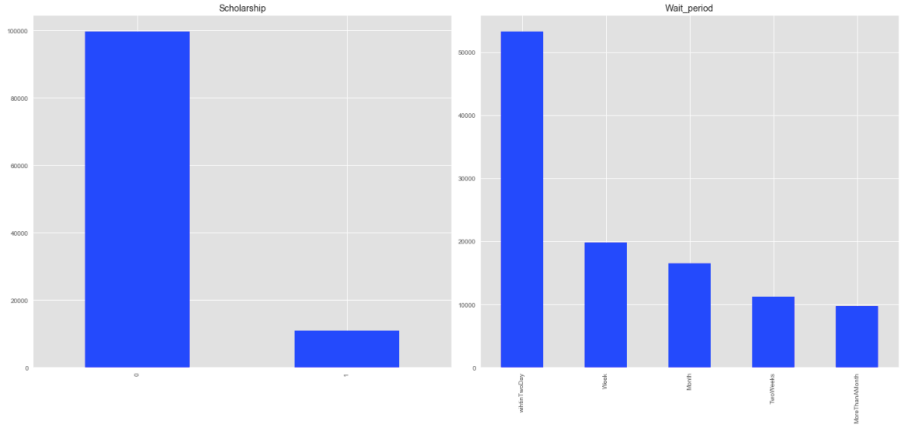




```
Out[34]: No-show
         No     79.806561
         Yes    20.193439
         dtype: float64
```




```
SMS_received
0    67.897146
1    32.102854
dtype: float64
```

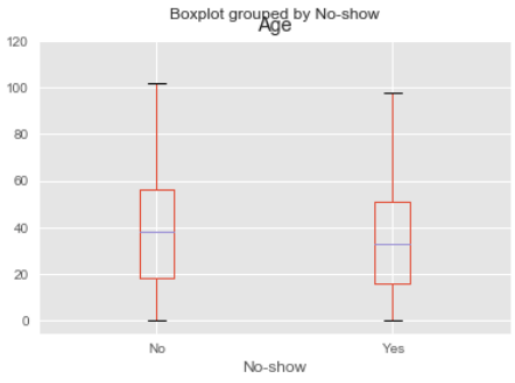


From the bar charts above for every categorical variable, there are few things being observed. First, target attribute, "noshow", imbalanced - 80% of observation does not miss appointment, and only 20% miss the appointment. Also, we have more case in female than male. Moreover, most people schedule their appointment within 2 days and most people do not receive scholarship (special offer). Last but not the least, majority of people doesn't have any labeled conditions (such as Diabetes, alcoholism, handicap…etc.) and ~67% of people signed up for sms reminder and ~32% are not.

Multivariate

For this section, I would like to investigate the relationship between target "noshow " and independent attributes so I created multiple plots by group including boxplot, bar chart, and cross table. Then, for further analysis, I also conduct ANOVA test for age by group (no-show and show) and chi-square test of independency for attribute 'no show' with every categorical predictors. The result of these test allow me to verify if there are significant statistic evidence of correlation between 'no-show' and each predictors. The graphs, crossable and test results are presented below:

*Boxplot and ANOVA test:*



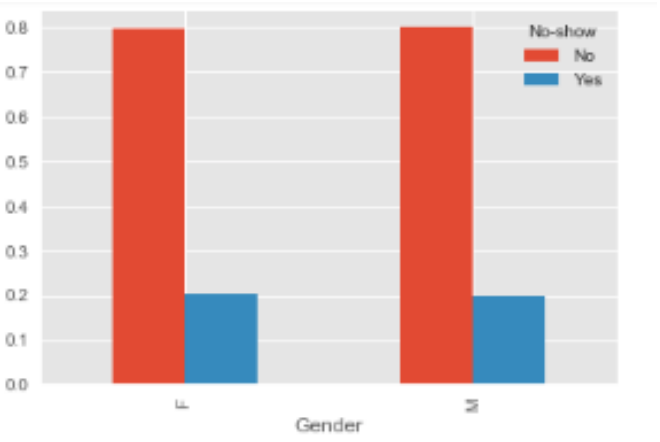

```
#Anova test
import scipy.stats as stats

stats.f_oneway(med_Aptdata.loc[med_Aptdata['No-show'] == 'Yes','Age'],med_Aptdata.loc[med_Aptdata['No-show'] == '
```

```
F_onewayResult(statistic=403.70201903382997, pvalue=1.2461264409419246e-89)
```

From the boxplot and Anova test above, we can conclude that mean of ages for two groups (no-show vs show) are different: overall, people who miss the appointment are younger.

*barchart by group ,crosstable, and chi square test:*

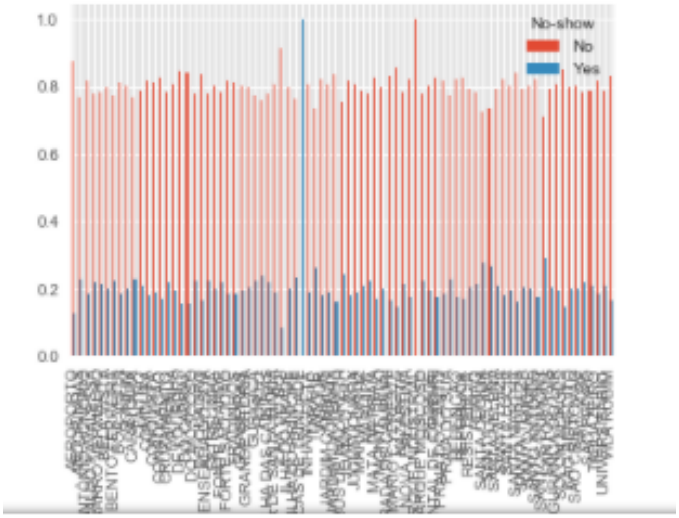
```
percentage cross table (* Comapring No- show = yes) :    Gender
No-show         No       Yes
Gender
F         0.796851  0.203149
M         0.800321  0.199679
```




```
percentage cross table (* Comapring No- show = yes) :    Scholarship
No-show            No       Yes
Scholarship
0            0.801926  0.198074
1            0.762637  0.237363
```

```
percentage cross table (* Comapring No- show = yes) :    Wait_period
No-show               No       Yes
Wait_period
wihtinTwoDay    0.903176  0.096824
Week            0.743789  0.256211
TwoWeeks        0.687746  0.312254
Month           0.674788  0.325212
MoreThanAMonth  0.669792  0.330208
```

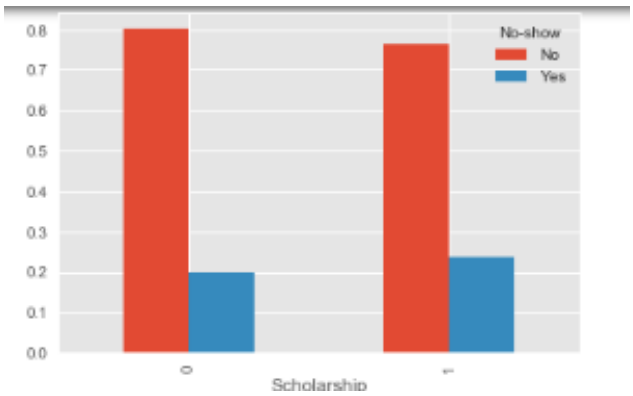
```
percentage cross table (* Comapring No- show = yes) :    Hipertension
No-show              No       Yes
Hipertension
0             0.790961  0.209039
1             0.826980  0.173020
```



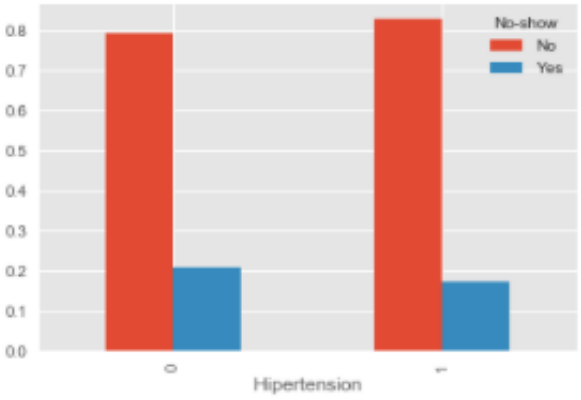

```
percentage cross table (* Comapring No- show = yes) :    Diabetes
No-show         No       Yes
Diabetes
0         0.796370  0.203630
1         0.819967  0.180033
```

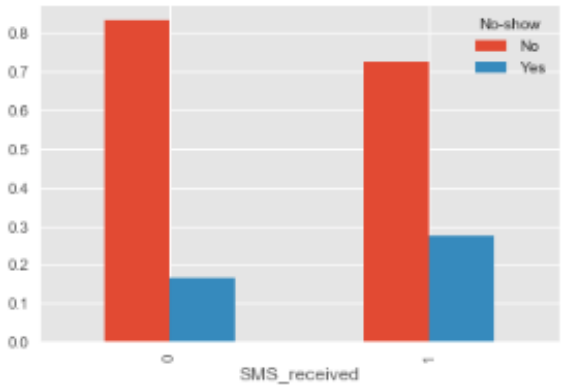
```
percentage cross table (* Comapring No- show = yes) :    Alcoholism
No-show           No       Yes
Alcoholism
0           0.798052  0.201948
1           0.798512  0.201488
```

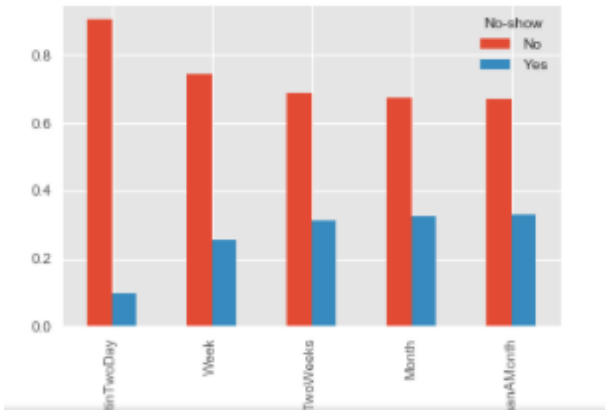
```
percentage cross table (* Comapring No- show = yes) :    Handcap
No-show        No       Yes
Handcap
0         0.797645  0.202355
1         0.820764  0.179236
2         0.797814  0.202186
3         0.769231  0.230769
4         0.666667  0.333333
```

