Cancer Analysis

This project is to do statistic and machine learning analyse. There are five .py files. **<CA\_Lib>**

**import** pandas **as** pd  
**import** matplotlib.pyplot **as** plt  
**from** sklearn.model\_selection **import** train\_test\_split  
**from** sklearn.linear\_model **import** LogisticRegression  
**from** sklearn.metrics **import** accuracy\_score  
**import** numpy **as** np  
**from** sklearn.preprocessing **import** StandardScaler  
**from** sklearn.linear\_model **import** Perceptron  
**from** sklearn.svm **import** SVC  
**from** sklearn.tree **import** DecisionTreeClassifier  
**from** pandas **import** DataFrame, Series  
**from** sklearn.ensemble **import** RandomForestClassifier  
**from** sklearn.neighbors **import** KNeighborsClassifier  
**from** sklearn.decomposition **import** PCA  
**from** sklearn.discriminant\_analysis **import** LinearDiscriminantAnalysis  
**from** sklearn.decomposition **import** KernelPCA  
**from** sklearn.preprocessing **import** LabelEncoder  
**from** sklearn.pipeline **import** Pipeline  
**from** sklearn.model\_selection **import** learning\_curve  
**from** sklearn.model\_selection **import** validation\_curve  
**from** sklearn.model\_selection **import** GridSearchCV  
**import** seaborn **as** sns  
  
columns **= ['radius\_mean'**, **'texture\_mean'**, **'perimeter\_mean'**,  
 **'area\_mean'**, **'smoothness\_mean'**, **'compactness\_mean'**,  
 **'concavity\_mean'**, **'concave points\_mean'**, **'symmetry\_mean'**,  
 **'fractal\_dimension\_mean']**

**<CancerAnalysis>**

**"""  
This is to do something about the cancer data.  
The data comes from Kaggle.  
We try to find some relationships inside the data, and  
build a predictive model, if possible.  
  
Statistic Analysis: plotting data distribution.  
Machine Learning Analysis:  
1, Training each classifier  
2, observe the accuracy of each classifier with the parameter varying  
3, plotting Learning curve and validation curve  
4, grid search optimization  
  
"""  
  
from** CA\_Lib **import \*  
from** StatisticalAnalysis **import** statistic\_analysis  
**from** MLAnalysis **import** ml\_analysis  
  
  
**def run():** print**('Loading Data >>> ......')** filename **= 'dataSet/cancer.csv'** df **=** pd.read\_csv**(**filename**)** X **=** df**[**columns**]** y **=** df**['diagnosis']** print**('Finished!')** print**('Starting Statistic Analysis >>> ......')** statistic\_analysis**(**X, y**)** print**('Finished!')** print**('Starting Machine Learning Analysis >>> ......')** ml\_analysis**(**X, y**)** print**('Finished!')  
  
  
def app():** run**()  
  
  
if** \_\_name\_\_ **== '\_\_main\_\_':** app**()**

**<Feature\_Selection>**

**from** sklearn.base **import** clone  
**from** itertools **import** combinations  
**import** numpy **as** np  
**from** sklearn.model\_selection **import** train\_test\_split  
**from** sklearn.metrics **import** accuracy\_score  
  
  
**class SBS(**object**):  
 def \_\_init\_\_(**self, estimator, k\_features,  
 scoring**=**accuracy\_score,  
 test\_size**=**0.25, random\_state**=**1**):** self.scoring **=** scoring  
 self.estimator **=** clone**(**estimator**)** self.k\_features **=** k\_features  
 self.test\_size **=** test\_size  
 self.random\_state **=** random\_state  
  
 **def fit(**self, X, y**):** X\_train, X\_test, y\_train, y\_test **=** \  
 train\_test\_split**(**X, y, test\_size**=**self.test\_size,  
 random\_state**=**self.random\_state**)** dim **=** X\_train.shape**[**1**]** self.indices\_ **=** tuple**(**range**(**dim**))** self.subsets\_ **= [**self.indices\_**]** score **=** self.\_calc\_score**(**X\_train, y\_train,  
 X\_test, y\_test, self.indices\_**)** self.scores\_ **= [**score**]  
 while** dim **>** self.k\_features**:** scores **= []** subsets **= []  
  
 for** p **in** combinations**(**self.indices\_, r**=**dim**-**1**):** score **=** self.\_calc\_score**(**X\_train, y\_train,  
 X\_test, y\_test, p**)** scores.append**(**score**)** subsets.append**(**p**)** best **=** np.argmax**(**scores**)** self.indices\_ **=** subsets**[**best**]** self.subsets\_.append**(**self.indices\_**)** dim **-=** 1  
 self.scores\_.append**(**scores**[**best**])** self.k\_score\_ **=** self.scores\_**[-**1**]  
  
 return** self  
  
 **def transform(**self, X**):  
 return** X**[:**, self.indices\_**]  
  
 def \_calc\_score(**self, X\_train, y\_train,  
 X\_test, y\_test, indices**):** self.estimator.fit**(**X\_train**[:**, indices**]**, y\_train**)** y\_pred **=** self.estimator.predict**(**X\_test**[:**, indices**])** score **=** self.scoring**(**y\_test, y\_pred**)  
  
 return** score

**<MLAnalysis>**

**from** CA\_Lib **import \*  
from** Feature\_Selection **import** SBS  
  
  
**def dimensionality\_reduction(**x\_train, x\_test, y\_train, option**):** x\_train\_, x\_test\_ **=** 0.0, 0.0  
  
 **if** option **== 'PCA':** pca **=** PCA**(**n\_components**=**7**)** x\_train\_ **=** pca.fit\_transform**(**x\_train**)** x\_test\_ **=** pca.transform**(**x\_test**)  
  
 elif** option **== 'KernelPCA':** pca **=** PCA**(**n\_components**=**7**)** x\_train\_ **=** pca.fit\_transform**(**x\_train**)** x\_test\_ **=** pca.transform**(**x\_test**)  
  
 elif** option **== 'LDA':** lda **=** LinearDiscriminantAnalysis**(**n\_components**=**8**)** x\_train\_ **=** lda.fit\_transform**(**x\_train, y\_train**)** x\_test\_ **=** lda.transform**(**x\_test**)  
  
 return** x\_train\_, x\_test\_  
  
  
**def standardize\_data(**x\_train, x\_test**):** sc **=** StandardScaler**()** x\_train\_std\_ **=** sc.fit\_transform**(**x\_train**)** x\_test\_std\_ **=** sc.transform**(**x\_test**)  
  
 return** x\_train\_std\_, x\_test\_std\_  
  
  
**def calculate\_accuracy(**x\_train, x\_test, y\_train, y\_test, clf**):** clf.fit**(**x\_train, y\_train**)** y\_pred **=** clf.predict**(**x\_test**)** accuracy\_ **=** accuracy\_score**(**y\_test, y\_pred**)  
  
 return** accuracy\_  
  
  
**def train(**x\_train\_std, x\_test\_std, y\_train, y\_test**):** param\_ **= {}** accuracy\_ **= {}** acc **= []** para **= []  
  
 for** eta **in** np.arange**(**0.000001, 0.4, 0.005**):** ppn **=** Perceptron**(**max\_iter**=**300, eta0**=**eta, tol**=**1e-5, random\_state**=**0**)** acc.append**(**calculate\_accuracy**(**x\_train\_std, x\_test\_std, y\_train, y\_test, ppn**))** para.append**(**eta**)** # acc.append(max(acc))  
 # para.append(para[acc.index(max(acc))])  
 param\_**['eta'] =** para  
 accuracy\_**['Perceptron'] =** acc  
  
 acc **= []** para **= []  
 for** c **in** range**(-**5, 5**):** C **=** 10**\*\*(**c**)** lr **=** LogisticRegression**(**C**=**C, random\_state**=**0, solver**='lbfgs')** acc.append**(**calculate\_accuracy**(**x\_train\_std, x\_test\_std, y\_train, y\_test, lr**))** para.append**(**c**)** # acc.append(max(acc))  
 # para.append(para[acc.index(max(acc))])  
 param\_**['C\_LR'] =** para  
 accuracy\_**['Logistic\_Regression'] =** acc  
  
 acc **= []** para **= []  
 for** c **in** range**(-**5, 5**):** C **=** 10**\*\*(**c**)** svm **=** SVC**(**kernel**='linear'**, C**=**C, random\_state**=**0**)** acc.append**(**calculate\_accuracy**(**x\_train\_std, x\_test\_std, y\_train, y\_test, svm**))** para.append**(**c**)** # acc.append(max(acc))  
 # para.append(para[acc.index(max(acc))])  
 param\_**['C\_SVMLin'] =** para  
 accuracy\_**['SVM\_Linear'] =** acc  
  
 acc **= []** para **= []  
 for** c **in** range**(-**5, 5**):** C **=** 10**\*\*(**c**)** svm **=** SVC**(**kernel**='rbf'**, random\_state**=**0, gamma**=**0.05, C**=**C**)** acc.append**(**calculate\_accuracy**(**x\_train\_std, x\_test\_std, y\_train, y\_test, svm**))** para.append**(**c**)** # acc.append(max(acc))  
 # para.append(para[acc.index(max(acc))])  
 param\_**['C\_SVMrbf'] =** para  
 accuracy\_**['SVM\_RBF'] =** acc  
  
