Decision Tree

Introduction

We will be using the wine quality data set for these exercises. This data set contains various chemical properties of wine, such as acidity, sugar, pH, and alcohol. It also contains a quality metric (3-9, with highest being better) and a color (red or white). The name of the file is `Wine_Quality_Data.csv`.

In [1]:
```
import os, pandas as pd, numpy as np, matplotlib.pyplot as plt, seaborn as sns
```

- Import the data and examine the features.
- We will be using all of them to predict `color` (white or red), but the colors feature will need to be integer encoded.

In [3]:
```
### BEGIN SOLUTION
filepath = 'Wine_Quality_Data.csv'
data = pd.read_csv("C:\\Users\\rsnen\\Desktop\\IBM\\Wine_Quality_Data.csv", sep=',')
```

In [5]:
```
data.head()
```

Out[5]:

	fixed_acidity	volatile_acidity	citric_acid	residual_sugar	chlorides	free_sulfur_dioxide	total_sulfur_dioxide	density	pH	sulphates	alcohol	quality	color
0	7.4	0.70	0.00	1.9	0.076	11.0	34.0	0.9978	3.51	0.56	9.4	5	red
1	7.8	0.88	0.00	2.6	0.098	25.0	67.0	0.9968	3.20	0.68	9.8	5	red
2	7.8	0.76	0.04	2.3	0.092	15.0	54.0	0.9970	3.26	0.65	9.8	5	red
3	11.2	0.28	0.56	1.9	0.075	17.0	60.0	0.9980	3.16	0.58	9.8	6	red
4	7.4	0.70	0.00	1.9	0.076	11.0	34.0	0.9978	3.51	0.56	9.4	5	red

In [6]:
```
data.dtypes
```

Out[6]:
```
fixed_acidity           float64
volatile_acidity        float64
citric_acid             float64
residual_sugar          float64
chlorides               float64
free_sulfur_dioxide     float64
total_sulfur_dioxide    float64
density                 float64
pH                      float64
sulphates               float64
alcohol                 float64
quality                   int64
color                    object
dtype: object
```

Convert the color feature to an integer. This is a quick way to do it using Pandas.

In [8]:
```
data['color'] = data.color.replace('white',0).replace('red',1).astype(np.int)
### END SOLUTION
```

- Use `StratifiedShuffleSplit` to split data into train and test sets that are stratified by wine quality. If possible, preserve the indices of the split below.
- Check the percent composition of each quality level for both the train and test data sets.

In [9]:
```
### BEGIN SOLUTION
# All data columns except for color
feature_cols = [x for x in data.columns if x not in 'color']
```

In [12]:
```
from sklearn.model_selection import StratifiedShuffleSplit

# Split the data into two parts with 1000 points in the test data
# This creates a generator
strat_shuff_split = StratifiedShuffleSplit(n_splits=1, test_size=1000, random_state=42)

# Get the index values from the generator
train_idx, test_idx = next(strat_shuff_split.split(data[feature_cols], data['color']))

# Create the data sets
X_train = data.loc[train_idx, feature_cols]
y_train = data.loc[train_idx, 'color']

X_test = data.loc[test_idx, feature_cols]
y_test = data.loc[test_idx, 'color']
```

Now check the percent composition of each quality level in the train and test data sets. The data set is mostly white wine, as can be seen below.

In [14]:
```
# normalize  will divide by the total so we get the actual percentage.
y_train.value_counts(normalize=True).sort_index()
```

Out[14]:
```
0    0.753866
1    0.246134
Name: color, dtype: float64
```

In [15]:
```
y_test.value_counts(normalize=True).sort_index()
### END SOLUTION
```

Out[15]:
```
0    0.754
1    0.246
Name: color, dtype: float64
```

- Fitting a decision tree classifier with no set limits on maximum depth, features, or leaves.
- Determine how many nodes are present and what the depth of this (very large) tree is.
- Using this tree, measure the prediction error in the train and test data sets. What do you think is going on here based on the differences in prediction error?

In [17]:
```
### BEGIN SOLUTION
from sklearn.tree import DecisionTreeClassifier

dt = DecisionTreeClassifier(random_state=42)
dt = dt.fit(X_train, y_train)
```

The number of nodes and the maximum actual depth.

In [19]:
```
# without pruning
dt.tree_.node_count, dt.tree_.max_depth
```

Out[19]:
```
(171, 22)
```

A function to return error metrics.

In [21]:
```
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score

def measure_error(y_true, y_pred, label):
    return pd.Series({'accuracy':accuracy_score(y_true, y_pred),
                      'precision': precision_score(y_true, y_pred),
                      'recall': recall_score(y_true, y_pred),
                      'f1': f1_score(y_true, y_pred)},
                      name=label)
```

In [29]:
```
def measure_error(y_true, y_pred,train):
    return pd.Series({'accuracy':accuracy_score(y_true, y_pred),
                      'precision': precision_score(y_true, y_pred),
                      'recall': recall_score(y_true, y_pred),
                      'f1': f1_score(y_true, y_pred)},
                      name=train)
```

In [30]:
```
def measure_error(y_true, y_pred,test):
    return pd.Series({'accuracy':accuracy_score(y_true, y_pred),
                      'precision': precision_score(y_true, y_pred),
                      'recall': recall_score(y_true, y_pred),
                      'f1': f1_score(y_true, y_pred)},
                      name=test)
```

The decision tree predicts a little better on the training data than the test data, which is consistent with (mild) overfitting. Also notice the perfect recall score for the training data. In many instances, this prediction difference is even greater than that seen here.

In [33]:
```
# The error on the training and test data sets
y_train_pred = dt.predict(X_train)
y_test_pred = dt.predict(X_test)

train_test_full_error = pd.concat([measure_error(y_train, y_train_pred, 'train'),
                              measure_error(y_test, y_test_pred, 'test')],
                              axis=1) # we concatenate these two together,
                                     # and we're saying we're concantenating across the columns.

train_test_full_error
### END SOLUTION
```

Out[33]:

	train	test
accuracy	0.999818	0.984000
precision	0.999261	0.963710
recall	1.000000	0.971545
f1	0.999631	0.967611

- Using grid search with cross validation, find a decision tree that performs well on the test data set. Use a different variable name for this decision tree model
- Determine the number of nodes and the depth of this tree.
- Measure the errors on the training and test sets as before and compare them to those from the tree .

we use max_depth as one of the hyperparameters and max_features as one of the parameters as well. So we have to make sure that those names match up. We pass in a list of values. So here it's going to be a range of one through the max_depth that we calculated, plus one so that we're able to include the last one counting by two, and then this is also going to be using the decision tree we classified, and the number of features that we can use are going to be up to the number of features that were used there. Generally, this is going to be all of the features if we didn't prune at all.

In [34]:
```
### BEGIN SOLUTION
from sklearn.model_selection import GridSearchCV

param_grid = {'max_depth':range(1, dt.tree_.max_depth+1, 2),
              'max_features': range(1, len(dt.feature_importances_)+1)}

GR = GridSearchCV(DecisionTreeClassifier(random_state=42),
                  param_grid=param_grid,
                  scoring='accuracy',
                  n_jobs=-1)

GR = GR.fit(X_train, y_train)
```

The number of nodes and the maximum depth of the tree.

In [37]:
```
GR.best_estimator_.tree_.node_count, GR.best_estimator_.tree_.max_depth
# smaller tree compared to earlier(171,22)
```

Out[37]:
```
(99, 7)
```

These test errors are a little better than the previous ones. So it would seem the previous example overfit the data, but only slightly so.