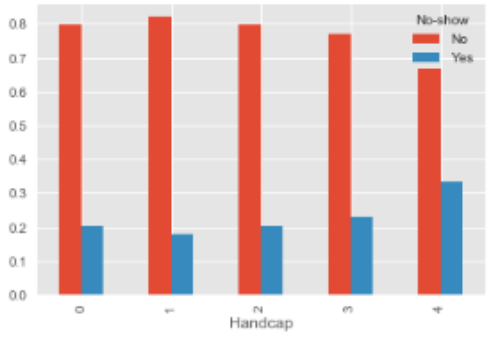


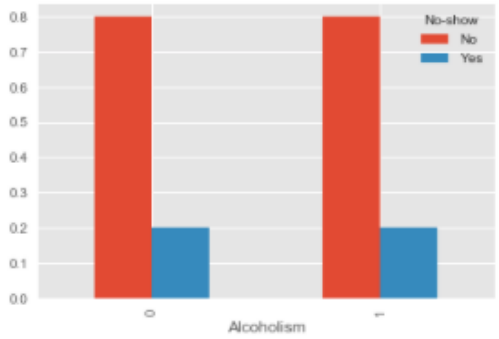

```
percentage cross table (* Comapring No- show = yes) :    SMS_received
No-show              No       Yes
SMS_received
0             0.832965  0.167035
1             0.724255  0.275745
```

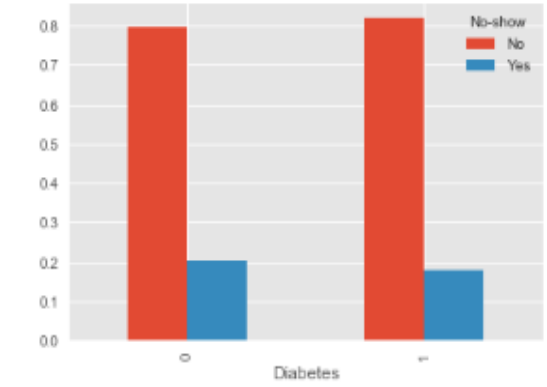


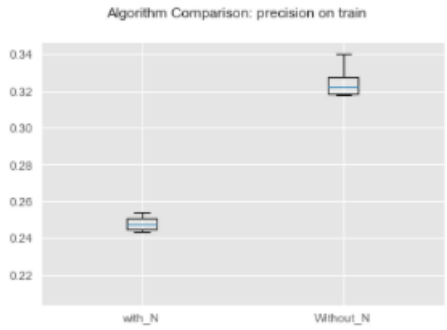




```
from scipy import stats
attributes =['Gender','Neighbourhood','Scholarship','Wait_period','Hipertension','Diabetes','Alcoholism','Handcap

for i in attributes:
    a= pd.crosstab(med_Aptdata[i],med_Aptdata["No-show"])
    g, p, dof, expctd = scipy.stats.chi2_contingency(a)
    print 'P-value for chi-square test of independency for ',i,': ', p
```

```
P-value for chi-square test of independency for  Gender :  0.173034161737
P-value for chi-square test of independency for  Neighbourhood :  1.52439814251e-60
P-value for chi-square test of independency for  Scholarship :  3.9268156991e-22
P-value for chi-square test of independency for  Wait_period :  0.0
P-value for chi-square test of independency for  Hipertension :  1.90112122415e-32
P-value for chi-square test of independency for  Diabetes :  4.83964682088e-07
P-value for chi-square test of independency for  Alcoholism :  0.965205424901
P-value for chi-square test of independency for  Handcap :  0.13401931355
P-value for chi-square test of independency for  SMS_received :  0.0
```

From the graph and chi square test above, we can see that gender, alcoholism, handicap does not affect show or no-show, and attributes such as sms_received, scholarship, wait_period, hypertension…and so on ($p << 0.05$) are correlated with target “no show”.

*Dropping 'Neighborhood':*




```
Ttest_relResult(statistic=-6.5879650852065614, pvalue=0.00010066076970928679)
```

Attribute neighborhood have too many level and based on the cross table created above, we can see that not every level have significant effect on target. Therefore, I test the prediction power by fitting naïve bayes model with and without neighborhood. Then validate the model use 10 fold cross validation. We can see from the boxplot, and paired t-test above that the model without a neighborhood returns better results. This makes me decide to drop the attribute neighborhood.

```
#dropping 'Neighbourhood'
med_Aptdata.drop(['Neighbourhood'], axis=1, inplace = True)
```

```
med_Aptdata.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 110526 entries, 0 to 110526
Data columns (total 10 columns):
Gender          110526 non-null object
Age             110526 non-null int64
Scholarship     110526 non-null object
Hipertension    110526 non-null object
Diabetes        110526 non-null object
Alcoholism      110526 non-null object
Handcap         110526 non-null object
SMS_received    110526 non-null object
No-show         110526 non-null object
waitday         110526 non-null float64
dtypes: float64(1), int64(1), object(8)
memory usage: 14.3+ MB
```

After I have done exploring the data by looking at multiple graph, table (cross table percentagewise), and t test, the selected features for later model searching are shown on the graph above.

Solution For Imbalanced Data And Data Splitting

After the feature are selected, I split the data into training and testing set using 80-20 ratios, then, I used a technique called Smote to remove imbalanced issue. Smote synthetically generated data points that are not too different from the minority class data points I actually have so unlike other oversampling technique, we actually create new observations. The advantage of it is that we can balance the dataset without losing information and having too many redundant observations. However, we need to keep in mind that, just like all the oversampling technique, over fitting will be an issue. The code and result are presented below:

```
# Separate the target attribute ("No-show")
# x : predictors, y : target
x2 = trial2.drop("No-show", axis=1, inplace = False)
y2 = trial2["No-show"]

#Convert the selected dataset into the Standard Spreadsheet format
# get_dummies" function to create dummy variables and converting to standard spreadsheet format
x_mat2 = pd.get_dummies(x2)
y2=pd.get_dummies(y2)
#no show up [1: noshow, 0: show]
y2=y2['Yes']

#80-20 split
from sklearn.cross_validation import train_test_split
x_train2, x_test2, y_train2, y_test2 = train_test_split(x_mat2, y2, test_size=0.2, random_state=33)
```

**Using Smote to deal with inbalance data**

```
[25]: # pip install -u imbalanced-learn
      #smote
      from imblearn.over_sampling import SMOTE
```

```
[26]: smote=SMOTE(kind = "regular",random_state=0, ratio = 1.0)
      smote_predictors,smote_target=smote.fit_sample(x_train2,y_train2)

      print y_train2.value_counts()

      print ''
      print (np.bincount(smote_target))
```

```
/Users/jasonwu/anaconda/lib/python2.7/site-packages/sklearn/utils/deprecation.py:75: DeprecationWarning: Function _r
atio_float is deprecated; Use a float for 'ratio' is deprecated from version 0.2. The support will be removed in 0.4
. Use a dict, str, or a callable instead.
  warnings.warn(msg, category=DeprecationWarning)
0    70629
1    17791
Name: Yes, dtype: int64

[70629 70629]
```

```
Now, for our training set, we have equal number of observation for both noshow and show
```

```
n [*]: #141258 obeservation on training and 23 dimensions
```

```
##balacne trainign set
#  Training set: [smote_predictors,smote_target]


## Unseen testing set : [x_test2, y_test2]

### target y [1: noshow, 0: show]
```

## Experimental Results

After 80-20 ratio data splitting and solving the imbalanced issues using an oversampling technique, smote, I ended up with equally number (70629) of observations for both show and no show in training set. The training set (smote_predictors, smote_target) will then be used to build classification models with different algorithms (naïve bayes, boosting, begging , stacking, and support vector machine). Also, 10 fold cross validation and grid search were performed to help to tune and find optimal parameters. I also conduct paired t_test on result of 10 fold cross validation to compare model performance based on the chosen performance metric. The unseen testing set then were used to fit the model to generate the performance metrics and ROC curve for evaluation purpose. The reports, including accuracies with 95 confidence interval returned by repeated 10 fold cross validation on training set, results of paired t_test for comparing performance between models, and prediction evaluation (performance metric and ROC curve) on the testing set are presented below.

Naïve bayes

- Naive Bayes

```
#different Naive Bayes
from sklearn.naive_bayes import GaussianNB
from sklearn.naive_bayes import BernoulliNB
from sklearn.naive_bayes import MultinomialNB


from sklearn import cross_validation
# Spot Check Algorithms
models = []
models.append(('MNB', MultinomialNB()))
models.append(('BNB', BernoulliNB()))
models.append(('GNB', GaussianNB()))

# evaluate each model in turn
results = []
names = []
for name, model in models:
    cv_results = cross_validation.cross_val_score(model, smote_predictors, smote_target, cv=10)
    results.append(cv_results)
    names.append(name)
    msg = "%s: %.3f (%.3f)" % (name, cv_results.mean(), cv_results.std())
    print(msg)
```

```
MNB: 0.626 (0.002)
BNB: 0.661 (0.008)
GNB: 0.536 (0.019)
```

```
BernoulliNB has better performacne than others, lets tune the parameters
```

```
BNB= BernoulliNB(class_prior=None, fit_prior=True)


# Use a grid over parameters of interest
param_grid = { "alpha" : [ 0 ,0.1, 1, 10, 100],'binarize':[0 ,0.1, 1, 10, 100]}

CV_BNB = GridSearchCV(estimator=BNB, param_grid=param_grid, cv= 10)
CV_BNB.fit(smote_predictors, smote_target)
print CV_linear.best_params_
```

```
{'binarize': 0, 'alpha': 0}
```

```
# Optimized BNB

BNB= BernoulliNB(class_prior=None, fit_prior=True)


# fit the model with training set  - CV(10)  and predict avg accuracy_
BNB_scores = model_selection.cross_val_score(BNB, smote_predictors, smote_target, cv=10, scoring='accuracy')
print("bernoulli NB Train accuracy: %0.2f (+/- %0.2f)" % (BNB_scores.mean()*100, BNB_scores.std()*100))

# confidence intervals
import numpy
alpha = 0.95
p = ((1.0-alpha)/2.0) * 100
lower = max(0.0, numpy.percentile(BNB_scores, p))
p = (alpha+((1.0-alpha)/2.0)) * 100
upper = min(1.0, numpy.percentile(BNB_scores, p))
print('%.1f confidence interval %.1f%% and %.1f%%' % (alpha*100, lower*100, upper*100))
```

```
bernoulli NB Train accuracy: 66.15 (+/- 0.76)
95.0 confidence interval 65.0% and 67.0%
```

```
#nb model predcition on test set
#alpha=1.0, binarize=2.0, class_prior=None, fit_prior=True)
# Make predictions on test dataset
BNB= BernoulliNB(alpha=0, binarize=0, class_prior=None, fit_prior=True)
BNB.fit(smote_predictors, smote_target)
BNBpd = BNB.predict(x_test2)
print('Prediction report: Naive Bayes')
print '\n'
print'accuracy: %0.3f' % accuracy_score(y_test2, BNBpd)
print '\n'
print 'confusion matrix: '
print confusion_matrix(y_test2, BNBpd)
print '\n'
print 'classification report : '
print(classification_report(y_test2, BNBpd))
```

```
Prediction report: Naive Bayes


accuracy: 0.582


confusion matrix:
[[9393 8185]
 [1060 3468]]


classification report :
             precision    recall  f1-score   support

          0       0.90      0.53      0.67     17578
          1       0.30      0.77      0.43      4528

avg / total       0.78      0.58      0.62     22106
```

Best-tuned Naive Bayes:

**BNB= BernoulliNB (alpha=0, binarize=0, class_prior=None, fit_prior=True)**

Prediction on unseen set:

Recall for no-show- 77%,

Precision for no-show- 30%

Average recall- 58%

<u>Begging (random forest)</u>

```
# Optimized RF classifier to maxize precisoon
rfcb = RandomForestClassifier(n_estimators=36, max_depth=10, max_features='sqrt', min_samples_leaf = 1)


# fit the model with training set  - CV(10)  and predict avg accuracy
rfcb_scores = model_selection.cross_val_score(rfc, smote_predictors, smote_target, cv=10)
print("Accuracy of Random forest on balanced Trainning set  : %0.2f (+/- %0.2f)" % (rfcb_scores.mean()*100, rfcb_s

# confidence intervals
import numpy
alpha = 0.95
p = ((1.0-alpha)/2.0) * 100
lower = max(0.0, numpy.percentile(rfcb_scores, p))
p = (alpha+((1.0-alpha)/2.0)) * 100
upper = min(1.0, numpy.percentile(rfcb_scores, p))
print('%.1f confidence interval %.1f%% and %.1f%%' % (alpha*100, lower*100, upper*100))
```

```
Accuracy of Random forest on balanced Trainning set  : 78.05 (+/- 8.08)
95.0 confidence interval 63.0% and 83.6%
```

```
# Make predictions on test dataset (RF) with optimal model trained with balanced data

rfcb = RandomForestClassifier(n_estimators=36, max_depth=10, max_features='sqrt', min_samples_leaf = 1)
rfcb.fit(smote_predictors, smote_target)
RFpdb = rfcb.predict(x_test2)
print('Prediction report: Random Forest')
print '\n'
print'accuracy: %0.3f' % accuracy_score(y_test2, RFpdb)
print '\n'
print 'confusion matrix: '
print confusion_matrix(y_test2, RFpdb)
print '\n'
print 'classification report : '
print(classification_report(y_test2, RFpdb))
```

```
Prediction report: Random Forest


accuracy: 0.648


confusion matrix:
[[11310  6268]
 [ 1517  3011]]


classification report :
             precision    recall  f1-score   support

          0       0.88      0.64      0.74     17578
          1       0.32      0.66      0.44      4528

avg / total       0.77      0.65      0.68     22106
```

best tuned random forest model (bagging technique):

**rfcb =RandomForestClassifier(n_estimators=36, max_depth=10, max_features='sqrt', min_samples_leaf = 1)**

Prediction on unseen set:

Recall for no show :66%, precision for no show : 32%

Best-tuned Random forest (begging technique):

**rfcb =RandomForestClassifier(n_estimators=36, max_depth=10, max_features='sqrt', min_samples_leaf = 1)**

Prediction on unseen set:

Recall for no show - 66%

Precision for no show - 32%

Average recall - 65%

<u>Boosting (gradient boosting tree)</u>

```
# Optimized GB classifier
gbc = GradientBoostingClassifier(n_estimators=36, max_depth=10, max_features='sqrt', min_samples_leaf = 1)


# fit the model with training set  - CV(10)  and predict avg accuracy_
gbc_scores = model_selection.cross_val_score(gbc, smote_predictors, smote_target, cv=10)
print("Accuracy of Gradiant boosting dT on balanced Trainning set: %0.2f (+/- %0.2f)" % (gbc_scores.mean()*100, gbc_sc

# confidence intervals
import numpy
alpha = 0.95
p = ((1.0-alpha)/2.0) * 100
lower = max(0.0, numpy.percentile(gbc_scores, p))
p = (alpha+((1.0-alpha)/2.0)) * 100
upper = min(1.0, numpy.percentile(gbc_scores, p))
print('%.1f confidence interval %.1f%% and %.1f%%' % (alpha*100, lower*100, upper*100))
```

```
Accuracy of Gradiant boosting dT on balanced Trainning set: 68.94 (+/- 1.87)
95.0 confidence interval 65.5% and 70.7%
```

```
# Make predictions on test dataset (GB)
gbc.fit(smote_predictors, smote_target)
gbcpd = gbc.predict(x_test2)
print('Prediction report: gradient boosting')
print '\n'
print'accuracy: %0.3f' % accuracy_score(y_test2, gbcpd)
print '\n'
print 'confusion matrix: '
print confusion_matrix(y_test2, gbcpd)
print '\n'
print 'classification report : '
print(classification_report(y_test2, gbcpd))
```

```
Prediction report: gradient boosting


accuracy: 0.659


confusion matrix:
[[11697  5881]
 [ 1662  2866]]


classification report :
             precision    recall  f1-score   support

          0       0.88      0.67      0.76     17578
          1       0.33      0.63      0.43      4528

avg / total       0.76      0.66      0.69     22106
```

Best tuned gradient boosting tree (boosting technique):

**__gbc =GradientBoostingClassifier(n_estimators=36, max_depth=10, max_features='sqrt', min_samples_leaf = 1)__**

Prediction on unseen set:

Recall for no show - 63%

Precision for no show - 33%

Average recall - 66%

Support vector machine (linear and with kernel trick)

It took way too long to run SVM, even if it is linear for this dataset (141258 observation on training and 23 dimensions after smote). Unable to get any result at first, I reduced the sample size to n = 1412 and ran it again. The result are shown below:

*Linear:*

In [38]:
```
# Linear SVM classifier
#####linear
from sklearn.svm import SVC
linear = SVC(kernel='linear',C=1)


# fit the model with training set  - CV(10)  and predict avg accuracy_
linear_scores = model_selection.cross_val_score(linear, smote_predictors, smote_target, cv=10, scoring='accuracy')
print("linear SVM accuracy: %0.2f (+/- %0.2f)" % (linear_scores.mean()*100, linear_scores.std()*100))

# confidence intervals
import numpy
alpha = 0.95
p = ((1.0-alpha)/2.0) * 100
lower = max(0.0, numpy.percentile(linear_scores, p))
p = (alpha+((1.0-alpha)/2.0)) * 100
upper = min(1.0, numpy.percentile(linear_scores, p))
print('%.1f confidence interval %.1f%% and %.1f%%' % (alpha*100, lower*100, upper*100))
```

```
linear SVM accuracy: 66.67 (+/- 4.51)
95.0 confidence interval 60.8% and 75.4%
```

In [39]:
```
#prediction linear svm

#LINEAR SVM
# Make predictions on test dataset
linear = SVC(kernel='linear',C=1)
linear.fit(smote_predictors, smote_target)
linearpd = linear.predict(x_test2)
print('Prediction report: linear SVM')
print '\n'
print'accuracy: %0.3f' % accuracy_score(y_test2, linearpd)
print '\n'
print 'confusion matrix: '
print confusion_matrix(y_test2, linearpd)
print '\n'
print 'classification report : '
print(classification_report(y_test2, linearpd))
```

```
Prediction report: linear SVM


accuracy: 0.591


confusion matrix:
[[9578 8000]
 [1045 3483]]


classification report :
             precision    recall  f1-score   support

          0       0.90      0.54      0.68     17578
          1       0.30      0.77      0.44      4528

avg / total       0.78      0.59      0.63     22106
```

SVM With kernal tricks and tuning paremater

*With kernel:*

Even with the reduced size, I was unable to get any results due to endless runtime when building the model.

In [51]:
```
# kernel support vector machine



# penalize(weighted)
kernelsvm=SVC(class_weight='balanced',probability=True)

|
# Use a grid over parameters of interest
param_grid = { "kernel" : ['rbf','poly','sigmoid'],
           "gamma" : [1e-1, 1e-2, 1e-3, 1e-4],
           "C" : [0.01 ,0.1, 1, 10, 100, 1000]}

CV_kernelsvm = GridSearchCV(estimator=kernelsvm, param_grid=param_grid, cv= 10)
CV_kernelsvm.fit(smote_predictors, smote_target)
print CV_kernelsvm.best_params_
```

In [ ]:
```
# Optimized kernal SVM classifier

poly_SVM = SVC(kernel = 'poly', C = 0.1, gamma = 0.1,class_weight='balanced',probability=True)


# fit the model with training set  - CV(10)  and predict avg accuracy_
poly_SVM_scores = model_selection.cross_val_score(poly_SVM, smote_predictors, smote_target, cv=10, scoring='accuracy')
print("poly_SVM Train accuracy: %0.2f (+/- %0.2f)" % (poly_SVM_scores.mean()*100, poly_SVM_scores.std()*100))

# confidence intervals
import numpy
alpha = 0.95
p = ((1.0-alpha)/2.0) * 100
lower = max(0.0, numpy.percentile(poly_SVM_scores, p))
p = (alpha+((1.0-alpha)/2.0)) * 100
upper = min(1.0, numpy.percentile(poly_SVM_scores, p))
print('%.1f confidence interval %.1f%% and %.1f%%' % (alpha*100, lower*100, upper*100))
```

In [ ]:
```
#prediction, kernal svm

# Make predictions on test dataset
poly_SVM = SVC(kernel = 'poly', C = 100, gamma = 0.1)
poly_SVM.fit(smote_predictors, smote_target)
poly_SVMpd = poly_SVM.predict(x_test2)
print('Prediction report: polynominal kernel SVM')
print '\n'
print'accuracy: %0.3f' % accuracy_score(y_test2, poly_SVMpd)
print '\n'
print 'confusion matrix: '
print confusion_matrix(y_test2, poly_SVMpd)
print '\n'
print 'classification report : '
print(classification_report(y_test2, poly_SVMpd))
```

Best tuned kernel SVM (linear):

**Linear = SVC(kernel='linear',C=1)**

Prediction on unseen set:

Recall for no show : 77%

Overall recall: 59%

Stacking (combination of all the algorithms- SVM, naïve Bayes, boosting, and begging)

