COMP-4448

Christopher Kramer

2021-01-04

Cancer detection via Tree

In [1]:
```
import pandas as pd
import numpy as np
import os, sys
from io import StringIO
from typing import Optional
import seaborn as sns
```

Find your own dataset suitable for classification with at least three input variables and 200 cases: You will build a decision tree classifier and a random forest classifier. Find some interesting dataset instead of the popular iris data, etc. Feel free to use a dataset suitable for classification from this link provided below or some other source of your choice:
https://vincentarelbundock.github.io/Rdatasets/articles/data.html Address the following and include code/output snippets from b) to f).
Include the response under each sub question.

a) State your research question, for example: Are decision trees and random forest good models for predicting whether someone will default on a loan or not based on their age and income level? This is just an example, your dataset does not have to be (or should not be) about loans.

Can the the presence of cancer in a digitized cell image be predicted through the features (image data extractions) provided in the UCI ML Breast Cancer Wisconsin (Diagnostic) dataset using a decision tree and random forest?

In [61]:
```
from sklearn.datasets import load_breast_cancer
```

In [62]:
```
cancer = load_breast_cancer(as_frame = True)
```

In [63]:
```
print(cancer['DESCR'])
```

```
.. _breast_cancer_dataset:

Breast cancer wisconsin (diagnostic) dataset
--------------------------------------------

**Data Set Characteristics:**

    :Number of Instances: 569

    :Number of Attributes: 30 numeric, predictive attributes and the class

    :Attribute Information:
        - radius (mean of distances from center to points on the perimeter)
        - texture (standard deviation of gray-scale values)
        - perimeter
        - area
        - smoothness (local variation in radius lengths)
        - compactness (perimeter^2 / area - 1.0)
        - concavity (severity of concave portions of the contour)
        - concave points (number of concave portions of the contour)
        - symmetry
        - fractal dimension ("coastline approximation" - 1)

        The mean, standard error, and "worst" or largest (mean of the three
        worst/largest values) of these features were computed for each image,
        resulting in 30 features.  For instance, field 0 is Mean Radius, field
        10 is Radius SE, field 20 is Worst Radius.

        - class:
                - WDBC-Malignant
                - WDBC-Benign

    :Summary Statistics:

    ===================================== ====== ======
                                           Min    Max
    ===================================== ====== ======
    radius (mean):                        6.981  28.11
    texture (mean):                       9.71   39.28
    perimeter (mean):                     43.79  188.5
    area (mean):                          143.5  2501.0
    smoothness (mean):                    0.053  0.163
    compactness (mean):                   0.019  0.345
    concavity (mean):                     0.0    0.427
    concave points (mean):                0.0    0.201
    symmetry (mean):                      0.106  0.304
    fractal dimension (mean):             0.05   0.097
    radius (standard error):              0.112  2.873
    texture (standard error):             0.36   4.885
    perimeter (standard error):           0.757  21.98
    area (standard error):                6.802  542.2
    smoothness (standard error):          0.002  0.031
    compactness (standard error):         0.002  0.135
    concavity (standard error):           0.0    0.396
    concave points (standard error):      0.0    0.053
    symmetry (standard error):            0.008  0.079
    fractal dimension (standard error):   0.001  0.03
    radius (worst):                       7.93   36.04
    texture (worst):                      12.02  49.54
    perimeter (worst):                    50.41  251.2
    area (worst):                         185.2  4254.0
    smoothness (worst):                   0.071  0.223
    compactness (worst):                  0.027  1.058
    concavity (worst):                    0.0    1.252
    concave points (worst):               0.0    0.291
    symmetry (worst):                     0.156  0.664
    fractal dimension (worst):            0.055  0.208
    ===================================== ====== ======

    :Missing Attribute Values: None

    :Class Distribution: 212 - Malignant, 357 - Benign

    :Creator:  Dr. William H. Wolberg, W. Nick Street, Olvi L. Mangasarian

    :Donor: Nick Street

    :Date: November, 1995

This is a copy of UCI ML Breast Cancer Wisconsin (Diagnostic) datasets.
https://goo.gl/U2Uwz2

Features are computed from a digitized image of a fine needle
aspirate (FNA) of a breast mass.  They describe
characteristics of the cell nuclei present in the image.

Separating plane described above was obtained using
Multisurface Method-Tree (MSM-T) [K. P. Bennett, Decision Tree
Construction Via Linear Programming." Proceedings of the 4th
Midwest Artificial Intelligence and Cognitive Science Society,
pp. 97-101, 1992], a classification method which uses linear
programming to construct a decision tree.  Relevant features
were selected using an exhaustive search in the space of 1-4
features and 1-3 separating planes.

The actual linear program used to obtain the separating plane
in the 3-dimensional space is that described in:
[K. P. Bennett and O. L. Mangasarian: "Robust Linear
Programming Discrimination of Two Linearly Inseparable Sets",
Optimization Methods and Software 1, 1992, 23-34].

This database is also available through the UW CS ftp server:

ftp ftp.cs.wisc.edu
cd math-prog/cpo-dataset/machine-learn/WDBC/

.. topic:: References

   - W.N. Street, W.H. Wolberg and O.L. Mangasarian. Nuclear feature extraction
     for breast tumor diagnosis. IS&T/SPIE 1993 International Symposium on
     Electronic Imaging: Science and Technology, volume 1905, pages 861-870,
     San Jose, CA, 1993.
   - O.L. Mangasarian, W.N. Street and W.H. Wolberg. Breast cancer diagnosis and
     prognosis via linear programming. Operations Research, 43(4), pages 570-577,
     July-August 1995.
   - W.H. Wolberg, W.N. Street, and O.L. Mangasarian. Machine learning techniques
     to diagnose breast cancer from fine-needle aspirates. Cancer Letters 77 (1994)
     163-171.
```

In [64]:
```
df = cancer['frame']
```

In [66]:
```
def min_max(x: pd.Series) -> pd.Series:
    return (x - x.min()) / (x.max() - x.min())

def minmax_column_helper(x: str):
    df[x] = min_max(df[x])
```

In [67]:
```
list(filter(minmax_column_helper, df.columns[~df.columns.isin(['target'])]))
```

Out[67]:
```
[]
```

In [68]:
```
df.sample(5)
```

Out[68]:

	mean radius	mean texture	mean perimeter	mean area	mean smoothness	mean compactness	mean concavity	mean concave points	mean symmetry	mean fractal dimension	...	worst texture	worst perim
92	0.297648	0.170781	0.282980	0.173150	0.188860	0.095608	0.076406	0.131610	0.164646	0.067818	...	0.275320	0.26
437	0.334091	0.212039	0.317808	0.195368	0.268435	0.121373	0.062802	0.146322	0.330303	0.185975	...	0.254797	0.25
542	0.367220	0.531382	0.351907	0.222736	0.271915	0.161831	0.096181	0.190447	0.393939	0.144061	...	0.540245	0.26
342	0.193052	0.177545	0.191417	0.097731	0.457434	0.219588	0.126453	0.166054	0.361616	0.402485	...	0.210021	0.14
497	0.259785	0.257017	0.253334	0.142778	0.330866	0.174591	0.084560	0.117744	0.255354	0.221146	...	0.282358	0.21

5 rows × 31 columns

c) Data split/scaling

In [69]:
```
X_train, X_test, y_train, y_test = train_test_split(df[df.columns[~df.columns.isin(['target'])]],
                                                    df['target'],
                                                    test_size=.1,
                                                    random_state=66)
```

d) Model construction (a decision tree and a random forest)

In [70]:
```
from sklearn.ensemble import RandomForestClassifier
```

DecisionTree:

In [71]:
```
dt_clf = DecisionTreeClassifier(random_state=42)
```

In [72]:
```
dt_clf.fit(X_train, y_train)
```

Out[72]:
```
DecisionTreeClassifier(random_state=42)
```

RandomForest:

In [75]:
```
rf_clf = RandomForestClassifier(random_state=42)
```

In [76]:
```
rf_clf.fit(X_train, y_train)
```

Out[76]:
```
RandomForestClassifier(random_state=42)
```

e) Hyperparameter tuning (for each model, tune the hyperparameter that is important to you and use any of the methods for hyperparameter tuning learned in class such as cross validation with for loop, gridsearch cross validation, etc. You could tune more than a single parameter for each model if you want).

DecisionTree:

In [79]:
```
dt_grid = GridSearchCV(dt_clf,
                       {
                           'max_depth': range(1, dt_clf.tree_.max_depth+1),
                           "criterion": ["gini", "entropy"],
                           "splitter": ["best", "random"],
                           "max_features": ["auto", "sqrt", "log2"]
                       },
                       cv=5
                      )
```

In [80]:
```
dt_grid.fit(X_train, y_train)
```

Out[80]:
```
GridSearchCV(cv=5, estimator=DecisionTreeClassifier(random_state=42),
             param_grid={'criterion': ['gini', 'entropy'],
                         'max_depth': range(1, 9),
                         'max_features': ['auto', 'sqrt', 'log2'],
                         'splitter': ['best', 'random']})
```

In [82]:
```
dt_grid.best_params_
```

Out[82]:
```
{'criterion': 'entropy',
 'max_depth': 4,
 'max_features': 'log2',
 'splitter': 'best'}
```

RandomForest:

In [83]:
```
rf_grid = GridSearchCV(rf_clf,
                       {
                           'n_estimators': range(50, 501, 50),
                           'max_depth': range(1, dt_clf.tree_.max_depth+1), #using max_depth of DecisionTr
ee since default is None for RandomForest
                           'criterion': ["gini", "entropy"],
#                           'splitter': ["best", "random"],
                           'max_features': ["auto", "sqrt", "log2"],
                       },
                       cv=5,
                       verbose=1,
                       n_jobs=-1
                      )
```

In [84]:
```
rf_grid.fit(X_train, y_train)
```

```
Fitting 5 folds for each of 480 candidates, totalling 3840 fits

[Parallel(n_jobs=-1)]: Using backend LokyBackend with 16 concurrent workers.
[Parallel(n_jobs=-1)]: Done  18 tasks      | elapsed:    1.8s
[Parallel(n_jobs=-1)]: Done 168 tasks      | elapsed:    6.8s
[Parallel(n_jobs=-1)]: Done 418 tasks      | elapsed:   15.2s
[Parallel(n_jobs=-1)]: Done 768 tasks      | elapsed:   27.7s
[Parallel(n_jobs=-1)]: Done 1218 tasks     | elapsed:   45.9s
[Parallel(n_jobs=-1)]: Done 1768 tasks     | elapsed:  1.1min
[Parallel(n_jobs=-1)]: Done 2418 tasks     | elapsed:  1.6min
[Parallel(n_jobs=-1)]: Done 3168 tasks     | elapsed:  2.1min
[Parallel(n_jobs=-1)]: Done 3840 out of 3840 | elapsed:  2.6min finished
```

Out[84]:
```
GridSearchCV(cv=5, estimator=RandomForestClassifier(random_state=42), n_jobs=-1,
             param_grid={'criterion': ['gini', 'entropy'],
                         'max_depth': range(1, 9),
                         'max_features': ['auto', 'sqrt', 'log2'],
                         'n_estimators': range(50, 501, 50),},
             verbose=1)
```

In [85]:
```
import joblib
```

In [86]:
```
joblib.dump(rf_grid, 'random_forest_grid.joblib')
```

Out[86]:
```
['random_forest_grid.joblib']
```

In [87]:
```
rf_grid.best_params_
```

Out[87]:
```
{'criterion': 'entropy',
 'max_depth': 7,
 'max_features': 'auto',
 'n_estimators': 150}
```

f) Use the best or optimal parameter values to build a model, then compute the accuracy score for the decision tree and for the random forest).

In [88]:
```
accuracy_scores = pd.DataFrame([[DecisionTreeClassifier(random_state=2).fit(X_train, y_train)),
                                 [DecisionTreeClassifier(random_state=2, **dt_grid.best_params_).fit(X_t
rain, y_train)],
                                 [RandomForestClassifier(random_state=2).fit(X_train, y_train)),
                                 [RandomForestClassifier(random_state=2, **rf_grid.best_params_).fit(X_tr
ain, y_train)]],
                                columns=['Model'])
```

In [89]:
```
accuracy_scores['Name'] = ['Default DecisionTree',
                           'GridSearchCV DecisionTree',
                           'Default RandomForest',
                           'GridSearchCV RandomForest']
accuracy_scores['Training Accuracy'] = accuracy_scores['Model'].apply(lambda x: accuracy_score(y_train,
                                                                                               x.predic
t(X_train)))
accuracy_scores['Testing/Validation Accuracy'] = accuracy_scores['Model'].apply(lambda x: accuracy_scor
e(y_test,
                                                                                x.predict(X_test)))
```

In [90]:
```
accuracy_scores
```

Out[90]:

	Model	Name	Training Accuracy	Testing/Validation Accuracy
0	DecisionTreeClassifier(random_state=2)	Default DecisionTree	1.000000	0.941520
1	DecisionTreeClassifier(criterion='entropy', ma...	GridSearchCV DecisionTree	0.969849	0.941520
2	DecisionTreeClassifier(max_features='auto', r...	Default RandomForest	1.000000	0.959064
3	(DecisionTreeClassifier(criterion='entropy', m...	GridSearchCV RandomForest	1.000000	0.970760

While the DecisionTree models share the same testing accuracy, the tuned hyperparameter model has some additional loss against training data. Of the two RandomForest models, the tuned hyperparameter model performs best against training data, and is also the best overall model.

g) Discuss about overfitting for both models and, also discuss which model is better for classification for your dataset and why?

The most evident example of overfitting with this model set occurs with the DecisionTree models. The default model shows training accuracy at 100%, with testing at ~94.15%. The cross-validated grid search implementation loses training accuracy, but maintains the same testing score. This is probably because some overfitting was reduced during the cross-validation fitting process. While there is no tangible improvement in testing accuracy, the cross-validated model is still most likely the best model, since it hypothetically generalizes better with less overfitting.

There is not obvious evidence for overfitting with the RandomForest models, although cross-validation provides boosted testing performance.

Overall, the RandomForest model seems the best fit for the use case given its overall validation score.