In [39]:
```
y_train_pred_gr = GR.predict(X_train)
y_test_pred_gr = GR.predict(X_test)

train_test_gr_error = pd.concat([measure_error(y_train, y_train_pred_gr, 'train'),
                                 measure_error(y_test, y_test_pred_gr, 'test')],
                                axis=1)
```

In [41]:
```
train_test_gr_error
#most of them we see improvements in scores
# not much improvement for accuracy
# improvement in precision.
# same for recall
#improvement in our f1 score.
```

Out[41]:

	train	test
accuracy	0.995816	0.989000
precision	0.998501	0.983539
recall	0.984479	0.971545
f1	0.991440	0.977505

- Re-split the data into `X` and `y` parts, this time with `residual_sugar` being the predicted (`y`) data. *Note:* if the indices were preserved from the `StratifiedShuffleSplit` output , they can be used again to split the data.
- Using grid search with cross validation, find a decision tree **regression** model that performs well on the test data set. *since this is regression we measure the errors on the training and test sets using **mean squared error**.
- Make a plot of actual *vs* predicted residual sugar.

In [44]:
```
### BEGIN SOLUTION
feature_cols = [x for x in data.columns if x != 'residual_sugar']

# Create the data sets
#We are then going to set x_train equal to our original data using
#train index that we defined earlier using stratified shuffle split.
X_train = data.loc[train_idx, feature_cols]
y_train = data.loc[train_idx, 'residual_sugar']

X_test = data.loc[test_idx, feature_cols]
y_test = data.loc[test_idx, 'residual_sugar']
```

In [47]:
```
from sklearn.tree import DecisionTreeRegressor

dr = DecisionTreeRegressor().fit(X_train, y_train)

param_grid = {'max_depth':range(1, dr.tree_.max_depth+1, 2),
              'max_features': range(1, len(dr.feature_importances_)+1)}

GR_sugar = GridSearchCV(DecisionTreeRegressor(random_state=42),
                     param_grid=param_grid,
                     scoring='neg_mean_squared_error', # we are maximizing the negative of mean squared error.
                      n_jobs=-1)                        #So in reality we are minimizing mean squared error

GR_sugar = GR_sugar.fit(X_train, y_train)
```

The number of nodes and the maximum depth of the tree. This tree has lots of nodes, which is not so surprising given the continuous data.

In [49]:
```
GR_sugar.best_estimator_.tree_.node_count, GR_sugar.best_estimator_.tree_.max_depth
# lot more increased in nodes ,and depth is 13 compared to earlier
```

Out[49]:
```
(2891, 13)
```

The error on train and test data sets. Since this is continuous, we will use mean squared error.

In [51]:
```
from sklearn.metrics import mean_squared_error
#we can't use the same classification matrix that we just used before.
#we use our GR_ sugar to predict our actual x_train while we would predict
#on the set that we trained on, as well as the prediction on our x_test.
y_train_pred_gr_sugar = GR_sugar.predict(X_train)
y_test_pred_gr_sugar  = GR_sugar.predict(X_test)

train_test_gr_sugar_error = pd.Series({'train': mean_squared_error(y_train, y_train_pred_gr_sugar),
                                         'test':  mean_squared_error(y_test, y_test_pred_gr_sugar)},
                                          name='MSE').to_frame().T

train_test_gr_sugar_error
```

Out[51]:

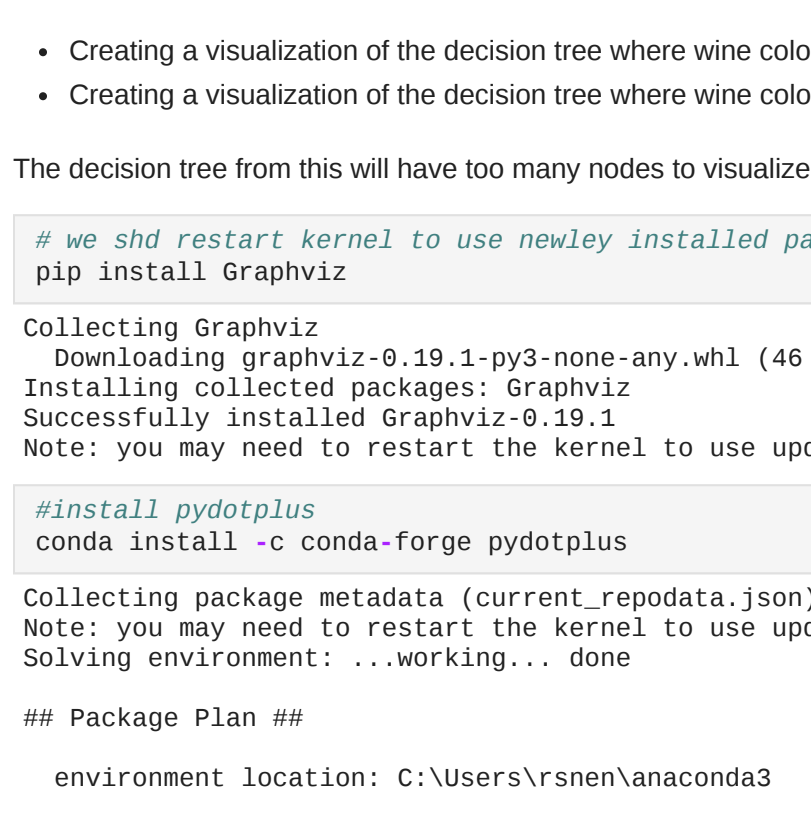
	train	test
MSE	0.401886	3.204129

A plot of actual vs predicted residual sugar.

In [54]:
```
sns.set_context('notebook')
sns.set_style('white')
fig = plt.figure(figsize=(6,6))
ax = plt.axes()

ph_test_predict = pd.DataFrame({'test':y_test.values,
                                'predict': y_test_pred_gr_sugar}).set_index('test').sort_index()

ph_test_predict.plot(marker='o', ls='', ax=ax)
ax.set(xlabel='Test', ylabel='Predict', xlim=(0,35), ylim=(0,35));
### END SOLUTION
```



Thisrequires an additional command line program (GraphViz) and Python library (PyDotPlus). GraphViz can be installed with a package manager on Linux and Mac. For PyDotPlus, either `pip` or `conda` (`conda install -c conda-forge pydotplus`) can be used to install the library.

Once these programs are installed:

- Creating a visualization of the decision tree where wine color was predicted and the number of features and/or splits are not limited.
- Creating a visualization of the decision tree where wine color was predicted but a grid search was used to find the optimal depth and number of features.

The decision tree from this will have too many nodes to visualize.

In [1]:
```
# we shd restart kernel to use newley installed packages, variables will be lost
pip install Graphviz
```

```
Collecting Graphviz
  Downloading graphviz-0.19.1-py3-none-any.whl (46 kB)
Installing collected packages: Graphviz
Successfully installed Graphviz-0.19.1
Note: you may need to restart the kernel to use updated packages.
```

In [2]:
```
#install pydotplus
conda install -c conda-forge pydotplus
```

```
Collecting package metadata (current_repodata.json): ...working... done
Note: you may need to restart the kernel to use updated packages.
Solving environment: ...working... done

## Package Plan ##

  environment location: C:\Users\rsnen\anaconda3

  added / updated specs:
    - pydotplus


The following packages will be downloaded:

    package                    |            build
    ---------------------------|-----------------
    conda-4.11.0               |   py38haa244fe_0        16.9 MB  conda-forge
    graphviz-2.38              |       hfd603c8_2        29.3 MB
    pydotplus-2.0.2            |             py_2          23 KB  conda-forge
    python_abi-3.8             |           2_cp38           4 KB  conda-forge
    ------------------------------------------------------------
                                           Total:        46.2 MB

The following NEW packages will be INSTALLED:

  graphviz           pkgs/main/win-64::graphviz-2.38-hfd603c8_2
  pydotplus          conda-forge/noarch::pydotplus-2.0.2-py_2
  python_abi         conda-forge/win-64::python_abi-3.8-2_cp38

The following packages will be UPDATED:

  conda               pkgs/main::conda-4.9.2-py38haa95532_0 --> conda-forge::conda-4.11.0-py38haa244fe_0



Downloading and Extracting Packages
```

```
==> WARNING: A newer version of conda exists. <==
  current version: 4.9.2
  latest version: 4.11.0

Please update conda by running

    $ conda update -n base -c defaults conda
```

```
conda-4.11.0         | 16.9 MB   |            |   0%
conda-4.11.0         | 16.9 MB   |            |   0%
conda-4.11.0         | 16.9 MB   | 2          |   3%
conda-4.11.0         | 16.9 MB   | 5          |   6%
conda-4.11.0         | 16.9 MB   | 8          |   8%
conda-4.11.0         | 16.9 MB   | 9          |  10%
conda-4.11.0         | 16.9 MB   | #2         |  13%
conda-4.11.0         | 16.9 MB   | #4         |  15%
conda-4.11.0         | 16.9 MB   | #7         |  18%
conda-4.11.0         | 16.9 MB   | ##1        |  21%
conda-4.11.0         | 16.9 MB   | ##3        |  24%
conda-4.11.0         | 16.9 MB   | ##5        |  26%
conda-4.11.0         | 16.9 MB   | ##8        |  28%
conda-4.11.0         | 16.9 MB   | ###1       |  31%
conda-4.11.0         | 16.9 MB   | ###4       |  34%
conda-4.11.0         | 16.9 MB   | ###7       |  38%
conda-4.11.0         | 16.9 MB   | ####2      |  42%
conda-4.11.0         | 16.9 MB   | ####8      |  49%
conda-4.11.0         | 16.9 MB   | #####2     |  53%
conda-4.11.0         | 16.9 MB   | ######1    |  61%
conda-4.11.0         | 16.9 MB   | ######8    |  68%
conda-4.11.0         | 16.9 MB   | #######4   |  75%
conda-4.11.0         | 16.9 MB   | ########   |  81%
conda-4.11.0         | 16.9 MB   | ########7  |  88%
conda-4.11.0         | 16.9 MB   | #########4 |  95%
conda-4.11.0         | 16.9 MB   | ########## | 100%

graphviz-2.38        | 29.3 MB   |            |   0%
graphviz-2.38        | 29.3 MB   |            |   0%
graphviz-2.38        | 29.3 MB   | 3          |   4%
graphviz-2.38        | 29.3 MB   | 7          |   8%
graphviz-2.38        | 29.3 MB   | #1         |  12%
graphviz-2.38        | 29.3 MB   | #5         |  16%
graphviz-2.38        | 29.3 MB   | #9         |  20%
graphviz-2.38        | 29.3 MB   | ##2        |  23%
graphviz-2.38        | 29.3 MB   | ##6        |  27%
graphviz-2.38        | 29.3 MB   | ###1       |  31%
graphviz-2.38        | 29.3 MB   | ###4       |  35%
graphviz-2.38        | 29.3 MB   | ###8       |  38%
graphviz-2.38        | 29.3 MB   | ####1      |  42%
graphviz-2.38        | 29.3 MB   | ####5      |  46%
graphviz-2.38        | 29.3 MB   | ####9      |  50%
graphviz-2.38        | 29.3 MB   | #####3     |  53%
graphviz-2.38        | 29.3 MB   | #####7     |  57%
graphviz-2.38        | 29.3 MB   | ######     |  61%
graphviz-2.38        | 29.3 MB   | ######4    |  64%
graphviz-2.38        | 29.3 MB   | ######7    |  67%
graphviz-2.38        | 29.3 MB   | ######9    |  70%
graphviz-2.38        | 29.3 MB   | #######2   |  72%
graphviz-2.38        | 29.3 MB   | #######5   |  76%
graphviz-2.38        | 29.3 MB   | #######9   |  79%
graphviz-2.38        | 29.3 MB   | ########3  |  83%
graphviz-2.38        | 29.3 MB   | ########7  |  87%
graphviz-2.38        | 29.3 MB   | #########1 |  91%
graphviz-2.38        | 29.3 MB   | #########5 |  95%
graphviz-2.38        | 29.3 MB   | #########8 |  99%
graphviz-2.38        | 29.3 MB   | ########## | 100%

pydotplus-2.0.2      | 23 KB     |            |   0%
pydotplus-2.0.2      | 23 KB     | ########## | 100%

python_abi-3.8       | 4 KB      |            |   0%
python_abi-3.8       | 4 KB      | ########## | 100%
Preparing transaction: ...working... done
Verifying transaction: ...working... done
Executing transaction: ...working... done
```

In [4]:
```
# updating conda for pydotplus if we have older version
conda update -n base -c defaults conda
```

```
Collecting package metadata (current_repodata.json): ...working... done
Note: you may need to restart the kernel to use updated packages.
Solving environment: ...working... done

## Package Plan ##


  environment location: C:\Users\rsnen\anaconda3

  added / updated specs:
    - conda


The following packages will be downloaded:

    package                    |            build
    ---------------------------|-----------------
    backports.functools_lru_cache-1.6.4|     pyhd3eb1b0_0           9 KB
    backports.tempfile-1.0     |     pyhd3eb1b0_1          11 KB
    conda-4.11.0               |   py38haa95532_0        14.4 MB
    conda-package-handling-1.7.3|   py38h8cc25b3_1         721 KB
    xmltodict-0.12.0           |     pyhd3eb1b0_0          13 KB
    ------------------------------------------------------------
                                           Total:        15.2 MB

The following packages will be REMOVED:

  python_abi-3.8-2_cp38

The following packages will be UPDATED:

  backports.functoo~                             1.6.1-py_0 --> 1.6.4-pyhd3eb1b0_0
  conda-package-han~                   1.7.2-py38h76e460a_0 --> 1.7.3-py38h8cc25b3_1

The following packages will be SUPERSEDED by a higher-priority channel:

  conda              conda-forge::conda-4.11.0-py38haa244f~ --> pkgs/main::conda-4.11.0-py38haa95532_0

The following packages will be DOWNGRADED:

  backports.tempfile                               1.0-py_1 --> 1.0-pyhd3eb1b0_1
  xmltodict                                    0.12.0-py_0 --> 0.12.0-pyhd3eb1b0_0



Downloading and Extracting Packages

backports.tempfile-1 | 11 KB     |            |   0%
backports.tempfile-1 | 11 KB     | ########## | 100%
backports.tempfile-1 | 11 KB     | ########## | 100%

conda-4.11.0         | 14.4 MB   |            |   0%
conda-4.11.0         | 14.4 MB   | 4          |   4%
conda-4.11.0         | 14.4 MB   | #3         |  13%
conda-4.11.0         | 14.4 MB   | ##1        |  22%
conda-4.11.0         | 14.4 MB   | ##8        |  29%
conda-4.11.0         | 14.4 MB   | ###6       |  36%
conda-4.11.0         | 14.4 MB   | ####3      |  43%
conda-4.11.0         | 14.4 MB   | #####2     |  52%
conda-4.11.0         | 14.4 MB   | #####9     |  59%
conda-4.11.0         | 14.4 MB   | ######5    |  65%
conda-4.11.0         | 14.4 MB   | #######2   |  72%
conda-4.11.0         | 14.4 MB   | ########   |  81%
conda-4.11.0         | 14.4 MB   | ########9  |  89%
conda-4.11.0         | 14.4 MB   | #########7 |  98%
conda-4.11.0         | 14.4 MB   | ########## | 100%

xmltodict-0.12.0     | 13 KB     |            |   0%
xmltodict-0.12.0     | 13 KB     | ########## | 100%
xmltodict-0.12.0     | 13 KB     | ########## | 100%

backports.functools_ | 9 KB      |            |   0%
backports.functools_ | 9 KB      | ########## | 100%

conda-package-handli | 721 KB    |            |   0%
conda-package-handli | 721 KB    | ########## | 100%
conda-package-handli | 721 KB    | ########## | 100%
Preparing transaction: ...working... done
Verifying transaction: ...working... done
Executing transaction: ...working... done
```

In [6]:
```
from io import StringIO
from IPython.display import Image
from sklearn.tree import export_graphviz
import pydotplus
```

The tree from question 3.

In [25]:
```
# Create an output destination for the file
dot_data = StringIO()

export_graphviz(GR.best_estimator_, out_file=dot_data, filled=True)
graph = pydotplus.graph_from_dot_data(dot_data.getvalue())

# View the tree image
filename = 'wine_tree_prune.png'
graph.write_png(filename)
Image(filename=filename)
### END SOLUTION
```

Out[25]: