Homework 5

Coding Tasks (Problem 2 & 4)

Group 45

Prachi Patel
Jignasuben Vekariya

424-394-6096 (Tel. of Prachi)
857-262-0803 (Tel of Jignasu)

patel.prachi2@northeastern.edu
vekariya.j@northeastern.edu

Percentage of Effort Contributed by Student 1: 50%

Percentage of Effort Contributed by Student 2: 50%

Signature of Student 1: Prachi Patel

Signature of Student 2: J.R.Vekariya

Submission Date: 04/08/2022

Homework_5

April 8, 2022

1 Homework 5

Before you start: Read Chapter 8 Naive Bayes and Chapter 9 Decision Trees in the textbook.

Note: Please enter the code along with your comments in the **TODO** section.

Alternative solutions are always welcomed.

1.0.1 Problem 2

is problem, we need to build a Naive Bayes model to classify whether a movie review is positive or negative.

The given data is a subset of the IMDB movie review dataset.

This might be your first time working with text mining. Therefore, the basic pre-processing steps are given below.

You have two major tasks:

- Go through the code and get to know the purpose of each preprocessing step. Summarize what a preprocessing step does when required.
- Build a multinomial Naive Bayes model to classify the reviews.

[ ]:
```
# # Please remove # and run the following code if you have an error while␣
↪importing the dataset
# !pip install --upgrade openpyxl
```

[ ]:
```
# Libraries used for the assignment
import pandas as pd           #Pandas library use for Data related task like␣
↪analysis, in Data science and Machine learning also
import numpy as np            #Numpy library use for perform mathematical task␣
↪on array
import matplotlib.pyplot as py   #Matplotlib use for visualization purpose
import seaborn as sns             #Seaborn use for make graphs in statistic

#plotly use for web base visualization
import plotly.graph_objs as gp_obj
import plotly.express as exp
from plotly.subplots import make_subplots
```

```
import math        #Math library use to perform basic maths function
import random      #Random library use for random number generation
```

skleran is mostly use for machine learning task like Build algorithms & model, confusion matrics, data operations

[ ]:
```
from sklearn.model_selection import RandomizedSearchCV
from sklearn.pipeline import Pipeline
from sklearn.linear_model import Ridge
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import train_test_split
from collections import Counter
import scipy.stats as sts
import sklearn.preprocessing as sp
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import confusion_matrix,classification_report
from sklearn.model_selection import cross_val_score, GridSearchCV
from sklearn.metrics import classification_report, confusion_matrix,␣
↪accuracy_score
from sklearn import svm
from sklearn.model_selection import train_test_split
from sklearn.feature_extraction.text import CountVectorizer, TfidfTransformer
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.feature_extraction.text import TfidfTransformer
from sklearn.multiclass import OneVsOneClassifier
from sklearn.svm import SVC
from sklearn.metrics import confusion_matrix
```

[ ]:
```
#!pip3 install catboost
import warnings
warnings.filterwarnings('ignore')
sns.set_style('darkgrid')
cmap = sns.cm.mako_r
%matplotlib inline
from sklearn.svm import SVC
from statsmodels.stats.outliers_influence import variance_inflation_factor
from sklearn import metrics
from sklearn.metrics import r2_score
from statsmodels.stats.outliers_influence import variance_inflation_factor
from sklearn.metrics import mean_squared_error
from sklearn.neighbors import KNeighborsRegressor
from sklearn.ensemble import␣
↪BaggingRegressor,AdaBoostRegressor,GradientBoostingRegressor
from sklearn.decomposition import PCA
from sklearn.pipeline import make_pipeline
from sklearn.preprocessing import StandardScaler
```

[ ]:
```
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
from sklearn import tree
from sklearn.datasets import load_iris
from sklearn.model_selection import cross_val_score
from sklearn.tree import DecisionTreeClassifier
from sklearn.naive_bayes import MultinomialNB
```

[ ]:
```
#Importing Dataset
from google.colab import files
file = files.upload()
df = pd.read_csv("IMDB Dataset_subset.csv")
df.head()
```

```
<IPython.core.display.HTML object>
```

```
Saving IMDB Dataset_subset.csv to IMDB Dataset_subset.csv
```

[ ]:
```
                                              review sentiment
0  One of the other reviewers has mentioned that …  positive
1  A wonderful little production. <br /><br />The…  positive
2  I thought this was a wonderful way to spend ti…  positive
3  Basically there's a family where a little boy …  negative
4  Petter Mattei's "Love in the Time of Money" is…  positive
```

[ ]:
```
# Packages required for preprocessing #
from sklearn.feature_extraction.text import CountVectorizer
from nltk.stem import WordNetLemmatizer   #for lemmatization
import re   #regular expression package
import nltk
from nltk.corpus import stopwords
nltk.download('stopwords')
nltk.download('wordnet')
```

```
[nltk_data] Downloading package stopwords to /root/nltk_data…
[nltk_data]   Unzipping corpora/stopwords.zip.
[nltk_data] Downloading package wordnet to /root/nltk_data…
[nltk_data]   Unzipping corpora/wordnet.zip.
```

[ ]: True

[ ]:
```
X = [row for row in df['review']] #list of reviews
classes = df['sentiment'] #list of true classes
```

[ ]:
```
# Pre-process the data
reviews = []
lmz = WordNetLemmatizer()

for review in range(0, len(X)):
```

```
    # part 1
    review = re.sub(r'[\W_]', ' ', str(X[review]))
    review = re.sub(r'\s+[a-zA-Z]\s+', ' ', review)
    review = re.sub(r'\^[a-zA-Z]\s+', ' ', review)
    review = re.sub(r'\s+', ' ', review, flags=re.I)
    review = re.sub(r'^b\s+', '', review) # if a review record is in bytes, the␣
↪corresponding line will have a letter 'b' appended at the start)
    review = review.lower()
    review = re.sub(r'[0-9]+', '', review)

    # part 2
    review = review.split()
    review = [lmz.lemmatize(word) for word in review]
    review = ' '.join(review)

    reviews.append(review)
```

TODO 1

Explain the function that part 1 and part 2 achieve in the loop.

Part 1 Explaination 1. In this case we are preprecessing the data,by removing spaces,changing the cases and removing numbers from the string. 2. Non numeric charecters are replaced. 3. Omitting all single characters from the dataset review. 4. Single character followed by empty space is substituted by emptyspace by this line of code. 5. s+ refers to removing multiple spaces and ages = re.I means Ignorecase. lower() converts data into lowercase. 6. Repetative numbers get replaced by space

Part 2 Explaination 1. split sentence into single words 2. Lemmatizing is use for sorting the words by grouping with same word and covert into base form, so, it provides list of real vocabulary. 3. by using join functio,it words and store it in review then add review at the end of the reviews.

[ ]:
```
vect = CountVectorizer(stop_words = "english", max_df=0.7, min_df=5)
text = vect.fit_transform(reviews).toarray()
vocab = vect.vocabulary_
vocab = sorted(vocab.items(), key = lambda x: x[1])
vocab = [v[0] for v in vocab]
```

TODO 2

What do “texts” and “vocab” represent? What is the relationship between them?

“texts” is used to store the words transformed into vectors from the reviews into an array, it can also be used to convert list of words into math sign 0 or 1 to get a count of words in each record of reviews.

“vocab” is used for encoding unseen texts later, it is an encoded vector that has been sorted based on the number of appearances in the “texts”.Vocab can also be used to find the position of the words in text matrix. vocab gives list of words in alphabetical order.

TODO 3

Partition the data into 80% training and 20% validation set.

```
[ ]: df_train, df_val = train_test_split(df, test_size=0.2, shuffle=True)
     X_train = vect.transform(df_train.review)                    # Splitting␣
      ↪the data onto trainig and validation sets into 80% and 20%
     X_val = vect.transform(df_val.review)
     X_train.shape, X_val.shape
```

```
[ ]: ((3200, 8396), (800, 8396))
```

TODO 4

Build a multinomial Naive Bayes model on the training set.

```
[ ]: vect_count= CountVectorizer(binary=False).fit(df.review)
     X_train_count = vect_count.transform(df_train.review)         #Count vectorizer␣
      ↪used to transform text into vector
     X_val_count = vect_count.transform(df_val.review)
     MNB = MultinomialNB().fit(X_train_count, df_train.sentiment)   #Making the data␣
      ↪usable for application of Multinomial Naive Bayes
```

Hint: Multinomial Naive Bayes with sklearn

TODO 5

Evaluate the model performance with the training and validation set. Comment on the model performance.

```
[ ]: #obtaining model performance summary
     X_val_count = vect_count.transform(df_val.review)
     predicts = MNB.predict(X_val_count)
     print(classification_report(df_val.sentiment, predicts))
```

```
              precision    recall  f1-score   support

    negative       0.82      0.90      0.86       402
    positive       0.89      0.80      0.84       398

    accuracy                           0.85       800
   macro avg       0.85      0.85      0.85       800
weighted avg       0.85      0.85      0.85       800
```

The model has good f1 score with high values for recall and precision indicating that the model is performing well and there is no overfitting.

Hint: Classification report with sklearn

If you are interested (this part is not graded):

Explore one or two records that were misclassified. Check the original text, vectorized text, and comment on the possible reason why the record got misclassified.

[ ]:

```

```

1.0.2 Problem 4

The wine dataset is the result of a chemical analysis of wines grown in the same region in Italy but derived from three different cultivars. (For illustration simplicity purpose, only 2 classes, 0 and 1, will be included for the classification task.) The analysis determined the quantities of 13 constituents found in each of the three types of wines.

The objective is to classify the wines into class 0 or 1 using the 13 given attributes and a decision tree classifier.

[ ]:

```
from sklearn import datasets

# load the wine dataset
wine = datasets.load_wine()
print(wine.DESCR)

# convert the data into dataframe format
X = pd.DataFrame(wine['data'], columns = wine['feature_names'])
y = wine['target']

# only consider wine class 0 and 1
X = X.loc[0:129, :]
y = y[0:130]

X.head()
```

```
.. _wine_dataset:

Wine recognition dataset
------------------------

**Data Set Characteristics:**

    :Number of Instances: 178 (50 in each of three classes)
    :Number of Attributes: 13 numeric, predictive attributes and the class
    :Attribute Information:
                - Alcohol
                - Malic acid
                - Ash
                - Alcalinity of ash
                - Magnesium
                - Total phenols
                - Flavanoids
                - Nonflavanoid phenols
                - Proanthocyanins
                - Color intensity
                - Hue
```

```
            - OD280/OD315 of diluted wines
            - Proline

    - class:
            - class_0
            - class_1
            - class_2

    :Summary Statistics:

    ============================= ==== ===== ======= =====
                                   Min   Max   Mean     SD
    ============================= ==== ===== ======= =====
    Alcohol:                      11.0  14.8    13.0   0.8
    Malic Acid:                   0.74  5.80    2.34  1.12
    Ash:                          1.36  3.23    2.36  0.27
    Alcalinity of Ash:            10.6  30.0    19.5   3.3
    Magnesium:                    70.0 162.0    99.7  14.3
    Total Phenols:                0.98  3.88    2.29  0.63
    Flavanoids:                   0.34  5.08    2.03  1.00
    Nonflavanoid Phenols:         0.13  0.66    0.36  0.12
    Proanthocyanins:              0.41  3.58    1.59  0.57
    Colour Intensity:              1.3  13.0     5.1   2.3
    Hue:                          0.48  1.71    0.96  0.23
    OD280/OD315 of diluted wines: 1.27  4.00    2.61  0.71
    Proline:                       278  1680     746   315
    ============================= ==== ===== ======= =====

    :Missing Attribute Values: None
    :Class Distribution: class_0 (59), class_1 (71), class_2 (48)
    :Creator: R.A. Fisher
    :Donor: Michael Marshall (MARSHALL%PLU@io.arc.nasa.gov)
    :Date: July, 1988

This is a copy of UCI ML Wine recognition datasets.
https://archive.ics.uci.edu/ml/machine-learning-databases/wine/wine.data

The data is the results of a chemical analysis of wines grown in the same
region in Italy by three different cultivators. There are thirteen different
measurements taken for different constituents found in the three types of
wine.

Original Owners:

Forina, M. et al, PARVUS -
An Extendible Package for Data Exploration, Classification and Correlation.
Institute of Pharmaceutical and Food Analysis and Technologies,
Via Brigata Salerno, 16147 Genoa, Italy.
```

```
Citation:

Lichman, M. (2013). UCI Machine Learning Repository
[https://archive.ics.uci.edu/ml]. Irvine, CA: University of California,
School of Information and Computer Science.

.. topic:: References

  (1) S. Aeberhard, D. Coomans and O. de Vel,
  Comparison of Classifiers in High Dimensional Settings,
  Tech. Rep. no. 92-02, (1992), Dept. of Computer Science and Dept. of
  Mathematics and Statistics, James Cook University of North Queensland.
  (Also submitted to Technometrics).

  The data was used with many others for comparing various
  classifiers. The classes are separable, though only RDA
  has achieved 100% correct classification.
  (RDA : 100%, QDA 99.4%, LDA 98.9%, 1NN 96.1% (z-transformed data))
  (All results using the leave-one-out technique)

  (2) S. Aeberhard, D. Coomans and O. de Vel,
  "THE CLASSIFICATION PERFORMANCE OF RDA"
  Tech. Rep. no. 92-01, (1992), Dept. of Computer Science and Dept. of
  Mathematics and Statistics, James Cook University of North Queensland.
  (Also submitted to Journal of Chemometrics).
```

[ ]:

```
   alcohol  malic_acid   ash  alcalinity_of_ash  magnesium  total_phenols  \
0    14.23        1.71  2.43               15.6      127.0           2.80
1    13.20        1.78  2.14               11.2      100.0           2.65
2    13.16        2.36  2.67               18.6      101.0           2.80
3    14.37        1.95  2.50               16.8      113.0           3.85
4    13.24        2.59  2.87               21.0      118.0           2.80

   flavanoids  nonflavanoid_phenols  proanthocyanins  color_intensity   hue  \
0        3.06                  0.28             2.29             5.64  1.04
1        2.76                  0.26             1.28             4.38  1.05
2        3.24                  0.30             2.81             5.68  1.03
3        3.49                  0.24             2.18             7.80  0.86
4        2.69                  0.39             1.82             4.32  1.04

   od280/od315_of_diluted_wines  proline
0                          3.92   1065.0
1                          3.40   1050.0
2                          3.17   1185.0
3                          3.45   1480.0
```

```
4                              2.93    735.0
```

TODO 1

Partition the data into 70% training and 30% validation set.

```
[ ]: list_y = list(y)
     df = pd.DataFrame(list_y, columns =['target'])
     x_train,x_val,y_train,y_val = train_test_split(X,df, test_size = 0.3,␣
      ↪random_state=10)   #split data into 70%-30%
     print(x_train.shape,x_val.shape,y_train.shape,y_val.shape)
```

```
(91, 13) (39, 13) (91, 1) (39, 1)
```

TODO 2

Fit a decision tree classifier on the training set with no pruning.

Plot the tree with the following requirements:

- The node with splitting rule should contain variable name instead of variable index.
- Pick the appropriate information to present in the node. The node should be of appropriate size so the information is clear for viewing.
- The node should be colored.

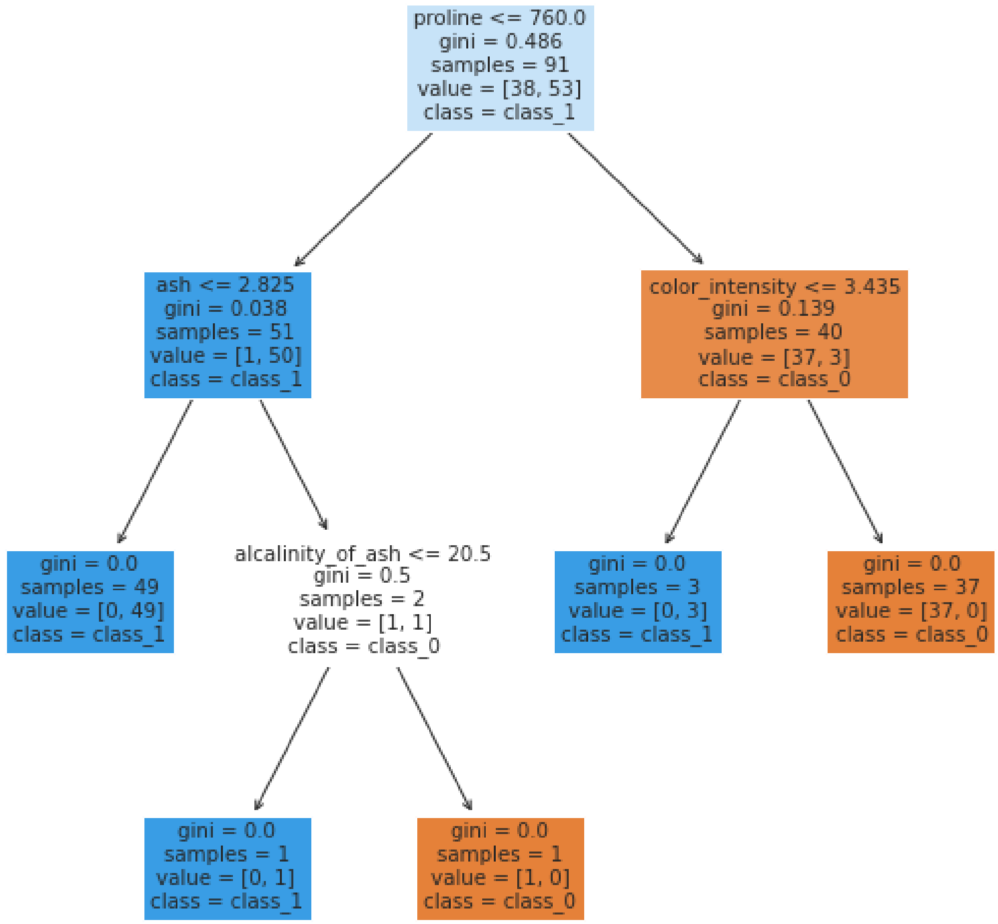
```
[ ]: Dtc = DecisionTreeClassifier()
     cross_val_score(Dtc, x_train, y_train, cv=10)
```

```
[ ]: array([0.9       , 1.        , 0.88888889, 1.        , 0.88888889,
            1.        , 1.        , 1.        , 0.77777778, 1.        ])
```

```
[ ]: Dtc.fit(x_train, y_train)
     acc_deci_tree = round(Dtc.score(x_val, y_val) * 100, 2)
     acc_deci_tree
```

```
[ ]: 92.31
```

```
[ ]: fig = py.figure(figsize = (10, 10))
     a= tree.plot_tree(Dtc,feature_names=wine.feature_names, class_names = wine.
      ↪target_names, filled = True)
```



Hint: Decision tree classifier with sklearn

TODO 3

Prune the tree with cost complexity. What is the best ccp value? Use visualization to back up your decision.

Plot the pruned tree in the same manner as TODO 2.

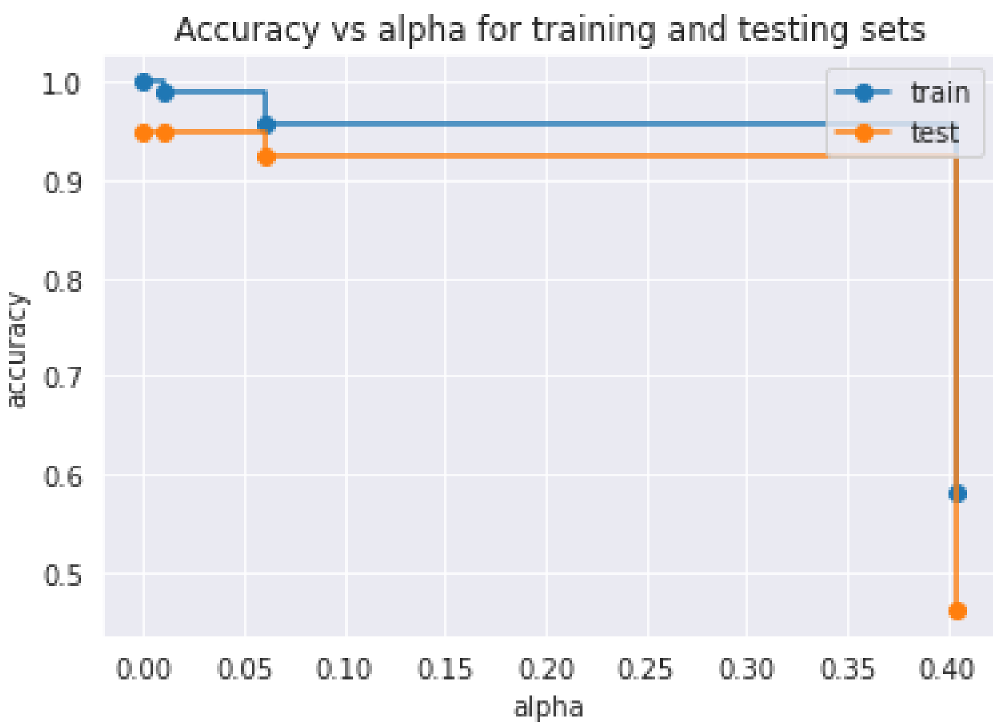
[ ]:
```
Dtc = DecisionTreeClassifier(random_state=40)
path = Dtc.cost_complexity_pruning_path(x_train, y_train)
ccp_alphas, impurities = path.ccp_alphas, path.impurities
ccp_alphas
```

[ ]:
```
array([0.        , 0.01077354, 0.06098901, 0.40387859])
```

[ ]:
```
#Determine the nodes for the different values of ccp
Nds = []
for ccp_alpha in ccp_alphas:
  Dtc2 = DecisionTreeClassifier(random_state=40, ccp_alpha=ccp_alpha)
  Dtc2.fit(x_train, y_train)
  Nds.append(Dtc2)
print("Number of nodes in the last tree is: {} with ccp_alpha: {}".
 ↪format(Nds[-1].tree_.node_count, ccp_alphas[-1]))
```

```
Number of nodes in the last tree is: 1 with ccp_alpha: 0.4038785928572613
```

[ ]:
```
train_score = [Dt2.score(x_train, y_train) for Dt2 in Nds]
test_score = [Dt2.score(x_val, y_val) for Dt2 in Nds]
fig, ax = py.subplots()
ax.set_xlabel("alpha")
ax.set_ylabel("accuracy")
ax.set_title("Accuracy vs alpha for training and testing sets")
ax.plot(ccp_alphas, train_score, marker='o', label="train",␣
 ↪drawstyle="steps-post")
ax.plot(ccp_alphas, test_score, marker='o', label="test",␣
 ↪drawstyle="steps-post")
ax.legend()
py.show()
```