 acc **= []** para **= []  
 for** c **in** range**(**1, 10**):** tree **=** DecisionTreeClassifier**(**criterion**='entropy'**, max\_depth**=**c, random\_state**=**0**)** acc.append**(**calculate\_accuracy**(**x\_train\_std, x\_test\_std, y\_train, y\_test, tree**))** para.append**(**c**)** # acc.append(max(acc))  
 # para.append(para[acc.index(max(acc))])  
 param\_**['maxdepth'] =** para  
 accuracy\_**['Decision\_Tree'] =** acc  
  
 acc **= []** para **= []  
  
 for** c **in** range**(**10, 200, 10**):** RF **=** RandomForestClassifier**(**criterion**='entropy'**, n\_estimators**=**c, random\_state**=**0**)** acc.append**(**calculate\_accuracy**(**x\_train\_std, x\_test\_std, y\_train, y\_test, RF**))** para.append**(**c**)** # acc.append(max(acc))  
 # para.append(para[acc.index(max(acc))])  
 param\_**['n\_estimator'] =** para  
 accuracy\_**['Random\_Forest'] =** acc  
  
 acc **= []** para **= []  
  
 for** c **in** range**(**1, 20**):** KNN **=** KNeighborsClassifier**(**n\_neighbors**=**c, p**=**2, metric**='minkowski')** acc.append**(**calculate\_accuracy**(**x\_train\_std, x\_test\_std, y\_train, y\_test, KNN**))** para.append**(**c**)** # acc.append(max(acc))  
 # para.append(para[acc.index(max(acc))])  
 param\_**['n\_neighbor'] =** para  
 accuracy\_**['KNN'] =** acc  
  
 **return** param\_, accuracy\_  
  
  
**def param\_accuracy\_plot(**param, accuracy**):** fig **=** plt.figure**()** ith\_fig **=** 0  
  
 **for** key1, key2 **in** zip**(**param, accuracy**):** ith\_fig **+=** 1  
 ax **=** fig.add\_subplot**(**3, 4, ith\_fig**)** ax.plot**(**param**[**key1**]**, accuracy**[**key2**])** plt.title**(**key2**)** plt.xlabel**(**key1**)** plt.ylabel**('accuracy')** # plt.draw()  
 # plt.pause(0.5)  
  
 plt.subplots\_adjust**(**wspace**=**0.5, hspace**=**0.5**)** plt.show**()  
  
  
def sequential\_feature\_selection(**x\_train\_std, y\_train, param**):** ppn **=** Perceptron**(**max\_iter**=**300, eta0**=**0.01, tol**=**1e-5, random\_state**=**0**)** lr **=** LogisticRegression**(**C**=**1, random\_state**=**0, solver**='lbfgs')** svm\_lin **=** SVC**(**kernel**='linear'**, C**=**10, random\_state**=**0**)** svm\_rbf **=** SVC**(**kernel**='rbf'**, random\_state**=**0, gamma**=**0.05, C**=**10**)** tree **=** DecisionTreeClassifier**(**criterion**='entropy'**, max\_depth**=**3, random\_state**=**0**)** rf **=** RandomForestClassifier**(**criterion**='entropy'**, n\_estimators**=**10, random\_state**=**0**)** knn **=** KNeighborsClassifier**(**n\_neighbors**=**4, p**=**2, metric**='minkowski')** clfs **= [**ppn, lr, svm\_lin, svm\_rbf, tree, rf, knn**]** names **= ['Perceptron'**, **'Logistic Regression'**, **'SVM\_Linear'**, **'SVM\_rbf'**, **'Decision Tree'**, **'Random Forest'**, **'KNN']  
  
 for** clf **in** clfs**:** sbs **=** SBS**(**clf, k\_features**=**1**)** sbs.fit**(**x\_train\_std, y\_train**)** k\_feat **= [**len**(**k**) for** k **in** sbs.subsets\_**]** plt.plot**(**k\_feat, sbs.scores\_, marker**='o')** plt.ylabel**('Accuracy')** plt.xlabel**('Number of features')** # plt.draw()  
 # plt.pause(0.5)  
  
 plt.legend**(**names**)** plt.show**()  
  
  
def feature\_importance(**x\_train, y\_train**):** feat\_labels **=** columns  
 forest **=** RandomForestClassifier**(**n\_estimators**=**10000, random\_state**=**0, n\_jobs**=-**1**)** forest.fit**(**x\_train, y\_train**)** importances **=** forest.feature\_importances\_  
 indices **=** np.argsort**(**importances**)[::-**1**]  
 for** f **in** range**(**x\_train.shape**[**1**]):** print**("%2d) %-\*s %f" % (**f **+** 1, 30,  
 feat\_labels**[**indices**[**f**]]**,  
 importances**[**indices**[**f**]]))  
  
  
def classifier\_learning\_curve(**x\_train, y\_train**):** pipe\_ppn **=** Pipeline**([('sc'**, StandardScaler**())**,  
 **('clf'**, Perceptron**(**max\_iter**=**300, eta0**=**0.01, tol**=**1e-5, random\_state**=**0**))])** pipe\_lr **=** Pipeline**([('sc'**, StandardScaler**())**,  
 **('clf'**, LogisticRegression**(**penalty**='l2'**, random\_state**=**0, solver**='lbfgs'))])** pipe\_svm\_lin **=** Pipeline**([('sc'**, StandardScaler**())**,  
 **('clf'**, SVC**(**kernel**='linear'**, C**=**10, random\_state**=**0**))])** pipe\_svm\_rbf **=** Pipeline**([('sc'**, StandardScaler**())**,  
 **('clf'**, SVC**(**kernel**='rbf'**, random\_state**=**0, gamma**=**0.05, C**=**10**))])** pipe\_tree **=** Pipeline**([('sc'**, StandardScaler**())**,  
 **('clf'**, DecisionTreeClassifier**(**criterion**='entropy'**, max\_depth**=**3, random\_state**=**0**))])** pipe\_rf **=** Pipeline**([('sc'**, StandardScaler**())**,  
 **('clf'**, RandomForestClassifier**(**criterion**='entropy'**, n\_estimators**=**20, random\_state**=**0**))])** pipe\_knn **=** Pipeline**([('sc'**, StandardScaler**())**,  
 **('clf'**, KNeighborsClassifier**(**n\_neighbors**=**4, p**=**2, metric**='minkowski'))])** clfs **= [**pipe\_ppn, pipe\_lr, pipe\_svm\_lin, pipe\_svm\_rbf, pipe\_tree, pipe\_rf, pipe\_knn**]** names **= ['Perceptron'**, **'Logistic Regression'**, **'SVM\_Linear'**, **'SVM\_rbf'**, **'Decision Tree'**, **'Random Forest'**, **'KNN']** fig **=** plt.figure**()** ith\_fig **=** 0  
  
 **for** name, clf **in** zip**(**names, clfs**):** ith\_fig **+=** 1  
 ax **=** fig.add\_subplot**(**3, 4, ith\_fig**)** learning\_curve\_plot**(**x\_train, y\_train, clf, name**)** plt.subplots\_adjust**(**wspace**=**0.5, hspace**=**0.5**)** plt.show**()  
  
  
def learning\_curve\_plot(**x\_train, y\_train, clf, title**):** train\_sizes, train\_scores, test\_scores **=** learning\_curve**(**estimator**=**clf,  
 X**=**x\_train, y**=**y\_train,  
 train\_sizes**=**np.linspace**(**0.1, 1.0, 10**)**,  
 cv**=**10, n\_jobs**=-**1**)** train\_mean **=** np.mean**(**train\_scores, axis**=**1**)** train\_std **=** np.std**(**train\_scores, axis**=**1**)** test\_mean **=** np.mean**(**test\_scores, axis**=**1**)** test\_std **=** np.std**(**test\_scores, axis**=**1**)** plt.plot**(**train\_sizes, train\_mean,  
 color**='blue'**, marker**='o'**,  
 markersize**=**5, label**='training accuracy')** plt.fill\_between**(**train\_sizes,  
 train\_mean **+** train\_std,  
 train\_mean **-** train\_std,  
 alpha**=**0.15, color**='blue')** plt.plot**(**train\_sizes, test\_mean,  
 color**='green'**, linestyle**='--'**,  
 marker**='s'**, markersize**=**5,  
 label**='validation accuracy')** plt.fill\_between**(**train\_sizes,  
 test\_mean **+** test\_std,  
 test\_mean **-** test\_std,  
 alpha**=**0.15, color**='green')** plt.grid**()** plt.title**(**title**)** plt.xlabel**('Number of training samples')** plt.ylabel**('Accuracy')** plt.legend**()** plt.ylim**([**0.4, 1.0**])  
  
