HD ML Example

March 17, 2017

0.0.1 See entire project information: https://github.com/sandiegohearts/sandiegohearts.github.io

0.0.2 See TabPy information (ipython and Tableau files) : https://github.com/sandiegohearts/sandiegohearts.github.io/tree/master/TabPy

1 Using Python and Machine Learning Algorithms within Tableau: Heart Disease

1.0.1 Orysya Stus

UCI Machine Learning Repository link:
https://archive.ics.uci.edu/ml/datasets/Heart+Disease
Used reference code to deploy functions to Tableau via TabPy:
https://www.tableau.com/about/blog/2017/1/building-advanced-analytics-applications-tabpy-64916

In [1]:
```
import math
import numpy as np
import pandas as pd
from sklearn.grid_search import GridSearchCV
from sklearn.linear_model import LogisticRegressionCV
from sklearn.naive_bayes import GaussianNB
from sklearn.cross_validation import cross_val_score, cross_val_predict, St
from sklearn import preprocessing, metrics, svm, ensemble
from sklearn.metrics import accuracy_score, classification_report
import tabpy_client
%pylab inline
```

```
C:\Users\Orysya\Anaconda\lib\site-packages\sklearn\cross_validation.py:44: Deprecat
  "This module will be removed in 0.20.", DeprecationWarning)
C:\Users\Orysya\Anaconda\lib\site-packages\sklearn\grid_search.py:43: DeprecationWa
  DeprecationWarning)


Populating the interactive namespace from numpy and matplotlib
```

1.0.2 Data pre-processing: drop nulls, examine class attribute

In [2]:
```
hd = pd.read_csv('./processed.cleveland.data.csv', names= ["age", "sex", "c
print 'The size of the file, before nulls dropped, is: ', hd.shape

hd = hd.replace('?', np.nan)
hd = hd.dropna()
# Define diagnosis of 1-4 as 'risk' and 0 as 'healthy'
def diagnosis(row):
    if row['diagnosis'] > 0:
        return 'risk'
    else:
        return 'healthy'
hd['diagnosis'] = hd.apply(diagnosis, axis=1)
print 'The size of the file, after nulls dropped, is: ', hd.shape
#hd.to_csv('./cleveland_data_for tableau.csv', index=False)
hd.head()
```

```
The size of the file, before nulls dropped, is:  (303, 14)
The size of the file, after nulls dropped, is:  (297, 14)
```

Out[2]:

	age	sex	chest_pain	resting_bp	chol	fbs	restecg	thalach	exang	\
0	63	1	1	145	233	1	2	150	0	
1	67	1	4	160	286	0	2	108	1	
2	67	1	4	120	229	0	2	129	1	
3	37	1	3	130	250	0	0	187	0	
4	41	0	2	130	204	0	2	172	0	

	oldpeak	slope	ca	thal	diagnosis
0	2.3	3	0	6	healthy
1	1.5	2	3	3	risk
2	2.6	2	2	7	risk
3	3.5	3	0	3	healthy
4	1.4	1	0	3	healthy

In [3]:
```
hd.groupby('diagnosis').describe()
```

Out[3]:

diagnosis		age	chest_pain	chol	exang	fbs
healthy	count	160.000000	160.000000	160.000000	160.000000	160.00000
	mean	52.643750	2.793750	243.493750	0.143750	0.14375
	std	9.551151	0.925508	53.757550	0.351938	0.35193
	min	29.000000	1.000000	126.000000	0.000000	0.00000
	25%	44.750000	2.000000	208.750000	0.000000	0.00000
	50%	52.000000	3.000000	235.500000	0.000000	0.00000
	75%	59.000000	3.000000	268.250000	0.000000	0.00000
	max	76.000000	4.000000	564.000000	1.000000	1.00000
risk	count	137.000000	137.000000	137.000000	137.000000	137.00000

```
             mean    56.759124    3.583942  251.854015    0.540146    0.145985
             std      7.899670    0.828201   49.679937    0.500215    0.354387
             min     35.000000    1.000000  131.000000    0.000000    0.000000
             25%     53.000000    4.000000  218.000000    0.000000    0.000000
             50%     58.000000    4.000000  253.000000    1.000000    0.000000
             75%     62.000000    4.000000  284.000000    1.000000    0.000000
             max     77.000000    4.000000  409.000000    1.000000    1.000000

                       oldpeak     restecg  resting_bp         sex       slope
diagnosis
healthy      count  160.000000  160.000000  160.000000  160.000000  160.000000
             mean     0.598750    0.843750  129.175000    0.556250    1.412500
             std      0.787160    0.987640   16.373990    0.498386    0.597558
             min      0.000000    0.000000   94.000000    0.000000    1.000000
             25%      0.000000    0.000000  120.000000    0.000000    1.000000
             50%      0.200000    0.000000  130.000000    1.000000    1.000000
             75%      1.100000    2.000000  140.000000    1.000000    2.000000
             max      4.200000    2.000000  180.000000    1.000000    3.000000
risk         count  137.000000  137.000000  137.000000  137.000000  137.000000
             mean     1.589051    1.175182  134.635036    0.817518    1.824818
             std      1.305006    0.976924   18.896730    0.387658    0.567474
             min      0.000000    0.000000  100.000000    0.000000    1.000000
             25%      0.600000    0.000000  120.000000    1.000000    1.000000
             50%      1.400000    2.000000  130.000000    1.000000    2.000000
             75%      2.500000    2.000000  145.000000    1.000000    2.000000
             max      6.200000    2.000000  200.000000    1.000000    3.000000

                       thalach
diagnosis
healthy      count  160.000000
             mean   158.581250
             std     19.043304
             min     96.000000
             25%    149.000000
             50%    161.000000
             75%    172.000000
             max    202.000000
risk         count  137.000000
             mean   139.109489
             std     22.710673
             min     71.000000
             25%    125.000000
             50%    142.000000
             75%    157.000000
             max    195.000000
```

In [4]:
```
# Since class attribute is vales 0-4, there is no need to convert text to r
encoder = preprocessing.LabelEncoder()
```

```
hd['diagnosis'] = encoder.fit_transform(hd['diagnosis'])
hd.head()
```

Out[4]:

```
   age  sex  chest_pain  resting_bp  chol  fbs  restecg  thalach  exang  \
0   63    1           1         145   233    1        2      150      0
1   67    1           4         160   286    0        2      108      1
2   67    1           4         120   229    0        2      129      1
3   37    1           3         130   250    0        0      187      0
4   41    0           2         130   204    0        2      172      0

   oldpeak  slope ca thal  diagnosis
0      2.3      3  0    6          0
1      1.5      2  3    3          1
2      2.6      2  2    7          1
3      3.5      3  0    3          0
4      1.4      1  0    3          0
```

In [5]:

```
# Split data into X, y
X = np.array(hd.drop(['diagnosis'], 1))
y = np.array(hd['diagnosis'])
```

1.1 Training the data using different models

In [ ]:

```
# Scale the data (Assume that all features are centered around 0 and have v
# Note in order for StandardScaler to work, need to remove any nulls in dat
scalar = preprocessing.StandardScaler().fit(X)
X = scalar.transform(X)

# 10 fold stratified cross validation
kf = StratifiedKFold(y, n_folds=10, random_state=None, shuffle=True)
```

In [8]:

```
# Logistic regression with 10 fold stratified cross-validation using model
lgclf = LogisticRegressionCV(Cs=list(np.power(10.0, np.arange(-10, 10))),pe
lgclf.fit(X, y)
y_pred = lgclf.predict(X)

# Show classification report for the best model (set of parameters) run ove
print("Classification report:")
print(classification_report(y, y_pred))

# Show accuracy and area under ROC curve
print("Accuracy: %0.3f" % accuracy_score(y, y_pred, normalize=True))
print("Aucroc: %0.3f" % metrics.roc_auc_score(y, y_pred))
```

```
Classification report:
             precision    recall  f1-score   support

          0       0.84      0.88      0.86       160
          1       0.85      0.80      0.83       137
```

```
avg / total       0.85      0.85      0.84       297

Accuracy: 0.845
Aucroc: 0.842
```

```
In [9]: # Naive Bayes with 10 fold stratified cross-validation
        nbclf = GaussianNB()
        scores = cross_val_score(nbclf, X, y, cv=kf, scoring= 'accuracy')

        print("Accuracy: %0.3f" % (scores.mean()))
        print("Aucroc: %0.3f" % metrics.roc_auc_score(y, cross_val_predict(nbclf, X
```

```
Accuracy: 0.841
Aucroc: 0.838
```

