TDT4173 – Assignment 4

Silius Mortensønn Vandeskog (siliusmv)

10 April 2018

Theory

1)

The core idea of deep learning is…

2)

3)

Programming

```
# Extract data
knn_class <- read.csv("dataset/dataset/knn_classification.csv")
knn_reg <- read.csv("dataset/dataset/knn_regression.csv")
ada_train <- read.csv("dataset/dataset/adaboost_train.csv")
ada_test <- read.csv("dataset/dataset/adaboost_test.csv")
```

2.1)

We implement a k-NN algorithm from scratch. Then the program is reused in order to implement a k-NN regression and classification. The code can be seen below

```
# Euclidian distance function
eucDist <- function(all_points, point){

  res <- sweep(all_points, 2, point, "-")
  res <- res^2
  res <- apply(res, 1, sum)
  res <- sqrt(res)

  return(res)
}


# Find k nearest neighbours
findKnn <- function(all_points, point, distFunc, k, values){

  distances <- distFunc(all_points, point)

  ord <- order(distances)

  res <- list(distances = distances[1:k],
```

```
              values = values[ord][1:k])

  return(res)

}


# Classification by voting
vote <- function(closest){

  t <- table(closest$values)

  or <- order(t, decreasing = TRUE)

  if(t[or[1]] == t[or[2]]){
    # return("You need to fix voting when two or more classes
           # are equally well represented")
    equal_votes <- t[which(t == t[or[1]])]

    for(i in 1:k){
      p <- which(names(equal_votes) == as.character(closest$values[i]))
      if(p){
        return(names(equal_votes)[p])
      }
    }
  }

  return(names(t)[or[1]])
}


knn <- function(k, data, point, type){

  all_points <- as.matrix(data[, -dim(data)[2]])
  values <- data[, dim(data)[2]]

  closest <- findKnn(all_points = all_points,
                     point = point,
                     k = k,
                     values = values,
                     distFunc = eucDist)

  if(type == "reg"){
    return(mean(closest$values))
  } else if(type == "class"){
    return(vote(closest))
  }

  return("This type is not accepted")
}
```

We now use the algorithms with $k = 10$ for the 124^{th} example of the given data sets. As seen below, the algorithm predicts a value of 1.6 for the regression and 2 for the classification.

```r
knn(k = 10,
    data = knn_reg,
    point = as.vector(as.matrix((knn_reg[124, 1:3]))),
    type = "reg")
```

```
## [1] 1.6
```

```r
knn(k = 10,
    data = knn_class,
    point = as.vector(as.matrix((knn_class[124, 1:4]))),
    type = "class")
```

```
## [1] "2"
```

2.2)

We implement tha AdaBoost algorithm from scratch. The code can be seen in the print-out below.

```python
# Returns classifier for some data
# using T iterations of adaBoost
def adaBoost(data, T):
    dim = data.shape

    x = data[0:dim[0], 2:(dim[1]-1)]
    y = data[0:dim[0], 1]
    classifiers = []
    all_as = []

    weight = np.ones(dim[0]) * (1. / dim[0])
    for i in range(T):

        clf = tree.DecisionTreeClassifier(max_depth = 1)
        clf = clf.fit(x, y, sample_weight = weight)
        pred = clf.predict(x)

        epsilon = sum((pred != y) * weight)
        a = (1. / 2) * np.log((1 - epsilon) / epsilon)

        nw = weight * np.e**(-a * y * pred)
        nw = nw / sum(nw)
        weight = nw

        classifiers.append(clf)
        all_as.append(a)

    return(list([classifiers, all_as]))

# Returns predictions from an
# adaBoost algorithm
def adaPred(points, classifier):

    n = len(classifier[0])

    s = np.zeros(points.shape[0])
```

```
    for i in range(n):
        s = s + classifier[0][i].predict(points) * classifier[1][i]

    return(np.sign(s))

# Misclassification error
def misClass(pred, y):
    return(sum(pred != y) / float(len(y)))
# Return an error vector for iteration 1 through max_iter
def testAdaBoost(max_iter, train_data, test_data):

    test_dim = test_data.shape
    x_test = test_data[0:test_dim[0], 2:(test_dim[1]-1)]
    y_test = test_data[0:test_dim[0], 1]
    errors = []

    for i in range(max_iter):
        classifier = adaBoost(train_data, i+1)
        pred = adaPred(x_test, classifier)
        errors.append(misClass(pred, y_test))

    return(errors)
```

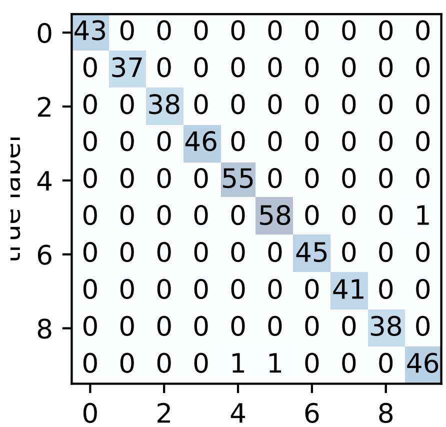
```
# -*- coding: utf-8 -*-
import sys
import numpy as np
import os
import sklearn
import matplotlib.pyplot as plt
cwd = os.getcwd()
sys.path.insert(0, cwd)
import ex_code as ss
knn_class = np.genfromtxt("dataset/dataset/knn_classification.csv",
delimiter=",", skip_header=1)
knn_reg = np.genfromtxt("dataset/dataset/knn_regression.csv",
delimiter=",", skip_header=1)
ada_test = np.genfromtxt("dataset/dataset/adaboost_test.csv",
delimiter=",", skip_header=1)
ada_train = np.genfromtxt("dataset/dataset/adaboost_train.csv",
delimiter=",", skip_header=1)
```

```
err = ss.testAdaBoost(15, ada_train, ada_test)
for e in err:
  print("%.2f" % e)
```

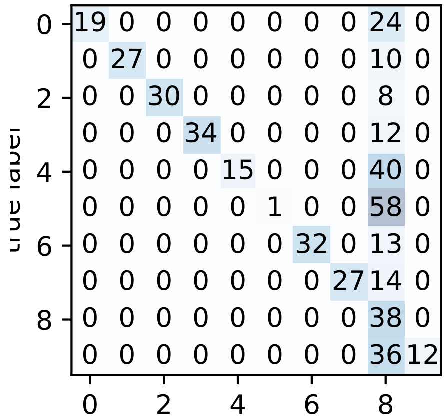
```
## 0.46
## 0.46
## 0.43
## 0.43
## 0.39
## 0.39
## 0.37
## 0.37
## 0.35
```

```
## 0.35
## 0.36
## 0.33
## 0.38
## 0.32
## 0.35
```

```
X_train, X_test, y_train, y_test = ss.getData()
from sklearn.neighbors import KNeighborsClassifier as KNN
neigh = KNN()
neigh.fit(X_train, y_train)
pred_neigh = neigh.predict(X_test)
from sklearn import svm
clf = svm.SVC()
clf.fit(X_train, y_train)
pred_svm = clf.predict(X_test)
from sklearn.ensemble import RandomForestClassifier as RFC
forest = RFC()
forest.fit(X_train, y_train)
pred_forest = forest.predict(X_test)
from sklearn.metrics import confusion_matrix
from mlxtend.plotting import plot_confusion_matrix
cnf_neigh = confusion_matrix(y_test, pred_neigh)
cnf_svm = confusion_matrix(y_test, pred_svm)
cnf_forest = confusion_matrix(y_test, pred_forest)
fig_neigh, ax = plot_confusion_matrix(conf_mat = cnf_neigh)
plt.show()
```




```
fig_svm, ax = plot_confusion_matrix(conf_mat = cnf_svm)
plt.show()
```




```
fig_forest, ax = plot_confusion_matrix(conf_mat = cnf_forest)
plt.show()
```