Your Name
email@fas.harvard.edu
CS181-S16

Assignment #2
Due: 5:00pm February 26, 2016

Collaborators: John Doe, Fred Doe

Homework 2: Linear Classification Solutions

There is a mathematical component and a programming component to this homework. Please submit your PDF to Canvas, and push everything in Github.

Grading Instructions: In the solutions, you will see several highlighted checkpoints. These each have a label that corresponds to an entry in the Canvas Quiz for this problem set. The highlighted statement should clearly indicate the criteria for being correct on that problem. If you satisfy the criteria for a problem being correct, mark "Yes" on the corresponding position on the Google Form. Otherwise, mark "No". Your homework scores will be verified by course staff at a later date.

This homework is about multi-class classification. Whereas in more simple classification models we build classifiers that discriminate between two classes, in multi-class regression, we discriminate between three or more classes. As usual, we imagine that we have the input matrix $\boldsymbol{X} \in \mathbb{R}^{N \times D}$ (or perhaps they have been mapped to some basis $\boldsymbol{\Phi}$, without loss of generality) but that our outputs are now "one-hot coded". What that means is that, if there are K output classes, rather than representing the output labels as integers $1, 2, \ldots, K$, we represent them as a binary vectors of length K. These vectors are zero in each component except for the one corresponding to the correct label, and that entry has a one. So, if there are 7 classes and a particular datum has label 3, then the target vector would be $[0, 0, 1, 0, 0, 0, 0]$ (assuming the labels are 1-indexed).

In the first problem, you will be exploring the properties of the softmax function, which is central to multiclass logistic regression. In the second problem, we will have you dive into the matrix algebra and methods behind generative classifications. Finally, in the third problem, you will implement a generative classifier and logistic regression from close to scratch, and the first two problems should inform this!

Problem 1 (Properties of Softmax , 5pts)

Logistic regression is a discriminative probabilistic model: a prediction consists of a distribution over the different classes. In other words, logistic regression outputs a vector of nonnegative numbers that sum to one.

The softmax function generalizes the logistic sigmoid to the case of K classes: it takes as input a vector, and outputs a K dimensional vector in the range $[0, 1]$ whose components sum to 1:

$$\boldsymbol{\sigma}(\mathbf{z}) = softmax(\mathbf{z}) = \frac{\exp(\mathbf{z})}{\sum_i \exp(z_i)}$$

In logistic regression, we often use the softmax-based parameterization over K vectors $\{\boldsymbol{w}_k\}$:

$$\Pr(t_{nk} = 1 \mid \boldsymbol{X}, \{\boldsymbol{w}_{k'}\}_{k'=1}^{K}) = \frac{\exp\{\boldsymbol{w}_k^{\mathsf{T}} \boldsymbol{x}_n\}}{\sum_{k'=1}^{K} \exp\{\boldsymbol{w}_{k'}^{\mathsf{T}} \boldsymbol{x}_n\}} \,.$$

Here we're using $t_{nk} = 1$ to indicate the probability that the nth entry is assigned to the kth class.

Softmax is a crucial function in logistic regression, and you will see it again in other models, such as neural networks. So, we want you to start gaining the intuitions for the properties of softmax, and for common methods that employ it.

Show that:

1. The output of the softmax function is always a vector with non-negative components that are at most 1.
2. The output of the softmax function forms a distribution (the components sum to 1).
3. Softmax preserves order. This means that if the elements of $\mathbf{z}$ have some order, then the elements of $\boldsymbol{\sigma}(\mathbf{z})$ have the same order.
4. Equation 4.106 from Bishop holds
5. Using your answer to the previous question, show that equation 4.109 holds. By the way, this may be useful for Problem 3!

Solution

The j component of the softmax function $\sigma(\mathbf{z})$ is:

$$\sigma(\mathbf{z})_j = \frac{\exp(z_j)}{\sum_i \exp(z_i)}.$$

1. As $\exp(x) > 0$ for all $x \in \mathbb{R}$, we have $\exp(z_j) > 0$ and $\sum_i \exp(z_i) > 0$. Thus the output of the softmax function is a vector with non-negative components. Since $\exp(z_j)$ appears in both the numerator and the denominator (as the $i = j$ term in the sum), the denominator must be at least as large as the numerator, and so the components are at most 1.

Check 1.1: You must both indicate that $\exp(z_j)$ is positive, and that $\sum_i \exp(z_i) \geq \exp(z_j)$. You will not get points if you missed either of these.

2. Summing over the components:

$$\sum_j \sigma(\mathbf{z})_j = \sum_j \frac{\exp(z_j)}{\sum_i \exp(z_i)} = \frac{\sum_j \exp(z_j)}{\sum_i \exp(z_i)} = 1.$$

Check 1.2: You summed over the components to show that it equals 1.

3. If $z_j \geq z_k$, then $\exp(z_j) \geq \exp(z_k)$ as the exponential is an increasing function. Dividing by the positive constant $\sum_i \exp(z_i)$, this inequality implies that:

$$\sigma(\mathbf{z})_j = \frac{\exp(z_j)}{\sum_i \exp(z_i)} \geq \frac{\exp(z_k)}{\sum_i \exp(z_i)} = \sigma(\mathbf{z})_k,$$

which shows that the softmax function preserves the order of the elements of $\mathbf{z}$.

Check 1.3: You either stated (or briefly showed via derivative) that exp is a monotonically increasing function. This means that it preserves order. You can either simply state that, or show it more formally as above.

4. To relate our notation to Bishop (4.106), note that $y_k = \sigma(\mathbf{z})_k$ and $a_j = z_j$. If $j \neq k$, then:

$$\begin{aligned}\frac{\partial \sigma(\mathbf{z})_k}{\partial z_j} &= \frac{\partial}{\partial z_j} \frac{\exp(z_k)}{\sum_i \exp(z_i)} = -\frac{\exp(z_k)}{\left(\sum_i \exp(z_i)\right)^2} \exp(z_j) \\ &= -\frac{\exp(z_k)}{\sum_i \exp(z_i)} \frac{\exp(z_j)}{\sum_i \exp(z_i)} = -\sigma(\mathbf{z})_k \sigma(\mathbf{z})_j.\end{aligned}$$

If $j = k$ then:

$$\begin{aligned}\frac{\partial \sigma(\mathbf{z})_k}{\partial z_j} &= \frac{\partial}{\partial z_j} \frac{\exp(z_k)}{\sum_i \exp(z_i)} = \frac{\exp(z_k)}{\sum_i \exp(z_i)} - \frac{\exp(z_j)^2}{\left(\sum_i \exp(z_i)\right)^2} \\ &= \left(1 - \frac{\exp(z_k)}{\sum_i \exp(z_i)}\right) \frac{\exp(z_k)}{\sum_i \exp(z_i)} = \sigma(\mathbf{z})_k (1 - \sigma(\mathbf{z})_j).\end{aligned}$$

Putting these results together:

$$\frac{\partial \sigma(\mathbf{z})_k}{\partial z_j} = \sigma(\mathbf{z})_k (I_{jk} - \sigma(\mathbf{z})_j).$$

Check 1.4: You broke the problem into the case where $j = k$ and $j \neq k$, and differentiated separately. Then, you combined them into a single term using the identity matrix, as above.

5. We begin with the cross-entropy error function from Bishop:

$$E(\mathbf{w}_1, \cdots, \mathbf{w}_K) = -\sum_{n=1}^{N} \sum_{k=1}^{K} t_{nk} \ln y_{nk}.$$

Taking the gradient of this expression and using the result of Part 4 in Bishop's notation:

$$\begin{aligned}\nabla_{\mathbf{w}_j} E(\mathbf{w}_1, \cdots, \mathbf{w}_K) &= -\sum_{n=1}^{N} \sum_{k=1}^{K} t_{nk} \nabla_{\mathbf{w}_j} \ln y_{nk} = -\sum_{n=1}^{N} \sum_{k=1}^{K} t_{nk} \frac{1}{y_{nk}} \nabla_{\mathbf{w}_j} y_{nk} \\ &= -\sum_{n=1}^{N} \sum_{k=1}^{K} t_{nk} \frac{1}{y_{nk}} \frac{\partial y_{nk}}{\partial a_j} \boldsymbol{\phi}_n = -\sum_{n=1}^{N} \sum_{k=1}^{K} t_{nk} \frac{1}{y_{nk}} y_{nk} (I_{kj} - y_{nj}) \boldsymbol{\phi}_n \\ &= -\sum_{n=1}^{N} \sum_{k=1}^{K} t_{nk} (I_{kj} - y_{nj}) \boldsymbol{\phi}_n = -\sum_{n=1}^{N} \sum_{k=1}^{K} t_{nk} I_{kj} \boldsymbol{\phi}_n + \sum_{n=1}^{N} y_{nj} \boldsymbol{\phi}_n \sum_{k=1}^{K} t_{nk}.\end{aligned}$$

The I_{kj} in the first sum collapses the sum over k to the term where $j = k$. As t_{nk} form the components of a 1-of-K encoding, we have that $\sum_{k=1}^{K} t_{nk} = 1$. Using these facts:

$$\nabla_{\mathbf{w}_j} E(\mathbf{w}_1, \cdots, \mathbf{w}_K) = -\sum_{n=1}^{N} t_{nj} \boldsymbol{\phi}_n + \sum_{n=1}^{N} y_{nj} \boldsymbol{\phi}_n = \sum_{n=1}^{N} (y_{nk} - t_{nk}) \boldsymbol{\phi}_n.$$

Check 1.5: You took the gradient correctly, and reduced it to the form shown in Bishop. You do not need to follow the exact same steps as the ones above, but they should be roughly the same.

Problem 2 (Mooooar matrix calculus , 10pts)

Note - this problem appears longer than it is, since we broke up one problem into separate parts rather than having you do all of these steps at once. Many of these subparts may be just one or two lines.

Consider a generative K-class model. We define the class prior with vector $\vec{\pi}$: $\mathbb{P}(\mathcal{C}_k) = \pi_k$. We define the class-conditional densities $\mathbb{P}(\phi|\mathcal{C}_k)$ where ϕ is the input feature vector. Consider the data set $\{\phi_n, \mathbf{t}_n\}$ where $n = 1 \ldots N$ where $\mathbf{t}_n \in \{0,1\}^K$ is a one-hot encoded target vector. This means that $\mathbf{t_n}$ is 0 everywhere, except for in the kth position, where k is the class assigned to the nth feature vector.

1. Write out the complete-data log-likelihood of the data set using only the notations introduced in the problem formulation above.
$$\ln \mathbb{P}(\{\phi_n, \mathbf{t}_n\}|\{\pi_k\}) = ?$$
2. Since the prior forms a distribution, it has the constraint that $\sum_k \pi_k - 1 = 0$. Using the hint at the end of the exercise, give the expression for the maximum-likelihood estimator for the prior class-membership probabilities:
$$\hat{\pi}_k = ?$$
Make sure to write out the intermediary equation you need to solve to obtain this estimator. Double-check your answer: the final result should be very intuitive!

We will suppose for the remaining questions of this exercise that the class-conditional probabilities are given by gaussian distributions with the same covariance matrix:

$$\mathbb{P}(\phi|C_k) = \mathcal{N}(\phi|\vec{\mu}_k, \Sigma)$$

3. Write out the gradient of log-likelihood with respect to vector μ_k. Write the expression in matrix form as a function of the variables defined throughout this exercise. Simplify as much as possible for full credit.
4. Write out the maximum-likelihood estimator for vector μ_k. Once again, your final answer should seem intuitive.
5. Write out the gradient for the log-likelihood with respect to the covariance matrix Σ. Even though the log-likelihood function is a scalar function, since you are differentiating with respect to a *matrix*, the resulting expression should be a matrix!
6. Express the maximum likelihood estimator of the covariance matrix.

Hint. When maximizing a function f with respect to an equality constraint that needs to be met at the optimum (which can always be written as $g(x) = 0$), we introduce a Lagrange multiplier λ and maximize:

$$\max_x f(x) + \lambda g(x)$$

Cookbook formulas. Here are some formulas you might want to consider using to compute difficult gradients. You can use them as is in the homework without proof. If you are looking to hone your matrix calculus skills, try to find different ways to prove these formulas yourself (will not be part of the evaluation of this homework). In general, you can use any formula from the matrix cookbook, as long as you cite it. We opt for the following common notation: $X^{-T} := (X^{-1})^{-T} = (X^T)^{-1}$

$$\frac{\partial a^T X^{-1} b}{\partial X} = -X^{-T} a b^T X^{-T}$$
$$\frac{\partial \ln|\det(X)|}{\partial X} = X^{-T}$$

Solution

1. The log-likelihood is given by:

$$\ln p(\{\phi_n, t_n\}|\{\pi_k\}) = \sum_{n=1}^{N}\sum_{k=1}^{K} t_{n,k}\left(\ln p(\phi_n|\mathcal{C}_k) + \ln \pi_k\right)$$

   Check 2.1: Your log likelihood should be the same as above (or an equivalent expression).

2. Using the note at the end of the exercise, we know we need to maximize

$$\ln p(\{\phi_n, t_n\}|\{\pi_k\}) = \sum_{n=1}^{N}\sum_{k=1}^{K} t_{n,k}\left(\ln p(\phi_n|\mathcal{C}_k) + \ln \pi_k\right) + \lambda\left(\left(\sum_{k=1} \pi_k\right) - 1\right)$$

   Check 2.2: You added the Lagrange multiplier term (the term with λ above).

   We take the derivative with respect to π_k and set it to 0:

$$\sum_{n=1}^{N} \frac{t_{nk}}{\pi_k} + \lambda = 0$$

   We obtain:

$$\forall k \in [K], -\lambda\pi_k = \sum_{n=1}^{N} t_{nk}$$

   By summing for every k, we get $\lambda = -N$. Substituting for λ, we get:

$$\hat{\pi}_k = \frac{1}{N}\sum_{n=1}^{N} t_{nk}$$

   Check 2.3: You correctly solved for $\hat{\pi}_k$.

3. The log-likelihood can be written as such:

$$\ln p(\{\phi_n, t_n\}|\{\pi_k\}) = \sum_{n=1}^{N}\sum_{k=1}^{K} t_{n,k}\left(-\frac{1}{2}(\phi_n - \mu_k)^T\Sigma^{-1}(\phi_n - \mu_k)\right) + \text{constants w.r.t } \mu_k$$

   The gradient w.r.t μ_k can be written as:

$$\sum_{n=1}^{N} t_{n,k}\Sigma^{-1}(\phi_n - \mu_k)$$

   Check 2.4: You correctly solved for the gradient.

4. We set the previous gradient equal to 0 to obtain:

$$\hat{\mu}_k = \frac{1}{\sum_{n=1}^{N} t_{n,k}}\sum_{n=1}^{N} t_{n,k}\phi_n$$

   Check 2.5: You correctly solved for $\hat{\mu}_k$. Note that this is just the average of the vectors in class k.

5. Using the two formulas in the cookbook, the gradient w.r.t Σ can be written as:

$$\sum_{n=1}^{N}\sum_{k=1}^{K} t_{n,k}\left[-\frac{1}{2}\Sigma^{-T}+\frac{1}{2}\Sigma^{-T}(\phi_n-\mu_k)(\phi_n-\mu_k)^T\Sigma^{-T}\right]$$

Check 2.6: You correctly calculated the gradient as above, or an equivalent expression.

6. Setting it to 0 and multiply both sides by Σ^T from the left and right:

$$\sum_{n,k} t_{n,k}\Sigma^T = \sum_{k,n} t_{n,k}(\phi_n-\mu_k)(\phi_n-\mu_k)^T$$

Taking the transpose (any matrix VV^T is symmetric), we have:

$$\hat{\Sigma} = \frac{1}{\sum_{n,k} t_{n,k}}\sum_{k,n} t_{n,k}(\phi_n-\mu_k)(\phi_n-\mu_k)^T$$

Check 2.7: You correctly found the MLE for the covariance. Note that this is the weighted average of the covariance matrix for each class separately.

3. Classifying Fruit [15pts]

You're tasked with classifying three different kinds of fruit, based on their heights and widths. Figure 1 is a plot of the data. Iain Murray collected these data and you can read more about this on his website at http://homepages.inf.ed.ac.uk/imurray2/teaching/oranges_and_lemons/. We have made a slightly simplified (collapsing the subcategories together) version of this available as `fruit.csv`, which you will find in the Github repository. The file has three columns: type (1=apple, 2=orange, 3=lemon), width, and height. The first few lines look like this:

```
fruit,width,height
1,8.4,7.3
1,8,6.8
1,7.4,7.2
1,7.1,7.8
...
```

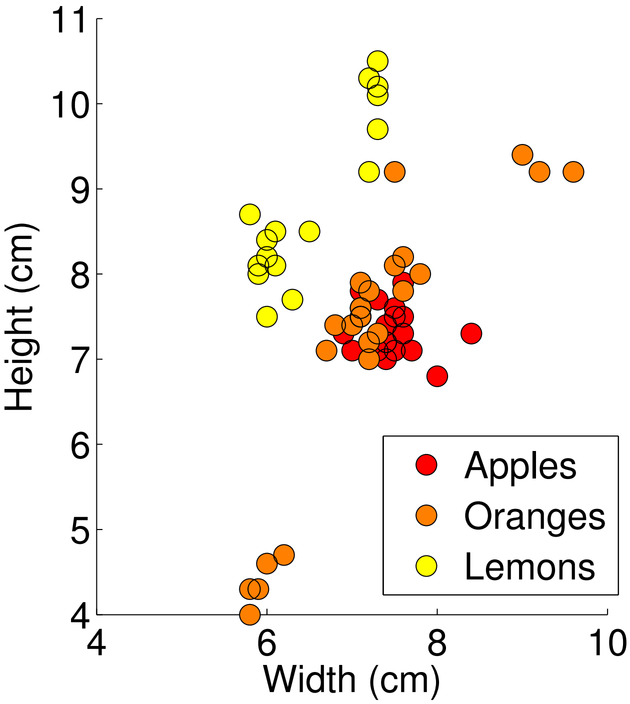


Figure 1: Heights and widths of apples, oranges, and lemons. These fruit were purchased and measured by Iain Murray: http://homepages.inf.ed.ac.uk/imurray2/teaching/oranges_and_lemons/.

Problem 3 (Classifying Fruit, 15pts)

Please implement the following:

- Implement the three-class generalization of logistic regression, also known as softmax regression, for these data. You will do this by implementing gradient descent on the log likelihood.

- After this, implement a simple generative classifier with Gaussian class-conditional densities, as in Bishop Section 4.2.2. In particular, make two implementations of this, one with a shared covariance matrix across all of the classes, and one with a separate covariance being learned for each class. Note that the staff implementation can switch between these two by the addition of just a few lines of code. The shared covariance matrix case is detailed in Bishop (and you worked on it in Problem 2), and the separate covariance case is only slightly different. In the separate covariance matrix case, the MLE for the covariance matrix of each class is simply the covariance of the data points assigned to that class, without combining them as in the shared case.

You may use anything in numpy or scipy, except for scipy.optimize. That being said, if you happen to find a function in numpy or scipy that seems like it is doing too much for you, run it by a staff member. In general, linear algebra and random variable functions are fine. The controller file is problem3.py, in which you will specify parameters. The actual implementations you will write will be in LogisticRegression.py and GaussianGenerativeModel.py.

You will be given unimplemented class interfaces for GaussianGenerativeModel and LogisticRegression in the distribution code, and the code will indicate certain lines that you should not change in your final submission. Naturally, don't change these. These classes will allow the final submissions to have consistency. There will also be a few hyperparameters that are set to irrelevant values at the moment. You may need to modify these to get your methods to work. The classes you implement follow the same pattern as scikit-learn, so they should be familiar to you. The distribution code currently outputs nonsense predictions just to show what the high-level interface should be, so you should completely remove the given predict() implementations and replace them with your implementations.

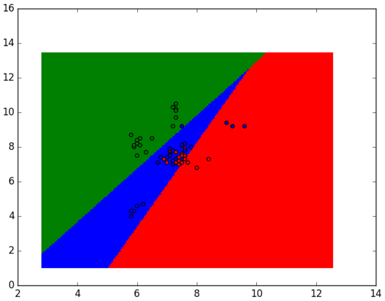
- The visualize() method for each classifier will save a plot that will show the decision boundaries. Please include those in this assignment.

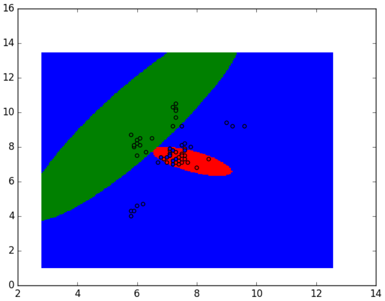
- Which classifiers model the distributions well?

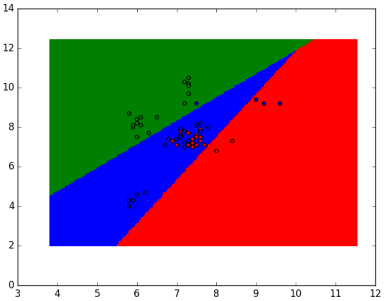
- What explains the differences?

Solution

Your plots should look like the ones below. These are, in order, the generative classifier with a shared covariance matrix, the generative classifier with separate covariance matrices, and the multiclass logistic regression classifier.







Check 3.1: Your plot for the generative classifier with shared covariances matches the one above. It should look almost identical, if not exactly identical.

Check 3.2: Your plot for the generative classifier with separative covariances matches the one above. It should look almost identical, if not exactly identical.

Check 3.3: Your plot for the logistic regression classifier matches the one above. The staff solution ran 500,000 epochs to achieve this solution.

Which classifiers model the distributions well?

This is a pretty open-ended questions, so we will generally accept a broad range of answers. A few things that should be pretty clear is that the generative model with separate covariance matrices seems to model the distribution better than the generative model with a shared covariance matrix. Also, Logistic Regression and the generative model with shared covariance perform similarly.

Check 3.4: Your observations are consistent with the plots. As long as you present reasonable observations, and none of them contradict the examples above, you get the points.

What explains the differences?

Have a shared covariance matrix means that each of the three classes have approximately the same shape, which is elongated from the bottom left to the top-right of the plot. However, just from visual inspection, we can see that the three classes actually have fairly different shapes, with some being elongated from bottom left to top right, and the red ones being at a right angle to those. So, having separate covariance matrices for each one seems to lead to a better-looking fit. Logistic Regression generally performs well, but since it enforces having linear decision boundaries, there is a limit to its ability to model this data.

Check 3.5: You noted why the separate covariance case is different from the one here. Also, you didn't say anything that directly contradicts the above.

Calibration [1pt]

Approximately how long did this homework take you to complete?