Different_methods_of_comparison

May 18, 2019

In [318]:
```
#Load the librarys
import pandas as pd #To work with dataset
import numpy as np #Math library
import seaborn as sns #Graph library that use matplot in background
import matplotlib.pyplot as plt #to plot some parameters in seaborn

#Importing the data
df_credit = pd.read_csv("C:/Users/Kai/Desktop/Assignment3/data/credit-g_preproccess.
```

In [319]:
```
# First Look at the data:
## Looking the Type of Data
## Null Numbers
## Unique values

df_credit.head(6)
```

Out[319]:

checking_status	duration	credit_history	purpose	credit_amount	savings_status	employment	installment_commitment	personal_status
<0	6	'critical/other existing credit'	radio/tv	1169	'no known savings'	>=7	4	'male single'
0<=X<200	48	'existing paid'	radio/tv	5951	<100			
'no checking'	12	'critical/other existing credit'	education	2096	<100			
<0	42	'existing paid'	furniture/equipment	7882	<100			
<0	24	'delayed previously'	'new car'	4870	<100			
'no checking'	36	'existing paid'	education	9055	'no known savings'			

```
0<=X<200            1<=X<4                      2  'female div/dep/mar'
'no checking'       4<=X<7                      2         'male single'
<0                  4<=X<7                      2         'male single'
<0                  1<=X<4                      3         'male single'
'no checking'       1<=X<4                      2         'male single'

                other_parties  residence_since   property_magnitude  age  \
checking_status
<0                       none                4        'real estate'   67
0<=X<200                 none                2        'real estate'   22
'no checking'            none                3        'real estate'   49
<0                   guarantor               4     'life insurance'   45
<0                       none                4  'no known property'   53
'no checking'            none                4  'no known property'   35

                other_payment_plans     housing  existing_credits  \
checking_status
<0                             none         own                 2
0<=X<200                       none         own                 1
'no checking'                  none         own                 1
<0                             none  'for free'                 1
<0                             none  'for free'                 2
'no checking'                  none  'for free'                 1

                                   job  num_dependents own_telephone  \
checking_status
<0                             skilled               1           yes
0<=X<200                       skilled               1          none
'no checking'    'unskilled resident'               2          none
<0                             skilled               2          none
<0                             skilled               2          none
'no checking'    'unskilled resident'               2           yes

                foreign_worker class
checking_status
<0                         yes  good
0<=X<200                   yes   bad
'no checking'              yes  good
<0                         yes  good
<0                         yes   bad
'no checking'              yes  good
```

In [320]:
```
#Searching for Missings,type of data and also known the shape of data
print(df_credit.info())

# Let us check if there is any null values
print(df_credit.isnull().sum())
```

```
          df_credit.shape
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 1000 entries, <0 to 0<=X<200
Data columns (total 20 columns):
duration                  1000 non-null int64
credit_history            1000 non-null object
purpose                   1000 non-null object
credit_amount             1000 non-null int64
savings_status            1000 non-null object
employment                1000 non-null object
installment_commitment    1000 non-null int64
personal_status           1000 non-null object
other_parties             1000 non-null object
residence_since           1000 non-null int64
property_magnitude        1000 non-null object
age                       1000 non-null int64
other_payment_plans       1000 non-null object
housing                   1000 non-null object
existing_credits          1000 non-null int64
job                       1000 non-null object
num_dependents            1000 non-null int64
own_telephone             1000 non-null object
foreign_worker            1000 non-null object
class                     1000 non-null object
dtypes: int64(7), object(13)
memory usage: 164.1+ KB
None
duration                  0
credit_history            0
purpose                   0
credit_amount             0
savings_status            0
employment                0
installment_commitment    0
personal_status           0
other_parties             0
residence_since           0
property_magnitude        0
age                       0
other_payment_plans       0
housing                   0
existing_credits          0
job                       0
num_dependents            0
own_telephone             0
foreign_worker            0
class                     0
```

```
dtype: int64
```

Out[320]: (1000, 20)