Support Vector Machine reference:
http://scikit-learn.org/stable/modules/svm.html
To determine which model evaluations work best, via 'scoring':
http://scikit-learn.org/stable/modules/model_evaluation.html#scoring-parameter

```
In [6]: # Define the parameter grid to use for tuning the Support Vector Machine
        parameters = [{'kernel': ['rbf'], 'gamma': [1e-3, 1e-4], 'C': [1, 10, 100,
                      {'kernel': ['linear'], 'C': [1, 10, 100, 1000]}]

        # Choose performance measures for modeling
        scoringmethods = ['f1','accuracy','precision', 'recall','roc_auc']
        # scoringmethods = ['f1_weighted', 'accuracy', 'precision_weighted', 'reca
```

```
C:\Users\Orysya\Anaconda\lib\site-packages\sklearn\utils\validation.py:429: DataCon
  warnings.warn(msg, _DataConversionWarning)
```

```
In [7]: # Iterate through different metrics looking for best parameter set
        for score in scoringmethods:
            print("~~~ Hyper-parameter tuning for best %s ~~~" % score)

            # Setup for grid search with cross-validation for Support Vector Machi
            # n_jobs=-1 for parallel execution using all available cores
            svmclf = GridSearchCV(svm.SVC(C=1), parameters, cv=kf, scoring=score,n_
            svmclf.fit(X, y)

            # Show each result from grid search
            print("Scores for different parameter combinations in the grid:")
            for params, mean_score, scores in svmclf.grid_scores_:
                print("  %0.3f (+/-%0.03f) for %r"
                      % (mean_score, scores.std() / 2, params))
```

```
            print("")

        # Show classification report for the best model (set of parameters) run ove
        print("Classification report:")
        y_pred = svmclf.predict(X)
        print(classification_report(y, y_pred))

        # Show the definition of the best model
        print("Best model:")
        print(svmclf.best_estimator_)

        # Show accuracy
        print("Accuracy: %0.3f" % accuracy_score(y, y_pred, normalize=True))
        print("Aucroc: %0.3f" % metrics.roc_auc_score(y, y_pred))
        print("")
```

```
~~~ Hyper-parameter tuning for best f1 ~~~
Scores for different parameter combinations in the grid:
  0.806 (+/-0.040) for {'kernel': 'rbf', 'C': 1, 'gamma': 0.001}
  0.000 (+/-0.000) for {'kernel': 'rbf', 'C': 1, 'gamma': 0.0001}
  0.821 (+/-0.053) for {'kernel': 'rbf', 'C': 10, 'gamma': 0.001}
  0.806 (+/-0.040) for {'kernel': 'rbf', 'C': 10, 'gamma': 0.0001}
  0.807 (+/-0.045) for {'kernel': 'rbf', 'C': 100, 'gamma': 0.001}
  0.817 (+/-0.053) for {'kernel': 'rbf', 'C': 100, 'gamma': 0.0001}
  0.814 (+/-0.044) for {'kernel': 'rbf', 'C': 1000, 'gamma': 0.001}
  0.803 (+/-0.044) for {'kernel': 'rbf', 'C': 1000, 'gamma': 0.0001}
  0.803 (+/-0.044) for {'kernel': 'linear', 'C': 1}
  0.807 (+/-0.045) for {'kernel': 'linear', 'C': 10}
  0.803 (+/-0.044) for {'kernel': 'linear', 'C': 100}
  0.803 (+/-0.044) for {'kernel': 'linear', 'C': 1000}

~~~ Hyper-parameter tuning for best accuracy ~~~
Scores for different parameter combinations in the grid:
  0.842 (+/-0.030) for {'kernel': 'rbf', 'C': 1, 'gamma': 0.001}
  0.539 (+/-0.004) for {'kernel': 'rbf', 'C': 1, 'gamma': 0.0001}
  0.845 (+/-0.043) for {'kernel': 'rbf', 'C': 10, 'gamma': 0.001}
  0.842 (+/-0.030) for {'kernel': 'rbf', 'C': 10, 'gamma': 0.0001}
  0.832 (+/-0.035) for {'kernel': 'rbf', 'C': 100, 'gamma': 0.001}
  0.842 (+/-0.043) for {'kernel': 'rbf', 'C': 100, 'gamma': 0.0001}
  0.835 (+/-0.037) for {'kernel': 'rbf', 'C': 1000, 'gamma': 0.001}
  0.828 (+/-0.033) for {'kernel': 'rbf', 'C': 1000, 'gamma': 0.0001}
  0.828 (+/-0.033) for {'kernel': 'linear', 'C': 1}
  0.832 (+/-0.035) for {'kernel': 'linear', 'C': 10}
  0.828 (+/-0.033) for {'kernel': 'linear', 'C': 100}
  0.828 (+/-0.033) for {'kernel': 'linear', 'C': 1000}

