STA414/2104: Practice Problems 4

January, 2018

These practice problems are not for credit. Students may complete independently, in groups, or however they like. Treat these as representative of what might be on the tests.

Questions involving coding may be completed in any language. If you would like help regarding your language of choice from the course team, use R or Python. There will not be any *code*-related questions on the tests, but you will be asked about comuptational algorithms.

1. *Logistic Regression.* Consider the logistic regression model discussed in class. Suppose we observe a dataset $(\mathbf{x}_n, t_n), n = 1 \dots N$ of p-dimensional features $\mathbf{x}_n$ and binary targets $t_n \in \{0, 1\}$. We wish to predict the probability that $t_n = 1$, which we call y_n. The probability mass function for a single datapoint is then

$$p(t_n|y_n) = y_n^{t_n}(1 - y_n)^{1-t_n}$$

We model y_n as a function of $\mathbf{x}_n$ and parameters $\mathbf{w}$ using the logistic sigmoid function,

$$y(\mathbf{x}_n, \mathbf{w}) = \frac{1}{1 + \exp(-\mathbf{x}_n'\mathbf{w})}$$

(a) Write down the likelihood function, and the log-likelihood function, as functions of $\mathbf{w}$.

(b) Find the gradient of the negative log-likelihood function (the lecture slides walk you through this).

(c) Load the Iris dataset, either in `R` using `data(iris)` or in Python from the `sklearn-datasets` module. We are going to build a model for predicting whether the species is *versicolor* or *virginica* using logistic regression. We need to

- Load the dataset
- Filter out the flowers of species *setosa*
- Define the binary target

$$t_n = \begin{cases} 1 \text{ if species} = \text{versicolor} \\ 0 \text{ else} \end{cases}$$

Here is some example `R` code for doing this. You can use R or Python though.

```
suppressMessages({
  suppressWarnings({
    library(dplyr)
  })
})
```

```
data(iris) # Load the data from R's disk storage
iris_data <- iris %>% # Take the original data and modify it
  filter(Species != "setosa") %>% # Keep only flowers having Species == setosa
  mutate(target = if_else(Species == "versicolor",1,0)) %>% # Create the binary target
  select(-Species) %>% # Drop the original species variable
  rename(petal_width=Petal.Width,
         petal_length=Petal.Length,
         sepal_width=Sepal.Width,
         sepal_length=Sepal.Length) %>% # Make the variable names prettier
  tbl_df()

iris_data
```

```
## # A tibble: 100 x 5
##    sepal_length sepal_width petal_length petal_width target
##           <dbl>       <dbl>        <dbl>       <dbl>  <dbl>
##  1          7.0         3.2          4.7         1.4      1
##  2          6.4         3.2          4.5         1.5      1
##  3          6.9         3.1          4.9         1.5      1
##  4          5.5         2.3          4.0         1.3      1
##  5          6.5         2.8          4.6         1.5      1
##  6          5.7         2.8          4.5         1.3      1
##  7          6.3         3.3          4.7         1.6      1
##  8          4.9         2.4          3.3         1.0      1
##  9          6.6         2.9          4.6         1.3      1
## 10          5.2         2.7          3.9         1.4      1
## # ... with 90 more rows
```

(a) Implement gradient descent to minimize the loss function (minus the log-likelihood) with respect to $\mathbf{w}$.

(b) Fit your model. This means

- Standardize your features to each have mean 0 and variance 1 (so, subtract each feature's mean and divide by its standard deviation). You can do this before doing the train/test split.
- Randomly split the data into a training and test dataset. A 70/30 split is standard (for no real reason).
- Fit the model to the training set using your gradient descent code
- Use the fitted $\mathbf{w}$ to predict t_n on both the training and the test set. This means evaluate $y(\mathbf{x}_n, \mathbf{w})$

for each $\mathbf{x}_n$; this gives you a number between $(0, 1)$. If it's > 0.5, assign $\hat{t}_n = 1$ to $\mathbf{x}_n$.

- Report the *training accuracy* and the *test accuracy* of your model. This means use your $\mathbf{w}$ (from fitting the model on the training set) to predict on the training and test sets, and calculate the proportion of correct classifications.

Here is some example data cleaning code:

```
iris_clean <- iris_data %>%
  mutate_at(c("sepal_width","petal_width","sepal_length","petal_length"),
            funs( (. - mean(.)) / sd(.)))

# Train test split
idx <- sample(1:nrow(iris_clean),floor(nrow(iris_clean) * 0.7),replace=FALSE)
iris_train <- iris_clean[idx, ]
iris_test <- iris_clean[-idx, ]

nrow(iris_train)
```

```
## [1] 70
```

```
nrow(iris_test)
```

```
## [1] 30
```

2. *Logistic Regression, Exponential Family.* We can derive the above fitting procedure using the theory of exponential families. This will serve to explain where the softmax function comes from.

   (a) Write your likelihood function from part 1 in exponential family form. The natural parameter is $\eta_n = \log \frac{y_n}{1-y_n}$ and the sufficient statistic is $u_n = t_n$.

   $$L(y_n) = \prod_{n=1}^{N} g(\eta_n) \times \exp\left(\sum_{n=1}^{N} \eta_n u_n\right)$$

   (b) Now, write the natural parameter as a linear function of parameters and features, $\eta_n = \mathbf{x}_n' \mathbf{w}$. Write down the likelihood and log-likelihood now as a function of $\mathbf{w}$.

   (c) Invert the natural parameter to get y_n as a function of η_n. The result should look familiar.

   (d) Use the above log-likelihood to find the maximum likelihood estimate of $\mathbf{w}$. You won't get a closed-form expression; just a system of equations that can be solved iteratively. Rearrange this (so all terms are on one side and zero is on the other side), and compare it to the equations that need to be solved in your gradient descent approach above.

   Hint: it's helpful to use the chain rule to compute the necessary log-likelihood derivatives,

   $$\frac{\partial \ell}{\partial \mathbf{w}} = \frac{\partial \ell}{\partial \boldsymbol{\eta}} \times \frac{\partial \boldsymbol{\eta}}{\partial \mathbf{w}}$$

   where $\boldsymbol{\eta} = (\eta_1, \ldots, \eta_N)$. You may notice that while the log-likelihood is not a linear function of $\mathbf{w}$, it *is* a linear function of $\boldsymbol{\eta}$, so gradients with respect to $\boldsymbol{\eta}$ can be computed element-wise. The jacobian $\frac{\partial \boldsymbol{\eta}}{\partial \mathbf{w}}$ works out very nicely when computed element-wise as well. And as a side-note, this is super sweet since if we had a different exponential family distribution, the only thing that would change in the above would be $\frac{\partial \boldsymbol{\eta}}{\partial \mathbf{w}}$. These are called *generalized linear models*.