In [321]:
```
#Looking unique values
print(df_credit.nunique())
```

```
duration                  33
credit_history             5
purpose                   10
credit_amount            921
savings_status             5
employment                 5
installment_commitment     4
personal_status            4
other_parties              3
residence_since            4
property_magnitude         4
age                       53
other_payment_plans        3
housing                    3
existing_credits           4
job                        4
num_dependents             2
own_telephone              2
foreign_worker             2
class                      2
dtype: int64
```

In [322]:
```
# Transforming the data into Dummy variables (IMPORTANT)
def one_hot_encoder(df, nan_as_category = False):
    original_columns = list(df.columns)
    categorical_columns = [col for col in df.columns if df[col].dtype == 'object']
    df = pd.get_dummies(df, columns= categorical_columns, dummy_na= nan_as_category,
    new_columns = [c for c in df.columns if c not in original_columns]
    return df, new_columns

df_credit, new_colunms = one_hot_encoder(df_credit)
df_credit.head(6)
```

Out[322]:

checking_status	duration	credit_amount	installment_commitment	\
<0	6	1169	4	
0<=X<200	48	5951	2	
'no checking'	12	2096	2	
<0	42	7882	2	
<0	24	4870	3	

```
'no checking'          36           9055                       2

                residence_since  age  existing_credits  num_dependents  \
checking_status
<0                            4   67                 2               1
0<=X<200                      2   22                 1               1
'no checking'                 3   49                 1               2
<0                            4   45                 1               2
<0                            4   53                 2               2
'no checking'                 4   35                 1               2

                credit_history_'critical/other existing credit'  \
checking_status
<0                                                             1
0<=X<200                                                       0
'no checking'                                                  1
<0                                                             0
<0                                                             0
'no checking'                                                  0

                credit_history_'delayed previously'  \
checking_status
<0                                                0
0<=X<200                                          0
'no checking'                                     0
<0                                                0
<0                                                1
'no checking'                                     0

                credit_history_'existing paid'     ...      \
checking_status                                    ...
<0                                             0   ...
0<=X<200                                       1   ...
'no checking'                                  0   ...
<0                                             1   ...
<0                                             0   ...
'no checking'                                  1   ...

                other_payment_plans_none  other_payment_plans_stores  \
checking_status
<0                                     1                           0
0<=X<200                               1                           0
'no checking'                          1                           0
<0                                     1                           0
<0                                     1                           0
'no checking'                          1                           0

                housing_own  housing_rent  job_'unemp/unskilled non res'  \
```

```
checking_status
<0                          1             0                                0
0<=X<200                    1             0                                0
'no checking'               1             0                                0
<0                          0             0                                0
<0                          0             0                                0
'no checking'               0             0                                0

                  job_'unskilled resident'  job_skilled  own_telephone_yes  \
checking_status
