|  |
| --- |
|  |
| Harjoitus 2 |
| Raportti |
|  |
| **Aleksi Haapsaari 500039 Ville Ahti 79062** |
| **24.10.2013** |

|  |
| --- |
|  |

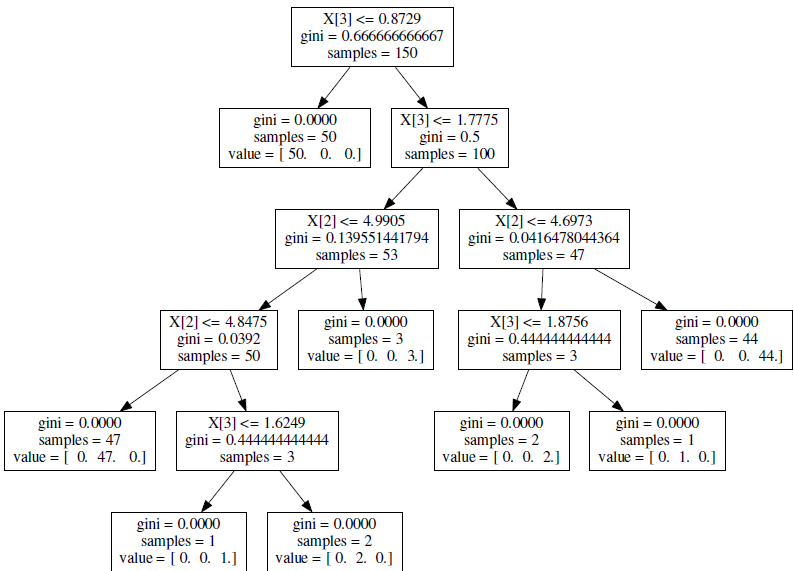
# 1. Puuttuvan arvon ennustaminen

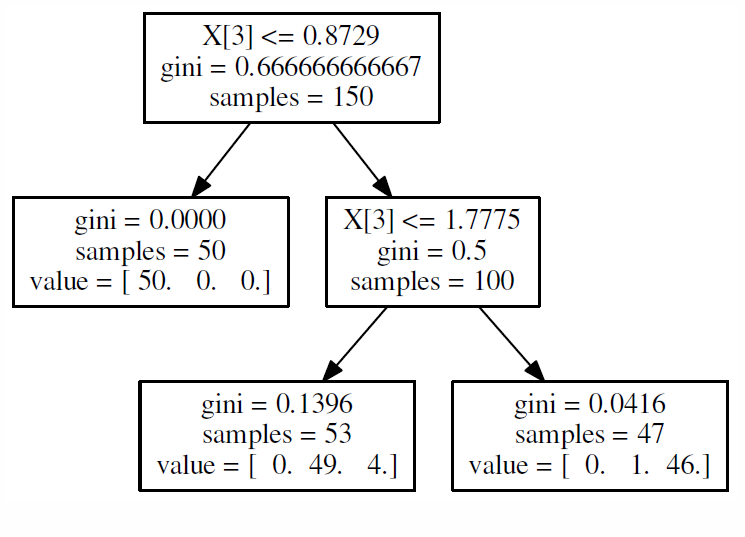
Käytimme puuttuvan arvon päättelemiseen k:n lähimmän naapurinarvojen keskiarvoa. Etsimme datasta 5 lähintä naapuria ja laskimme niiden arvoista keskiarvon, jonka sijoitimme puuttuvan arvon tilalle. Käytimme lähimpien naapureiden löytämiseen kolmea arvoa, jotka löytyivät kaikista datapisteistä.

Saadut 5 arvoa ovat 3.4776441, 3.2564541, 3.1618804, 3.8657509, 3.0560231.

Arvioitu puuttuva arvo on **3.36355052.**

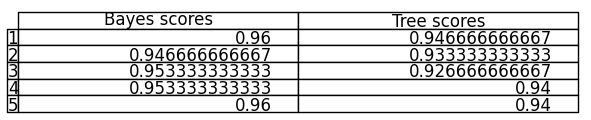
# 2. Datan luokittelu päätöspuun avulla



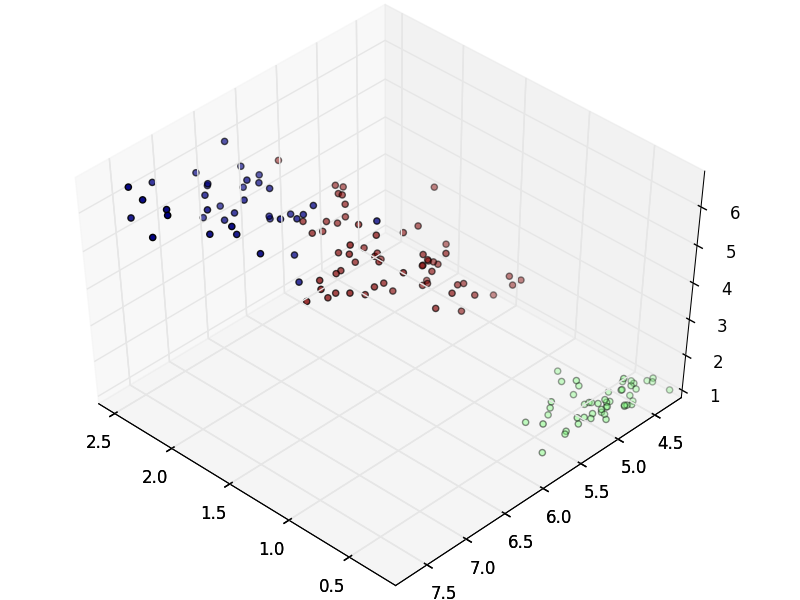
Päätöspuun luomiseen käytimme scikit-learn-kirjaston valmista funktiota ja puun visualisointiin graphviz-työkalua. Puun luominen työkaluilla oli triviaalia, mutta lopputuloksesta oli havaittavissa mallin ylisovittamista. Tästä johtuen päätöspuun luomisessa olisi hyvä jollain tavalla rajoittaa solumujen määrää. Päädyimme rajoittaamaan puun syvyyttä, sillä huomasimme kuvasta, että virheluokitteluiden lukumäärä oli tarpeeksi pieni jo toisella tasolla. Tällä rajoituksella saatiin huomattavasti mielekkäämpi päätöspuu. 

# 3. Datan luokittelu Naïve Bayes -luokittelijalla

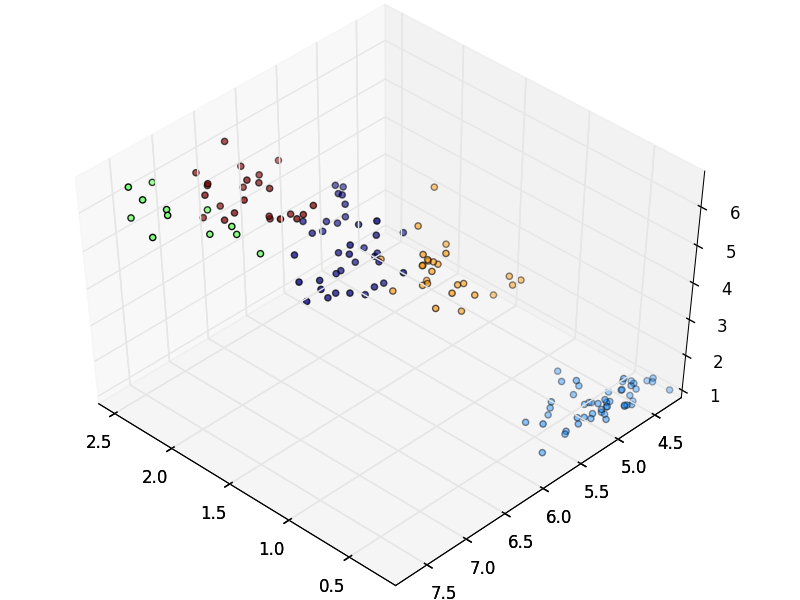
Naïve Bayes -luokittelijan luontiin käytimme scikit-learn-kirjaston GaussianNB-luokkaa, joka soveltuu parhaiten numeeriselle datalle ja olettaa, että se noudattaa normaalijakaumaa. Käytimme luokittelijan arvioimiseen n-fold cross-validation -menetelmää, jossa data jaettiin 10 yhtä satunnaiseen suureen osaan. Näistä osista jokaista käytettiin vuorollaan luokittelijan testaamiseen, kun muut osat käytettiin luokittelijan opettamiseen. Luokittelijan testaamisesta saatiin sen tarkkuutta kuvaavia arvoja, joista laskimme keskiarvon. Tätä keskiarvoa voidaan pitää luokittelujen yleisenä kykynä luokittella dataa.

Useiden toistojen jälkeen oli haivaittavissa, että Bayes-luokittelijan tarkkuus on noin 95 %. Vertasimme Bayes-luokittelijaa myös päätöspuuhun ja havaitsimme Bayes-luokkitelijan hieman tarkempi. Seuraavassa kuvassa on luokittelijoiden keskimääräiset tarkkuudet viideltä eri ajokerralta.

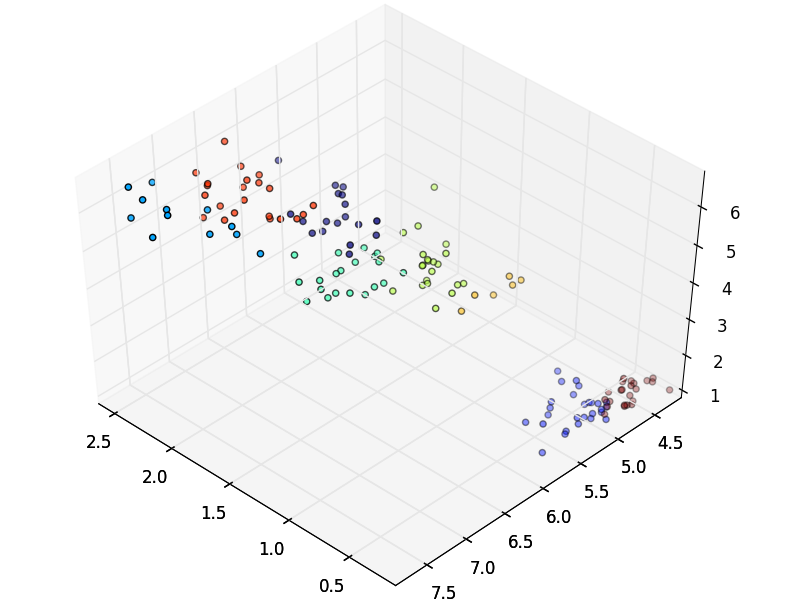
# 4. K-means klusterointi

Klusteroimme dataa k:n arvoilla 3,5,8 ja 10. 

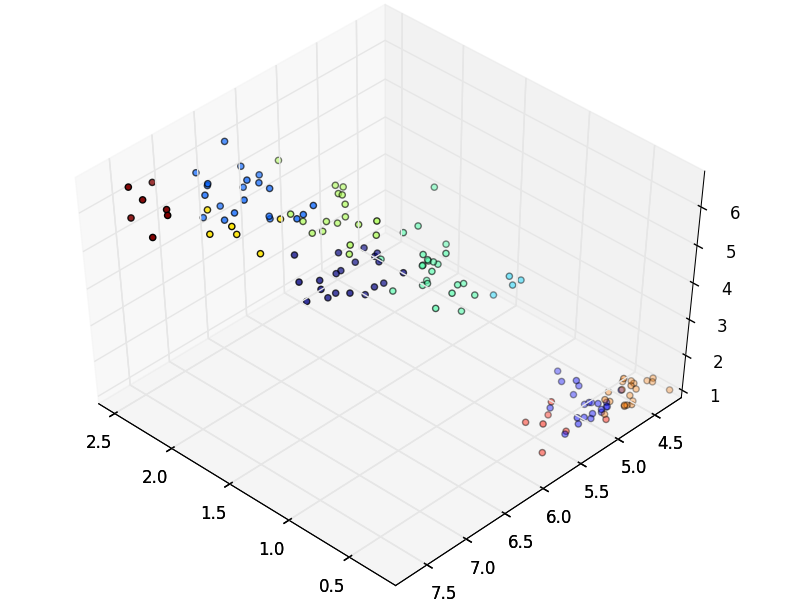
K:n arvolla 3 saatu klusterointi on hyvin samankaltainen verrattuna datan alkuperäisiin luokkiin. Kuvaajasta on havaittavissa kolme selkeää klusteria.



K:n arvolla 5 on havaittavissa luokkien sisäisiä ryppäitä. Poikkeuksena yksi luokka, joka pysyy hyvin yhtenäisenä.



K:n arvolla 8 myös viimeinen yhtenäinen luokka on jakautunut useammaksi ryppääksi.



K:n arvolla 10 on kuvaajasta havaittavissa, että ryppäiden määrä on liian suuri. Kasvattamalla k:n arvoa ei saada datasta uutta tietoa irti. Mielestämme parhaimman kuvan datasta saa k:n arvoilla 3 tai 5.

# 5. Työkalut

* Python 2.7
* Scikit-learn + riippuvuudet
* Graphviz

# 6. Lähdekoodit

Task1.py lukee datan read.py:llä. Task2.py, task3.py ja task4.py käyttävät kaikki datan hakemiseen tiedostoa task1.py:tä

## task1.py

import numpy as np

import read, copy

from sklearn.neighbors import NearestNeighbors

def get\_index(val):

for row in val:

for index in range(len(row)):

if "#" in str(row[index]):

return index

def format\_data(data, index):

result = copy.deepcopy(data)

for i in range(len(result)):

result[i].pop(index)

result[i] = result[i][:-1]

return result

def get\_nearest\_neighbors(data, datapoint):

work\_data = copy.deepcopy(data)

point = copy.deepcopy(datapoint)

ar = np.array(work\_data)

neighbors = NearestNeighbors().fit(ar)

distances, indices = neighbors.kneighbors(point)

return indices[0]

def predict(original\_data, missing\_value, nearest\_indices, missing\_value\_index):

values = []

for index in nearest\_indices:

values.append(original\_data[index][missing\_value\_index])

#Ennustettu arvo. Saatetaan tarvita

average\_val = np.mean(values)

print type(average\_val)

print values

print 'predicted missing value: %s' %str(average\_val)

#

missing\_value[0][missing\_value\_index] = average\_val

result = copy.deepcopy(original\_data)

result.append(missing\_value[0])

return result

def get\_full\_data():

original\_data, missing\_value = read.get\_data()

missing\_value\_index = get\_index(missing\_value)

data = format\_data(original\_data, missing\_value\_index)

m\_value = format\_data(missing\_value, missing\_value\_index)

neares\_indices = get\_nearest\_neighbors(data, m\_value)

full\_data = predict(original\_data, missing\_value, neares\_indices, missing\_value\_index)

return full\_data

## task2.py

import task1, os

from sklearn import tree

import numpy as np

def separate\_classifiers(data):

res\_data = []

cls = []

for row in data:

res\_data.append(row[:-1])

cls.append(row[4])

return res\_data, cls

def decision\_tree(data, cls):

classifier = tree.DecisionTreeClassifier(max\_depth=2).fit(data, cls)

#print classifier.predict([7,4,5,2])

visualize(classifier)

def main():

original\_data = task1.get\_full\_data()

data, classifier = separate\_classifiers(original\_data)

decision\_tree(data, classifier)

def visualize(data):

with open("decision\_tree\_visualization.dot", 'w') as graph:

graph = tree.export\_graphviz(data, out\_file=graph)

os.system('dot -Tpdf decision\_tree\_visualization.dot -o decision\_tree\_visualization.pdf')

os.unlink('decision\_tree\_visualization.dot')

if \_\_name\_\_ == '\_\_main\_\_':

main()

## task3.py

import task1, task2

from sklearn import cross\_validation

import numpy as np

from sklearn.naive\_bayes import GaussianNB as nb

from sklearn import tree

import matplotlib.pyplot as mplot

def cross\_val(data, classifiers):

averages = []

for i in range(5):

bayes\_estimates = []

tree\_estimates =[]

data = np.array(data)

classifiers = np.array(classifiers)

folds = cross\_validation.KFold(len(classifiers), n\_folds=10, shuffle=True)

for train, test in folds:

bayes = naive\_bayes(data[train], classifiers[train])

bayes\_estimates.append(bayes.score(data[test], classifiers[test]))

tree = decision\_tree(data[train], classifiers[train])

tree\_estimates.append(tree.score(data[test], classifiers[test]))

temp = []

temp.append(np.mean(bayes\_estimates))

temp.append(np.mean(tree\_estimates))

averages.append(temp)

visualize(averages)

def visualize(avg):

columns = ['Bayes scores', 'Tree scores']

rows = [1,2,3,4,5]

ax = mplot.subplot(111,frame\_on=False)

ax.xaxis.set\_visible(False)

ax.yaxis.set\_visible(False)

table = mplot.table(cellText=avg,

colLabels=columns,

rowLabels=rows,

loc='center')

mplot.subplots\_adjust(left=0.2)

mplot.show()

def naive\_bayes(data, classifiers):

bayes = nb()

return bayes.fit(data, classifiers)

def decision\_tree(data, cls):

return tree.DecisionTreeClassifier(max\_depth=2).fit(data, cls)

def main():

original\_data = task1.get\_full\_data()

data, classifiers = task2.separate\_classifiers(original\_data)

cross\_val(data, classifiers)

if \_\_name\_\_ == '\_\_main\_\_':

main()

## task4.py

from sklearn.cluster import KMeans

from mpl\_toolkits.mplot3d import Axes3D

import numpy as np

import pylab as pl

import task1, task2

def k\_means(data):

data = np.array(data)

for i in [3,5,8,10]:

name = 'k=%d'%i

km = KMeans(n\_clusters = i).fit(data)

figure = pl.figure(name, figsize=(8,6))

pl.clf()

ax = Axes3D(figure, rect=[0, 0, .95, 1], elev=48, azim=134)

pl.cla()

labels = km.labels\_

ax.scatter(data[:,3], data[:,0], data[:,2], c=labels.astype(float))

pl.show()

def main():

original\_data = task1.get\_full\_data()

data, classifier = task2.separate\_classifiers(original\_data)

k\_means(data)

if \_\_name\_\_ == '\_\_main\_\_':

main()

## read.py

def get\_data():

raw = open("iris2.txt")

result = []

missing = []

for row in raw:

if "#" not in row:

result.append(map(float, row.split()))

else:

temp = row.split()

for i in range(len(temp)):

if "#" not in temp[i]:

temp[i] = float(temp[i])

missing.append(temp)

return result, missing