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
```

```python
"""

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>

```python
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):
```

```python
    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):
```

```python
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, 
```

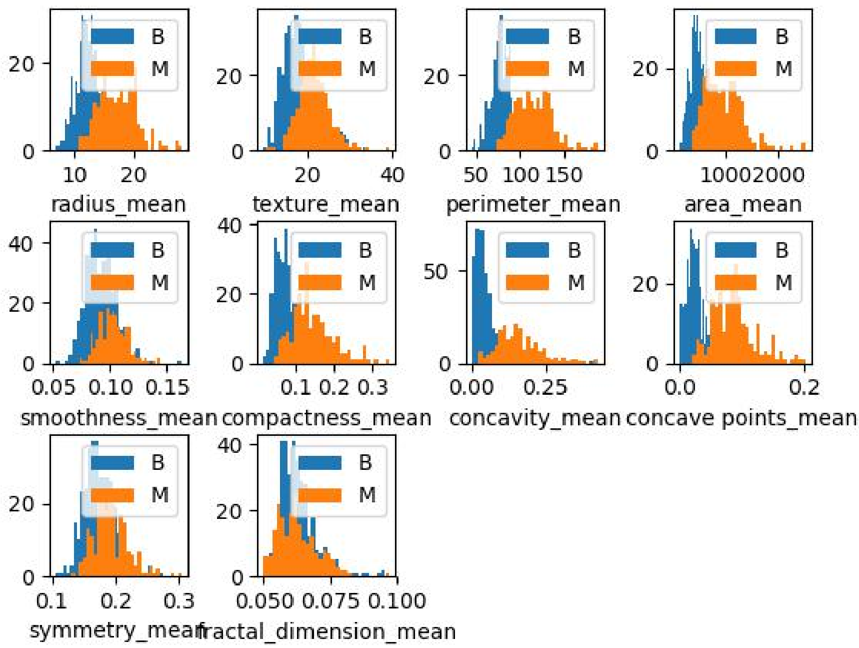
```
Random Forest, KNN')
    classifier_learning_curve(x_train, y_train)

    # 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>

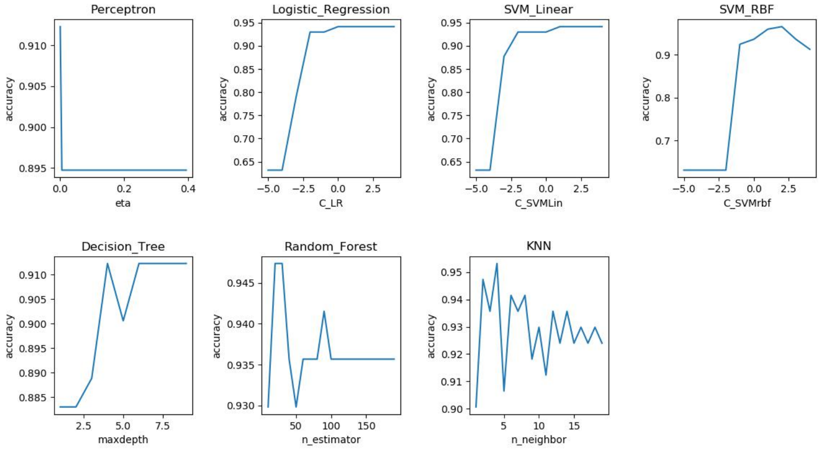
```
from CA_Lib import *


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

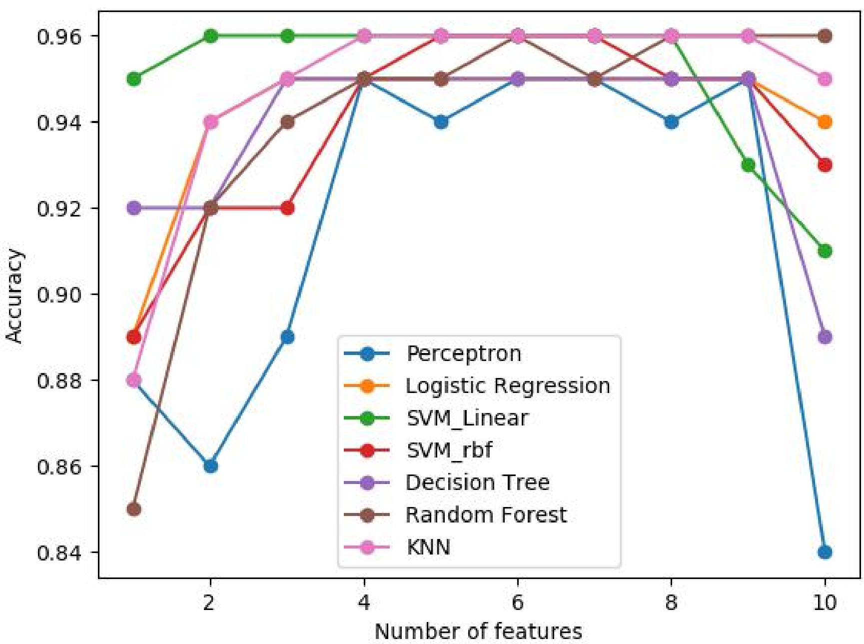


Finished!
Starting Machine Learning Analysis >>> ......
Encoding Label >>> ......
Standardize Data >>> ......
Plotting the test accuracy with the parameter varying >>> ......



Classifiers involved:Perceptron, Logistic Regression, SVM_Linear, SVM_rbf, Decision Tree, Random Forest, KNN

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

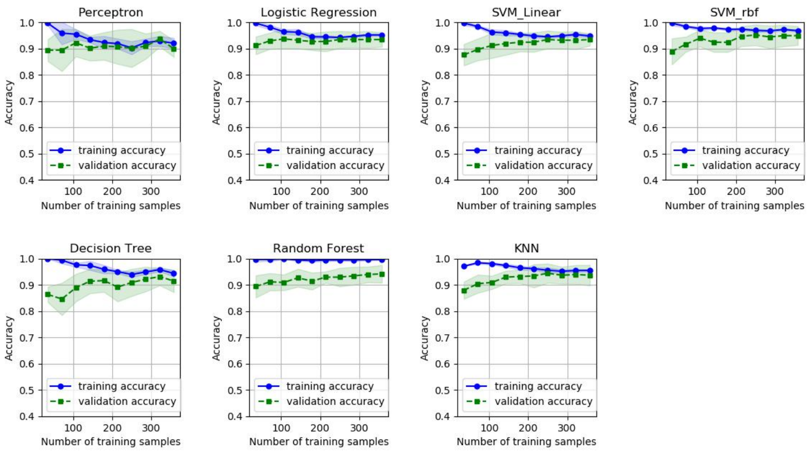


Classifiers involved:Perceptron, Logistic Regression, SVM_Linear, SVM_rbf, Decision Tree, Random Forest, KNN

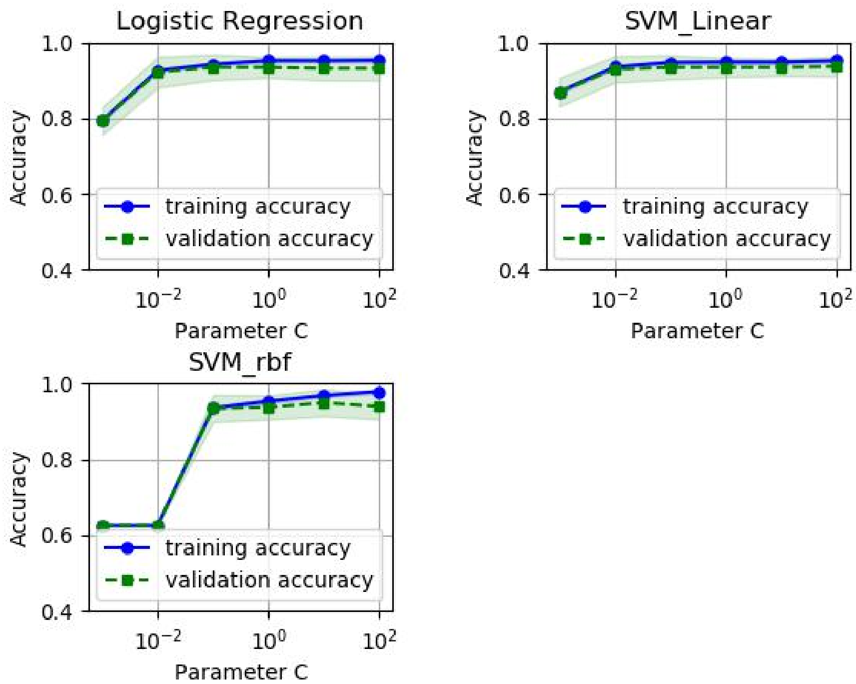
Feature Importance >>> ......

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!