<0                                       0            1                  1
0<=X<200                                 0            1                  0
'no checking'                            1            0                  0
<0                                       0            1                  0
<0                                       0            1                  0
'no checking'                            1            0                  1

                  foreign_worker_yes  class_good
checking_status
<0                                 1           1
0<=X<200                           1           0
'no checking'                      1           1
<0                                 1           1
<0                                 1           0
'no checking'                      1           1

[6 rows x 46 columns]
```

```
In [323]: print(df_credit.info())
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 1000 entries, <0 to 0<=X<200
Data columns (total 46 columns):
duration                                          1000 non-null int64
credit_amount                                     1000 non-null int64
installment_commitment                            1000 non-null int64
residence_since                                   1000 non-null int64
age                                               1000 non-null int64
existing_credits                                  1000 non-null int64
num_dependents                                    1000 non-null int64
credit_history_'critical/other existing credit'   1000 non-null uint8
credit_history_'delayed previously'               1000 non-null uint8
credit_history_'existing paid'                    1000 non-null uint8
credit_history_'no credits/all paid'              1000 non-null uint8
purpose_'new car'                                 1000 non-null uint8
purpose_'used car'                                1000 non-null uint8
purpose_business                                  1000 non-null uint8
purpose_education                                 1000 non-null uint8
```

```
purpose_furniture/equipment                          1000 non-null uint8
purpose_other                                        1000 non-null uint8
purpose_radio/tv                                     1000 non-null uint8
purpose_repairs                                      1000 non-null uint8
purpose_retraining                                   1000 non-null uint8
savings_status_100<=X<500                            1000 non-null uint8
savings_status_500<=X<1000                           1000 non-null uint8
savings_status_<100                                  1000 non-null uint8
savings_status_>=1000                                1000 non-null uint8
employment_4<=X<7                                    1000 non-null uint8
employment_<1                                        1000 non-null uint8
employment_>=7                                       1000 non-null uint8
employment_unemployed                                1000 non-null uint8
personal_status_'male div/sep'                       1000 non-null uint8
personal_status_'male mar/wid'                       1000 non-null uint8
personal_status_'male single'                        1000 non-null uint8
other_parties_guarantor                              1000 non-null uint8
other_parties_none                                   1000 non-null uint8
property_magnitude_'no known property'               1000 non-null uint8
property_magnitude_'real estate'                     1000 non-null uint8
property_magnitude_car                               1000 non-null uint8
other_payment_plans_none                             1000 non-null uint8
other_payment_plans_stores                           1000 non-null uint8
housing_own                                          1000 non-null uint8
housing_rent                                         1000 non-null uint8
job_'unemp/unskilled non res'                        1000 non-null uint8
job_'unskilled resident'                             1000 non-null uint8
job_skilled                                          1000 non-null uint8
own_telephone_yes                                    1000 non-null uint8
foreign_worker_yes                                   1000 non-null uint8
class_good                                           1000 non-null uint8
dtypes: int64(7), uint8(39)
memory usage: 100.6+ KB
None
```

