HDS5230 HW11 - Miao Cai

April 2, 2019

1 HDS 5230 Homework 11

Author: Miao Cai

1.1 Read data

In [12]:
```
import os
import sys
import numpy as np
import pandas as pd
import matplotlib
import matplotlib.pyplot as plt
from patsy import dmatrices
from sklearn.metrics import confusion_matrix
import sklearn
from sklearn import datasets
#import seaborn as sns
```

In [13]:
```
data = pd.read_csv('wdbc.data',header=None,names = ['id_number', 'diagnosis', 'radius_
          'texture_mean', 'perimeter_mean', 'area_mean',
          'smoothness_mean', 'compactness_mean',
          'concavity_mean','concave_points_mean',
          'symmetry_mean', 'fractal_dimension_mean',
          'radius_se', 'texture_se', 'perimeter_se',
          'area_se', 'smoothness_se', 'compactness_se',
          'concavity_se', 'concave_points_se',
          'symmetry_se', 'fractal_dimension_se',
          'radius_worst', 'texture_worst',
          'perimeter_worst', 'area_worst',
          'smoothness_worst', 'compactness_worst',
          'concavity_worst', 'concave_points_worst',
          'symmetry_worst', 'fractal_dimension_worst'])
data[:5]
```

Out[13]:
```
   id_number diagnosis  radius_mean  texture_mean  perimeter_mean  area_mean  \
0     842302         M        17.99         10.38          122.80     1001.0
1     842517         M        20.57         17.77          132.90     1326.0
2   84300903         M        19.69         21.25          130.00     1203.0
```

```
3   84348301         M        11.42         20.38            77.58      386.1
4   84358402         M        20.29         14.34           135.10     1297.0

   smoothness_mean  compactness_mean  concavity_mean  concave_points_mean  \
0          0.11840           0.27760          0.3001              0.14710
1          0.08474           0.07864          0.0869              0.07017
2          0.10960           0.15990          0.1974              0.12790
3          0.14250           0.28390          0.2414              0.10520
4          0.10030           0.13280          0.1980              0.10430

            ...             radius_worst  texture_worst  perimeter_worst  \
0           ...                    25.38          17.33           184.60
1           ...                    24.99          23.41           158.80
2           ...                    23.57          25.53           152.50
3           ...                    14.91          26.50            98.87
4           ...                    22.54          16.67           152.20

   area_worst  smoothness_worst  compactness_worst  concavity_worst  \
0      2019.0            0.1622             0.6656           0.7119
1      1956.0            0.1238             0.1866           0.2416
2      1709.0            0.1444             0.4245           0.4504
3       567.7            0.2098             0.8663           0.6869
4      1575.0            0.1374             0.2050           0.4000

   concave_points_worst  symmetry_worst  fractal_dimension_worst
0                0.2654          0.4601                  0.11890
1                0.1860          0.2750                  0.08902
2                0.2430          0.3613                  0.08758
3                0.2575          0.6638                  0.17300
4                0.1625          0.2364                  0.07678

[5 rows x 32 columns]
```

In [14]:
```
data['diagnosis'].replace("M", 0, inplace=True)
data['diagnosis'].replace("B", 1, inplace=True)
vars = ['radius_mean','texture_mean','perimeter_mean', 'area_mean','smoothness_mean',
       'compactness_mean','concavity_mean','concave_points_mean','symmetry_mean','fra
formula = "diagnosis ~ "  + " + ".join(vars)
Y,X = dmatrices(formula, data)
```

1.2 Establish 20% test sample

In [15]:
```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = \
    train_test_split(X,
                     np.ravel(Y), # prevents dimensionality error later!
                     test_size=0.2,
                     random_state=0)
```

```
X_test.shape
```

Out[15]: (114, 11)

1.2.1 Logistic regression without regularization

```
In [16]: ## import linear model
         from sklearn import linear_model
         ## Define model parameters
         ## can implement penalties, but check docs for appropriate solver
         clf = linear_model.LogisticRegression(fit_intercept=True, # already have the intercep
                                               solver='liblinear') # could change to lbfgs!
         ## fit model using data with .fit
         clf.fit(X_train,y_train)
         clf.coef_
         clf.coef_.shape
```

```
Out[16]: (1, 11)
```

```
In [17]: ## mean accuracy
         clf.score(X_train,y_train)
```

```
Out[17]: array(0.90769231)
```

```
In [18]: ## get confusion matrix
         from sklearn.metrics import confusion_matrix
         confusion_matrix(y_train,
                          clf.predict(X_train))
```

```
Out[18]: array([[135,  30],
                [ 12, 278]])
```

```
In [19]: # Get kappa
         sklearn.metrics.cohen_kappa_score(y_train,
                                           clf.predict(X_train))
```

```
Out[19]: 0.7955056179775281
```

```
In [20]: ## Create dict to store all these results:
         result_scores = {}
         ## Score the Model on Training and Testing Set
         result_scores['Logistic'] = \
                     (sklearn.metrics.accuracy_score(y_train,clf.predict(X_train)),
                      sklearn.metrics.accuracy_score(y_test,clf.predict(X_test)))
```

```
In [21]: ## Create Function to Print Results
         def get_results(x1):
             print("\n{0:20}   {1:4}    {2:4}".format('Model','Train','Test'))
             print('--------------------------------------------')
             for i in x1.keys():
                 print("{0:20}   {1:<6.4}   {2:<6.4}".format(i,x1[i][0],x1[i][1]))
         get_results(result_scores)
```

```
Model                   Train    Test
-------------------------------------------
Logistic                0.9077   0.9211
```

1.2.2 Null model

```
In [22]: ## Dummy classifier
         from sklearn.dummy import DummyClassifier
         clf = DummyClassifier(strategy='most_frequent',
                               random_state=0)
         clf.fit(X_train, y_train)
         clf.score(X_train, y_train)

Out[22]: array(0.63736264)

In [23]: ## Score the Model on Training and Testing Set
         result_scores['Null'] = \
                     (sklearn.metrics.accuracy_score(y_train,clf.predict(X_train)),
                      sklearn.metrics.accuracy_score(y_test,clf.predict(X_test)))
         get_results(result_scores)
```

```
Model                   Train    Test
-------------------------------------------
Logistic                0.9077   0.9211
Null                    0.6374   0.5877
```

1.2.3 LASSO

```
In [24]: ## Logistic Regression with l1 penalty
         ## Specify penalty directly as C = 1
         clf = linear_model.LogisticRegression(penalty='l1',
                                               C=1, solver = 'liblinear') # specify penalty
         clf.fit(X_train,y_train)
         ## get confusion matrix
         confusion_matrix(y_train,clf.predict(X_train))

/Users/miaocai/anaconda3/lib/python3.7/site-packages/sklearn/svm/base.py:922: ConvergenceWarni
  "the number of iterations.", ConvergenceWarning)


Out[24]: array([[137,  28],
                [ 11, 279]])

In [25]: ## Score the Model on Training and Testing Set
         result_scores['Logistic_L1_C_1'] = \
```

```
                        (sklearn.metrics.accuracy_score(y_train,clf.predict(X_train)),
                         sklearn.metrics.accuracy_score(y_test,clf.predict(X_test)))
        get_results(result_scores)
```

```
Model                   Train    Test
-------------------------------------------
Logistic                0.9077   0.9211
Null                    0.6374   0.5877
Logistic_L1_C_1         0.9143   0.9211
```

1.2.4 Ridge regression

```
In [26]: ## Logistic Regression with l1 penalty
         ## Specify penalty directly as C = 1
         clf = linear_model.LogisticRegression(penalty='l2',
                                               C=1, solver = 'liblinear') # specify penalty
         clf.fit(X_train,y_train)
         ## get confusion matrix
         confusion_matrix(y_train,clf.predict(X_train))
```

```
Out[26]: array([[135,  30],
                [ 12, 278]])
```

```
In [27]: clf.score(X_train,y_train)
         clf.score(X_test,y_test)
```

```
Out[27]: array(0.92105263)
```

```
In [28]: ## Score the Model on Training and Testing Set
         result_scores['Logistic_L2_C_1'] = \
                        (sklearn.metrics.accuracy_score(y_train,clf.predict(X_train)),
                         sklearn.metrics.accuracy_score(y_test,clf.predict(X_test)))
         get_results(result_scores)
```

```
Model                   Train    Test
-------------------------------------------
Logistic                0.9077   0.9211
Null                    0.6374   0.5877
Logistic_L1_C_1         0.9143   0.9211
Logistic_L2_C_1         0.9077   0.9211
```

