Uppsala University

Machine Learning

Assignment 1

General information

- The recommended tool in this course is R (with the IDE R-Studio). You can download R **here** and R-Studio **here**. You can use Python and Jupyter Notebooks, although the assignments may use data available only through the R package, a problem you would need to solve yourself.
- Report all results in a single, `*.pdf`-file. *Other formats, such as Word, Rmd, Jupyter Notebook, or similar, will automatically be failed.* Although, you are allowed first to knit your document to Word or HTML and then print the assignment as a PDF from Word or HTML if you find it difficult to get TeX to work.
- The report should be written in English.
- If a question is unclear in the assignment. Write down how you interpret the question and formulate your answer.
- You should submit the report to **Studium**. Deadlines for all assignments are **Sunday 23.59**. See **Studium** for dates. Assignments will be graded within 10 working days from the assignment deadline.
- To pass the assignments, *you should answer all questions not marked with **, and get at least 75% correct.
- To get VG on the assignment, *all questions should be answered, including questions marked with a **, although minor errors are ok. VG will only be awarded on the first deadline of the assignment.
- A report that does not contain the general information (see the **template**), will be automatically rejected.
- When working with R, we recommend writing the reports using R markdown and the provided **R markdown template**. The template includes the formatting instructions and how to include code and figures.
- Instead of R markdown, you can use other software to make the pdf report, but you should use the same instructions for formatting. These instructions are also available in **the PDF produced from the R markdown template**.
- The course has its own R package `uuml` with data and functionality to simplify coding. To install the package just run the following:

  1. `install.packages("remotes")`
  2. ```
     remotes::install_github("MansMeg/IntroML",
     subdir = "rpackage")
     ```

- We collect common questions regarding installation and technical problems in a course Frequently Asked Questions (FAQ). This can be found **here**.
- You are not allowed to show your assignments (text or code) to anyone. Only discuss the assignments with your fellow students. The student that show their assignment to anyone else could also be considered to cheat. Similarly, on zoom labs, only screen share when you are in a separate zoom room with teaching assistants.
     ```

- The computer labs are for asking all types of questions. Do not hesitate to ask! The purpose of the computer labs are to improve your learning. We will hence focus on more computer labs and less on assignment feedback. *Warning!* There might be bugs in the assignments! Hence, it is important to ask questions early on so you dont waste time of unintentional bugs.
- If you have any suggestions or improvements to the course material, please post in the course chat feedback channel, create an issue **here**, or submit a pull request to the public repository.

Contents

1 Basic, Stochastic, and Mini-Batch Gradient Descent **4**

1.1 Implement the gradient for logistic regression 4

1.2 Implement Gradient Descent 5

2 Regularized Regression **8**

3 * Gradient Descent for penalized logistic regression **10**

4 * Implementation of the Adam optimizer **12**

1 Basic, Stochastic, and Mini-Batch Gradient Descent

This assignment will study different ways to optimize common objective functions in many areas of Machine Learning, namely, stochastic gradient descent. Here we will test to implement these optimizers for a well-known model, logistic regression. See Ruder (2016) for pseudo-code and details on the algorithms.

We are going to work with this data as a test case:

```
library(uuml)
data("binary")

binary$gre_sd <- (binary$gre - mean(binary$gre))/sd(binary$gre)
binary$gpa_sd <- (binary$gpa - mean(binary$gpa))/sd(binary$gpa)
X <- model.matrix(admit ~ gre_sd + gpa_sd, binary)
y <- binary$admit
```

1.1 Implement the gradient for logistic regression

The likelihood function for logistic regression is

$$L(\theta, \mathbf{y}, \mathbf{X}) = \prod_{i=1}^{n} p_i^{y_i} (1 - p_i)^{1-y_i} \,,$$

where

$$\log \frac{p_i}{1 - p_i} = \mathbf{x}_i \theta \,,$$

and $\mathbf{x}_i$ is the ith row from the design matrix $\mathbf{X}$ and $\theta \in \mathbb{R}^P$ is a $1 \times P$ matrix with the parameters of interest.

Commonly, to find maximum likelihood estimates of θ, we usually use the log-likelihood as the objective function we want to optimize, i.e.

$$\begin{aligned} l(\theta, \mathbf{y}, \mathbf{X}) &= \sum_{i=1}^{n} y_i \log(p_i) + (1 - y_i) \log(1 - p_i) && (1) \\ &= \sum_{i=1}^{n} y_i \mathbf{x}_i \theta + \log(1 - p_i) && (2) \\ &= \sum_{i=1}^{n} y_i \mathbf{x}_i \theta - \log(1 + \exp(\mathbf{x}_i \theta)) \,. && (3) \end{aligned}$$

Although, in our case we instead want to minimize the negative log likelihood $\text{NLL}(\theta, \mathbf{y}, \mathbf{X}) = -l(\theta, \mathbf{y}, \mathbf{X})$, which is essentially the same as the cross-entropy loss.

1. (4p) Derive the the gradient for $\text{NLL}(\theta, \mathbf{y}, \mathbf{X})$ with respect to θ.
   Hint! See Hastie et al. (2009, Ch. 4.4).

2. (2p) Implement the gradient as a function in R. Below are two examples of how it should work. Note, `l_grad(y, X, theta)` return the gradient of $(1/n)l(\theta, \mathbf{y}, \mathbf{X})$. Show that your implementation give the same result as these test cases in your report.

```
ll_grad(y, X, theta = c(0,0,0))
```

```
## (Intercept)      gre_sd      gpa_sd
##     -0.1825      0.0857      0.0829
```

```
ll_grad(y, X, theta = c(-1,0.5,0.5))
```

```
## (Intercept)      gre_sd      gpa_sd
##      0.0217     -0.0395     -0.0426
```

1.2 Implement Gradient Descent

We now have the primary tool for implementing gradient descent and stochastic gradient descent. The log-likelihood function has been implemented in the `uuml` package and can be accessed as follows.

```
# The inside of the function
ll
```

```
## function(y, X, theta){
##   checkmate::assert_integerish(y, lower = 0, upper = 1, any.missing = FALSE)
##   checkmate::assert_matrix(X, mode = "numeric", nrows = length(y))
##   checkmate::assert_vector(theta, len = ncol(X))
##   Xtheta <- as.matrix(X %*% theta)
##   ll_i <- y * Xtheta - log(1 + exp(Xtheta))
##   sum(ll_i)
## }
## <bytecode: 0x157164af0>
## <environment: namespace:uuml>
```

```
# How we can use it
theta <- c(0, 0, 0)
ll(y, X, theta)
```

```
## [1] -277.2589
```

1. (1p) Run logistic regression in R to get an MLE estimate of `theta` using the `glm()` function. Print the three parameter estimates.

   Hint! The design matrix `X` already include an intercept so you need to remove that either from `X` or from the R formula using `y ~ -1 + X`.

2. Implement the following gradient descent algorithms (see Ruder, 2016, for details and psuedo-code) as three separate R functions. Print these functions in your report.

   Note! Each epoch should include all data points. A tip is to order the data points randomly in each iteration.

   (a) (2p) ordinary (full/batch) gradient descent

   (b) (2p) stochastic gradient descent

   (c) (2p) mini-batch (stochastic) gradient descent using ten samples to estimate the gradient

   A boilerplate for the algorithm can be found here.

```r
mbsgd <- function(y, X, sample_size, eta, epochs){
  # Setup output matrix
  results <- matrix(0.0, ncol = ncol(X) + 2L, nrow = epochs)
  colnames(results) <- c("epoch", "nll", colnames(X))

  # Run algorithm
  theta <- rep(0.0, ncol(X)) # Init theta to the 0 vector
  for(j in 1:epochs){

    ### Put the algorithm code here ###

    # Store epoch, nll and output results
    results[j, "epoch"] <- j
    results[j, "nll"] <- ll(y, X, theta)
    results[j, -(1:2)] <- theta
  }
  return(results)
}
```

   The function can then be called as:

```r
result <- mbsgd(y, X, sample_size = 10, eta = 0.1, epochs = 500)
```

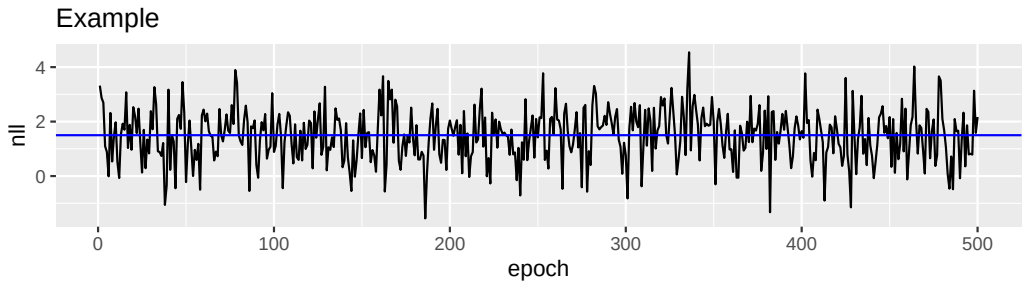
3. Try different learning parameters η and run the algorithm for roughly 500 epochs. You should find three different η for each algorithm. One η where the optimizer diverge, one η where the optimizer converge very slowly and one η that converges quick(er). Note that η should be fixed/the same during all 500 epochs. Visualize your results in two plots per algorithm and choice of η as a line graph (hence 18 figures in total).

   (a) (2p) The (negative) log-likelihood value *for all observations* for the given θ (y-axis) and the epochs (full data iterations, x-axis).

   (b) (2p) The value of one θ parameter of your choice (y-axis) and the epochs (full data iterations, x-axis). Also include the true values you got from using the `glm()` function above as a horizontal line in the figure.

Below is an example of how you can plot the loss and the parameter over epochs using `ggplot2`. Note, this is just random generated data as an example.

```
library(ggplot2)
# create data
set.seed(4711)
epoch <- 1:500
nll <- rnorm(500,1.5) # Just random data.
data <- data.frame(nll, epoch)
# Plot
ggplot(data, aes(x=epoch, y=nll)) +
  geom_line() +
  geom_hline(yintercept = 1.5, color = "blue") +
  ggtitle("Example")
```



2 Regularized Regression

The datasets `prob2_train` and `prob2_test` contains simulated data with 240 explanatory variables (`V1-V240`) and 1 numerical response variable (`y`). As per the dataset names, the first dataset contains training data and the second contains test data for this problem. To access the data, just run:

```
library(uuml)
data("prob2_train")
data("prob2_test")
dim(prob2_train)

## [1] 200 241

X <- as.matrix(prob2_train[,-241])
y <- as.matrix(prob2_train["y"])
X_test <- as.matrix(prob2_test[,-241])
y_test <- as.matrix(prob2_test["y"])
```

You should do the following and present the results in your report:

1. (1p) Fit a linear model to the training data. What are the results? Why does this happen?

2. (1p) Use `glmnet()` function from the `glmnet` package to fit a linear lasso regression to the training data with $\lambda = 1$. How many coefficients are used in the model? *Hint!* `coefficients()`

3. Split the data into ten folds randomly. *Hint!* Use the `sample()` function and create a new variable called fold. Here is an example on how to select the folds

   ```
   fold <- sample(1:10, nrow(X), replace = TRUE)

   X_trainfold1 <- X[fold==1,]
   X_valfold1 <- X[fold!=1,]
   y_trainfold1 <- y[fold==1,]
   y_valfold1 <- y[fold!=1,]
   ```

4. (5p) Now implement 10-fold cross-validation on the training data. *Hint!* Implement this as function called `glmnet_cv()` that takes the `fold` variable, `X`, `y` and `lambda` and then outputs the RMSE.

   (a) For each fold, hold out the validation fold and train on the the rest of the training set. See below.

   ```
   mod <- glmnet(X_trainfold, y_trainfold, alpha = 1, lambda = [different lambda here])
   pred_y_valfold <- predict(mod, newx = X_valfold)
   ```

   (b) Use the `rmse()` function in the `uuml` R package to compute the root mean squared error (RMSE) for each fold as follows:

```
rmse(pred_y_valfold, y_valfold)
```

(c) Take the mean RMSE for the five folds.

5. Now, compute the RMSE for $\lambda = 1$ using 5-fold cross-validation.
6. Test different values for the hyper parameter λ. What is the best RMSE you can get with 10-fold cross-validation? What is the λ for this value?
7. Now use the best model to do predictions on the test set. What is your RMSE on the test set?

3 * Gradient Descent for penalized logistic regression

To get a pass with distinction point, you can choose between this task or the task below.

This task is only necessary to get one point toward a *pass with distinction* (VG) grade. Hence, if you do not want to get a *pass with distinction* (VG) point, you can ignore this part of the assignment.

Here we use the same data as in task 1 above.

```
library(uuml)
data("binary")

binary$gre_sd <- (binary$gre - mean(binary$gre))/sd(binary$gre)
binary$gpa_sd <- (binary$gpa - mean(binary$gpa))/sd(binary$gpa)
X <- model.matrix(admit ~ gre_sd + gpa_sd, binary)
y <- binary$admit
```

In ridge regression we penalize the regression coefficients by λ. This gives the likelihood function for logistic regression with ridge penalty as

$$l_r(\theta, \mathbf{y}, \mathbf{X}, \lambda) = \frac{1}{n} l(\theta, \mathbf{y}, \mathbf{X}) - \frac{\lambda}{2} \sum_{i=1}^{P} \theta_i^2 \,, \tag{4}$$

where $l(\theta, \mathbf{y}, \mathbf{X})$ is defined as in task 1 above. We also want to minimize the negative penalized log likelihood $\text{NLL}_r(\theta, \mathbf{y}, \mathbf{X}, \lambda) = -l_r(\theta, \mathbf{y}, \mathbf{X}, \lambda)$ here as well.

Note! In general, we should not regularize the intercept (see Hastie et al., 2009, Ch. 3.4). Hence below the gradient don't regularize intercept.

Note! The objective function above is the one used in the `glmnet` package. We can define this slightly different (for example not divide l with n). This will give different estimates because λ will have different meanings. More information on the `glmnet` objective function can be found here.

1. (2p) Derive the the gradient for $\text{NLL}_r(\theta, \mathbf{y}, \mathbf{X}, \lambda)$ with respect to θ.
2. (1p) Implement the gradient as a function in R. Below are two examples of how it should work. *Note!* `l_grad(y, X, theta, lambda)` implemented above return the gradient of $(1/n)l(\theta, \mathbf{y}, \mathbf{X}, \lambda)$. Show that your implementation give the same result as these test cases in your report.

```
lr_grad(y, X, theta = c(0,0,0), lambda = 0)

## (Intercept)      gre_sd      gpa_sd
##     -0.1825      0.0857      0.0829
```

```
lr_grad(y, X, theta = c(0,0,0), lambda = 1)

## (Intercept)      gre_sd      gpa_sd
##     -0.1825      0.0857      0.0829
```

```
lr_grad(y, X, theta = c(-1,0.5,0.5), lambda = 1)

## (Intercept)      gre_sd      gpa_sd
##      0.0217     -0.5395     -0.5426
```

3. (1p) Implement the regularized log-likelihood l_r or the negative log-likelihood NLL_r in R. *Note!* This differs from the log-likelihood above (where we used the sum over the observations). Show that your implementation give the same result as these test cases in your report.

```
lr(y, X, theta = c(0,0,0), lambda = 0)

## [1] -0.6931
```

```
lr(y, X, theta = c(0,0,0), lambda = 1)

## [1] -0.6931
```

```
lr(y, X, theta = c(-1,0.5,0.5), lambda = 1)

## [1] -0.8613
```

4. (1p) Run logistic regression with ridge penalty in R using `glmnet` estimate of `theta`. Set `lambda` to 1 and `alpha` to 0 to run a penalized logistic regression. You also need to remove the intercept from `X`.

5. (2p) Implement the following gradient descent algorithms for the penalized logistic objective:

   (a) ordinary (full or batch) gradient descent

   (b) mini-batch (stochastic) gradient descent using ten samples to estimate the gradient

6. (2p) Try different learning parameters η and λ. When does the algorithm converge or diverge? Visualize the iterations (x-axis) and the log-likelihood (y-axis). Show at least one plot per algorithm that converges.

4 * Implementation of the Adam optimizer

To get a pass with distinction point, you can choose between this task or the task above

This task is only necessary to get one point toward a *pass with distinction* (VG) grade. Hence, if you do not want to get a *pass with distinction* (VG) point, you can ignore this part of the assignment.

Here we use the same data as in task 1 above.

```
library(uuml)
data("binary")

binary$gre_sd <- (binary$gre - mean(binary$gre))/sd(binary$gre)
binary$gpa_sd <- (binary$gpa - mean(binary$gpa))/sd(binary$gpa)
X <- model.matrix(admit ~ gre_sd + gpa_sd, binary)
y <- binary$admit
```

1. Implement the Adam mini-batch gradient descent algorithm (see Ruder, 2016, for details and psuedo-code) and (Kingma and Ba, 2014, for details) as an R function. Not that in Kingma and Ba (2014), the stepsize is denoted by α.

   Note! Each epoch should include all data points. A tip is to order the data points randomly in each iteration.

   A boilerplate for the algorithm can be found here.

   ```
   adam <- function(y, X, sample_size, eta, beta1, beta2, epochs){
     # Setup output matrix
     results <- matrix(0.0, ncol = ncol(X) + 2L, nrow = epochs)
     colnames(results) <- c("epoch", "nll", colnames(X))

     # Run algorithm
     theta <- rep(0.0, ncol(X)) # Init theta to the 0 vector
     for(j in 1:epochs){

       ### Put the algorithm code here ###

       # Store epoch, nll and output results
       results[j, "epoch"] <- j
       results[j, "nll"] <- ll(y, X, theta)
       results[j, -(1:2)] <- theta
     }
     return(results)
   }
   ```

   The function can then be called as:

   ```
   result <- mbsgd(y, X, sample_size = 10, eta = 0.1, epochs = 500)
   ```

2. Print the function in your report.

3. Try different learning parameters η, β_1 and β_2, and run the algorithm for roughly 500 epochs. You should test three different parameters setup. Visualize your results in the same way as in the SGD task above, namely:

   (a) (2p) The (negative) log-likelihood value *for all observations* for the given θ (y-axis) and the epochs (full data iterations, x-axis).

   (b) (2p) The value of one θ parameter of your choice (y-axis) and the epochs (full data iterations, x-axis). Also include the true values you got from using the `glm()` function above as a horizontal line in the figure.

References

Sebastian Ruder. An overview of gradient descent optimization algorithms. *arXiv preprint arXiv:1609.04747*, 2016.

Trevor Hastie, Robert Tibshirani, and Jerome Friedman. *The elements of statistical learning: data mining, inference, and prediction*. Springer Science & Business Media, 2009.

Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.