~~~ Hyper-parameter tuning for best precision ~~~
Scores for different parameter combinations in the grid:
```

```
  0.918 (+/-0.045) for {'kernel': 'rbf', 'C': 1, 'gamma': 0.001}
  0.000 (+/-0.000) for {'kernel': 'rbf', 'C': 1, 'gamma': 0.0001}
  0.858 (+/-0.044) for {'kernel': 'rbf', 'C': 10, 'gamma': 0.001}
  0.918 (+/-0.045) for {'kernel': 'rbf', 'C': 10, 'gamma': 0.0001}
  0.836 (+/-0.032) for {'kernel': 'rbf', 'C': 100, 'gamma': 0.001}
  0.856 (+/-0.044) for {'kernel': 'rbf', 'C': 100, 'gamma': 0.0001}
  0.838 (+/-0.040) for {'kernel': 'rbf', 'C': 1000, 'gamma': 0.001}
  0.835 (+/-0.032) for {'kernel': 'rbf', 'C': 1000, 'gamma': 0.0001}
  0.835 (+/-0.032) for {'kernel': 'linear', 'C': 1}
  0.836 (+/-0.032) for {'kernel': 'linear', 'C': 10}
  0.835 (+/-0.032) for {'kernel': 'linear', 'C': 100}
  0.835 (+/-0.032) for {'kernel': 'linear', 'C': 1000}

~~~ Hyper-parameter tuning for best recall ~~~
Scores for different parameter combinations in the grid:
  0.729 (+/-0.055) for {'kernel': 'rbf', 'C': 1, 'gamma': 0.001}
  0.000 (+/-0.000) for {'kernel': 'rbf', 'C': 1, 'gamma': 0.0001}
  0.795 (+/-0.068) for {'kernel': 'rbf', 'C': 10, 'gamma': 0.001}
  0.729 (+/-0.055) for {'kernel': 'rbf', 'C': 10, 'gamma': 0.0001}
  0.788 (+/-0.063) for {'kernel': 'rbf', 'C': 100, 'gamma': 0.001}
  0.788 (+/-0.068) for {'kernel': 'rbf', 'C': 100, 'gamma': 0.0001}
  0.795 (+/-0.054) for {'kernel': 'rbf', 'C': 1000, 'gamma': 0.001}
  0.781 (+/-0.060) for {'kernel': 'rbf', 'C': 1000, 'gamma': 0.0001}
  0.781 (+/-0.060) for {'kernel': 'linear', 'C': 1}
  0.788 (+/-0.063) for {'kernel': 'linear', 'C': 10}
  0.781 (+/-0.060) for {'kernel': 'linear', 'C': 100}
  0.781 (+/-0.060) for {'kernel': 'linear', 'C': 1000}

~~~ Hyper-parameter tuning for best roc_auc ~~~
Scores for different parameter combinations in the grid:
  0.905 (+/-0.028) for {'kernel': 'rbf', 'C': 1, 'gamma': 0.001}
  0.904 (+/-0.026) for {'kernel': 'rbf', 'C': 1, 'gamma': 0.0001}
  0.904 (+/-0.029) for {'kernel': 'rbf', 'C': 10, 'gamma': 0.001}
  0.904 (+/-0.027) for {'kernel': 'rbf', 'C': 10, 'gamma': 0.0001}
  0.911 (+/-0.026) for {'kernel': 'rbf', 'C': 100, 'gamma': 0.001}
  0.905 (+/-0.029) for {'kernel': 'rbf', 'C': 100, 'gamma': 0.0001}
  0.902 (+/-0.026) for {'kernel': 'rbf', 'C': 1000, 'gamma': 0.001}
  0.907 (+/-0.027) for {'kernel': 'rbf', 'C': 1000, 'gamma': 0.0001}
  0.904 (+/-0.028) for {'kernel': 'linear', 'C': 1}
  0.903 (+/-0.028) for {'kernel': 'linear', 'C': 10}
  0.903 (+/-0.027) for {'kernel': 'linear', 'C': 100}
  0.903 (+/-0.027) for {'kernel': 'linear', 'C': 1000}

Classification report:
             precision    recall  f1-score   support

          0       0.84      0.89      0.86       160
          1       0.86      0.80      0.83       137
```

```
avg / total       0.85      0.85      0.85       297

Best model:
SVC(C=100, cache_size=200, class_weight=None, coef0=0.0,
  decision_function_shape=None, degree=3, gamma=0.001, kernel='rbf',
  max_iter=-1, probability=False, random_state=None, shrinking=True,
  tol=0.001, verbose=False)
Accuracy: 0.848
Aucroc: 0.845
```

```
In [10]: # Define the parameter grid to use for tuning the Gradient Boosting Classi
         gridparams = dict(learning_rate=[0.01, 0.1],loss=['deviance','exponential'

         # Parameters we're not tuning for this classifier
         params = {'n_estimators': 1500, 'max_depth': 4}

         # Setup for grid search with cross-validation for Gradient Boosting Classi
         # n_jobs=-1 for parallel execution using all available cores
         gbclf = GridSearchCV(ensemble.GradientBoostingClassifier(**params), gridpa
         gbclf.fit(X,y)

         # Show the definition of the best model
         print("Best model:")
         print(gbclf.best_estimator_)
         print("")

         # Show classification report for the best model (set of parameters) run ov
         print("Classification report:")
         y_pred = gbclf.predict(X)
         print(classification_report(y, y_pred))

         # Show accuracy and area under ROC curve
         print("Accuracy: %0.3f" % accuracy_score(y, y_pred, normalize=True))
         print("Aucroc: %0.3f" % metrics.roc_auc_score(y, y_pred))
```

```
Best model:
GradientBoostingClassifier(criterion='friedman_mse', init=None,
              learning_rate=0.01, loss='deviance', max_depth=4,
              max_features=None, max_leaf_nodes=None,
              min_impurity_split=1e-07, min_samples_leaf=1,
              min_samples_split=2, min_weight_fraction_leaf=0.0,
              n_estimators=1500, presort='auto', random_state=None,
              subsample=1.0, verbose=0, warm_start=False)

Classification report:
```

```
             precision    recall  f1-score   support

          0       1.00      1.00      1.00       160
          1       1.00      1.00      1.00       137

avg / total       1.00      1.00      1.00       297

Accuracy: 1.000
Aucroc: 1.000
```

In [11]:
```python
# Connect to TabPy server using the client library
connection = tabpy_client.Client('http://localhost:9004/')
```

In [12]:
```python
# The scoring function that will use the Gradient Boosting Classifier to c
def HDDiagnosis(age, sex, chest_pain, resting_bp, chol, fbs, restecg, thal
    X = np.column_stack([age, sex, chest_pain, resting_bp, chol, fbs, rest
    X = scalar.transform(X)
    return encoder.inverse_transform(gbclf.predict(X)).tolist()
```

In [14]:
```python
connection.deploy('HDDiagnosis',
                  HDDiagnosis,
                  'Returns diagnosis suggestion (healthy or risk) based or
                  override=True)
```

In [ ]:

In [ ]: