TDDE01 – Machine Learning
Individual Laboration Report 1

Martin Estgren <mares480>

9 november 2016

1 Assignment 1

The first assignment involves implementing the *K-Nearest neighbor* classifier and comparing to the built in *Weighted K-Nearest Neighbor* (kknn) classifier. The classifiers will be tested using a data set based on spam classification of emails given frequency of words. The distance measure that will be implemented is based on the *cosine similarity*.

1.0.1 *K-Nearest neighbor* implementation

The first step of implementing the *K-Nearest neighbor* involves creating a function with the training and test data as inputs and the predicted outcomes as output. At this point no pre-processing has been done on the data set apart from splitting it into one training and one testing set. Because of this we need to remove the last column where the spam classification for each observation is located.

Once the data has been cleaned of the last row the cosine similarity distance measure for each observation is calculated using the algorithm

$$\widehat{\mathbf{X}} = X_i / \sqrt{\sum_{j=rows} X_{ij}^2}$$

$$\widehat{\mathbf{Y}} = Y_i / \sqrt{\sum_{j=rows} Y_{ij}^2}$$

$$\mathbf{C} = 1 - \widehat{\mathbf{X}}\widehat{\mathbf{Y}}^{\mathbf{T}}$$

Where $\mathbf{X}$ is equal to the matrix of the training data set, $\mathbf{Y}$ is the matrix of the testing/observations data set and $\mathbf{C}$ is the distance measure matrix for each observation.

The next step requires us to sort each test observation by the distance to each training observation in order to get the K closest training observations. Because we transposed the $\mathbf{Y}$ training data set we need to sort the result in $\mathbf{C}$ by column instead of row.

Since we are looking for the K smallest values we get the K first columns of $\mathbf{C}$. Now we have the index (and distance) of all the relevant observations and need to convert this to probabilities. This is done by doing a lookup of the last column (containing spam classifications as 0:s or 1:s) of the original training data for each of the K indexes. We then mean the result of each lookup to get the probability of said observation being classified as spam.

$$\frac{\sum_{i=indexes}(traning_i)}{|indexes|} \tag{1}$$

The values are returned as a vector where each row is a observation from the testing data set and the values are the probability of said observation being classified as spam.

Next we classify the result where each observation with a probability above $p(spam) > 0.5$ as spam. We set $K = 5$ meaning we take into account the K nearest observations from the training set. The result is plotted as a confusion matrix, as can be observed bellow.

```
            knearest, k = 5
observations   0   1
           0 682 263
           1 186 239
misclassification rate = 0.3277372
```

The result indicates a hit rate of about 0.68.

We then redo the classification with the same probability threshold but set $K = 1$ meaning the data set will only take the closest training observation into account.

```
            knearest, k = 1
observations   0   1
           0 650 295
           1 174 251
misclassification rate = 0.3423358
```

The result indicates a hit rate of about 0.66 which is slightly lower than $K = 5$.

As reference we perform the same classification on the same data set with the built in classifier *kknn*.

```
             kknn, k = 5, k = 1
observations   0   1
           0 626 319
           1 184 241
misclassification rate = 0.3671533
```

The biggest difference from out own implementation is that the hit rate and confusion matrix are identical for both $K = 1$ and $K = 5$. The hit rate is also slightly lower that ours at about 0.63.

The final step involves testing the two classifiers when we use different probability thresholds. In this case we will test for all thresholds $p(x) > 0.05, ..., 0.95$ with steps of 0.01. This will be done by drawing a *ROC- curve* over the different probability thresholds. The *ROC-curve* is calculated as a relation between the *true negative rate* over the *true positive rate*. The *true positive rate* is calculated as

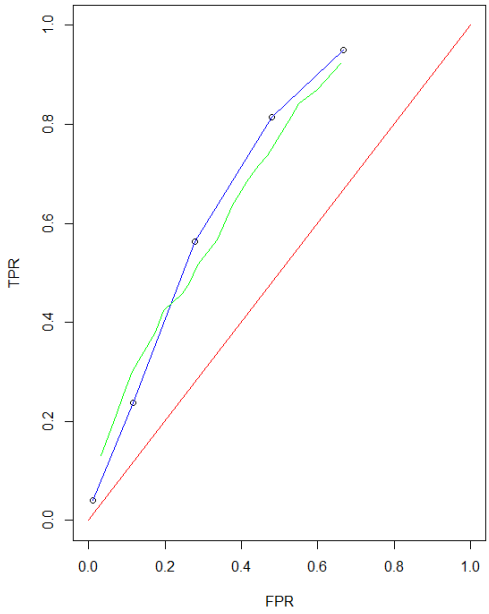
$$\frac{Tp}{Tp + Fn} \tag{2}$$

where Tp stands for the *true positives* i.e. the cases where both the prediction and observations classified the case as *true*, and Fn stands for the *false negatives* i.e. the cases where the prediction was *false* but the observation was *true*.

The *true negative rate* is calculated as $1 - specificity$ where *specificity* is equal to:

$$\frac{Tn}{Tn + Fp} \tag{3}$$

where Tn stands for the *true negatives* i.e. the cases where both the prediction and observations classified the case as *false*, and Fp stands for the *false positives* i.e. the cases where the prediction was *true* but the observation was *false*.



Figur 1: ROC - curve of *knearest* and *kknn*.

The blue curve is our *knearest* classifier and the green curve *kknn*. The red line is the reference line of a random uniform binary classifier.

We see that out classifier, as observed in the confusion matrices, performs a couple of percentages better than *kknn* in some values of the threshold values.

2 Assignment 2

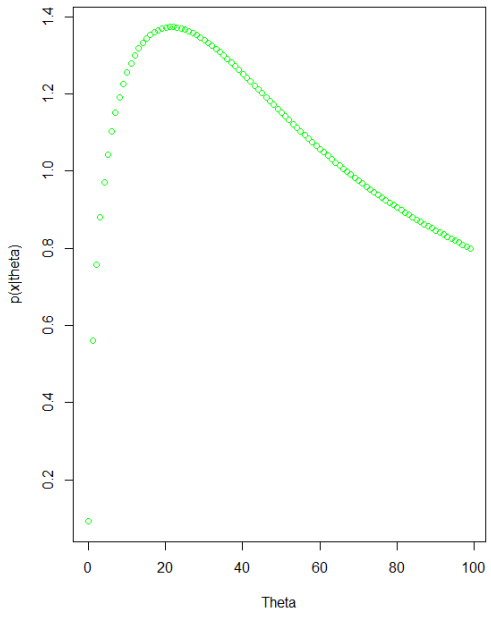
In this section the second assignment will be walked presented where the goal is to analyses a data set about the life length of a set of machines. The first part involves estimating the distribution and calculating the *maximum logarithmic likelihood* for a machine to break down.

We start by assuming the distribution

$$\mathbf{D} = \theta * e^{-\theta * \mathbf{x}} \tag{4}$$

where $\mathbf{x}$ is the observed life time of a set of machines and $\mathbf{D}$ is the resulting vector ob observations. We will be trying out different values of θ. For this

analysis we will try the range $\theta = 0.1, ..., 100$ in 1:s increments. The result can be observed in the plot below.



Figur 2: Distribution of the data set *machines*.

The distribution seems to follow a typical exponential distribution where each observation is independent (as already specified for the data set).

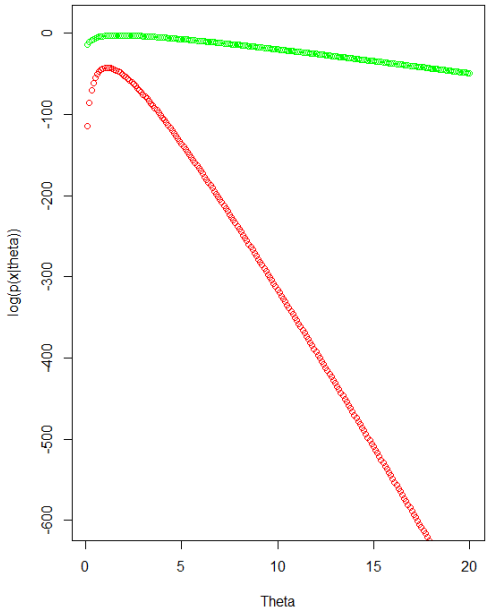
We then try to find the log-likelihood $log(p(\mathbf{b}|\theta))$ using the result of above analysis. This is done by $log(\mathbf{D})$ for all θ in the range $0.1, ..., 20$ with 0.1 increments. multiple observations for each θ we use the formula

$$log(\prod p(\mathbf{x}|\theta)) \tag{5}$$

where we do the $\prod$ over all observations for a given θ this is equivalent with the formula

$$\sum log(p(\mathbf{x}|\theta)) \tag{6}$$

The result is plotted below, both for the full data set (red) and for the first six observations (green).



Figur 3: Maximum log-likelihood for the full data set and the first 6 observations.

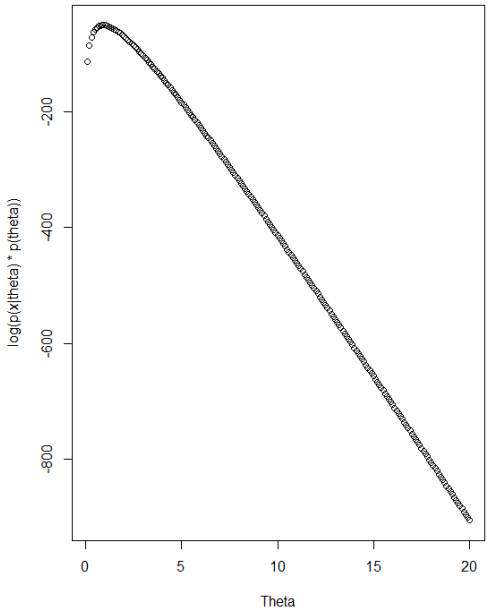
What we can observe from this plot is that 6 observations are not enough to get a reliable likelihood estimation of the data set and that the theta resulting in the highest likelihood is $\theta = 1.1$.

The next task is to calculate the posterior probabilities using the *Bayesian model*:

$$p(x|\theta) = \theta * e^{-\theta * \mathbf{x}} \tag{7}$$

and prior probability $p(\theta) = \lambda * e^{-\lambda * \theta}$ with $\lambda = 10$.

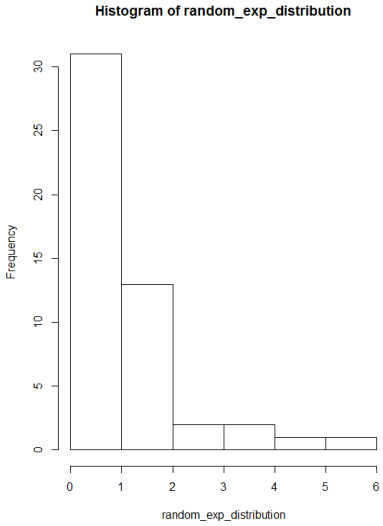
To calculate this we create the function $I(\theta) = log(p(\mathbf{x}|\theta) * p(\theta))$. The resulting "curve" can be observed bellow.



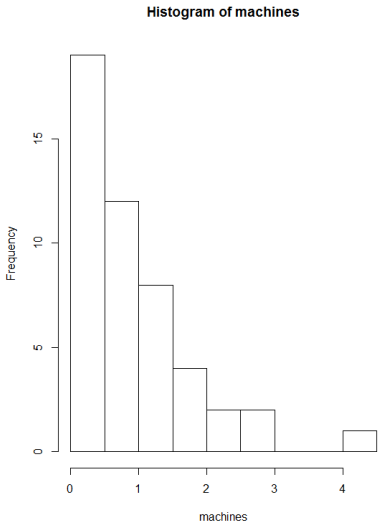
Figur 4: Posterior likelihood of the data set *machines*.

Although $log(p(\mathbf{x}))$ follows a exponential decrease curve the $p(\theta)$ follows a linear curve and has very large values compared to $log(p(\mathbf{x}))$. The maximum θ has gone from 1.1 to 0.9.

The final step involves generating 50 new observations from the exponential distribution by using the $\theta = 1.1$. These observations will then be plotted in a histogram and compared to a histogram of the machine lifetime data set.



Figur 5: Histogram over 50 random samples from exponential distribution.



Figur 6: Histogram over the data set *machines*.

In comparison to the random samples from a exponential distribution with $\theta = 1.1$, the observations in the machines data set are very close to the exponential distribution.

3 Appendix: A - Code assignment 1

Listing 1: Code for assignment 1

```
library(readxl)
library(kknn)

# Import data
spambase <- read_excel("spambase.xlsx")

# Get number of observations
n = dim(spambase)[1]

# Set psuedo random seed
set.seed(12345)
```

```
# Get half of the indexes
id = sample(1:n, floor(n * 0.5))

# Assign 50% of the observations as traning data
train = spambase[id, ]

# Assign 50% of the observations as test data
test = spambase[-id, ]

# Function to get the spam classification of a given
    index
spam_lookup <- function(indexes, lookup_table) {
  return (mean(lookup_table[indexes, ncol(lookup_
     table)]))
}

# Function of knearest algortihm, returns the
   predictited probabilites for each observation
knearest <- function(train, K, test) {
  nospam_train = train[, -ncol(train)]
  nospam_test = test[,-ncol(test)]
  euclidean_train = sqrt(rowSums(nospam_train ^ 2))
  euclidean_test = sqrt(rowSums(nospam_test ^ 2))

  train_hat = nospam_train / euclidean_train
  test_hat = nospam_test / euclidean_test
  cost = train_hat %*% t(test_hat)
  distance = 1 - cost

  k_nearest_observations = as.matrix(t(apply(
     distance, 2, order))[, 1:K])

  spam_predictions = apply(k_nearest_observations,
     1, spam_lookup, lookup_table =
                              train)
  return (spam_predictions)

}

# Returns if a specific arrays elements are over a
   specific threshold
```

```
threshold <- function(threshold, data) {
  return (as.numeric(data > threshold))
}

#Function to calcuate the sensitivity of a data set
sensitivity <- function(observations, predictions) {

  result = sum((observations == 1 & predictions ==
     1)) / (sum((observations == 1 &predictions ==
     1)) + sum(observations == 1 & predictions == 0)
     )
  return(result)

}

#Function to calcuate the specificity of a data set
specificity <- function(observations, predictions) {
  result = sum((observations == 0 &
    predictions == 0)) / (sum((observations == 0 &
    predictions == 0)) + sum(observations == 0 &
       predictions == 1))
  return(result)
}

observations = test$Spam
knearest_k5 = as.numeric(knearest(as.matrix(train),
   5, as.matrix(test)) > 0.5)
knearest_k1 = as.numeric(knearest(as.matrix(train),
   1, as.matrix(test)) > 0.5)

mst_k5 = table(observations, knearest_k5)
mst_k1 = table(observations, knearest_k1)
print(mst_k5)
print(mst_k1)

print(1 - sum(diag(mst_k5) / sum(mst_k5)))
print(1 - sum(diag(mst_k1) / sum(mst_k1)))

# Check with build-in functions
kknn_predictions_k5 = kknn(
  formula = Spam ~ .,
```

```
  train = train,
  test = test ,
  k = 5
)
kknn_predictions_k1 = kknn(
  formula = Spam ~ .,
  train = train,
  test = test,
  k = 1
)

kknn_predictions_k5 = as.numeric(fitted.values(kknn_
   predictions_k5) > 0.5)
kknn_predictions_k1 = as.numeric(fitted.values(kknn_
   predictions_k1) > 0.5)

print(table(observations, kknn_predictions_k5))
print(table(observations, kknn_predictions_k1))

print(mean(kknn_predictions_k5 != observations))
print(mean(kknn_predictions_k1 != observations))

# Check with other prediction thresholds

# Converts to matrix for apply operations
thresholds = seq(from = 0.05, to = 0.95, by = 0.05)
thresholds = matrix(thresholds, length(thresholds),
   1)

# Get the knearest predictions of a data set
knearest_predictions = knearest(as.matrix(train), 5,
    as.matrix(test))

# Apply the threshold function to get the
   predictions
knearest_outcomes = t(apply(thresholds, 1, threshold
   , data = knearest_predictions))

# Calculate the specificity and the sensitivty of
   said predictions
knearest_sensitivity = apply(knearest_outcomes, 1,
```

```
   sensitivity, observations = observations)
knearest_specificity = apply(knearest_outcomes, 1,
   specificity, observations = observations)


# Calcualte the kknn predictions
kknn_predictions = kknn(
  formula = Spam ~ .,
  train = train,
  test = test ,
  k = 5
)
kknn_outcomes = t(apply(thresholds, 1, threshold,
   data = fitted.values(kknn_predictions)))

# Calculate the sensitivty and specificty of the
   kknn predictions
kknn_sensitivity = apply(kknn_outcomes, 1,
   sensitivity, observations = observations)
kknn_specificity = apply(kknn_outcomes, 1,
   specificity, observations = observations)

# Plot the values as ROC curves
plot(
  1 - knearest_specificity,
  knearest_sensitivity,
  xlim = c(0, 1),
  ylim = c(0, 1),
  xlab = "FPR",
  ylab = "TPR"
)
lines(1 - knearest_specificity, knearest_sensitivity
   , col = "Blue")
lines(1 - kknn_specificity, kknn_sensitivity, col =
   "Green")
lines(c(0, 1), c(0, 1), col = "Red")
```

4 Appendix: B - Code assignment 2

Listing 2: Code for assignment2

```
library(readxl)

machines <- data.matrix(read_excel("machines.xlsx"))


probability <- function(theta, data) {
  return(theta * exp((-1 * theta) * data))
}

log_likelihood <- function(theta, data) {
  ll = log(probability(theta, data))
  return(ll)
}

max_log_likelihood <- function(theta, data) {
  ll = log_likelihood(theta, data)
  return(sum(ll))
}

l <- function(log_likelihoods, theta, lambda) {
  return(log_likelihoods + log_likelihood(lambda,
     theta))
}

# Create a matrix with all the theta samples to be
   tested
theta = seq(from = 0.1, to = 100, by = 1)
theta = matrix(theta, length(theta), 1)

# Generate the disitrbution of the machines data set
distribution = t(apply(theta, 1, probability, data =
    machines))

# Display the plot
plot(
  theta,
  rowMeans(distribution),
  col = "Green",
  xlab = "Theta",
```

```
  ylab = "p(x|theta)"
)
theta = seq(from = 0.1, to = 20, by = 0.1)
theta = matrix(theta, length(theta), 1)

# Calculate max log likelihood for the full data set
max_likelihoods_full = apply(theta, 1, max_log_
   likelihood, data = machines)

#Get the best theta (for task 5)
max_theta = which.max(max_likelihoods_full) / 10
print(max_theta)
# Calculate max log likelihood for the first 6
   observations in the data set
max_likelihoods_k6 = apply(theta, 1, max_log_
   likelihood, data = machines[1:6, ])

plot(
  theta,
  max_likelihoods_full,
  col = "Red",
  xlab = "Theta" ,
  ylab = "log(p(x|theta))",
  ylim = c(-600, 10)
)
points(theta, max_likelihoods_k6, col = "Green")

# l(theta) = log(p(x|theta) * p(theta)) = log(p(x|
   theta)) + log(p(theta)), log(p(x|theta)) = max_
   likelihoods_full
posteriori_likelihood = l(max_likelihoods_full,
   theta, 10)

# Looks like a linear function since log(p(theta))
   is huge compared to log(p(x|theta))
plot(theta,
     posteriori_likelihood,
     xlab = "Theta" ,
     ylab = "log(p(x|theta)␣*␣p(theta))")

# Generate 50 samples from the exponential
```

```
   distribution with theta is equal to maximum
   likleihood theta for the given data set
set.seed(12345)
random_exp_distribution = rexp(50, max_theta)

hist(random_exp_distribution)
hist(machines)
```