In [37]:
```
#import majority_vote_classifier as mvc


from sklearn.ensemble import VotingClassifier


estimators = []
model1 = BernoulliNB(alpha=0, binarize=0, class_prior=None, fit_prior=True)
estimators.append(('bernouli NB', model1))
model2 = RandomForestClassifier(n_estimators=36, max_depth=10, max_features='sqrt', min_samples_leaf = 10)
estimators.append(('Random forest', model2))
model3 = GradientBoostingClassifier(n_estimators=36, max_depth=10, max_features='sqrt', min_samples_leaf = 1)
estimators.append(('gradient boosting', model3))
model4 = SVC(kernel='linear',C=1)
estimators.append(('linear svm', model4))


# create the ensemble model
stacking = VotingClassifier(estimators)
results = model_selection.cross_val_score(stacking, smote_predictors, smote_target, cv=10)
print("stacking model Train accuracy: %0.2f (+/- %0.2f)" % (results.mean()*100, results.std()*100))

# confidence intervals
import numpy
alpha = 0.95
p = ((1.0-alpha)/2.0) * 100
lower = max(0.0, numpy.percentile(results , p))
p = (alpha+((1.0-alpha)/2.0)) * 100
upper = min(1.0, numpy.percentile(results , p))
print('%.1f confidence interval %.1f%% and %.1f%%' % (alpha*100, lower*100, upper*100))
```

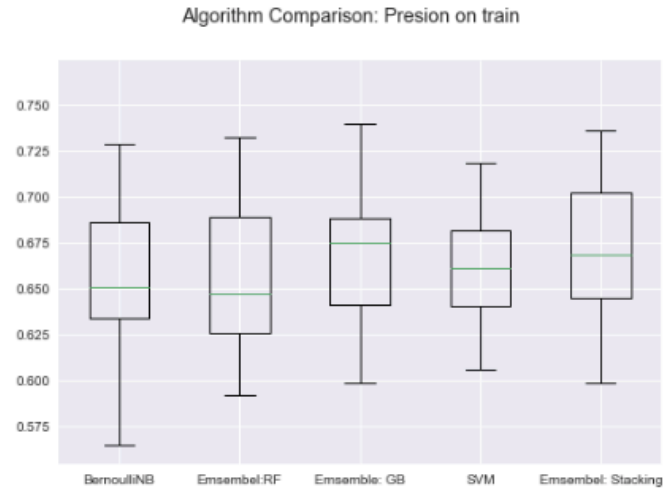
```
stacking model Train accuracy: 66.96 (+/- 4.05)
95.0 confidence interval 60.5% and 73.6%
```

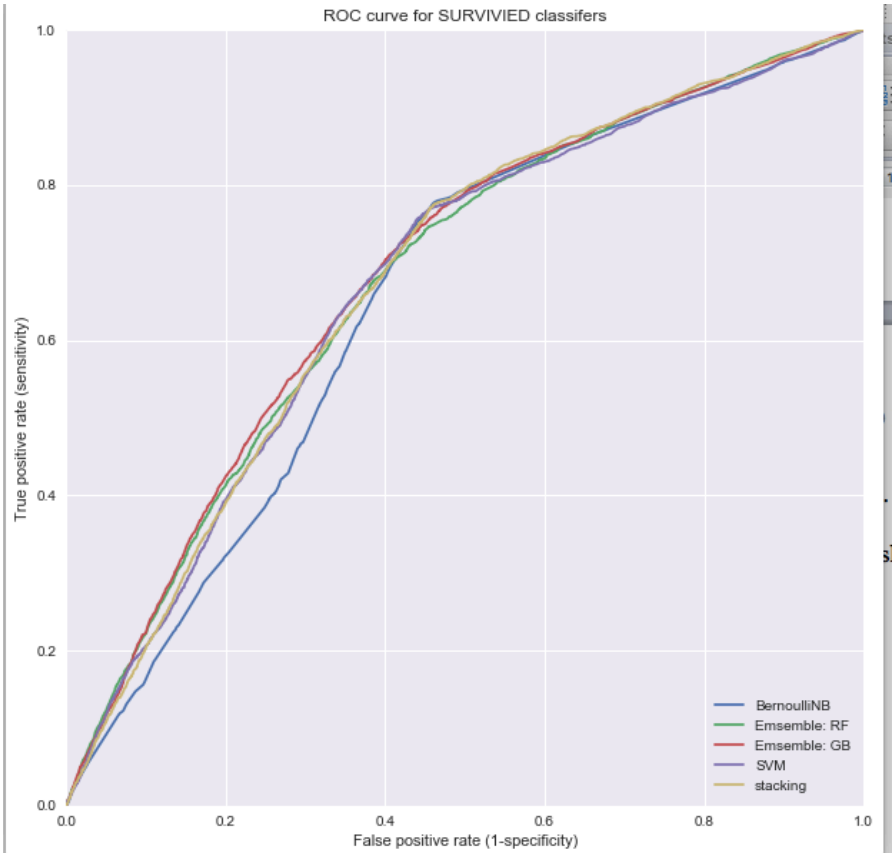
```
#prediction


stacking.fit(smote_predictors, smote_target)
stpd = stacking.predict(x_test2)
print('Prediction report: Stacking')
print '\n'
print'accuracy: %0.3f' % accuracy_score(y_test2, stpd)
print '\n'
print 'confusion matrix: '
print confusion_matrix(y_test2, stpd)
print '\n'
print 'classification report : '
print(classification_report(y_test2, stpd))
```

```
Prediction report: Stacking


accuracy: 0.619


confusion matrix:
[[10472  7106]
 [ 1311  3217]]


classification report :
             precision    recall  f1-score   support

          0       0.89      0.60      0.71     17578
          1       0.31      0.71      0.43      4528

avg / total       0.77      0.62      0.66     22106
```

Model Comparison And Evaluation (Pair t_test, Roc, and Performance Metric)

```
10 fold Cross validaiton report for each model: average Acuuracy and Standardiviation
```

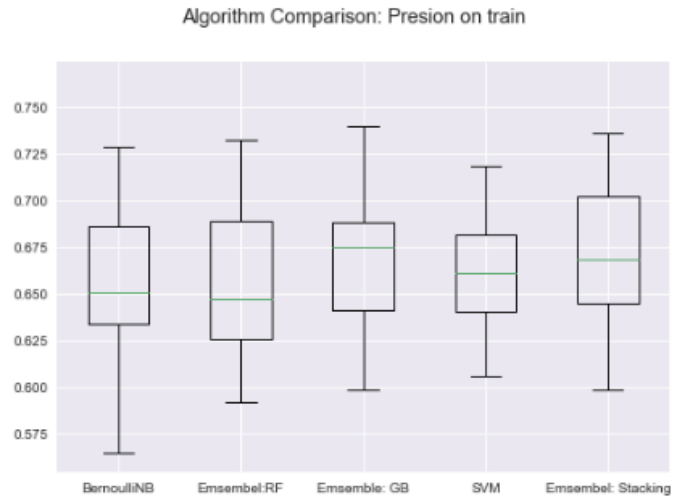


## Experimental Analysis

The goal of this project is to explore different classification technique and find a model that can accurately predict no-show case based on the features in the medical dataset. After the dataset has been prepared (cleaned missing value, created new feature, and took care of imbalanced issue using smote), and explored (distribution, center, spread, and correlation with graph, percentage wise cross table, and statistic test), the selected features that are suitable for predicting target (‘no show’) were ‘gender’, ’age’, ’wait period’, ’sms received’, ’diabetes’, ’ hypertension’, ‘alcoholism’, ‘Handicap’, and ‘scholarship’. Then, Multiple machine learning algorithms including individual (naïve bayes), SVM, and ensemble (random forest, gradient boosting tree, stacking) learning technique were implemented into training set to build classifiers. Also, grid search was used to find the best parameter (such as split, tree depth) to optimize selected performance metric. Based on the pair t test and result (average accuracy with 95 confidence level) of 10 fold cross validation, stacking, gradient boosting and random forest classifiers have slightly better result than others (refer to the experimental result above); However, Time to build the model are significantly slower which beg the question: is a more complicated model really better?

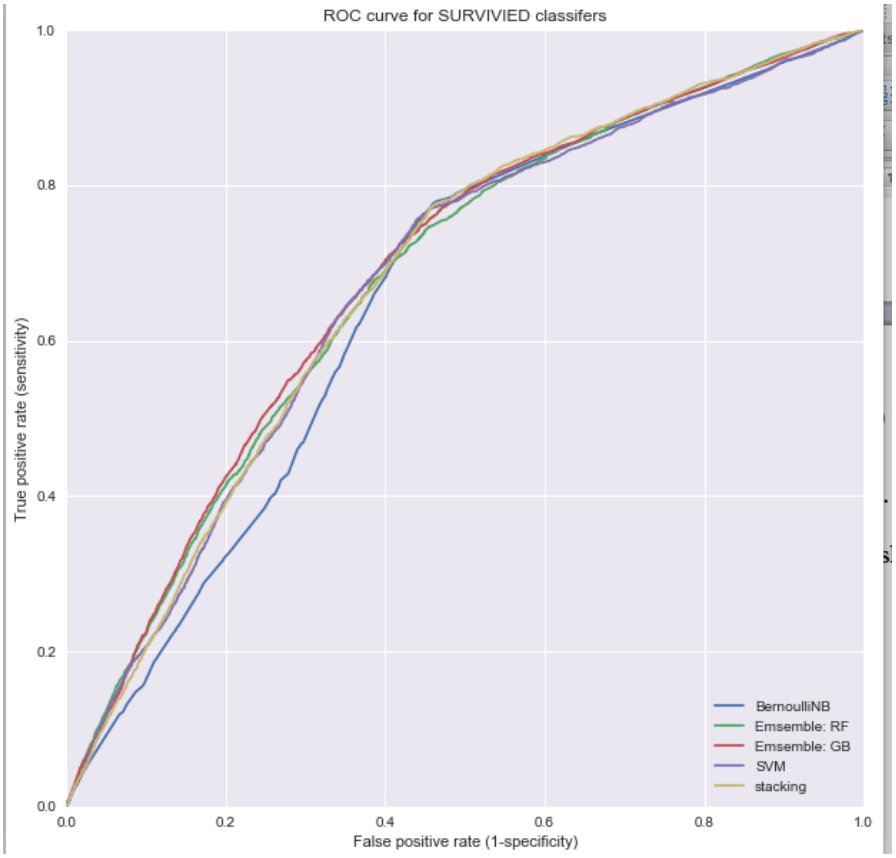
```
10 fold Cross validaiton report for each model: average Acuuracy and Standardiviation
```




In our case, after the imbalanced issue is removed. The accuracies on the test set for each algorithm with tuned parameters are: 58% for naïve Bayes, 66% for random forest, 67% for gradient boosting tree, 59% for linear SVM, and 61% for stacking. We can see that all ensembles outperform the rest (individual and svm) for this specific unseen test set in term of accuracy. However, our goal is to identify the no show case and we don't care about people who show up that much. Therefore, the recall will be more appropriate for the inference. The reason is that recall (TP/(TP+FN)) measures how much no-show cases we have found based on the model which fit our goal "identify the no show case". Prediction on test set, recalls of no-show for each model are: 69% (naïve bayes), 65% (random forest), 64% (gradient boosting), and 77% (liner support vector machine). We can see that linear support vector machine have the highest recall: 77%. Support vector machine with kernel trick may return even better result but I was not able to get any result because of endless runtime to build the model. As for comparing

stacking with linear support vector machine, support vector machine has better recall on no-show than stacking (77% vs 71%).

Overall, support vector machine give us the best recall for no show. As for general accuracy, gradient boosting and random forest out preform others. To evaluate the result deeper, we take a look at the roc curve. Based on roc curve, Area under curve for all models are close to each other. Therefore, the final chosen model would depend on the purpose of the study. In our case, since our goal is to find the no show case, I would chose the model with best recalls value for no show which is support vector machine. However, there are downsides of choosing support vector machine. Support vector machine gave me the better recall value for no show but it also took much longer to build the model and make prediction.



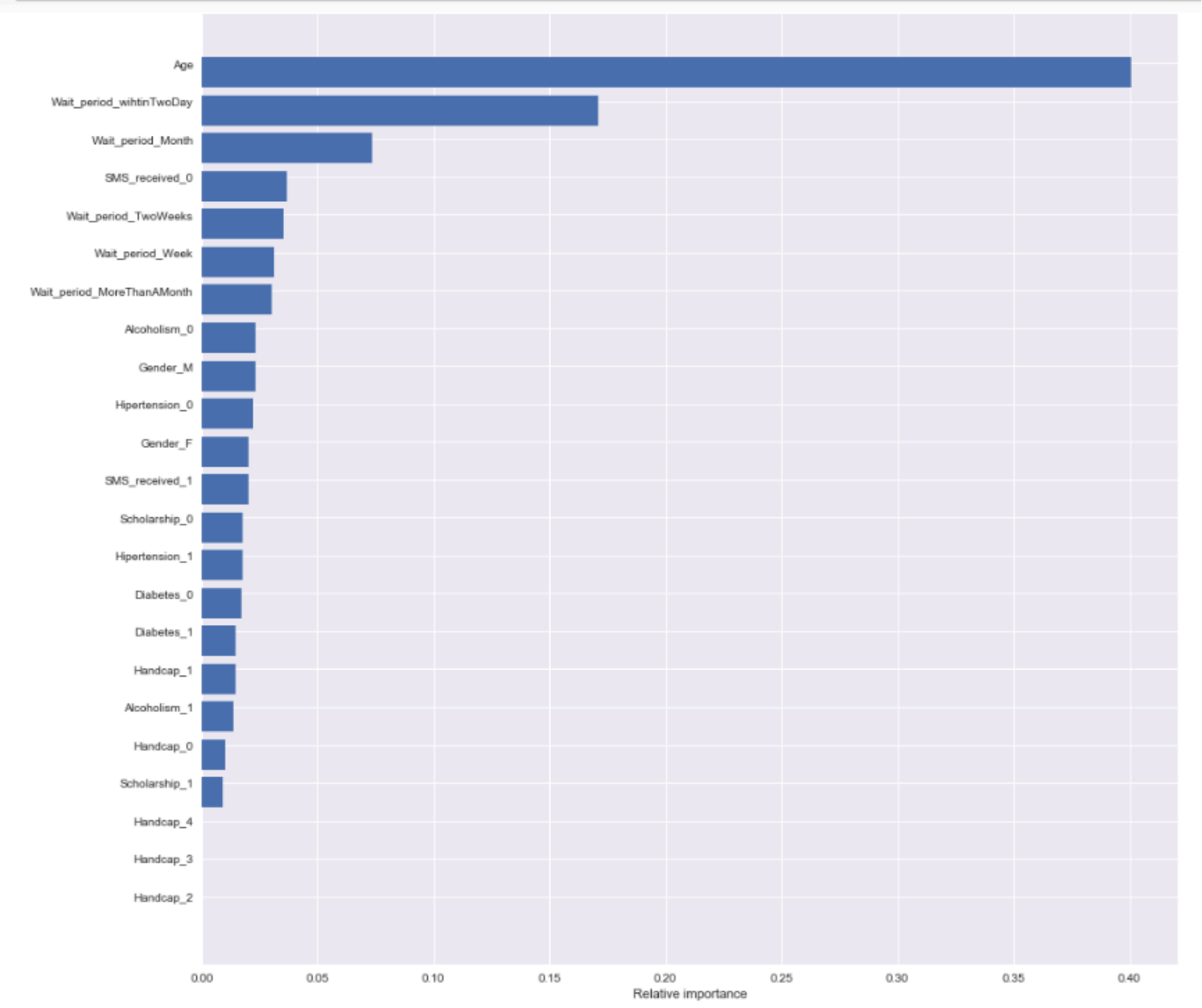
## Conclusion

The purpose of the project is to analyze the medical no show dataset and find out why patients miss their doctor appointments and find the best algorithms with tuned parameters to be used identify no-show cases based on the given information.

The features (gender', 'age', 'wait period', 'sms received', 'diabetes', ' hypertension', 'alcoholism', 'Handicap', and 'scholarship') were selected for investigation after data preparation (checked missing value, created new feature, and took care of imbalanced issue using smote), and exploration (distribution, center, spread, and correlation with graph, percentage wise cross table, and statistic test).

Then, after searching over several machine-learning algorithms (naïve bayes, SVM, random forest, gradient boosting tree, and stacking) with tuned parameters, linear support vector machine turned out to have best performance on unseen dataset in term of recall for no-show case. The recall value for it is 77%. The second best is stacking (71%). As for general accuracy, gradient boosting (65%) and random forest (64%) out perform others. For our case, recall would be better metric for inference since our goal is to find the model that can help to identify the no show case. Roc curves were also used to evaluate the classification performance. The conclusion was that even if there exist a problem of long runtime to build the model and make prediction, support vector machine still is chosen as our solution for the study due to the significantly better recall value for

no-show case. Also, the top 3 most important feature reported for the classifying no-show and show based on gradient boosting tree are: age, wait period: within Two day, and wait period, with in a month.



## Appendix

Source:

https://www.kaggle.com/joniarroba/noshowappointments/data

Code:

```
#import libarary
import sys
import scipy
import numpy as np
import matplotlib
import pylab
import seaborn as sns
import pandas as pd
import sklearn
from pandas.tools.plotting import scatter_matrix
import matplotlib.pyplot as plt

# Machine Learning
from sklearn.svm import SVC
from sklearn.naive_bayes import GaussianNB
from sklearn.tree import DecisionTreeClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.ensemble import GradientBoostingClassifier
```

```
#model report and sklearn package
from sklearn import preprocessing
from sklearn.metrics import accuracy_score
from sklearn.metrics import confusion_matrix
from sklearn.metrics import classification_report
from sklearn import model_selection
from sklearn import cross_validation
from sklearn.metrics import accuracy_score
from sklearn.grid_search import GridSearchCV
from sklearn.preprocessing import MinMaxScaler
from sklearn.cross_validation import cross_val_score
from sklearn.cross_validation import train_test_split
from scipy import stats


# In[4]:


#load data


med_Aptdata = pd.read_csv("KaggleV2-May-2016.csv",na_values='?')


# In[7]:


#look at data struture
```

```
med_Aptdata.info()

print  'data struture(row, colunm) : ', med_Aptdata.shape



# In[47]:


# first 10 row

med_Aptdata.head(10)



# In[11]:


med_Aptdata.describe()



#

# ## data discription and goal

#


# 11057 observation and 14 variables.

#

# target(categorical) : No_show : Yes = no show, no = show up,

# 13 factors

#
```

```
# useful precitors(age,neighborhood,
scholarship,hipertension,diabetes,alcoholism,handcap,sms_received,gender, appointdate,
scheduledate)
#
# 8 categorical predictor and 2 numerical predictors
#


# ## data cleaning


# After the first look at dataset. here are the few thing i m going to do to prepare the data
#
# 1. check for missing variable and take care of missing value
# 2. relable the datatype
# 3. drop useless variable


# In[5]:


#check missing values
print "attribute with missing values: \n"
print [col for col in med_Aptdata.columns if med_Aptdata[col].isnull().any()]



# No missign value


# In[6]:
```

```
#drop useless columns
med_Aptdata.drop(['PatientId','AppointmentID'], axis=1, inplace = True)



# In[7]:


#relable categorical datatype to str
med_Aptdata.Scholarship=med_Aptdata.Scholarship.astype(str)
med_Aptdata.Hipertension=med_Aptdata.Hipertension.astype(str)
med_Aptdata.Diabetes=med_Aptdata.Diabetes.astype(str)
med_Aptdata.Alcoholism=med_Aptdata.Alcoholism.astype(str)
med_Aptdata.Handcap=med_Aptdata.Handcap.astype(str)
med_Aptdata.SMS_received=med_Aptdata.SMS_received.astype(str)



# In[39]:


med_Aptdata.info()



# In[16]:


med_Aptdata.describe()
```

```
# age min = -1 make no sense.


# In[17]:


med_Aptdata[med_Aptdata.Age < 0].count()



# In[8]:


#only ONE record has age < 0 so remove it wont hurt

med_Aptdata= med_Aptdata[med_Aptdata['Age'] >= 0]



# In[20]:


med_Aptdata[med_Aptdata.Age < 0].count()



# In[14]:


med_Aptdata.info()


# In[22]:
```

```
med_Aptdata.head()



# In[9]:


#new colum, waitday  (day between ScheduledDay  and AppointmentDay)
import numpy as np

# Converts the two variables to datetime variables
med_Aptdata['ScheduledDay'] = pd.to_datetime(med_Aptdata['ScheduledDay'])
med_Aptdata['AppointmentDay'] = pd.to_datetime(med_Aptdata['AppointmentDay'])

# Create a variable called "waitday" by subtracting the date.
med_Aptdata['waitday'] =
med_Aptdata["AppointmentDay"].sub(med_Aptdata["ScheduledDay"], axis=0)

# Convert the result "waitday" to number of days between appointment day and scheduled day.
med_Aptdata["waitday"] = (med_Aptdata["waitday"] / np.timedelta64(1,
'D')).abs().apply(np.floor)



# In[14]:


med_Aptdata.head()
```

```
# In[10]:

med_Aptdata.drop(['ScheduledDay','AppointmentDay'],axis=1,inplace=True)



# In[11]:

med_Aptdata.info()



# Innitial data cleaning done. Now, we have two numerical attributes(Age and waitday), and 8 categorical attributes.

# # Exploratory

# ## Univariate

# numerical

# In[27]:

med_Aptdata.describe()



# In[28]:
```

```
#boxplot


fig = plt.figure(figsize=(20, 20))
ax1 = fig.add_subplot(2,3,1)
bp= ax1.boxplot(med_Aptdata['Age'])
y = med_Aptdata.Age
x = np.random.normal(1, 0.04, size=len(y))
ax1.plot(x,y, 'y.', alpha=0.05)
ax1.set_title("boxplot of Age")
ax2 = fig.add_subplot(2,3,2)
bp2= ax2.boxplot(med_Aptdata['waitday'])
y2 = med_Aptdata.waitday
x2 = np.random.normal(1, 0.04, size=len(y2))
ax2.plot(x2,y2, 'y.', alpha=0.08)
ax2.set_title("boxplot of waitday")
plt.show()



# In[29]:


#box plot  and histogram
import matplotlib.pyplot as plt
import pylab
import matplotlib
matplotlib.style.use('ggplot')
```

```
get_ipython().magic(u'matplotlib inline')



fig = plt.figure(figsize=(10, 10))
ax1 = fig.add_subplot(2,3,1)
ax1.boxplot(med_Aptdata['Age'])
ax1.set_title("boxplot of Age")
ax2 = fig.add_subplot(2,3,2)
ax2.boxplot(med_Aptdata['waitday'])
ax2.set_title("boxplot of waitday")


plt.show()



# In[30]:


#histogram
plt.figure()
med_Aptdata[['Age', 'waitday']].hist(color='blue',alpha=0.7,edgecolor="black",figsize=(10, 10) )



# there are few outlier in age but based on box plot and histogram, it doent look like it will affect our model building latr. On the other hand, have long tail on the right (right skwenness) in waitday meaning that we have some extreme high values for waitday and it may cause problem for prediction. there are many way  such as over/under sampling , transformation  to address
```

(skewnness) issue. In this case, I will use biinning method to try to group large value togather and balance out the distribution since it will be more interpretable and help to solve the outlier issue.

```
#
#
#
#
# most obeservation are under age ~70.
#
# most people make appoitment fo no later than a month
#


# In[12]:


#additional preparation after exploring the data(binning the waitday)
# group: (sameday, 0; withinweek,7. over_a_month,30 )


bins = [-1,2,7,14,30,360]
labels = ["wihtinTwoDay", "Week", "TwoWeeks", "Month","MoreThanAMonth"]
wait_period = pd.cut(med_Aptdata['waitday'], bins, labels=labels)
med_Aptdata['Wait_period'] = wait_period



# In[13]:


med_Aptdata.drop(['waitday'],axis=1,inplace=True)
```

```
# In[13]:


med_Aptdata.head()



# Categorical


# In[34]:


#bar plot target: noshow


a=['No-show']
fig = plt.figure(figsize=(20, 20))
for i in range(len(a)):

    ax=fig.add_subplot(3,3,i+1)
    ax.set_title(a[i])
    med_Aptdata[a[i]].value_counts().plot(kind='bar',color='blue')
fig.tight_layout()
plt.show()


#percentable table
med_Aptdata.groupby('No-show').size() * 100 / len(med_Aptdata)
```

```
# target is inbalance! 80% of observation does not miss appoitment. only 20% miss the
appoitment. Need to consider about the inbalanced target when we build models

# In[35]:



#bar plot 1
a=['Gender','Neighbourhood','Scholarship','Wait_period']
fig = plt.figure(figsize=(20, 20))
for i in range(len(a)):

    ax=fig.add_subplot(2,2,i+1)
    ax.set_title(a[i])
    med_Aptdata[a[i]].value_counts().plot(kind='bar',color='blue')
fig.tight_layout()
plt.show()



# for  our obesravation, we have more case in female than male. most poepl's waitday is within 2
days. most people dont recieved scholarship(special offer)

# In[166]:


#bat plot 2
a=['Hipertension','Diabetes','Alcoholism','Handcap','SMS_received']
fig = plt.figure(figsize=(20, 20))
```

```
for i in range(len(a)):

    ax=fig.add_subplot(3,3,i+1)
    ax.set_title(a[i])
    med_Aptdata[a[i]].value_counts().plot(kind='bar',color='blue')
fig.tight_layout()
plt.show()



# In[200]:


#percentable table
med_Aptdata.groupby('SMS_received').size() * 100 / len(med_Aptdata)



# Also, majorraity of people doesnt have any labeled conditions(such as Diabetes, alchoholism,
handicap..etc).
#
# ~67% of people signed up for sms reminder and ~32 are not.


# ## mutivariate


# #### Age vs noshow
#


# In[174]:
```

```
# box plot by survive for numerical

fig = plt.figure(figsize=(30, 30))

a= med_Aptdata.boxplot(column= 'Age',by='No-show')


plt.show()


# in general, people who miss the appointment tend to be younger.

# In[251]:

#Anova test
import scipy.stats as stats

stats.f_oneway(med_Aptdata.loc[med_Aptdata['No-show'] ==
'Yes','Age'],med_Aptdata.loc[med_Aptdata['No-show'] == 'No','Age'])


# Statistically comfimed that there are significant evidence that age is correlated with noshow
based on anova test (p value < 0.05 indicates that mean of ages for two group(noshow vs show)
are differnt )
```

```
# #### Comparing no show rate between groups for each categorical attributes


# In[216]:


#Cross table percentagewise
attributes
=['Gender','Neighbourhood','Scholarship','Wait_period','Hipertension','Diabetes','Alcoholism','Ha
ndcap','SMS_received']

for i in attributes:
    print 'percentage cross table (* Comapring No- show = yes) :', '  ', i
    print pd.crosstab(med_Aptdata[i],med_Aptdata["No-show"]).apply(lambda x: x/x.sum(), 1)
    print "
    print "
    print "



# some significant differnt between group can be detected for some attribute which indicate that
we may have some good predictor.
#
#
# For example:
#
# for gender : female and male no show rate are not significantly different (20% vs 19%)
#
# for neighbor : PARQUE INDUSTRIAL has 0% no show rate.
```

```
#
# for Wait_period: no show rate is significant lower for waittime withintwoday( 0.09% ) than
other groups (25%-33%)
#
# for handicap: Handicap_4 has significantly higher no show rate(33%) compare to the other
groups(17%-23%)
#
# for sms recievied: surpringly , people who are given sms remider are 9% more likley to miss the
appointment.
#


# In[222]:


#stack bar plot by survive for categorical
attributes
=['Gender','Neighbourhood','Scholarship','Wait_period','Hipertension','Diabetes','Alcoholism','Ha
ndcap','SMS_received']

for i in attributes:
    pd.crosstab(med_Aptdata[i],med_Aptdata["No-show"]).apply(lambda x: x/x.sum(),
1).plot.bar()



# Visulization version of the crosstable


# In[243]:
```

```
from scipy import stats
attributes
=['Gender','Neighbourhood','Scholarship','Wait_period','Hipertension','Diabetes','Alcoholism','Ha
ndcap','SMS_received']

for i in attributes:
    a= pd.crosstab(med_Aptdata[i],med_Aptdata["No-show"])
    g, p, dof, expctd = scipy.stats.chi2_contingency(a)
    print 'P-value for chi-square test of independency for ',i,': ', p



# for comfirmation of corrlation, for chi-square test of independency give us more clear idea:
#
# the result shows there are significant evidence that no-show is correlated with following
attributes:
#
# Neighbourhood, Scholarship, Wait_period,Hipertension,Diabetes,and SMS_received.
#
#


# ##  innitail feature selection


# In[14]:
```

```
# Neighbourhood has too many level, do the intial  prediction with and without neigborhood
######
#and comapre performance using ttest to see if dropping Neighbourhood will affect our
prediction



#####trial --- with neigborhood, x_mat1,y1 ######
trial = med_Aptdata
# Separate the target attribute ("No-show")
# x : predictors, y : target
x1 = trial.drop("No-show", axis=1, inplace = False)
y1 = trial["No-show"]

#Convert the selected dataset into the Standard Spreadsheet format

# get_dummies" function to create dummy variables and converting to standard spreadsheet
format
x_mat1 = pd.get_dummies(x1)
x_mat1.head(5)
y1=pd.get_dummies(y1)
y1.head()
#no show up [1: noshow, 0: show]
y1=y1['Yes']

#80-20 split
from sklearn.cross_validation import train_test_split
```

```
x_train1, x_test1, y_train1, y_test1 = train_test_split(x_mat1, y1, test_size=0.2,
random_state=33)




#####trial 2 --- without neigborhood ######
trial2 = med_Aptdata.drop(['Neighbourhood'], axis=1, inplace = False)
# Separate the target attribute ("No-show")
# x : predictors, y : target
x2 = trial2.drop("No-show", axis=1, inplace = False)
y2 = trial2["No-show"]

#Convert the selected dataset into the Standard Spreadsheet format
# get_dummies" function to create dummy variables and converting to standard spreadsheet
format
x_mat2 = pd.get_dummies(x2)
y2=pd.get_dummies(y2)
#no show up [1: noshow, 0: show]
y2=y2['Yes']

#80-20 split
from sklearn.cross_validation import train_test_split
x_train2, x_test2, y_train2, y_test2 = train_test_split(x_mat2, y2, test_size=0.2,
random_state=33)
```

```
# In[83]:


#comapre model wiht NB
cv_N = model_selection.cross_val_score(GaussianNB(), x_train1, y_train1,
cv=10,scoring='precision')
cv_Non = model_selection.cross_val_score(GaussianNB(), x_train2, y_train2,
cv=10,scoring='precision')

results = [cv_N,cv_Non]
names = ['with_N','Without_N']



# Compare Algorithms
fig = plt.figure()
fig.suptitle('Algorithm Comparison: precision on train')
ax = fig.add_subplot(111)
plt.boxplot(results)
ax.set_xticklabels(names)
plt.show()


# paried t test to compare two model(Random forest vs gradient boosting)
from scipy import stats
stats.ttest_rel(cv_N, cv_Non)
```

```
# In[70]:


# Make predictions on test dataset (NB)

NB= GaussianNB()

NB.fit(x_train1, y_train1)

NBpd = NB.predict(x_test1)

print('Prediction report: NB')

print '\n'

print'accuracy: %0.3f' % accuracy_score(y_test1, NBpd)

print '\n'

print 'confusion matrix: '

print confusion_matrix(y_test1, NBpd)

print '\n'

print 'classification report wiht N : '

print(classification_report(y_test1, NBpd))



# In[72]:


# Make predictions on test dataset (NB)

NB= GaussianNB()

NB.fit(x_train2, y_train2)

NBpd = NB.predict(x_test2)

print('Prediction report: NB')

print '\n'
```

```
print'accuracy: %0.3f' % accuracy_score(y_test2, NBpd)
print '\n'
print 'confusion matrix: '
print confusion_matrix(y_test2, NBpd)
print '\n'
print 'classification report wihtout N : '
print(classification_report(y_test2, NBpd))



# Surprisingly, the average precison is better without attribute neighbor , t test also comfirm there are significant differnece in average accuracy between two
#
# Therefore, dropping attribute neighboorhood not only wont hurt the result but improve the result and it simplified the model.
#
#
# Also, the prediciton performance on people who doesnt show up for appointment(noshow_yes =1) are really bad for both (even worse than random guess). The reason is that we have inbalance data so we have to find a way to fix this problem.


# In[15]:


#dropping 'Neighbourhood'
med_Aptdata.drop(['Neighbourhood'], axis=1, inplace = True)
```

```
# In[16]:


med_Aptdata.info()



# In[17]:


med_Aptdata.head()



# ### potential ways of fixing  imbalnaced data


# few way to deal with imbalance dataset
#
# * use tree based model (decision tree, emsemble trees) with smote(oversample with
synthetically generated data points that are not too different from the minority class data points
you actually have)
# * Penalized-SVM
# * oversampling (potential overfitting)
# * undersampling (loss of information)
#
# Evaluation which model is better
#
# * Precision/Specificity: how many selected instances are relevant.
# * using auroc
#
```

```
#

#

#

#


# ## Using Smote to deal with inbalance data


# In[18]:


# pip install -u imbalanced-learn

#smote

from imblearn.over_sampling import SMOTE



# In[19]:


smote=SMOTE(kind = "regular",random_state=0, ratio = 1.0)

smote_predictors,smote_target=smote.fit_sample(x_train2,y_train2)


print y_train2.value_counts()


print "

print (np.bincount(smote_target))



# Now, for our training set, we have equal number of observation for both noshow and show
```

```
# In[ ]:


#141258 obeservation on training and 23 dimensions



# In[21]:


#too big, take sampele, run twice to get 1%

A,smote_predictors,B,smote_target= train_test_split(smote_predictors,smote_target,
test_size=0.1, random_state=33)



# In[22]:


print (np.bincount(smote_target))



# In[23]:


smote_target.shape



# In[24]:
```

```
smote_predictors.shape



# # Modeling with parameter tuning on the selected features

# In[275]:



## imbalance training set

# Training set: [x_train2, y_train2]

##balacne trainign set

#  Training set: [smote_predictors,smote_target]



## Unseen testing set : [x_test2, y_test2]

### target y [1: noshow, 0: show]



# performance metrix: precision and recall for noshow =1

#

# precision measure how accuracy our prediction on noshow =1

#

# recall measure the proportion of the noshow case found.

#
```

```
# in this case, we want to be able to idnetify noshow case so recall will be more useful.(looking for model's that maxmize the average recall)

# ## Naive bayes

# In[25]:

#different Naive Bayes
from sklearn.naive_bayes import GaussianNB
from sklearn.naive_bayes import BernoulliNB
from sklearn.naive_bayes import MultinomialNB


from sklearn import cross_validation
# Spot Check Algorithms
models = []
models.append(('MNB', MultinomialNB()))
models.append(('BNB', BernoulliNB()))
models.append(('GNB', GaussianNB()))

# evaluate each model in turn
results = []
names = []
for name, model in models:
        cv_results = cross_validation.cross_val_score(model, smote_predictors, smote_target, cv=10)
```

```
        results.append(cv_results)
        names.append(name)
        msg = "%s: %.3f (%.3f)" % (name, cv_results.mean(), cv_results.std())
        print(msg)
```

# BernoulliNB has better performacne(recall) than others, lets tune the parameters

# In[30]:

```
BNB= BernoulliNB(class_prior=None, fit_prior=True)
```

# Use a grid over parameters of interest

```
param_grid = { "alpha" : [ 0 ,0.1, 1, 10, 100],'binarize':[0 ,0.1, 1, 10, 100]}

CV_BNB = GridSearchCV(estimator=BNB, param_grid=param_grid, cv= 10)
CV_BNB.fit(smote_predictors, smote_target)
print CV_BNB.best_params_
```

# In[28]:

# Optimized BNB

```
BNB= BernoulliNB(binarize=0,alpha=0,class_prior=None, fit_prior=True)



# fit the model with training set  - CV(10)  and predict avg accuracy_
BNB_scores = model_selection.cross_val_score(BNB, smote_predictors, smote_target, cv=10)
print("recall of bernoulli NB Train : %0.2f (+/- %0.2f)" % (BNB_scores.mean()*100,
BNB_scores.std()*100))

# confidence intervals
import numpy
alpha = 0.95
p = ((1.0-alpha)/2.0) * 100
lower = max(0.0, numpy.percentile(BNB_scores, p))
p = (alpha+((1.0-alpha)/2.0)) * 100
upper = min(1.0, numpy.percentile(BNB_scores, p))
print('%.1f confidence interval %.1f%% and %.1f%%' % (alpha*100, lower*100, upper*100))



# In[54]:

#nb model predcition on test set
#alpha=1.0, binarize=2.0, class_prior=None, fit_prior=True)
# Make predictions on test dataset
BNB= BernoulliNB(class_prior=None, fit_prior=True)
BNB.fit(smote_predictors, smote_target)
```

```
BNBpd = BNB.predict(x_test2)

print('Prediction report: Naive Bayes')

print '\n'

print'accuracy: %0.3f' % accuracy_score(y_test2, BNBpd)

print '\n'

print 'confusion matrix: '

print confusion_matrix(y_test2, BNBpd)

print '\n'

print 'classification report : '

print(classification_report(y_test2, BNBpd))



# best tuned Naive bayes:

#

# __BNB= BernoulliNB(alpha=0, binarize=0, class_prior=None, fit_prior=True)__

#

#

# Prediction on unseen set:

#

# Recall for no show :77%, precision for no show : 30%

#

# Average recall: 58%


# ## Tree based model and penalized SVM

#
```

```
# * __Ramdom forest with balance set__


# In[33]:


#rfc = RandomForestClassifier

#gris search to tune


from sklearn.grid_search import GridSearchCV


rfc = RandomForestClassifier(n_jobs=-1, max_features='sqrt', oob_score = True)


# Use a grid over parameters of interest

param_grid = {

        "n_estimators" : [9, 18, 27, 36, 45, 54, 63],

        "max_depth" : [1, 5, 10, 15, 20, 25, 30],

        "min_samples_leaf" : [1, 5, 10,50]}


CV_rfc = GridSearchCV(estimator=rfc, param_grid=param_grid, cv= 10)

CV_rfc.fit(smote_predictors, smote_target)

print CV_rfc.best_params_



# In[30]:


# Optimized RF classifier to maxize precisoon
```

```
rfcb = RandomForestClassifier(n_estimators=36, max_depth=10, max_features='sqrt',
min_samples_leaf = 10)



# fit the model with training set  - CV(10)  and predict avg accuracy
rfcb_scores = model_selection.cross_val_score(rfcb, smote_predictors, smote_target, cv=10)
print("Accuracy of Random forest on balanced Trainning set  : %0.2f (+/- %0.2f)" %
(rfcb_scores.mean()*100, rfcb_scores.std()*100))

# confidence intervals
import numpy
alpha = 0.95
p = ((1.0-alpha)/2.0) * 100
lower = max(0.0, numpy.percentile(rfcb_scores, p))
p = (alpha+((1.0-alpha)/2.0)) * 100
upper = min(1.0, numpy.percentile(rfcb_scores, p))
print('%.1f confidence interval %.1f%% and %.1f%%' % (alpha*100, lower*100, upper*100))



# In[56]:

# Make predictions on test dataset (RF) with optimal model trained with balanced data
```

```
rfcb = RandomForestClassifier(n_estimators=36, max_depth=10, max_features='sqrt',
min_samples_leaf = 1)
rfcb.fit(smote_predictors, smote_target)
RFpdb = rfcb.predict(x_test2)
print('Prediction report: Random Forest')
print '\n'
print'accuracy: %0.3f' % accuracy_score(y_test2, RFpdb)
print '\n'
print 'confusion matrix: '
print confusion_matrix(y_test2, RFpdb)
print '\n'
print 'classification report : '
print(classification_report(y_test2, RFpdb))



# best tuned random forest model (bagging technique):
#
# __rfcb =RandomForestClassifier(n_estimators=36, max_depth=10, max_features='sqrt',
min_samples_leaf = 1)__
#
# Prediction on unseen set:
#
# Recall for no show :66%, precision for no show : 32%
#
# Average recall : 65%
```

```
# __comparing with NB__
#
# Better perfomacne than NB (accuracy is higher and average recall is higher meaning we
identify more case correctly,overall; however recall for noshow is 66% (RF) which is less than
77%(NB) )


# * __gradian boosting tree _balanced training data__


# In[ ]:


gbc = GradientBoostingClassifier()



# Use a grid over parameters of interest
param_grid = {
        "n_estimators" : [9, 18, 27, 36, 45],
        "max_depth" : [5, 10, 15, 20],
        "min_samples_leaf" : [5,10,20,50]}


CV_gbc = GridSearchCV(estimator=gbc, param_grid=param_grid, cv= 10)
CV_gbc.fit(smote_predictors, smote_target)
print CV_gbc.best_params_



# In[32]:
```

```
# Optimized GB classifier
gbc = GradientBoostingClassifier(n_estimators=36, max_depth=10, max_features='sqrt',
min_samples_leaf = 10)



# fit the model with training set  - CV(10)  and predict avg accuracy_
gbc_scores = model_selection.cross_val_score(gbc, smote_predictors, smote_target, cv=10)
print("Accuracy of Gradiant boosting dT on balanced Trainning set: %0.2f (+/- %0.2f)" %
(gbc_scores.mean()*100, gbc_scores.std()*100))

# confidence intervals
import numpy
alpha = 0.95
p = ((1.0-alpha)/2.0) * 100
lower = max(0.0, numpy.percentile(gbc_scores, p))
p = (alpha+((1.0-alpha)/2.0)) * 100$
upper = min(1.0, numpy.percentile(gbc_scores, p))
print('%.1f confidence interval %.1f%% and %.1f%%' % (alpha*100, lower*100, upper*100))



# In[60]:

# Make predictions on test dataset (GB)
gbc.fit(smote_predictors, smote_target)
gbcpd = gbc.predict(x_test2)
```

```
print('Prediction report: gradient boosting')

print '\n'

print'accuracy: %0.3f' % accuracy_score(y_test2, gbcpd)

print '\n'

print 'confusion matrix: '

print confusion_matrix(y_test2, gbcpd)

print '\n'

print 'classification report : '

print(classification_report(y_test2, gbcpd))



# best tuned gradian boosing tree(boosting technique):

#

# __gbc =GradientBoostingClassifier(n_estimators=36, max_depth=10, max_features='sqrt',

min_samples_leaf = 1)__

#

#

# Prediction on unseen set:

#

# Recall for no show :63%, precision for no show : 33%

#

# Average recall: 66%


# * __kernel support vector machine__


# linear SVM
```

```
# In[ ]:


#####linear

from sklearn.svm import SVC


linear=SVC(kernel='linear',,probability=True)



# Use a grid over parameters of interest

param_grid = { "C" : [0.01 ,0.1, 1, 10]}


CV_linear = GridSearchCV(estimator=linear, param_grid=param_grid, cv= 10)

CV_linear.fit(smote_predictors, smote_target)

print CV_linear.best_params_



# In[34]:


# Linear SVM classifier

#####linear

from sklearn.svm import SVC

linear = SVC(kernel='linear',C=1,,probability=True)



# fit the model with training set  - CV(10)  and predict avg accuracy_
```

```
linear_scores = model_selection.cross_val_score(linear, smote_predictors, smote_target, cv=10,
scoring='accuracy')
print("linear SVM accuracy: %0.2f (+/- %0.2f)" % (linear_scores.mean()*100,
linear_scores.std()*100))

# confidence intervals
import numpy
alpha = 0.95
p = ((1.0-alpha)/2.0) * 100
lower = max(0.0, numpy.percentile(linear_scores, p))
p = (alpha+((1.0-alpha)/2.0)) * 100
upper = min(1.0, numpy.percentile(linear_scores, p))
print('%.1f confidence interval %.1f%% and %.1f%%' % (alpha*100, lower*100, upper*100))



# In[61]:


#prediction linear svm


#LINEAR SVM
# Make predictions on test dataset
linear = SVC(kernel='linear',C=1,probability=True)
linear.fit(smote_predictors, smote_target)
linearpd = linear.predict(x_test2)
print('Prediction report: linear SVM')
print '\n'
```

```
print'accuracy: %0.3f' % accuracy_score(y_test2, linearpd)

print '\n'

print 'confusion matrix: '

print confusion_matrix(y_test2, linearpd)

print '\n'

print 'classification report : '

print(classification_report(y_test2, linearpd))



# SVM With kernal tricks and tuning paremater

#


# In[51]:


# kernel support vector machine




# penalize(weighted)

kernelsvm=SVC(class_weight='balanced',probability=True)



# Use a grid over parameters of interest

param_grid = { "kernel" : ['rbf','poly','sigmoid'],

          "gamma" : [1e-1, 1e-2, 1e-3, 1e-4],

          "C" : [0.01 ,0.1, 1, 10, 100, 1000]}
```

```
CV_kernelsvm = GridSearchCV(estimator=kernelsvm, param_grid=param_grid, cv= 10)
CV_kernelsvm.fit(smote_predictors, smote_target)
print CV_kernelsvm.best_params_


# In[ ]:


# Optimized kernal SVM classifier


poly_SVM = SVC(kernel = 'poly', C = 0.1, gamma =
0.1,class_weight='balanced',probability=True)



# fit the model with training set  - CV(10)  and predict avg accuracy_
poly_SVM_scores = model_selection.cross_val_score(poly_SVM, smote_predictors,
smote_target, cv=10, scoring='accuracy')
print("poly_SVM Train accuracy: %0.2f (+/- %0.2f)" % (poly_SVM_scores.mean()*100,
poly_SVM_scores.std()*100))


# confidence intervals
import numpy
alpha = 0.95
p = ((1.0-alpha)/2.0) * 100
lower = max(0.0, numpy.percentile(poly_SVM_scores, p))
p = (alpha+((1.0-alpha)/2.0)) * 100
```

```
upper = min(1.0, numpy.percentile(poly_SVM_scores, p))
print('%.1f confidence interval %.1f%% and %.1f%%' % (alpha*100, lower*100, upper*100))



# In[ ]:


#prediction, kernal svm


# Make predictions on test dataset
poly_SVM = SVC(kernel = 'poly', C = 100, gamma = 0.1)
poly_SVM.fit(smote_predictors, smote_target)
poly_SVMpd = poly_SVM.predict(x_test2)
print('Prediction report: polynominal kernel SVM')
print '\n'
print'accuracy: %0.3f' % accuracy_score(y_test2, poly_SVMpd)
print '\n'
print 'confusion matrix: '
print confusion_matrix(y_test2, poly_SVMpd)
print '\n'
print 'classification report : '
print(classification_report(y_test2, poly_SVMpd))



# In[ ]:
```

```
# In[ ]:


#compare svm and linear svm


# paried t test to compare linear vs kernal svm


from scipy import stats

stats.ttest_rel(linear_scores, poly_SVM_scores)




# best tuned kernel SVM:

#

#

# __linear = SVC(kernel='linear',C=1)__

#

# Prediction on unseen set:

#

# acurracy: 59%

#

# Recall for no show :77%

#

# overall recall: 59%


# ## Stacking (Naive bayes ,boosting ,begging, svm)
```

```
# In[48]:


#import majority_vote_classifier as mvc



from sklearn.ensemble import VotingClassifier



estimators = []
model1 = BernoulliNB(alpha=0, binarize=0, class_prior=None, fit_prior=True)
estimators.append(('bernouli NB', model1))
model2 = RandomForestClassifier(n_estimators=36, max_depth=10, max_features='sqrt',
min_samples_leaf = 10)
estimators.append(('Random forest', model2))
model3 = GradientBoostingClassifier(n_estimators=36, max_depth=10, max_features='sqrt',
min_samples_leaf = 1)
estimators.append(('gradient boosting', model3))
model4 = SVC(kernel='linear',C=1,probability=True)
estimators.append(('linear svm', model4))



# create the ensemble model
stacking = VotingClassifier(estimators,voting='soft')
results = model_selection.cross_val_score(stacking, smote_predictors, smote_target, cv=10)
```

```
print("stacking model Train accuracy: %0.2f (+/- %0.2f)" % (results.mean()*100,
results.std()*100))

# confidence intervals
import numpy
alpha = 0.95
p = ((1.0-alpha)/2.0) * 100
lower = max(0.0, numpy.percentile(results , p))
p = (alpha+((1.0-alpha)/2.0)) * 100
upper = min(1.0, numpy.percentile(results , p))
print('%.1f confidence interval %.1f%% and %.1f%%' % (alpha*100, lower*100, upper*100))



# In[50]:

#prediction



stacking.fit(smote_predictors, smote_target)
stpd = stacking.predict(x_test2)
print('Prediction report: Stacking')
print '\n'
print'accuracy: %0.3f' % accuracy_score(y_test2, stpd)
print '\n'
print 'confusion matrix: '
print confusion_matrix(y_test2, stpd)
```

```
print '\n'

print 'classification report : '

print(classification_report(y_test2, stpd))



# ## compare performacne of machine learning Techniques in term of recall


# In[40]:


from sklearn import cross_validation

# Spot Check Algorithms

models = []

models.append(('BernoulliNB', BernoulliNB(alpha=0, binarize=0, class_prior=None,

fit_prior=True)))

models.append(('Emsembel:RF', RandomForestClassifier(n_estimators=36, max_depth=10,

max_features='sqrt', min_samples_leaf = 1)))

models.append(('Emsemble: GB', GradientBoostingClassifier(n_estimators=45, max_depth=15,

max_features='sqrt', min_samples_leaf = 6)))

models.append(('SVM',SVC(kernel='linear',C=1)))

models.append(('Emsembel: Stacking',stacking ))


# evaluate each model in turn

results = []

names = []

print '10 fold Cross validaiton report for each model: average Acuuracy and Standardiviation'

print '\n'
```

```
for name, model in models:
        cv_results = cross_validation.cross_val_score(model, smote_predictors, smote_target,
cv=10)
        results.append(cv_results)
        names.append(name)

# boxplot
fig = plt.figure()
fig.suptitle('Algorithm Comparison: Presion on train')
ax = fig.add_subplot(111)
plt.boxplot(results)
ax.set_xticklabels(names)
plt.show()



# In[ ]:


# paried t test to compare best model with second best
from scipy import stats
stats.ttest_rel(gbc_scores, rfc_scores,gbc_scores,)



# In[ ]:


results=[BNB_scores,rfcb_scores,linear_scores,results]
```

```
names=['BernoulliNB','Emsemble: RF','Emsemble: GB','SVM', 'stacking']


# boxplot
fig = plt.figure()
fig.suptitle('Algorithm Comparison: Presion on train')
ax = fig.add_subplot(111)
plt.boxplot(results)
ax.set_xticklabels(names)
plt.show()



# ## compare classifiers's prediction performacne on testing set (precision on noshow)


# ### Model Evaluation


# In[84]:


#### ROC Curve



nbP=BNB.predict_proba(x_test2)[:,1]
rfcP=rfcb.predict_proba(x_test2)[:,1]
gbcP=gbc.predict_proba(x_test2)[:,1]
#linearP=linear.predict_proba(x_test2)[:,1]
#stcP=stacking.predict_proba(x_test2)[:,1]
```

```
models=[nbP,rfcP,gbcP]
#stcP
label=['BernoulliNB','Emsemble: RF','Emsemble: GB']

from sklearn import metrics
# plotting ROC curves
plt.figure(figsize=(20, 20))

for i in range(len(models)):
    fpr, tpr,thresholds= metrics.roc_curve(y_test2,models[i])
    plt.plot(fpr,tpr,label=label[i])
plt.xlim([0.0,1.0])
plt.ylim([0.0,1.0])
plt.title('ROC curve for SURVIVIED classifers')
plt.xlabel('False positive rate (1-specificity)')
plt.ylabel('True positive rate (sensitivity)')
plt.legend(loc=4,)

plt.show()



# In[71]:

#AUC
```

```
from sklearn.metrics import roc_curve, auc, roc_auc_score


roc_auc_score(y_test2, linearP)



# ## Feature importacne


# In[87]:


# sort importances
indices = np.argsort(gbc.feature_importances_)
names = list(x_train2)
# plot as bar chart
plt.figure(figsize=(15, 15))
plt.barh(np.arange(len(names)), gbc.feature_importances_[indices])
plt.yticks(np.arange(len(names)) + 0.25, np.array(names)[indices])
_ = plt.xlabel('Relative importance')


plt.show()
```