1.2.5 Elastic net penalty logistic regression

```
In [29]: clf= sklearn.linear_model.ElasticNet(alpha=1.0, l1_ratio=0.5, random_state=32)
         clf.fit(X_train,y_train)
```

```
Out[29]: ElasticNet(alpha=1.0, copy_X=True, fit_intercept=True, l1_ratio=0.5,
               max_iter=1000, normalize=False, positive=False, precompute=False,
               random_state=32, selection='cyclic', tol=0.0001, warm_start=False)
```

```
In [30]: result_scores['ElasticNet'] = \
                     (clf.score(X_train,y_train),
                      clf.score(X_test,y_test) )
         get_results(result_scores)
```

```
Model                   Train    Test
-------------------------------------------
Logistic                0.9077   0.9211
Null                    0.6374   0.5877
Logistic_L1_C_1         0.9143   0.9211
Logistic_L2_C_1         0.9077   0.9211
ElasticNet              0.5094   0.5578
```

1.2.6 Random forest

```
In [31]: from sklearn import ensemble
         clf = ensemble.RandomForestClassifier(n_estimators=100,
                                               max_features=10,
                                               random_state=42)
         clf.fit(X_train,y_train)
         ## get confusion matrix
         confusion_matrix(y_train,clf.predict(X_train))
```

```
Out[31]: array([[165,   0],
                [  0, 290]])
```

```
In [32]: ## Score the Model on Training and Testing Set
         result_scores['RandomForest_noCV'] = \
                     (sklearn.metrics.accuracy_score(y_train,clf.predict(X_train)),
                      sklearn.metrics.accuracy_score(y_test,clf.predict(X_test)))
         get_results(result_scores)
```

```
Model                   Train    Test
-------------------------------------------
Logistic                0.9077   0.9211
Null                    0.6374   0.5877
Logistic_L1_C_1         0.9143   0.9211
Logistic_L2_C_1         0.9077   0.9211
ElasticNet              0.5094   0.5578
RandomForest_noCV       1.0      0.9561
```

1.2.7 Gradient tree boosting: Classification

```
In [33]: from sklearn import ensemble
         clf= sklearn.ensemble.GradientBoostingClassifier(loss='deviance', learning_rate=0.1, n
         clf.fit(X_train,y_train)
         ## Score the Model on Training and Testing Set
         result_scores['GradientTree'] = \
                     (sklearn.metrics.accuracy_score(y_train,clf.predict(X_train)),
                      sklearn.metrics.accuracy_score(y_test,clf.predict(X_test)))
         get_results(result_scores)
```

```
Model                  Train    Test
-------------------------------------------
Logistic               0.9077   0.9211
Null                   0.6374   0.5877
Logistic_L1_C_1        0.9143   0.9211
Logistic_L2_C_1        0.9077   0.9211
ElasticNet             0.5094   0.5578
RandomForest_noCV      1.0      0.9561
GradientTree           1.0      0.9561
```

1.3 Hyperpramaters

For the logistic regression withour regularization, no hyperparameter. For LASSO, and Ridge, hyperparameters are alpha, which can affect the regularization effects. For Elastic net penelty regression, there are hyperparameter accounts for the relative importance of the LASSO and Ridge regularizations. There is another hyperparameter , that accounts for the amount of regularization used in the model. For random forest, the hyperparameters are number of trees, and the max number of variables to consider for each tree, depth of the trees. For grandient tree boosting, hyperparameters are learning rate, the number of boosting stages to perform, subsample.

1.3.1 K-fold cross validation for LASSO

```
In [34]: ## Select the alpha through cross validation (k-folds leave one out)
         # auto generate 20 values between 1e-4 and 1e4 on log scale
         clf = linear_model.LogisticRegressionCV(cv=5,
                                                 Cs=20, ## takes awhile to fit 20 models!
                                                 penalty='l1',  #lasso
                                                 solver='liblinear')
         clf.fit(X_train,y_train)
```

```
/Users/miaocai/anaconda3/lib/python3.7/site-packages/sklearn/svm/base.py:922: ConvergenceWarni
  "the number of iterations.", ConvergenceWarning)
/Users/miaocai/anaconda3/lib/python3.7/site-packages/sklearn/svm/base.py:922: ConvergenceWarni
  "the number of iterations.", ConvergenceWarning)
/Users/miaocai/anaconda3/lib/python3.7/site-packages/sklearn/svm/base.py:922: ConvergenceWarni
  "the number of iterations.", ConvergenceWarning)
```

```
/Users/miaocai/anaconda3/lib/python3.7/site-packages/sklearn/svm/base.py:922: ConvergenceWarni
  "the number of iterations.", ConvergenceWarning)
/Users/miaocai/anaconda3/lib/python3.7/site-packages/sklearn/svm/base.py:922: ConvergenceWarni
  "the number of iterations.", ConvergenceWarning)
/Users/miaocai/anaconda3/lib/python3.7/site-packages/sklearn/svm/base.py:922: ConvergenceWarni
  "the number of iterations.", ConvergenceWarning)
```

```
Out[34]: LogisticRegressionCV(Cs=20, class_weight=None, cv=5, dual=False,
                     fit_intercept=True, intercept_scaling=1.0, max_iter=100,
                     multi_class='warn', n_jobs=None, penalty='l1',
                     random_state=None, refit=True, scoring=None, solver='liblinear',
                     tol=0.0001, verbose=0)
```

```
In [35]: ## Score the Model on Training and Testing Set
         result_scores['Logistic_L1_C_auto'] = \
                     (sklearn.metrics.accuracy_score(y_train,clf.predict(X_train)),
                      sklearn.metrics.accuracy_score(y_test,clf.predict(X_test)))
         get_results(result_scores)
```

```
Model                    Train    Test
-------------------------------------------
Logistic                 0.9077   0.9211
Null                     0.6374   0.5877
Logistic_L1_C_1          0.9143   0.9211
Logistic_L2_C_1          0.9077   0.9211
ElasticNet               0.5094   0.5578
RandomForest_noCV        1.0      0.9561
GradientTree             1.0      0.9561
Logistic_L1_C_auto       0.9516   0.9561
```

```
In [36]: ## 20 C's were fit
         clf.Cs
         ## The values of C's
         clf.Cs_
         ## The best fit C
         clf.C_
```

```
Out[36]: array([545.55947812])
```

1.3.2 K fold validation for Ridge

```
In [37]: ## Select the alpha through cross validation (k-folds leave one out)
         # auto generate 20 values between 1e-4 and 1e4 on log scale
         clf = linear_model.LogisticRegressionCV(cv=5,
                                                 Cs=20, ## takes awhile to fit 20 models!
                                                 penalty='l2',  #ridge
```

```
                                                  solver='liblinear')
        clf.fit(X_train,y_train)
```

```
Out[37]: LogisticRegressionCV(Cs=20, class_weight=None, cv=5, dual=False,
                    fit_intercept=True, intercept_scaling=1.0, max_iter=100,
                    multi_class='warn', n_jobs=None, penalty='l2',
                    random_state=None, refit=True, scoring=None, solver='liblinear',
                    tol=0.0001, verbose=0)
```

```
In [38]: ## Score the Model on Training and Testing Set
         result_scores['Logistic_L2_C_auto'] = \
                     (sklearn.metrics.accuracy_score(y_train,clf.predict(X_train)),
                      sklearn.metrics.accuracy_score(y_test,clf.predict(X_test)))
         get_results(result_scores)
```

```
Model                  Train    Test
-------------------------------------------
Logistic               0.9077   0.9211
Null                   0.6374   0.5877
Logistic_L1_C_1        0.9143   0.9211
Logistic_L2_C_1        0.9077   0.9211
ElasticNet             0.5094   0.5578
RandomForest_noCV      1.0      0.9561
GradientTree           1.0      0.9561
Logistic_L1_C_auto     0.9516   0.9561
Logistic_L2_C_auto     0.9473   0.9386
```

```
In [39]: ## 20 C's were fit
         clf.Cs
         ## The values of C's
         clf.Cs_
         ## The best fit C
         clf.C_
```

```
Out[39]: array([10000.])
```

1.3.3 K fold validation for elastic net

```
In [53]: clf = linear_model.ElasticNetCV(\
             l1_ratio=[.1, .5, .7, .9, .95, .99, 1], cv=5, eps=0.001, n_alphas=100, alphas=None
             fit_intercept=True, precompute='auto', max_iter=1000, tol=0.0001,  verbose=0,
             positive=False, random_state=32)
         clf.fit(X_train,y_train)

         clf.alpha_
         clf.l1_ratio_
```

```
Out[53]: 1.0
```

In [41]:
```
clf.score(X_train,y_train)
clf.score(X_test,y_test)
```

Out[41]: 0.6098683896120006

In [42]:
```
result_scores['ElasticNetCV'] = \
            (clf.score(X_train,y_train),
             clf.score(X_test,y_test) )
get_results(result_scores)
```

```
Model                 Train    Test
-------------------------------------------
Logistic              0.9077   0.9211
Null                  0.6374   0.5877
Logistic_L1_C_1       0.9143   0.9211
Logistic_L2_C_1       0.9077   0.9211
ElasticNet            0.5094   0.5578
RandomForest_noCV     1.0      0.9561
GradientTree          1.0      0.9561
Logistic_L1_C_auto    0.9516   0.9561
Logistic_L2_C_auto    0.9473   0.9386
ElasticNetCV          0.5952   0.6099
```

1.3.4 Grid search, random forest CV

In [43]:
```
from sklearn.model_selection import GridSearchCV
## specify grid
parameters = {'n_estimators':(50,100,200,300),
              'max_features':(2,4,6,8,10)}
## specify model without hyperparameters
rf_model = ensemble.RandomForestClassifier(random_state=32)
## specify search with model
clf = GridSearchCV(rf_model,
                   parameters,
                   cv=5,
                   return_train_score=True)
clf.fit(X_train,y_train)
```

Out[43]:
```
GridSearchCV(cv=5, error_score='raise-deprecating',
       estimator=RandomForestClassifier(bootstrap=True, class_weight=None, criterion=
            max_depth=None, max_features='auto', max_leaf_nodes=None,
            min_impurity_decrease=0.0, min_impurity_split=None,
            min_samples_leaf=1, min_samples_split=2,
            min_weight_fraction_leaf=0.0, n_estimators='warn', n_jobs=None,
            oob_score=False, random_state=32, verbose=0, warm_start=False),
       fit_params=None, iid='warn', n_jobs=None,
       param_grid={'n_estimators': (50, 100, 200, 300), 'max_features': (2, 4, 6, 8, 1
```

```
             pre_dispatch='2*n_jobs', refit=True, return_train_score=True,
             scoring=None, verbose=0)
```

```
In [44]: ## explore best hyperparameters
         clf.best_params_
```

```
Out[44]: {'max_features': 4, 'n_estimators': 50}
```

```
In [45]: ## add model score
         ## Score the Model on Training and Testing Set
         result_scores['RandomForest_CV'] = \
                     (sklearn.metrics.accuracy_score(y_train,clf.predict(X_train)),
                      sklearn.metrics.accuracy_score(y_test,clf.predict(X_test)))
         get_results(result_scores)
```

```
Model                  Train    Test
-------------------------------------------
Logistic               0.9077   0.9211
Null                   0.6374   0.5877
Logistic_L1_C_1        0.9143   0.9211
Logistic_L2_C_1        0.9077   0.9211
ElasticNet             0.5094   0.5578
RandomForest_noCV      1.0      0.9561
GradientTree           1.0      0.9561
Logistic_L1_C_auto     0.9516   0.9561
Logistic_L2_C_auto     0.9473   0.9386
ElasticNetCV           0.5952   0.6099
RandomForest_CV        0.9978   0.9386
```

```
In [46]: # depth of the tree
         from sklearn.model_selection import GridSearchCV
         ## specify grid
         parameters2 = {'max_depth':(2,5,7,10,20)}
         ## specify model without hyperparameters
         rf_model = ensemble.RandomForestClassifier(max_features=4,
                                                    n_estimators=50,
                                                    random_state=32)

         ## specify search with model
         clf = GridSearchCV(rf_model,
                            parameters2,
                            cv=5,
                            return_train_score=True)
         clf.fit(X_train,y_train)
```

```
Out[46]: GridSearchCV(cv=5, error_score='raise-deprecating',
                estimator=RandomForestClassifier(bootstrap=True, class_weight=None, criterion=
                     max_depth=None, max_features=4, max_leaf_nodes=None,
```

```
                     min_impurity_decrease=0.0, min_impurity_split=None,
                     min_samples_leaf=1, min_samples_split=2,
                     min_weight_fraction_leaf=0.0, n_estimators=50, n_jobs=None,
                     oob_score=False, random_state=32, verbose=0, warm_start=False),
                fit_params=None, iid='warn', n_jobs=None,
                param_grid={'max_depth': (2, 5, 7, 10, 20)},
                pre_dispatch='2*n_jobs', refit=True, return_train_score=True,
                scoring=None, verbose=0)
```

In [47]:
```
## explore best hyperparameters
clf.best_params_
```

Out[47]:
```
{'max_depth': 7}
```

In [48]:
```
## add model score
## Score the Model on Training and Testing Set
result_scores['RandomForest_CV2'] = \
            (sklearn.metrics.accuracy_score(y_train,clf.predict(X_train)),
             sklearn.metrics.accuracy_score(y_test,clf.predict(X_test)))
get_results(result_scores)
```

```
Model                  Train    Test
-------------------------------------------
Logistic               0.9077   0.9211
Null                   0.6374   0.5877
Logistic_L1_C_1        0.9143   0.9211
Logistic_L2_C_1        0.9077   0.9211
ElasticNet             0.5094   0.5578
RandomForest_noCV      1.0      0.9561
GradientTree           1.0      0.9561
Logistic_L1_C_auto     0.9516   0.9561
Logistic_L2_C_auto     0.9473   0.9386
ElasticNetCV           0.5952   0.6099
RandomForest_CV        0.9978   0.9386
RandomForest_CV2       0.9978   0.9298
```

1.3.5 Tuning Gradient boost classification

In [49]:
```
## specify grid
parameters = {'learning_rate':(0.1,0.3,0.5,0.7),
              'n_estimators':(50,100,150,200)}
## specify model without hyperparameters
rf_model = ensemble.GradientBoostingClassifier(random_state=32)
## specify search with model
clf = GridSearchCV(rf_model,
                   parameters,
                   cv=5,
```

```
                            return_train_score=True)
        clf.fit(X_train,y_train)
```

```
Out[49]: GridSearchCV(cv=5, error_score='raise-deprecating',
                estimator=GradientBoostingClassifier(criterion='friedman_mse', init=None,
                       learning_rate=0.1, loss='deviance', max_depth=3,
                       max_features=None, max_leaf_nodes=None,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=1, min_sampl...      subsample=1.0, tol=0.0001, valida
                       verbose=0, warm_start=False),
                fit_params=None, iid='warn', n_jobs=None,
                param_grid={'learning_rate': (0.1, 0.3, 0.5, 0.7), 'n_estimators': (50, 100, 1
                pre_dispatch='2*n_jobs', refit=True, return_train_score=True,
                scoring=None, verbose=0)
```

```
In [50]: ## explore best hyperparameters
         clf.best_params_
```

```
Out[50]: {'learning_rate': 0.5, 'n_estimators': 100}
```

```
In [51]: ## add model score
         ## Score the Model on Training and Testing Set
         result_scores['GBC_cv'] = \
                     (sklearn.metrics.accuracy_score(y_train,clf.predict(X_train)),
                      sklearn.metrics.accuracy_score(y_test,clf.predict(X_test)))
         get_results(result_scores)
```

```
Model                  Train    Test
-------------------------------------------
Logistic               0.9077   0.9211
Null                   0.6374   0.5877
Logistic_L1_C_1        0.9143   0.9211
Logistic_L2_C_1        0.9077   0.9211
ElasticNet             0.5094   0.5578
RandomForest_noCV      1.0      0.9561
GradientTree           1.0      0.9561
Logistic_L1_C_auto     0.9516   0.9561
Logistic_L2_C_auto     0.9473   0.9386
ElasticNetCV           0.5952   0.6099
RandomForest_CV        0.9978   0.9386
RandomForest_CV2       0.9978   0.9298
GBC_cv                 1.0      0.9386
```