Harjoitus 2

Raportti

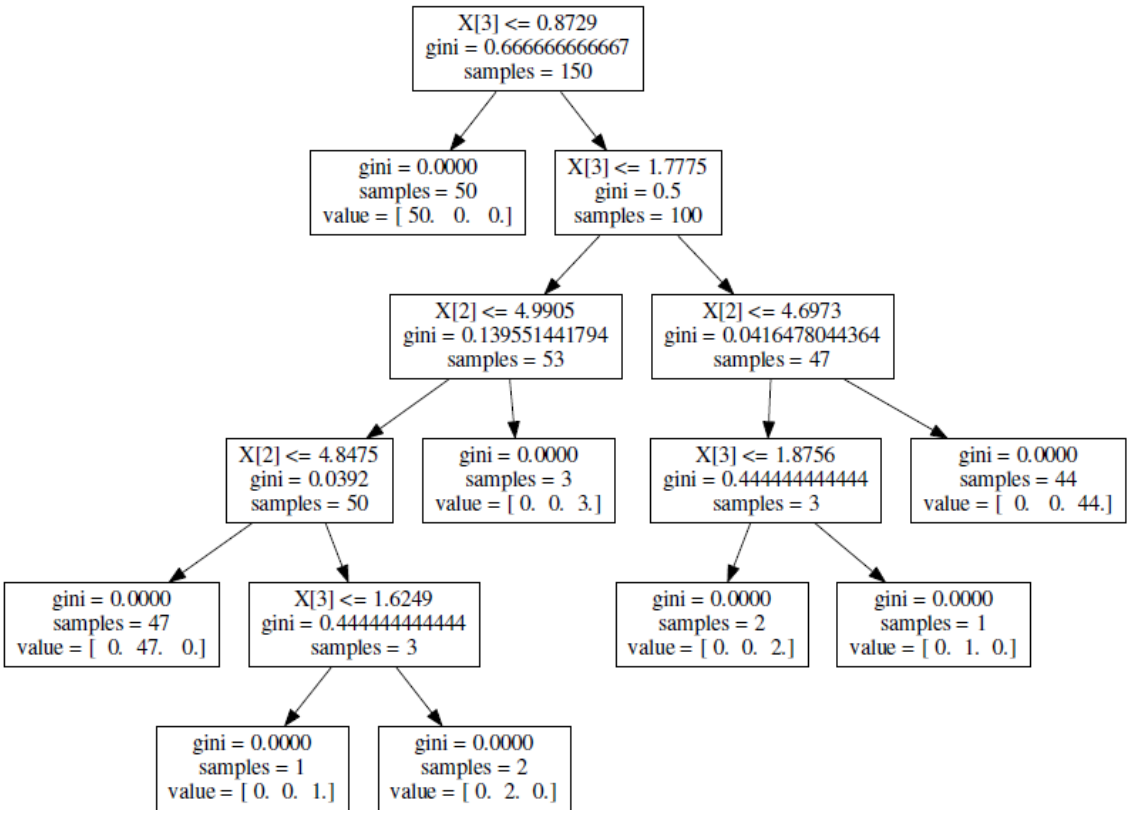
Aleksi Haapsaari 500039
Ville Ahti 79062
24.10.2013

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.

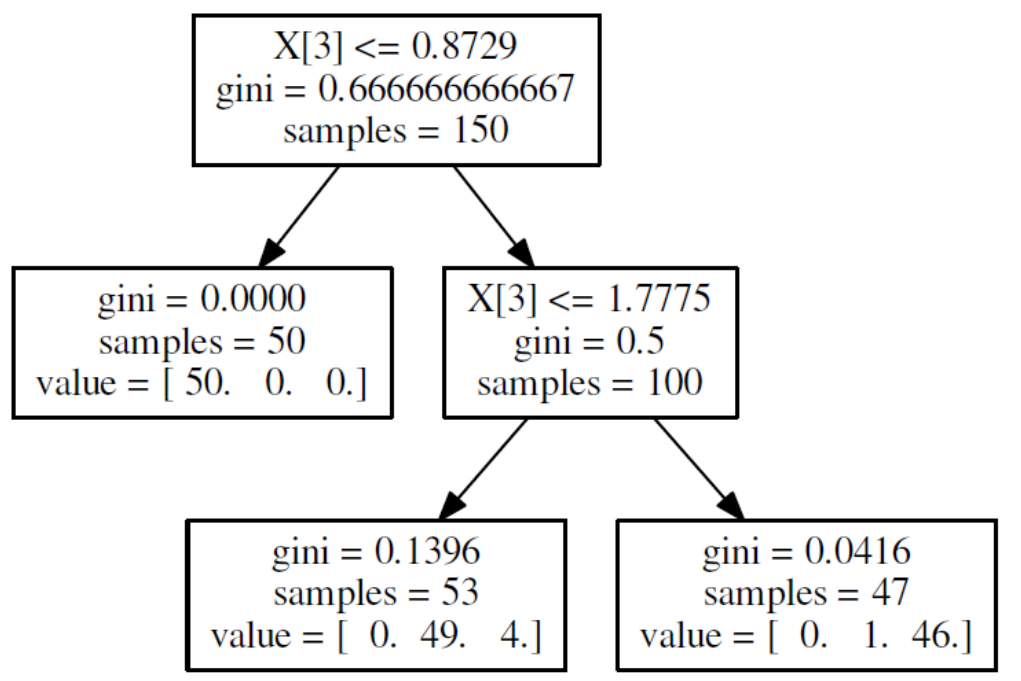
Saadut 5 arvoa ovat 3.4776441, 3.2564541, 3.1618804, 3.8657509, 3.0560231.
Arvioitu puuttuva arvo on **3.36355052.**

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

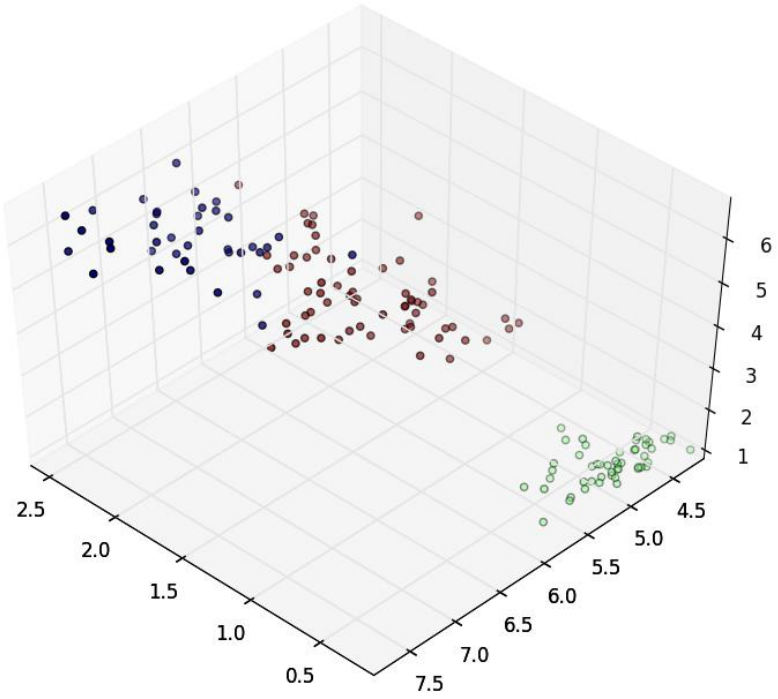
Naïve Bayes -luokittelijan luontiin käytimme scikit-learn-kirjaston GaussianNB-luokkaa, joka soveltuu parhaiten numeeriselle datalle ja olettaa, että se noudattaa normaalijakaumaa. Käytimme luokittelijan arvioimiseen n-fold cross-validation -menetelmää, jossa data jaettiin 10 yhtä satunnaiseen suureen 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.

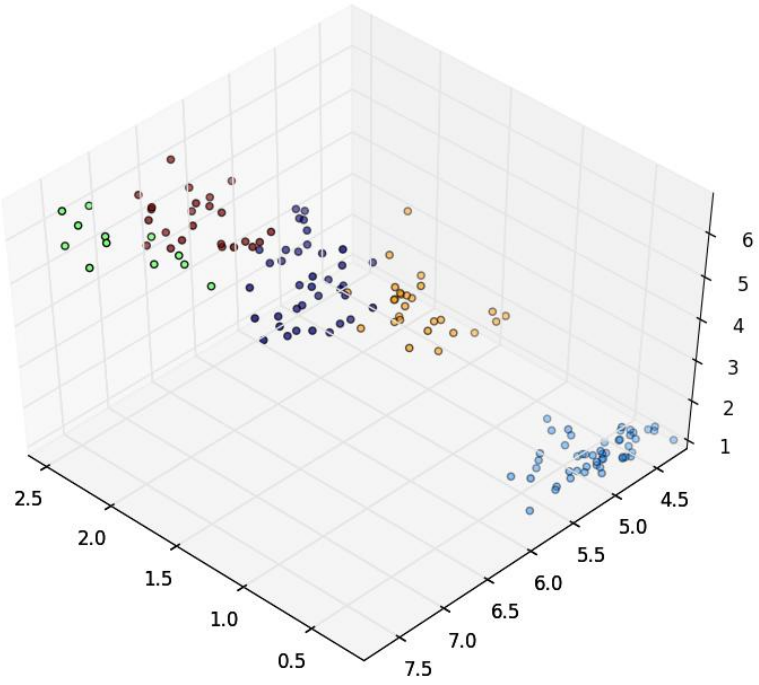
	Bayes scores	Tree scores
1	0.96	0.946666666667
2	0.946666666667	0.933333333333
3	0.953333333333	0.926666666667
4	0.953333333333	0.94
5	0.96	0.94

4. K-means klusterointi

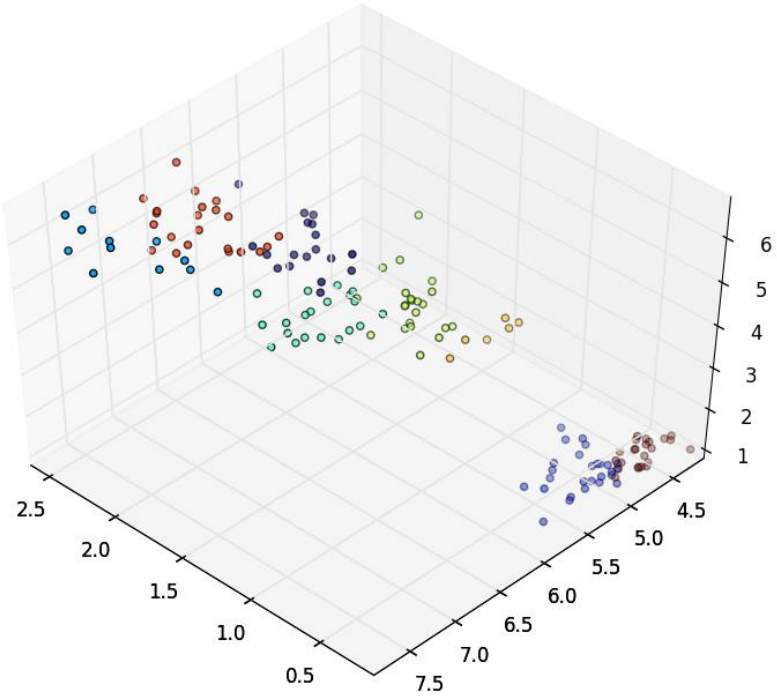
Klusteroimme dataa k:n arvoilla 3,5,8 ja 10.



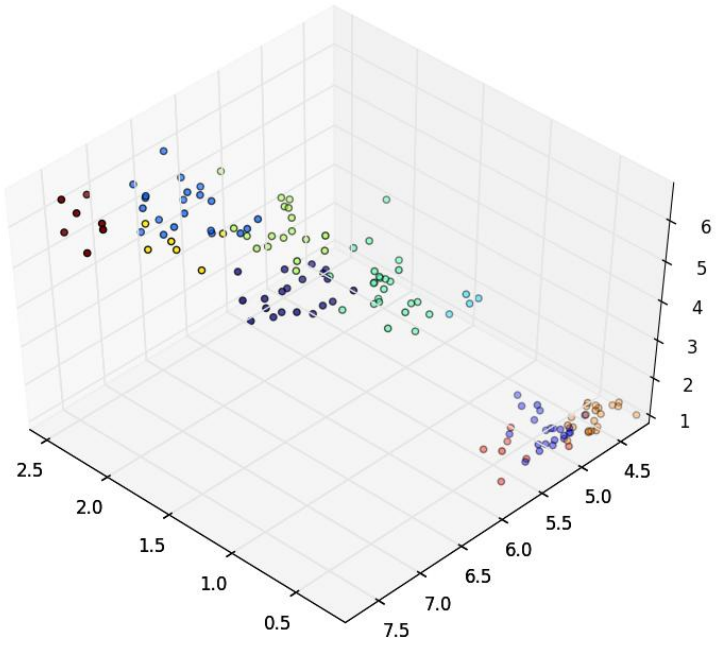
K:n arvolla 3 saatu klusterointi on hyvin samankaltainen verrattuna datan alkuperäisiin luokkiin. Kuvaajasta on havaittavissa kolme selkeää klusteria.



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
```