SOLUTIONS TO EXERCISES IN STATISTICAL LEARNING

MARTIN HJELM

0.1. **Exercise 2.9.** Prove the inequality for the expected value of residual sum of squares for a linear regression estimation parameter,

$$E_{X,Y}[R_{tr}(\hat{\beta}_X)] \leq E_{\tilde{X},\tilde{Y}}[R_{te}(\hat{\beta}_X)]. \tag{1}$$

Solution
This problem was a bit tricky but it was a great way to review OLS and projection matrices etc. I came up with three approaches. The first two are quite hand wavy and the third more proof-like.

First approach: To show this we need two results which we get from Leber et al. - Linear Regression Analysis.

Let $Y = X\beta + \epsilon$ where ϵ is the error with zero mean. From the proof of theorem 3.3 in Leber we have if $E[Y] = X\beta$ where X is $n \times p$ of rank r, where $r \leq p$ and $Var[Y] = \sigma I$ then

$$E[RSS] = \sigma^2(n - r) \tag{2}$$

From theorem 3.2 of Leber it follows that β is among the class of linear unbiased estimates, the estimate with minimum variance. This implies that if we use our estimated $\hat{\beta}$ for any other dataset sampled from the same distribution, then the variance over that dataset will be greater or equal to the original variance. Since we are sampling the data points from the same distribution we can assume that the factor $n - r$ does not play a role.

I am guessing here that I am stretching it a bit. This problem was hard. A more thorough proof would probably need to manipulate both sides into expressions involving only σ and using theorem 3.2 of Leber to for claiming the inequality.

Second approach: For a linear regression problem, $Y = X\beta + \epsilon$, we have that,

$$\hat{Y} = X\hat{\beta} = HY, \tag{3}$$

where H is the symmetric idempotent projection matrix onto the column space of X. The distance to Y from the projection HY is just the orthogonal distance from Y to the space spanned by X.

This means that no other distance is smaller since the smallest distance between two points is the straight line (unless the space is warped...). Imagine now that we sample new X and Y such that we get $\tilde{Y}$ and $\tilde{X}$, then the projection by H is now no longer the orthogonal projection of $\tilde{Y}$ onto to the space spanned by $\tilde{X}$. As such the distance is no longer the smallest. Therefore the expected RSS will be higher for the right hand side of the expression.

Third approach: Assume a linear model between the random variables, (X, Y), such that the following holds:

$$
\begin{aligned}
&\text{Model}: Y = X\beta + \varepsilon \\
&\qquad E[\varepsilon] = 0,\ Var(\varepsilon) = \sigma^2 \\
&\qquad Var(Y|X) = N\sigma^2 \\
&\qquad E[Y|X] = X\beta \\
&\qquad \text{LSE}(\beta):\ \hat{\beta} = (X^TX)^{-1}X^TY \\
&\qquad E[\hat{\beta}] = \beta \\
&\qquad Var(\beta) = \sigma^2(X^TX)^{-1}
\end{aligned}
\tag{4}
$$

where X is a $N \times p$ matrix, Y and β is a $N \times 1$ vector.

Since there is linear relations between X and Y, we start by computing the conditional expectation of the RSS over Y given X and then take the expectation over X if needed,

(5)

$$
\begin{aligned}
&E[RSS|X] = E[(Y - X\hat{\beta})^2|X] = E[Y^TY|X] - 2E[Y^TX\hat{\beta}|X] + E[(X\hat{\beta})^2|X] \\
&E[Y^TY|X] = Var(Y|X) + E[Y|X]^2 = N\sigma^2 + \beta^TX^TX\beta \\
&E[Y^TX\hat{\beta}|X] = \operatorname{Tr} Cov(Y^T, X\hat{\beta}|X) + E[Y|X]^TE[\hat{\beta}|X] = \sigma^2 \operatorname{Tr} X(X^TX)^{-1}X^T + \beta^TX^TX\beta \\
&E[(X\hat{\beta})^2|X] = \operatorname{Tr} Cov(X\beta|X) + \beta^TX^TX\beta = \sigma^2 \operatorname{Tr} X(X^TX)^{-1}X^T + \beta^TX^TX\beta \\
\Rightarrow &E[RSS|X] = N\sigma^2 - \sigma^2\operatorname{Tr} X(X^TX)^{-1}X^T = N\sigma^2 - (p+1)\sigma^2 = (N-1-p)\sigma^2
\end{aligned}
$$

where $p + 1$ is the rank of the projection matrix.

For the right hand side of the inequality we have,

$$
E[RSS|X, \tilde{X}] = N\sigma^2 - \sigma^2\operatorname{Tr} \tilde{X}(X^TX)^{-1}X^T. \tag{6}
$$

For most cases this means that,

$$\operatorname{Tr} \tilde{X}(X^T X)^{-1} X^T \leq p + 1 \tag{7}$$

Thus,

$$E[R_{tr}(\hat{\beta}_X)] \leq E[R_{te}(\hat{\beta}_X)]. \quad \square \tag{8}$$

0.2. **Exercise 3.12.** Show that the ridge regression estimates can be obtained by ordinary least squares regression on an augmented data set.

Solution
Let,

$$X_{new} = \begin{bmatrix} X \\ \sqrt{\lambda} I_p \end{bmatrix} \; y_{new} = \begin{bmatrix} y & 0_p \end{bmatrix}, \tag{9}$$

we can then write $\hat{\beta}$ as,

$$\begin{aligned} \hat{\beta} &= \Big(\begin{bmatrix} X & \sqrt{\lambda} I_p \end{bmatrix} \begin{bmatrix} X \\ \sqrt{\lambda} I_p \end{bmatrix} \Big)^{-1} \begin{bmatrix} X & \sqrt{\lambda} I_p \end{bmatrix} \begin{bmatrix} y \\ 0_p \end{bmatrix} \\ &= \Big(X^T X + \lambda I_p \Big)^{-1} X^T y, \end{aligned} \tag{10}$$

which is equal to 3.43. $\square$

0.3. **Exercise 3.29.** Show that for ridge regression the same parameters are obtained if the original dataset is augmented with identical data.

Solution
In ridge regression the solution to minimizing the RSS is

$$\Big(X^T X + \lambda I \Big) \beta = X^T y \tag{11}$$

If we let

$$
\begin{aligned}
&X_{new} = [X_1, X_2, \ldots, X_m] \text{ where } X_1 = X_2 = \ldots = X_m \\
&y_{new} = [y_1, y_2, \ldots, y_m] \text{ where } y_1 = y_2 = \ldots = y_m \\
&A = \left(X^T X + \lambda I\right) \\
&v = X^T y \\
&\beta = A^{-1} v
\end{aligned} \tag{12}
$$

for some $m \in \mathcal{N}$. If we assume we specify a λ such that A is invertible then the inverse of A will also be symmetric. Further on, since A's all of-diagonal element are the same, all of the inverse's of-diagonal elements will be the same. It is easy to see that all the entries of v are the same. This means that all the entries of β will be the same as the multiplication $A^{-1}v$ are just the same entries multiplied over and over again for each entry in β. □

0.4. **Exercise 4.7.** Does the formulation,

$$
\begin{aligned}
&\arg\min_{\beta,\beta_0} D^*(\beta, \beta_0) = -\sum_{i=1}^{N} y_i (x_i^T \beta + \beta_0) \\
&sb.\,to.\ \|\beta\| = 1
\end{aligned} \tag{13}
$$

solve the optimal hyperplane problem?

Solution
Eq.13 is the sum of the signed distances to the plane, where the sign is negative if correctly classified by the plane and positive otherwise. The optimal hyperplane is the plane that separates the two classes and maximizes the distance to the closest point from the two classes. If we consider two classes with just one point in each that is linearly separable then we can place a plane in between the two points. This plane can be moved in the normal direction of the plane to be close to either of the two points. The sum will still be the same and the points are still correctly classified, we have just made one of the distances smaller and the other longer. Therefore the optimization problem will not solve the problem of finding the optimal hyperplane all the time it will depend on the starting point. □

0.5. **Exercise 5.11.** Prove that for a smoothing spline the null space of K is spanned by functions linear in X

Solution
This one is a bit tricky. I am not sure I got it right. If we assume S_λ is described by functions linear in x, that is, $\alpha + \beta x$, then,

$$\begin{aligned} &\Omega = 0 \implies N(N^T N + \underbrace{\lambda\Omega}_{=0})^{-1} N^T u = \mu u \\ &\implies Pu = \mu u \implies \mu = \{1, 0\}. \end{aligned} \tag{14}$$

This holds for both constant and linear basis functions so we have two eigenvalues of value 1. This corresponds to zero valued eigenvalues of K since,

$$p_k(\lambda) = \frac{1}{1 + \lambda d_k} = 1,\ i = 1, 2 \implies d_1 = d_2 = 0 \tag{15}$$

This implies that $(S - \lambda I)^{-1} w = Kw = 0$ is spanned by functions constant and linear in x. □

0.6. **Exercise 5.13.** Derive the N-fold cross validation for a smoothing spline.

Solution
For a smoothing spline we have $\hat{f} = S_\lambda y$ where S_λ only depends on X and λ. We have

$$\hat{f}_\lambda(x_i) = \sum_j S_{ij} y_j \tag{16}$$

Define:

$$\tilde{y} = \begin{cases} y_j & j \neq i \\ \hat{f}_\lambda^{-i}(x_i) & j = i \end{cases} \tag{17}$$

where $\tilde{f}_\lambda^{-i}(x_i)$ is the estimate of f with data $\tilde{y}$. Using the Leave-one-out lemma of Wahba we have,

$$\begin{aligned} &\tilde{f}_\lambda^{-i}(x_i) = \hat{f}_\lambda^{-i}(x_i), i = 1, \ldots, u \\ &\implies \hat{f}_\lambda^{-i}(x_i) = \tilde{f}_\lambda^{-i}(x_i) = \sum_{j=1}^{n} S_{ij} y_j = \sum_{j \neq i} S_{ij} y_j + S_{ii} \hat{f}_\lambda^{-i}(x_i) \\ &\qquad = \sum_{j=1}^{n} S_{ij} y_j - S_{ii} y_i + S_{ii} \hat{f}_\lambda^{-i}(x_i) \\ &\implies y_i - \hat{f}_\lambda^{-i}(x_i) = \frac{y_i - \hat{f}(x_i)}{1 - S_{ii}} \end{aligned} \tag{18}$$

From here 5.27 follows from substitution. □

0.7. **Exercise 6.5.** Ex. 6.5 Show that fitting a locally constant multinomial logit model of the form (6.19) amounts to smoothing the binary response indicators for each class separately using a Nadaraya–Watson kernel smoother with kernel weights $K_\lambda(x_0, x_i)$

Solution
To be honest I did understand the question fully but I have tried to understand as best as I could. My main assumptions were that the cml likelihood in 6.19 only depended on x in the kernel, that is, $\beta_{g_i}(x_0) = 0$. Putting the derivative with respect to β_{g_i} to zero we get after some manipulation,

$$\sum_{i \in g_i}^{N} K(x_i, x_0) = \frac{e^{\beta_{g_i}}}{1 + \sum_{k=1}^{J-1} e^{\beta_{g_i}}} \tag{19}$$

This is exactly the class probability for class g_i written using a kernel smoother. In the book they use a binary indicator variable for binomial case, see Eq. 4.20. Here the indicator variable is baked into g_i. So if this is what they mean than qed.

However, the probability for a class using the Nadaraya–Watson kernel smoother is, using Bayes' rule,

$$p(y|x) = \frac{\sum_{y_i=y}^{N} K(x_i, x)}{\sum_{i}^{N} K(x_i, x)} \tag{20}$$

Which is not very similar to equation 19.

0.8. **Exercise 7.5.** For a linear smoother $\hat{y} = Sy$ show that

$$\sum_{i=1}^{N} Cov(\hat{y}_i, y_i) = \mathrm{trace}(S)\sigma_\epsilon^2 \tag{21}$$

Solution
We have,

$$\begin{aligned} y &= f(x) + \epsilon, \\ \hat{y} &= Sy, \\ Var(y) &= \sigma_\epsilon^2. \end{aligned} \tag{22}$$

Then,

$$
\begin{aligned}
Cov(\hat{y}_i, y_i) &= E[(\hat{y}_i - \hat{\bar{y}}_i)(y_i - \bar{y}_i)] \\
&= \left\{ \hat{y} = Sy \implies \hat{y}_i = \sum_j S_{ij} y_j \right\} \\
&= \sum_j S_{ij} E[(y_j - \hat{y}_j)(y_i - \bar{y}_i)] = \left\{ y_j, y_i \text{ are i.i.d} \right\} \\
&= S_{ii} Var(y_i) = S_{ii} \sigma_\epsilon^2 \quad \square
\end{aligned} \tag{23}
$$

0.9. **Exercise 8.2.** Find

$$
\arg\max_{\tilde{P}} F(\Theta', \tilde{P}) = E_{\tilde{P}}[l_o(\Theta';)] - E_{\tilde{P}}[\ln \tilde{P}],\ \tilde{P} = \tilde{P}(Z_m) \tag{24}
$$

Solution
Book says to use Lagrange multiplier but a simpler solution can be found by rewriting in the following way,

$$
\begin{aligned}
F(\Theta', \tilde{P}) =& \sum_{Z^m} \tilde{P} \ln P(Z, Z^m | \Theta') - \sum_{Z^m} \tilde{P} \ln \tilde{P} \\
=& E_{\tilde{P}}[\ln P(Z^m | Z, \Theta')] + E_{\tilde{P}}[\ln P(Z | \Theta')] - E_{\tilde{P}}[\ln \tilde{P}] \\
=& E_{\tilde{P}}[\ln P(Z | \Theta')] - KL(\tilde{P} | P(Z^m | Z, \Theta'))
\end{aligned} \tag{25}
$$

The KL-divergence is always ≥ 0 with equality only iff $\tilde{P} = P(Z^m | Z, \Theta)$. Since the first term is independent of Z^m it follows that F is maximized when $\tilde{P} = P(Z^m | Z, \Theta)$.

0.10. **Exercise 9.5. (a)** In terms of the number of terminal nodes m, give a rough formula for the degrees of freedom of the fit.

Solution
According Hastie the degrees of freedom in classical statistics are the number of independent parameters.

If we think of a binary tree as a partitioning of space in $\mathbb{R}^d$, of a subset of the variables involved in a decision tree. Then we would have for each a split one degree of freedom, one split in space. m end nodes would then mean #m degrees of freedom.

(b,c,d) Evaluate the measure empirically and compare to the estimate.

Solution
As we we can see the difference in value between the estimate and the empirical value are quite big.

Further on, trying different number of samples gives weird results showing no linear relationship with the number of end nodes and dfg. I might be doing something wrong or it just lays in the independence between X and Y.

- Dfg for tree with # 1(2) terminal nodes: 1.10035156084
- Dfg for tree with # 5(10) terminal nodes: 1.28341040833
- Dfg for tree with # 10(20) terminal nodes: 1.53500925173
- Dfg for tree with # 20(40) terminal nodes: 1.80548043202

```
import numpy as np
from sklearn.tree import DecisionTreeRegressor

Ntnodes = np.array((2,5,10,20))
dfg = np.zeros((len(Ntnodes)))
N = 10
d = 10
Niter = 100
sigmaY = 1.0
Yall = np.zeros((N,Niter))
Ypredall = np.zeros((N,Niter))
for itm, ntm in enumerate(Ntnodes):
    for s in range(0,Niter):
        # 1. Generate 100 observations of 10 standard Gaussian variables
        X = np.zeros((N,d))
        muX = 100 * np.random.random(10) - 5
        sigmaX = 100 * np.random.random_sample(10)
        X = np.random.normal(muX,sigmaX,size=(N,10))

        # 2. Generate response values also as standard Gaussian (sigma2 = 1),
        # independent of the predictors.
        muY = 100 * np.random.random_sample() - 50
        Y = np.random.normal(muY,sigmaY,N)

        # 3. Fit regression trees to the data of fixed size 1,5 and 10 terminal
        # nodes and hence estimate the degrees of freedom of each fit.
        estimator = DecisionTreeRegressor(random_state=0, max_leaf_nodes=ntm)
        estimator.fit(X,Y)
        yPred = estimator.predict(X)
        dfg[itm] += (Y-np.mean(Y)).dot((yPred-np.mean(yPred)).T)

        # if itm == 2 and s == 0:
            # get_code(estimator, df.columns)
```

```
dfg[itm] /= Niter
print "Dfg for tree with #",ntm, "terminal nodes:", dfg[itm]
```

(e) Suggest a way to compute an approximate S matrix for a regression tree, compute it and compare the resulting degrees of freedom to those in **a** and **c**.

Solution
One way would be to treat the decision rules in the tree as basis functions, h_i, and then compute the hat matrix from the resulting X as in linear regression. I did not implement this since extraction of the decision rules from numpy would take way to long.

0.11. **Exercise 10.7.** Show that the solution to

$$\gamma_{jm} = \arg\min_{\gamma_{jm}} \sum_{x_i \in R_{jm}} L(y_i, f_{m-1}(x_i) + \gamma_{jm}) \tag{26}$$

is

$$\gamma_{jm} = \log \frac{\sum_{x_i \in R_{jm}} w_i^m I(y_i = 1)}{\sum_{x_i \in R_{jm}} w_i^m I(y_i = -1)} \tag{27}$$

Solution
Eq.26 is the expected exponential loss. Friedman(2000) gives that for a current model $F(x)$ and an update $f(x)$, the $f(x)$ that minimizes

$$\arg\min_{f(x)} E[e^{-y(F(x)+f(x))}] \tag{28}$$

is

$$f(x) = \frac{1}{2} \log \frac{E_w[I_{[y=1]}|x]}{E_w[I_{[y=-1]}|x]} \tag{29}$$

which is the same as in Eq.27 where the weights in E_w are given by

$$w_i^m = e^{-y_i f_{m-1}(x_i)} \tag{30}$$

□

0.12. **Exercise 10.9.** Consider a K-class problem where the targets y_{ik} are coded as 1 if observation i is in class k and zero otherwise. Using the multinomial deviance loss function (10.22) and the symmetric logistic transform, use the arguments leading to the gradient boosting Algorithm 10.3 to derive Algorithm 10.4. Hint: See exercise 10.8 for step 2(b) iii.

Solution

0.13. **The Gradient Boosting Algorithm.** The gradient boosting (GB) problem can be formulated as follows,

$$\text{For } \{X_{tr}, y_{tr}\} \ \min_{f \in F} L[f] \tag{31}$$

where,

$$f(x) = \sum_{T} \alpha_t h_t(x, \theta_t), \ \forall t : h \in H, \alpha_t \in \mathbb{R} \tag{32}$$

where T is the index set $T = \{1, 2, \ldots, M\}$, α_t constants and $h_t(x, \theta_t)$ is some base learner parametrized by θ_t. Simply put we want to fit an additive model, f, such that it minimizes some specific loss function, L, over the (training) dataset $\{X_{tr}, y_{tr}\}$.

L can be considered a functional that operates on a function, f, where the points f are evaluated at is restricted to the set of observed data points, that is,

$$L[f] \to \mathcal{R} \tag{33}$$

One approach to minimizing L is to perform gradient descent, for this we need the derivate of L with respect to f, the functional derivative. In most cases it will not be possible to obtain $f = -\nabla L[f]$, because finding the exact solution to the parameters of the new f is infeasible, for example for a tree. In addition, the derivative will only be defined at the observed data points which implies that any gradient descent will overfit to the training data. However, if we perturb the current f with a function g for some small ϵ we can write,

$$L[f + \epsilon g] = L[f] + \epsilon \langle \nabla L[f], g \rangle + \mathcal{O}(\epsilon^2). \tag{34}$$

We see that instead of using the derivative as the update we can find the function that maximizes the inner product, $-\langle \nabla L[f], g \rangle$. This means that we try to find an update that is projected onto the direction of the gradient.

When computing the gradient it turns out that if the functional does not contain derivatives of the function, then we can treat the function in the functional analogously to a variable of an ordinary function(see Wikipedia page on functional derivatives). For example, for squared error loss we get,

$$\nabla L[f] = \nabla \sum_{i=1}^{N} \frac{1}{2}(y_i - f(x_i))^2 = \begin{cases} \sum_{i=1}^{N}(y_i - f(x_i)) & \text{if } (x,y) = (x_i, y_i) \\ undef & else \end{cases} \tag{35}$$

With the basic building blocks for computing the gradient out of the way we proceed to formulate the generic gradient boosting algorithm as,

Algorithm 1 Gradient Boosting Algorithm

1: Set set of base learners $h(x, \theta) \in H$
2: Set max number of base learners M
3: Set loss function(al) $L[f]$
4: Init f_0 to a constant
5: **for** $t = 1$ to M **do**
6: Compute the gradient of the loss $\nabla L[f_{t-1}]$
7: Learn the parameters θ_t for h_t that maximizes the inner product $\langle \nabla L[f_{t-1}], h_t \rangle$
8: Compute the optimal step size α_t by: $\arg\min_{\alpha_t} L[f_{t-1} + \alpha_t h_t]$
9: Set $f_t = f_{t-1} + \alpha_t h_t$
10: **end for**

0.14. **The Gradient Boosting Algorithm for K-class Classification Using Trees.** For modeling K-classification we can use a K-class logistic model,

$$p_k(x) = \frac{e^{f_k(x)}}{\sum_{l=1}^{K} e^{f_l(x)}}, \tag{36}$$

where $f_K(x) = 0$ and $\sum_{l=1}^{K} f_k(x) = 0$. The centering condition is for numeric stability, that is, we do not want f_k to run amok. That this is useful can be seen by adding an arbitrary constant to each f_k and noting that the probabilities are still constant. The additive model is now baked into the multinomial such that each class exponential is an additive model. This means that we are fitting K individual additive tree models.

The loss function we want to use is multinomial loss function which is just the negative log-likelihood of the multinomial,

$$L(y, p(x)) = -\sum_{k=1}^{K} I(y_i = \mathcal{G}_k) \log p_k(x) = -\sum_{k=1}^{K} I(y_i = \mathcal{G}_k) f_k(x) + \log \sum_{l=1}^{K} e^{f_l(x)}, \tag{37}$$

To formulate the gradient boosting algorithm for classification using the K-class logistic model with its accompanying loss function, with trees as the base learner we must:

- compute the first and second derivatives of the loss function(al)
- formulate the maximization of the tree parameters over the inner product with the gradient

0.15. **Derivatives for the K-class Logistic Loss Function(al).** The loss function(al) gradient is ($p_{ki} = p_k(x_i), f_{ki} = f_k(x_i)$),

$$-g_{ik} = \frac{\partial L(y_i, f_{1i}, f_{2i}, \ldots, f_{Ki})}{\partial f_{ki}} = I(y_i = \mathcal{G}_k) - p_{ki}, \tag{38}$$

and the second derivative,

$$-h_{ikm} = \frac{\partial L(y_i, f_{1i}, f_{2i}, \ldots, f_{Ki})}{\partial f_{ki}\, \partial f_{mi}} = \begin{cases} -p_{ki}\,(1 - p_{ki}) & k = m \\ -p_{ki}\, p_{ki} & k \neq m \end{cases} \tag{39}$$

where h_{ikm} is the second derivate with respect to class k and m.

0.16. **Computing the Gradient Projection Tree Parameters.** Each of the K additive tree models can be written as,

$$f_M = \sum_{m=1}^{M} T(x; \Theta_m),\ \Theta = \{R_j, \gamma_j{}_1^J\} \tag{40}$$

The optimization in each step, m, becomes,

$$\arg\min_{\Theta_m} \sum_{i=1}^{N} L(y_i, f_{m-1} + T(y_i, \Theta_m)) \Rightarrow \arg\max_{\Theta_m} \langle \nabla L, T(x; \Theta_m) \rangle \tag{41}$$

The crux with computing the projection parameters is that the tree's terminal regions, R_j, are difficult to find simultaneously with the responses in each region, γ_{R_j}. Finding them separately is still hard for the terminal regions but simpler for the responses. However, it turns out that fitting the terminal regions as regression tree using squared-error-loss is as simple as fitting a single tree.

Thus by fitting the gradients for each class, g_{ik}, using a regression tree with squared error loss we can find the terminal regions, R_{kj}, that approximates the projection. The caveat is that these regions are not the ones that minimizes the loss given the current additive model but luckily they are close enough.

Once we we have the terminal regions we need to compute the updates for the responses in the terminal regions. Eq.41 does not have a closed form solution for optimizing γ but we can use a Newton-Raphson step in the direction towards the zero of the derivative of the multinomial deviance loss function. The Newton-Raphson method gives the following update, putting $\gamma_k^0 = 0$, and using the diagonal of the Hessian, $m = k$, gives,

$$\gamma_k^1 = \gamma_k^0 - \frac{L'}{L''} = \frac{\sum_{x_i \in R}(y_{ki} - p_{ki})}{\sum_{x_i \in R} p_{ki}\,(1 - p_{ki})} \tag{42}$$

To retain the symmetric parametrization, such that the updates all sums to zero, we can demand that the symmetric logit transform holds for the update,

$$\hat{\gamma}_k^t = \gamma_k^t - \frac{1}{K}\sum_{l=1}^{K} \gamma_l^t. \tag{43}$$

This will ensure that the centering condition is fulfilled. Averaging over all possible base classes gives the update rule,

$$\hat{\gamma}_k^t = \frac{K-1}{K}\Big(\gamma_k^t - \frac{1}{K}\sum_{l=1}^{K} \gamma_l^t\Big). \tag{44}$$

Since we are fitting the tree to the gradients, $g_{ik} = I(y_i = \mathcal{G}_k) - p_{ki}$ we can simplify the expression in the parentheses a bit, and write,

$$\gamma_{jkm} = \frac{K-1}{K}\Big(\frac{\sum_{x_i \in R_{jkm}} g_{ikm}}{\sum_{x_i \in R_{jkm}} |g_{ikm}|(1 - |g_{ikm}|)}\Big). \tag{45}$$

The absolute values here are just to make sure that the denominator never becomes zero. The replacement of the γ minus the means are either due some simplification or a trick in Friedman et al. which I haven't figured out yet.

0.17. **Putting It All Together.** Finally we can put the algorithm together,

Algorithm 2 Gradient Boosting for K-class Classification

Set $f_{k0} = 0$ for $k = 1, 2, \ldots, K$
for $t = 1$ to M **do**
  Set $p_k(x) = \frac{e^{f_k(x)}}{\sum_{l=1}^{K} e^{f_l(x)}}$, for $k = 1, 2, \ldots, K$
  for $k = 1$ to K **do**
    Compute $g_{ikm} = y_{ik} - p_k x_i,\ i = 1, 2, \ldots, N$
    Fit a regression tree to the gradients g_{ikm} using
    squared error loss, giving J_m terminal regions R_{jkm}
    Use the terminal regions to compute the Newton-Raphson update
    to the responses for each terminal region.
    $\gamma_{jkm} = \frac{K-1}{K}\left(\frac{\sum_{x_i \in R_{jkm}} g_{ikm}}{\sum_{x_i \in R_{jkm}} |g_{ikm}|(1-|g_{ikm}|)}\right)$
    Update $f_{km}(x) = f_{k,m-1} + \sum_{j=1}^{J_m} \gamma_{jkm} I(x \in R_{jkm})$
  end for
end for

0.18. **Exercise 11.2.** Show that if g_k is the identity output and the weights α_m are nearly zero, then the model is linear in the inputs.

Solution
The only nonlinear transformation that happens in a network is at the sigmoid function. Because of this all we have to do is prove that the sigmoid function is linear in the input when close to zero. The Mclaurin series, that is, the Taylor expansion around 0 gives,

$$\sigma(x) \approx \frac{1}{2} + \frac{1}{4}x - \frac{1}{48}x^3 + \frac{1}{480}x^5 + \ldots \tag{46}$$

If x is small only the two first terms will matter and therefore the network is linear in the inputs. □

0.19. **Exercise 11.3.** Derive the forward and backward propagation equations for the cross-entropy loss function.

Solution
This question is strange. Either it is very simple or I am missing something. The difference in using the cross-entropy to the squared error loss is just the difference between the derivatives. All we have to do is update eq. 11.12 in the book with the $\log$ derivative using the chain rule. This is trivial and therefore I will omit it.

0.20. **Exercise 11.4.** Consider a neural network for a K class outcome that uses cross-entropy loss. If the network has no hidden layer, show that the model is equivalent to the multinomial logistic model described in Chapter 4.

Solution
Since we have no hidden layer T becomes a linear function of the inputs,

$$T_k = \beta_{ok} + \beta_k^T X \tag{47}$$

Using the softmax function we get

$$g_k(T) = \frac{e^{\beta_{ok}+\beta_k^T X}}{\sum_{l=1}^K e^{\beta_{ok}+\beta_k^T X}}. \tag{48}$$

This shows that we have the same form for the class probabilities. To show equivalence we need to show that the update rules for the network reaches the same coefficients as the likelihood estimates in logistic model. We have

$$R(\theta) = \sum_{i=1}^N \sum_{k=1}^K y_{ik} \log f_k(x_i) = \sum_{i=1}^N \sum_{k=1}^K y_{ik} \Big(\beta_{ok} + \beta_k^T X - \log(\sum_{l=1}^K e^{\beta_{ol}+\beta_l^T X}) \Big) \tag{49}$$

If we let $K = 2$, ($K \geq 2$ is a bit more complicated) we get,

$$R(\theta) = \sum_{i=1}^N \sum_{k=1}^K y_{ik} \log f_k(x_i) = \sum_{i=1}^N (y_i \log f_1(x_i) + (1 - y_i) \log(1 - f_1(x_i)). \tag{50}$$

Which is the same as the log-probability eq. 4.20 in the book. Since the weight updates are done in the same way we have proven the equivavlence. □

0.21. **Exercise 12.1.** Show that the solution to the optimization problem

$$\arg\min_{\beta,\beta_0} \sum_{i=1}^N [1 - y_i f(x_i)]_+ + \frac{\lambda}{2}\|\beta\|^2, \tag{51}$$

with $\lambda = \frac{1}{C}$ is the same as for Eq.12.8

Solution
We can multiply Eq.51 by C as the optimization problem will not change, which gives us,

$$\arg\min_{\beta,\beta_0} C \sum_{i=1}^N [1 - y_i f(x_i)]_+ + \frac{1}{2}\|\beta\|^2. \tag{52}$$

Now

$$[1 - y_i f(x_i)]_+ = \begin{cases} 1 - y_i f(x_i) & y_i f(x_i) < 1 \\ 0 & else \end{cases} \tag{53}$$

Which means that penalization will only happen outside the margin. If we let,

$$\xi_i \geq 1 - y_i f(x_i),\ \xi_i \geq 0\ \forall i, \tag{54}$$

we can write eq. 51 as

$$\begin{aligned} &\arg\min_{\beta,\beta_0} C \sum_{i=1}^{N} \xi_i + \frac{1}{2}\|\beta\|^2 \\ &sb.to.\ \xi_i \geq 1 - y_i f(x_i),\ \xi_i \geq 0\ \forall i. \end{aligned} \tag{55}$$

We see that ξ_i will fulfill the conditions in 12.8 and with C as a constant the optimization function becomes the same. □

0.22. **Exercise 12.2.** Show that the solution to (12.29) is the same as the solution to (12.25) for a particular kernel.

Solution
To be honest I did not understand the question. Firstly if Exercise 12.1 holds then we have just proven the equivalence of the two optimization formulations. 12.29 is just the more generalized version drawing from chapter 5.8 and using the weighted sum of kernels to represent the function. Thus this seems like we are plugging in the kernel function representation into the hinge loss formulation and then ask if the solution still will be the same. But we have just proven that the hinge loss and the optimal hyperplane optimization formulations are the same! The solutions will be the same since this is a convex optimization problem. Anyway, I made a solution formulating the dual from 12.29.

We can use Ex.12.1 to rewrite Eq.12.29 into a form involving the slack variables,

$$\begin{aligned} &\arg\min_{\beta,\beta_0,C} C \sum_{i=1}^{N} \xi_i + \frac{1}{2}\alpha K \alpha \\ &sb.to.\ \xi_i \geq 1 - y_i f(x_i),\ \xi_i \geq 0\ \forall i \end{aligned} \tag{56}$$

Using the fact that

$$f(x) = \beta_0 + \sum_{i=1}^{N} \alpha_i K(x, x_i), \tag{57}$$

when forming the Langrangian we arrive after some manipulations, the same manipulations done in chapter 12.2.1 in the book, at the same form as in 12.19. □

0.23. **Exercise 12.6.** Show some results relating to FDA.

Solution a
We have,

$$\arg\min_{\theta,\beta} \sum_{i=1}^{N} (\theta(g_i) - x^T\beta)^2 \tag{58}$$

Replace x by $h(x)$, such that H is $N \times J$ with β as a $J \times 1$ vector. Let θ be the scoring vector of size $K \times 1$ and Y be the $N \times K$ indicator matrix, then it is obvious that we can write,

$$\arg\min_{\theta,\beta} \|Y\theta - H\beta\|^2 \tag{59}$$

Solution b
Firstly, we have $\Theta D_\pi 1 = \Theta \frac{1}{N} Y^T Y 1 = 0$, by inspection this is just that that the scores have zero mean over the class proportions.

Secondly we have $D_\pi = Y^T Y/N$ so that $\Theta^T D_\pi \Theta = \frac{1}{N}\Theta^T Y^T Y \Theta = 1$ which just says that the scores have unit variance.

Solution c
For a symmetric idempotent projection matrix S we have,

$$\begin{aligned} &S = S^2 \Rightarrow (1-S)^2 = (1-S) \\ &\hat{Y\theta} = H\beta = SY\theta \end{aligned} \tag{60}$$

this gives the following,

$$\begin{aligned} &\arg\min_{\theta,\beta} \|Y\theta - H\beta\|^2 \Rightarrow \arg\min_{\theta} \|Y\theta - SY\|^2 = (Y\theta - SY\theta)^T(Y\theta - SY\theta) \\ &\theta^T Y^T (I-S)^T (I-S) Y\theta = \theta^T Y^T (I-S) Y\theta = \theta^T Y^T (I-S) Y\theta = N1 - \theta^T Y^T S Y\theta \end{aligned} \tag{61}$$

The squared residuals can never be less than zero therefore to minimize the residuals we want to maximize $\theta^T Y^T S Y \theta$ with respect to θ.

Solution d
A projection matrix, P, has eigenvalues either 0 or 1. If $P = P^2$ and $Pv = \lambda v$ then

$$Pv = P^2 v = P\lambda v = \lambda^2 v = \lambda v \Rightarrow \lambda(1-\lambda) = 0 \quad \square \tag{62}$$

Solution e
This is shown in the Hastie paper on FDA. However, there $S = Y^T P Y$.

0.24. **Exercise 12.10.** Show that LDA can be kernelized and that the solution will not depend on any explicit $h(x)$

Solution
Like most questions in this book this one is vague too. Obviously the covariance matrix can be kernelized. Since LDA only uses covariance matrices the model will be fully kernelizable (γ does not depend on $h(x)$). There is a neat derivation of Kernel LDA in the paper by Mika et. al. (Fisher Discriminant Analysis With Kernels) using the fact that $w = \sum_i \alpha_i h(x_i)$, since the solution lays in the span of the training examples. However to show that W_b is kernelizable we can simply write the following,

(63)

$$\begin{aligned}
Sw &= \sum_{i \in C_1} \Big(h(x_i) - m_1\Big)^T \Big(h(x_i) - m_1\Big) + \sum_{i \in C_2} \Big(h(x_i) - m_2\Big)^T \Big(h(x_i) - m_2\Big) \\
&= \sum_{i \in C_1} h_i^T h_i + m_1^T m_1 - 2h(x_i)^T m_1 + \ldots = \{h(x_i) = h_i,\ k_{ij} = k(x_i, x_j) = h_i^T h_j,\ m_1 = \frac{1}{N_{C_1}} \sum_{i \in C_1} h_i\} \\
&= \sum_{i \in C_1} k_{ii} + \big(\frac{1}{N_{C_1}} \sum_{i \in C_1} h_i\Big)^T \big(\frac{1}{N_{C_1}} \sum_{i \in C_1} h_i\Big) - \frac{2}{N_{C_1}} h_i^T \sum_{j \in C_1} h_j + \ldots \\
&= \sum_{i \in C_1} k_{ii} + \frac{1}{N_{C_1}^2} \sum_{l \in C_1} \sum_{r \in C_1} k_{lr} - \frac{2}{N_{C_1}} \sum_{j \in C_1} k_{ij} + \ldots \quad \square
\end{aligned}$$

0.25. **Exercise 15.1.** Prove formula 15.1

Solution
Let

$$S = \frac{1}{B} \sum_i X_i, \tag{64}$$

then

$$
\begin{aligned}
Var(S) =& \frac{1}{B^2}\sum_i Var(X_i) + \frac{1}{B^2}\sum_{i\neq j} Cov(X_i, X_j) \\
=& \frac{1}{B}\sigma^2 + \frac{1}{B^2}\sum_{i\neq j} p\sigma^2 = \frac{1}{B}\sigma^2 + \frac{1}{B^2}p\sigma^2(B^2 - B) \\
=& p\sigma^2 + \frac{1-p}{B}\sigma^2 \quad \square
\end{aligned} \tag{65}
$$

0.26. **Exercise 15.2.** Show that as $B \to \infty$ the OOB-error approaches that of N-fold crossvalidation.

Solution
Since bagging uses sampling with replacement, as we increase the number of trees, the chance of a point not being included in at least one of the tress goes to zero. If we have n points and r samples for each Bootstrap sample, then the probability of a point x_i not being included is $\left(\frac{n-1}{n}\right)^r$. If we have B trees then the probability for any tree not containing x_i goes to zero:

$$
\lim_{B\to\infty} \left(\frac{n-1}{n}\right)^{rB} \to 0. \tag{66}
$$

Here it gets a bit hand wavy but as B increases we will get a proportion of trees that does not contain x_i. The OOB-error is the mean prediction error on each training sample using only the trees that did not contain x_i. For a large enough B the number of trees will be big enough to on average give the same error as for leave-one-out cross-validation on a random forest. That is, if B is infinity we will have an infinite amount of trees not containing x_i and this will obviously be the same as leave-one-out cross-validation.

I read Breiman and bunch of other papers on this, and it seems that this fact is only demonstrated in simulation only.

0.27. **Exercise 16.3.** Show that fitting a linear regression model using rules 1, 4, 5 and 6 in equation (16.14) gives the same fit as the regression tree corresponding to this tree. Show the same is true for classification, if a logistic regression model is fit.

Solution
It is obvious that the rules are just paths through the tree to the correct partition of the space. It is trivial to see that fitting a linear expansion of the rules,

$$
y(x) = c_1 R_1(x) + c_4 R_4(x) + c_5 R_5(x) + c_6 R_6(x) \tag{67}
$$

should give the same results as the tree for any x.

For a logistic regression model, where each terminal node in tree represents a class, we have,

(68)
$$\log(p(y = k|x)) = c_{1k}R_1(x) + c_{2k}R_4(x) + c_{5k}R_5(x) + c_{1k} + R_6(x) - \log Z, k = \{1, 2, \ldots, K\}$$

When for example $x \in R_1$ then if $c_{11} > 0$ we see that the probability will be highest for class 1. We can then design a decision rule that predicts to the highest probability