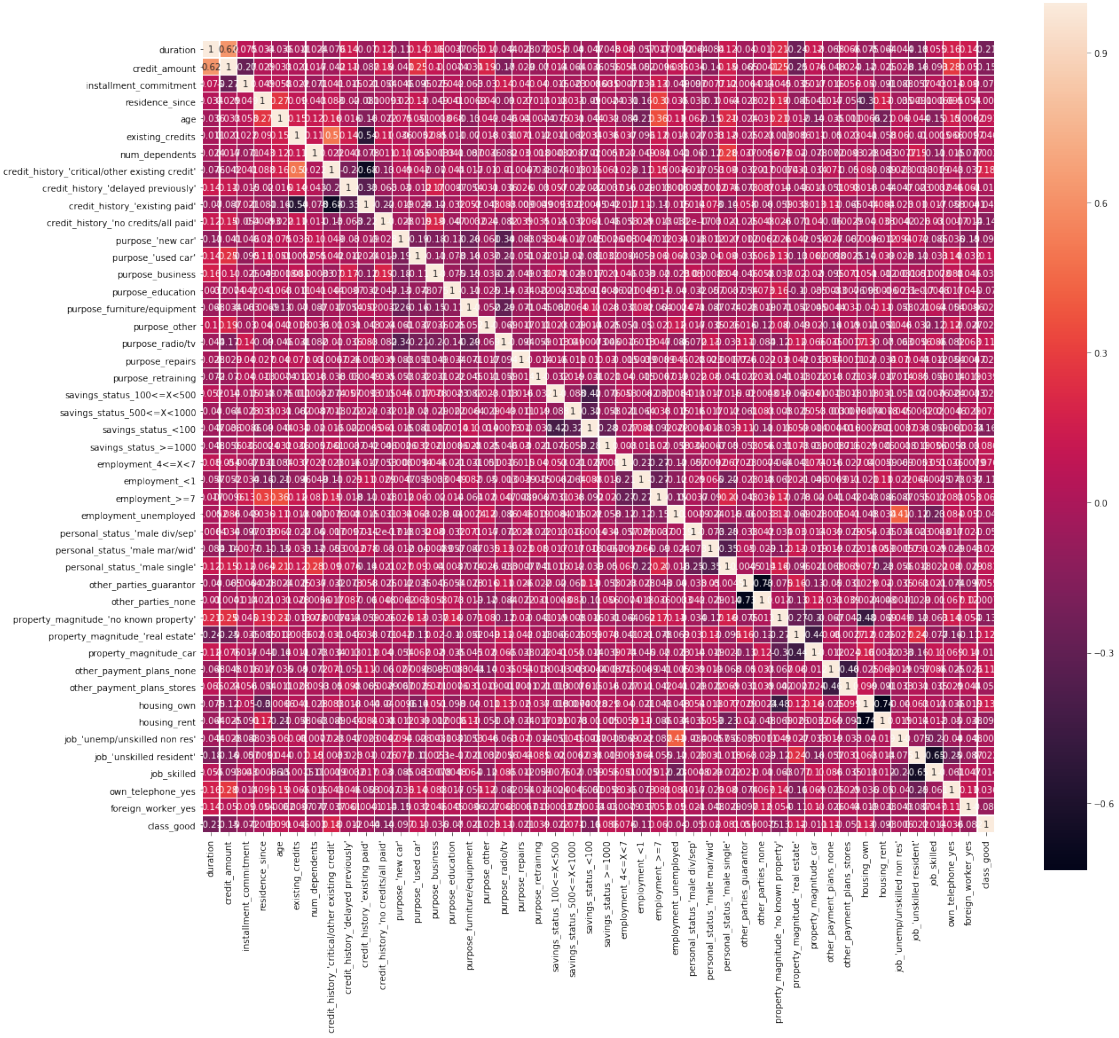
```
In [324]: new_colunms
```

```
Out[324]: ["credit_history_'critical/other existing credit'",
           "credit_history_'delayed previously'",
           "credit_history_'existing paid'",
           "credit_history_'no credits/all paid'",
           "purpose_'new car'",
           "purpose_'used car'",
           'purpose_business',
           'purpose_education',
           'purpose_furniture/equipment',
           'purpose_other',
```

```
 'purpose_radio/tv',
 'purpose_repairs',
 'purpose_retraining',
 'savings_status_100<=X<500',
 'savings_status_500<=X<1000',
 'savings_status_<100',
 'savings_status_>=1000',
 'employment_4<=X<7',
 'employment_<1',
 'employment_>=7',
 'employment_unemployed',
 "personal_status_'male div/sep'",
 "personal_status_'male mar/wid'",
 "personal_status_'male single'",
 'other_parties_guarantor',
 'other_parties_none',
 "property_magnitude_'no known property'",
 "property_magnitude_'real estate'",
 'property_magnitude_car',
 'other_payment_plans_none',
 'other_payment_plans_stores',
 'housing_own',
 'housing_rent',
 "job_'unemp/unskilled non res'",
 "job_'unskilled resident'",
 'job_skilled',
 'own_telephone_yes',
 'foreign_worker_yes',
 'class_good']
```

In [325]:
```
#Purpose to Dummies Variable
# df_credit = df_credit.merge(pd.get_dummies(df_credit.purpose, drop_first=True, pre
```

In [326]:
```
plt.figure(figsize=(20,18))
sns.heatmap(df_credit.astype(float).corr(),linewidths=0.1,vmax=1.0,
            square=True,  linecolor='white', annot=True)
plt.show()
```



In [327]:
```
from sklearn.model_selection import train_test_split, KFold, cross_val_score # to sp
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report,

from sklearn.model_selection import GridSearchCV

# Algorithmms models to be compared
from sklearn.ensemble import RandomForestClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
from sklearn.naive_bayes import GaussianNB
from sklearn.svm import SVC
```

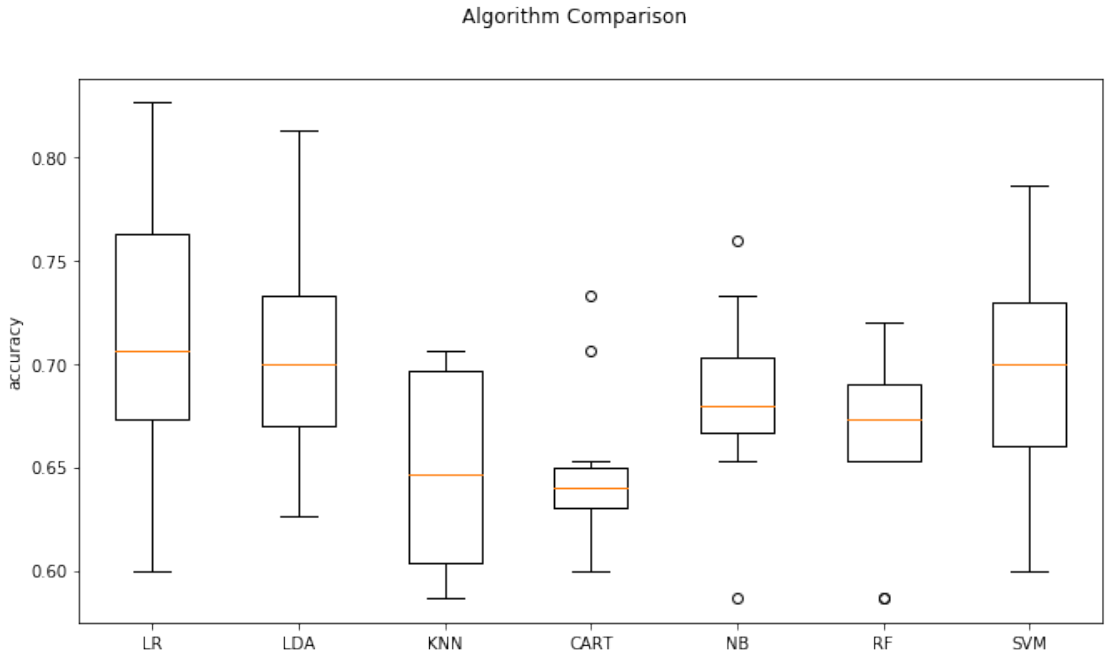
```
#Creating the X and y variables
X = df_credit.drop('class_good', 1).values
y = df_credit["class_good"].values
```

In [328]:
```
# Spliting X and y into train and test version
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.25, random_st
# to feed the random state
seed = 7

# prepare models
models = []
models.append(('LR', LogisticRegression()))
models.append(('LDA', LinearDiscriminantAnalysis()))
models.append(('KNN', KNeighborsClassifier()))
models.append(('CART', DecisionTreeClassifier()))
models.append(('NB', GaussianNB()))
models.append(('RF', RandomForestClassifier()))
models.append(('SVM', SVC(gamma='auto')))

# evaluate each model in turn
def pltFoldMethodsResult(scor):
    results = []
    names = []
    scoring = scor

    for name, model in models:
            kfold = KFold(n_splits=10, random_state=seed)
            cv_results = cross_val_score(model, X_train, y_train, cv=kfold, scoring=s
            results.append(cv_results)
            names.append(name)
            msg = "%s: %f (%f)" % (name, cv_results.mean(), cv_results.std())
            print(msg)

    # boxplot algorithm comparison
    fig = plt.figure(figsize=(11,6))
    fig.suptitle('Algorithm Comparison')
    ax = fig.add_subplot(111)
    plt.boxplot(results)
    ax.set_xticklabels(names)
    plt.ylabel(scor)
    plt.show()
```

In [329]:
```
pltFoldMethodsResult('accuracy')
```

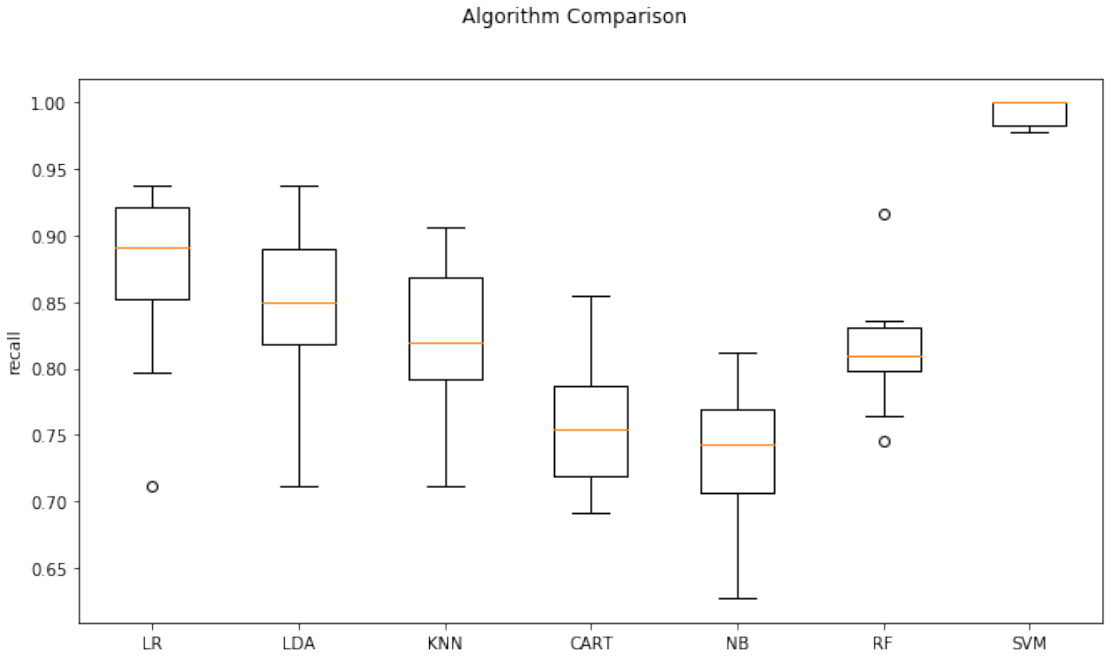
```
LR: 0.716000 (0.063400)
LDA: 0.705333 (0.057643)
```

```
KNN: 0.646667 (0.046667)
CART: 0.649333 (0.038667)
NB: 0.682667 (0.044542)
RF: 0.661333 (0.041825)
SVM: 0.694667 (0.051103)
```

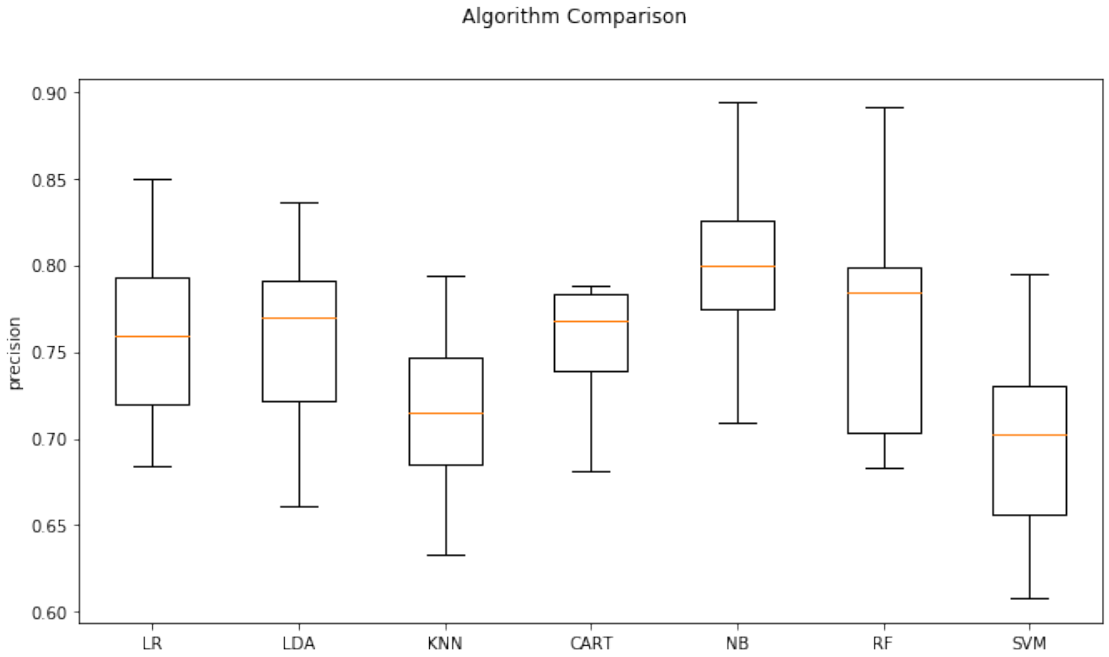



```
In [330]: pltFoldMethodsResult('recall')
```

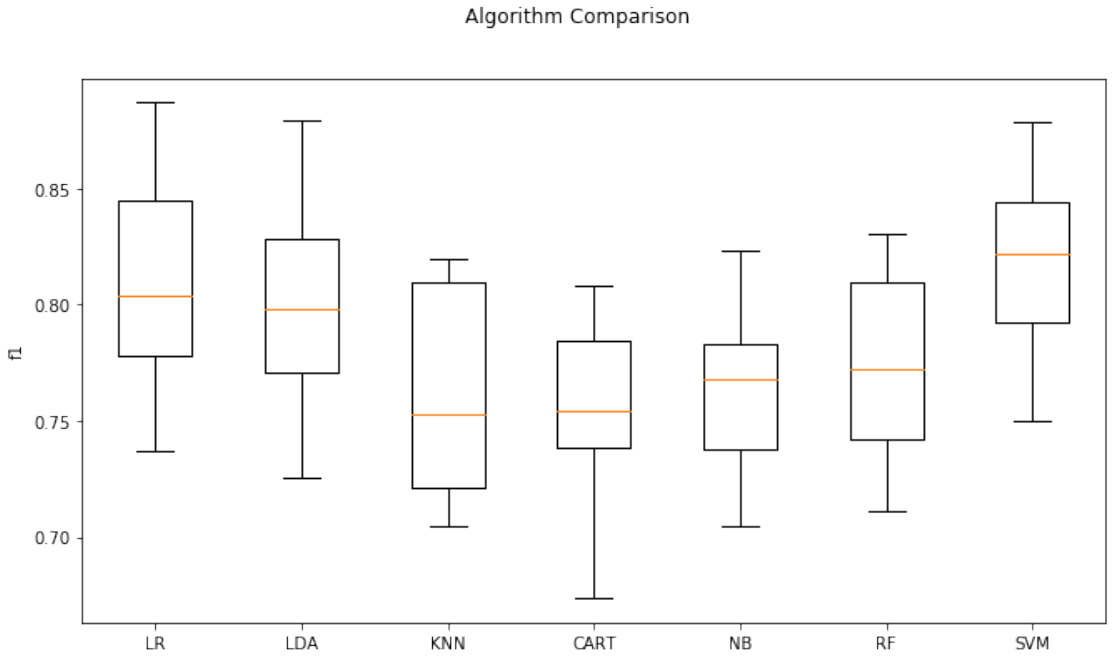
```
LR: 0.869930 (0.066475)
LDA: 0.850152 (0.062721)
KNN: 0.822758 (0.056701)
CART: 0.762427 (0.054170)
NB: 0.733099 (0.055276)
RF: 0.813792 (0.043599)
SVM: 0.992459 (0.009306)
```



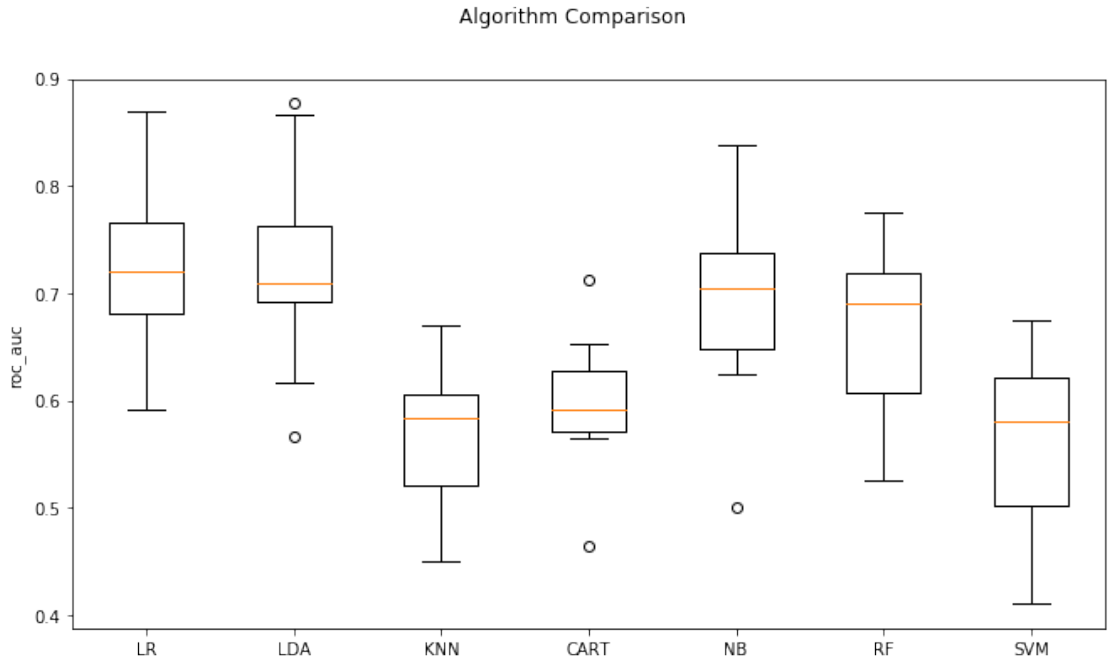

```
In [331]: pltFoldMethodsResult('precision')

LR: 0.758665 (0.048387)
LDA: 0.757335 (0.051154)
KNN: 0.712175 (0.049782)
CART: 0.752406 (0.038537)
NB: 0.797683 (0.051433)
RF: 0.764043 (0.063048)
SVM: 0.697352 (0.052462)
```




```
In [332]: pltFoldMethodsResult('f1')

LR: 0.808803 (0.045077)
LDA: 0.799266 (0.043606)
KNN: 0.762132 (0.042968)
CART: 0.754797 (0.038232)
NB: 0.761749 (0.035167)
RF: 0.772415 (0.041581)
SVM: 0.817950 (0.035744)
```




```
In [333]: pltFoldMethodsResult('roc_auc')

LR: 0.728378 (0.084313)
LDA: 0.724571 (0.092501)
KNN: 0.570450 (0.064943)
CART: 0.596835 (0.061732)
NB: 0.699443 (0.094225)
RF: 0.668016 (0.081963)
SVM: 0.566436 (0.081009)
```



In [334]:
```
from sklearn.utils import resample
from sklearn.metrics import roc_curve
# Criando o classificador logreg
GNB = GaussianNB()

# Fitting with train data
model = GNB.fit(X_train, y_train)
# Printing the Training Score
print("Training score data: ")
print(model.score(X_train, y_train))

y_pred = model.predict(X_test)

print(accuracy_score(y_test,y_pred))
print("\n")
print(confusion_matrix(y_test, y_pred))
print("\n")
print(classification_report(y_test, y_pred))

#Predicting proba
model.predict_proba(X_test)[:,1]
y_pred_prob = model.predict_proba(X_test)[:,1]

# Generate ROC curve values: fpr, tpr, thresholds
fpr, tpr, thresholds = roc_curve(y_test, y_pred_prob)
```

```
            # Plot ROC curve
            plt.plot([0, 1], [0, 1], 'k--')
            plt.plot(fpr, tpr)
            plt.xlabel('False Positive Rate')
            plt.ylabel('True Positive Rate')
            plt.title('ROC Curve')
            plt.show()

Training score data:
0.72
0.684


[[ 44  28]
 [ 51 127]]


              precision    recall  f1-score   support

           0       0.46      0.61      0.53        72
           1       0.82      0.71      0.76       178

avg / total       0.72      0.68      0.69       250
```