# Department of Computing

**CS471: Machine Learning**

**Class: BESE-7AB**

**Lab 07: Decision Tree**

**CL03: Apply a variety of learning algorithms to data for solution development**

**Date: 14-03-2019**

**Time: 10:00 am– 1:00 pm & 2:00 pm-5:00 pm**

**Instructor: Dr. Pakeeza Akram**

KASHIF NAZIR KHAN

193105

BESE 7B

# Lab 07: Decision Tree

**Introduction**

Decision Trees (DTs) are a non-parametric supervised learning method used for [classification](http://scikit-learn.org/stable/modules/tree.html#tree-classification) and [regression](http://scikit-learn.org/stable/modules/tree.html#tree-regression). The goal is to create a model that predicts the value of a target variable by learning simple decision rules inferred from the data features. Decision trees learn from data to approximate a sine curve with a set of if-then-else decision rules. The deeper the tree, the more complex the decision rules and the fitter the model.

**Objective**

In this lab you will use cleaned IMDB reviews from lab 6 to train a classifier. Each movie review is labeled with a positive or a negative sentiment. This is your training dataset. You will use cleaned reviews with labels to train a Decision Tree. The trained classifier when presented with a new review will predict if the review is positive or negative.

**Tools/Software Requirement**

Python, scikit-learn

**Reference**

<https://www.kaggle.com/c/word2vec-nlp-tutorial/details/part-1-for-beginners-bag-of-words>

http://scikit-learn.org/stable/modules/tree.html

**Description**

In the below tutorial there is an example of Decision tree classifier using iris data set.

[**DecisionTreeClassifier**](http://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeClassifier.html#sklearn.tree.DecisionTreeClassifier) is a class capable of performing multi-class classification on a dataset.

As with other classifiers, **[DecisionTreeClassifier](http://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeClassifier.html" \l "sklearn.tree.DecisionTreeClassifier" \o "sklearn.tree.DecisionTreeClassifier)** takes as input two arrays: an array X, sparse or dense, of size [n\_samples, n\_features] holding the training samples, and an array Y of integer values, size [n\_samples], holding the class labels for the training samples:

**>>> from** **sklearn** **import** tree

**>>>** X = [[0, 0], [1, 1]]

**>>>** Y = [0, 1]

**>>>** clf = tree.DecisionTreeClassifier()

**>>>** clf = clf.fit(X, Y)

After being fitted, the model can then be used to predict the class of samples:

**>>>** clf.predict([[2., 2.]])

array([1])

Alternatively, the probability of each class can be predicted, which is the fraction of training samples of the same class in a leaf:

**>>>** clf.predict\_proba([[2., 2.]])

array([[ 0., 1.]])

[**DecisionTreeClassifier**](http://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeClassifier.html#sklearn.tree.DecisionTreeClassifier) is capable of both binary (where the labels are [-1, 1]) classification and multiclass (where the labels are [0, …, K-1]) classification.

Using the Iris dataset, we can construct a tree as follows:

**>>> from** **sklearn.datasets** **import** load\_iris

**>>> from** **sklearn** **import** tree

**>>>** iris = load\_iris()

**>>>** clf = tree.DecisionTreeClassifier()

**>>>** clf = clf.fit(iris.data, iris.target)

Once trained, we can export the tree in [Graphviz](http://www.graphviz.org/) format using the **[export\_graphviz](http://scikit-learn.org/stable/modules/generated/sklearn.tree.export_graphviz.html" \l "sklearn.tree.export_graphviz" \o "sklearn.tree.export_graphviz)** exporter. If you use the [conda](http://conda.io/) package manager, the graphviz binaries and the python package can be installed with

conda install python-graphviz

Alternatively binaries for graphviz can be downloaded from the graphviz project homepage, and the Python wrapper installed from pypi with pip install graphviz.

Below is an example graphviz export of the above tree trained on the entire iris dataset; the results are saved in an output file iris.pdf:

**>>> import** **graphviz**

**>>>** dot\_data = tree.export\_graphviz(clf, out\_file=**None**)

**>>>** graph = graphviz.Source(dot\_data)

**>>>** graph.render("iris")

The **[export\_graphviz](http://scikit-learn.org/stable/modules/generated/sklearn.tree.export_graphviz.html" \l "sklearn.tree.export_graphviz" \o "sklearn.tree.export_graphviz)** exporter also supports a variety of aesthetic options, including coloring nodes by their class (or value for regression) and using explicit variable and class names if desired. Jupyter notebooks also render these plots inline automatically:

**>>>** dot\_data = tree.export\_graphviz(clf, out\_file=**None**,

feature\_names=iris.feature\_names,

class\_names=iris.target\_names,

filled=True, rounded=True,

special\_characters=True)

**>>>** graph = graphviz.Source(dot\_data)

**>>>** graph

After being fitted, the model can then be used to predict the class of samples:

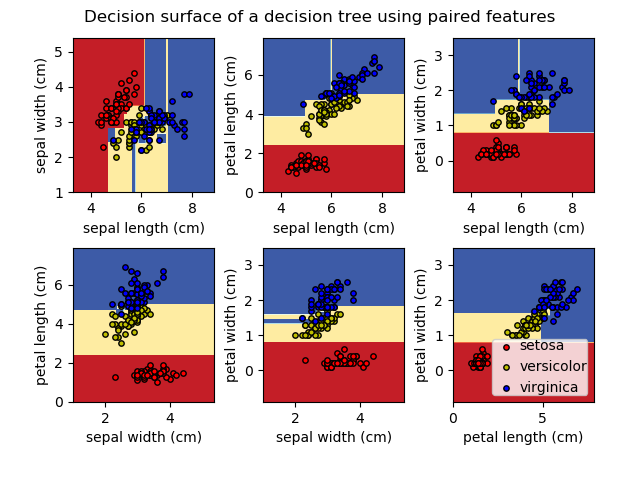
**>>>** clf.predict(iris.data[:1, :])

array([0])

Alternatively, the probability of each class can be predicted, which is the fraction of training samples of the same class in a leaf:

**>>>** clf.predict\_proba(iris.data[:1, :])

array([[ 1., 0., 0.]])

[](http://scikit-learn.org/stable/auto_examples/tree/plot_iris.html)

**Lab Task**

1. In the previous Lab you learnt how to clean data. Your lab task is to train a decision tree classifier using IMDB movie reviews (25000 movie reviews). Remember to clean the reviews before building bag of word representation. You can either use the function you wrote in the last lab to clean the reviews or use review\_to\_words.py.

Code:

import re

from bs4 import BeautifulSoup

import nltk

from nltk.corpus import stopwords

import numpy as np

import pandas as pd

from sklearn.feature\_extraction.text import CountVectorizer

train = pd.read\_csv('labeledTrainData.tsv', header=0, delimiter='\t', quoting=3)

all\_reviews = []

def review\_to\_words( raw\_review ):

# Function to convert a raw review to a string of words

# The input is a single string (a raw movie review), and

# the output is a single string (a preprocessed movie review)

#

# 1. Remove HTML

review\_text = BeautifulSoup(raw\_review).get\_text()

#

# 2. Remove non-letters

letters\_only = re.sub("[^a-zA-Z]", " ", review\_text)

#

# 3. Convert to lower case, split into individual words

words = letters\_only.lower().split()

#

# 4. In Python, searching a set is much faster than searching

# a list, so convert the stop words to a set

stops = set(stopwords.words("english"))

#

# 5. Remove stop words

meaningful\_words = [w for w in words if not w in stops]

#

# 6. Join the words back into one string separated by space,

# and return the result.

return( " ".join( meaningful\_words ))

'''def word\_extraction(sentence):

ignore = ['a', "the", "is"]

words = re.sub("[^\w]", " ", sentence).split()

cleaned\_text = [w.lower() for w in words if w not in ignore]

return cleaned\_text

def tokenize(sentences):

words = []

for sentence in sentences:

w = word\_extraction(sentence)

words.extend(w)

words = sorted(list(set(words)))

print(words[0])

return words

def generate\_bow(allsentences):

vocab = tokenize(allsentences)

full\_bag\_vector = []

#print("Word List for Document \n{0} \n".format(vocab));

for sentence in allsentences:

words = word\_extraction(sentence)

bag\_vector = np.zeros(len(vocab))

for w in words:

for i,word in enumerate(vocab):

if word == w:

bag\_vector[i] += 1

#print("{0}\n{1}\n".format(sentence,numpy.array(bag\_vector)))

full\_bag\_vector.append(bag\_vector)

'''

for review in train["review"]:

all\_reviews.append(review\_to\_words(review))

train["review"] = all\_reviews

vectorizer = CountVectorizer()

X = vectorizer.fit\_transform(train["review"][0:500])

from sklearn import tree

X = X.toarray()

Y = train["sentiment"][0:500]

clf = tree.DecisionTreeClassifier()

clf = clf.fit(X, Y)

vector = vectorizer.transform(['bad movie']).toarray()

clf.predict(vector)

test = vectorizer.transform(train["review"][10000:20000]).toarray()

yhat\_test = clf.predict(test)

# Compute accuracy based on test samples

result = accuracy\_score((train["sentiment"][10000:20000]), yhat\_test)

print(result)

import graphviz

dot\_data = tree.export\_graphviz(clf, out\_file=None)

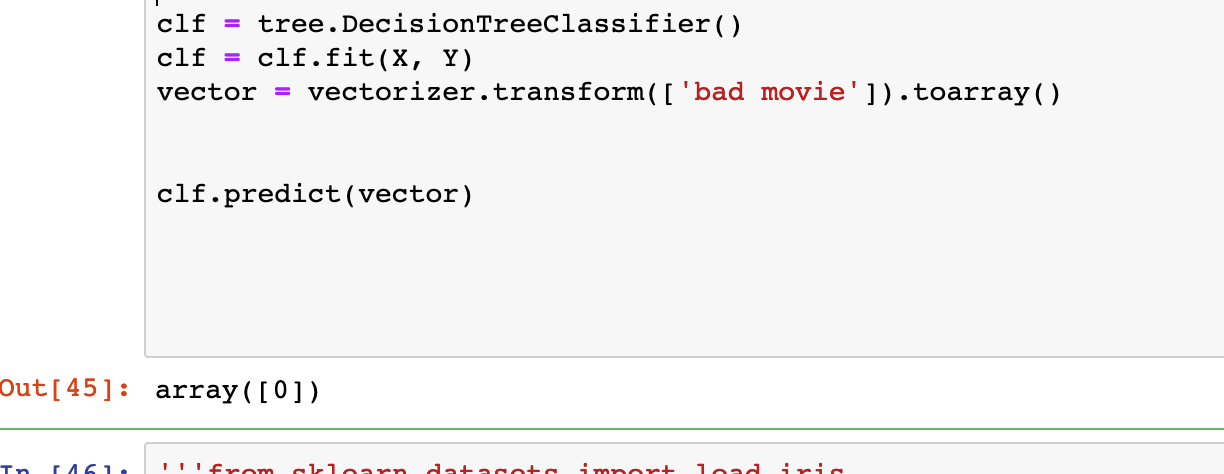
graph = graphviz.Source(dot\_data)

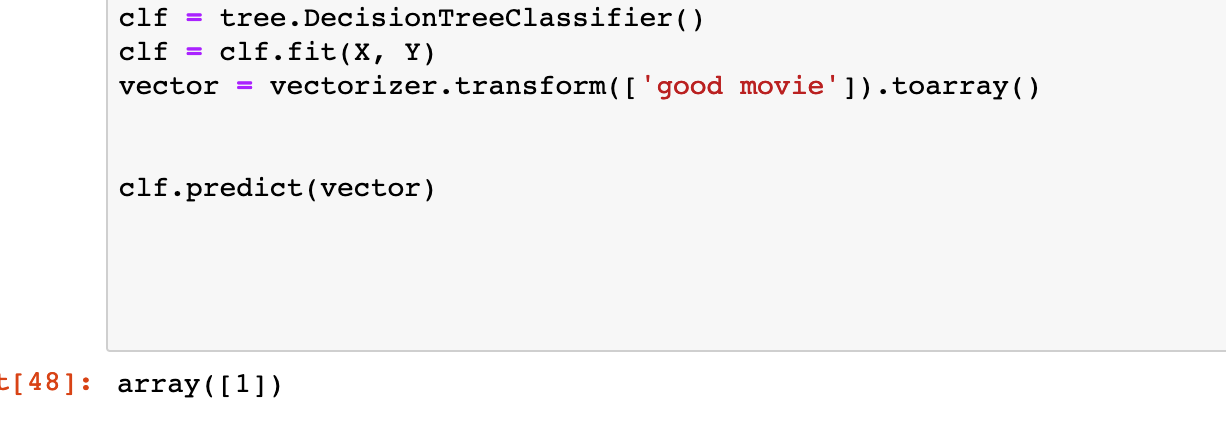
dot\_data = tree.export\_graphviz(clf)

graph = graphviz.Source(dot\_data)

graph

Negative Sentiment Prediction:





1. Calculate accuracy on different vocabulary sizes.

Using 0:500 and testing on 500 : 1000 gives average of 0.666.



Using 0:5000 and testing on 10000 : 15000 gives average of 0.6912.



Using 0:10000 and testing on 10000 : 20000 gives average of 0.7089.



**Deliverables**

Upload Word file containing the task

**Reference**

http://scikit-learn.org/stable/modules/tree.html