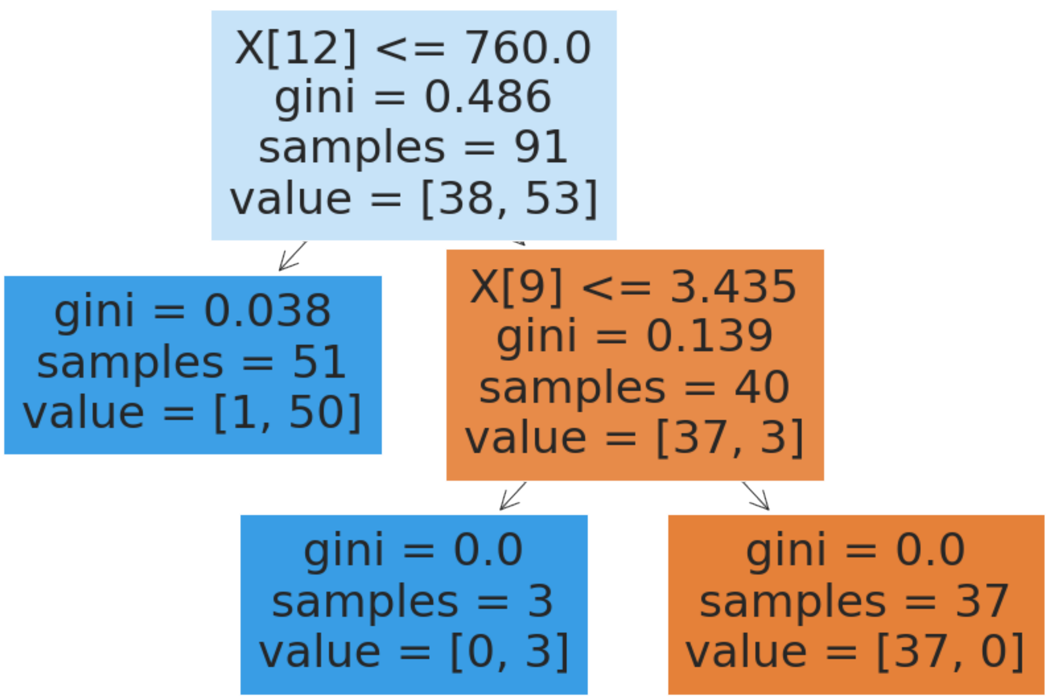


[ ]:
```
Dtc3 = DecisionTreeClassifier(random_state=10, ccp_alpha=0.012)
Dtc3.fit(x_train,y_train)
pred=Dtc3.predict(x_val)
from sklearn.metrics import accuracy_score
accuracy_score(y_val, pred)
```

[ ]:
```
0.9487179487179487
```

[ ]:
```
#plot decision tree
py.figure(figsize=(15,10))
tree.plot_tree(Dtc3,filled=True)
```

[ ]:
```
[Text(0.4, 0.8333333333333334, 'X[12] <= 760.0\ngini = 0.486\nsamples =
91\nvalue = [38, 53]'),
 Text(0.2, 0.5, 'gini = 0.038\nsamples = 51\nvalue = [1, 50]'),
 Text(0.6, 0.5, 'X[9] <= 3.435\ngini = 0.139\nsamples = 40\nvalue = [37, 3]'),
 Text(0.4, 0.16666666666666666, 'gini = 0.0\nsamples = 3\nvalue = [0, 3]'),
 Text(0.8, 0.16666666666666666, 'gini = 0.0\nsamples = 37\nvalue = [37, 0]')]
```



Hint: Minimal cost complexity pruning

Post pruning decision trees with cost complexity with sklearn