  
def classifier\_validation\_curve(**x\_train, y\_train**):** pipe\_ppn **=** Pipeline**([('sc'**, StandardScaler**())**,  
 **('clf'**, Perceptron**(**max\_iter**=**300, eta0**=**0.01, tol**=**1e-5, random\_state**=**0**))])** pipe\_lr **=** Pipeline**([('sc'**, StandardScaler**())**,  
 **('clf'**, LogisticRegression**(**penalty**='l2'**, random\_state**=**0, solver**='lbfgs'))])** pipe\_svm\_lin **=** Pipeline**([('sc'**, StandardScaler**())**,  
 **('clf'**, SVC**(**kernel**='linear'**, C**=**10, random\_state**=**0**))])** pipe\_svm\_rbf **=** Pipeline**([('sc'**, StandardScaler**())**,  
 **('clf'**, SVC**(**kernel**='rbf'**, random\_state**=**0, gamma**=**0.05, C**=**10**))])** pipe\_tree **=** Pipeline**([('sc'**, StandardScaler**())**,  
 **('clf'**, DecisionTreeClassifier**(**criterion**='entropy'**, max\_depth**=**3, random\_state**=**0**))])** pipe\_rf **=** Pipeline**([('sc'**, StandardScaler**())**,  
 **('clf'**, RandomForestClassifier**(**criterion**='entropy'**, n\_estimators**=**20, random\_state**=**0**))])** pipe\_knn **=** Pipeline**([('sc'**, StandardScaler**())**,  
 **('clf'**, KNeighborsClassifier**(**n\_neighbors**=**4, p**=**2, metric**='minkowski'))])** clfs **= [**pipe\_lr, pipe\_svm\_lin, pipe\_svm\_rbf**]** names **= ['Logistic Regression'**, **'SVM\_Linear'**, **'SVM\_rbf']** fig **=** plt.figure**()** ith\_fig **=** 0  
  
 **for** name, clf **in** zip**(**names, clfs**):** ith\_fig **+=** 1  
 ax **=** fig.add\_subplot**(**1, 3, ith\_fig**)** validation\_curve\_plot**(**x\_train, y\_train, clf, name**)** plt.subplots\_adjust**(**wspace**=**0.5, hspace**=**0.5**)** plt.show**()  
  
  
def validation\_curve\_plot(**x\_train, y\_train, clf, title**):** param\_range **= [**0.001, 0.01, 0.1, 1.0, 10.0, 100.0**]** train\_scores, test\_scores **=** validation\_curve**(**estimator**=**clf,  
 X**=**x\_train, y**=**y\_train,  
 param\_name**='clf\_\_C'**,  
 param\_range**=**param\_range,  
 cv**=**10**)** train\_mean **=** np.mean**(**train\_scores, axis**=**1**)** train\_std **=** np.std**(**train\_scores, axis**=**1**)** test\_mean **=** np.mean**(**test\_scores, axis**=**1**)** test\_std **=** np.std**(**test\_scores, axis**=**1**)** plt.plot**(**param\_range, train\_mean,  
 color**='blue'**, marker**='o'**,  
 markersize**=**5,  
 label**='training accuracy')** plt.fill\_between**(**param\_range, train\_mean **+** train\_std,  
 train\_mean **-** train\_std, alpha**=**0.15,  
 color**='blue')** plt.plot**(**param\_range, test\_mean,  
 color**='green'**, linestyle**='--'**,  
 marker**='s'**, markersize**=**5,  
 label**='validation accuracy')** plt.fill\_between**(**param\_range,  
 test\_mean **+** test\_std,  
 test\_mean **-** test\_std,  
 alpha**=**0.15, color**='green')** plt.grid**()** plt.xscale**('log')** plt.legend**(**loc**='lower right')** plt.xlabel**('Parameter C')** plt.ylabel**('Accuracy')** plt.title**(**title**)** plt.ylim**([**0.4, 1.0**])  
  
  
def grid\_search\_optimization(**x\_train, y\_train**):** pipe\_svc **=** Pipeline**([('sc'**, StandardScaler**())**,  
 **('clf'**, SVC**(**random\_state**=**0**))])** param\_range **= [**0.0001, 0.001, 0.01, 0.1, 1.0, 10.0, 100.0, 1000.0**]** param\_grid **= [{'clf\_\_C':** param\_range,  
 **'clf\_\_kernel': ['linear']}**,  
 **{'clf\_\_C':** param\_range,  
 **'clf\_\_gamma':** param\_range,  
 **'clf\_\_kernel': ['rbf']}]** gs **=** GridSearchCV**(**estimator**=**pipe\_svc,  
 param\_grid**=**param\_grid,  
 scoring**='accuracy'**,  
 cv**=**10,  
 n\_jobs**=-**1**)** gs **=** gs.fit**(**x\_train, y\_train**)** print**(**gs.best\_params\_**)  
  
  
def ml\_analysis(**X, y**):** print**('Encoding Label >>> ......')** # preprocessing  
 le **=** LabelEncoder**()** y **=** le.fit\_transform**(**y**)** x\_train, x\_test, y\_train, y\_test **=** train\_test\_split**(**X, y, test\_size**=**0.3,  
 random\_state**=**0**)** # dimensionality reduction  
 # print('Dimensionality Reduction')  
 # x\_train, x\_test = dimensionality\_reduction(x\_train, x\_test, None, 'PCA')  
 print**('Standardize Data >>> ......')** x\_train\_std, x\_test\_std **=** standardize\_data**(**x\_train, x\_test**)** # train data using different algorithms  
 print**('Plotting the test accuracy with the parameter varying >>> ......')** print**('Classifiers involved:'  
 'Perceptron, Logistic Regression, SVM\_Linear, SVM\_rbf, Decision Tree, Random Forest, KNN')** param, accuracy **=** train**(**x\_train\_std, x\_test\_std, y\_train, y\_test**)** param\_accuracy\_plot**(**param, accuracy**)** # Sequential feature selection algorithms  
 print**('Plotting the accuracy with the number of features varying >>> ......')** print**('Classifiers involved:'  
 'Perceptron, Logistic Regression, SVM\_Linear, SVM\_rbf, Decision Tree, Random Forest, KNN')** sequential\_feature\_selection**(**x\_train\_std, y\_train, param**)** # importances of features  
 print**('Feature Importance >>> ......')** feature\_importance**(**x\_train, y\_train**)** # learning and validation curves  
 print**('Plotting Learning Curve >>> ......')** print**('Classifiers involved:'  
 'Perceptron, Logistic Regression, SVM\_Linear, SVM\_rbf, Decision Tree, Random Forest, KNN')** classifier\_learning\_curve**(**x\_train, y\_train**)** # validation curve  
 print**('Plotting Learning Curve >>> ......')** print**('Classifiers involved:'  
 'Logistic Regression, SVM\_Linear, SVM\_rbf')** classifier\_validation\_curve**(**x\_train, y\_train**)** # grid search  
 print**('Starting Grid Search Optimization >>> ......')** print**('Classifiers involved:'  
 'SVM\_Linear, SVM\_rbf')** grid\_search\_optimization**(**x\_train, y\_train**)**

**<StatisticalAnalysis>**

**from** CA\_Lib **import \*  
  
  
def statistic\_analysis(**X, y**):** data **=** X  
 df **= {}  
  
 for** label **in** np.unique**(**y**):** df**[**label**] =** data**[**y **==** label**]** print**('Plotting data distribution')** cols **=** 4  
 rows **=** np.ceil**(**len**(**columns**) /** cols**)** # plot data distribution  
 fig **=** plt.figure**()  
 for** i, column **in** enumerate**(**columns**):** ax **=** fig.add\_subplot**(**rows, cols, i **+** 1**)  
 for** key **in** df**:** ax.hist**(**df**[**key**][**column**]**, bins**=**30, label**=**key**)** plt.xlabel**(**column**)** plt.legend**()** plt.subplots\_adjust**(**wspace**=**0.5, hspace**=**0.5**)** plt.show**()** # print('Plotting Correlation Map')  
 # # plot correlation  
 # cm = np.corrcoef(data.values.T)  
 # sns.set(font\_scale=0.8)  
 # \_ = sns.heatmap(cm,  
 # cbar=True,  
 # annot=True,  
 # square=True,  
 # fmt='.2f',  
 # annot\_kws={'size': 9},  
 # yticklabels=columns,  
 # xticklabels=columns)  
 # plt.title('Correlation')  
 # plt.show()

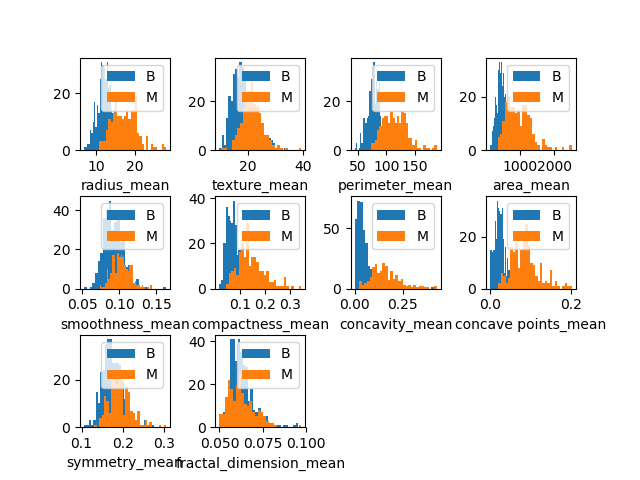
The Result

Loading Data >>> ......

Finished!

Starting Statistic Analysis >>> ......

Plotting data distribution



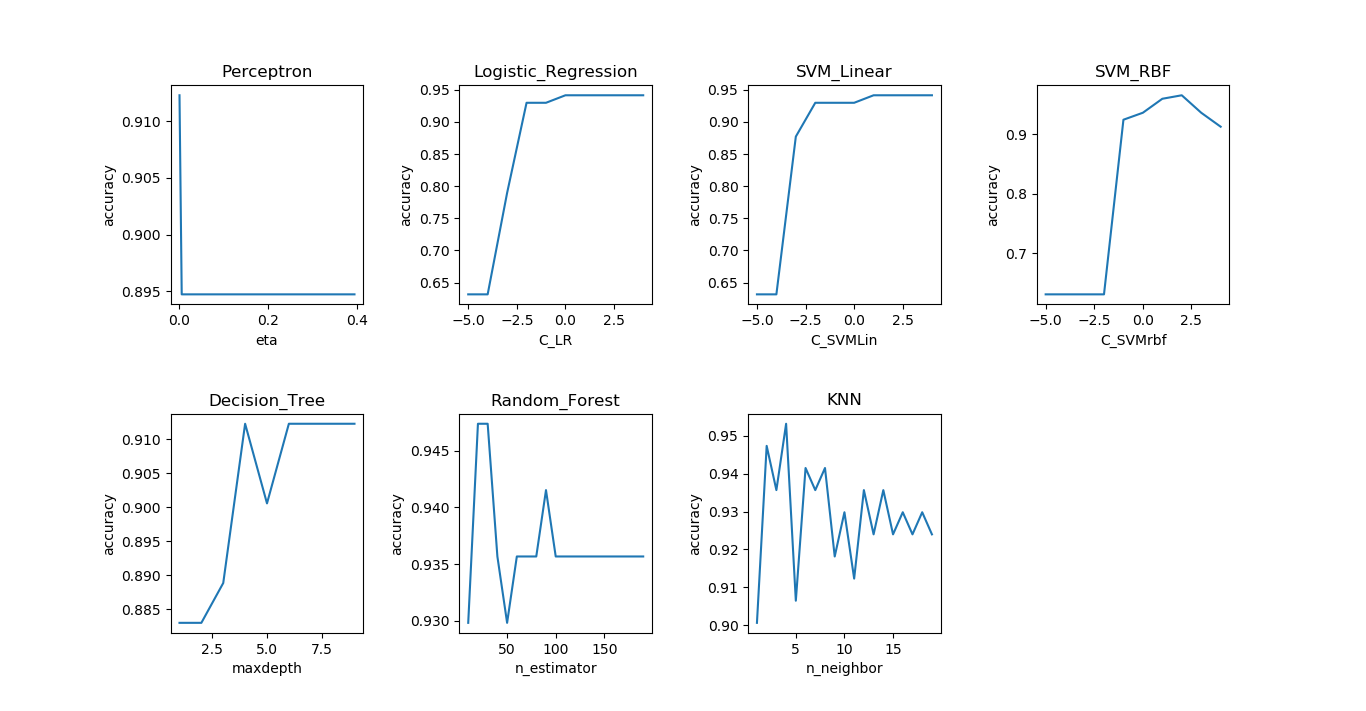
Finished!

Starting Machine Learning Analysis >>> ......

Encoding Label >>> ......

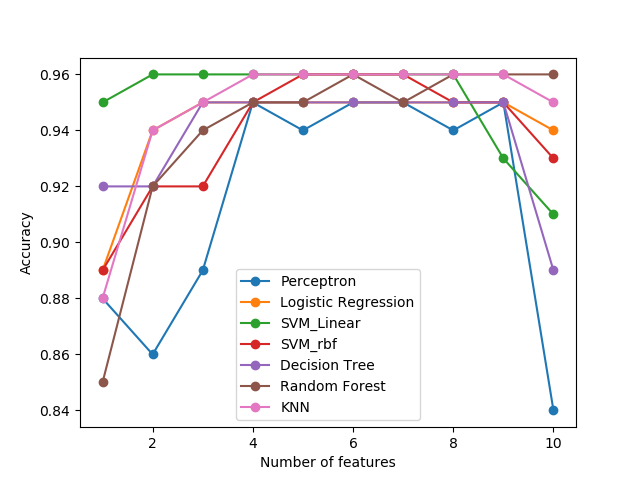
Standardize Data >>> ......

Plotting the test accuracy with the parameter varying >>> ......



Classifiers involved:Perceptron, Logistic Regression, SVM\_Linear, SVM\_rbf, Decision Tree, Random Forest, KNN

Plotting the accuracy with the number of features varying >>> ......



Classifiers involved:Perceptron, Logistic Regression, SVM\_Linear, SVM\_rbf, Decision Tree, Random Forest, KNN

Feature Importance >>> ......

1) concave points\_mean 0.292361

2) concavity\_mean 0.177156

3) perimeter\_mean 0.143749

4) area\_mean 0.115876

5) radius\_mean 0.096895

6) texture\_mean 0.059167

7) compactness\_mean 0.051360

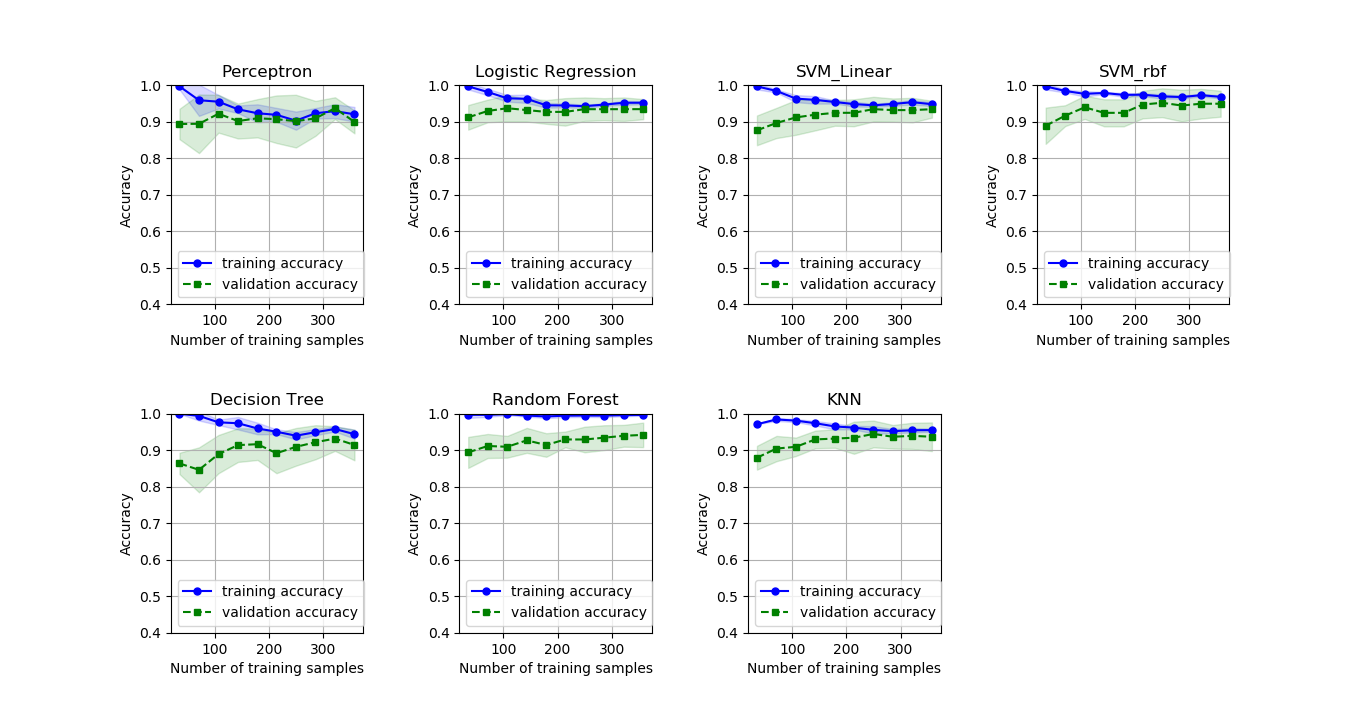
8) smoothness\_mean 0.027144

9) symmetry\_mean 0.019209

10) fractal\_dimension\_mean 0.017083

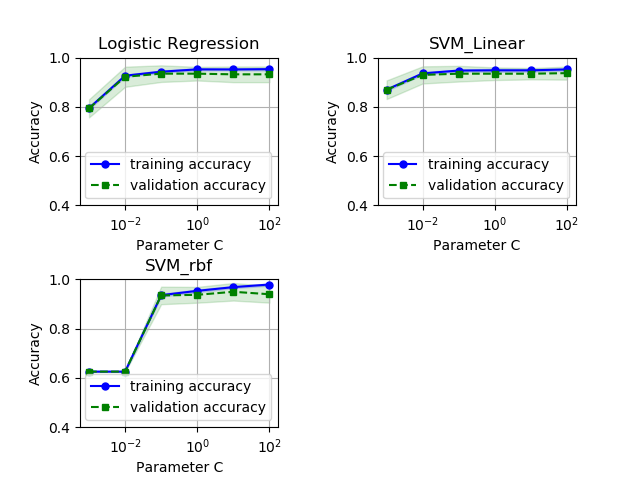
Plotting Learning Curve >>> ......

Classifiers involved:Perceptron, Logistic Regression, SVM\_Linear, SVM\_rbf, Decision Tree, Random Forest, KNN



Plotting Learning Curve >>> ......

Classifiers involved:Logistic Regression, SVM\_Linear, SVM\_rbf



Starting Grid Search Optimization >>> ......

Classifiers involved:SVM\_Linear, SVM\_rbf

{'clf\_\_C': 100.0, 'clf\_\_gamma': 0.01, 'clf\_\_kernel': 'rbf'}

Finished!